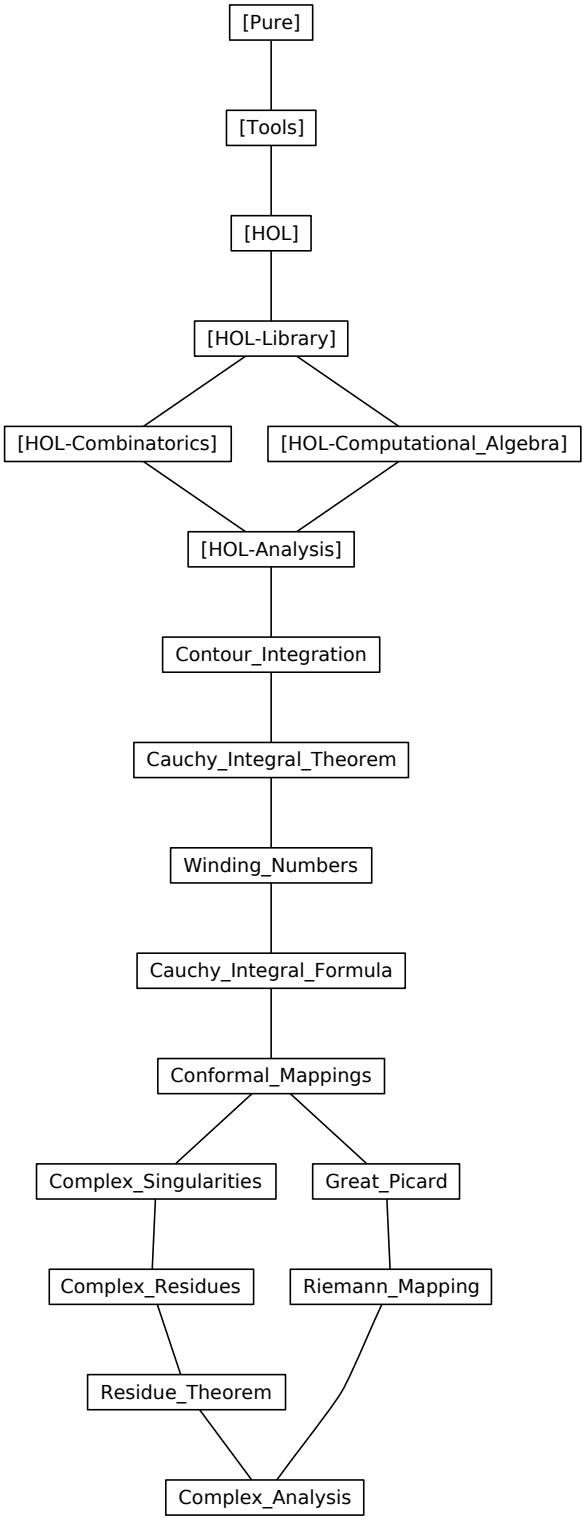
Complex Analysis

December 12, 2021

Contents


[Pure]
[Tools]
[HOL]
[HOL-Library]
[HOL-Combinatorics]
[HOL-Computational_Algebra]
[HOL-Analysis]
Contour_Integration
Cauchy_Integral_Theorem
Winding_Numbers
Cauchy_Integral_Formula
Conformal_Mappings
Complex_Singularities
Great_Picard
Complex_Residues
Riemann_Mapping
Residue_Theorem
Complex_Analysis

1 Contour integration

theory *Contour_Integration*
imports *HOL−Analysis.Analysis*
begin

lemma *lhopital_complex_simple*:
assumes (*f has_field_derivative f′*) (*at z*)
assumes (*g has_field_derivative g′*) (*at z*)
assumes *f z* = *0 g z* = *0 g′* ≠ *0 f′* / *g′* = *c*
shows ((λ*w*. *f w* / *g w*) ⟶ *c*) (*at z*)
proof −
have *eventually* (λ*w*. *w* ≠ *z*) (*at z*)
by (*auto simp*: *eventually_at_filter*)
hence *eventually* (λ*w*. ((*f w* − *f z*) / (*w* − *z*)) / ((*g w* − *g z*) / (*w* − *z*)) = *f w* / *g w*) (*at z*)
by *eventually_elim* (*simp add*: *assms field_split_simps*)
moreover have ((λ*w*. ((*f w* − *f z*) / (*w* − *z*)) / ((*g w* − *g z*) / (*w* − *z*))) ⟶ *f′* / *g′*) (*at z*)
by (*intro tendsto_divide has_field_derivativeD assms*)
ultimately have ((λ*w*. *f w* / *g w*) ⟶ *f′* / *g′*) (*at z*)
by (*blast intro*: *Lim_transform_eventually*)
with *assms* **show** *?thesis* **by** *simp*
qed

1.1 Definition

This definition is for complex numbers only, and does not generalise to line integrals in a vector field

definition *has_contour_integral* :: (*complex* ⇒ *complex*) ⇒ *complex* ⇒ (*real* ⇒ *complex*) ⇒ *bool*
(**infixr** *has′_contour′_integral* *50*)
where (*f has_contour_integral i*) *g* ≡
((λ*x*. *f*(*g x*) ∗ *vector_derivative g* (*at x within* {*0..1*}))
has_integral i) {*0..1*}

definition *contour_integrable_on*
(**infixr** *contour′_integrable′_on* *50*)
where *f contour_integrable_on g* ≡ ∃ *i*. (*f has_contour_integral i*) *g*

definition *contour_integral*
where *contour_integral g f* ≡ *SOME i*. (*f has_contour_integral i*) *g* ∨ ¬ *f contour_integrable_on g* ∧ *i*=*0*

lemma *not_integrable_contour_integral*: ¬ *f contour_integrable_on g* ⟹ *contour_integral g f* = *0*
unfolding *contour_integrable_on_def contour_integral_def* **by** *blast*

lemma *contour_integral_unique*: (*f has_contour_integral i*) *g* ⟹ *contour_integral*

g f = i
apply (*simp add*: *contour_integral_def has_contour_integral_def contour_integrable_on_def*)
using *has_integral_unique* **by** *blast*

lemma *has_contour_integral_eqpath*:
⟦(*f has_contour_integral y*) *p*; *f contour_integrable_on γ*;
contour_integral p f = contour_integral γ f⟧
⟹ (*f has_contour_integral y*) *γ*
using *contour_integrable_on_def contour_integral_unique* **by** *auto*

lemma *has_contour_integral_integral*:
f contour_integrable_on i ⟹ (*f has_contour_integral* (*contour_integral i f*))
i
by (*metis contour_integral_unique contour_integrable_on_def*)

lemma *has_contour_integral_unique*:
(*f has_contour_integral i*) *g* ⟹ (*f has_contour_integral j*) *g* ⟹ *i = j*
using *has_integral_unique*
by (*auto simp*: *has_contour_integral_def*)

lemma *has_contour_integral_integrable*: (*f has_contour_integral i*) *g* ⟹ *f contour_integrable_on g*
using *contour_integrable_on_def* **by** *blast*

Show that we can forget about the localized derivative.

lemma *has_integral_localized_vector_derivative*:
((λ*x*. *f* (*g x*) ∗ *vector_derivative p* (*at x within* {*a..b*})) *has_integral i*) {*a..b*}
⟷
((λ*x*. *f* (*g x*) ∗ *vector_derivative p* (*at x*)) *has_integral i*) {*a..b*}
proof −
have ∗: {*a..b*} − {*a*,*b*} = *interior* {*a..b*}
by (*simp add*: *atLeastAtMost_diff_ends*)
show *?thesis*
by (*rule has_integral_spike_eq* [*of* {*a*,*b*}]) (*auto simp*: *at_within_interior* [*of* _ {*a..b*}])
qed

lemma *integrable_on_localized_vector_derivative*:
(λ*x*. *f* (*g x*) ∗ *vector_derivative p* (*at x within* {*a..b*})) *integrable_on* {*a..b*}
⟷
(λ*x*. *f* (*g x*) ∗ *vector_derivative p* (*at x*)) *integrable_on* {*a..b*}
by (*simp add*: *integrable_on_def has_integral_localized_vector_derivative*)

lemma *has_contour_integral*:
(*f has_contour_integral i*) *g* ⟷
((λ*x*. *f* (*g x*) ∗ *vector_derivative g* (*at x*)) *has_integral i*) {*0..1*}
by (*simp add*: *has_integral_localized_vector_derivative has_contour_integral_def*)

lemma *contour_integrable_on*:

f contour_integrable_on g ⟷
(λ*t*. *f*(*g t*) ∗ *vector_derivative g* (*at t*)) *integrable_on* {*0..1*}
by (*simp add*: *has_contour_integral integrable_on_def contour_integrable_on_def*)

1.2 Reversing a path

lemma *has_contour_integral_reversepath*:
assumes *valid_path g* **and** *f*: (*f has_contour_integral i*) *g*
shows (*f has_contour_integral* (−*i*)) (*reversepath g*)
proof −
{ fix *S x*
assume *xs*: *g C1_differentiable_on* ({*0..1*} − *S*) *x* ∉ (−) *1* ‘ *S 0* ≤ *x x* ≤ *1*
have *vector_derivative* (λ*x*. *g* (*1* − *x*)) (*at x within* {*0..1*}) =
− *vector_derivative g* (*at* (*1* − *x*) *within* {*0..1*})
proof −
obtain *f′* **where** *f′*: (*g has_vector_derivative f′*) (*at* (*1* − *x*))
using *xs*
by (*force simp*: *has_vector_derivative_def C1_differentiable_on_def*)
have (*g* ∘ (λ*x*. *1* − *x*) *has_vector_derivative* −*1* ∗*R f′*) (*at x*)
by (*intro vector_diff_chain_within has_vector_derivative_at_within* [*OF f′*] *derivative_eq_intros* | *simp*)+
then have *mf′*: ((λ*x*. *g* (*1* − *x*)) *has_vector_derivative* −*f′*) (*at x*)
by (*simp add*: *o_def*)
show *?thesis*
using *xs*
by (*auto simp*: *vector_derivative_at_within_ivl* [*OF mf′*] *vector_derivative_at_within_ivl* [*OF f′*])
qed
} **note** ∗ = *this*
obtain *S* **where** *S*: *continuous_on* {*0..1*} *g finite S g C1_differentiable_on* {*0..1*} − *S*
using *assms* **by** (*auto simp*: *valid_path_def piecewise_C1_differentiable_on_def*)
have ((λ*x*. − (*f* (*g* (*1* − *x*)) ∗ *vector_derivative g* (*at* (*1* − *x*) *within* {*0..1*})))
has_integral −*i*)
{*0..1*}
using *has_integral_affinity01* [**where** *m*= −*1* **and** *c*=*1*, *OF f* [*unfolded has_contour_integral_def*]]
by (*simp add*: *has_integral_neg*)
then show *?thesis*
using *S*
unfolding *reversepath_def has_contour_integral_def*
by (*rule_tac S* = (λ*x*. *1* − *x*) ‘ *S* **in** *has_integral_spike_finite*) (*auto simp*: ∗)
qed

lemma *contour_integrable_reversepath*:
valid_path g ⟹ *f contour_integrable_on g* ⟹ *f contour_integrable_on* (*reversepath g*)
using *has_contour_integral_reversepath contour_integrable_on_def* **by** *blast*

lemma *contour_integrable_reversepath_eq*:
valid_path g $\Longrightarrow$ (*f contour_integrable_on* (*reversepath g*) $\longleftrightarrow$ *f contour_integrable_on g*)
using *contour_integrable_reversepath valid_path_reversepath* **by** *fastforce*

lemma *contour_integral_reversepath*:
assumes *valid_path g*
shows *contour_integral* (*reversepath g*) *f* = − (*contour_integral g f*)
proof (*cases f contour_integrable_on g*)
case *True* **then show** *?thesis*
by (*simp add*: *assms contour_integral_unique has_contour_integral_integral has_contour_integral_reversepath*)
next
case *False* **then have** ¬ *f contour_integrable_on* (*reversepath g*)
by (*simp add*: *assms contour_integrable_reversepath_eq*)
with *False* **show** *?thesis* **by** (*simp add*: *not_integrable_contour_integral*)
qed

1.3 Joining two paths together

lemma *has_contour_integral_join*:
assumes (*f has_contour_integral i1*) *g1* (*f has_contour_integral i2*) *g2*
valid_path g1 valid_path g2
shows (*f has_contour_integral* (*i1* + *i2*)) (*g1* +++ *g2*)
proof −
obtain *s1 s2*
where *s1*: *finite s1* ∀ *x*∈{*0..1*} − *s1*. *g1 differentiable at x*
and *s2*: *finite s2* ∀ *x*∈{*0..1*} − *s2*. *g2 differentiable at x*
using *assms*
by (*auto simp*: *valid_path_def piecewise_C1_differentiable_on_def C1_differentiable_on_eq*)
have *1*: ((λ*x*. *f* (*g1 x*) ∗ *vector_derivative g1* (*at x*)) *has_integral i1*) {*0..1*}
and *2*: ((λ*x*. *f* (*g2 x*) ∗ *vector_derivative g2* (*at x*)) *has_integral i2*) {*0..1*}
using *assms*
by (*auto simp*: *has_contour_integral*)
have *i1*: ((λ*x*. (*2*∗*f* (*g1* (*2*∗*x*))) ∗ *vector_derivative g1* (*at* (*2*∗*x*))) *has_integral i1*) {*0..1/2*}
and *i2*: ((λ*x*. (*2*∗*f* (*g2* (*2*∗*x* − *1*))) ∗ *vector_derivative g2* (*at* (*2*∗*x* − *1*))) *has_integral i2*) {*1/2..1*}
using *has_integral_affinity01* [*OF 1*, **where** *m*= *2* **and** *c*=*0*, *THEN has_integral_cmul* [**where** *c*=*2*]]
has_integral_affinity01 [*OF 2*, **where** *m*= *2* **and** *c*=−*1*, *THEN has_integral_cmul* [**where** *c*=*2*]]
by (*simp_all only*: *image_affinity_atLeastAtMost_div_diff*, *simp_all add*: *scaleR_conv_of_real mult_ac*)
have *g1*: *vector_derivative* (λ*x*. *if x*∗*2* ≤ *1 then g1* (*2*∗*x*) *else g2* (*2*∗*x* − *1*)) (*at z*) =
2 ∗$_R$ *vector_derivative g1* (*at* (*z*∗*2*))
if *0* ≤ *z z*∗*2* < *1 z*∗*2* ∉ *s1* **for** *z*
proof (*rule vector_derivative_at* [*OF has_vector_derivative_transform_within*])

```
    show 0 < |z − 1/2|
      using that by auto
    have ((∗) 2 has_vector_derivative 2) (at z)
    by (simp add: has_vector_derivative_def has_derivative_def bounded_linear_mult_left)
    moreover have (g1 has_vector_derivative vector_derivative g1 (at (z ∗ 2)))
(at (2 ∗ z))
      using s1 that by (auto simp: algebra_simps vector_derivative_works)
    ultimately
    show ((λx. g1 (2 ∗ x)) has_vector_derivative 2 ∗R vector_derivative g1 (at
(z ∗ 2))) (at z)
      by (intro vector_diff_chain_at [simplified o_def])
  qed (use that in ‹simp_all add: dist_real_def abs_if split: if_split_asm›)

  have g2: vector_derivative (λx. if x∗2 ≤ 1 then g1 (2∗x) else g2 (2∗x − 1))
(at z) =
          2 ∗R vector_derivative g2 (at (z∗2 − 1))
         if 1 < z∗2 z ≤ 1 z∗2 − 1 ∉ s2 for z
  proof (rule vector_derivative_at [OF has_vector_derivative_transform_within])
    show 0 < |z − 1/2|
      using that by auto
    have ((λx. 2 ∗ x − 1) has_vector_derivative 2) (at z)
    by (simp add: has_vector_derivative_def has_derivative_def bounded_linear_mult_left)
    moreover have (g2 has_vector_derivative vector_derivative g2 (at (z ∗ 2 −
1))) (at (2 ∗ z − 1))
      using s2 that by (auto simp: algebra_simps vector_derivative_works)
    ultimately
    show ((λx. g2 (2 ∗ x − 1)) has_vector_derivative 2 ∗R vector_derivative g2
(at (z ∗ 2 − 1))) (at z)
      by (intro vector_diff_chain_at [simplified o_def])
  qed (use that in ‹simp_all add: dist_real_def abs_if split: if_split_asm›)

  have ((λx. f ((g1 +++ g2) x) ∗ vector_derivative (g1 +++ g2) (at x)) has_integral
i1) {0..1/2}
  proof (rule has_integral_spike_finite [OF _ _ i1])
    show finite (insert (1/2) ((∗) 2 −‘ s1))
      using s1 by (force intro: finite_vimageI [where h = (∗)2] inj_onI)
  qed (auto simp add: joinpaths_def scaleR_conv_of_real mult_ac g1)
  moreover have ((λx. f ((g1 +++ g2) x) ∗ vector_derivative (g1 +++ g2) (at
x)) has_integral i2) {1/2..1}
  proof (rule has_integral_spike_finite [OF _ _ i2])
    show finite (insert (1/2) ((λx. 2 ∗ x − 1) −‘ s2))
      using s2 by (force intro: finite_vimageI [where h = λx. 2∗x−1] inj_onI)
  qed (auto simp add: joinpaths_def scaleR_conv_of_real mult_ac g2)
  ultimately
  show ?thesis
    by (simp add: has_contour_integral has_integral_combine [where c = 1/2])
qed

lemma contour_integrable_joinI:
```

assumes *f contour_integrable_on g1 f contour_integrable_on g2*
valid_path g1 valid_path g2
shows *f contour_integrable_on* (*g1 +++ g2*)
using *assms*
by (*meson has_contour_integral_join contour_integrable_on_def*)

lemma *contour_integrable_joinD1*:
assumes *f contour_integrable_on* (*g1 +++ g2*) *valid_path g1*
shows *f contour_integrable_on g1*
proof −
obtain *s1*
where *s1*: *finite s1* ∀ *x*∈{*0..1*} − *s1*. *g1 differentiable at x*
using *assms* **by** (*auto simp*: *valid_path_def piecewise_C1_differentiable_on_def C1_differentiable_on_eq*)
have (λ*x*. *f* ((*g1 +++ g2*) (*x*/*2*)) ∗ *vector_derivative* (*g1 +++ g2*) (*at* (*x*/*2*))) *integrable_on* {*0..1*}
using *assms integrable_affinity* [*of* _ *0 1/2 1/2 0*] *integrable_on_subcbox* [**where** *a=0* **and** *b=1/2*]
by (*fastforce simp*: *contour_integrable_on*)
then have ∗: (λ*x*. (*f* ((*g1 +++ g2*) (*x*/*2*))/*2*) ∗ *vector_derivative* (*g1 +++ g2*) (*at* (*x*/*2*))) *integrable_on* {*0..1*}
by (*auto dest*: *integrable_cmul* [**where** *c=1/2*] *simp*: *scaleR_conv_of_real*)
have *g1*: *vector_derivative* (λ*x*. *if x*∗*2* ≤ *1 then g1* (*2*∗*x*) *else g2* (*2*∗*x* − *1*)) (*at* (*z*/*2*)) =
2 ∗$_R$ *vector_derivative g1* (*at z*)
if *0* < *z z* < *1 z* ∉ *s1* **for** *z*
proof (*rule vector_derivative_at* [*OF has_vector_derivative_transform_within*])
show *0* < |(*z* − *1*)/*2*|
using *that* **by** *auto*
have §: ((λ*x*. *x* ∗ *2*) *has_vector_derivative 2*) (*at* (*z*/*2*))
using *s1* **by** (*auto simp*: *has_vector_derivative_def has_derivative_def bounded_linear_mult_left*)
have (*g1 has_vector_derivative vector_derivative g1* (*at z*)) (*at z*)
using *s1 that* **by** (*auto simp*: *vector_derivative_works*)
then show ((λ*x*. *g1* (*2* ∗ *x*)) *has_vector_derivative 2* ∗$_R$ *vector_derivative g1* (*at z*)) (*at* (*z*/*2*))
using *vector_diff_chain_at* [*OF* §] **by** (*auto simp*: *field_simps o_def*)
qed (*use that* **in** ‹*simp_all add*: *field_simps dist_real_def abs_if split*: *if_split_asm*›)
have *fin01*: *finite* ({*0*, *1*} ∪ *s1*)
by (*simp add*: *s1*)
show *?thesis*
unfolding *contour_integrable_on*
by (*intro integrable_spike_finite* [*OF fin01* _ ∗]) (*auto simp*: *joinpaths_def scaleR_conv_of_real g1*)
qed

lemma *contour_integrable_joinD2*:
assumes *f contour_integrable_on* (*g1 +++ g2*) *valid_path g2*
shows *f contour_integrable_on g2*

proof −
obtain *s2*
where *s2*: *finite s2* ∀ *x*∈{*0..1*} − *s2*. *g2 differentiable at x*
using *assms* **by** (*auto simp*: *valid_path_def piecewise_C1_differentiable_on_def C1_differentiable_on_eq*)
have (λ*x*. *f* ((*g1* +++ *g2*) (*x*/*2* + *1*/*2*)) ∗ *vector_derivative* (*g1* +++ *g2*) (*at* (*x*/*2* + *1*/*2*))) *integrable_on* {*0..1*}
using *assms integrable_affinity* [*of* _ *1*/*2*::*real 1 1*/*2 1*/*2*]
integrable_on_subcbox [**where** *a*=*1*/*2* **and** *b*=*1*]
by (*fastforce simp*: *contour_integrable_on image_affinity_atLeastAtMost_diff*)
then have ∗: (λ*x*. (*f* ((*g1* +++ *g2*) (*x*/*2* + *1*/*2*))/*2*) ∗ *vector_derivative* (*g1* +++ *g2*) (*at* (*x*/*2* + *1*/*2*)))
integrable_on {*0..1*}
by (*auto dest*: *integrable_cmul* [**where** *c*=*1*/*2*] *simp*: *scaleR_conv_of_real*)
have *g2*: *vector_derivative* (λ*x*. *if x*∗*2* ≤ *1 then g1* (*2*∗*x*) *else g2* (*2*∗*x* − *1*)) (*at* (*z*/*2*+*1*/*2*)) =
2 ∗*R* *vector_derivative g2* (*at z*)
if *0* < *z z* < *1 z* ∉ *s2* **for** *z*
proof (*rule vector_derivative_at* [*OF has_vector_derivative_transform_within*])
show *0* < |*z*/*2*|
using *that* **by** *auto*
have §: ((λ*x*. *x* ∗ *2* − *1*) *has_vector_derivative 2*) (*at* ((*1* + *z*)/*2*))
using *s2* **by** (*auto simp*: *has_vector_derivative_def has_derivative_def bounded_linear_mult_left*)
have (*g2 has_vector_derivative vector_derivative g2* (*at z*)) (*at z*)
using *s2 that* **by** (*auto simp*: *vector_derivative_works*)
then show ((λ*x*. *g2* (*2*∗*x* − *1*)) *has_vector_derivative 2* ∗*R* *vector_derivative g2* (*at z*)) (*at* (*z*/*2* + *1*/*2*))
using *vector_diff_chain_at* [*OF* §] **by** (*auto simp*: *field_simps o_def*)
qed (*use that* **in** ‹*simp_all add*: *field_simps dist_real_def abs_if split*: *if_split_asm*›)
have *fin01*: *finite* ({*0*, *1*} ∪ *s2*)
by (*simp add*: *s2*)
show *?thesis*
unfolding *contour_integrable_on*
by (*intro integrable_spike_finite* [*OF fin01* _ ∗]) (*auto simp*: *joinpaths_def scaleR_conv_of_real g2*)
qed

lemma *contour_integrable_join* [*simp*]:
⟦*valid_path g1*; *valid_path g2*⟧
⟹ *f contour_integrable_on* (*g1* +++ *g2*) ⟷ *f contour_integrable_on g1* ∧ *f contour_integrable_on g2*
using *contour_integrable_joinD1 contour_integrable_joinD2 contour_integrable_joinI*
by *blast*

lemma *contour_integral_join* [*simp*]:
⟦*f contour_integrable_on g1*; *f contour_integrable_on g2*; *valid_path g1*; *valid_path g2*⟧
⟹ *contour_integral* (*g1* +++ *g2*) *f* = *contour_integral g1 f* + *con-*

tour_integral g2 f
by (*simp add*: *has_contour_integral_integral has_contour_integral_join contour_integral_unique*)

1.4 Shifting the starting point of a (closed) path

lemma *has_contour_integral_shiftpath*:
assumes *f*: (*f has_contour_integral i*) *g valid_path g*
and *a*: *a* ∈ {*0..1*}
shows (*f has_contour_integral i*) (*shiftpath a g*)
proof −
obtain *S*
where *S*: *finite S* **and** *g*: ∀ *x*∈{*0..1*} − *S*. *g differentiable at x*
using *assms* **by** (*auto simp*: *valid_path_def piecewise_C1_differentiable_on_def C1_differentiable_on_eq*)
have ∗: ((λ*x*. *f* (*g x*) ∗ *vector_derivative g* (*at x*)) *has_integral i*) {*0..1*}
using *assms* **by** (*auto simp*: *has_contour_integral*)
then have *i*: *i* = *integral* {*a..1*} (λ*x*. *f* (*g x*) ∗ *vector_derivative g* (*at x*)) +
integral {*0..a*} (λ*x*. *f* (*g x*) ∗ *vector_derivative g* (*at x*))
apply (*rule has_integral_unique*)
apply (*subst add.commute*)
apply (*subst Henstock_Kurzweil_Integration.integral_combine*)
using *assms* ∗ *integral_unique* **by** *auto*

have *vd1*: *vector_derivative* (*shiftpath a g*) (*at x*) = *vector_derivative g* (*at* (*x* + *a*))
if *0* ≤ *x x* + *a* < *1 x* ∉ (λ*x*. *x* − *a*) ‘ *S* **for** *x*
unfolding *shiftpath_def*
proof (*rule vector_derivative_at* [*OF has_vector_derivative_transform_within*])
have ((λ*x*. *g* (*x* + *a*)) *has_vector_derivative vector_derivative g* (*at* (*a* + *x*))) (*at x*)
proof (*rule vector_diff_chain_at* [*of _ 1*, *simplified o_def scaleR_one*])
show ((λ*x*. *x* + *a*) *has_vector_derivative 1*) (*at x*)
by (*rule derivative_eq_intros* | *simp*)+
have *g differentiable at* (*x* + *a*)
using *g a that* **by** *force*
then show (*g has_vector_derivative vector_derivative g* (*at* (*a* + *x*))) (*at* (*x* + *a*))
by (*metis add.commute vector_derivative_works*)
qed
then
show ((λ*x*. *g* (*a* + *x*)) *has_vector_derivative vector_derivative g* (*at* (*x* + *a*))) (*at x*)
by (*auto simp*: *field_simps*)
show *0* < *dist* (*1* − *a*) *x*
using *that* **by** *auto*
qed (*use that* **in** ‹*auto simp*: *dist_real_def*›)

have *vd2*: *vector_derivative* (*shiftpath a g*) (*at x*) = *vector_derivative g* (*at* (*x*

```
+ a − 1))
    if x ≤ 1 1 < x + a x ∉ (λx. x − a + 1) ‘ S for x
    unfolding shiftpath_def
  proof (rule vector_derivative_at [OF has_vector_derivative_transform_within])
     have ((λx. g (x + a − 1)) has_vector_derivative vector_derivative g (at
(a+x−1))) (at x)
    proof (rule vector_diff_chain_at [of _ 1, simplified o_def scaleR_one])
      show ((λx. x + a − 1) has_vector_derivative 1) (at x)
        by (rule derivative_eq_intros | simp)+
      have g differentiable at (x+a−1)
        using g a that by force
      then show (g has_vector_derivative vector_derivative g (at (a+x−1))) (at
(x + a − 1))
        by (metis add.commute vector_derivative_works)
    qed
   then show ((λx. g (a + x − 1)) has_vector_derivative vector_derivative g (at
(x + a − 1))) (at x)
      by (auto simp: field_simps)
    show 0 < dist (1 − a) x
      using that by auto
  qed (use that in ‹auto simp: dist_real_def›)

  have va1: (λx. f (g x) ∗ vector_derivative g (at x)) integrable_on ({a..1})
    using ∗ a   by (fastforce intro: integrable_subinterval_real)
  have v0a: (λx. f (g x) ∗ vector_derivative g (at x)) integrable_on ({0..a})
    using ∗ a by (force intro: integrable_subinterval_real)
  have finite ({1 − a} ∪ (λx. x − a) ‘ S)
    using S by blast
  then have ((λx. f (shiftpath a g x) ∗ vector_derivative (shiftpath a g) (at x))
       has_integral integral {a..1} (λx. f (g x) ∗ vector_derivative g (at x))) {0..1
− a}
    apply (rule has_integral_spike_finite
        [where f = λx. f(g(a+x)) ∗ vector_derivative g (at(a+x))])
    subgoal
      using a by (simp add: va1) (force simp: shiftpath_def add.commute)
    subgoal
      using has_integral_affinity [where m=1 and c=a] integrable_integral [OF
va1]
      by (force simp add: add.commute)
    done
  moreover
  have finite ({1 − a} ∪ (λx. x − a + 1) ‘ S)
    using S by blast
  then have ((λx. f (shiftpath a g x) ∗ vector_derivative (shiftpath a g) (at x))
          has_integral  integral {0..a} (λx. f (g x) ∗ vector_derivative g (at x)))
{1 − a..1}
    apply (rule has_integral_spike_finite
        [where f = λx. f(g(a+x−1)) ∗ vector_derivative g (at(a+x−1))])
    subgoal
```

```
    using a by (simp add: vd2) (force simp: shiftpath_def add.commute)
  subgoal
    using has_integral_affinity [where m=1 and c=a−1, simplified, OF integrable_integral [OF v0a]]
    by (force simp add: algebra_simps)
  done
 ultimately show ?thesis
  using a
  by (auto simp: i has_contour_integral intro: has_integral_combine [where c = 1−a])
qed

lemma has_contour_integral_shiftpath_D:
 assumes (f has_contour_integral i) (shiftpath a g)
         valid_path g pathfinish g = pathstart g a ∈ {0..1}
  shows (f has_contour_integral i) g
proof −
 obtain S
  where S: finite S and g: ∀ x∈{0..1} − S. g differentiable at x
  using assms by (auto simp: valid_path_def piecewise_C1_differentiable_on_def C1_differentiable_on_eq)
 { fix x
  assume x: 0 < x x < 1 x ∉ S
  then have gx: g differentiable at x
   using g by auto
  have §: shiftpath (1 − a) (shiftpath a g) differentiable at x
   using assms x
   by (intro differentiable_transform_within [OF gx, of min x (1−x)])
     (auto simp: dist_real_def shiftpath_shiftpath abs_if split: if_split_asm)
  have vector_derivative g (at x within {0..1}) =
       vector_derivative (shiftpath (1 − a) (shiftpath a g)) (at x within {0..1})
   apply (rule vector_derivative_at_within_ivl
            [OF has_vector_derivative_transform_within_open
                [where f = (shiftpath (1 − a) (shiftpath a g)) and S = {0<..<1}−S]])
   using S assms x §
   apply (auto simp: finite_imp_closed open_Diff shiftpath_shiftpath
                     at_within_interior [of _ {0..1}] vector_derivative_works [symmetric])
   done
 } note vd = this
 have fi: (f has_contour_integral i) (shiftpath (1 − a) (shiftpath a g))
  using assms  by (auto intro!: has_contour_integral_shiftpath)
 show ?thesis
  unfolding has_contour_integral_def
  proof (rule has_integral_spike_finite [of {0,1} ∪ S, OF _ _  fi [unfolded has_contour_integral_def]])
  show finite ({0, 1} ∪ S)
   by (simp add: S)
```

qed (*use S assms vd* **in** ‹*auto simp*: *shiftpath_shiftpath*›)
qed

lemma *has_contour_integral_shiftpath_eq*:
assumes *valid_path g pathfinish g = pathstart g a* ∈ {*0..1*}
shows (*f has_contour_integral i*) (*shiftpath a g*) ⟷ (*f has_contour_integral i*) *g*
using *assms has_contour_integral_shiftpath has_contour_integral_shiftpath_D*
by *blast*

lemma *contour_integrable_on_shiftpath_eq*:
assumes *valid_path g pathfinish g = pathstart g a* ∈ {*0..1*}
shows *f contour_integrable_on* (*shiftpath a g*) ⟷ *f contour_integrable_on g*
using *assms contour_integrable_on_def has_contour_integral_shiftpath_eq* **by** *auto*

lemma *contour_integral_shiftpath*:
assumes *valid_path g pathfinish g = pathstart g a* ∈ {*0..1*}
shows *contour_integral* (*shiftpath a g*) *f* = *contour_integral g f*
using *assms*
by (*simp add*: *contour_integral_def contour_integrable_on_def has_contour_integral_shiftpath_eq*)

1.5 More about straight-line paths

lemma *has_contour_integral_linepath*:
shows (*f has_contour_integral i*) (*linepath a b*) ⟷
((λ*x*. *f*(*linepath a b x*) ∗ (*b* − *a*)) *has_integral i*) {*0..1*}
by (*simp add*: *has_contour_integral*)

lemma *has_contour_integral_trivial* [*iff*]: (*f has_contour_integral 0*) (*linepath a a*)
by (*simp add*: *has_contour_integral_linepath*)

lemma *has_contour_integral_trivial_iff* [*simp*]: (*f has_contour_integral i*) (*linepath a a*) ⟷ *i=0*
using *has_contour_integral_unique* **by** *blast*

lemma *contour_integral_trivial* [*simp*]: *contour_integral* (*linepath a a*) *f* = *0*
using *has_contour_integral_trivial contour_integral_unique* **by** *blast*

1.6 Relation to subpath construction

lemma *has_contour_integral_subpath_refl* [*iff*]: (*f has_contour_integral 0*) (*subpath u u g*)
by (*simp add*: *has_contour_integral subpath_def*)

lemma *contour_integrable_subpath_refl* [*iff*]: *f contour_integrable_on* (*subpath u u g*)
using *has_contour_integral_subpath_refl contour_integrable_on_def* **by** *blast*

lemma *contour_integral_subpath_refl* [*simp*]: *contour_integral* (*subpath u u g*) *f* = *0*
by (*simp add*: *contour_integral_unique*)

lemma *has_contour_integral_subpath*:
assumes *f*: *f contour_integrable_on g* **and** *g*: *valid_path g*
and *uv*: *u* ∈ {*0..1*} *v* ∈ {*0..1*} *u* ≤ *v*
shows (*f has_contour_integral integral* {*u..v*} (λ*x*. *f*(*g x*) ∗ *vector_derivative g* (*at x*)))
(*subpath u v g*)
proof (*cases v*=*u*)
case *True*
then show *?thesis*
using *f* **by** (*simp add*: *contour_integrable_on_def subpath_def has_contour_integral*)
next
case *False*
obtain *S* **where** *S*: ⋀*x*. *x* ∈ {*0..1*} − *S* ⟹ *g differentiable at x* **and** *fs*: *finite S*
using *g* **unfolding** *piecewise_C1_differentiable_on_def C1_differentiable_on_eq valid_path_def* **by** *blast*
have §: (λ*t*. *f* (*g t*) ∗ *vector_derivative g* (*at t*)) *integrable_on* {*u..v*}
using *contour_integrable_on f integrable_on_subinterval uv* **by** *fastforce*
then have ∗: ((λ*x*. *f* (*g* ((*v* − *u*) ∗ *x* + *u*)) ∗ *vector_derivative g* (*at* ((*v* − *u*) ∗ *x* + *u*)))
has_integral (*1* / (*v* − *u*)) ∗ *integral* {*u..v*} (λ*t*. *f* (*g t*) ∗ *vector_derivative g* (*at t*)))
{*0..1*}
using *uv False* **unfolding** *has_integral_integral*
apply *simp*
apply (*drule has_integral_affinity* [**where** *m*=*v*−*u* **and** *c*=*u*, *simplified*])
apply (*simp_all add*: *image_affinity_atLeastAtMost_div_diff scaleR_conv_of_real*)
apply (*simp add*: *divide_simps*)
done

have *vd*: *vector_derivative* (λ*x*. *g* ((*v*−*u*) ∗ *x* + *u*)) (*at x*) = (*v*−*u*) ∗$_R$ *vector_derivative g* (*at* ((*v*−*u*) ∗ *x* + *u*))
if *x* ∈ {*0..1*} *x* ∉ (λ*t*. (*v*−*u*) ∗$_R$ *t* + *u*) −‘ *S* **for** *x*
proof (*rule vector_derivative_at* [*OF vector_diff_chain_at* [*simplified o_def*]])
show ((λ*x*. (*v* − *u*) ∗ *x* + *u*) *has_vector_derivative v* − *u*) (*at x*)
by (*intro derivative_eq_intros* | *simp*)+
qed (*use S uv mult_left_le* [*of x v*−*u*] *that* **in** ‹*auto simp*: *vector_derivative_works*›)

have *fin*: *finite* ((λ*t*. (*v* − *u*) ∗$_R$ *t* + *u*) −‘ *S*)
using *fs* **by** (*auto simp*: *inj_on_def False finite_vimageI*)
show *?thesis*
unfolding *subpath_def has_contour_integral*
apply (*rule has_integral_spike_finite* [*OF fin*])
using *has_integral_cmul* [*OF* ∗, **where** *c* = *v*−*u*] *fs assms*
by (*auto simp*: *False vd scaleR_conv_of_real*)

qed

lemma *contour_integrable_subpath*:
assumes *f contour_integrable_on g valid_path g u* ∈ {*0..1*} *v* ∈ {*0..1*}
shows *f contour_integrable_on* (*subpath u v g*)
proof (*cases u v rule*: *linorder_class.le_cases*)
case *le*
then show *?thesis*
by (*metis contour_integrable_on_def has_contour_integral_subpath* [*OF assms*])
next
case *ge*
with *assms* **show** *?thesis*
by (*metis* (*no_types*, *lifting*) *contour_integrable_on_def contour_integrable_reversepath_eq has_contour_integral_subpath reversepath_subpath valid_path_subpath*)
qed

lemma *has_integral_contour_integral_subpath*:
assumes *f contour_integrable_on g valid_path g u* ∈ {*0..1*} *v* ∈ {*0..1*} *u* ≤ *v*
shows (((λ*x*. *f*(*g x*) ∗ *vector_derivative g* (*at x*)))
has_integral contour_integral (*subpath u v g*) *f*) {*u..v*}
using *assms*
proof −
have (λ*r*. *f* (*g r*) ∗ *vector_derivative g* (*at r*)) *integrable_on* {*u..v*}
by (*metis* (*full_types*) *assms*(*1*) *assms*(*3*) *assms*(*4*) *atLeastAtMost_iff atLeastAtMost_subset_iff contour_integrable_on integrable_on_subinterval*)
then have ((λ*r*. *f* (*g r*) ∗ *vector_derivative g* (*at r*)) *has_integral integral* {*u..v*} (λ*r*. *f* (*g r*) ∗ *vector_derivative g* (*at r*))) {*u..v*}
by *blast*
then show *?thesis*
by (*metis* (*full_types*) *assms contour_integral_unique has_contour_integral_subpath*)
qed

lemma *contour_integral_subcontour_integral*:
assumes *f contour_integrable_on g valid_path g u* ∈ {*0..1*} *v* ∈ {*0..1*} *u* ≤ *v*
shows *contour_integral* (*subpath u v g*) *f* =
integral {*u..v*} (λ*x*. *f*(*g x*) ∗ *vector_derivative g* (*at x*))
using *assms has_contour_integral_subpath contour_integral_unique* **by** *blast*

lemma *contour_integral_subpath_combine_less*:
assumes *f contour_integrable_on g valid_path g u* ∈ {*0..1*} *v* ∈ {*0..1*} *w* ∈ {*0..1*}
u<*v v*<*w*
shows *contour_integral* (*subpath u v g*) *f* + *contour_integral* (*subpath v w g*) *f* =
contour_integral (*subpath u w g*) *f*
proof −
have (λ*x*. *f* (*g x*) ∗ *vector_derivative g* (*at x*)) *integrable_on* {*u..w*}
using *integrable_on_subcbox* [**where** *a*=*u* **and** *b*=*w* **and** *S* = {*0..1*}] *assms*
by (*auto simp*: *contour_integrable_on*)

with *assms* **show** *?thesis*
by (*auto simp*: *contour_integral_subcontour_integral Henstock_Kurzweil_Integration.integral_combine*)
qed

lemma *contour_integral_subpath_combine*:
assumes *f contour_integrable_on g valid_path g u* ∈ {*0..1*} *v* ∈ {*0..1*} *w* ∈ {*0..1*}
shows *contour_integral* (*subpath u v g*) *f* + *contour_integral* (*subpath v w g*) *f* =
contour_integral (*subpath u w g*) *f*
proof (*cases u*≠*v* ∧ *v*≠*w* ∧ *u*≠*w*)
case *True*
have ∗: *subpath v u g* = *reversepath*(*subpath u v g*) ∧
subpath w u g = *reversepath*(*subpath u w g*) ∧
subpath w v g = *reversepath*(*subpath v w g*)
by (*auto simp*: *reversepath_subpath*)
have *u* < *v* ∧ *v* < *w* ∨
u < *w* ∧ *w* < *v* ∨
v < *u* ∧ *u* < *w* ∨
v < *w* ∧ *w* < *u* ∨
w < *u* ∧ *u* < *v* ∨
w < *v* ∧ *v* < *u*
using *True assms* **by** *linarith*
with *assms* **show** *?thesis*
using *contour_integral_subpath_combine_less* [*of f g u v w*]
contour_integral_subpath_combine_less [*of f g u w v*]
contour_integral_subpath_combine_less [*of f g v u w*]
contour_integral_subpath_combine_less [*of f g v w u*]
contour_integral_subpath_combine_less [*of f g w u v*]
contour_integral_subpath_combine_less [*of f g w v u*]
by (*elim disjE*) (*auto simp*: ∗ *contour_integral_reversepath contour_integrable_subpath valid_path_subpath algebra_simps*)
next
case *False*
with *assms* **show** *?thesis*
by (*metis add.right_neutral contour_integral_reversepath contour_integral_subpath_refl diff_0 eq_diff_eq add_0 reversepath_subpath valid_path_subpath*)
qed

lemma *contour_integral_integral*:
contour_integral g f = *integral* {*0..1*} (λ*x*. *f* (*g x*) ∗ *vector_derivative g* (*at x*))
by (*simp add*: *contour_integral_def integral_def has_contour_integral contour_integrable_on*)

lemma *contour_integral_cong*:
assumes *g* = *g*′ ⋀*x*. *x* ∈ *path_image g* ⟹ *f x* = *f*′ *x*
shows *contour_integral g f* = *contour_integral g*′ *f*′
unfolding *contour_integral_integral* **using** *assms*
by (*intro integral_cong*) (*auto simp*: *path_image_def*)

Contour integral along a segment on the real axis

lemma *has_contour_integral_linepath_Reals_iff*:
fixes *a b* :: *complex* **and** *f* :: *complex* ⇒ *complex*
assumes *a* ∈ *Reals b* ∈ *Reals Re a* < *Re b*
shows (*f has_contour_integral I*) (*linepath a b*) ⟷
((λ*x*. *f* (*of_real x*)) *has_integral I*) {*Re a*..*Re b*}
proof −
from *assms* **have** [*simp*]: *of_real* (*Re a*) = *a of_real* (*Re b*) = *b*
by (*simp_all add*: *complex_eq_iff*)
from *assms* **have** *a* ≠ *b* **by** *auto*
have ((λ*x*. *f* (*of_real x*)) *has_integral I*) (*cbox* (*Re a*) (*Re b*)) ⟷
((λ*x*. *f* (*a* + *b* ∗ *of_real x* − *a* ∗ *of_real x*)) *has_integral I* /*R* (*Re b* − *Re a*)) {*0*..*1*}
by (*subst has_integral_affinity_iff* [*of Re b* − *Re a* _ *Re a*, *symmetric*])
(*insert assms*, *simp_all add*: *field_simps scaleR_conv_of_real*)
also have (λ*x*. *f* (*a* + *b* ∗ *of_real x* − *a* ∗ *of_real x*)) =
(λ*x*. (*f* (*a* + *b* ∗ *of_real x* − *a* ∗ *of_real x*) ∗ (*b* − *a*)) /*R* (*Re b* − *Re a*))
using ‹*a* ≠ *b*› **by** (*auto simp*: *field_simps fun_eq_iff scaleR_conv_of_real*)
also have (. . . *has_integral I* /*R* (*Re b* − *Re a*)) {*0*..*1*} ⟷
((λ*x*. *f* (*linepath a b x*) ∗ (*b* − *a*)) *has_integral I*) {*0*..*1*} **using** *assms*
by (*subst has_integral_cmul_iff*) (*auto simp*: *linepath_def scaleR_conv_of_real algebra_simps*)
also have . . . ⟷ (*f has_contour_integral I*) (*linepath a b*) **unfolding** *has_contour_integral_def*
by (*intro has_integral_cong*) (*simp add*: *vector_derivative_linepath_within*)
finally show *?thesis* **by** *simp*
qed

lemma *contour_integrable_linepath_Reals_iff*:
fixes *a b* :: *complex* **and** *f* :: *complex* ⇒ *complex*
assumes *a* ∈ *Reals b* ∈ *Reals Re a* < *Re b*
shows (*f contour_integrable_on linepath a b*) ⟷
(λ*x*. *f* (*of_real x*)) *integrable_on* {*Re a*..*Re b*}
using *has_contour_integral_linepath_Reals_iff*[*OF assms*, *of f*]
by (*auto simp*: *contour_integrable_on_def integrable_on_def*)

lemma *contour_integral_linepath_Reals_eq*:
fixes *a b* :: *complex* **and** *f* :: *complex* ⇒ *complex*
assumes *a* ∈ *Reals b* ∈ *Reals Re a* < *Re b*
shows *contour_integral* (*linepath a b*) *f* = *integral* {*Re a*..*Re b*} (λ*x*. *f* (*of_real x*))
proof (*cases f contour_integrable_on linepath a b*)
case *True*
thus *?thesis* **using** *has_contour_integral_linepath_Reals_iff*[*OF assms*, *of f*]
using *has_contour_integral_integral has_contour_integral_unique* **by** *blast*
next
case *False*
thus *?thesis* **using** *contour_integrable_linepath_Reals_iff*[*OF assms*, *of f*]
by (*simp add*: *not_integrable_contour_integral not_integrable_integral*)

qed

1.7 Cauchy's theorem where there's a primitive

lemma *contour_integral_primitive_lemma*:
fixes $f :: complex \Rightarrow complex$ **and** $g :: real \Rightarrow complex$
assumes $a \leq b$
and $\bigwedge x.\ x \in S \Longrightarrow (f\ has\_field\_derivative\ f'\ x)\ (at\ x\ within\ S)$
and $g\ piecewise\_differentiable\_on\ \{a..b\}$ $\bigwedge x.\ x \in \{a..b\} \Longrightarrow g\ x \in S$
shows $((\lambda x.\ f'(g\ x) * vector\_derivative\ g\ (at\ x\ within\ \{a..b\}))$
$has\_integral\ (f(g\ b) - f(g\ a)))\ \{a..b\}$
proof −
obtain K **where** *finite* K **and** K: $\forall x{\in}\{a..b\} - K.\ g\ differentiable\ (at\ x\ within\ \{a..b\})$ **and** *cg*: $continuous\_on\ \{a..b\}\ g$
using *assms* **by** (*auto simp*: *piecewise_differentiable_on_def*)
have $continuous\_on\ (g\ `\ \{a..b\})\ f$
using *assms*
by (*metis field_differentiable_def field_differentiable_imp_continuous_at continuous_on_eq_continuous_within continuous_on_subset image_subset_iff*)
then have *cfg*: $continuous\_on\ \{a..b\}\ (\lambda x.\ f\ (g\ x))$
by (*rule continuous_on_compose* [*OF cg, unfolded o_def*])
{ fix x::*real*
assume a: $a < x$ **and** b: $x < b$ **and** xk: $x \notin K$
then have $g\ differentiable\ at\ x\ within\ \{a..b\}$
using K **by** (*simp add*: *differentiable_at_withinI*)
then have $(g\ has\_vector\_derivative\ vector\_derivative\ g\ (at\ x\ within\ \{a..b\}))$ $(at\ x\ within\ \{a..b\})$
by (*simp add*: *vector_derivative_works has_field_derivative_def scaleR_conv_of_real*)
then have *gdiff*: $(g\ has\_derivative\ (\lambda u.\ u * vector\_derivative\ g\ (at\ x\ within\ \{a..b\})))\ (at\ x\ within\ \{a..b\})$
by (*simp add*: *has_vector_derivative_def scaleR_conv_of_real*)
have $(f\ has\_field\_derivative\ (f'\ (g\ x)))\ (at\ (g\ x)\ within\ g\ `\ \{a..b\})$
using *assms* **by** (*metis a atLeastAtMost_iff b DERIV_subset image_subset_iff less_eq_real_def*)
then have *fdiff*: $(f\ has\_derivative\ (*)\ (f'\ (g\ x)))\ (at\ (g\ x)\ within\ g\ `\ \{a..b\})$
by (*simp add*: *has_field_derivative_def*)
have $((\lambda x.\ f\ (g\ x))\ has\_vector\_derivative\ f'\ (g\ x) * vector\_derivative\ g\ (at\ x\ within\ \{a..b\}))\ (at\ x\ within\ \{a..b\})$
using *diff_chain_within* [*OF gdiff fdiff*]
by (*simp add*: *has_vector_derivative_def scaleR_conv_of_real o_def mult_ac*)
} note $* = this$
show *?thesis*
using *assms cfg* $*$
by (*force simp*: *at_within_Icc_at intro*: *fundamental_theorem_of_calculus_interior_strong* [*OF* ‹*finite K*›])
qed

lemma *contour_integral_primitive*:
assumes $\bigwedge x.\ x \in S \Longrightarrow (f\ has\_field\_derivative\ f'\ x)\ (at\ x\ within\ S)$

and *valid_path g path_image g* $\subseteq$ *S*
shows (*f′ has_contour_integral* (*f*(*pathfinish g*) − *f*(*pathstart g*))) *g*
using *assms*
apply (*simp add*: *valid_path_def path_image_def pathfinish_def pathstart_def has_contour_integral_def*)
apply (*auto intro*!: *piecewise_C1_imp_differentiable contour_integral_primitive_lemma* [*of 0 1 S*])
done

corollary *Cauchy_theorem_primitive*:
assumes $\bigwedge x.\ x \in S \Longrightarrow$ (*f has_field_derivative f′ x*) (*at x within S*)
and *valid_path g path_image g* $\subseteq$ *S pathfinish g* = *pathstart g*
shows (*f′ has_contour_integral 0*) *g*
using *assms* **by** (*metis diff_self contour_integral_primitive*)

Existence of path integral for continuous function

lemma *contour_integrable_continuous_linepath*:
assumes *continuous_on* (*closed_segment a b*) *f*
shows *f contour_integrable_on* (*linepath a b*)
proof −
have *continuous_on* (*closed_segment a b*) (λ*x*. *f x* ∗ (*b* − *a*))
by (*rule continuous_intros* | *simp add*: *assms*)+
then have *continuous_on* {*0..1*} (λ*x*. *f* (*linepath a b x*) ∗ (*b* − *a*))
by (*metis* (*no_types*, *lifting*) *continuous_on_compose continuous_on_cong continuous_on_linepath linepath_image_01 o_apply*)
then have (λ*x*. *f* (*linepath a b x*) ∗
vector_derivative (*linepath a b*)
(*at x within* {*0..1*})) *integrable_on*
{*0..1*}
by (*metis* (*no_types*, *lifting*) *continuous_on_cong integrable_continuous_real vector_derivative_linepath_within*)
then show *?thesis*
by (*simp add*: *contour_integrable_on_def has_contour_integral_def integrable_on_def* [*symmetric*])
qed

lemma *has_field_der_id*: ((λ*x*. $x^2/2$) *has_field_derivative x*) (*at x*)
by (*rule has_derivative_imp_has_field_derivative*)
(*rule derivative_intros* | *simp*)+

lemma *contour_integral_id* [*simp*]: *contour_integral* (*linepath a b*) (λ*y*. *y*) = (*b*^2 − *a*^2)/2
using *contour_integral_primitive* [*of UNIV* λ*x*. *x*^2/2 λ*x*. *x linepath a b*] *contour_integral_unique*
by (*simp add*: *has_field_der_id*)

lemma *contour_integrable_on_const* [*iff*]: (λ*x*. *c*) *contour_integrable_on* (*linepath a b*)
by (*simp add*: *contour_integrable_continuous_linepath*)

lemma *contour_integrable_on_id* [*iff*]: $(\lambda x.\ x)$ *contour_integrable_on* (*linepath a b*)
by (*simp add*: *contour_integrable_continuous_linepath*)

1.8 Arithmetical combining theorems

lemma *has_contour_integral_neg*:
$(f$ *has_contour_integral* $i)\ g \Longrightarrow ((\lambda x.\ -(f\ x))$ *has_contour_integral* $(-i))\ g$
by (*simp add*: *has_integral_neg has_contour_integral_def*)

lemma *has_contour_integral_add*:
$[\![(f1$ *has_contour_integral* $i1)\ g;\ (f2$ *has_contour_integral* $i2)\ g]\!]$
$\Longrightarrow ((\lambda x.\ f1\ x + f2\ x)$ *has_contour_integral* $(i1 + i2))\ g$
by (*simp add*: *has_integral_add has_contour_integral_def algebra_simps*)

lemma *has_contour_integral_diff*:
$[\![(f1$ *has_contour_integral* $i1)\ g;\ (f2$ *has_contour_integral* $i2)\ g]\!]$
$\Longrightarrow ((\lambda x.\ f1\ x - f2\ x)$ *has_contour_integral* $(i1 - i2))\ g$
by (*simp add*: *has_integral_diff has_contour_integral_def algebra_simps*)

lemma *has_contour_integral_lmul*:
$(f$ *has_contour_integral* $i)\ g \Longrightarrow ((\lambda x.\ c * (f\ x))$ *has_contour_integral* $(c*i))\ g$
by (*simp add*: *has_contour_integral_def algebra_simps has_integral_mult_right*)

lemma *has_contour_integral_rmul*:
$(f$ *has_contour_integral* $i)\ g \Longrightarrow ((\lambda x.\ (f\ x) * c)$ *has_contour_integral* $(i*c))\ g$
by (*simp add*: *mult.commute has_contour_integral_lmul*)

lemma *has_contour_integral_div*:
$(f$ *has_contour_integral* $i)\ g \Longrightarrow ((\lambda x.\ f\ x/c)$ *has_contour_integral* $(i/c))\ g$
by (*simp add*: *field_class.field_divide_inverse*) (*metis has_contour_integral_rmul*)

lemma *has_contour_integral_eq*:
$[\![(f$ *has_contour_integral* $y)\ p;\ \bigwedge x.\ x \in$ *path_image* $p \Longrightarrow f\ x = g\ x]\!] \Longrightarrow (g$ *has_contour_integral* $y)\ p$
by (*metis* (*mono_tags*, *lifting*) *has_contour_integral_def has_integral_eq image_eqI path_image_def*)

lemma *has_contour_integral_bound_linepath*:
assumes $(f$ *has_contour_integral* $i)$ (*linepath a b*)
$0 \le B$ **and** B: $\bigwedge x.\ x \in$ *closed_segment a b* $\Longrightarrow$ *norm*$(f\ x) \le B$
shows *norm* $i \le B * norm(b - a)$
proof −
have *norm* $i \le (B *$ *norm* $(b - a)) *$ *content* (*cbox 0* (*1*::*real*))
proof (*rule has_integral_bound*
[*of* _ $\lambda x.\ f$ (*linepath a b x*) $*$ *vector_derivative* (*linepath a b*) (*at x within* $\{0..1\}$)])
show *cmod* $(f$ (*linepath a b x*) $*$ *vector_derivative* (*linepath a b*) (*at x within*

$\{0..1\}))$
$\leq B * cmod\ (b - a)$
if $x \in cbox\ 0\ 1$ **for** x::*real*
using *that box_real*(2) *norm_mult*
by (*metis B linepath_in_path mult_right_mono norm_ge_zero vector_derivative_linepath_within*)
qed (*use assms has_contour_integral_def* **in** *auto*)
then show *?thesis*
by (*auto simp*: *content_real*)
qed

lemma *has_contour_integral_const_linepath*: $((\lambda x.\ c)$ *has_contour_integral* $c*(b - a))(linepath\ a\ b)$
unfolding *has_contour_integral_linepath*
by (*metis content_real diff_0_right has_integral_const_real lambda_one of_real_1 scaleR_conv_of_real zero_le_one*)

lemma *has_contour_integral_0*: $((\lambda x.\ 0)$ *has_contour_integral* $0)\ g$
by (*simp add*: *has_contour_integral_def*)

lemma *has_contour_integral_is_0*:
$(\bigwedge z.\ z \in path\_image\ g \Longrightarrow f\ z = 0) \Longrightarrow (f$ *has_contour_integral* $0)\ g$
by (*rule has_contour_integral_eq* [*OF has_contour_integral_0*]) *auto*

lemma *has_contour_integral_sum*:
$[\![finite\ s;\ \bigwedge a.\ a \in s \Longrightarrow (f\ a$ *has_contour_integral* $i\ a)\ p]\!]$
$\Longrightarrow ((\lambda x.\ sum\ (\lambda a.\ f\ a\ x)\ s)$ *has_contour_integral sum* $i\ s)\ p$
by (*induction s rule*: *finite_induct*) (*auto simp*: *has_contour_integral_0 has_contour_integral_add*)

1.9 Operations on path integrals

lemma *contour_integral_const_linepath* [*simp*]: *contour_integral* $(linepath\ a\ b)$ $(\lambda x.\ c) = c*(b - a)$
by (*rule contour_integral_unique* [*OF has_contour_integral_const_linepath*])

lemma *contour_integral_neg*:
f *contour_integrable_on* $g \Longrightarrow$ *contour_integral* $g\ (\lambda x.\ -(f\ x)) = -($*contour_integral* $g\ f)$
by (*simp add*: *contour_integral_unique has_contour_integral_integral has_contour_integral_neg*)

lemma *contour_integral_add*:
f1 contour_integrable_on $g \Longrightarrow$ *f2 contour_integrable_on* $g \Longrightarrow$ *contour_integral* $g\ (\lambda x.\ f1\ x + f2\ x) =$
contour_integral g f1 $+$ *contour_integral g f2*
by (*simp add*: *contour_integral_unique has_contour_integral_integral has_contour_integral_add*)

lemma *contour_integral_diff*:
f1 contour_integrable_on $g \Longrightarrow$ *f2 contour_integrable_on* $g \Longrightarrow$ *contour_integral* $g\ (\lambda x.\ f1\ x - f2\ x) =$
contour_integral g f1 $-$ *contour_integral g f2*

by (*simp add*: *contour_integral_unique has_contour_integral_integral has_contour_integral_diff*)

lemma *contour_integral_lmul*:
shows *f contour_integrable_on g*
⟹ *contour_integral g* (λ*x*. *c* ∗ *f x*) = *c*∗*contour_integral g f*
by (*simp add*: *contour_integral_unique has_contour_integral_integral has_contour_integral_lmul*)

lemma *contour_integral_rmul*:
shows *f contour_integrable_on g*
⟹ *contour_integral g* (λ*x*. *f x* ∗ *c*) = *contour_integral g f* ∗ *c*
by (*simp add*: *contour_integral_unique has_contour_integral_integral has_contour_integral_rmul*)

lemma *contour_integral_div*:
shows *f contour_integrable_on g*
⟹ *contour_integral g* (λ*x*. *f x* / *c*) = *contour_integral g f* / *c*
by (*simp add*: *contour_integral_unique has_contour_integral_integral has_contour_integral_div*)

lemma *contour_integral_eq*:
(⋀*x*. *x* ∈ *path_image p* ⟹ *f x* = *g x*) ⟹ *contour_integral p f* = *contour_integral p g*
using *contour_integral_cong contour_integral_def* **by** *fastforce*

lemma *contour_integral_eq_0*:
(⋀*z*. *z* ∈ *path_image g* ⟹ *f z* = 0) ⟹ *contour_integral g f* = 0
by (*simp add*: *has_contour_integral_is_0 contour_integral_unique*)

lemma *contour_integral_bound_linepath*:
shows
⟦*f contour_integrable_on* (*linepath a b*);
0 ≤ *B*; ⋀*x*. *x* ∈ *closed_segment a b* ⟹ *norm*(*f x*) ≤ *B*⟧
⟹ *norm*(*contour_integral* (*linepath a b*) *f*) ≤ *B*∗*norm*(*b* − *a*)
using *has_contour_integral_bound_linepath* [*of f*]
by (*auto simp*: *has_contour_integral_integral*)

lemma *contour_integral_0* [*simp*]: *contour_integral g* (λ*x*. 0) = 0
by (*simp add*: *contour_integral_unique has_contour_integral_0*)

lemma *contour_integral_sum*:
⟦*finite s*; ⋀*a*. *a* ∈ *s* ⟹ (*f a*) *contour_integrable_on p*⟧
⟹ *contour_integral p* (λ*x*. *sum* (λ*a*. *f a x*) *s*) = *sum* (λ*a*. *contour_integral p* (*f a*)) *s*
by (*auto simp*: *contour_integral_unique has_contour_integral_sum has_contour_integral_integral*)

lemma *contour_integrable_eq*:
⟦*f contour_integrable_on p*; ⋀*x*. *x* ∈ *path_image p* ⟹ *f x* = *g x*⟧ ⟹ *g contour_integrable_on p*
unfolding *contour_integrable_on_def*
by (*metis has_contour_integral_eq*)

1.10 Arithmetic theorems for path integrability

lemma *contour_integrable_neg*:
  f contour_integrable_on g $\Longrightarrow$ $(\lambda x.\ -(f\ x))$ *contour_integrable_on g*
 using *has_contour_integral_neg contour_integrable_on_def* **by** *blast*

lemma *contour_integrable_add*:
  $[\![$*f1 contour_integrable_on g*; *f2 contour_integrable_on g*$]\!] \Longrightarrow (\lambda x.\ f1\ x + f2\ x)$ *contour_integrable_on g*
 using *has_contour_integral_add contour_integrable_on_def*
 by *fastforce*

lemma *contour_integrable_diff*:
  $[\![$*f1 contour_integrable_on g*; *f2 contour_integrable_on g*$]\!] \Longrightarrow (\lambda x.\ f1\ x - f2\ x)$ *contour_integrable_on g*
 using *has_contour_integral_diff contour_integrable_on_def*
 by *fastforce*

lemma *contour_integrable_lmul*:
  f contour_integrable_on g $\Longrightarrow$ $(\lambda x.\ c * f\ x)$ *contour_integrable_on g*
 using *has_contour_integral_lmul contour_integrable_on_def*
 by *fastforce*

lemma *contour_integrable_rmul*:
  f contour_integrable_on g $\Longrightarrow$ $(\lambda x.\ f\ x * c)$ *contour_integrable_on g*
 using *has_contour_integral_rmul contour_integrable_on_def*
 by *fastforce*

lemma *contour_integrable_div*:
  f contour_integrable_on g $\Longrightarrow$ $(\lambda x.\ f\ x\ /\ c)$ *contour_integrable_on g*
 using *has_contour_integral_div contour_integrable_on_def*
 by *fastforce*

lemma *contour_integrable_sum*:
  $[\![$*finite s*; $\bigwedge a.\ a \in s \Longrightarrow (f\ a)$ *contour_integrable_on p*$]\!]$
   $\Longrightarrow (\lambda x.\ sum\ (\lambda a.\ f\ a\ x)\ s)$ *contour_integrable_on p*
 unfolding *contour_integrable_on_def*
 by (*metis has_contour_integral_sum*)

1.11 Reversing a path integral

lemma *has_contour_integral_reverse_linepath*:
  $(f$ *has_contour_integral i*$)$ $(linepath\ a\ b)$
   $\Longrightarrow (f$ *has_contour_integral* $(-i))$ $(linepath\ b\ a)$
 using *has_contour_integral_reversepath valid_path_linepath* **by** *fastforce*

lemma *contour_integral_reverse_linepath*:
  continuous_on $(closed\_segment\ a\ b)\ f$
   $\Longrightarrow$ *contour_integral* $(linepath\ a\ b)\ f = -$ $(contour\_integral(linepath\ b\ a)\ f)$
 by (*metis contour_integrable_continuous_linepath contour_integral_unique has_contour_integral_integral*

has_contour_integral_reverse_linepath)

Splitting a path integral in a flat way.*)

lemma *has_contour_integral_split*:
assumes *f*: (*f has_contour_integral i*) (*linepath a c*) (*f has_contour_integral j*) (*linepath c b*)
and *k*: $0 \le k\ k \le 1$
and *c*: $c - a = k *_R (b - a)$
shows (*f has_contour_integral* ($i + j$)) (*linepath a b*)
proof (*cases* $k = 0 \vee k = 1$)
case *True*
then show *?thesis*
using *assms* **by** *auto*
next
case *False*
then have *k*: $0 < k\ k < 1$ *complex_of_real* $k \neq 1$
using *assms* **by** *auto*
have *c'*: $c = k *_R (b - a) + a$
by (*metis diff_add_cancel c*)
have *bc*: $(b - c) = (1 - k) *_R (b - a)$
by (*simp add*: *algebra_simps c'*)
{ assume $*$: $((\lambda x.\ f\ ((1 - x) *_R a + x *_R c) * (c - a))$ *has_integral i*) $\{0..1\}$
have $\bigwedge x.\ (x\ /\ k) *_R a + ((k - x)\ /\ k) *_R a = a$
using *False* **by** (*simp add*: *field_split_simps flip*: *real_vector.scale_left_distrib*)
then have $\bigwedge x.\ ((k - x)\ /\ k) *_R a + (x\ /\ k) *_R c = (1 - x) *_R a + x *_R b$
using *False* **by** (*simp add*: *c' algebra_simps*)
then have $((\lambda x.\ f\ ((1 - x) *_R a + x *_R b) * (b - a))$ *has_integral i*) $\{0..k\}$
using *k has_integral_affinity01* [*OF* $*$, *of inverse k 0*]
by (*force dest*: *has_integral_cmul* [**where** $c = inverse\ k$]
simp add: *divide_simps mult.commute* [*of* _ *k*] *image_affinity_atLeastAtMost c*)
} note *fi* = *this*
{ assume $*$: $((\lambda x.\ f\ ((1 - x) *_R c + x *_R b) * (b - c))$ *has_integral j*) $\{0..1\}$
have $**$: $\bigwedge x.\ (((1 - x)\ /\ (1 - k)) *_R c + ((x - k)\ /\ (1 - k)) *_R b) = ((1 - x) *_R a + x *_R b)$
using *k* **unfolding** *c' scaleR_conv_of_real*
apply (*simp add*: *divide_simps*)
apply (*simp add*: *distrib_right distrib_left right_diff_distrib left_diff_distrib*)
done
have $((\lambda x.\ f\ ((1 - x) *_R a + x *_R b) * (b - a))$ *has_integral j*) $\{k..1\}$
using *k has_integral_affinity01* [*OF* $*$, *of inverse*$(1 - k)$ $-(k/(1 - k))$]
apply (*simp add*: *divide_simps mult.commute* [*of* _ $1-k$] *image_affinity_atLeastAtMost* $**$ *bc*)
apply (*auto dest*: *has_integral_cmul* [**where** $k = (1 - k) *_R j$ **and** $c = inverse\ (1 - k)$])
done
} note *fj* = *this*
show *?thesis*
using *f k* **unfolding** *has_contour_integral_linepath*

by (*simp add*: *linepath_def has_integral_combine* [*OF _ _ fi fj*])
qed

lemma *continuous_on_closed_segment_transform*:
assumes *f*: *continuous_on* (*closed_segment a b*) *f*
and *k*: *0* ≤ *k k* ≤ *1*
and *c*: *c* − *a* = *k* ∗*R* (*b* − *a*)
shows *continuous_on* (*closed_segment a c*) *f*
proof −
have *c′*: *c* = (*1* − *k*) ∗*R* *a* + *k* ∗*R* *b*
using *c* **by** (*simp add*: *algebra_simps*)
have *closed_segment a c* ⊆ *closed_segment a b*
by (*metis c′ ends_in_segment*(*1*) *in_segment*(*1*) *k subset_closed_segment*)
then show *continuous_on* (*closed_segment a c*) *f*
by (*rule continuous_on_subset* [*OF f*])
qed

lemma *contour_integral_split*:
assumes *f*: *continuous_on* (*closed_segment a b*) *f*
and *k*: *0* ≤ *k k* ≤ *1*
and *c*: *c* − *a* = *k* ∗*R* (*b* − *a*)
shows *contour_integral*(*linepath a b*) *f* = *contour_integral*(*linepath a c*) *f* + *contour_integral*(*linepath c b*) *f*
proof −
have *c′*: *c* = (*1* − *k*) ∗*R* *a* + *k* ∗*R* *b*
using *c* **by** (*simp add*: *algebra_simps*)
have *closed_segment a c* ⊆ *closed_segment a b*
by (*metis c′ ends_in_segment*(*1*) *in_segment*(*1*) *k subset_closed_segment*)
moreover have *closed_segment c b* ⊆ *closed_segment a b*
by (*metis c′ ends_in_segment*(*2*) *in_segment*(*1*) *k subset_closed_segment*)
ultimately
have ∗: *continuous_on* (*closed_segment a c*) *f continuous_on* (*closed_segment c b*) *f*
by (*auto intro*: *continuous_on_subset* [*OF f*])
show *?thesis*
by (*rule contour_integral_unique*) (*meson* ∗ *c contour_integrable_continuous_linepath has_contour_integral_integral has_contour_integral_split k*)
qed

lemma *contour_integral_split_linepath*:
assumes *f*: *continuous_on* (*closed_segment a b*) *f*
and *c*: *c* ∈ *closed_segment a b*
shows *contour_integral*(*linepath a b*) *f* = *contour_integral*(*linepath a c*) *f* + *contour_integral*(*linepath c b*) *f*
using *c* **by** (*auto simp*: *closed_segment_def algebra_simps intro*!: *contour_integral_split* [*OF f*])

1.12 Reversing the order in a double path integral

The condition is stronger than needed but it's often true in typical situations

lemma *fst_im_cbox* [*simp*]: *cbox c d* ≠ {} ⟹ (*fst* ‘ *cbox* (*a*,*c*) (*b*,*d*)) = *cbox a b*
by (*auto simp*: *cbox_Pair_eq*)

lemma *snd_im_cbox* [*simp*]: *cbox a b* ≠ {} ⟹ (*snd* ‘ *cbox* (*a*,*c*) (*b*,*d*)) = *cbox c d*
by (*auto simp*: *cbox_Pair_eq*)

proposition *contour_integral_swap*:
assumes *fcon*: *continuous_on* (*path_image g* × *path_image h*) (λ(*y1*,*y2*). *f y1 y2*)
and *vp*: *valid_path g valid_path h*
and *gvcon*: *continuous_on* {*0..1*} (λ*t*. *vector_derivative g* (*at t*))
and *hvcon*: *continuous_on* {*0..1*} (λ*t*. *vector_derivative h* (*at t*))
shows *contour_integral g* (λ*w*. *contour_integral h* (*f w*)) =
contour_integral h (λ*z*. *contour_integral g* (λ*w*. *f w z*))
proof −
have *gcon*: *continuous_on* {*0..1*} *g* **and** *hcon*: *continuous_on* {*0..1*} *h*
using *assms* **by** (*auto simp*: *valid_path_def piecewise_C1_differentiable_on_def*)
have *fgh1*: ⋀*x*. (λ*t*. *f* (*g x*) (*h t*)) = (λ(*y1*,*y2*). *f y1 y2*) ∘ (λ*t*. (*g x*, *h t*))
by (*rule ext*) *simp*
have *fgh2*: ⋀*x*. (λ*t*. *f* (*g t*) (*h x*)) = (λ(*y1*,*y2*). *f y1 y2*) ∘ (λ*t*. (*g t*, *h x*))
by (*rule ext*) *simp*
have *fcon_im1*: ⋀*x*. *0* ≤ *x* ⟹ *x* ≤ *1* ⟹ *continuous_on* ((λ*t*. (*g x*, *h t*)) ‘ {*0..1*}) (λ(*x*, *y*). *f x y*)
by (*rule continuous_on_subset* [*OF fcon*]) (*auto simp*: *path_image_def*)
have *fcon_im2*: ⋀*x*. *0* ≤ *x* ⟹ *x* ≤ *1* ⟹ *continuous_on* ((λ*t*. (*g t*, *h x*)) ‘ {*0..1*}) (λ(*x*, *y*). *f x y*)
by (*rule continuous_on_subset* [*OF fcon*]) (*auto simp*: *path_image_def*)
have *continuous_on* (*cbox* (*0*, *0*) (*1*, *1*::*real*)) ((λ*x*. *vector_derivative g* (*at x*)) ∘ *fst*)
continuous_on (*cbox* (*0*, *0*) (*1*::*real*, *1*)) ((λ*x*. *vector_derivative h* (*at x*)) ∘ *snd*)
by (*rule continuous_intros* | *simp add*: *gvcon hvcon*)+
then have *gvcon′*: *continuous_on* (*cbox* (*0*, *0*) (*1*, *1*::*real*)) (λ*z*. *vector_derivative g* (*at* (*fst z*)))
and *hvcon′*: *continuous_on* (*cbox* (*0*, *0*) (*1*::*real*, *1*)) (λ*x*. *vector_derivative h* (*at* (*snd x*)))
by *auto*
have *continuous_on* (*cbox* (*0*, *0*) (*1*, *1*)) ((λ(*y1*, *y2*). *f y1 y2*) ∘ (λ*w*. ((*g* ∘ *fst*) *w*, (*h* ∘ *snd*) *w*)))
apply (*intro gcon hcon continuous_intros* | *simp*)+
apply (*auto simp*: *path_image_def intro*: *continuous_on_subset* [*OF fcon*])
done
then have *fgh*: *continuous_on* (*cbox* (*0*, *0*) (*1*, *1*)) (λ*x*. *f* (*g* (*fst x*)) (*h* (*snd x*)))
by *auto*

```
  have integral {0..1} (λx. contour_integral h (f (g x)) * vector_derivative g (at
x)) =
        integral {0..1} (λx. contour_integral h (λy. f (g x) y * vector_derivative g
(at x)))
  proof (rule integral_cong [OF contour_integral_rmul [symmetric]])
    have ⋀x. x ∈ {0..1} ⟹
         continuous_on {0..1} (λxa. f (g x) (h xa))
    by (subst fgh1) (rule fcon_im1 hcon continuous_intros | simp)+
    then show ⋀x. x ∈ {0..1} ⟹ f (g x) contour_integrable_on h
      unfolding contour_integrable_on
      using continuous_on_mult hvcon integrable_continuous_real by blast
  qed
  also have ... = integral {0..1}
                    (λy. contour_integral g (λx. f x (h y) * vector_derivative h (at
y)))
    unfolding contour_integral_integral
    apply (subst integral_swap_continuous [where 'a = real and 'b = real, of 0
0 1 1, simplified])
    subgoal
      by (rule fgh gvcon' hvcon' continuous_intros | simp add: split_def)+
    subgoal
      unfolding integral_mult_left [symmetric]
      by (simp only: mult_ac)
    done
  also have ... = contour_integral h (λz. contour_integral g (λw. f w z))
    unfolding contour_integral_integral integral_mult_left [symmetric]
    by (simp add: algebra_simps)
  finally show ?thesis
    by (simp add: contour_integral_integral)
qed

lemma valid_path_negatepath: valid_path γ ⟹ valid_path (uminus ∘ γ)
    unfolding o_def using piecewise_C1_differentiable_neg valid_path_def by
blast

lemma has_contour_integral_negatepath:
  assumes γ: valid_path γ and cint: ((λz. f (− z)) has_contour_integral − i) γ
  shows (f has_contour_integral i) (uminus ∘ γ)
proof −
 obtain S where cont: continuous_on {0..1} γ and finite S and diff: γ C1_differentiable_on
{0..1} − S
    using γ by (auto simp: valid_path_def piecewise_C1_differentiable_on_def)
  have ((λx. − (f (− γ x) * vector_derivative γ (at x within {0..1}))) has_integral
i) {0..1}
    using cint by (auto simp: has_contour_integral_def dest: has_integral_neg)
  then
   have ((λx. f (− γ x) * vector_derivative (uminus ∘ γ) (at x within {0..1}))
has_integral i) {0..1}
  proof (rule rev_iffD1 [OF _ has_integral_spike_eq])
```

show *negligible S*
by (*simp add*: ‹*finite S*› *negligible_finite*)
show *f* (− *γ x*) ∗ *vector_derivative* (*uminus* ∘ *γ*) (*at x within* {*0..1*}) =
− (*f* (− *γ x*) ∗ *vector_derivative γ* (*at x within* {*0..1*}))
if *x* ∈ {*0..1*} − *S* **for** *x*
proof −
have *vector_derivative* (*uminus* ∘ *γ*) (*at x within cbox 0 1*) = − *vector_derivative γ* (*at x within cbox 0 1*)
proof (*rule vector_derivative_within_cbox*)
show (*uminus* ∘ *γ has_vector_derivative* − *vector_derivative γ* (*at x within cbox 0 1*)) (*at x within cbox 0 1*)
using *that* **unfolding** *o_def*
by (*metis C1_differentiable_on_eq UNIV_I diff differentiable_subset has_vector_derivative_minus subsetI that vector_derivative_works*)
qed (*use that* **in** *auto*)
then show *?thesis*
by *simp*
qed
qed
then show *?thesis* **by** (*simp add*: *has_contour_integral_def*)
qed

lemma *contour_integrable_negatepath*:
assumes *γ*: *valid_path γ* **and** *pi*: (*λz. f* (− *z*)) *contour_integrable_on γ*
shows *f contour_integrable_on* (*uminus* ∘ *γ*)
by (*metis γ add.inverse_inverse contour_integrable_on_def has_contour_integral_negatepath pi*)

lemma *C1_differentiable_polynomial_function*:
fixes *p* :: *real* ⇒ *'a*::*euclidean_space*
shows *polynomial_function p* ⟹ *p C1_differentiable_on S*
by (*metis continuous_on_polymonial_function C1_differentiable_on_def has_vector_derivative_polynomi*

lemma *valid_path_polynomial_function*:
fixes *p* :: *real* ⇒ *'a*::*euclidean_space*
shows *polynomial_function p* ⟹ *valid_path p*
by (*force simp*: *valid_path_def piecewise_C1_differentiable_on_def continuous_on_polymonial_function C1_differentiable_polynomial_function*)

lemma *valid_path_subpath_trivial* [*simp*]:
fixes *g* :: *real* ⇒ *'a*::*euclidean_space*
shows *z* ≠ *g x* ⟹ *valid_path* (*subpath x x g*)
by (*simp add*: *subpath_def valid_path_polynomial_function*)

1.13 Partial circle path

definition *part_circlepath* :: [*complex*, *real*, *real*, *real*, *real*] ⇒ *complex*
where *part_circlepath z r s t* ≡ *λx. z* + *of_real r* ∗ *exp* (i ∗ *of_real* (*linepath s t x*))

lemma *pathstart_part_circlepath* [*simp*]:
  pathstart(*part_circlepath z r s t*) = *z* + *r*∗*exp*(i ∗ *s*)
by (*metis part_circlepath_def pathstart_def pathstart_linepath*)

lemma *pathfinish_part_circlepath* [*simp*]:
  pathfinish(*part_circlepath z r s t*) = *z* + *r*∗*exp*(i∗*t*)
by (*metis part_circlepath_def pathfinish_def pathfinish_linepath*)

lemma *reversepath_part_circlepath*[*simp*]:
  reversepath (*part_circlepath z r s t*) = *part_circlepath z r t s*
 unfolding *part_circlepath_def reversepath_def linepath_def*
 by (*auto simp*:*algebra_simps*)

lemma *has_vector_derivative_part_circlepath* [*derivative_intros*]:
  ((*part_circlepath z r s t*) *has_vector_derivative*
    (i ∗ *r* ∗ (*of_real t* − *of_real s*) ∗ *exp*(i ∗ *linepath s t x*)))
   (*at x within X*)
 unfolding *part_circlepath_def linepath_def scaleR_conv_of_real*
 by (*rule has_vector_derivative_real_field derivative_eq_intros* | *simp*)+

lemma *differentiable_part_circlepath*:
 part_circlepath c r a b differentiable at x within A
 using *has_vector_derivative_part_circlepath*[*of c r a b x A*] *differentiableI_vector*
by *blast*

lemma *vector_derivative_part_circlepath*:
  vector_derivative (*part_circlepath z r s t*) (*at x*) =
    i ∗ *r* ∗ (*of_real t* − *of_real s*) ∗ *exp*(i ∗ *linepath s t x*)
 using *has_vector_derivative_part_circlepath vector_derivative_at* **by** *blast*

lemma *vector_derivative_part_circlepath01*:
  ⟦*0* ≤ *x*; *x* ≤ *1*⟧
   ⟹ *vector_derivative* (*part_circlepath z r s t*) (*at x within* {*0..1*}) =
      i ∗ *r* ∗ (*of_real t* − *of_real s*) ∗ *exp*(i ∗ *linepath s t x*)
 using *has_vector_derivative_part_circlepath*
 by (*auto simp*: *vector_derivative_at_within_ivl*)

lemma *valid_path_part_circlepath* [*simp*]: *valid_path* (*part_circlepath z r s t*)
 unfolding *valid_path_def*
 by (*auto simp*: *C1_differentiable_on_eq vector_derivative_works vector_derivative_part_circlepath*
has_vector_derivative_part_circlepath
          intro!: *C1_differentiable_imp_piecewise continuous_intros*)

lemma *path_part_circlepath* [*simp*]: *path* (*part_circlepath z r s t*)
 by (*simp add*: *valid_path_imp_path*)

proposition *path_image_part_circlepath*:
 assumes *s* ≤ *t*

shows *path_image* (*part_circlepath z r s t*) = {*z* + *r* ∗ *exp*(i ∗ *of_real x*) | *x*. *s* ≤ *x* ∧ *x* ≤ *t*}
proof −
{ fix *z*::*real*
assume *0* ≤ *z z* ≤ *1*
with ‹*s* ≤ *t*› **have** ∃ *x*. (*exp* (i ∗ *linepath s t z*) = *exp* (i ∗ *of_real x*)) ∧ *s* ≤ *x* ∧ *x* ≤ *t*
apply (*rule_tac x*=(*1* − *z*) ∗ *s* + *z* ∗ *t* **in** *exI*)
apply (*simp add*: *linepath_def scaleR_conv_of_real algebra_simps*)
by (*metis* (*no_types*) *affine_ineq mult.commute mult_left_mono*)
}
moreover
{ fix *z*
assume *s* ≤ *z z* ≤ *t*
then have *z* + *of_real r* ∗ *exp* (i ∗ *of_real z*) ∈ (λ*x*. *z* + *of_real r* ∗ *exp* (i ∗ *linepath s t x*)) ‘ {*0*..*1*}
apply (*rule_tac x*=(*z* − *s*)/(*t* − *s*) **in** *image_eqI*)
apply (*simp add*: *linepath_def scaleR_conv_of_real divide_simps exp_eq*)
apply (*auto simp*: *field_split_simps*)
done
}
ultimately show *?thesis*
by (*fastforce simp add*: *path_image_def part_circlepath_def*)
qed

lemma *path_image_part_circlepath′*:
path_image (*part_circlepath z r s t*) = (λ*x*. *z* + *r* ∗ *cis x*) ‘ *closed_segment s t*
proof −
have *path_image* (*part_circlepath z r s t*) =
(λ*x*. *z* + *r* ∗ *exp*(i ∗ *of_real x*)) ‘ *linepath s t* ‘ {*0*..*1*}
by (*simp add*: *image_image path_image_def part_circlepath_def*)
also have *linepath s t* ‘ {*0*..*1*} = *closed_segment s t*
by (*rule linepath_image_01*)
finally show *?thesis* **by** (*simp add*: *cis_conv_exp*)
qed

lemma *path_image_part_circlepath_subset*:
⟦*s* ≤ *t*; *0* ≤ *r*⟧ ⟹ *path_image*(*part_circlepath z r s t*) ⊆ *sphere z r*
by (*auto simp*: *path_image_part_circlepath sphere_def dist_norm algebra_simps norm_mult*)

lemma *in_path_image_part_circlepath*:
assumes *w* ∈ *path_image*(*part_circlepath z r s t*) *s* ≤ *t* *0* ≤ *r*
shows *norm*(*w* − *z*) = *r*
proof −
have *w* ∈ {*c*. *dist z c* = *r*}
by (*metis* (*no_types*) *path_image_part_circlepath_subset sphere_def subset_eq assms*)
thus *?thesis*

by (*simp add*: *dist_norm norm_minus_commute*)
qed

lemma *path_image_part_circlepath_subset'*:
assumes $r \geq 0$
shows *path_image* (*part_circlepath z r s t*) $\subseteq$ *sphere z r*
proof (*cases* $s \leq t$)
case *True*
thus *?thesis* **using** *path_image_part_circlepath_subset*[*of s t r z*] *assms* **by** *simp*
next
case *False*
thus *?thesis* **using** *path_image_part_circlepath_subset*[*of t s r z*] *assms*
by (*subst reversepath_part_circlepath* [*symmetric*], *subst path_image_reversepath*) *simp_all*
qed

lemma *part_circlepath_cnj*: *cnj* (*part_circlepath c r a b x*) = *part_circlepath* (*cnj c*) *r* (−*a*) (−*b*) *x*
by (*simp add*: *part_circlepath_def exp_cnj linepath_def algebra_simps*)

lemma *contour_integral_bound_part_circlepath*:
assumes *f contour_integrable_on part_circlepath c r a b*
assumes $B \geq 0$ $r \geq 0$ $\bigwedge x.\ x \in$ *path_image* (*part_circlepath c r a b*) $\Longrightarrow$ *norm* (*f x*) $\leq B$
shows *norm* (*contour_integral* (*part_circlepath c r a b*) *f*) $\leq B * r * |b - a|$
proof −
let *?I* = *integral* {*0..1*} (λ*x*. *f* (*part_circlepath c r a b x*) ∗ i ∗ *of_real* (*r* ∗ (*b* − *a*)) ∗
exp (i ∗ *linepath a b x*))
have *norm ?I* ≤ *integral* {*0..1*} (λ*x*::*real*. *B* ∗ *1* ∗ (*r* ∗ |*b* − *a*|) ∗ *1*)
proof (*rule integral_norm_bound_integral*, *goal_cases*)
case *1*
with *assms*(*1*) **show** *?case*
by (*simp add*: *contour_integrable_on vector_derivative_part_circlepath mult_ac*)
next
case (*3 x*)
with *assms*(*2*−) **show** *?case* **unfolding** *norm_mult norm_of_real abs_mult*
by (*intro mult_mono*) (*auto simp*: *path_image_def*)
qed *auto*
also have *?I* = *contour_integral* (*part_circlepath c r a b*) *f*
by (*simp add*: *contour_integral_integral vector_derivative_part_circlepath mult_ac*)
finally show *?thesis* **by** *simp*
qed

lemma *has_contour_integral_part_circlepath_iff*:
assumes $a < b$
shows (*f has_contour_integral I*) (*part_circlepath c r a b*) ⟷
((λ*t*. *f* (*c* + *r* ∗ *cis t*) ∗ *r* ∗ i ∗ *cis t*) *has_integral I*) {*a..b*}

```
proof −
  have (f has_contour_integral I) (part_circlepath c r a b) ⟷
        ((λx. f (part_circlepath c r a b x) * vector_derivative (part_circlepath c r
a b)
          (at x within {0..1})) has_integral I) {0..1}
    unfolding has_contour_integral_def ..
  also have ... ⟷ ((λx. f (part_circlepath c r a b x) * r * (b − a) * i *
                        cis (linepath a b x)) has_integral I) {0..1}
    by (intro has_integral_cong, subst vector_derivative_part_circlepath01)
       (simp_all add: cis_conv_exp)
  also have ... ⟷ ((λx. f (c + r * exp (i * linepath (of_real a) (of_real b) x))
*
                     r * i * exp (i * linepath (of_real a) (of_real b) x) *
                     vector_derivative (linepath (of_real a) (of_real b))
                       (at x within {0..1})) has_integral I) {0..1}
    by (intro has_integral_cong, subst vector_derivative_linepath_within)
       (auto simp: part_circlepath_def cis_conv_exp of_real_linepath [symmetric])
  also have ... ⟷ ((λz. f (c + r * exp (i * z)) * r * i * exp (i * z)) has_contour_integral
I)
                    (linepath (of_real a) (of_real b))
    by (simp add: has_contour_integral_def)
  also have ... ⟷ ((λt. f (c + r * cis t) * r * i * cis t) has_integral I) {a..b}
using assms
    by (subst has_contour_integral_linepath_Reals_iff) (simp_all add: cis_conv_exp)
  finally show ?thesis .
qed

lemma contour_integrable_part_circlepath_iff:
  assumes a < b
  shows f contour_integrable_on (part_circlepath c r a b) ⟷
        (λt. f (c + r * cis t) * r * i * cis t) integrable_on {a..b}
  using assms by (auto simp: contour_integrable_on_def integrable_on_def
                             has_contour_integral_part_circlepath_iff)

lemma contour_integral_part_circlepath_eq:
  assumes a < b
  shows contour_integral (part_circlepath c r a b) f =
        integral {a..b} (λt. f (c + r * cis t) * r * i * cis t)
proof (cases f contour_integrable_on part_circlepath c r a b)
  case True
  hence (λt. f (c + r * cis t) * r * i * cis t) integrable_on {a..b}
    using assms by (simp add: contour_integrable_part_circlepath_iff)
  with True show ?thesis
    using has_contour_integral_part_circlepath_iff[OF assms]
          contour_integral_unique has_integral_integrable_integral by blast
next
  case False
  hence ¬(λt. f (c + r * cis t) * r * i * cis t) integrable_on {a..b}
    using assms by (simp add: contour_integrable_part_circlepath_iff)
```

with *False* **show** *?thesis*
by (*simp add*: *not_integrable_contour_integral not_integrable_integral*)
qed

lemma *contour_integral_part_circlepath_reverse*:
contour_integral (*part_circlepath c r a b*) *f* = −*contour_integral* (*part_circlepath c r b a*) *f*
by (*subst reversepath_part_circlepath* [*symmetric*], *subst contour_integral_reversepath*) *simp_all*

lemma *contour_integral_part_circlepath_reverse′*:
b < *a* ⟹ *contour_integral* (*part_circlepath c r a b*) *f* =
−*contour_integral* (*part_circlepath c r b a*) *f*
by (*rule contour_integral_part_circlepath_reverse*)

lemma *finite_bounded_log*: *finite* {*z*::*complex*. *norm z* ≤ *b* ∧ *exp z* = *w*}
proof (*cases w* = *0*)
case *True* **then show** *?thesis* **by** *auto*
next
case *False*
have ∗: *finite* {*x*. *cmod* ((*2* ∗ *real_of_int x* ∗ *pi*) ∗ i) ≤ *b* + *cmod* (*Ln w*)}
proof (*simp add*: *norm_mult finite_int_iff_bounded_le*)
show ∃ *k*. *abs* ‘ {*x*. *2* ∗ |*of_int x*| ∗ *pi* ≤ *b* + *cmod* (*Ln w*)} ⊆ {..*k*}
apply (*rule_tac x*=⌊(*b* + *cmod* (*Ln w*)) / (*2*∗*pi*)⌋ **in** *exI*)
apply (*auto simp*: *field_split_simps le_floor_iff*)
done
qed
have [*simp*]: ⋀*P f*. {*z*. *P z* ∧ (∃ *n*. *z* = *f n*)} = *f* ‘ {*n*. *P* (*f n*)}
by *blast*
have *finite* {*z*. *cmod z* ≤ *b* ∧ *exp z* = *exp* (*Ln w*)}
using *norm_add_leD* **by** (*fastforce intro*: *finite_subset* [*OF* _ ∗] *simp*: *exp_eq*)
then show *?thesis*
using *False* **by** *auto*
qed

lemma *finite_bounded_log2*:
fixes *a*::*complex*
assumes *a* ≠ *0*
shows *finite* {*z*. *norm z* ≤ *b* ∧ *exp*(*a*∗*z*) = *w*}
proof −
have ∗: *finite* ((λ*z*. *z* / *a*) ‘ {*z*. *cmod z* ≤ *b* ∗ *cmod a* ∧ *exp z* = *w*})
by (*rule finite_imageI* [*OF finite_bounded_log*])
show *?thesis*
by (*rule finite_subset* [*OF* _ ∗]) (*force simp*: *assms norm_mult*)
qed

lemma *has_contour_integral_bound_part_circlepath_strong*:
assumes *fi*: (*f has_contour_integral i*) (*part_circlepath z r s t*)
and *finite k* **and** *le*: *0* ≤ *B* *0* < *r s* ≤ *t*

```
    and B: ⋀x. x ∈ path_image(part_circlepath z r s t) − k ⟹ norm(f x) ≤ B
  shows cmod i ≤ B * r * (t − s)
proof −
 consider s = t | s < t using ‹s ≤ t› by linarith
 then show ?thesis
 proof cases
   case 1 with fi [unfolded has_contour_integral]
   have i = 0  by (simp add: vector_derivative_part_circlepath)
   with assms show ?thesis by simp
 next
   case 2
   have [simp]: |r| = r using ‹r > 0› by linarith
   have [simp]: cmod (complex_of_real t − complex_of_real s) = t−s
     by (metis 2 abs_of_pos diff_gt_0_iff_gt norm_of_real of_real_diff)
   have finite (part_circlepath z r s t −‘ {y} ∩ {0..1}) if y ∈ k for y
   proof −
     let ?w = (y − z)/of_real r / exp(i * of_real s)
     have fin: finite (of_real −‘ {z. cmod z ≤ 1 ∧ exp (i * complex_of_real (t −
s) * z) = ?w})
       using ‹s < t›
      by (intro finite_vimageI [OF finite_bounded_log2]) (auto simp: inj_of_real)
     show ?thesis
       unfolding part_circlepath_def linepath_def vimage_def
       using le
     by (intro finite_subset [OF _ fin]) (auto simp: algebra_simps scaleR_conv_of_real
exp_add exp_diff)
   qed
   then have fin01: finite ((part_circlepath z r s t) −‘ k ∩ {0..1})
     by (rule finite_finite_vimage_IntI [OF ‹finite k›])
   have **: ((λx. if (part_circlepath z r s t x) ∈ k then 0
                  else f(part_circlepath z r s t x) *
                     vector_derivative (part_circlepath z r s t) (at x)) has_integral
i) {0..1}
    by (rule has_integral_spike [OF negligible_finite [OF fin01]]) (use fi has_contour_integral
in auto)
     have *: ⋀x. ⟦0 ≤ x; x ≤ 1; part_circlepath z r s t x ∉ k⟧ ⟹ cmod (f
(part_circlepath z r s t x)) ≤ B
      by (auto intro!: B [unfolded path_image_def image_def, simplified])
   show ?thesis
     apply (rule has_integral_bound [where 'a=real, simplified, OF _ **, sim-
plified])
   using assms le * 2 ‹r > 0› by (auto simp add: norm_mult vector_derivative_part_circlepath)
 qed
qed

lemma has_contour_integral_bound_part_circlepath:
     ⟦(f has_contour_integral i) (part_circlepath z r s t);
       0 ≤ B; 0 < r; s ≤ t;
       ⋀x. x ∈ path_image(part_circlepath z r s t) ⟹ norm(f x) ≤ B⟧
```

$\Longrightarrow$ *norm i* $\leq$ *B*r*(*t* − *s*)
by (*auto intro*: *has_contour_integral_bound_part_circlepath_strong*)

lemma *contour_integrable_continuous_part_circlepath*:
continuous_on (*path_image* (*part_circlepath z r s t*)) *f*
$\Longrightarrow$ *f contour_integrable_on* (*part_circlepath z r s t*)
unfolding *contour_integrable_on has_contour_integral_def vector_derivative_part_circlepath path_image_def*
apply (*rule integrable_continuous_real*)
apply (*fast intro*: *path_part_circlepath* [*unfolded path_def*] *continuous_intros continuous_on_compose2* [**where** *g*=*f*, *OF* _ _ *order_refl*])
done

lemma *simple_path_part_circlepath*:
simple_path(*part_circlepath z r s t*) $\longleftrightarrow$ (*r* ≠ *0* ∧ *s* ≠ *t* ∧ |*s* − *t*| ≤ *2***pi*)
proof (*cases r* = *0* ∨ *s* = *t*)
case *True*
then show *?thesis*
unfolding *part_circlepath_def simple_path_def*
by (*rule disjE*) (*force intro*: *bexI* [**where** *x* = *1/4*] *bexI* [**where** *x* = *1/3*])+
next
case *False* **then have** *r* ≠ *0 s* ≠ *t* **by** *auto*
have ∗: ⋀*x y z s t*. i∗((*1* − *x*) ∗ *s* + *x* ∗ *t*) = i∗(((*1* − *y*) ∗ *s* + *y* ∗ *t*)) + *z* $\longleftrightarrow$ i∗(*x* − *y*) ∗ (*t* − *s*) = *z*
by (*simp add*: *algebra_simps*)
have *abs01*: ⋀*x y*::*real*. *0* ≤ *x* ∧ *x* ≤ *1* ∧ *0* ≤ *y* ∧ *y* ≤ *1*
$\Longrightarrow$ (*x* = *y* ∨ *x* = *0* ∧ *y* = *1* ∨ *x* = *1* ∧ *y* = *0* $\longleftrightarrow$ |*x* − *y*| ∈ {*0*,*1*})
by *auto*
have ∗∗: ⋀*x y*. (∃ *n*. (*complex_of_real x* − *of_real y*) ∗ (*of_real t* − *of_real s*) = *2* ∗ (*of_int n* ∗ *of_real pi*)) $\longleftrightarrow$
(∃ *n*. |*x* − *y*| ∗ (*t* − *s*) = *2* ∗ (*of_int n* ∗ *pi*))
by (*force simp*: *algebra_simps abs_if dest*: *arg_cong* [**where** *f*=*Re*] *arg_cong* [**where** *f*=*complex_of_real*]
intro: *exI* [**where** *x* = −*n* **for** *n*])
have *1*: |*s* − *t*| ≤ *2* ∗ *pi*
if ⋀*x*. *0* ≤ *x* ∧ *x* ≤ *1* $\Longrightarrow$ (∃ *n*. *x* ∗ (*t* − *s*) = *2* ∗ (*real_of_int n* ∗ *pi*)) $\longrightarrow$ *x* = *0* ∨ *x* = *1*
proof (*rule ccontr*)
assume ¬ |*s* − *t*| ≤ *2* ∗ *pi*
then have ∗: ⋀*n*. *t* − *s* ≠ *of_int n* ∗ |*s* − *t*|
using *False that* [*of 2*∗*pi* / |*t* − *s*|]
by (*simp add*: *abs_minus_commute divide_simps*)
show *False*
using ∗ [*of 1*] ∗ [*of* −*1*] **by** *auto*
qed
have *2*: |*s* − *t*| = |*2* ∗ (*real_of_int n* ∗ *pi*) / *x*| **if** *x* ≠ *0 x* ∗ (*t* − *s*) = *2* ∗ (*real_of_int n* ∗ *pi*) **for** *x n*
proof −

```
    have t−s = 2 * (real_of_int n * pi)/x
      using that by (simp add: field_simps)
    then show ?thesis by (metis abs_minus_commute)
  qed
  have abs_away: ⋀P. (∀ x∈{0..1}. ∀ y∈{0..1}. P |x − y|) ⟷ (∀ x::real. 0 ≤ x
∧ x ≤ 1 ⟶ P x)
    by force
  show ?thesis using False
    apply (simp add: simple_path_def)
     apply (simp add: part_circlepath_def linepath_def exp_eq  * ** abs01 del:
Set.insert_iff)
    apply (subst abs_away)
    apply (auto simp: 1)
    apply (rule ccontr)
    apply (auto simp: 2 field_split_simps abs_mult dest: of_int_leD)
    done
qed

lemma arc_part_circlepath:
  assumes r ≠ 0 s ≠ t |s − t| < 2*pi
    shows arc (part_circlepath z r s t)
proof −
  have *: x = y if eq: i * (linepath s t x) = i * (linepath s t y) + 2 * of_int n *
complex_of_real pi * i
    and x: x ∈ {0..1} and y: y ∈ {0..1} for x y n
  proof (rule ccontr)
    assume x ≠ y
    have (linepath s t x) = (linepath s t y) + 2 * of_int n * complex_of_real pi
     by (metis add_divide_eq_iff complex_i_not_zero mult.commute nonzero_mult_div_cancel_left
eq)
    then have s*y + t*x = s*x + (t*y + of_int n * (pi * 2))
      by (force simp: algebra_simps linepath_def dest: arg_cong [where f=Re])
    with ‹x ≠ y› have st: s−t = (of_int n * (pi * 2)) / (y−x))
      by (force simp: field_simps)
    have |real_of_int n| < |y − x|
      using assms ‹x ≠ y› by (simp add: st abs_mult field_simps)
    then show False
      using assms x y st by (auto dest: of_int_lessD)
  qed
  then have inj_on (part_circlepath z r s t) {0..1}
    using assms by (force simp add: part_circlepath_def inj_on_def exp_eq)
  then show ?thesis
    by (simp add: arc_def)
qed
```

1.14 Special case of one complete circle

```
definition circlepath :: [complex, real, real] ⇒ complex
  where circlepath z r ≡ part_circlepath z r 0 (2*pi)
```

lemma *circlepath*: *circlepath z r* = (λ*x*. *z* + *r* ∗ *exp*(*2* ∗ *of_real pi* ∗ i ∗ *of_real x*))
by (*simp add*: *circlepath_def part_circlepath_def linepath_def algebra_simps*)

lemma *pathstart_circlepath* [*simp*]: *pathstart* (*circlepath z r*) = *z* + *r*
by (*simp add*: *circlepath_def*)

lemma *pathfinish_circlepath* [*simp*]: *pathfinish* (*circlepath z r*) = *z* + *r*
by (*simp add*: *circlepath_def*) (*metis exp_two_pi_i mult.commute*)

lemma *circlepath_minus*: *circlepath z* (−*r*) *x* = *circlepath z r* (*x* + *1/2*)
proof −
have *z* + *of_real r* ∗ *exp* (*2* ∗ *pi* ∗ i ∗ (*x* + *1/2*)) =
z + *of_real r* ∗ *exp* (*2* ∗ *pi* ∗ i ∗ *x* + *pi* ∗ i)
by (*simp add*: *divide_simps*) (*simp add*: *algebra_simps*)
also have ... = *z* − *r* ∗ *exp* (*2* ∗ *pi* ∗ i ∗ *x*)
by (*simp add*: *exp_add*)
finally show *?thesis*
by (*simp add*: *circlepath path_image_def sphere_def dist_norm*)
qed

lemma *circlepath_add1*: *circlepath z r* (*x*+*1*) = *circlepath z r x*
using *circlepath_minus* [*of z r x*+*1/2*] *circlepath_minus* [*of z* −*r x*]
by (*simp add*: *add.commute*)

lemma *circlepath_add_half*: *circlepath z r* (*x* + *1/2*) = *circlepath z r* (*x* − *1/2*)
using *circlepath_add1* [*of z r x*−*1/2*]
by (*simp add*: *add.commute*)

lemma *path_image_circlepath_minus_subset*:
path_image (*circlepath z* (−*r*)) ⊆ *path_image* (*circlepath z r*)
proof −
have ∃*x*∈{*0*..*1*}. *circlepath z r* (*y* + *1/2*) = *circlepath z r x*
if *0* ≤ *y y* ≤ *1* **for** *y*
proof (*cases y* ≤ *1/2*)
case *False*
with *that* **show** *?thesis*
by (*force simp*: *circlepath_add_half*)
qed (*use that* **in** *force*)
then show *?thesis*
by (*auto simp add*: *path_image_def image_def circlepath_minus*)
qed

lemma *path_image_circlepath_minus*: *path_image* (*circlepath z* (−*r*)) = *path_image* (*circlepath z r*)
using *path_image_circlepath_minus_subset* **by** *fastforce*

lemma *has_vector_derivative_circlepath* [*derivative_intros*]:

```
 ((circlepath z r) has_vector_derivative (2 * pi * ii * r * exp (2 * of_real pi * ii *
x)))
   (at x within X)
  unfolding circlepath_def scaleR_conv_of_real
  by (rule derivative_eq_intros) (simp add: algebra_simps)

lemma vector_derivative_circlepath:
  vector_derivative (circlepath z r) (at x) =
    2 * pi * ii * r * exp(2 * of_real pi * ii * x)
  using has_vector_derivative_circlepath vector_derivative_at by blast

lemma vector_derivative_circlepath01:
    ⟦0 ≤ x; x ≤ 1⟧
     ⟹ vector_derivative (circlepath z r) (at x within {0..1}) =
          2 * pi * ii * r * exp(2 * of_real pi * ii * x)
  using has_vector_derivative_circlepath
  by (auto simp: vector_derivative_at_within_ivl)

lemma valid_path_circlepath [simp]: valid_path (circlepath z r)
  by (simp add: circlepath_def)

lemma path_circlepath [simp]: path (circlepath z r)
  by (simp add: valid_path_imp_path)

lemma path_image_circlepath_nonneg:
  assumes 0 ≤ r shows path_image (circlepath z r) = sphere z r
proof −
  have ∗: x ∈ (λu. z + (cmod (x − z)) * exp (ii * (of_real u * (of_real pi * 2))))
` {0..1} for x
  proof (cases x = z)
    case True then show ?thesis by force
  next
    case False
    define w where w = x − z
    then have w ≠ 0 by (simp add: False)
   have ∗∗: ⋀t. ⟦Re w = cos t * cmod w; Im w = sin t * cmod w⟧ ⟹ w = of_real
(cmod w) * exp (ii * t)
      using cis_conv_exp complex_eq_iff by auto
    obtain t where 0 ≤ t t < 2∗pi Re(w/norm w) = cos t Im(w/norm w) = sin t
      apply (rule sincos_total_2pi [of Re(w/(norm w)) Im(w/(norm w))])
      by (auto simp add: divide_simps ‹w ≠ 0› cmod_power2 [symmetric])
    then
    show ?thesis
      using False ∗∗ w_def ‹w ≠ 0›
      by (rule_tac x=t / (2∗pi) in image_eqI) (auto simp add: field_simps)
  qed
  show ?thesis
    unfolding circlepath path_image_def sphere_def dist_norm
    by (force simp: assms algebra_simps norm_mult norm_minus_commute intro:
```

```
*)
qed

lemma path_image_circlepath [simp]:
    path_image (circlepath z r) = sphere z |r|
  using path_image_circlepath_minus
  by (force simp: path_image_circlepath_nonneg abs_if)

lemma has_contour_integral_bound_circlepath_strong:
      [[(f has_contour_integral i) (circlepath z r);
        finite k; 0 ≤ B; 0 < r;
        ⋀x. [[norm(x − z) = r; x ∉ k]] ⟹ norm(f x) ≤ B]]
        ⟹ norm i ≤ B*(2*pi*r)
  unfolding circlepath_def
  by (auto simp: algebra_simps in_path_image_part_circlepath dest!: has_contour_integral_bound_part_circ

lemma has_contour_integral_bound_circlepath:
      [[(f has_contour_integral i) (circlepath z r);
        0 ≤ B; 0 < r; ⋀x. norm(x − z) = r ⟹ norm(f x) ≤ B]]
        ⟹ norm i ≤ B*(2*pi*r)
  by (auto intro: has_contour_integral_bound_circlepath_strong)

lemma contour_integrable_continuous_circlepath:
    continuous_on (path_image (circlepath z r)) f
     ⟹ f contour_integrable_on (circlepath z r)
  by (simp add: circlepath_def contour_integrable_continuous_part_circlepath)

lemma simple_path_circlepath: simple_path(circlepath z r) ⟷ (r ≠ 0)
  by (simp add: circlepath_def simple_path_part_circlepath)

lemma notin_path_image_circlepath [simp]: cmod (w − z) < r ⟹ w ∉ path_image
(circlepath z r)
  by (simp add: sphere_def dist_norm norm_minus_commute)

lemma contour_integral_circlepath:
  assumes r > 0
  shows contour_integral (circlepath z r) (λw. 1 / (w − z)) = 2 * complex_of_real
pi * i
proof (rule contour_integral_unique)
   show ((λw. 1 / (w − z)) has_contour_integral 2 * complex_of_real pi * i)
(circlepath z r)
    unfolding has_contour_integral_def using assms has_integral_const_real [of
_ 0 1]
    apply (subst has_integral_cong)
     apply (simp add: vector_derivative_circlepath01)
    apply (force simp: circlepath)
    done
qed
```

1.15 Uniform convergence of path integral

Uniform convergence when the derivative of the path is bounded, and in particular for the special case of a circle.

proposition *contour_integral_uniform_limit*:
assumes *ev_fint*: *eventually* (λn::'a. (*f n*) *contour_integrable_on* γ) *F*
and *ul_f*: *uniform_limit* (*path_image* γ) *f l F*
and *noleB*: ⋀*t*. *t* ∈ {*0..1*} ⟹ *norm* (*vector_derivative* γ (*at t*)) ≤ *B*
and γ: *valid_path* γ
and [*simp*]: ¬ *trivial_limit F*
shows *l contour_integrable_on* γ ((λn. *contour_integral* γ (*f n*)) ⟶ *contour_integral* γ *l*) *F*
proof −
have *0* ≤ *B* **by** (*meson noleB* [*of 0*] *atLeastAtMost_iff norm_ge_zero order_refl order_trans zero_le_one*)
{ fix *e*::*real*
assume *0* < *e*
then have *0* < *e* / (|*B*| + *1*) **by** *simp*
then have ∀ *F n in F*. ∀ *x*∈*path_image* γ. *cmod* (*f n x* − *l x*) < *e* / (|*B*| + *1*)
using *ul_f* [*unfolded uniform_limit_iff dist_norm*] **by** *auto*
with *ev_fint*
obtain *a* **where** *fga*: ⋀*x*. *x* ∈ {*0..1*} ⟹ *cmod* (*f a* (γ *x*) − *l* (γ *x*)) < *e* / (|*B*| + *1*)
and *inta*: (λt. *f a* (γ *t*) ∗ *vector_derivative* γ (*at t*)) *integrable_on* {*0..1*}
using *eventually_happens* [*OF eventually_conj*]
by (*fastforce simp*: *contour_integrable_on path_image_def*)
have *Ble*: *B* ∗ *e* / (|*B*| + *1*) ≤ *e*
using ‹*0* ≤ *B*› ‹*0* < *e*› **by** (*simp add*: *field_split_simps*)
have ∃ *h*. (∀ *x*∈{*0..1*}. *cmod* (*l* (γ *x*) ∗ *vector_derivative* γ (*at x*) − *h x*) ≤ *e*) ∧ *h integrable_on* {*0..1*}
proof (*intro exI conjI ballI*)
show *cmod* (*l* (γ *x*) ∗ *vector_derivative* γ (*at x*) − *f a* (γ *x*) ∗ *vector_derivative* γ (*at x*)) ≤ *e*
if *x* ∈ {*0..1*} **for** *x*
apply (*rule order_trans* [*OF _ Ble*])
using *noleB* [*OF that*] *fga* [*OF that*] ‹*0* ≤ *B*› ‹*0* < *e*›
apply (*fastforce simp*: *mult_ac dest*: *mult_mono* [*OF less_imp_le*] *simp add*: *norm_mult left_diff_distrib* [*symmetric*] *norm_minus_commute divide_simps*)
done
qed (*rule inta*)
}
then show *lintg*: *l contour_integrable_on* γ
unfolding *contour_integrable_on* **by** (*metis* (*mono_tags*, *lifting*)*integrable_uniform_limit_real*)
{ fix *e*::*real*
define *B'* **where** *B'* = *B* + *1*
have *B'*: *B'* > *0* *B'* > *B* **using** ‹*0* ≤ *B*› **by** (*auto simp*: *B'_def*)
assume *0* < *e*
then have *ev_no'*: ∀ *F n in F*. ∀ *x*∈*path_image* γ. *2* ∗ *cmod* (*f n x* − *l x*) < *e*

```
/ B'
    using ul_f [unfolded uniform_limit_iff dist_norm, rule_format, of e / B'/2]
B'
      by (simp add: field_simps)
  have ie: integral {0..1::real} (λx. e/2) < e using ‹0 < e› by simp
  have ∗: cmod (f x (γ t) ∗ vector_derivative γ (at t) − l (γ t) ∗ vector_derivative
γ (at t)) ≤ e/2
          if t: t∈{0..1} and leB': 2 ∗ cmod (f x (γ t) − l (γ t)) < e / B' for x t
  proof −
    have 2 ∗ cmod (f x (γ t) − l (γ t)) ∗ cmod (vector_derivative γ (at t)) ≤ e
∗ (B/ B')
      using mult_mono [OF less_imp_le [OF leB'] noleB] B' ‹0 < e› t by auto
    also have ... < e
      by (simp add: B' ‹0 < e› mult_imp_div_pos_less)
    finally have 2 ∗ cmod (f x (γ t) − l (γ t)) ∗ cmod (vector_derivative γ (at
t)) < e .
    then show ?thesis
      by (simp add: left_diff_distrib [symmetric] norm_mult)
  qed
  have le_e: ⋀x. ⟦∀ xa∈{0..1}. 2 ∗ cmod (f x (γ xa) − l (γ xa)) < e / B'; f x
contour_integrable_on γ⟧
        ⟹ cmod (integral {0..1}
            (λu. f x (γ u) ∗ vector_derivative γ (at u) − l (γ u) ∗ vector_derivative
γ (at u))) < e
    apply (rule le_less_trans [OF integral_norm_bound_integral ie])
     apply (simp add: lintg integrable_diff contour_integrable_on [symmetric])
    apply (blast intro: ∗)+
    done
  have ∀ F x in F. dist (contour_integral γ (f x)) (contour_integral γ l) < e
    apply (rule eventually_mono [OF eventually_conj [OF ev_no' ev_fint]])
     apply (simp add: dist_norm contour_integrable_on path_image_def con-
tour_integral_integral)
      apply (simp add: lintg integral_diff [symmetric] contour_integrable_on
[symmetric] le_e)
    done
 }
 then show ((λn. contour_integral γ (f n)) ⟶ contour_integral γ l) F
  by (rule tendstoI)
qed

corollary contour_integral_uniform_limit_circlepath:
 assumes ∀ F n::'a in F. (f n) contour_integrable_on (circlepath z r)
    and uniform_limit (sphere z r) f l F
    and ¬ trivial_limit F 0 < r
  shows l contour_integrable_on (circlepath z r)
          ((λn. contour_integral (circlepath z r) (f n)) ⟶ contour_integral
(circlepath z r) l) F
 using assms by (auto simp: vector_derivative_circlepath norm_mult intro!: con-
tour_integral_uniform_limit)
```

end

2 Complex Path Integrals and Cauchy's Integral Theorem

By John Harrison et al. Ported from HOL Light by L C Paulson (2015)

theory *Cauchy_Integral_Theorem*
imports
HOL−Analysis.Analysis
Contour_Integration
begin

lemma *leibniz_rule_holomorphic*:
fixes f::*complex* ⇒ 'b::*euclidean_space* ⇒ *complex*
assumes ⋀x t. $x \in U \Longrightarrow t \in$ *cbox a b* ⟹ ((λx. $f\ x\ t$) *has_field_derivative* $fx\ x\ t$) (*at x within U*)
assumes ⋀x. $x \in U \Longrightarrow$ ($f\ x$) *integrable_on cbox a b*
assumes *continuous_on* ($U \times$ (*cbox a b*)) (λ(x, t). $fx\ x\ t$)
assumes *convex U*
shows (λx. *integral* (*cbox a b*) ($f\ x$)) *holomorphic_on U*
using *leibniz_rule_field_differentiable*[*OF assms*(1−3) _ *assms*(4)]
by (*auto simp*: *holomorphic_on_def*)

lemma *Ln_measurable* [*measurable*]: *Ln* ∈ *measurable borel borel*
proof −
have ∗: *Ln* (−*of_real x*) = *of_real* (*ln x*) + i ∗ *pi* **if** $x > 0$ **for** x
using *that* **by** (*subst Ln_minus*) (*auto simp*: *Ln_of_real*)
have ∗∗: *Ln* (*of_real x*) = *of_real* (*ln* (−x)) + i ∗ *pi* **if** $x < 0$ **for** x
using ∗[*of* −x] *that* **by** *simp*
have *cont*: (λx. *indicat_real* (− $\mathbb{R}_{\leq 0}$) x ∗$_R$ *Ln x*) ∈ *borel_measurable borel*
by (*intro borel_measurable_continuous_on_indicator continuous_intros*) *auto*
have (λx. *if* $x \in \mathbb{R}_{\leq 0}$ *then ln* (−*Re x*) + i ∗ *pi else indicator* (−$\mathbb{R}_{\leq 0}$) x ∗$_R$ *Ln x*) ∈ *borel* →$_M$ *borel*
(**is** *?f* ∈ _) **by** (*rule measurable_If_set*[*OF* _ *cont*]) *auto*
hence (λx. *if* $x = 0$ *then Ln 0 else ?f x*) ∈ *borel* →$_M$ *borel* **by** *measurable*
also have (λx. *if* $x = 0$ *then Ln 0 else ?f x*) = *Ln*
by (*auto simp*: *fun_eq_iff* ∗∗ *nonpos_Reals_def*)
finally show *?thesis* **.**
qed

lemma *powr_complex_measurable* [*measurable*]:
assumes [*measurable*]: $f \in$ *measurable M borel* $g \in$ *measurable M borel*
shows (λx. $f\ x$ *powr* $g\ x$ :: *complex*) ∈ *measurable M borel*
using *assms* **by** (*simp add*: *powr_def*)

The special case of midpoints used in the main quadrisection

lemma *has_contour_integral_midpoint*:

assumes (*f has_contour_integral i*) (*linepath a* (*midpoint a b*))
(*f has_contour_integral j*) (*linepath* (*midpoint a b*) *b*)
shows (*f has_contour_integral* (*i* + *j*)) (*linepath a b*)
proof (*rule has_contour_integral_split*)
show *midpoint a b* − *a* = (*1*/*2*) ∗$_R$ (*b* − *a*)
using *assms* **by** (*auto simp*: *midpoint_def scaleR_conv_of_real*)
qed (*use assms* **in** *auto*)

lemma *contour_integral_midpoint*:
assumes *continuous_on* (*closed_segment a b*) *f*
shows *contour_integral* (*linepath a b*) *f* =
contour_integral (*linepath a* (*midpoint a b*)) *f* + *contour_integral* (*linepath* (*midpoint a b*) *b*) *f*
proof (*rule contour_integral_split*)
show *midpoint a b* − *a* = (*1*/*2*) ∗$_R$ (*b* − *a*)
using *assms* **by** (*auto simp*: *midpoint_def scaleR_conv_of_real*)
qed (*use assms* **in** *auto*)

A couple of special case lemmas that are useful below

lemma *triangle_linear_has_chain_integral*:
((λ*x*. *m*∗*x* + *d*) *has_contour_integral 0*) (*linepath a b* +++ *linepath b c* +++ *linepath c a*)
proof (*rule Cauchy_theorem_primitive*)
show ⋀*x*. *x* ∈ *UNIV* ⟹ ((λ*x*. *m* / *2* ∗ x^2 + *d* ∗ *x*) *has_field_derivative m* ∗ *x* + *d*) (*at x*)
by (*auto intro*!: *derivative_eq_intros*)
qed *auto*

lemma *has_chain_integral_chain_integral3*:
assumes (*f has_contour_integral i*) (*linepath a b* +++ *linepath b c* +++ *linepath c d*)
(**is** (*f has_contour_integral i*) *?g*)
shows *contour_integral* (*linepath a b*) *f* + *contour_integral* (*linepath b c*) *f* + *contour_integral* (*linepath c d*) *f* = *i*
(**is** *?lhs* = _)
proof −
have *f contour_integrable_on ?g*
using *assms contour_integrable_on_def* **by** *blast*
then have *?lhs* = *contour_integral ?g f*
by (*simp add*: *valid_path_join has_contour_integral_integrable*)
then show *?thesis*
using *assms contour_integral_unique* **by** *blast*
qed

lemma *has_chain_integral_chain_integral4*:
assumes (*f has_contour_integral i*) (*linepath a b* +++ *linepath b c* +++ *linepath c d* +++ *linepath d e*)
(**is** (*f has_contour_integral i*) *?g*)
shows *contour_integral* (*linepath a b*) *f* + *contour_integral* (*linepath b c*) *f* +

contour_integral (*linepath c d*) *f* + *contour_integral* (*linepath d e*) *f* = *i*
(**is** *?lhs* = _)
proof −
have *f contour_integrable_on ?g*
using *assms contour_integrable_on_def* **by** *blast*
then have *?lhs* = *contour_integral ?g f*
by (*simp add*: *valid_path_join has_contour_integral_integrable*)
then show *?thesis*
using *assms contour_integral_unique* **by** *blast*
qed

2.1 The key quadrisection step

lemma *norm_sum_half*:
assumes *norm*(*a* + *b*) ≥ *e*
shows *norm a* ≥ *e*/2 ∨ *norm b* ≥ *e*/2
proof −
have *e* ≤ *norm* (− *a* − *b*)
by (*simp add*: *add.commute assms norm_minus_commute*)
thus *?thesis*
using *norm_triangle_ineq4 order_trans* **by** *fastforce*
qed

lemma *norm_sum_lemma*:
assumes *e* ≤ *norm* (*a* + *b* + *c* + *d*)
shows *e* / 4 ≤ *norm a* ∨ *e* / 4 ≤ *norm b* ∨ *e* / 4 ≤ *norm c* ∨ *e* / 4 ≤ *norm d*
proof −
have *e* ≤ *norm* ((*a* + *b*) + (*c* + *d*)) **using** *assms*
by (*simp add*: *algebra_simps*)
then show *?thesis*
by (*auto dest*!: *norm_sum_half*)
qed

lemma *Cauchy_theorem_quadrisection*:
assumes *f*: *continuous_on* (*convex hull* {*a*,*b*,*c*}) *f*
and *dist*: *dist a b* ≤ *K dist b c* ≤ *K dist c a* ≤ *K*
and *e*: *e* ∗ *K*^2 ≤
norm (*contour_integral*(*linepath a b*) *f* + *contour_integral*(*linepath b c*) *f* + *contour_integral*(*linepath c a*) *f*)
shows ∃ *a*′ *b*′ *c*′.
a′ ∈ *convex hull* {*a*,*b*,*c*} ∧ *b*′ ∈ *convex hull* {*a*,*b*,*c*} ∧ *c*′ ∈ *convex hull* {*a*,*b*,*c*} ∧
dist a′ *b*′ ≤ *K*/2 ∧ *dist b*′ *c*′ ≤ *K*/2 ∧ *dist c*′ *a*′ ≤ *K*/2 ∧
e ∗ (*K*/2)^2 ≤ *norm*(*contour_integral*(*linepath a*′ *b*′) *f* + *contour_integral*(*linepath b*′ *c*′) *f* + *contour_integral*(*linepath c*′ *a*′) *f*)
(**is** ∃ *x y z*. *?Φ x y z*)
proof −
note *divide_le_eq_numeral1* [*simp del*]
define *a*′ **where** *a*′ = *midpoint b c*

define b' **where** $b' = midpoint\ c\ a$
define c' **where** $c' = midpoint\ a\ b$
have *fabc*: *continuous_on* (*closed_segment a b*) *f continuous_on* (*closed_segment b c*) *f continuous_on* (*closed_segment c a*) *f*
using *f continuous_on_subset segments_subset_convex_hull* **by** *metis+*
have *fcont'*: *continuous_on* (*closed_segment c' b'*) *f*
continuous_on (*closed_segment a' c'*) *f*
continuous_on (*closed_segment b' a'*) *f*
unfolding *a'_def b'_def c'_def*
by (*rule continuous_on_subset* [*OF f*],
metis midpoints_in_convex_hull convex_hull_subset hull_subset insert_subset segment_convex_hull)+
define *pathint* **where** *pathint x y* ≡ *contour_integral*(*linepath x y*) *f* **for** *x y*
have ∗: *pathint a b* + *pathint b c* + *pathint c a* =
(*pathint a c'* + *pathint c' b'* + *pathint b' a*) +
(*pathint a' c'* + *pathint c' b* + *pathint b a'*) +
(*pathint a' c* + *pathint c b'* + *pathint b' a'*) +
(*pathint a' b'* + *pathint b' c'* + *pathint c' a'*)
unfolding *pathint_def*
by (*simp add*: *fcont' contour_integral_reverse_linepath*) (*simp add*: *a'_def b'_def c'_def contour_integral_midpoint fabc*)
have [*simp*]: ⋀*x y*. *cmod* (*x* ∗ *2* − *y* ∗ *2*) = *cmod* (*x* − *y*) ∗ *2*
by (*metis left_diff_distrib mult.commute norm_mult_numeral1*)
have [*simp*]: ⋀*x y*. *cmod* (*x* − *y*) = *cmod* (*y* − *x*)
by (*simp add*: *norm_minus_commute*)
consider *e* ∗ K^2 / *4* ≤ *cmod* (*pathint a c'* + *pathint c' b'* + *pathint b' a*) |
e ∗ K^2 / *4* ≤ *cmod* (*pathint a' c'* + *pathint c' b* + *pathint b a'*) |
e ∗ K^2 / *4* ≤ *cmod* (*pathint a' c* + *pathint c b'* + *pathint b' a'*) |
e ∗ K^2 / *4* ≤ *cmod* (*pathint a' b'* + *pathint b' c'* + *pathint c' a'*)
using *assms* **by** (*metis* ∗ *norm_sum_lemma pathint_def*)
then show *?thesis*
proof *cases*
case *1* **then have** *?*Φ *a c' b'*
using *assms* **unfolding** *pathint_def* [*symmetric*]
apply (*clarsimp simp*: *c'_def b'_def midpoints_in_convex_hull hull_subset* [*THEN subsetD*])
apply (*auto simp*: *midpoint_def dist_norm scaleR_conv_of_real field_split_simps*)
done
then show *?thesis* **by** *blast*
next
case *2* **then have** *?*Φ *a' c' b*
using *assms* **unfolding** *pathint_def* [*symmetric*]
apply (*clarsimp simp*: *a'_def c'_def midpoints_in_convex_hull hull_subset* [*THEN subsetD*])
apply (*auto simp*: *midpoint_def dist_norm scaleR_conv_of_real field_split_simps*)
done
then show *?thesis* **by** *blast*
next
case *3* **then have** *?*Φ *a' c b'*

using *assms* **unfolding** *pathint_def* [*symmetric*]
apply (*clarsimp simp*: *a'_def b'_def midpoints_in_convex_hull hull_subset* [*THEN subsetD*])
apply (*auto simp*: *midpoint_def dist_norm scaleR_conv_of_real field_split_simps*)
done
then show *?thesis* **by** *blast*
next
case *4* **then have** *?Φ a' b' c'*
using *assms* **unfolding** *pathint_def* [*symmetric*]
apply (*clarsimp simp*: *a'_def c'_def b'_def midpoints_in_convex_hull hull_subset* [*THEN subsetD*])
apply (*auto simp*: *midpoint_def dist_norm scaleR_conv_of_real field_split_simps*)
done
then show *?thesis* **by** *blast*
qed
qed

2.2 Cauchy's theorem for triangles

lemma *triangle_points_closer*:
fixes *a*::*complex*
shows ⟦*x* ∈ *convex hull* {*a*,*b*,*c*}; *y* ∈ *convex hull* {*a*,*b*,*c*}⟧
⟹ *norm*(*x* − *y*) ≤ *norm*(*a* − *b*) ∨
norm(*x* − *y*) ≤ *norm*(*b* − *c*) ∨
norm(*x* − *y*) ≤ *norm*(*c* − *a*)
using *simplex_extremal_le* [*of* {*a*,*b*,*c*}]
by (*auto simp*: *norm_minus_commute*)

lemma *holomorphic_point_small_triangle*:
assumes *x*: *x* ∈ *S*
and *f*: *continuous_on S f*
and *cd*: *f field_differentiable* (*at x within S*)
and *e*: *0* < *e*
shows ∃*k*>*0*. ∀ *a b c*. *dist a b* ≤ *k* ∧ *dist b c* ≤ *k* ∧ *dist c a* ≤ *k* ∧
x ∈ *convex hull* {*a*,*b*,*c*} ∧ *convex hull* {*a*,*b*,*c*} ⊆ *S*
⟶ *norm*(*contour_integral*(*linepath a b*) *f* + *contour_integral*(*linepath b c*) *f* +
contour_integral(*linepath c a*) *f*)
≤ *e*∗(*dist a b* + *dist b c* + *dist c a*)^2
(**is** ∃*k*>*0*. ∀ *a b c*. _ ⟶ *?normle a b c*)
proof −
have *le_of_3*: ⋀*a x y z*. ⟦*0* ≤ *x*∗*y*; *0* ≤ *x*∗*z*; *0* ≤ *y*∗*z*; *a* ≤ (*e*∗(*x* + *y* + *z*))∗*x* + (*e*∗(*x* + *y* + *z*))∗*y* + (*e*∗(*x* + *y* + *z*))∗*z*⟧
⟹ *a* ≤ *e*∗(*x* + *y* + *z*)^2
by (*simp add*: *algebra_simps power2_eq_square*)
have *disj_le*: ⟦*x* ≤ *a* ∨ *x* ≤ *b* ∨ *x* ≤ *c*; *0* ≤ *a*; *0* ≤ *b*; *0* ≤ *c*⟧ ⟹ *x* ≤ *a* + *b* + *c*
for *x*::*real* **and** *a b c*
by *linarith*

have *fabc*: *f contour_integrable_on linepath a b f contour_integrable_on linepath b c f contour_integrable_on linepath c a*
if *convex hull* $\{a, b, c\} \subseteq S$ **for** *a b c*
using *segments_subset_convex_hull that*
by (*metis continuous_on_subset f contour_integrable_continuous_linepath*)+
note *path_bound* = *has_contour_integral_bound_linepath* [*simplified norm_minus_commute*, *OF has_contour_integral_integral*]
{ fix f' *a b c d*
assume *d*: $0 < d$
and f': $\bigwedge y.\ [\![cmod\ (y - x) \le d;\ y \in S]\!] \Longrightarrow cmod\ (f\ y - f\ x - f' * (y - x)) \le e * cmod\ (y - x)$
and *le*: $cmod\ (a - b) \le d\ cmod\ (b - c) \le d\ cmod\ (c - a) \le d$
and *xc*: $x \in$ *convex hull* $\{a, b, c\}$
and *S*: *convex hull* $\{a, b, c\} \subseteq S$
have *pa*: *contour_integral* (*linepath a b*) *f* + *contour_integral* (*linepath b c*) *f* + *contour_integral* (*linepath c a*) *f* =
contour_integral (*linepath a b*) ($\lambda y.\ f\ y - f\ x - f' * (y - x)$) +
contour_integral (*linepath b c*) ($\lambda y.\ f\ y - f\ x - f' * (y - x)$) +
contour_integral (*linepath c a*) ($\lambda y.\ f\ y - f\ x - f' * (y - x)$)
apply (*simp add*: *contour_integral_diff contour_integral_lmul contour_integrable_lmul contour_integrable_diff fabc* [*OF S*])
apply (*simp add*: *field_simps*)
done
{ fix *y*
assume *yc*: $y \in$ *convex hull* $\{a,b,c\}$
have $cmod\ (f\ y - f\ x - f' * (y - x)) \le e * norm(y - x)$
proof (*rule f'*)
show $cmod\ (y - x) \le d$
by (*metis triangle_points_closer* [*OF xc yc*] *le norm_minus_commute order_trans*)
qed (*use S yc* **in** *blast*)
also have $\ldots \le e * (cmod\ (a - b) + cmod\ (b - c) + cmod\ (c - a))$
by (*simp add*: *yc e xc disj_le* [*OF triangle_points_closer*])
finally have $cmod\ (f\ y - f\ x - f' * (y - x)) \le e * (cmod\ (a - b) + cmod\ (b - c) + cmod\ (c - a))$ **.**
} note *cm_le* = *this*
have *?normle a b c*
unfolding *dist_norm pa*
using f' *xc S e*
apply (*intro le_of_3 norm_triangle_le add_mono path_bound*)
apply (*simp_all add*: *contour_integral_diff contour_integral_lmul contour_integrable_lmul contour_integrable_diff fabc*)
apply (*blast intro*: *cm_le elim*: *dest*: *segments_subset_convex_hull* [*THEN subsetD*])+
done
} note $*$ = *this*
show *?thesis*
using *cd e*
apply (*simp add*: *field_differentiable_def has_field_derivative_def has_derivative_within_alt*

approachable_lt_le2 Ball_def)
apply (*clarify dest*!: *spec mp*)
using ∗ **unfolding** *dist_norm*
apply *blast*
done
qed

Hence the most basic theorem for a triangle.

locale *Chain* =
fixes *x0 At Follows*
assumes *At0*: *At x0 0*
and *AtSuc*: ⋀*x n*. *At x n* ⟹ ∃ *x'*. *At x' (Suc n)* ∧ *Follows x' x*
begin
primrec *f* **where**
f 0 = *x0*
| *f (Suc n)* = (*SOME x*. *At x (Suc n)* ∧ *Follows x (f n)*)

lemma *At*: *At (f n) n*
proof (*induct n*)
case *0* **show** *?case*
by (*simp add*: *At0*)
next
case (*Suc n*) **show** *?case*
by (*metis* (*no_types*, *lifting*) *AtSuc* [*OF Suc*] *f.simps*(*2*) *someI_ex*)
qed

lemma *Follows*: *Follows* (*f*(*Suc n*)) (*f n*)
by (*metis* (*no_types*, *lifting*) *AtSuc* [*OF At* [*of n*]] *f.simps*(*2*) *someI_ex*)

declare *f.simps*(*2*) [*simp del*]
end

lemma *Chain3*:
assumes *At0*: *At x0 y0 z0 0*
and *AtSuc*: ⋀*x y z n*. *At x y z n* ⟹ ∃ *x' y' z'*. *At x' y' z' (Suc n)* ∧ *Follows x' y' z' x y z*
obtains *f g h* **where**
f 0 = *x0 g 0* = *y0 h 0* = *z0*
⋀*n*. *At (f n) (g n) (h n) n*
⋀*n*. *Follows* (*f*(*Suc n*)) (*g*(*Suc n*)) (*h*(*Suc n*)) (*f n*) (*g n*) (*h n*)
proof −
interpret *three*: *Chain* (*x0*,*y0*,*z0*) λ(*x*,*y*,*z*). *At x y z* λ(*x'*,*y'*,*z'*). λ(*x*,*y*,*z*). *Follows x' y' z' x y z*
proof qed (*use At0 AtSuc* **in** *auto*)
show *?thesis*
proof
show ⋀*n*. *Follows* (*fst* (*three.f* (*Suc n*))) (*fst* (*snd* (*three.f* (*Suc n*))))
(*snd* (*snd* (*three.f* (*Suc n*)))) (*fst* (*three.f n*))
(*fst* (*snd* (*three.f n*))) (*snd* (*snd* (*three.f n*)))

```
      ⋀n. At (fst (three.f n)) (fst (snd (three.f n))) (snd (snd (three.f n))) n
    using three.At three.Follows
    by (simp_all add: split_beta')
  qed auto
qed


proposition Cauchy_theorem_triangle:
  assumes f holomorphic_on (convex hull {a,b,c})
   shows (f has_contour_integral 0) (linepath a b +++ linepath b c +++ linepath
c a)
proof −
  have contf: continuous_on (convex hull {a,b,c}) f
    by (metis assms holomorphic_on_imp_continuous_on)
  let ?pathint = λx y. contour_integral(linepath x y) f
  { fix y::complex
    assume fy: (f has_contour_integral y) (linepath a b +++ linepath b c +++
linepath c a)
       and ynz: y ≠ 0
    define K where K = 1 + max (dist a b) (max (dist b c) (dist c a))
    define e where e = norm y / K^2
    have K1: K ≥ 1  by (simp add: K_def max.coboundedI1)
    then have K: K > 0 by linarith
    have [iff]: dist a b ≤ K dist b c ≤ K dist c a ≤ K
      by (simp_all add: K_def)
    have e: e > 0
      unfolding e_def using ynz K1 by simp
    define At where At x y z n ⟷
        convex hull {x,y,z} ⊆ convex hull {a,b,c} ∧
        dist x y ≤ K/2^n ∧ dist y z ≤ K/2^n ∧ dist z x ≤ K/2^n ∧
        norm(?pathint x y + ?pathint y z + ?pathint z x) ≥ e*(K/2^n)^2
      for x y z n
    have At0: At a b c 0
      using fy
      by (simp add: At_def e_def has_chain_integral_chain_integral3)
    { fix x y z n
      assume At: At x y z n
      then have contf': continuous_on (convex hull {x,y,z}) f
        using contf At_def continuous_on_subset by metis
       have ∃x' y' z'. At x' y' z' (Suc n) ∧ convex hull {x',y',z'} ⊆ convex hull
{x,y,z}
        using At Cauchy_theorem_quadrisection [OF contf', of K/2^n e]
        apply (simp add: At_def algebra_simps)
        apply (meson convex_hull_subset empty_subsetI insert_subset subsetCE)
        done
    } note AtSuc = this
    obtain fa fb fc
      where f0 [simp]: fa 0 = a fb 0 = b fc 0 = c
        and cosb: ⋀n. convex hull {fa n, fb n, fc n} ⊆ convex hull {a,b,c}
```

```
      and dist: ⋀n. dist (fa n) (fb n) ≤ K/2^n
                ⋀n. dist (fb n) (fc n) ≤ K/2^n
                ⋀n. dist (fc n) (fa n) ≤ K/2^n
      and no: ⋀n. norm(?pathint (fa n) (fb n) +
                       ?pathint (fb n) (fc n) +
                       ?pathint (fc n) (fa n)) ≥ e * (K/2^n)^2
       and conv_le: ⋀n. convex hull {fa(Suc n), fb(Suc n), fc(Suc n)} ⊆ convex
hull {fa n, fb n, fc n}
     by (rule Chain3 [of At, OF At0 AtSuc]) (auto simp: At_def)
   obtain x where x: ⋀n. x ∈ convex hull {fa n, fb n, fc n}
   proof (rule bounded_closed_nest)
     show ⋀n. closed (convex hull {fa n, fb n, fc n})
       by (simp add: compact_imp_closed finite_imp_compact_convex_hull)
     show ⋀m n. m ≤ n ⟹ convex hull {fa n, fb n, fc n} ⊆ convex hull {fa m,
fb m, fc m}
       by (erule transitive_stepwise_le) (auto simp: conv_le)
   qed (fastforce intro: finite_imp_bounded_convex_hull)+
   then have xin: x ∈ convex hull {a,b,c}
     using assms f0 by blast
   then have fx: f field_differentiable at x within (convex hull {a,b,c})
     using assms holomorphic_on_def by blast
   { fix k n
     assume k: 0 < k
       and le:
         ⋀x' y' z'.
           ⟦dist x' y' ≤ k; dist y' z' ≤ k; dist z' x' ≤ k;
            x ∈ convex hull {x',y',z'};
            convex hull {x',y',z'} ⊆ convex hull {a,b,c}⟧
           ⟹
           cmod (?pathint x' y' + ?pathint y' z' + ?pathint z' x') * 10
                ≤ e * (dist x' y' + dist y' z' + dist z' x')^2
       and Kk: K / k < 2 ^ n
     have K / 2 ^ n < k using Kk k
       by (auto simp: field_simps)
     then have DD: dist (fa n) (fb n) ≤ k dist (fb n) (fc n) ≤ k dist (fc n) (fa n)
≤ k
       using dist [of n]  k
       by linarith+
     have dle: (dist (fa n) (fb n) + dist (fb n) (fc n) + dist (fc n) (fa n))^2
           ≤ (3 * K / 2 ^ n)^2
       using dist [of n] e K
       by (simp add: abs_le_square_iff [symmetric])
     have less10: ⋀x y::real. 0 < x ⟹ y ≤ 9*x ⟹ y < x*10
       by linarith
     have e * (dist (fa n) (fb n) + dist (fb n) (fc n) + dist (fc n) (fa n))^2 ≤ e *
(3 * K / 2 ^ n)^2
       using ynz dle e mult_le_cancel_left_pos by blast
     also have ... <
       cmod (?pathint (fa n) (fb n) + ?pathint (fb n) (fc n) + ?pathint (fc n) (fa
```

```
n)) * 10
        using no [of n] e K
      by (simp add: e_def field_simps) (simp only: zero_less_norm_iff [symmetric])
      finally have False
        using le [OF DD x cosb] by auto
    } then
    have ?thesis
      using holomorphic_point_small_triangle [OF xin contf fx, of e/10] e
      apply clarsimp
      apply (rule_tac y1=K/k in exE [OF real_arch_pow[of 2]], force+)
      done
  }
  moreover have f contour_integrable_on (linepath a b +++ linepath b c +++
linepath c a)
   by simp (meson contf continuous_on_subset contour_integrable_continuous_linepath
segments_subset_convex_hull(1)
                segments_subset_convex_hull(3) segments_subset_convex_hull(5))
  ultimately show ?thesis
    using has_contour_integral_integral by fastforce
qed
```

2.3 Version needing function holomorphic in interior only

```
lemma Cauchy_theorem_flat_lemma:
  assumes f: continuous_on (convex hull {a,b,c}) f
      and c: c - a = k *_R (b - a)
      and k: 0 ≤ k
    shows contour_integral (linepath a b) f + contour_integral (linepath b c) f +
          contour_integral (linepath c a) f = 0
proof -
  have fabc: continuous_on (closed_segment a b) f continuous_on (closed_segment
b c) f continuous_on (closed_segment c a) f
    using f continuous_on_subset segments_subset_convex_hull by metis+
  show ?thesis
  proof (cases k ≤ 1)
    case True show ?thesis
    by (simp add: contour_integral_split [OF fabc(1) k True c] contour_integral_reverse_linepath
fabc)
  next
    case False
    show ?thesis
    proof (subst contour_integral_split [symmetric])
      show b - a = (1/k) *_R (c - a)
        using False c by force
     show contour_integral (linepath a c) f + contour_integral (linepath c a) f =
0
        by (simp add: contour_integral_reverse_linepath fabc(3))
      show continuous_on (closed_segment a c) f
        by (metis closed_segment_commute fabc(3))
```

```
    qed (use False in auto)
  qed
qed

lemma Cauchy_theorem_flat:
  assumes f: continuous_on (convex hull {a,b,c}) f
      and c: c − a = k *R (b − a)
    shows contour_integral (linepath a b) f +
          contour_integral (linepath b c) f +
          contour_integral (linepath c a) f = 0
proof (cases 0 ≤ k)
  case True with assms show ?thesis
    by (blast intro: Cauchy_theorem_flat_lemma)
next
  case False
  have continuous_on (closed_segment a b) f continuous_on (closed_segment b
c) f continuous_on (closed_segment c a) f
    using f continuous_on_subset segments_subset_convex_hull by metis+
  moreover have contour_integral (linepath b a) f + contour_integral (linepath
a c) f +
                contour_integral (linepath c b) f = 0
  proof (rule Cauchy_theorem_flat_lemma [of b a c f 1−k])
    show continuous_on (convex hull {b, a, c}) f
      by (simp add: f insert_commute)
    show c − b = (1 − k) *R (a − b)
      using c by (auto simp: algebra_simps)
  qed (use False in auto)
  ultimately show ?thesis
   by (metis (no_types, lifting) contour_integral_reverse_linepath eq_neg_iff_add_eq_0
minus_add_cancel)
qed


proposition Cauchy_theorem_triangle_interior:
  assumes contf: continuous_on (convex hull {a,b,c}) f
      and holf:  f holomorphic_on interior (convex hull {a,b,c})
    shows (f has_contour_integral 0) (linepath a b +++ linepath b c +++ linepath
c a)
proof −
  define pathint where pathint ≡ λx y. contour_integral(linepath x y) f
  have fabc: continuous_on (closed_segment a b) f continuous_on (closed_segment
b c) f continuous_on (closed_segment c a) f
    using contf continuous_on_subset segments_subset_convex_hull by metis+
  have bounded (f ‘ (convex hull {a,b,c}))
   by (simp add: compact_continuous_image compact_convex_hull compact_imp_bounded
contf)
  then obtain B where 0 < B and Bnf: ⋀x. x ∈ convex hull {a,b,c} ⟹ norm
(f x) ≤ B
    by (auto simp: dest!: bounded_pos [THEN iffD1])
```

```
  have bounded (convex hull {a,b,c})
    by (simp add: bounded_convex_hull)
  then obtain C where C: 0 < C and Cno: ⋀y. y ∈ convex hull {a,b,c} ⟹
norm y < C
    using bounded_pos_less by blast
  then have diff_2C: norm(x − y) ≤ 2∗C
          if x: x ∈ convex hull {a, b, c} and y: y ∈ convex hull {a, b, c} for x y
  proof −
    have cmod x ≤ C
      using x by (meson Cno not_le not_less_iff_gr_or_eq)
    hence cmod (x − y) ≤ C + C
     using y by (meson Cno add_mono_thms_linordered_field(4) less_eq_real_def
norm_triangle_ineq4 order_trans)
    thus cmod (x − y) ≤ 2 ∗ C
      by (metis mult_2)
  qed
  have contf': continuous_on (convex hull {b,a,c}) f
    using contf by (simp add: insert_commute)
  { fix y::complex
    assume fy: (f has_contour_integral y) (linepath a b +++ linepath b c +++
linepath c a)
       and ynz: y ≠ 0
    have pi_eq_y: pathint a b + pathint b c + pathint c a= y
       unfolding pathint_def by (rule has_chain_integral_chain_integral3 [OF
fy])
    have ?thesis
    proof (cases c=a ∨ a=b ∨ b=c)
      case True then show ?thesis
        using Cauchy_theorem_flat [OF contf, of 0]
        using has_chain_integral_chain_integral3 [OF fy] ynz
        by (force simp: fabc contour_integral_reverse_linepath)
    next
      case False
      then have car3: card {a, b, c} = Suc (DIM(complex))
        by auto
      { assume interior(convex hull {a,b,c}) = {}
        then have collinear{a,b,c}
          using interior_convex_hull_eq_empty [OF car3]
          by (simp add: collinear_3_eq_affine_dependent)
        with False obtain d where c ≠ a a ≠ b b ≠ c c − b = d *R (a − b)
          by (auto simp: collinear_3 collinear_lemma)
        then have False
          using False Cauchy_theorem_flat [OF contf'] pi_eq_y ynz
            by (simp add: fabc add_eq_0_iff contour_integral_reverse_linepath
pathint_def)
      }
      then obtain d where d: d ∈ interior (convex hull {a, b, c})
        by blast
      { fix d1
```

```
      assume d1_pos: 0 < d1
        and d1: ⋀x x'. ⟦x∈convex hull {a, b, c}; x'∈convex hull {a, b, c}; cmod
(x' − x) < d1⟧
                    ⟹ cmod (f x' − f x) < cmod y / (24 * C)
      define e where e = min 1 (min (d1/(4*C)) ((norm y / 24 / C) / B))
      define shrink where shrink x = x − e *R (x − d) for x
      have e: 0 < e e ≤ 1 e ≤ d1 / (4 * C) e ≤ cmod y / 24 / C / B
        using d1_pos ‹C>0› ‹B>0› ynz by (simp_all add: e_def)
      have e_le_d1: e * (4 * C) ≤ d1
        using e ‹C>0› by (simp add: field_simps)
      have shrink a ∈ interior(convex hull {a,b,c})
           shrink b ∈ interior(convex hull {a,b,c})
           shrink c ∈ interior(convex hull {a,b,c})
        using d e by (auto simp: hull_inc mem_interior_convex_shrink shrink_def)
      then have fhp0: (f has_contour_integral 0)
                  (linepath (shrink a) (shrink b) +++ linepath (shrink b) (shrink c)
+++ linepath (shrink c) (shrink a))
        by (simp add: Cauchy_theorem_triangle holomorphic_on_subset [OF holf]
hull_minimal)
      then have f_0_shrink: pathint (shrink a) (shrink b) + pathint (shrink b)
(shrink c) + pathint (shrink c) (shrink a) = 0
        by (simp add: has_chain_integral_chain_integral3 pathint_def)
      have fpi_abc: f contour_integrable_on linepath (shrink a) (shrink b)
                    f contour_integrable_on linepath (shrink b) (shrink c)
                    f contour_integrable_on linepath (shrink c) (shrink a)
        using fhp0 by (auto simp: valid_path_join dest: has_contour_integral_integrable)
      have cmod_shr: ⋀x y. cmod (shrink y − shrink x − (y − x)) = e * cmod
(x − y)
        using e by (simp add: shrink_def real_vector.scale_right_diff_distrib
[symmetric])
      have sh_eq: ⋀a b d::complex. (b − e *R (b − d)) − (a − e *R (a − d)) −
(b − a) = e *R (a − b)
        by (simp add: algebra_simps)
      have cmod y / (24 * C) ≤ cmod y / cmod (b − a) / 12
        using False ‹C>0› diff_2C [of b a] ynz
        by (auto simp: field_split_simps hull_inc)
      have less_C: x * cmod u < C if u ∈ convex hull {a,b,c} 0 ≤ x x ≤ 1 for
x u
      proof (cases x=0)
        case False
        with that show ?thesis
          using Cno [of u] mult_left_le_one_le [of cmod u x] le_less_trans
norm_ge_zero by blast
      qed (simp add: ‹0<C›)
      { fix u v
        assume uv: u ∈ convex hull {a, b, c} v ∈ convex hull {a, b, c} u≠v
          and fpi_uv: f contour_integrable_on linepath (shrink u) (shrink v)
        have shr_uv: shrink u ∈ interior(convex hull {a,b,c})
                     shrink v ∈ interior(convex hull {a,b,c})
```

```
          using d e uv
          by (auto simp: hull_inc mem_interior_convex_shrink shrink_def)
          have cmod_fuv: ⋀x. 0≤x ⟹ x≤1 ⟹ cmod (f (linepath (shrink u)
(shrink v) x)) ≤ B
        using shr_uv by (blast intro: Bnf linepath_in_convex_hull interior_subset
[THEN subsetD])
          { fix x::real  assume x: 0≤x x≤1
          have |1 − x| ∗ cmod u < C |x| ∗ cmod v < C
            using uv x by (auto intro!: less_C)
          moreover have  |x| ∗ cmod d < C |1 − x| ∗ cmod d < C
            using x d interior_subset by (auto intro!: less_C)
          ultimately
          have cmod_less_4C: cmod ((1 − x) *R u − (1 − x) *R d) + cmod (x
*R v − x *R d) < (C+C) + (C+C)
         by (metis add_strict_mono le_less_trans norm_scaleR norm_triangle_ineq4)
          have ll: linepath (shrink u) (shrink v) x − linepath u v x = −e ∗ ((1 −
x) *R (u − d) + x *R (v − d))
         by (simp add: linepath_def shrink_def algebra_simps scaleR_conv_of_real)
          have cmod_less_dt: cmod (linepath (shrink u) (shrink v) x − linepath u
v x) < d1
            unfolding ll norm_mult scaleR_diff_right
                using ‹e>0› cmod_less_4C by (force intro: norm_triangle_lt
less_le_trans [OF _ e_le_d1])
           have cmod (f (linepath (shrink u) (shrink v) x)) ∗ cmod (shrink v −
shrink u − (v − u)) +
                        cmod (v − u) ∗ cmod (f (linepath (shrink u) (shrink v) x) −
f (linepath u v x))
                        ≤ B ∗ (cmod y / 24 / C / B ∗ 2 ∗ C) + 2 ∗ C ∗ (cmod y /
24 / C)
          proof (intro add_mono [OF mult_mono])
            show cmod (f (linepath (shrink u) (shrink v) x)) ≤ B
              using cmod_fuv x by blast
            have B ∗ (12 ∗ (e ∗ cmod (u − v))) ≤ 24 ∗ e ∗ C ∗ B
              using e ‹B>0› diff_2C [of u v] uv by (auto simp: field_simps)
            also have ... ≤ cmod y
              using ‹C>0› ‹B>0› e by (simp add: field_simps)
            finally show cmod (shrink v − shrink u − (v − u)) ≤ cmod y / 24 /
C / B ∗ 2 ∗ C
          using ‹0 < B› ‹0 < C› by (simp add: cmod_shr mult_ac divide_simps)
            have cmod (f (linepath (shrink u) (shrink v) x) − f (linepath u v x))
< cmod y / (24 ∗ C)
              using x uv shr_uv cmod_less_dt
               by (auto simp: hull_inc intro: d1 interior_subset [THEN subsetD]
linepath_in_convex_hull)
            also have ... ≤ cmod y / cmod (v − u) / 12
              using False uv ‹C>0› diff_2C [of v u] ynz
              by (auto simp: field_split_simps hull_inc)
            finally have cmod (f (linepath (shrink u) (shrink v) x) − f (linepath
u v x)) ≤ cmod y / cmod (v − u) / 12
```

```
            by simp
          then show cmod (v − u) ∗ cmod (f (linepath (shrink u) (shrink v) x)
− f (linepath u v x))
                  ≤ 2 ∗ C ∗ (cmod y / 24 / C)
            using uv C by (simp add: field_simps)
          qed (use ‹0 < B› in auto)
          also have ... ≤ cmod y / 6
            by simp
          finally have cmod (f (linepath (shrink u) (shrink v) x)) ∗ cmod (shrink
v − shrink u − (v − u)) +
                       cmod (v − u) ∗ cmod (f (linepath (shrink u) (shrink v) x) −
f (linepath u v x))
                       ≤ cmod y / 6 .
        } note cmod_diff_le = this
        have f_uv: continuous_on (closed_segment u v) f
      by (blast intro: uv continuous_on_subset [OF contf closed_segment_subset_convex_hull])
        have **: ⋀f' x' f x::complex. f'*x' − f*x = f' ∗ (x' − x) + x ∗ (f' − f)
          by (simp add: algebra_simps)
        have norm (pathint (shrink u) (shrink v) − pathint u v)
            ≤ (B*(norm y /24/C/B)*2*C + (2*C)*(norm y/24/C)) ∗ content
(cbox 0 (1::real))
          apply (rule has_integral_bound
                [of _ λx. f(linepath (shrink u) (shrink v) x) ∗ (shrink v − shrink
u) − f(linepath u v x)*(v − u)
                  _ 0 1])
          using ynz ‹0 < B› ‹0 < C›
      apply (simp_all add: pathint_def has_integral_diff has_contour_integral_linepath
[symmetric] has_contour_integral_integral
          fpi_uv f_uv contour_integrable_continuous_linepath del: le_divide_eq_numeral1)
          apply (auto simp: ** norm_triangle_le norm_mult cmod_diff_le simp
del: le_divide_eq_numeral1)
          done
        also have ... ≤ norm y / 6
          by simp
        finally have norm (pathint (shrink u) (shrink v) − pathint u v) ≤ norm
y / 6 .
        } note ∗ = this
        have norm (pathint (shrink a) (shrink b) − pathint a b) ≤ norm y / 6
          using False fpi_abc by (rule_tac ∗) (auto simp: hull_inc)
        moreover
        have norm (pathint (shrink b) (shrink c) − pathint b c) ≤ norm y / 6
          using False fpi_abc by (rule_tac ∗) (auto simp: hull_inc)
        moreover
        have norm (pathint (shrink c) (shrink a) − pathint c a) ≤ norm y / 6
          using False fpi_abc by (rule_tac ∗) (auto simp: hull_inc)
        ultimately
        have norm((pathint (shrink a) (shrink b) − pathint a b) +
              (pathint (shrink b) (shrink c) − pathint b c) + (pathint (shrink c)
(shrink a) − pathint c a))
```

```
            ≤ norm y / 6 + norm y / 6 + norm y / 6
          by (metis norm_triangle_le add_mono)
        also have ... = norm y / 2
          by simp
        finally have norm((pathint (shrink a) (shrink b) + pathint (shrink b)
(shrink c) + pathint (shrink c) (shrink a)) −
                        (pathint a b + pathint b c + pathint c a))
            ≤ norm y / 2
          by (simp add: algebra_simps)
        then
        have norm(pathint a b + pathint b c + pathint c a) ≤ norm y / 2
            by (simp add: f_0_shrink) (metis (mono_tags) add.commute mi-
nus_add_distrib norm_minus_cancel uminus_add_conv_diff)
        then have False
          using pi_eq_y ynz by auto
      }
      note ∗ = this
      have uniformly_continuous_on (convex hull {a,b,c}) f
       by (simp add: contf compact_convex_hull compact_uniformly_continuous)
      moreover have norm y / (24 ∗ C) > 0
        using ynz ‹C > 0› by auto
      ultimately obtain δ where δ > 0 and
        ∀ x∈convex hull {a, b, c}. ∀ x′∈convex hull {a, b, c}.
          dist x′ x < δ ⟶ dist (f x′) (f x) < cmod y / (24 ∗ C)
        using ‹C > 0› ynz unfolding uniformly_continuous_on_def dist_norm
by blast
      hence False using ∗[of δ] by (auto simp: dist_norm)
      then show ?thesis ..
    qed
  }
  moreover have f contour_integrable_on (linepath a b +++ linepath b c +++
linepath c a)
    using fabc contour_integrable_continuous_linepath by auto
  ultimately show ?thesis
    using has_contour_integral_integral by fastforce
qed
```

2.4 Version allowing finite number of exceptional points

```
proposition Cauchy_theorem_triangle_cofinite:
  assumes continuous_on (convex hull {a,b,c}) f
      and finite S
      and (⋀x. x ∈ interior(convex hull {a,b,c}) − S ⟹ f field_differentiable (at
x))
    shows (f has_contour_integral 0) (linepath a b +++ linepath b c +++ linepath
c a)
using assms
proof (induction card S arbitrary: a b c S rule: less_induct)
  case (less S a b c)
```

```
  show ?case
  proof (cases S={})
    case True with less show ?thesis
    by (fastforce simp: holomorphic_on_def field_differentiable_at_within Cauchy_theorem_triangle_interio
  next
    case False
    then obtain d S' where d: S = insert d S' d ∉ S'
      by (meson Set.set_insert all_not_in_conv)
    then show ?thesis
    proof (cases d ∈ convex hull {a,b,c})
      case False
     show (f has_contour_integral 0) (linepath a b +++ linepath b c +++ linepath
c a)
      proof (rule less.hyps)
       show ⋀x. x ∈ interior (convex hull {a, b, c}) − S' ⟹ f field_differentiable
at x
        using False d interior_subset by (auto intro!: less.prems)
    qed (use d less.prems in auto)
    next
      case True
      have ∗: convex hull {a, b, d} ⊆ convex hull {a, b, c}
        by (meson True hull_subset insert_subset convex_hull_subset)
       have abd: (f has_contour_integral 0) (linepath a b +++ linepath b d +++
linepath d a)
      proof (rule less.hyps)
       show ⋀x. x ∈ interior (convex hull {a, b, d}) − S' ⟹ f field_differentiable
at x
          using d not_in_interior_convex_hull_3
            by (clarsimp intro!: less.prems) (metis ∗ insert_absorb insert_subset
interior_mono)
      qed (use d continuous_on_subset [OF _ ∗] less.prems in auto)
      have ∗: convex hull {b, c, d} ⊆ convex hull {a, b, c}
        by (meson True hull_subset insert_subset convex_hull_subset)
       have bcd: (f has_contour_integral 0) (linepath b c +++ linepath c d +++
linepath d b)
      proof (rule less.hyps)
       show ⋀x. x ∈ interior (convex hull {b, c, d}) − S' ⟹ f field_differentiable
at x
          using d not_in_interior_convex_hull_3
            by (clarsimp intro!: less.prems) (metis ∗ insert_absorb insert_subset
interior_mono)
      qed (use d continuous_on_subset [OF _ ∗] less.prems in auto)
      have ∗: convex hull {c, a, d} ⊆ convex hull {a, b, c}
        by (meson True hull_subset insert_subset convex_hull_subset)
       have cad: (f has_contour_integral 0) (linepath c a +++ linepath a d +++
linepath d c)
      proof (rule less.hyps)
       show ⋀x. x ∈ interior (convex hull {c, a, d}) − S' ⟹ f field_differentiable
at x
```

```
        using d not_in_interior_convex_hull_3
          by (clarsimp intro!: less.prems) (metis * insert_absorb insert_subset
interior_mono)
      qed (use d continuous_on_subset [OF _ *] less.prems in auto)
      have f contour_integrable_on linepath a b
       using less.prems abd contour_integrable_joinD1 contour_integrable_on_def
by blast
      moreover have f contour_integrable_on linepath b c
       using less.prems bcd contour_integrable_joinD1 contour_integrable_on_def
by blast
      moreover have f contour_integrable_on linepath c a
       using less.prems cad contour_integrable_joinD1 contour_integrable_on_def
by blast
      ultimately have fpi: f contour_integrable_on (linepath a b +++ linepath b
c +++ linepath c a)
        by auto
      { fix y::complex
       assume fy: (f has_contour_integral y) (linepath a b +++ linepath b c +++
linepath c a)
          and ynz: y ≠ 0
        have cont_ad: continuous_on (closed_segment a d) f
       by (meson * continuous_on_subset less.prems(1) segments_subset_convex_hull(3))
        have cont_bd: continuous_on (closed_segment b d) f
       by (meson True closed_segment_subset_convex_hull continuous_on_subset
hull_subset insert_subset less.prems(1))
        have cont_cd: continuous_on (closed_segment c d) f
       by (meson * continuous_on_subset less.prems(1) segments_subset_convex_hull(2))
        have contour_integral (linepath a b) f = - (contour_integral (linepath b
d) f + (contour_integral (linepath d a) f))
             contour_integral (linepath b c) f = - (contour_integral (linepath c d)
f + (contour_integral (linepath d b) f))
             contour_integral (linepath c a) f = - (contour_integral (linepath a d)
f + contour_integral (linepath d c) f)
          using has_chain_integral_chain_integral3 [OF abd]
                has_chain_integral_chain_integral3 [OF bcd]
                has_chain_integral_chain_integral3 [OF cad]
          by (simp_all add: algebra_simps add_eq_0_iff)
       then have ?thesis
         using cont_ad cont_bd cont_cd fy has_chain_integral_chain_integral3
contour_integral_reverse_linepath by fastforce
      }
      then show ?thesis
       using fpi contour_integrable_on_def by blast
    qed
  qed
qed
```

2.5 Cauchy's theorem for an open starlike set

lemma *starlike_convex_subset*:
assumes *S*: $a \in S$ *closed_segment b c* $\subseteq S$ **and** *subs*: $\bigwedge x.\ x \in S \Longrightarrow$ *closed_segment a x* $\subseteq S$
shows *convex hull* $\{a,b,c\} \subseteq S$
proof −
have *convex hull* $\{b, c\} \subseteq S$
using *assms*(2) *segment_convex_hull* **by** *auto*
then have $\bigwedge u\ v\ d.\ [\![0 \le u;\ 0 \le v;\ u + v = 1;\ d \in$ *convex hull* $\{b, c\}]\!] \Longrightarrow u *_R a + v *_R d \in S$
by (*meson subs convexD convex_closed_segment ends_in_segment subsetCE*)
then show *?thesis*
by (*auto simp add*: *convex_hull_insert* [*of* $\{b,c\}$ *a*])
qed

lemma *triangle_contour_integrals_starlike_primitive*:
assumes *contf*: *continuous_on S f*
and *S*: $a \in S$ *open S*
and *x*: $x \in S$
and *subs*: $\bigwedge y.\ y \in S \Longrightarrow$ *closed_segment a y* $\subseteq S$
and *zer*: $\bigwedge b\ c.$ *closed_segment b c* $\subseteq S$
$\Longrightarrow$ *contour_integral* (*linepath a b*) *f* + *contour_integral* (*linepath b c*) *f* +
contour_integral (*linepath c a*) *f* = *0*
shows $((\lambda x.$ *contour_integral*(*linepath a x*) *f*) *has_field_derivative f x*) (*at x*)
proof −
let *?pathint* = $\lambda x\ y.$ *contour_integral*(*linepath x y*) *f*
{ fix *e y*
assume *e*: *0* < *e* **and** *bxe*: *ball x e* $\subseteq S$ **and** *close*: *cmod* $(y - x) < e$
have *y*: $y \in S$
using *bxe close* **by** (*force simp*: *dist_norm norm_minus_commute*)
have *cont_ayf*: *continuous_on* (*closed_segment a y*) *f*
using *contf continuous_on_subset subs y* **by** *blast*
have *xys*: *closed_segment x y* $\subseteq S$
by (*metis bxe centre_in_ball close closed_segment_subset convex_ball dist_norm dual_order.trans e mem_ball norm_minus_commute*)
have *?pathint a y − ?pathint a x = ?pathint x y*
using *zer* [*OF xys*] *contour_integral_reverse_linepath* [*OF cont_ayf*] *add_eq_0_iff* **by** *force*
} note [*simp*] = *this*
{ fix *e*::*real*
assume *e*: *0* < *e*
have *cont_atx*: *continuous* (*at x*) *f*
using *x S contf continuous_on_eq_continuous_at* **by** *blast*
then obtain *d1* **where** *d1*: *d1*>*0* **and** *d1_less*: $\bigwedge y.$ *cmod* $(y - x) < d1 \Longrightarrow$ *cmod* $(f\ y - f\ x) < e/2$
unfolding *continuous_at Lim_at dist_norm* **using** *e*
by (*drule_tac x*=*e*/*2* **in** *spec*) *force*
obtain *d2* **where** *d2*: *d2*>*0 ball x d2* $\subseteq S$ **using** ‹*open S*› *x*

```
      by (auto simp: open_contains_ball)
    have dpos: min d1 d2 > 0 using d1 d2 by simp
    { fix y
      assume yx: y ≠ x and close: cmod (y − x) < min d1 d2
      have y: y ∈ S
        using d2 close  by (force simp: dist_norm norm_minus_commute)
      have closed_segment x y ⊆ S
         using close d2  by (auto simp: dist_norm norm_minus_commute dest!:
segment_bound(1))
      then have fxy: f contour_integrable_on linepath x y
        by (metis contour_integrable_continuous_linepath continuous_on_subset
[OF contf])
      then obtain i where i: (f has_contour_integral i) (linepath x y)
        by (auto simp: contour_integrable_on_def)
       then have ((λw. f w − f x) has_contour_integral (i − f x ∗ (y − x)))
(linepath x y)
      by (rule has_contour_integral_diff [OF _ has_contour_integral_const_linepath])
      then have cmod (i − f x ∗ (y − x)) ≤ e / 2 ∗ cmod (y − x)
      proof (rule has_contour_integral_bound_linepath)
        show ⋀u. u ∈ closed_segment x y ⟹ cmod (f u − f x) ≤ e / 2
          by (meson close d1_less le_less_trans less_imp_le min.strict_boundedE
segment_bound1)
      qed (use e in simp)
      also have ... < e ∗ cmod (y − x)
        by (simp add: e yx)
      finally have cmod (?pathint x y − f x ∗ (y−x)) / cmod (y−x) < e
        using i yx  by (simp add: contour_integral_unique divide_less_eq)
    }
    then have ∃ d>0. ∀ y. y ≠ x ∧ cmod (y−x) < d ⟶ cmod (?pathint x y − f
x ∗ (y−x)) / cmod (y−x) < e
      using dpos by blast
  }
  then have (λy. (?pathint x y − f x ∗ (y − x)) /R cmod (y − x)) −x→ 0
    by (simp add: Lim_at dist_norm inverse_eq_divide)
  then have (λy. (1 / cmod (y − x)) ∗R (?pathint a y − (?pathint a x + f x ∗ (y
− x)))) −x→ 0
    using ‹open S› x
    by (force simp: dist_norm open_contains_ball inverse_eq_divide [symmetric]
eventually_at intro:  Lim_transform [OF _ tendsto_eventually])
  then show ?thesis
   by (simp add: has_field_derivative_def has_derivative_at2 bounded_linear_mult_right)
qed


lemma holomorphic_starlike_primitive:
  fixes f :: complex ⇒ complex
  assumes contf: continuous_on S f
      and S: starlike S and os: open S
      and k: finite k
```

```
      and fcd: ⋀x. x ∈ S − k ⟹ f field_differentiable at x
    shows ∃ g. ∀ x ∈ S. (g has_field_derivative f x) (at x)
proof −
  obtain a where a: a∈S and a_cs: ⋀x. x∈S ⟹ closed_segment a x ⊆ S
    using S by (auto simp: starlike_def)
  { fix x b c
    assume x ∈ S closed_segment b c ⊆ S
    then have abcs: convex hull {a, b, c} ⊆ S
      by (simp add: a a_cs starlike_convex_subset)
    then have continuous_on (convex hull {a, b, c}) f
      by (simp add: continuous_on_subset [OF contf])
     then have (f has_contour_integral 0) (linepath a b +++ linepath b c +++
linepath c a)
     using abcs interior_subset by (force intro: fcd Cauchy_theorem_triangle_cofinite
[OF _ k])
  } note 0 = this
  show ?thesis
  proof (intro exI ballI)
   show ⋀x. x ∈ S ⟹ ((λx. contour_integral (linepath a x) f) has_field_derivative
f x) (at x)
    using 0 a a_cs contf has_chain_integral_chain_integral3 os triangle_contour_integrals_starlike_primitiv
by force
  qed
qed

lemma Cauchy_theorem_starlike:
 ⟦open S; starlike S; finite k; continuous_on S f;
   ⋀x. x ∈ S − k ⟹ f field_differentiable at x;
   valid_path g; path_image g ⊆ S; pathfinish g = pathstart g⟧
   ⟹ (f has_contour_integral 0)  g
 by (metis holomorphic_starlike_primitive Cauchy_theorem_primitive at_within_open)

lemma Cauchy_theorem_starlike_simple:
  ⟦open S; starlike S; f holomorphic_on S; valid_path g; path_image g ⊆ S; pathfin-
ish g = pathstart g⟧
   ⟹ (f has_contour_integral 0) g
  using Cauchy_theorem_starlike [OF _ _ finite.emptyI]
 by (simp add: holomorphic_on_imp_continuous_on holomorphic_on_imp_differentiable_at)
```

2.6 Cauchy's theorem for a convex set

For a convex set we can avoid assuming openness and boundary analyticity

```
lemma triangle_contour_integrals_convex_primitive:
  assumes contf: continuous_on S f
      and S: a ∈ S convex S
      and x: x ∈ S
      and zer: ⋀b c. ⟦b ∈ S; c ∈ S⟧
                ⟹ contour_integral (linepath a b) f + contour_integral (linepath
b c) f +
```

contour_integral (linepath c a) f = 0
shows ((λx. *contour_integral*(*linepath a x*) *f*) *has_field_derivative f x*) (*at x within S*)
proof −
let *?pathint* = λx y. *contour_integral*(*linepath x y*) *f*
{ fix *y*
assume *y*: *y* ∈ *S*
have *cont_ayf*: *continuous_on* (*closed_segment a y*) *f*
using *S y* **by** (*meson contf continuous_on_subset convex_contains_segment*)
have *xys*: *closed_segment x y* ⊆ *S*
using *convex_contains_segment S x y* **by** *auto*
have *?pathint a y − ?pathint a x = ?pathint x y*
using *zer* [*OF x y*] *contour_integral_reverse_linepath* [*OF cont_ayf*] *add_eq_0_iff*
by *force*
} note [*simp*] = *this*
{ fix *e*::*real*
assume *e*: *0 < e*
have *cont_atx*: *continuous* (*at x within S*) *f*
using *x S contf* **by** (*simp add*: *continuous_on_eq_continuous_within*)
then obtain *d1* **where** *d1*: *d1>0* **and** *d1_less*: ⋀y. ⟦*y* ∈ *S*; *cmod* (*y − x*) < *d1*⟧ ⟹ *cmod* (*f y − f x*) < *e*/*2*
unfolding *continuous_within Lim_within dist_norm* **using** *e*
by (*drule_tac x*=*e*/*2* **in** *spec*) *force*
{ fix *y*
assume *yx*: *y* ≠ *x* **and** *close*: *cmod* (*y − x*) < *d1* **and** *y*: *y* ∈ *S*
have *fxy*: *f contour_integrable_on linepath x y*
using *convex_contains_segment S x y*
by (*blast intro*!: *contour_integrable_continuous_linepath continuous_on_subset* [*OF contf*])
then obtain *i* **where** *i*: (*f has_contour_integral i*) (*linepath x y*)
by (*auto simp*: *contour_integrable_on_def*)
then have ((λw. *f w − f x*) *has_contour_integral* (*i − f x* ∗ (*y − x*))) (*linepath x y*)
by (*rule has_contour_integral_diff* [*OF _ has_contour_integral_const_linepath*])
then have *cmod* (*i − f x* ∗ (*y − x*)) ≤ *e* / *2* ∗ *cmod* (*y − x*)
proof (*rule has_contour_integral_bound_linepath*)
show ⋀u. *u* ∈ *closed_segment x y* ⟹ *cmod* (*f u − f x*) ≤ *e* / *2*
by (*meson assms*(*3*) *close convex_contains_segment d1_less le_less_trans less_imp_le segment_bound1 subset_iff x y*)
qed (*use e* **in** *simp*)
also have ... < *e* ∗ *cmod* (*y − x*)
by (*simp add*: *e yx*)
finally have *cmod* (*?pathint x y − f x* ∗ (*y*−*x*)) / *cmod* (*y*−*x*) < *e*
using *i yx* **by** (*simp add*: *contour_integral_unique divide_less_eq*)
}
then have ∃ *d*>*0*. ∀ *y*∈*S*. *y* ≠ *x* ∧ *cmod* (*y*−*x*) < *d* ⟶ *cmod* (*?pathint x y − f x* ∗ (*y*−*x*)) / *cmod* (*y*−*x*) < *e*
using *d1* **by** *blast*
}

then have $((\lambda y.\ (?pathint\ x\ y - f\ x * (y - x))\ /_R\ cmod\ (y - x)) \longrightarrow 0)$ $(at\ x\ within\ S)$
by $(simp\ add: Lim\_within\ dist\_norm\ inverse\_eq\_divide)$
then have $((\lambda y.\ (1\ /\ cmod\ (y - x)) *_R (?pathint\ a\ y - (?pathint\ a\ x + f\ x * (y - x)))) \longrightarrow 0)$
$(at\ x\ within\ S)$
using *linordered_field_no_ub*
by $(force\ simp: inverse\_eq\_divide\ [symmetric]\ eventually\_at\ intro: Lim\_transform\ [OF\ \_\ tendsto\_eventually])$
then show *?thesis*
by $(simp\ add: has\_field\_derivative\_def\ has\_derivative\_within\ bounded\_linear\_mult\_right)$
qed

lemma *contour_integral_convex_primitive*:
assumes *convex S continuous_on S f*
$\bigwedge a\ b\ c.\ [\![a \in S;\ b \in S;\ c \in S]\!] \Longrightarrow (f\ has\_contour\_integral\ 0)\ (linepath\ a\ b +++ linepath\ b\ c +++ linepath\ c\ a)$
obtains *g* **where** $\bigwedge x.\ x \in S \Longrightarrow (g\ has\_field\_derivative\ f\ x)\ (at\ x\ within\ S)$
proof $(cases\ S{=}\{\})$
case *False*
with *assms that* **show** *?thesis*
by $(blast\ intro: triangle\_contour\_integrals\_convex\_primitive\ has\_chain\_integral\_chain\_integral3)$
qed *auto*

lemma *holomorphic_convex_primitive*:
fixes $f :: complex \Rightarrow complex$
assumes *convex S finite K* **and** *contf*: *continuous_on S f*
and *fd*: $\bigwedge x.\ x \in interior\ S - K \Longrightarrow f\ field\_differentiable\ at\ x$
obtains *g* **where** $\bigwedge x.\ x \in S \Longrightarrow (g\ has\_field\_derivative\ f\ x)\ (at\ x\ within\ S)$
proof $(rule\ contour\_integral\_convex\_primitive\ [OF\ \langle convex\ S\rangle\ contf\ Cauchy\_theorem\_triangle\_cofinite])$
have $*$: $convex\ hull\ \{a, b, c\} \subseteq S$ **if** $a \in S\ b \in S\ c \in S$ **for** *a b c*
by $(simp\ add: \langle convex\ S\rangle\ hull\_minimal\ that)$
show $continuous\_on\ (convex\ hull\ \{a, b, c\})\ f$ **if** $a \in S\ b \in S\ c \in S$ **for** *a b c*
by $(meson * contf\ continuous\_on\_subset\ that)$
show $f\ field\_differentiable\ at\ x$ **if** $a \in S\ b \in S\ c \in S\ x \in interior\ (convex\ hull\ \{a, b, c\}) - K$ **for** *a b c x*
by $(metis * DiffD1\ DiffD2\ DiffI\ fd\ interior\_mono\ subsetCE\ that)$
qed $(use\ assms\ \textbf{in}\ \langle force+\rangle)$

lemma *holomorphic_convex_primitive'*:
fixes $f :: complex \Rightarrow complex$
assumes *convex S* **and** *open S* **and** *f holomorphic_on S*
obtains *g* **where** $\bigwedge x.\ x \in S \Longrightarrow (g\ has\_field\_derivative\ f\ x)\ (at\ x\ within\ S)$
proof $(rule\ holomorphic\_convex\_primitive)$
fix *x* **assume** $x \in interior\ S - \{\}$
with *assms* **show** *f field_differentiable at x*
by $(auto\ intro!: holomorphic\_on\_imp\_differentiable\_at\ simp: interior\_open)$
qed $(use\ assms\ \textbf{in}\ \langle auto\ intro: holomorphic\_on\_imp\_continuous\_on\rangle)$

corollary *Cauchy_theorem_convex*:
  ⟦*continuous_on S f*; *convex S*; *finite K*;
    ⋀*x*. *x* ∈ *interior S* − *K* ⟹ *f field_differentiable at x*;
    valid_path g; *path_image g* ⊆ *S*; *pathfinish g* = *pathstart g*⟧
  ⟹ (*f has_contour_integral 0*) *g*
 by (*metis holomorphic_convex_primitive Cauchy_theorem_primitive*)

corollary *Cauchy_theorem_convex_simple*:
 assumes *holf*: *f holomorphic_on S*
    and *convex S valid_path g path_image g* ⊆ *S pathfinish g* = *pathstart g*
 shows (*f has_contour_integral 0*) *g*
proof −
 have *f holomorphic_on interior S*
   by (*meson holf holomorphic_on_subset interior_subset*)
 with *Cauchy_theorem_convex* [**where** *K* = {}] **show** *?thesis*
   using *assms*
  by (*metis Diff_empty finite.emptyI holomorphic_on_imp_continuous_on holomorphic_on_imp_differentiable_at open_interior*)
qed

In particular for a disc

corollary *Cauchy_theorem_disc*:
  ⟦*finite K*; *continuous_on* (*cball a e*) *f*;
    ⋀*x*. *x* ∈ *ball a e* − *K* ⟹ *f field_differentiable at x*;
    valid_path g; *path_image g* ⊆ *cball a e*;
    pathfinish g = *pathstart g*⟧ ⟹ (*f has_contour_integral 0*) *g*
 by (*auto intro*: *Cauchy_theorem_convex*)

corollary *Cauchy_theorem_disc_simple*:
  ⟦*f holomorphic_on* (*ball a e*); *valid_path g*; *path_image g* ⊆ *ball a e*;
    pathfinish g = *pathstart g*⟧ ⟹ (*f has_contour_integral 0*) *g*
by (*simp add*: *Cauchy_theorem_convex_simple*)

2.7 Generalize integrability to local primitives

lemma *contour_integral_local_primitive_lemma*:
 fixes *f* :: *complex*⇒*complex*
 assumes *gpd*: *g piecewise_differentiable_on* {*a*..*b*}
    and *dh*: ⋀*x*. *x* ∈ *S* ⟹ (*f has_field_derivative f*′ *x*) (*at x within S*)
    and *gs*: ⋀*x*. *x* ∈ {*a*..*b*} ⟹ *g x* ∈ *S*
 shows
   (λ*x*. *f*′ (*g x*) ∗ *vector_derivative g* (*at x within* {*a*..*b*})) *integrable_on* {*a*..*b*}
proof (*cases cbox a b* = {})
 case *False*
 then show *?thesis*
  unfolding *integrable_on_def* **by** (*auto intro*: *assms contour_integral_primitive_lemma*)
qed *auto*

lemma *contour_integral_local_primitive_any*:

```
  fixes f :: complex ⇒ complex
  assumes gpd: g piecewise_differentiable_on {a..b}
      and dh: ⋀x. x ∈ S
              ⟹ ∃ d h. 0 < d ∧
                        (∀ y. norm(y − x) < d ⟶ (h has_field_derivative f y) (at y
within S))
      and gs: ⋀x. x ∈ {a..b} ⟹ g x ∈ S
  shows (λx. f(g x) ∗ vector_derivative g (at x)) integrable_on {a..b}
proof −
  { fix x
    assume x: a ≤ x x ≤ b
    obtain d h where d: 0 < d
              and h: (⋀y. norm(y − g x) < d ⟹ (h has_field_derivative f y) (at
y within S))
      using x gs dh by (metis atLeastAtMost_iff)
    have continuous_on {a..b} g using gpd piecewise_differentiable_on_def by
blast
    then obtain e where e: e>0 and lessd: ⋀x'. x' ∈ {a..b} ⟹ |x' − x| < e
⟹ cmod (g x' − g x) < d
      using x d by (fastforce simp: dist_norm continuous_on_iff)
    have ∃ e>0. ∀ u v. u ≤ x ∧ x ≤ v ∧ {u..v} ⊆ ball x e ∧ (u ≤ v ⟶ a ≤ u ∧
v ≤ b) ⟶
                       (λx. f (g x) ∗ vector_derivative g (at x)) integrable_on {u..v}
    proof −
      have (λx. f (g x) ∗ vector_derivative g (at x within {u..v})) integrable_on
{u..v}
        if u ≤ x x ≤ v and ball: {u..v} ⊆ ball x e and auvb: u ≤ v ⟹ a ≤ u ∧ v
≤ b
        for u v
      proof (rule contour_integral_local_primitive_lemma)
        show g piecewise_differentiable_on {u..v}
        by (metis atLeastatMost_subset_iff gpd piecewise_differentiable_on_subset
auvb)
        show ⋀x. x ∈ g ‘ {u..v} ⟹ (h has_field_derivative f x) (at x within g ‘
{u..v})
        using that by (force simp: ball_def dist_norm intro: lessd gs DERIV_subset
[OF h])
      qed auto
      then show ?thesis
        using e integrable_on_localized_vector_derivative by blast
    qed
  } then
  show ?thesis
    by (force simp: intro!: integrable_on_little_subintervals [of a b, simplified])
qed

lemma contour_integral_local_primitive:
  fixes f :: complex ⇒ complex
  assumes g: valid_path g path_image g ⊆ S
```

```
    and dh: ⋀x. x ∈ S
          ⟹ ∃ d h. 0 < d ∧
                (∀ y. norm(y − x) < d ⟶ (h has_field_derivative f y) (at y
within S))
  shows f contour_integrable_on g
proof −
 have (λx. f (g x) ∗ vector_derivative g (at x)) integrable_on {0..1}
  using contour_integral_local_primitive_any [OF _ dh] g
  unfolding path_image_def valid_path_def
 by (metis (no_types, lifting) image_subset_iff piecewise_C1_imp_differentiable)
 then show ?thesis
  using contour_integrable_on by presburger
qed
```

In particular if a function is holomorphic

```
lemma contour_integrable_holomorphic:
 assumes contf: continuous_on S f
    and os: open S
    and k: finite k
    and g: valid_path g path_image g ⊆ S
    and fcd: ⋀x. x ∈ S − k ⟹ f field_differentiable at x
  shows f contour_integrable_on g
proof −
 { fix z
  assume z: z ∈ S
  obtain d where d>0 and d: ball z d ⊆ S using ‹open S› z
   by (auto simp: open_contains_ball)
  then have contfb: continuous_on (ball z d) f
   using contf continuous_on_subset by blast
  obtain h where ∀ y∈ball z d. (h has_field_derivative f y) (at y within ball z d)
    by (metis holomorphic_convex_primitive [OF convex_ball k contfb fcd] d
interior_subset Diff_iff subsetD)
  then have ∀ y∈ball z d. (h has_field_derivative f y) (at y within S)
   by (metis open_ball at_within_open d os subsetCE)
  then have ∃ h. (∀ y. cmod (y − z) < d ⟶ (h has_field_derivative f y) (at y
within S))
    by (force simp: dist_norm norm_minus_commute)
  then have ∃ d h. 0 < d ∧ (∀ y. cmod (y − z) < d ⟶ (h has_field_derivative
f y) (at y within S))
    using ‹0 < d› by blast
 }
 then show ?thesis
  by (rule contour_integral_local_primitive [OF g])
qed

lemma contour_integrable_holomorphic_simple:
 assumes fh: f holomorphic_on S
    and os: open S
    and g: valid_path g path_image g ⊆ S
```

shows *f contour_integrable_on g*
proof −
have $\bigwedge x.\ x \in S \Longrightarrow f$ *field_differentiable at x*
using *fh holomorphic_on_imp_differentiable_at os* **by** *blast*
moreover have *continuous_on S f*
by (*simp add*: *fh holomorphic_on_imp_continuous_on*)
ultimately show *?thesis*
by (*metis Diff_empty contour_integrable_holomorphic finite.emptyI g os*)
qed

lemma *continuous_on_inversediff*:
fixes *z*:: *'a*::*real_normed_field* **shows** $z \notin S \Longrightarrow$ *continuous_on S* ($\lambda w.\ 1\ /\ (w - z)$)
by (*rule continuous_intros* | *force*)+

lemma *contour_integrable_inversediff*:
assumes *g*: *valid_path g*
and *notin*: $z \notin$ *path_image g*
shows ($\lambda w.\ 1\ /\ (w-z)$) *contour_integrable_on g*
proof (*rule contour_integrable_holomorphic_simple*)
show ($\lambda w.\ 1\ /\ (w-z)$) *holomorphic_on UNIV* − {*z*}
by (*auto simp*: *holomorphic_on_open open_delete intro*!: *derivative_eq_intros*)
qed (*use assms* **in** *auto*)

Key fact that path integral is the same for a "nearby" path. This is the main lemma for the homotopy form of Cauchy's theorem and is also useful if we want "without loss of generality" to assume some nice properties of a path (e.g. smoothness). It can also be used to define the integrals of analytic functions over arbitrary continuous paths. This is just done for winding numbers now.

A technical definition to avoid duplication of similar proofs, for paths joined at the ends versus looping paths

definition *linked_paths* :: *bool* ⇒ (*real* ⇒ *'a*) ⇒ (*real* ⇒ *'a*::*topological_space*) ⇒ *bool*
where *linked_paths atends g h* ==
(*if atends then pathstart h* = *pathstart g* ∧ *pathfinish h* = *pathfinish g*
else pathfinish g = *pathstart g* ∧ *pathfinish h* = *pathstart h*)

This formulation covers two cases: g and h share their start and end points; g and h both loop upon themselves.

lemma *contour_integral_nearby*:
assumes *os*: *open S* **and** *p*: *path p path_image p* ⊆ *S*
shows ∃ *d*. *0* < *d* ∧
(∀ *g h*. *valid_path g* ∧ *valid_path h* ∧
(∀ *t* ∈ {*0*..*1*}. *norm*(*g t* − *p t*) < *d* ∧ *norm*(*h t* − *p t*) < *d*) ∧
linked_paths atends g h
⟶ *path_image g* ⊆ *S* ∧ *path_image h* ⊆ *S* ∧

(∀ *f*. *f holomorphic_on S* ⟶ *contour_integral h f* = *contour_integral g f*))
proof −
have ∀ *z*. ∃ *e*. *z* ∈ *path_image p* ⟶ 0 < *e* ∧ *ball z e* ⊆ *S*
using *open_contains_ball os p*(*2*) **by** *blast*
then obtain *ee* **where** *ee*: ⋀*z*. *z* ∈ *path_image p* ⟹ 0 < *ee z* ∧ *ball z* (*ee z*) ⊆ *S*
by *metis*
define *cover* **where** *cover* = (λ*z*. *ball z* (*ee z*/*3*)) ‘ (*path_image p*)
have *compact* (*path_image p*)
by (*metis p*(*1*) *compact_path_image*)
moreover have *path_image p* ⊆ (⋃ *c*∈*path_image p*. *ball c* (*ee c* / *3*))
using *ee* **by** *auto*
ultimately have ∃ *D* ⊆ *cover*. *finite D* ∧ *path_image p* ⊆ ⋃ *D*
by (*simp add*: *compact_eq_Heine_Borel cover_def*)
then obtain *D* **where** *D*: *D* ⊆ *cover finite D path_image p* ⊆ ⋃ *D*
by *blast*
then obtain *k* **where** *k*: *k* ⊆ {*0..1*} *finite k* **and** *D_eq*: *D* = ((λ*z*. *ball z* (*ee z* / *3*)) ∘ *p*) ‘ *k*
unfolding *cover_def path_image_def image_comp*
by (*meson finite_subset_image*)
then have *kne*: *k* ≠ {}
using *D* **by** *auto*
have *pi*: ⋀*i*. *i* ∈ *k* ⟹ *p i* ∈ *path_image p*
using *k* **by** (*auto simp*: *path_image_def*)
then have *eepi*: ⋀*i*. *i* ∈ *k* ⟹ 0 < *ee*((*p i*))
by (*metis ee*)
define *e* **where** *e* = *Min*((*ee* ∘ *p*) ‘ *k*)
have *fin_eep*: *finite* ((*ee* ∘ *p*) ‘ *k*)
using *k* **by** *blast*
have 0 < *e*
using *ee k* **by** (*simp add*: *kne e_def Min_gr_iff* [*OF fin_eep*] *eepi*)
have *uniformly_continuous_on* {*0..1*} *p*
using *p* **by** (*simp add*: *path_def compact_uniformly_continuous*)
then obtain *d*::*real* **where** *d*: *d*>*0*
and *de*: ⋀*x x*′. |*x*′ − *x*| < *d* ⟹ *x*∈{*0..1*} ⟹ *x*′∈{*0..1*} ⟹ *cmod* (*p x*′ − *p x*) < *e*/*3*
unfolding *uniformly_continuous_on_def dist_norm real_norm_def*
by (*metis divide_pos_pos* ‹*0* < *e*› *zero_less_numeral*)
then obtain *N*::*nat* **where** *N*: *N*>*0 inverse N* < *d*
using *real_arch_inverse* [*of d*] **by** *auto*
show *?thesis*
proof (*intro exI conjI allI*; *clarify?*)
show *e*/*3* > *0*
using ‹*0* < *e*› **by** *simp*
fix *g h*
assume *g*: *valid_path g* **and** *ghp*: ∀ *t*∈{*0..1*}. *cmod* (*g t* − *p t*) < *e* / *3* ∧ *cmod* (*h t* − *p t*) < *e* / *3*
and *h*: *valid_path h*

```
    and joins: linked_paths atends g h
  { fix t::real
    assume t: 0 ≤ t t ≤ 1
    then obtain u where u: u ∈ k and ptu: p t ∈ ball(p u) (ee(p u) / 3)
      using ‹path_image p ⊆ ⋃ D› D_eq by (force simp: path_image_def)
    then have ele: e ≤ ee (p u) using fin_eep
      by (simp add: e_def)
    have cmod (g t − p t) < e / 3 cmod (h t − p t) < e / 3
      using ghp t by auto
    with ele have cmod (g t − p t) < ee (p u) / 3
                  cmod (h t − p t) < ee (p u) / 3
      by linarith+
    then have g t ∈ ball(p u) (ee(p u))  h t ∈ ball(p u) (ee(p u))
      using norm_diff_triangle_ineq [of g t p t p t p u]
            norm_diff_triangle_ineq [of h t p t p t p u] ptu eepi u
      by (force simp: dist_norm ball_def norm_minus_commute)+
    then have g t ∈ S h t ∈ S using ee u k
      by (auto simp: path_image_def ball_def)
  }
  then have ghs: path_image g ⊆ S path_image h ⊆ S
    by (auto simp: path_image_def)
  moreover
  { fix f
    assume fhols: f holomorphic_on S
    then have fpa: f contour_integrable_on g  f contour_integrable_on h
    using g ghs h holomorphic_on_imp_continuous_on os contour_integrable_holomorphic_simple
      by blast+
    have contf: continuous_on S f
      by (simp add: fhols holomorphic_on_imp_continuous_on)
    { fix z
      assume z: z ∈ path_image p
      have f holomorphic_on ball z (ee z)
        using fhols ee z holomorphic_on_subset by blast
      then have ∃ff. (∀ w ∈ ball z (ee z). (ff has_field_derivative f w) (at w))
        using holomorphic_convex_primitive [of ball z (ee z) {} f, simplified]
          by (metis open_ball at_within_open holomorphic_on_def holomor-
phic_on_imp_continuous_on mem_ball)
    }
    then obtain ff where ff:
        ⋀z w. ⟦z ∈ path_image p; w ∈ ball z (ee z)⟧ ⟹ (ff z has_field_derivative
f w) (at w)
      by metis
    { fix n
      assume n: n ≤ N
    then have contour_integral(subpath 0 (n/N) h) f − contour_integral(subpath
0 (n/N) g) f =
                        contour_integral(linepath (g(n/N)) (h(n/N))) f − con-
tour_integral(linepath (g 0) (h 0)) f
      proof (induct n)
```

```
      case 0 show ?case by simp
    next
      case (Suc n)
      obtain t where t: t ∈ k and p (n/N) ∈ ball(p t) (ee(p t) / 3)
       using ‹path_image p ⊆ ⋃D› [THEN subsetD, where c=p (n/N)] D_eq
N Suc.prems
        by (force simp: path_image_def)
      then have ptu: cmod (p t − p (n/N)) < ee (p t) / 3
        by (simp add: dist_norm)
      have e3le: e/3 ≤ ee (p t) / 3  using fin_eep t
        by (simp add: e_def)
      { fix x
        assume x: n/N ≤ x x ≤ (1 + n)/N
        then have nN01: 0 ≤ n/N (1 + n)/N ≤ 1
          using Suc.prems by auto
        then have x01: 0 ≤ x x ≤ 1
          using x by linarith+
        have cmod (p t − p x)  < ee (p t) / 3 + e/3
        proof (rule norm_diff_triangle_less [OF ptu de])
          show |real n / real N − x| < d
            using x N by (auto simp: field_simps)
        qed (use x01 Suc.prems in auto)
        then have ptx: cmod (p t − p x) < 2∗ee (p t)/3
          using e3le eepi [OF t] by simp
        have cmod (p t − g x) < 2∗ee (p t)/3 + e/3
          using ghp x01
      by (force simp add: norm_minus_commute intro!: norm_diff_triangle_less
[OF ptx])
        also have ... ≤ ee (p t)
          using e3le eepi [OF t] by simp
        finally have gg: cmod (p t − g x) < ee (p t) .
        have cmod (p t − h x) < 2∗ee (p t)/3 + e/3
          using ghp x01
      by (force simp add: norm_minus_commute intro!: norm_diff_triangle_less
[OF ptx])
        also have ... ≤ ee (p t)
          using e3le eepi [OF t] by simp
        finally have cmod (p t − g x) < ee (p t) cmod (p t − h x) < ee (p t)
          using gg by auto
      } note ptgh_ee = this
        have closed_segment (g (n/N)) (h (n/N)) = path_image (linepath (h
(n/N)) (g (n/N)))
        by (simp add: closed_segment_commute)
      also have pi_hgn: ... ⊆ ball (p t) (ee (p t))
        using ptgh_ee [of n/N] Suc.prems
       by (auto simp: field_simps dist_norm dest: segment_furthest_le [where
y=p t])
      finally have gh_ns: closed_segment (g (n/N)) (h (n/N)) ⊆ S
        using ee pi t by blast
```

```
      have pi_ghn': path_image (linepath (g ((1 + n) / N)) (h ((1 + n) / N)))
⊆ ball (p t) (ee (p t))
        using ptgh_ee [of (1+n)/N] Suc.prems
        by (auto simp: field_simps dist_norm dest: segment_furthest_le [where
y=p t])
      then have gh_n's: closed_segment (g ((1 + n) / N)) (h ((1 + n) / N))
⊆ S
        using ‹N>0› Suc.prems ee pi t
        by (auto simp: Path_Connected.path_image_join field_simps)
      have pi_subset_ball:
          path_image (subpath (n/N) ((1+n) / N) g +++ linepath (g ((1+n)
/ N)) (h ((1+n) / N)) +++
                      subpath ((1+n) / N) (n/N) h +++ linepath (h (n/N)) (g
(n/N)))
            ⊆ ball (p t) (ee (p t))
      proof (intro subset_path_image_join pi_hgn pi_ghn')
       show path_image (subpath (n/N) ((1+n) / N) g) ⊆ ball (p t) (ee (p t))
            path_image (subpath ((1+n) / N) (n/N) h) ⊆ ball (p t) (ee (p t))
        using ‹N>0› Suc.prems
        by (auto simp: path_image_subpath dist_norm field_simps ptgh_ee)
      qed
      have pi0: (f has_contour_integral 0)
                (subpath (n/ N) ((Suc n)/N) g +++ linepath(g ((Suc n) / N))
(h((Suc n) / N)) +++
                    subpath ((Suc n) / N) (n/N) h +++ linepath(h (n/N)) (g
(n/N)))
      proof (rule Cauchy_theorem_primitive)
        show ⋀x. x ∈ ball (p t) (ee (p t))
                 ⟹ (ff (p t) has_field_derivative f x) (at x within ball (p t) (ee
(p t)))
          by (metis ff open_ball at_within_open pi t)
      qed (use Suc.prems pi_subset_ball in ‹simp_all add: valid_path_subpath
g h›)
      have fpa1: f contour_integrable_on subpath (n/N) (real (Suc n) / real N)
g
        using Suc.prems by (simp add: contour_integrable_subpath g fpa)
      have fpa2: f contour_integrable_on linepath (g (real (Suc n) / real N)) (h
(real (Suc n) / real N))
        using gh_n's
          by (auto intro!: contour_integrable_continuous_linepath continu-
ous_on_subset [OF contf])
      have fpa3: f contour_integrable_on linepath (h (n/N)) (g (n/N))
        using gh_ns
      by (auto simp: closed_segment_commute intro!: contour_integrable_continuous_linepath
continuous_on_subset [OF contf])
      have eq0: contour_integral (subpath (n/N) ((Suc n) / real N) g) f +
                contour_integral (linepath (g ((Suc n) / N)) (h ((Suc n) / N))) f
+
                 contour_integral (subpath ((Suc n) / N) (n/N) h) f +
```

```
                contour_integral (linepath (h (n/N)) (g (n/N))) f = 0
          using contour_integral_unique [OF pi0] Suc.prems
          by (simp add: g h fpa valid_path_subpath contour_integrable_subpath
                fpa1 fpa2 fpa3 algebra_simps del: of_nat_Suc)
        have *: ⋀hn he hn' gn gd gn' hgn ghn gh0 ghn'.
                ⟦hn − gn = ghn − gh0;
                 gd + ghn' + he + hgn = (0::complex);
                  hn − he = hn'; gn + gd = gn'; hgn = −ghn⟧ ⟹ hn' − gn' =
ghn' − gh0
          by (auto simp: algebra_simps)
        have contour_integral (subpath 0 (n/N) h) f − contour_integral (subpath
((Suc n) / N) (n/N) h) f =
              contour_integral (subpath 0 (n/N) h) f + contour_integral (subpath
(n/N) ((Suc n) / N) h) f
          unfolding reversepath_subpath [symmetric, of ((Suc n) / N)]
            using Suc.prems by (simp add: h fpa contour_integral_reversepath
valid_path_subpath contour_integrable_subpath)
        also have ... = contour_integral (subpath 0 ((Suc n) / N) h) f
            using Suc.prems by (simp add: contour_integral_subpath_combine h
fpa)
        finally have pi0_eq:
              contour_integral (subpath 0 (n/N) h) f − contour_integral (subpath
((Suc n) / N) (n/N) h) f =
              contour_integral (subpath 0 ((Suc n) / N) h) f .
        show ?case
        proof (rule * [OF Suc.hyps eq0 pi0_eq])
          show contour_integral (subpath 0 (n/N) g) f +
                 contour_integral (subpath (n/N) ((Suc n) / N) g) f =
                 contour_integral (subpath 0 ((Suc n) / N) g) f
            using Suc.prems contour_integral_subpath_combine fpa(1) g by auto
           show contour_integral (linepath (h (n/N)) (g (n/N))) f = − con-
tour_integral (linepath (g (n/N)) (h (n/N))) f
            by (metis contour_integral_unique fpa3 has_contour_integral_integral
has_contour_integral_reverse_linepath)
        qed (use Suc.prems in auto)
      qed
      } note ind = this
      have contour_integral h f = contour_integral g f
        using ind [OF order_refl] N joins
      by (simp add: linked_paths_def pathstart_def pathfinish_def split: if_split_asm)
    }
    ultimately
   show path_image g ⊆ S ∧ path_image h ⊆ S ∧ (∀f. f holomorphic_on S ⟶
contour_integral h f = contour_integral g f)
      by metis
  qed
qed
```

lemma
assumes *open S path p path_image p ⊆ S*
shows *contour_integral_nearby_ends*:
∃ *d*. *0 < d* ∧
(∀ *g h*. *valid_path g* ∧ *valid_path h* ∧
(∀ *t* ∈ {*0..1*}. *norm*(*g t* − *p t*) < *d* ∧ *norm*(*h t* − *p t*) < *d*) ∧
pathstart h = *pathstart g* ∧ *pathfinish h* = *pathfinish g*
⟶ *path_image g* ⊆ *S* ∧
path_image h ⊆ *S* ∧
(∀ *f*. *f holomorphic_on S*
⟶ *contour_integral h f* = *contour_integral g f*))
and *contour_integral_nearby_loops*:
∃ *d*. *0 < d* ∧
(∀ *g h*. *valid_path g* ∧ *valid_path h* ∧
(∀ *t* ∈ {*0..1*}. *norm*(*g t* − *p t*) < *d* ∧ *norm*(*h t* − *p t*) < *d*) ∧
pathfinish g = *pathstart g* ∧ *pathfinish h* = *pathstart h*
⟶ *path_image g* ⊆ *S* ∧
path_image h ⊆ *S* ∧
(∀ *f*. *f holomorphic_on S*
⟶ *contour_integral h f* = *contour_integral g f*))
using *contour_integral_nearby* [*OF assms*, **where** *atends*=*True*]
using *contour_integral_nearby* [*OF assms*, **where** *atends*=*False*]
unfolding *linked_paths_def* **by** *simp_all*

lemma *contour_integral_bound_exists*:
assumes *S*: *open S*
and *g*: *valid_path g*
and *pag*: *path_image g* ⊆ *S*
shows ∃ *L*. *0 < L* ∧
(∀ *f B*. *f holomorphic_on S* ∧ (∀ *z* ∈ *S*. *norm*(*f z*) ≤ *B*)
⟶ *norm*(*contour_integral g f*) ≤ *L*∗*B*)
proof −
have *path g* **using** *g*
by (*simp add*: *valid_path_imp_path*)
then obtain *d*::*real* **and** *p*
where *d*: *0 < d*
and *p*: *polynomial_function p path_image p* ⊆ *S*
and *pi*: ⋀*f*. *f holomorphic_on S* ⟹ *contour_integral g f* = *contour_integral p f*
using *contour_integral_nearby_ends* [*OF S* ‹*path g*› *pag*]
by (*metis cancel_comm_monoid_add_class.diff_cancel g norm_zero path_approx_polynomial_function valid_path_polynomial_function*)
then obtain *p*′ **where** *p*′: *polynomial_function p*′
⋀*x*. (*p has_vector_derivative* (*p*′ *x*)) (*at x*)
by (*blast intro*: *has_vector_derivative_polynomial_function that*)
then have *bounded*(*p*′ ‘ {*0..1*})
using *continuous_on_polymonial_function*
by (*force simp*: *intro*!: *compact_imp_bounded compact_continuous_image*)
then obtain *L* **where** *L*: *L*>*0* **and** *nop*′: ⋀*x*. ⟦*0* ≤ *x*; *x* ≤ *1*⟧ ⟹ *norm* (*p*′ *x*)

```
≤ L
    by (force simp: bounded_pos)
  { fix f B
    assume f: f holomorphic_on S and B: ⋀z. z∈S ⟹ cmod (f z) ≤ B
    then have f contour_integrable_on p ∧ valid_path p
      using p S
    by (blast intro: valid_path_polynomial_function contour_integrable_holomorphic_simple
holomorphic_on_imp_continuous_on)
    moreover have cmod (vector_derivative p (at x)) * cmod (f (p x)) ≤ L * B
if 0 ≤ x x ≤ 1 for x
    proof (rule mult_mono)
      show cmod (vector_derivative p (at x)) ≤ L
        by (metis nop' p'(2) that vector_derivative_at)
      show cmod (f (p x)) ≤ B
        by (metis B atLeastAtMost_iff imageI p(2) path_defs(4) subset_eq that)
    qed (use ‹L>0› in auto)
    ultimately
    have cmod (integral {0..1} (λx. f (p x) * vector_derivative p (at x))) ≤ L * B
      by (intro order_trans [OF integral_norm_bound_integral])
         (auto simp: mult.commute norm_mult contour_integrable_on)
    then have cmod (contour_integral g f) ≤ L * B
      using contour_integral_integral f pi by presburger
  } then
  show ?thesis using ‹L > 0›
    by (intro exI[of _ L]) auto
qed
```

2.8 Homotopy forms of Cauchy's theorem

```
lemma Cauchy_theorem_homotopic:
    assumes hom: if atends then homotopic_paths S g h else homotopic_loops S g
h
        and open S and f: f holomorphic_on S
        and vpg: valid_path g and vph: valid_path h
    shows contour_integral g f = contour_integral h f
proof −
  have pathsf: linked_paths atends g h
    using hom  by (auto simp: linked_paths_def homotopic_paths_imp_pathstart
homotopic_paths_imp_pathfinish homotopic_loops_imp_loop)
  obtain k :: real × real ⇒ complex
    where contk: continuous_on ({0..1} × {0..1}) k
      and ks: k ‘ ({0..1} × {0..1}) ⊆ S
      and k [simp]: ∀x. k (0, x) = g x ∀x. k (1, x) = h x
      and ksf: ∀t∈{0..1}. linked_paths atends g (λx. k (t, x))
        using hom pathsf by (auto simp: linked_paths_def homotopic_paths_def
homotopic_loops_def homotopic_with_def split: if_split_asm)
  have ucontk: uniformly_continuous_on ({0..1} × {0..1}) k
    by (blast intro: compact_Times compact_uniformly_continuous [OF contk])
  { fix t::real assume t: t ∈ {0..1}
```

```
  have Pair t ` {0..1} ⊆ {0..1} × {0..1}
    using t by force
  then have pak: path (k ∘ (λu. (t, u)))
    unfolding path_def
    by (intro continuous_intros continuous_on_subset [OF contk])+
  have pik: path_image (k ∘ Pair t) ⊆ S
    using ks t by (auto simp: path_image_def)
  obtain e where e>0 and e:
      ⋀g h. ⟦valid_path g; valid_path h;
             ∀u∈{0..1}. cmod (g u − (k ∘ Pair t) u) < e ∧ cmod (h u − (k ∘
Pair t) u) < e;
             linked_paths atends g h⟧
            ⟹ contour_integral h f = contour_integral g f
    using contour_integral_nearby [OF ‹open S› pak pik, of atends] f by metis
  obtain d where d>0 and d:
      ⋀x x'. ⟦x ∈ {0..1} × {0..1}; x' ∈ {0..1} × {0..1}; norm (x'−x) < d⟧ ⟹
norm (k x' − k x) < e/4
    by (rule uniformly_continuous_onE [OF ucontk, of e/4]) (auto simp: dist_norm
‹e>0›)
  { fix t1 t2
    assume t1: 0 ≤ t1 t1 ≤ 1 and t2: 0 ≤ t2 t2 ≤ 1 and ltd: |t1 − t| < d |t2
− t| < d
     have no2: norm(g1 − kt) < e if norm(g1 − k1) < e/4 norm(k1 − kt) <
e/4 for g1 k1 kt :: complex
    proof (rule norm_triangle_half_l)
      show cmod (g1 − k1) < e/2 cmod (kt − k1) < e/2
          using ‹e > 0› that by (auto simp: norm_minus_commute intro: or-
der_less_trans)
    qed
    have ∃ d>0. ∀ g1 g2. valid_path g1 ∧ valid_path g2 ∧
                         (∀u∈{0..1}. cmod (g1 u − k (t1, u)) < d ∧ cmod (g2 u − k
(t2, u)) < d) ∧
                         linked_paths atends g1 g2 ⟶
                         contour_integral g2 f = contour_integral g1 f
      using t t1 t2 ltd ‹e > 0›
    by (rule_tac x=e/4 in exI) (auto intro!: e simp: d no2 simp del: less_divide_eq_numeral1)
  }
  then have ∃ e. 0 < e ∧
          (∀t1 t2. t1 ∈ {0..1} ∧ t2 ∈ {0..1} ∧ |t1 − t| < e ∧ |t2 − t| < e
            ⟶ (∃ d. 0 < d ∧
                (∀ g1 g2. valid_path g1 ∧ valid_path g2 ∧
                  (∀ u ∈ {0..1}.
                    norm(g1 u − k((t1,u))) < d ∧ norm(g2 u − k((t2,u))) < d) ∧
                    linked_paths atends g1 g2
                    ⟶ contour_integral g2 f = contour_integral g1 f)))
    by (rule_tac x=d in exI) (simp add: ‹d > 0›)
 }
 then obtain ee where ee:
     ⋀t. t ∈ {0..1} ⟹ ee t > 0 ∧
```

```
        (∀ t1 t2. t1 ∈ {0..1} ⟶ t2 ∈ {0..1} ⟶ |t1 − t| < ee t ⟶ |t2 − t| <
ee t
          ⟶ (∃ d. 0 < d ∧
              (∀ g1 g2. valid_path g1 ∧ valid_path g2 ∧
                (∀ u ∈ {0..1}.
                   norm(g1 u − k((t1,u))) < d ∧ norm(g2 u − k((t2,u))) < d) ∧
                   linked_paths atends g1 g2
                   ⟶ contour_integral g2 f = contour_integral g1 f)))
    by metis
  note ee_rule = ee [THEN conjunct2, rule_format, of 0 0 0]
  define C where C = (λt. ball t (ee t / 3)) ‘ {0..1}
  obtain C' where C': C' ⊆ C finite C' and C'01: {0..1} ⊆ ⋃ C'
  proof (rule compactE [OF compact_interval])
    show {0..1} ⊆ ⋃ C
      using ee [THEN conjunct1] by (auto simp: C_def dist_norm)
  qed (use C_def in auto)
  define kk where kk = {t ∈ {0..1}. ball t (ee t / 3) ∈ C'}
  have kk01: kk ⊆ {0..1} by (auto simp: kk_def)
  define e where e = Min (ee ‘ kk)
  have C'_eq: C' = (λt. ball t (ee t / 3)) ‘ kk
    using C' by (auto simp: kk_def C_def)
  have ee_pos[simp]: ⋀t. t ∈ {0..1} ⟹ ee t > 0
    by (simp add: kk_def ee)
  moreover have finite kk
    using ‹finite C'› kk01 by (force simp: C'_eq inj_on_def ball_eq_ball_iff dest:
ee_pos finite_imageD)
  moreover have kk ≠ {} using ‹{0..1} ⊆ ⋃ C'› C'_eq by force
  ultimately have e > 0
    using finite_less_Inf_iff [of ee ‘ kk 0] kk01 by (force simp: e_def)
  then obtain N::nat where N > 0 and N: 1/N < e/3
   by (meson divide_pos_pos nat_approx_posE zero_less_Suc zero_less_numeral)
  have e_le_ee: ⋀i. i ∈ kk ⟹ e ≤ ee i
    using ‹finite kk› by (simp add: e_def Min_le_iff [of ee ‘ kk])
  have plus: ∃ t ∈ kk. x ∈ ball t (ee t / 3) if x ∈ {0..1} for x
    using C' subsetD [OF C'01 that] unfolding C'_eq by blast
  have [OF order_refl]:
      ∃ d. 0 < d ∧ (∀ j. valid_path j ∧ (∀ u ∈ {0..1}. norm(j u − k (n/N, u)) <
d) ∧ linked_paths atends g j
                    ⟶ contour_integral j f = contour_integral g f)
      if n ≤ N for n
  using that
  proof (induct n)
    case 0 show ?case
      using ee_rule
      by clarsimp (metis diff_self norm_eq_zero vpg)
  next
    case (Suc n)
    then have N01: n/N ∈ {0..1} (Suc n)/N ∈ {0..1}  by auto
    then obtain t where t: t ∈ kk n/N ∈ ball t (ee t / 3)
```

```
    using plus [of n/N] by blast
  then have nN_less: |n/N − t| < ee t
    by (simp add: dist_norm del: less_divide_eq_numeral1)
  have n'N_less: |real (Suc n) / real N − t| < ee t
    using t N ‹N > 0› e_le_ee [of t]
  by (simp add: dist_norm add_divide_distrib abs_diff_less_iff del: less_divide_eq_numeral1)
(simp add: field_simps)
  have t01: t ∈ {0..1} using ‹kk ⊆ {0..1}› ‹t ∈ kk› by blast
  obtain d1 where d1 > 0 and d1:
     ⋀g1 g2. ⟦valid_path g1; valid_path g2;
              ∀u∈{0..1}. cmod (g1 u − k (n/N, u)) < d1 ∧ cmod (g2 u − k
((Suc n) / N, u)) < d1;
              linked_paths atends g1 g2⟧
              ⟹ contour_integral g2 f = contour_integral g1 f
    using ee [THEN conjunct2, rule_format, OF t01 N01 nN_less n'N_less] by
fastforce
  have n ≤ N using Suc.prems by auto
  with Suc.hyps
  obtain d2 where d2 > 0
    and d2: ⋀j. ⟦valid_path j; ∀u∈{0..1}. cmod (j u − k (n/N, u)) < d2;
linked_paths atends g j⟧
                ⟹ contour_integral j f = contour_integral g f
    by auto
  have Pair (n/ N) ‘ {0..1} ⊆ {0..1} × {0..1}
    using N01 by auto
  then have continuous_on {0..1} (k ∘ (λu. (n/N, u)))
    by (intro continuous_intros continuous_on_subset [OF contk])
  then have pkn: path (λu. k (n/N, u))
    by (simp add: path_def)
  have min12: min d1 d2 > 0 by (simp add: ‹0 < d1› ‹0 < d2›)
  obtain p where polynomial_function p
    and psf: pathstart p = pathstart (λu. k (n/N, u))
             pathfinish p = pathfinish (λu. k (n/N, u))
    and pk_le: ⋀t. t∈{0..1} ⟹ cmod (p t − k (n/N, t)) < min d1 d2
    using path_approx_polynomial_function [OF pkn min12] by blast
  then have vpp: valid_path p using valid_path_polynomial_function by blast
  have lpa: linked_paths atends g p
    by (metis (mono_tags, lifting) N01(1) ksf linked_paths_def pathfinish_def
pathstart_def psf)
  show ?case
  proof (intro exI; safe)
    fix j
    assume valid_path j linked_paths atends g j
      and ∀u∈{0..1}. cmod (j u − k (real (Suc n) / real N, u)) < min d1 d2
    then have contour_integral j f = contour_integral p f
     using pk_le N01(1) ksf by (force intro!: vpp d1 simp add: linked_paths_def
psf)
    also have ... = contour_integral g f
      using pk_le by (force intro!: vpp d2 lpa)
```

```
    finally show contour_integral j f = contour_integral g f .
  qed (simp add: ‹0 < d1› ‹0 < d2›)
 qed
 then obtain d where 0 < d
                    ⋀j. valid_path j ∧ (∀ u ∈ {0..1}. norm(j u − k (1,u)) < d) ∧
linked_paths atends g j
                        ⟹ contour_integral j f = contour_integral g f
   using ‹N>0› by auto
 then have linked_paths atends g h ⟹ contour_integral h f = contour_integral
g f
   using ‹N>0› vph by fastforce
 then show ?thesis
   by (simp add: pathsf)
qed

proposition Cauchy_theorem_homotopic_paths:
   assumes hom: homotopic_paths S g h
      and open S and f: f holomorphic_on S
      and vpg: valid_path g and vph: valid_path h
   shows contour_integral g f = contour_integral h f
 using Cauchy_theorem_homotopic [of True S g h] assms by simp

proposition Cauchy_theorem_homotopic_loops:
   assumes hom: homotopic_loops S g h
      and open S and f: f holomorphic_on S
      and vpg: valid_path g and vph: valid_path h
   shows contour_integral g f = contour_integral h f
 using Cauchy_theorem_homotopic [of False S g h] assms by simp

lemma has_contour_integral_newpath:
   ⟦(f has_contour_integral y) h; f contour_integrable_on g; contour_integral g f
= contour_integral h f⟧
    ⟹ (f has_contour_integral y) g
 using has_contour_integral_integral contour_integral_unique by auto

lemma Cauchy_theorem_null_homotopic:
    ⟦f holomorphic_on S; open S; valid_path g; homotopic_loops S g (linepath a
a)⟧
    ⟹ (f has_contour_integral 0) g
 by (metis Cauchy_theorem_homotopic_loops contour_integrable_holomorphic_simple
valid_path_linepath
       contour_integral_trivial has_contour_integral_integral homotopic_loops_imp_subset)

end
```

3 Winding numbers

```
theory Winding_Numbers
 imports Cauchy_Integral_Theorem
```

begin

3.1 Definition

definition *winding_number_prop* :: [*real* ⇒ *complex*, *complex*, *real*, *real* ⇒ *complex*, *complex*] ⇒ *bool* **where**
winding_number_prop γ *z e p n* ≡
valid_path p ∧ *z* ∉ *path_image p* ∧
pathstart p = *pathstart* γ ∧
pathfinish p = *pathfinish* γ ∧
(∀ *t* ∈ {*0..1*}. *norm*(γ *t* − *p t*) < *e*) ∧
contour_integral p (λ*w*. *1*/(*w* − *z*)) = *2* ∗ *pi* ∗ i ∗ *n*

definition *winding_number*:: [*real* ⇒ *complex*, *complex*] ⇒ *complex* **where**
winding_number γ *z* ≡ *SOME n*. ∀ *e* > *0*. ∃ *p*. *winding_number_prop* γ *z e p n*

lemma *winding_number*:
assumes *path* γ *z* ∉ *path_image* γ *0* < *e*
shows ∃ *p*. *winding_number_prop* γ *z e p* (*winding_number* γ *z*)
proof −
have *path_image* γ ⊆ *UNIV* − {*z*}
using *assms* **by** *blast*
then obtain *d*
where *d*: *d*>*0*
and *pi_eq*: ⋀*h1 h2*. *valid_path h1* ∧ *valid_path h2* ∧
(∀ *t*∈{*0..1*}. *cmod* (*h1 t* − γ *t*) < *d* ∧ *cmod* (*h2 t* − γ *t*) < *d*) ∧
pathstart h2 = *pathstart h1* ∧ *pathfinish h2* = *pathfinish h1* ⟶
path_image h1 ⊆ *UNIV* − {*z*} ∧ *path_image h2* ⊆ *UNIV* − {*z*} ∧
(∀ *f*. *f holomorphic_on UNIV* − {*z*} ⟶ *contour_integral h2 f* = *contour_integral h1 f*)
using *contour_integral_nearby_ends* [*of UNIV* − {*z*} γ] *assms* **by** (*auto simp*: *open_delete*)
then obtain *h* **where** *h*: *polynomial_function h* ∧ *pathstart h* = *pathstart* γ ∧ *pathfinish h* = *pathfinish* γ ∧
(∀ *t* ∈ {*0..1*}. *norm*(*h t* − γ *t*) < *d*/*2*)
using *path_approx_polynomial_function* [*OF* ‹*path* γ›, *of d*/*2*] *d* **by** (*metis half_gt_zero_iff*)
define *nn* **where** *nn* = *1*/(*2*∗ *pi*∗i) ∗ *contour_integral h* (λ*w*. *1*/(*w* − *z*))
have ∃ *n*. ∀ *e* > *0*. ∃ *p*. *winding_number_prop* γ *z e p n*
proof (*rule_tac x*=*nn* **in** *exI*, *clarify*)
fix *e*::*real*
assume *e*: *e*>*0*
obtain *p* **where** *p*: *polynomial_function p* ∧
pathstart p = *pathstart* γ ∧ *pathfinish p* = *pathfinish* γ ∧ (∀ *t*∈{*0..1*}. *cmod* (*p t* − γ *t*) < *min e* (*d*/*2*))
using *path_approx_polynomial_function* [*OF* ‹*path* γ›, *of min e* (*d*/*2*)] *d* ‹*0*<*e*›

```
      by (metis min_less_iff_conj zero_less_divide_iff zero_less_numeral)
    have (λw. 1 / (w − z)) holomorphic_on UNIV − {z}
      by (auto simp: intro!: holomorphic_intros)
    then have winding_number_prop γ z e p nn
      using pi_eq [of h p] h p d
      by (auto simp: valid_path_polynomial_function norm_minus_commute
nn_def winding_number_prop_def)
    then show ∃p. winding_number_prop γ z e p nn
      by metis
  qed
 then show ?thesis
  unfolding winding_number_def by (rule someI2_ex) (blast intro: ‹0<e›)
qed

lemma winding_number_unique:
 assumes γ: path γ z ∉ path_image γ
    and pi: ⋀e. e>0 ⟹ ∃p. winding_number_prop γ z e p n
  shows winding_number γ z = n
proof −
 have path_image γ ⊆ UNIV − {z}
  using assms by blast
 then obtain e
  where e: e>0
    and pi_eq: ⋀h1 h2 f. ⟦valid_path h1; valid_path h2;
                (∀t∈{0..1}. cmod (h1 t − γ t) < e ∧ cmod (h2 t − γ t) < e);
                   pathstart h2 = pathstart h1; pathfinish h2 = pathfinish h1; f
holomorphic_on UNIV − {z}⟧ ⟹
                contour_integral h2 f = contour_integral h1 f
  using contour_integral_nearby_ends [of UNIV − {z} γ] assms by (auto simp:
open_delete)
 obtain p where p: winding_number_prop γ z e p n
  using pi [OF e] by blast
 obtain q where q: winding_number_prop γ z e q (winding_number γ z)
  using winding_number [OF γ e] by blast
 have 2 * complex_of_real pi * i * n = contour_integral p (λw. 1 / (w − z))
  using p by (auto simp: winding_number_prop_def)
 also have ... = contour_integral q (λw. 1 / (w − z))
 proof (rule pi_eq)
  show (λw. 1 / (w − z)) holomorphic_on UNIV − {z}
   by (auto intro!: holomorphic_intros)
 qed (use p q in ‹auto simp: winding_number_prop_def norm_minus_commute›)
 also have ... = 2 * complex_of_real pi * i * winding_number γ z
  using q by (auto simp: winding_number_prop_def)
 finally have 2 * complex_of_real pi * i * n = 2 * complex_of_real pi * i *
winding_number γ z .
 then show ?thesis
  by simp
qed
```

lemma *winding_number_unique_loop*:
assumes γ: *path* γ *z* $\notin$ *path_image* γ
and *loop*: *pathfinish* γ = *pathstart* γ
and *pi*:
$\bigwedge e.\ e{>}0 \Longrightarrow \exists p.\ \mathit{valid\_path}\ p \wedge z \notin \mathit{path\_image}\ p \wedge$
pathfinish p = pathstart p $\wedge$
$(\forall t \in \{0..1\}.\ \mathit{norm}\ (\gamma\ t - p\ t) < e) \wedge$
contour_integral p $(\lambda w.\ 1/(w - z)) = 2 * \mathit{pi} * \mathrm{i} * n$
shows *winding_number* γ *z* = *n*
proof −
have *path_image* $\gamma \subseteq$ *UNIV* − $\{z\}$
using *assms* **by** *blast*
then obtain *e*
where *e*: *e>0*
and *pi_eq*: $\bigwedge h1\ h2\ f.\ [\![\mathit{valid\_path}\ h1;\ \mathit{valid\_path}\ h2;$
$(\forall t{\in}\{0..1\}.\ \mathit{cmod}\ (h1\ t - \gamma\ t) < e \wedge \mathit{cmod}\ (h2\ t - \gamma\ t) < e);$
pathfinish h1 = pathstart h1; *pathfinish h2 = pathstart h2*; *f holomorphic_on UNIV* − $\{z\}$$]\!] \Longrightarrow$
contour_integral h2 f = contour_integral h1 f
using *contour_integral_nearby_loops* [*of UNIV* − $\{z\}$ γ] *assms* **by** (*auto simp*: *open_delete*)
obtain *p* **where** *p*:
valid_path p $\wedge$ *z* $\notin$ *path_image p* $\wedge$ *pathfinish p = pathstart p* $\wedge$
$(\forall t \in \{0..1\}.\ \mathit{norm}\ (\gamma\ t - p\ t) < e) \wedge$
contour_integral p $(\lambda w.\ 1/(w - z)) = 2 * \mathit{pi} * \mathrm{i} * n$
using *pi* [*OF e*] **by** *blast*
obtain *q* **where** *q*: *winding_number_prop* γ *z e q* (*winding_number* γ *z*)
using *winding_number* [*OF* γ *e*] **by** *blast*
have *2 * complex_of_real pi * i * n = contour_integral p* $(\lambda w.\ 1\ /\ (w - z))$
using *p* **by** *auto*
also have ... = *contour_integral q* $(\lambda w.\ 1\ /\ (w - z))$
proof (*rule pi_eq*)
show $(\lambda w.\ 1\ /\ (w - z))$ *holomorphic_on UNIV* − $\{z\}$
by (*auto intro*!: *holomorphic_intros*)
qed (*use p q loop* **in** ‹*auto simp*: *winding_number_prop_def norm_minus_commute*›)
also have ... = *2 * complex_of_real pi * i * winding_number* γ *z*
using *q* **by** (*auto simp*: *winding_number_prop_def*)
finally have *2 * complex_of_real pi * i * n = 2 * complex_of_real pi * i * winding_number* γ *z* **.**
then show *?thesis*
by *simp*
qed

proposition *winding_number_valid_path*:
assumes *valid_path* γ *z* $\notin$ *path_image* γ
shows *winding_number* γ *z* = *1*/(*2**pi**i) * *contour_integral* γ $(\lambda w.\ 1/(w - z))$
by (*rule winding_number_unique*)
(*use assms* **in** ‹*auto simp*: *valid_path_imp_path winding_number_prop_def*›)

proposition *has_contour_integral_winding_number*:
assumes γ: *valid_path* γ $z \notin$ *path_image* γ
shows $((\lambda w.\ 1/(w - z))$ *has_contour_integral* $(2*pi*\mathrm{i}*winding\_number\ \gamma\ z))$ γ
by (*simp add*: *winding_number_valid_path has_contour_integral_integral contour_integrable_inversediff assms*)

lemma *winding_number_trivial* [*simp*]: $z \neq a \Longrightarrow$ *winding_number*(*linepath a a*) $z = 0$
by (*simp add*: *winding_number_valid_path*)

lemma *winding_number_subpath_trivial* [*simp*]: $z \neq g\ x \Longrightarrow$ *winding_number* (*subpath x x g*) $z = 0$
by (*simp add*: *path_image_subpath winding_number_valid_path*)

lemma *winding_number_join*:
assumes $\gamma 1$: *path* $\gamma 1$ $z \notin$ *path_image* $\gamma 1$
and $\gamma 2$: *path* $\gamma 2$ $z \notin$ *path_image* $\gamma 2$
and *pathfinish* $\gamma 1$ = *pathstart* $\gamma 2$
shows *winding_number*($\gamma 1$ +++ $\gamma 2$) z = *winding_number* $\gamma 1$ z + *winding_number* $\gamma 2$ z
proof (*rule winding_number_unique*)
show $\exists p.$ *winding_number_prop* ($\gamma 1$ +++ $\gamma 2$) z e p
(*winding_number* $\gamma 1$ z + *winding_number* $\gamma 2$ z) **if** $e > 0$ **for** e
proof −
obtain *p1* **where** *winding_number_prop* $\gamma 1$ z e *p1* (*winding_number* $\gamma 1$ z)
using ‹$0 < e$› $\gamma 1$ *winding_number* **by** *blast*
moreover
obtain *p2* **where** *winding_number_prop* $\gamma 2$ z e *p2* (*winding_number* $\gamma 2$ z)
using ‹$0 < e$› $\gamma 2$ *winding_number* **by** *blast*
ultimately
have *winding_number_prop* ($\gamma 1$+++$\gamma 2$) z e (*p1*+++*p2*) (*winding_number* $\gamma 1$ z + *winding_number* $\gamma 2$ z)
using *assms*
apply (*simp add*: *winding_number_prop_def not_in_path_image_join contour_integrable_inversediff algebra_simps*)
apply (*auto simp*: *joinpaths_def*)
done
then show *?thesis*
by *blast*
qed
qed (*use assms* **in** ‹*auto simp*: *not_in_path_image_join*›)

lemma *winding_number_reversepath*:
assumes *path* γ $z \notin$ *path_image* γ
shows *winding_number*(*reversepath* γ) $z = -$ (*winding_number* γ z)
proof (*rule winding_number_unique*)
show $\exists p.$ *winding_number_prop* (*reversepath* γ) z e p ($-$ *winding_number* γ z)

if $e > 0$ **for** e
proof −
obtain p **where** *winding_number_prop* γ z e p (*winding_number* γ z)
using ‹$0 < e$› *assms winding_number* **by** *blast*
then have *winding_number_prop* (*reversepath* γ) z e (*reversepath* p) (− *winding_number* γ z)
using *assms* **unfolding** *winding_number_prop_def*
apply (*simp add*: *contour_integral_reversepath contour_integrable_inversediff valid_path_imp_reverse*)
apply (*auto simp*: *reversepath_def*)
done
then show *?thesis*
by *blast*
qed
qed (*use assms* **in** *auto*)

lemma *winding_number_shiftpath*:
assumes γ: *path* γ $z \notin$ *path_image* γ
and *pathfinish* γ = *pathstart* γ $a \in \{0..1\}$
shows *winding_number*(*shiftpath* a γ) z = *winding_number* γ z
proof (*rule winding_number_unique_loop*)
show $\exists p.$ *valid_path* $p \wedge z \notin$ *path_image* $p \wedge$ *pathfinish* p = *pathstart* $p \wedge$
$(\forall t \in \{0..1\}.$ *cmod* (*shiftpath* a γ t − p t) < $e) \wedge$
contour_integral p ($\lambda w.$ 1 / (w − z)) =
2 ∗ *pi* ∗ i ∗ *winding_number* γ z
if $e > 0$ **for** e
proof −
obtain p **where** *winding_number_prop* γ z e p (*winding_number* γ z)
using ‹$0 < e$› *assms winding_number* **by** *blast*
then show *?thesis*
apply (*rule_tac x*=*shiftpath a p* **in** *exI*)
using *assms that*
apply (*auto simp*: *winding_number_prop_def path_image_shiftpath pathfinish_shiftpath pathstart_shiftpath contour_integral_shiftpath*)
apply (*simp add*: *shiftpath_def*)
done
qed
qed (*use assms* **in** ‹*auto simp*: *path_shiftpath path_image_shiftpath pathfinish_shiftpath pathstart_shiftpath*›)

lemma *winding_number_split_linepath*:
assumes $c \in$ *closed_segment* a b $z \notin$ *closed_segment* a b
shows *winding_number*(*linepath* a b) z = *winding_number*(*linepath* a c) z + *winding_number*(*linepath* c b) z
proof −
have $z \notin$ *closed_segment* a c $z \notin$ *closed_segment* c b
using *assms* **by** (*meson convex_contains_segment convex_segment ends_in_segment subsetCE*)+
then show *?thesis*

using *assms*
by (*simp add*: *winding_number_valid_path contour_integral_split_linepath* [*symmetric*] *continuous_on_inversediff field_simps*)
qed

lemma *winding_number_cong*:
(⋀*t*. ⟦*0* ≤ *t*; *t* ≤ *1*⟧ ⟹ *p t* = *q t*) ⟹ *winding_number p z* = *winding_number q z*
by (*simp add*: *winding_number_def winding_number_prop_def pathstart_def pathfinish_def*)

lemma *winding_number_constI*:
assumes *c*≠*z* **and** *g*: ⋀*t*. ⟦*0*≤*t*; *t*≤*1*⟧ ⟹ *g t* = *c*
shows *winding_number g z* = *0*
proof −
have *winding_number g z* = *winding_number* (*linepath c c*) *z*
using *g winding_number_cong* **by** *fastforce*
moreover have *winding_number* (*linepath c c*) *z* = *0*
using ‹*c*≠*z*› **by** *auto*
ultimately show *?thesis* **by** *auto*
qed

lemma *winding_number_offset*: *winding_number p z* = *winding_number* (λ*w*. *p w* − *z*) *0*
unfolding *winding_number_def*
proof (*intro ext arg_cong* [**where** *f* = *Eps*] *arg_cong* [**where** *f* = *All*] *imp_cong refl*, *safe*)
fix *n e g*
assume *0* < *e* **and** *g*: *winding_number_prop p z e g n*
then show ∃*r*. *winding_number_prop* (λ*w*. *p w* − *z*) *0 e r n*
by (*rule_tac x*=λ*t*. *g t* − *z* **in** *exI*)
(*force simp*: *winding_number_prop_def contour_integral_integral valid_path_def path_defs*
vector_derivative_def has_vector_derivative_diff_const piecewise_C1_differentiable_diff C1_differentiable_imp_piecewise)
next
fix *n e g*
assume *0* < *e* **and** *g*: *winding_number_prop* (λ*w*. *p w* − *z*) *0 e g n*
then have *winding_number_prop p z e* (λ*t*. *g t* + *z*) *n*
apply (*simp add*: *winding_number_prop_def contour_integral_integral valid_path_def path_defs*
piecewise_C1_differentiable_add vector_derivative_def has_vector_derivative_add_const C1_differentiable_imp_piecewise)
apply (*force simp*: *algebra_simps*)
done
then show ∃*r*. *winding_number_prop p z e r n*
by *metis*
qed

lemma *winding_number_negatepath*:
assumes γ: *valid_path* γ **and** *0*: $0 \notin$ *path_image* γ
shows *winding_number*(*uminus* $\circ$ γ) *0* = *winding_number* γ *0*
proof −
have (/) *1 contour_integrable_on* γ
using *0* γ *contour_integrable_inversediff* **by** *fastforce*
then have ((λ*z*. *1*/*z*) *has_contour_integral contour_integral* γ ((/) *1*)) γ
by (*rule has_contour_integral_integral*)
then have ((λ*z*. *1* / − *z*) *has_contour_integral* − *contour_integral* γ ((/) *1*)) γ
using *has_contour_integral_neg* **by** *auto*
then have *contour_integral* (*uminus* $\circ$ γ) ((/) *1*) =
contour_integral γ ((/) *1*)
using γ **by** (*simp add*: *contour_integral_unique has_contour_integral_negatepath*)
then show *?thesis*
using *assms* **by** (*simp add*: *winding_number_valid_path valid_path_negatepath image_def path_defs*)
qed

A combined theorem deducing several things piecewise.

lemma *winding_number_join_pos_combined*:
⟦*valid_path* γ*1*; *z* $\notin$ *path_image* γ*1*; *0* < *Re*(*winding_number* γ*1 z*);
valid_path γ*2*; *z* $\notin$ *path_image* γ*2*; *0* < *Re*(*winding_number* γ*2 z*); *pathfinish* γ*1* = *pathstart* γ*2*⟧
⟹ *valid_path*(γ*1* +++ γ*2*) ∧ *z* $\notin$ *path_image*(γ*1* +++ γ*2*) ∧ *0* < *Re*(*winding_number*(γ*1* +++ γ*2*) *z*)
by (*simp add*: *valid_path_join path_image_join winding_number_join valid_path_imp_path*)

3.1.1 Useful sufficient conditions for the winding number to be positive

lemma *Re_winding_number*:
⟦*valid_path* γ; *z* $\notin$ *path_image* γ⟧
⟹ *Re*(*winding_number* γ *z*) = *Im*(*contour_integral* γ (λ*w*. *1*/(*w* − *z*))) / (*2*∗*pi*)
by (*simp add*: *winding_number_valid_path field_simps Re_divide power2_eq_square*)

lemma *winding_number_pos_le*:
assumes γ: *valid_path* γ *z* $\notin$ *path_image* γ
and *ge*: ⋀*x*. ⟦*0* < *x*; *x* < *1*⟧ ⟹ *0* ≤ *Im* (*vector_derivative* γ (*at x*) ∗ *cnj*(γ *x* − *z*))
shows *0* ≤ *Re*(*winding_number* γ *z*)
proof −
have *ge0*: *0* ≤ *Im* (*vector_derivative* γ (*at x*) / (γ *x* − *z*)) **if** *x*: *0* < *x x* < *1* **for** *x*
using *ge* **by** (*simp add*: *Complex.Im_divide algebra_simps x*)
let *?vd* = λ*x*. *1* / (γ *x* − *z*) ∗ *vector_derivative* γ (*at x*)
let *?int* = λ*z*. *contour_integral* γ (λ*w*. *1* / (*w* − *z*))
have *0* ≤ *Im* (*?int z*)

```
  proof (rule has_integral_component_nonneg [of i, simplified])
    show ⋀x. x ∈ cbox 0 1 ⟹ 0 ≤ Im (if 0 < x ∧ x < 1 then ?vd x else 0)
      by (force simp: ge0)
    have ((λa. 1 / (a − z)) has_contour_integral contour_integral γ (λw. 1 / (w
− z))) γ
     using γ by (simp flip: add: contour_integrable_inversediff has_contour_integral_integral)
    then have hi: (?vd has_integral ?int z) (cbox 0 1)
      using has_contour_integral by auto
    show ((λx. if 0 < x ∧ x < 1 then ?vd x else 0) has_integral ?int z) (cbox 0 1)
      by (rule has_integral_spike_interior [OF hi]) simp
  qed
  then show ?thesis
    by (simp add: Re_winding_number [OF γ] field_simps)
qed

lemma winding_number_pos_lt_lemma:
  assumes γ: valid_path γ z ∉ path_image γ
      and e: 0 < e
      and ge: ⋀x. ⟦0 < x; x < 1⟧ ⟹ e ≤ Im (vector_derivative γ (at x) / (γ x
− z))
    shows 0 < Re(winding_number γ z)
proof −
  let ?vd = λx. 1 / (γ x − z) ∗ vector_derivative γ (at x)
  let ?int = λz. contour_integral γ (λw. 1 / (w − z))
  have e ≤ Im (contour_integral γ (λw. 1 / (w − z)))
  proof (rule has_integral_component_le [of i λx. i∗e i∗e {0..1}, simplified])
    have ((λa. 1 / (a − z)) has_contour_integral contour_integral γ (λw. 1 / (w
− z))) γ
      thm has_integral_component_le [of i λx. i∗e i∗e {0..1}, simplified]
     using γ by (simp add: contour_integrable_inversediff has_contour_integral_integral)
    then have hi: (?vd has_integral ?int z) (cbox 0 1)
      using has_contour_integral by auto
    show ((λx. if 0 < x ∧ x < 1 then ?vd x else i ∗ e) has_integral ?int z) {0..1}
      by (rule has_integral_spike_interior [OF hi, simplified box_real]) (use e in
simp)
    show ⋀x. 0 ≤ x ∧ x ≤ 1 ⟹ e ≤ Im (if 0 < x ∧ x < 1 then ?vd x else i ∗ e)
      by (simp add: ge)
  qed (use has_integral_const_real [of _ 0 1] in auto)
  with e show ?thesis
    by (simp add: Re_winding_number [OF γ] field_simps)
qed

lemma winding_number_pos_lt:
  assumes γ: valid_path γ z ∉ path_image γ
      and e: 0 < e
      and ge: ⋀x. ⟦0 < x; x < 1⟧ ⟹ e ≤ Im (vector_derivative γ (at x) ∗ cnj(γ
x − z))
    shows 0 < Re (winding_number γ z)
proof −
```

have *bm*: *bounded* ((λ*w*. *w* − *z*) ‘ (*path_image* γ))
using *bounded_translation* [*of* _ −*z*] γ **by** (*simp add*: *bounded_valid_path_image*)
then obtain *B* **where** *B*: *B* > *0* **and** *Bno*: ⋀*x*. *x* ∈ (λ*w*. *w* − *z*) ‘ (*path_image* γ) ⟹ *norm x* ≤ *B*
using *bounded_pos* [*THEN iffD1*, *OF bm*] **by** *blast*
{ fix *x*::*real* **assume** *x*: *0* < *x x* < *1*
then have *B2*: *cmod* (γ *x* − *z*)^2 ≤ *B*^2 **using** *Bno* [*of* γ *x* − *z*]
by (*simp add*: *path_image_def power2_eq_square mult_mono'*)
with *x* **have** γ *x* ≠ *z* **using** γ
using *path_image_def* **by** *fastforce*
then have *e* / *B*^2 ≤ *e* / (*cmod* (γ *x* − *z*))^2
using *B B2 e* **by** (*auto simp*: *divide_left_mono*)
also have ... ≤ *Im* (*vector_derivative* γ (*at x*) ∗ *cnj* (γ *x* − *z*)) / (*cmod* (γ *x* − *z*))^2
using *ge* [*OF x*] **by** (*auto simp*: *divide_right_mono*)
finally have *e* / *B*^2 ≤ *Im* (*vector_derivative* γ (*at x*) ∗ *cnj* (γ *x* − *z*)) / (*cmod* (γ *x* − *z*))^2 **.**
then have *e* / *B*^2 ≤ *Im* (*vector_derivative* γ (*at x*) / (γ *x* − *z*))
by (*simp add*: *complex_div_cnj* [*of* _ γ *x* − *z* **for** *x*] *del*: *complex_cnj_diff times_complex.sel*)
} note ∗ = *this*
show *?thesis*
using *e B* **by** (*simp add*: ∗ *winding_number_pos_lt_lemma* [*OF* γ, *of e*/*B*^2])
qed

3.2 The winding number is an integer

Proof from the book Complex Analysis by Lars V. Ahlfors, Chapter 4, section 2.1, Also on page 134 of Serge Lang's book with the name title, etc.

lemma *exp_fg*:
fixes *z*::*complex*
assumes *g*: (*g has_vector_derivative g'*) (*at x within s*)
and *f*: (*f has_vector_derivative* (*g'* / (*g x* − *z*))) (*at x within s*)
and *z*: *g x* ≠ *z*
shows ((λ*x*. *exp*(−*f x*) ∗ (*g x* − *z*)) *has_vector_derivative 0*) (*at x within s*)
proof −
have ∗: (*exp* ∘ (λ*x*. (− *f x*)) *has_vector_derivative* − (*g'* / (*g x* − *z*)) ∗ *exp* (− *f x*)) (*at x within s*)
using *assms* **unfolding** *has_vector_derivative_def scaleR_conv_of_real*
by (*auto intro*!: *derivative_eq_intros*)
show *?thesis*
using *z* **by** (*auto intro*!: *derivative_eq_intros* ∗ [*unfolded o_def*] *g*)
qed

lemma *winding_number_exp_integral*:
fixes *z*::*complex*
assumes γ: γ *piecewise_C1_differentiable_on* {*a*..*b*}
and *ab*: *a* ≤ *b*
and *z*: *z* ∉ γ ‘ {*a*..*b*}

shows ($\lambda x.$ *vector_derivative* γ (*at* x) / (γ x − z)) *integrable_on* {a..b}
(**is** *?thesis1*)
exp (− (*integral* {a..b} ($\lambda x.$ *vector_derivative* γ (*at* x) / (γ x − z)))) ∗ (γ b − z) = γ a − z
(**is** *?thesis2*)
proof −
let *?D*γ = $\lambda x.$ *vector_derivative* γ (*at* x)
have [*simp*]: $\bigwedge x.$ $a \le x \Longrightarrow x \le b \Longrightarrow \gamma\ x \neq z$
using z **by** *force*
have *con_g*: *continuous_on* {a..b} γ
using γ **by** (*simp add*: *piecewise_C1_differentiable_on_def*)
obtain k **where** *fink*: *finite* k **and** *g_C1_diff*: γ *C1_differentiable_on* ({a..b} − k)
using γ **by** (*force simp*: *piecewise_C1_differentiable_on_def*)
have ○: *open* ({a<..<b} − k)
using ‹*finite* k› **by** (*simp add*: *finite_imp_closed open_Diff*)
moreover have {a<..<b} − k ⊆ {a..b} − k
by *force*
ultimately have *g_diff_at*: $\bigwedge x.$ ⟦$x \notin k$; $x \in$ {a<..<b}⟧ ⟹ γ *differentiable at* x
by (*metis Diff_iff differentiable_on_subset C1_diff_imp_diff* [*OF g_C1_diff*] *differentiable_on_def at_within_open*)
{ fix w
assume $w \neq z$
have *continuous_on* (*ball* w (*cmod* (w − z))) ($\lambda w.$ 1 / (w − z))
by (*auto simp*: *dist_norm intro*!: *continuous_intros*)
moreover have $\bigwedge x.$ *cmod* (w − x) < *cmod* (w − z) ⟹ $\exists f'.$ (($\lambda w.$ 1 / (w − z)) *has_field_derivative* f') (*at* x)
by (*auto simp*: *intro*!: *derivative_eq_intros*)
ultimately have $\exists h.$ $\forall y.$ *norm*(y − w) < *norm*(w − z) ⟶ (h *has_field_derivative* 1/(y − z)) (*at* y)
using *holomorphic_convex_primitive* [*of ball* w (*norm*(w − z)) {} $\lambda w.$ 1/(w − z)]
by (*force simp*: *field_differentiable_def Ball_def dist_norm at_within_open_NO_MATCH norm_minus_commute*)
}
then obtain h **where** h: $\bigwedge w\ y.$ $w \neq z \Longrightarrow$ *norm*(y − w) < *norm*(w − z) ⟹ (h w *has_field_derivative* 1/(y − z)) (*at* y)
by *meson*
have *exy*: $\exists y.$ (($\lambda x.$ *inverse* (γ x − z) ∗ *?D*γ x) *has_integral* y) {a..b}
unfolding *integrable_on_def* [*symmetric*]
proof (*rule contour_integral_local_primitive_any* [*OF piecewise_C1_imp_differentiable* [*OF* γ]])
show $\exists d\ h.$ 0 < d ∧
($\forall y.$ *cmod* (y − w) < d ⟶ (h *has_field_derivative inverse* (y − z))(*at* y *within* − {z}))
if $w \in -\{z\}$ **for** w
using *that inverse_eq_divide has_field_derivative_at_within h*
by (*metis Compl_insert DiffD2 insertCI right_minus_eq zero_less_norm_iff*)

qed *simp*
have *vg_int*: $(\lambda x.\ ?D\gamma\ x\ /\ (\gamma\ x - z))$ *integrable_on* $\{a..b\}$
unfolding *box_real* [*symmetric*] *divide_inverse_commute*
by (*auto intro*!: *exy integrable_subinterval simp add*: *integrable_on_def ab*)
with *ab* **show** *?thesis1*
by (*simp add*: *divide_inverse_commute integral_def integrable_on_def*)
{ fix *t*
assume *t*: $t \in \{a..b\}$
have *cball*: *continuous_on* (*ball* (γ *t*) (*dist* (γ *t*) *z*)) ($\lambda x.$ *inverse* ($x - z$))
using *z* **by** (*auto intro*!: *continuous_intros simp*: *dist_norm*)
have *icd*: $\bigwedge x.$ *cmod* (γ *t* − *x*) < *cmod* (γ *t* − *z*) ⟹ ($\lambda w.$ *inverse* ($w - z$)) *field_differentiable at x*
unfolding *field_differentiable_def* **by** (*force simp*: *intro*!: *derivative_eq_intros*)
obtain *h* **where** *h*: $\bigwedge x.$ *cmod* (γ *t* − *x*) < *cmod* (γ *t* − *z*) ⟹
(*h has_field_derivative inverse* ($x - z$)) (*at x within* {*y*. *cmod* (γ *t* − *y*) < *cmod* (γ *t* − *z*)})
using *holomorphic_convex_primitive* [**where** $f = \lambda w.$ *inverse*($w - z$), *OF convex_ball finite.emptyI cball icd*]
by *simp* (*auto simp*: *ball_def dist_norm that*)
have *exp* (− (*integral* {*a*..*t*} ($\lambda x.$ *?D*γ *x* / (γ *x* − *z*)))) * (γ *t* − *z*) =γ *a* − *z*
proof (*rule has_derivative_zero_unique_strong_interval* [*of* {*a*,*b*} ∪ *k a b*])
show *continuous_on* {*a*..*b*} ($\lambda b.$ *exp* (− *integral* {*a*..*b*} ($\lambda x.$ *?D*γ *x* / (γ *x* − *z*))) * (γ *b* − *z*))
by (*auto intro*!: *continuous_intros con_g indefinite_integral_continuous_1* [*OF vg_int*])
show (($\lambda b.$ *exp* (− *integral* {*a*..*b*} ($\lambda x.$ *?D*γ *x* / (γ *x* − *z*))) * (γ *b* − *z*)) *has_derivative* ($\lambda h.$ *0*))
(*at x within* {*a*..*b*})
if $x \in \{a..b\} - (\{a, b\} \cup k)$ **for** *x*
proof −
have *x*: $x \notin k$ $a < x$ $x < b$
using *that* **by** *auto*
then have $x \in$ *interior* ($\{a..b\} - k$)
using *open_subset_interior* [*OF* ○] **by** *fastforce*
then have *con*: *isCont ?D*γ *x*
using *g_C1_diff x* **by** (*auto simp*: *C1_differentiable_on_eq intro*: *continuous_on_interior*)
then have *con_vd*: *continuous* (*at x within* {*a*..*b*}) ($\lambda x.$ *?D*γ *x*)
by (*rule continuous_at_imp_continuous_within*)
have *gdx*: γ *differentiable at x*
using *x* **by** (*simp add*: *g_diff_at*)
then obtain *d* **where** *d*: (γ *has_derivative* ($\lambda x.$ *x* $*_R$ *d*)) (*at x*)
by (*auto simp add*: *differentiable_iff_scaleR*)
show (($\lambda c.$ *exp* (− *integral* {*a*..*c*} ($\lambda x.$ *?D*γ *x* / (γ *x* − *z*))) * (γ *c* − *z*)) *has_derivative* ($\lambda h.$ *0*))
(*at x within* {*a*..*b*})
proof (*rule exp_fg* [*unfolded has_vector_derivative_def*, *simplified*])
show (γ *has_derivative* ($\lambda c.$ *c* $*_R$ *d*)) (*at x within* {*a*..*b*})
using *d* **by** (*blast intro*: *has_derivative_at_withinI*)

```
      have ((λx. integral {a..x} (λx. ?Dγ x / (γ x − z))) has_vector_derivative
d / (γ x − z))
           (at x within {a..b})
     proof (rule has_vector_derivative_eq_rhs [OF integral_has_vector_derivative_continuous_at
[where S = {}, simplified]])
       show continuous (at x within {a..b}) (λx. vector_derivative γ (at x) /
(γ x − z))
        using continuous_at_imp_continuous_at_within differentiable_imp_continuous_within
gdx x
          by (intro con_vd continuous_intros) (force+)
        show vector_derivative γ (at x) / (γ x − z) = d / (γ x − z)
          using d vector_derivative_at
          by (simp add: vector_derivative_at has_vector_derivative_def)
       qed (use x vg_int in auto)
       then show ((λx. integral {a..x} (λx. ?Dγ x / (γ x − z))) has_derivative
(λc. c *R (d / (γ x − z))))
            (at x within {a..b})
        by (auto simp: has_vector_derivative_def)
     qed (use x in auto)
   qed
  qed (use fink t in auto)
 }
 with ab show ?thesis2
  by (simp add: divide_inverse_commute integral_def)
qed

lemma winding_number_exp_2pi:
  ⟦path p; z ∉ path_image p⟧
   ⟹ pathfinish p − z = exp (2 * pi * i * winding_number p z) * (pathstart p
− z)
using winding_number [of p z 1] unfolding valid_path_def path_image_def path-
start_def pathfinish_def winding_number_prop_def
 by (force dest: winding_number_exp_integral(2) [of _ 0 1 z] simp: field_simps
contour_integral_integral exp_minus)

lemma integer_winding_number_eq:
 assumes γ: path γ and z: z ∉ path_image γ
 shows winding_number γ z ∈ ℤ ⟷ pathfinish γ = pathstart γ
proof −
 obtain p where p: valid_path p z ∉ path_image p
                   pathstart p = pathstart γ pathfinish p = pathfinish γ
        and eq: contour_integral p (λw. 1 / (w − z)) = complex_of_real (2 *
pi) * i * winding_number γ z
  using winding_number [OF assms, of 1] unfolding winding_number_prop_def
by auto
 then have wneq: winding_number γ z = winding_number p z
    using eq winding_number_valid_path by force
 have iff: (winding_number γ z ∈ ℤ) ⟷ (exp (contour_integral p (λw. 1 / (w
− z))) = 1)
```

```
    using eq by (simp add: exp_eq_1 complex_is_Int_iff)
  have γ 0 ≠ z
    by (metis pathstart_def pathstart_in_path_image z)
  then have exp (contour_integral p (λw. 1 / (w − z))) = (γ 1 − z) / (γ 0 − z)
    using p winding_number_exp_integral(2) [of p 0 1 z]
    by (simp add: valid_path_def path_defs contour_integral_integral exp_minus
field_split_simps)
  then have winding_number p z ∈ ℤ ⟷ pathfinish p = pathstart p
    using p wneq iff by (auto simp: path_defs)
  then show ?thesis using p eq
    by (auto simp: winding_number_valid_path)
qed

theorem integer_winding_number:
  ⟦path γ; pathfinish γ = pathstart γ; z ∉ path_image γ⟧ ⟹ winding_number γ
z ∈ ℤ
by (metis integer_winding_number_eq)
```

If the winding number's magnitude is at least one, then the path must contain points in every direction.*) We can thus bound the winding number of a path that doesn't intersect a given ray.

```
lemma winding_number_pos_meets:
  fixes z::complex
  assumes γ: valid_path γ and z: z ∉ path_image γ and 1: Re (winding_number
γ z) ≥ 1
      and w: w ≠ z
  shows ∃a::real. 0 < a ∧ z + of_real a * (w − z) ∈ path_image γ
proof −
  have [simp]: ⋀x. 0 ≤ x ⟹ x ≤ 1 ⟹ γ x ≠ z
    using z by (auto simp: path_image_def)
  have [simp]: z ∉ γ ‘ {0..1}
    using path_image_def z by auto
  have gpd: γ piecewise_C1_differentiable_on {0..1}
    using γ valid_path_def by blast
  define r where r = (w − z) / (γ 0 − z)
  have [simp]: r ≠ 0
    using w z by (auto simp: r_def)
  have cont: continuous_on {0..1}
     (λx. Im (integral {0..x} (λx. vector_derivative γ (at x) / (γ x − z))))
   by (intro continuous_intros indefinite_integral_continuous_1 winding_number_exp_integral
[OF gpd]; simp)
  have Arg2pi r ≤ 2∗pi
    by (simp add: Arg2pi less_eq_real_def)
  also have ... ≤ Im (integral {0..1} (λx. vector_derivative γ (at x) / (γ x −
z)))
    using 1
   by (simp add: winding_number_valid_path [OF γ z] contour_integral_integral
Complex.Re_divide field_simps power2_eq_square)
  finally have Arg2pi r ≤ Im (integral {0..1} (λx. vector_derivative γ (at x) /
```

```
(γ x − z))) .
  then have ∃t. t ∈ {0..1} ∧ Im(integral {0..t} (λx. vector_derivative γ (at x)/(γ
x − z))) = Arg2pi r
    by (simp add: Arg2pi_ge_0 cont IVT')
  then obtain t where t:      t ∈ {0..1}
                  and eqArg: Im (integral {0..t} (λx. vector_derivative γ (at x)/(γ x
− z))) = Arg2pi r
    by blast
  define i where i = integral {0..t} (λx. vector_derivative γ (at x) / (γ x − z))
  have gpdt: γ piecewise_C1_differentiable_on {0..t}
   by (metis atLeastAtMost_iff atLeastatMost_subset_iff order_refl piecewise_C1_differentiable_on_subset
gpd t)
  have exp (− i) ∗ (γ t − z) = γ 0 − z
    unfolding i_def
  proof (rule winding_number_exp_integral [OF gpdt])
    show z ∉ γ ‘ {0..t}
      using t z unfolding path_image_def by force
  qed (use t in auto)
  then have ∗: γ t − z = exp i ∗ (γ 0 − z)
    by (simp add: exp_minus field_simps)
  then have (w − z) = r ∗ (γ 0 − z)
    by (simp add: r_def)
  moreover have z + exp (Re i) ∗ (exp (i ∗ Im i) ∗ (γ 0 − z)) = γ t
    using ∗ by (simp add: exp_eq_polar field_simps)
  moreover have Arg2pi r = Im i
    using eqArg by (simp add: i_def)
  ultimately have z + complex_of_real (exp (Re i)) ∗ (w − z) / complex_of_real
(cmod r) = γ t
    using Complex_Transcendental.Arg2pi_eq [of r] ‹r ≠ 0›
     by (metis  mult.left_commute nonzero_mult_div_cancel_left norm_eq_zero
of_real_0 of_real_eq_iff times_divide_eq_left)
  with t show ?thesis
    by (rule_tac x=exp(Re i) / norm r in exI) (auto simp: path_image_def)
qed

lemma winding_number_big_meets:
  fixes z::complex
  assumes γ: valid_path γ and z: z ∉ path_image γ and |Re (winding_number
γ z)| ≥ 1
      and w: w ≠ z
  shows ∃a::real. 0 < a ∧ z + of_real a ∗ (w − z) ∈ path_image γ
proof −
  { assume Re (winding_number γ z) ≤ − 1
    then have Re (winding_number (reversepath γ) z) ≥ 1
      by (simp add: γ valid_path_imp_path winding_number_reversepath z)
    moreover have valid_path (reversepath γ)
      using γ valid_path_imp_reverse by auto
    moreover have z ∉ path_image (reversepath γ)
      by (simp add: z)
```

```
    ultimately have ∃a::real. 0 < a ∧ z + of_real a * (w − z) ∈ path_image
(reversepath γ)
      using winding_number_pos_meets w by blast
    then have ?thesis
      by simp
  }
  then show ?thesis
    using assms
    by (simp add: abs_if winding_number_pos_meets split: if_split_asm)
qed

lemma winding_number_less_1:
  fixes z::complex
  shows
  ⟦valid_path γ; z ∉ path_image γ; w ≠ z;
    ⋀a::real. 0 < a ⟹ z + of_real a * (w − z) ∉ path_image γ⟧
   ⟹ Re(winding_number γ z) < 1
   by (auto simp: not_less dest: winding_number_big_meets)
```

One way of proving that WN=1 for a loop.

```
lemma winding_number_eq_1:
  fixes z::complex
   assumes γ: valid_path γ and z: z ∉ path_image γ and loop: pathfinish γ =
pathstart γ
      and 0: 0 < Re(winding_number γ z) and 2: Re(winding_number γ z) < 2
  shows winding_number γ z = 1
proof −
  have winding_number γ z ∈ Ints
    by (simp add: γ integer_winding_number loop valid_path_imp_path z)
  then show ?thesis
    using 0 2 by (auto simp: Ints_def)
qed
```

3.3 Continuity of winding number and invariance on connected sets

```
theorem continuous_at_winding_number:
  fixes z::complex
  assumes γ: path γ and z: z ∉ path_image γ
  shows continuous (at z) (winding_number γ)
proof −
  obtain e where e>0 and cbg: cball z e ⊆ − path_image γ
    using open_contains_cball [of − path_image γ]  z
    by (force simp: closed_def [symmetric] closed_path_image [OF γ])
  then have ppag: path_image γ ⊆ − cball z (e/2)
    by (force simp: cball_def dist_norm)
  have oc: open (− cball z (e/2))
    by (simp add: closed_def [symmetric])
  obtain d where d>0 and pi_eq:
```

```
  ⋀h1 h2. ⟦valid_path h1; valid_path h2;
          (∀t∈{0..1}. cmod (h1 t − γ t) < d ∧ cmod (h2 t − γ t) < d);
          pathstart h2 = pathstart h1; pathfinish h2 = pathfinish h1⟧
         ⟹
          path_image h1 ⊆ − cball z (e/2) ∧
          path_image h2 ⊆ − cball z (e/2) ∧
          (∀f. f holomorphic_on − cball z (e/2) ⟶ contour_integral h2 f =
contour_integral h1 f)
   using contour_integral_nearby_ends [OF oc γ ppag] by metis
 obtain p where valid_path p z ∉ path_image p
          and p: pathstart p = pathstart γ pathfinish p = pathfinish γ
          and pg: ⋀t. t∈{0..1} ⟹ cmod (γ t − p t) < min d e/2
          and pi: contour_integral p (λx. 1 / (x − z)) = complex_of_real (2 *
pi) * i * winding_number γ z
   using winding_number [OF γ z, of min d e/2] ‹d>0› ‹e>0› by (auto simp:
winding_number_prop_def)
 { fix w
   assume d2: cmod (w − z) < d/2 and e2: cmod (w − z) < e/2
   have wnotp: w ∉ path_image p
   proof (clarsimp simp add: path_image_def)
     show False if w: w = p x and 0 ≤ x x ≤ 1 for x
     proof −
       have cmod (γ x − p x) < min d e/2
         using pg that by auto
       then have cmod (z − γ x) < e
      by (metis e2 less_divide_eq_numeral1(1) min_less_iff_conj norm_minus_commute
norm_triangle_half_l w)
       then show ?thesis
         using cbg that by (auto simp add: path_image_def cball_def dist_norm
less_eq_real_def)
     qed
   qed
   have wnotg: w ∉ path_image γ
     using cbg e2 ‹e>0› by (force simp: dist_norm norm_minus_commute)
   { fix k::real
     assume k: k>0
     then obtain q where q: valid_path q w ∉ path_image q
                          pathstart q = pathstart γ ∧ pathfinish q = pathfinish γ
                    and qg: ⋀t. t ∈ {0..1} ⟹ cmod (γ t − q t) < min k (min d e)
/ 2
                    and qi: contour_integral q (λu. 1 / (u − w)) = complex_of_real
(2 * pi) * i * winding_number γ w
       using winding_number [OF γ wnotg, of min k (min d e) / 2] ‹d>0› ‹e>0›
k
       by (force simp: min_divide_distrib_right winding_number_prop_def)
     moreover have ⋀t. t ∈ {0..1} ⟹ cmod (q t − γ t) < d ∧ cmod (p t − γ
t) < d
       using pg qg ‹0 < d› by (fastforce simp add: norm_minus_commute)
     moreover have (λu. 1 / (u−w)) holomorphic_on − cball z (e/2)
```

using *e2* **by** (*auto simp*: *dist_norm norm_minus_commute intro*!: *holomorphic_intros*)
ultimately have *contour_integral p* (λ*u*. *1* / (*u* − *w*)) = *contour_integral q* (λ*u*. *1* / (*u* − *w*))
by (*metis p* ‹*valid_path p*› *pi_eq*)
then have *contour_integral p* (λ*x*. *1* / (*x* − *w*)) = *complex_of_real* (*2* ∗ *pi*) ∗ i ∗ *winding_number* γ *w*
by (*simp add*: *pi qi*)
} note *pip* = *this*
have *path p*
by (*simp add*: ‹*valid_path p*› *valid_path_imp_path*)
moreover have ⋀*e*. *e* > *0* ⟹ *winding_number_prop p w e p* (*winding_number* γ *w*)
by (*simp add*: ‹*valid_path p*› *pip winding_number_prop_def wnotp*)
ultimately have *winding_number p w* = *winding_number* γ *w*
using *winding_number_unique wnotp* **by** *blast*
} note *wnwn* = *this*
obtain *pe* **where** *pe*>*0* **and** *cbp*: *cball z* (*3* / *4* ∗ *pe*) ⊆ − *path_image p*
using ‹*valid_path p*› ‹*z* ∉ *path_image p*› *open_contains_cball* [*of* − *path_image p*]
by (*force simp*: *closed_def* [*symmetric*] *closed_path_image* [*OF valid_path_imp_path*])
obtain *L*
where *L*>*0*
and *L*: ⋀*f B*. ⟦*f holomorphic_on* − *cball z* (*3* / *4* ∗ *pe*);
∀ *z* ∈ − *cball z* (*3* / *4* ∗ *pe*). *cmod* (*f z*) ≤ *B*⟧ ⟹
cmod (*contour_integral p f*) ≤ *L* ∗ *B*
using *contour_integral_bound_exists* [*of* − *cball z* (*3*/*4*∗*pe*) *p*] *cbp* ‹*valid_path p*› **by** *blast*
{ fix *e*::*real* **and** *w*::*complex*
assume *e*: *0* < *e* **and** *w*: *cmod* (*w* − *z*) < *pe*/*4* *cmod* (*w* − *z*) < *e* ∗ *pe*^{2} / (*8* ∗ *L*)
then have [*simp*]: *w* ∉ *path_image p*
using *cbp p*(*2*) ‹*0* < *pe*›
by (*force simp*: *dist_norm norm_minus_commute path_image_def cball_def*)
have [*simp*]: *contour_integral p* (λ*x*. *1*/(*x* − *w*)) − *contour_integral p* (λ*x*. *1*/(*x* − *z*)) =
contour_integral p (λ*x*. *1*/(*x* − *w*) − *1*/(*x* − *z*))
by (*simp add*: ‹*valid_path p*› ‹*z* ∉ *path_image p*› *contour_integrable_inversediff contour_integral_diff*)
{ fix *x*
assume *pe*: *3*/*4* ∗ *pe* < *cmod* (*z* − *x*)
have *cmod* (*w* − *x*) < *pe*/*4* + *cmod* (*z* − *x*)
by (*meson add_less_cancel_right norm_diff_triangle_le order_refl order_trans_rules*(*21*) *w*(*1*))
then have *wx*: *cmod* (*w* − *x*) < *4*/*3* ∗ *cmod* (*z* − *x*) **using** *pe* **by** *simp*
have *cmod* (*z* − *x*) ≤ *cmod* (*z* − *w*) + *cmod* (*w* − *x*)
using *norm_diff_triangle_le* **by** *blast*
also have ... < *pe*/*4* + *cmod* (*w* − *x*)
using *w* **by** (*simp add*: *norm_minus_commute*)

finally have $pe/2 < cmod\ (w - x)$
using *pe* **by** *auto*
then have *pe_less*: $(pe/2)\hat{}2 < cmod\ (w - x)\ \hat{}\ 2$
by (*simp add*: ‹*0 < pe*› *less_eq_real_def power_strict_mono*)
then have *pe2*: $pe\hat{}2 < 4 * cmod\ (w - x)\ \hat{}\ 2$
by (*simp add*: *power_divide*)
have $8 * L * cmod\ (w - z) < e * pe^2$
using *w* ‹*L>0*› **by** (*simp add*: *field_simps*)
also have $\ldots < e * 4 * cmod\ (w - x) * cmod\ (w - x)$
using *pe2* ‹*e>0*› **by** (*simp add*: *power2_eq_square*)
also have $\ldots < e * 4 * cmod\ (w - x) * (4/3 * cmod\ (z - x))$
using ‹*0 < pe*› *pe_less e less_eq_real_def wx* **by** *fastforce*
finally have $L * cmod\ (w - z) < 2/3 * e * cmod\ (w - x) * cmod\ (z - x)$
by *simp*
also have $\ldots \leq e * cmod\ (w - x) * cmod\ (z - x)$
using *e* **by** *simp*
finally have *Lwz*: $L * cmod\ (w - z) < e * cmod\ (w - x) * cmod\ (z - x)$ **.**
have $L * cmod\ (1\ /\ (x - w) - 1\ /\ (x - z)) \leq e$
proof (*cases x=z* ∨ *x=w*)
case *True*
with *pe* ‹*pe>0*› *w* ‹*L>0*›
show *?thesis*
by (*force simp*: *norm_minus_commute*)
next
case *False*
with *wx w(2)* ‹*L>0*› *pe pe2 Lwz*
show *?thesis*
by (*auto simp*: *divide_simps mult_less_0_iff norm_minus_commute norm_divide norm_mult power2_eq_square*)
qed
} note *L_cmod_le* = *this*
let *?f* = $(\lambda x.\ 1\ /\ (x - w) - 1\ /\ (x - z))$
have $cmod\ (contour\_integral\ p\ ?f) \leq L * (e * pe^2\ /\ L\ /\ 4 * (inverse\ (pe\ /\ 2))^2)$
proof (*rule L*)
show *?f holomorphic_on − cball z* (*3 / 4 * pe*)
using ‹*pe>0*› *w*
by (*force simp*: *dist_norm norm_minus_commute intro*!: *holomorphic_intros*)
show $\forall u \in -\ cball\ z\ (3\ /\ 4 * pe).\ cmod\ (?f\ u) \leq e * pe^2\ /\ L\ /\ 4 * (inverse\ (pe\ /\ 2))^2$
using ‹*pe>0*› *w* ‹*L>0*›
by (*auto simp*: *cball_def dist_norm field_simps L_cmod_le simp del*: *less_divide_eq_numeral1 le_divide_eq_numeral1*)
qed
also have $\ldots < 2*e$
using ‹*L>0*› *e* **by** (*force simp*: *field_simps*)
finally have $cmod\ (winding\_number\ p\ w - winding\_number\ p\ z) < e$
using *pi_ge_two e*
by (*force simp*: *winding_number_valid_path* ‹*valid_path p*› ‹*z* ∉ *path_image*

```
p› field_simps norm_divide norm_mult intro: less_le_trans)
  } note cmod_wn_diff = this
  have isCont (winding_number p) z
  proof (clarsimp simp add: continuous_at_eps_delta)
    fix e::real assume e>0
   show ∃ d>0. ∀ x'. dist x' z < d ⟶ dist (winding_number p x') (winding_number
p z) < e
      using ‹pe>0› ‹L>0› ‹e>0›
      by (rule_tac x=min (pe/4) (e/2*pe^2/L/4) in exI) (simp add: dist_norm
cmod_wn_diff)
  qed
  then show ?thesis
    apply (rule continuous_transform_within [where d = min d e/2])
    apply (auto simp: ‹d>0› ‹e>0› dist_norm wnwn)
    done
qed

corollary continuous_on_winding_number:
    path γ ⟹ continuous_on (- path_image γ) (λw. winding_number γ w)
 by (simp add: continuous_at_imp_continuous_on continuous_at_winding_number)
```

3.4 The winding number is constant on a connected region

```
lemma winding_number_constant:
  assumes γ: path γ and loop: pathfinish γ = pathstart γ and cs: connected S
and sg: S ∩ path_image γ = {}
  shows winding_number γ constant_on S
proof -
  have *: 1 ≤ cmod (winding_number γ y - winding_number γ z)
      if ne: winding_number γ y ≠ winding_number γ z and y ∈ S z ∈ S for y z
  proof -
    have winding_number γ y ∈ ℤ  winding_number γ z ∈ ℤ
      using that integer_winding_number [OF γ loop] sg ‹y ∈ S› by auto
    with ne show ?thesis
      by (auto simp: Ints_def simp flip: of_int_diff)
  qed
  have cont: continuous_on S (λw. winding_number γ w)
    using continuous_on_winding_number [OF γ] sg
    by (meson continuous_on_subset disjoint_eq_subset_Compl)
  show ?thesis
    using * zero_less_one
    by (blast intro: continuous_discrete_range_constant [OF cs cont])
qed

lemma winding_number_eq:
    ⟦path γ; pathfinish γ = pathstart γ; w ∈ S; z ∈ S; connected S; S ∩ path_image
γ = {}⟧
      ⟹ winding_number γ w = winding_number γ z
  using winding_number_constant by (metis constant_on_def)
```

lemma *open_winding_number_levelsets*:
assumes γ: *path* γ **and** *loop*: *pathfinish* γ = *pathstart* γ
shows *open* {*z*. *z* ∉ *path_image* γ ∧ *winding_number* γ *z* = *k*}
proof (*clarsimp simp*: *open_dist*)
fix *z* **assume** *z*: *z* ∉ *path_image* γ **and** *k*: *k* = *winding_number* γ *z*
have *open* (− *path_image* γ)
by (*simp add*: *closed_path_image* γ *open_Compl*)
then obtain *e* **where** *e*>*0* *ball z e* ⊆ − *path_image* γ
using *open_contains_ball* [*of* − *path_image* γ] *z* **by** *blast*
then show ∃ *e*>*0*. ∀ *y*. *dist y z* < *e* ⟶ *y* ∉ *path_image* γ ∧ *winding_number* γ *y* = *winding_number* γ *z*
using ‹*e*>*0*› **by** (*force simp*: *norm_minus_commute dist_norm intro*: *winding_number_eq* [*OF assms*, **where** *S* = *ball z e*])
qed

3.5 Winding number is zero "outside" a curve

proposition *winding_number_zero_in_outside*:
assumes γ: *path* γ **and** *loop*: *pathfinish* γ = *pathstart* γ **and** *z*: *z* ∈ *outside* (*path_image* γ)
shows *winding_number* γ *z* = *0*
proof −
obtain *B*::*real* **where** *0* < *B* **and** *B*: *path_image* γ ⊆ *ball 0 B*
using *bounded_subset_ballD* [*OF bounded_path_image* [*OF* γ]] **by** *auto*
obtain *w*::*complex* **where** *w*: *w* ∉ *ball 0* (*B* + *1*)
by (*metis abs_of_nonneg le_less less_irrefl mem_ball_0 norm_of_real*)
have − *ball 0* (*B* + *1*) ⊆ *outside* (*path_image* γ)
using *B subset_ball* **by** (*intro outside_subset_convex*) *auto*
then have *wout*: *w* ∈ *outside* (*path_image* γ)
using *w* **by** *blast*
moreover have *winding_number* γ *constant_on outside* (*path_image* γ)
using *winding_number_constant* [*OF* γ *loop*, *of outside*(*path_image* γ)] *connected_outside*
by (*metis DIM_complex bounded_path_image dual_order.refl* γ *outside_no_overlap*)
ultimately have *winding_number* γ *z* = *winding_number* γ *w*
by (*metis* (*no_types*, *opaque_lifting*) *constant_on_def z*)
also have ... = *0*
proof −
have *wnot*: *w* ∉ *path_image* γ **using** *wout* **by** (*simp add*: *outside_def*)
{ **fix** *e*::*real* **assume** *0*<*e*
obtain *p* **where** *p*: *polynomial_function p pathstart p* = *pathstart* γ *pathfinish p* = *pathfinish* γ
and *pg1*: (⋀*t*. ⟦*0* ≤ *t*; *t* ≤ *1*⟧ ⟹ *cmod* (*p t* − γ *t*) < *1*)
and *pge*: (⋀*t*. ⟦*0* ≤ *t*; *t* ≤ *1*⟧ ⟹ *cmod* (*p t* − γ *t*) < *e*)
using *path_approx_polynomial_function* [*OF* γ, *of min 1 e*] ‹*e*>*0*›
by (*metis atLeastAtMost_iff min_less_iff_conj zero_less_one*)
have ∃ *p*. *valid_path p* ∧ *w* ∉ *path_image p* ∧
pathstart p = *pathstart* γ ∧ *pathfinish p* = *pathfinish* γ ∧

```
                (∀t∈{0..1}. cmod (γ t − p t) < e) ∧ contour_integral p (λwa. 1
/ (wa − w)) = 0
    proof (intro exI conjI)
      have ⋀x. ⟦0 ≤ x; x ≤ 1⟧ ⟹ cmod (p x) < B + 1
        using B unfolding image_subset_iff path_image_def
       by (meson add_strict_mono atLeastAtMost_iff le_less_trans mem_ball_0
norm_triangle_sub pg1)
      then have pip: path_image p ⊆ ball 0 (B + 1)
        by (auto simp add: path_image_def dist_norm ball_def)
      then show w ∉ path_image p using w by blast
      show vap: valid_path p
        by (simp add: p(1) valid_path_polynomial_function)
      show ∀t∈{0..1}. cmod (γ t − p t) < e
        by (metis atLeastAtMost_iff norm_minus_commute pge)
      show contour_integral p (λwa. 1 / (wa − w)) = 0
      proof (rule contour_integral_unique [OF Cauchy_theorem_convex_simple
[OF _ convex_ball [of 0 B+1]]])
        have ⋀z. cmod z < B + 1 ⟹ z ≠ w
          using mem_ball_0 w by blast
        then show (λz. 1 / (z − w)) holomorphic_on ball 0 (B + 1)
          by (intro holomorphic_intros; simp add: dist_norm)
      qed (use p vap pip loop in auto)
    qed (use p in auto)
  }
  then show ?thesis
  by (auto intro: winding_number_unique [OF γ] simp add: winding_number_prop_def
wnot)
 qed
 finally show ?thesis .
qed

corollary winding_number_zero_const: a ≠ z ⟹ winding_number (λt. a) z =
0
 by (rule winding_number_zero_in_outside)
    (auto simp: pathfinish_def pathstart_def path_polynomial_function)

corollary winding_number_zero_outside:
   ⟦path γ; convex s; pathfinish γ = pathstart γ; z ∉ s; path_image γ ⊆ s⟧ ⟹
winding_number γ z = 0
 by (meson convex_in_outside outside_mono subsetCE winding_number_zero_in_outside)

lemma winding_number_zero_at_infinity:
 assumes γ: path γ and loop: pathfinish γ = pathstart γ
   shows ∃B. ∀z. B ≤ norm z ⟶ winding_number γ z = 0
proof −
 obtain B::real where 0 < B and B: path_image γ ⊆ ball 0 B
   using bounded_subset_ballD [OF bounded_path_image [OF γ]] by auto
 have winding_number γ z = 0 if B + 1 ≤ cmod z for z
 proof (rule winding_number_zero_outside [OF γ convex_cball loop])
```

```
    show z ∉ cball 0 B
      using that by auto
    show path_image γ ⊆ cball 0 B
      using B order.trans by blast
  qed
  then show ?thesis
    by metis
qed

lemma winding_number_zero_point:
    ⟦path γ; convex S; pathfinish γ = pathstart γ; open S; path_image γ ⊆ S⟧
    ⟹ ∃z. z ∈ S ∧ winding_number γ z = 0
  using outside_compact_in_open [of path_image γ S] path_image_nonempty
winding_number_zero_in_outside
  by (fastforce simp add: compact_path_image)
```

If a path winds round a set, it winds rounds its inside.

```
lemma winding_number_around_inside:
  assumes γ: path γ and loop: pathfinish γ = pathstart γ
      and cls: closed S and cos: connected S and S_disj: S ∩ path_image γ = {}
      and z: z ∈ S and wn_nz: winding_number γ z ≠ 0 and w: w ∈ S ∪ inside
S
    shows winding_number γ w = winding_number γ z
proof −
  have ssb: S ⊆ inside(path_image γ)
  proof
    fix x :: complex
    assume x ∈ S
    hence x ∉ path_image γ
      by (meson disjoint_iff_not_equal S_disj)
    thus x ∈ inside (path_image γ)
       by (metis Compl_iff S_disj UnE γ ‹x ∈ S› cos inside_outside loop wind-
ing_number_eq winding_number_zero_in_outside wn_nz z)
  qed
  show ?thesis
  proof (rule winding_number_eq [OF γ loop w])
    show z ∈ S ∪ inside S
      using z by blast
    show connected (S ∪ inside S)
      by (simp add: cls connected_with_inside cos)
    show (S ∪ inside S) ∩ path_image γ = {}
      unfolding disjoint_iff Un_iff
      by (metis ComplD UnI1 γ cls compact_path_image connected_path_image
inside_Un_outside inside_inside_compact_connected ssb subsetD)
  qed
qed
```

Bounding a WN by 1/2 for a path and point in opposite halfspaces.

```
lemma winding_number_subpath_continuous:
```

assumes γ: *valid_path* γ **and** z: $z \notin$ *path_image* γ
shows *continuous_on* $\{0..1\}$ $(\lambda x.$ *winding_number*$($*subpath* 0 x $\gamma)$ $z)$
proof (*rule continuous_on_eq*)
let *?f* $= \lambda x.$ *integral* $\{0..x\}$ $(\lambda t.$ *vector_derivative* γ $(at\ t)$ / $(\gamma\ t - z))$
show *continuous_on* $\{0..1\}$ $(\lambda x.\ 1$ / $(2 * pi *$ i$) *$ *?f* $x)$
proof (*intro indefinite_integral_continuous_1 winding_number_exp_integral continuous_intros*)
show γ *piecewise_C1_differentiable_on* $\{0..1\}$
using γ *valid_path_def* **by** *blast*
qed (*use path_image_def z* **in** *auto*)
show *1* / $(2 * pi *$ i$) *$ *?f* $x =$ *winding_number* $($*subpath* 0 x $\gamma)$ z
if x: $x \in \{0..1\}$ **for** x
proof −
have *1* / $(2*pi*$i$) *$ *?f* $x =$ *1* / $(2*pi*$i$) *$ *contour_integral* $($*subpath* 0 x $\gamma)$ $(\lambda w.\ 1/(w - z))$
using *assms x*
by (*simp add*: *contour_integral_subcontour_integral* [*OF contour_integrable_inversediff*])
also have $\ldots =$ *winding_number* $($*subpath* 0 x $\gamma)$ z
proof (*subst winding_number_valid_path*)
show $z \notin$ *path_image* $($*subpath* 0 x $\gamma)$
using *assms x atLeastAtMost_iff path_image_subpath_subset* **by** *force*
qed (*use assms x valid_path_subpath* **in** ‹*force*+›)
finally show *?thesis* **.**
qed
qed

lemma *winding_number_ivt_pos*:
assumes γ: *valid_path* γ **and** z: $z \notin$ *path_image* γ **and** $0 \le w\ w \le$ *Re*$($*winding_number* $\gamma\ z)$
shows $\exists\, t \in \{0..1\}.$ *Re*$($*winding_number*$($*subpath* 0 t $\gamma)$ $z) = w$
proof −
have *continuous_on* $\{0..1\}$ $(\lambda x.$ *winding_number* $($*subpath* 0 x $\gamma)$ $z)$
using γ *winding_number_subpath_continuous z* **by** *blast*
moreover have *Re* $($*winding_number* $($*subpath* 0 0 $\gamma)$ $z) \le w\ w \le$ *Re* $($*winding_number* $($*subpath* 0 1 $\gamma)$ $z)$
using *assms* **by** (*auto simp*: *path_image_def image_def*)
ultimately show *?thesis*
using *ivt_increasing_component_on_1*[*of 0 1*, **where** *?k = 1*] **by** *force*
qed

lemma *winding_number_ivt_neg*:
assumes γ: *valid_path* γ **and** z: $z \notin$ *path_image* γ **and** *Re*$($*winding_number* $\gamma\ z) \le w\ w \le 0$
shows $\exists\, t \in \{0..1\}.$ *Re*$($*winding_number*$($*subpath* 0 t $\gamma)$ $z) = w$
proof −
have *continuous_on* $\{0..1\}$ $(\lambda x.$ *winding_number* $($*subpath* 0 x $\gamma)$ $z)$
using γ *winding_number_subpath_continuous z* **by** *blast*
moreover have *Re* $($*winding_number* $($*subpath* 0 0 $\gamma)$ $z) \ge w\ w \ge$ *Re* $($*winding_number* $($*subpath* 0 1 $\gamma)$ $z)$

```
    using assms by (auto simp: path_image_def image_def)
  ultimately show ?thesis
    using ivt_decreasing_component_on_1[of 0 1, where ?k = 1] by force
qed

lemma winding_number_ivt_abs:
  assumes γ: valid_path γ and z: z ∉ path_image γ and 0 ≤ w w ≤ |Re(winding_number
γ z)|
      shows ∃ t ∈ {0..1}. |Re (winding_number (subpath 0 t γ) z)| = w
  using assms winding_number_ivt_pos [of γ z w] winding_number_ivt_neg [of
γ z −w]
  by force

lemma winding_number_lt_half_lemma:
  assumes γ: valid_path γ and z: z ∉ path_image γ and az: a · z ≤ b and pag:
path_image γ ⊆ {w. a · w > b}
    shows Re(winding_number γ z) < 1/2
proof −
  { assume Re(winding_number γ z) ≥ 1/2
    then obtain t::real where t: 0 ≤ t t ≤ 1 and sub12: Re (winding_number
(subpath 0 t γ) z) = 1/2
      using winding_number_ivt_pos [OF γ z, of 1/2] by auto
   have gt: γ t − z = − (of_real (exp (− (2 ∗ pi ∗ Im (winding_number (subpath
0 t γ) z)))) ∗ (γ 0 − z))
      using winding_number_exp_2pi [of subpath 0 t γ z]
      apply (simp add: t γ valid_path_imp_path)
       using closed_segment_eq_real_ivl path_image_def t z by (fastforce simp:
path_image_subpath Euler sub12)
    have b < a · γ 0
    proof −
      have γ 0 ∈ {c. b < a · c}
        by (metis (no_types) pag atLeastAtMost_iff image_subset_iff order_refl
path_image_def zero_le_one)
      thus ?thesis
        by blast
    qed
    moreover have b < a · γ t
     by (metis atLeastAtMost_iff image_eqI mem_Collect_eq pag path_image_def
subset_iff t)
    ultimately have 0 < a · (γ 0 − z) 0 < a · (γ t − z) using az
      by (simp add: inner_diff_right)+
    then have False
      by (simp add: gt inner_mult_right mult_less_0_iff)
  }
  then show ?thesis by force
qed

lemma winding_number_lt_half:
  assumes valid_path γ a · z ≤ b path_image γ ⊆ {w. a · w > b}
```

shows $|Re\ (winding\_number\ \gamma\ z)| < 1/2$
proof −
have $z \notin path\_image\ \gamma$ **using** *assms* **by** *auto*
with *assms* **have** $Re\ (winding\_number\ \gamma\ z) < 0 \Longrightarrow - Re\ (winding\_number\ \gamma\ z) < 1/2$
by (*metis complex_inner_1_right winding_number_lt_half_lemma* [*OF valid_path_imp_reverse*, *of* γ *z a b*]
winding_number_reversepath valid_path_imp_path inner_minus_left path_image_reversepath)
with *assms* **show** *?thesis*
using ‹$z \notin path\_image\ \gamma$› *winding_number_lt_half_lemma* **by** *fastforce*
qed

lemma *winding_number_le_half*:
assumes γ: *valid_path* γ **and** z: $z \notin path\_image\ \gamma$
and *anz*: $a \neq 0$ **and** *azb*: $a \cdot z \leq b$ **and** *pag*: $path\_image\ \gamma \subseteq \{w.\ a \cdot w \geq b\}$
shows $|Re\ (winding\_number\ \gamma\ z)| \leq 1/2$
proof −
{ assume *wnz_12*: $|Re\ (winding\_number\ \gamma\ z)| > 1/2$
have *isCont* (*winding_number* γ) z
by (*metis continuous_at_winding_number valid_path_imp_path* γ z)
then obtain d **where** $d{>}0$ **and** d: $\bigwedge x'.\ dist\ x'\ z < d \Longrightarrow dist\ (winding\_number\ \gamma\ x')\ (winding\_number\ \gamma\ z) < |Re(winding\_number\ \gamma\ z)| - 1/2$
using *continuous_at_eps_delta wnz_12 diff_gt_0_iff_gt* **by** *blast*
define z' **where** $z' = z - (d\ /\ (2 * cmod\ a)) *_R a$
have $a \cdot z * 6 \leq d * cmod\ a + b * 6$
by (*metis* ‹$0 < d$› *add_increasing azb less_eq_real_def mult_nonneg_nonneg mult_right_mono norm_ge_zero norm_numeral*)
with *anz* **have** $*$: $a \cdot z' \leq b - d\ /\ 3 * cmod\ a$
unfolding z'*_def inner_mult_right' divide_inverse*
by (*simp add*: *field_split_simps algebra_simps dot_square_norm power2_eq_square*)
have $cmod\ (winding\_number\ \gamma\ z' - winding\_number\ \gamma\ z) < |Re\ (winding\_number\ \gamma\ z)| - 1/2$
using d [*of* z'] *anz* ‹$d{>}0$› **by** (*simp add*: *dist_norm* z'*_def*)
then have $1/2 < |Re\ (winding\_number\ \gamma\ z)| - cmod\ (winding\_number\ \gamma\ z' - winding\_number\ \gamma\ z)$
by *simp*
then have $1/2 < |Re\ (winding\_number\ \gamma\ z)| - |Re\ (winding\_number\ \gamma\ z') - Re\ (winding\_number\ \gamma\ z)|$
using *abs_Re_le_cmod* [*of winding_number* γ z' − *winding_number* γ z]
by *simp*
then have *wnz_12'*: $|Re\ (winding\_number\ \gamma\ z')| > 1/2$
by *linarith*
moreover have $|Re\ (winding\_number\ \gamma\ z')| < 1/2$
proof (*rule winding_number_lt_half* [*OF* γ $*$])
show $path\_image\ \gamma \subseteq \{w.\ b - d\ /\ 3 * cmod\ a < a \cdot w\}$
using *azb* ‹$d{>}0$› *pag* **by** (*auto simp*: *add_strict_increasing anz field_split_simps dest*!: *subsetD*)
qed
ultimately have *False*

```
    by simp
  }
  then show ?thesis by force
qed

lemma winding_number_lt_half_linepath:
  assumes z ∉ closed_segment a b shows |Re (winding_number (linepath a b) z)|
< 1/2
proof −
  obtain u v where u · z ≤ v and uv: ⋀x. x ∈ closed_segment a b ⟹ inner u
x > v
   using separating_hyperplane_closed_point assms closed_segment convex_closed_segment
less_eq_real_def by metis
  moreover have path_image (linepath a b) ⊆ {w. v < u · w}
    using in_segment(1) uv by auto
  ultimately show ?thesis
    using winding_number_lt_half by auto
qed
```

Positivity of WN for a linepath.

```
lemma winding_number_linepath_pos_lt:
    assumes 0 < Im ((b − a) ∗ cnj (b − z))
      shows 0 < Re(winding_number(linepath a b) z)
proof −
  have z: z ∉ path_image (linepath a b)
    using assms
    by (simp add: closed_segment_def) (force simp: algebra_simps)
  show ?thesis
     by (intro winding_number_pos_lt [OF valid_path_linepath z assms]) (simp
add: linepath_def algebra_simps)
qed
```

3.6 More winding number properties

including the fact that it's +-1 inside a simple closed curve.

```
lemma winding_number_homotopic_paths:
    assumes homotopic_paths (−{z}) g h
      shows winding_number g z = winding_number h z
proof −
  have path g path h using homotopic_paths_imp_path [OF assms] by auto
  moreover have pag: z ∉ path_image g and pah: z ∉ path_image h
    using homotopic_paths_imp_subset [OF assms] by auto
  ultimately obtain d e where d > 0 e > 0
      and d: ⋀p. ⟦path p; pathstart p = pathstart g; pathfinish p = pathfinish g;
∀ t∈{0..1}. norm (p t − g t) < d⟧
          ⟹ homotopic_paths (−{z}) g p
      and e: ⋀q. ⟦path q; pathstart q = pathstart h; pathfinish q = pathfinish h;
∀ t∈{0..1}. norm (q t − h t) < e⟧
          ⟹ homotopic_paths (−{z}) h q
```

```
    using homotopic_nearby_paths [of g -{z}] homotopic_nearby_paths [of h
-{z}] by force
  obtain p where p:
      valid_path p z ∉ path_image p
      pathstart p = pathstart g pathfinish p = pathfinish g
      and gp_less:∀ t∈{0..1}. cmod (g t - p t) < d
      and pap: contour_integral p (λw. 1 / (w - z)) = complex_of_real (2 * pi)
* i * winding_number g z
   using winding_number [OF ‹path g› pag ‹0 < d›] unfolding winding_number_prop_def
by blast
  obtain q where q:
      valid_path q z ∉ path_image q
      pathstart q = pathstart h pathfinish q = pathfinish h
      and hq_less: ∀ t∈{0..1}. cmod (h t - q t) < e
      and paq:  contour_integral q (λw. 1 / (w - z)) = complex_of_real (2 * pi)
* i * winding_number h z
   using winding_number [OF ‹path h› pah ‹0 < e›] unfolding winding_number_prop_def
by blast
  have homotopic_paths (- {z}) g p
    by (simp add: d p valid_path_imp_path norm_minus_commute gp_less)
  moreover have homotopic_paths (- {z}) h q
    by (simp add: e q valid_path_imp_path norm_minus_commute hq_less)
  ultimately have homotopic_paths (- {z}) p q
    by (blast intro: homotopic_paths_trans homotopic_paths_sym assms)
  then have contour_integral p (λw. 1/(w - z)) = contour_integral q (λw. 1/(w
- z))
    by (rule Cauchy_theorem_homotopic_paths) (auto intro!: holomorphic_intros
simp: p q)
  then show ?thesis
    by (simp add: pap paq)
qed

lemma winding_number_homotopic_loops:
    assumes homotopic_loops (-{z}) g h
      shows winding_number g z = winding_number h z
proof -
  have path g path h using homotopic_loops_imp_path [OF assms] by auto
  moreover have pag: z ∉ path_image g and pah: z ∉ path_image h
    using homotopic_loops_imp_subset [OF assms] by auto
   moreover have gloop: pathfinish g = pathstart g and hloop: pathfinish h =
pathstart h
    using homotopic_loops_imp_loop [OF assms] by auto
  ultimately obtain d e where d > 0 e > 0
      and d: ⋀p. ⟦path p; pathfinish p = pathstart p; ∀ t∈{0..1}. norm (p t - g t)
< d⟧
            ⟹ homotopic_loops (-{z}) g p
      and e: ⋀q. ⟦path q; pathfinish q = pathstart q; ∀ t∈{0..1}. norm (q t - h t)
< e⟧
            ⟹ homotopic_loops (-{z}) h q
```

using *homotopic_nearby_loops* [*of g* −{*z*}] *homotopic_nearby_loops* [*of h* −{*z*}] **by** *force*
obtain *p* **where** *p*:
valid_path p z ∉ *path_image p*
pathstart p = *pathstart g pathfinish p* = *pathfinish g*
and *gp_less*:∀ *t*∈{*0..1*}. *cmod* (*g t* − *p t*) < *d*
and *pap*: *contour_integral p* (λ*w*. *1* / (*w* − *z*)) = *complex_of_real* (*2* ∗ *pi*) ∗ i ∗ *winding_number g z*
using *winding_number* [*OF* ‹*path g*› *pag* ‹*0* < *d*›] **unfolding** *winding_number_prop_def* **by** *blast*
obtain *q* **where** *q*:
valid_path q z ∉ *path_image q*
pathstart q = *pathstart h pathfinish q* = *pathfinish h*
and *hq_less*: ∀ *t*∈{*0..1*}. *cmod* (*h t* − *q t*) < *e*
and *paq*: *contour_integral q* (λ*w*. *1* / (*w* − *z*)) = *complex_of_real* (*2* ∗ *pi*) ∗ i ∗ *winding_number h z*
using *winding_number* [*OF* ‹*path h*› *pah* ‹*0* < *e*›] **unfolding** *winding_number_prop_def* **by** *blast*
have *gp*: *homotopic_loops* (− {*z*}) *g p*
by (*simp add*: *gloop d gp_less norm_minus_commute p valid_path_imp_path*)
have *hq*: *homotopic_loops* (− {*z*}) *h q*
by (*simp add*: *e hloop hq_less norm_minus_commute q valid_path_imp_path*)
have *contour_integral p* (λ*w*. *1*/(*w* − *z*)) = *contour_integral q* (λ*w*. *1*/(*w* − *z*))
proof (*rule Cauchy_theorem_homotopic_loops*)
show *homotopic_loops* (− {*z*}) *p q*
by (*blast intro*: *homotopic_loops_trans homotopic_loops_sym gp hq assms*)
qed (*auto intro*!: *holomorphic_intros simp*: *p q*)
then show *?thesis*
by (*simp add*: *pap paq*)
qed

lemma *winding_number_paths_linear_eq*:
⟦*path g*; *path h*; *pathstart h* = *pathstart g*; *pathfinish h* = *pathfinish g*;
⋀*t*. *t* ∈ {*0..1*} ⟹ *z* ∉ *closed_segment* (*g t*) (*h t*)⟧
⟹ *winding_number h z* = *winding_number g z*
by (*blast intro*: *sym homotopic_paths_linear winding_number_homotopic_paths*)

lemma *winding_number_loops_linear_eq*:
⟦*path g*; *path h*; *pathfinish g* = *pathstart g*; *pathfinish h* = *pathstart h*;
⋀*t*. *t* ∈ {*0..1*} ⟹ *z* ∉ *closed_segment* (*g t*) (*h t*)⟧
⟹ *winding_number h z* = *winding_number g z*
by (*blast intro*: *sym homotopic_loops_linear winding_number_homotopic_loops*)

lemma *winding_number_nearby_paths_eq*:
⟦*path g*; *path h*; *pathstart h* = *pathstart g*; *pathfinish h* = *pathfinish g*;
⋀*t*. *t* ∈ {*0..1*} ⟹ *norm*(*h t* − *g t*) < *norm*(*g t* − *z*)⟧
⟹ *winding_number h z* = *winding_number g z*
by (*metis segment_bound*(*2*) *norm_minus_commute not_le winding_number_paths_linear_eq*)

lemma *winding_number_nearby_loops_eq*:
⟦*path g*; *path h*; *pathfinish g* = *pathstart g*; *pathfinish h* = *pathstart h*;
⋀*t*. *t* ∈ {*0..1*} ⟹ *norm*(*h t* − *g t*) < *norm*(*g t* − *z*)⟧
⟹ *winding_number h z* = *winding_number g z*
by (*metis segment_bound*(*2*) *norm_minus_commute not_le winding_number_loops_linear_eq*)

lemma *winding_number_subpath_combine*:
assumes *path g* **and** *notin*: *z* ∉ *path_image g* **and** *u* ∈ {*0..1*} *v* ∈ {*0..1*} *w* ∈ {*0..1*}
shows *winding_number* (*subpath u v g*) *z* + *winding_number* (*subpath v w g*) *z* =
winding_number (*subpath u w g*) *z* (**is** *?lhs* = *?rhs*)
proof −
have *?lhs* = *winding_number* (*subpath u v g* +++ *subpath v w g*) *z*
using *assms*
by (*metis* (*no_types*, *lifting*) *path_image_subpath_subset path_subpath pathfinish_subpath pathstart_subpath subsetD winding_number_join*)
also have ... = *?rhs*
by (*meson assms homotopic_join_subpaths subset_Compl_singleton winding_number_homotopic_paths*)
finally show *?thesis* **.**
qed

Winding numbers of circular contours

proposition *winding_number_part_circlepath_pos_less*:
assumes *s* < *t* **and** *no*: *norm*(*w* − *z*) < *r*
shows *0* < *Re* (*winding_number*(*part_circlepath z r s t*) *w*)
proof −
have *0* < *r* **by** (*meson no norm_not_less_zero not_le order.strict_trans2*)
note *valid_path_part_circlepath*
moreover have *w* ∉ *path_image* (*part_circlepath z r s t*)
using *assms* **by** (*auto simp*: *path_image_def image_def part_circlepath_def norm_mult linepath_def*)
moreover have *0* < *r* ∗ (*t* − *s*) ∗ (*r* − *cmod* (*w* − *z*))
using *assms* **by** (*metis* ‹*0* < *r*› *diff_gt_0_iff_gt mult_pos_pos*)
ultimately show *?thesis*
apply (*rule winding_number_pos_lt* [**where** *e* = *r*∗(*t* − *s*)∗(*r* − *norm*(*w* − *z*))])
apply (*simp add*: *vector_derivative_part_circlepath right_diff_distrib* [*symmetric*] *mult_ac mult_le_cancel_left_pos assms* ‹*0*<*r*›)
using *Re_Im_le_cmod* [*of w*−*z linepath s t x* **for** *x*]
by (*simp add*: *exp_Euler cos_of_real sin_of_real part_circlepath_def algebra_simps cos_squared_eq* [*unfolded power2_eq_square*])
qed

lemma *winding_number_circlepath_centre*: *0* < *r* ⟹ *winding_number* (*circlepath z r*) *z* = *1*
apply (*rule winding_number_unique_loop*)

apply (*simp_all add*: *sphere_def valid_path_imp_path*)
apply (*rule_tac x*=*circlepath z r* **in** *exI*)
apply (*simp add*: *sphere_def contour_integral_circlepath*)
done

proposition *winding_number_circlepath*:
assumes *norm*(*w* − *z*) < *r* **shows** *winding_number*(*circlepath z r*) *w* = *1*
proof (*cases w* = *z*)
case *True* **then show** *?thesis*
using *assms winding_number_circlepath_centre* **by** *auto*
next
case *False*
have [*simp*]: *r* > *0*
using *assms le_less_trans norm_ge_zero* **by** *blast*
define *r′* **where** *r′* = *norm*(*w* − *z*)
have *r′* < *r*
by (*simp add*: *assms r′_def*)
have *disjo*: *cball z r′* ∩ *sphere z r* = {}
using ‹*r′* < *r*› **by** (*force simp*: *cball_def sphere_def*)
have *winding_number*(*circlepath z r*) *w* = *winding_number*(*circlepath z r*) *z*
proof (*rule winding_number_around_inside* [**where** *S* = *cball z r′*])
show *winding_number* (*circlepath z r*) *z* ≠ *0*
by (*simp add*: *winding_number_circlepath_centre*)
show *cball z r′* ∩ *path_image* (*circlepath z r*) = {}
by (*simp add*: *disjo less_eq_real_def*)
qed (*auto simp*: *r′_def dist_norm norm_minus_commute*)
also have ... = *1*
by (*simp add*: *winding_number_circlepath_centre*)
finally show *?thesis* **.**
qed

lemma *no_bounded_connected_component_imp_winding_number_zero*:
assumes *g*: *path g path_image g* ⊆ *S pathfinish g* = *pathstart g z* ∉ *S*
and *nb*: ⋀*z*. *bounded* (*connected_component_set* (− *S*) *z*) ⟹ *z* ∈ *S*
shows *winding_number g z* = *0*
proof −
have *z* ∈ *outside* (*path_image g*)
by (*metis nb* [*of z*] ‹*path_image g* ⊆ *S*› ‹*z* ∉ *S*› *subsetD mem_Collect_eq outside outside_mono*)
then show *?thesis*
by (*simp add*: *g winding_number_zero_in_outside*)
qed

lemma *no_bounded_path_component_imp_winding_number_zero*:
assumes *g*: *path g path_image g* ⊆ *S pathfinish g* = *pathstart g z* ∉ *S*
and *nb*: ⋀*z*. *bounded* (*path_component_set* (− *S*) *z*) ⟹ *z* ∈ *S*
shows *winding_number g z* = *0*
by (*simp add*: *bounded_subset nb path_component_subset_connected_component*

no_bounded_connected_component_imp_winding_number_zero [*OF g*])

3.7 Winding number for a triangle

lemma *wn_triangle1*:
assumes *0* ∈ *interior*(*convex hull* {*a*,*b*,*c*})
shows ¬ (*Im*(*a*/*b*) ≤ *0* ∧ *0* ≤ *Im*(*b*/*c*))
proof −
{ assume *0*: *Im*(*a*/*b*) ≤ *0* *0* ≤ *Im*(*b*/*c*)
have *0* ∉ *interior* (*convex hull* {*a*,*b*,*c*})
proof (*cases a=0* ∨ *b=0* ∨ *c=0*)
case *True* **then show** *?thesis*
by (*auto simp*: *not_in_interior_convex_hull_3*)
next
case *False*
then have *b* ≠ *0* **by** *blast*
{ fix *x y*::*complex* **and** *u*::*real*
assume *eq_f'*: *Im x* ∗ *Re b* ≤ *Im b* ∗ *Re x Im y* ∗ *Re b* ≤ *Im b* ∗ *Re y* *0* ≤ *u u* ≤ *1*
then have ((*1* − *u*) ∗ *Im x*) ∗ *Re b* ≤ *Im b* ∗ ((*1* − *u*) ∗ *Re x*)
by (*simp add*: *mult_left_mono mult.assoc mult.left_commute* [*of Im b*])
then have ((*1* − *u*) ∗ *Im x* + *u* ∗ *Im y*) ∗ *Re b* ≤ *Im b* ∗ ((*1* − *u*) ∗ *Re x* + *u* ∗ *Re y*)
using *eq_f' ordered_comm_semiring_class.comm_mult_left_mono*
by (*fastforce simp add*: *algebra_simps*)
}
then have *convex* {*z*. *Im z* ∗ *Re b* ≤ *Im b* ∗ *Re z*}
by (*auto simp*: *algebra_simps convex_alt*)
with *False 0* **have** *convex hull* {*a*,*b*,*c*} ≤ {*z*. *Im z* ∗ *Re b* ≤ *Im b* ∗ *Re z*}
by (*simp add*: *subset_hull mult.commute Complex.Im_divide divide_simps complex_neq_0* [*symmetric*])
moreover have *0* ∉ *interior*({*z*. *Im z* ∗ *Re b* ≤ *Im b* ∗ *Re z*})
proof
assume *0* ∈ *interior* {*z*. *Im z* ∗ *Re b* ≤ *Im b* ∗ *Re z*}
then obtain *e* **where** *e>0* **and** *e*: *ball 0 e* ⊆ {*z*. *Im z* ∗ *Re b* ≤ *Im b* ∗ *Re z*}
by (*meson mem_interior*)
define *z* **where** *z* ≡ − *sgn* (*Im b*) ∗ (*e*/*3*) + *sgn* (*Re b*) ∗ (*e*/*3*) ∗ i
have *cmod z* = *cmod* (*e*/*3*) ∗ *cmod* (− *sgn* (*Im b*) + *sgn* (*Re b*) ∗ i)
unfolding *z_def norm_mult* [*symmetric*] **by** (*simp add*: *algebra_simps*)
also have ... < *e*
using ‹*e>0*› **by** (*auto simp*: *norm_mult intro*: *le_less_trans* [*OF norm_triangle_ineq4*])
finally have *z* ∈ *ball 0 e*
using ‹*e>0*› **by** (*simp add*:)
then have *Im z* ∗ *Re b* ≤ *Im b* ∗ *Re z*
using *e* **by** *blast*
then have *b*: *0* < *Re b* *0* < *Im b* **and** *disj*: *e* ∗ *Re b* < − (*Im b* ∗ *e*) ∨ *e* ∗

```
Re b = - (Im b * e)
        using ‹e>0› ‹b ≠ 0›
      by (auto simp add: z_def dist_norm sgn_if less_eq_real_def mult_less_0_iff
complex.expand split: if_split_asm)
       show False — or just one smt line
        using disj
       proof
        assume e * Re b < - (Im b * e)
        with b ‹0 < e› less_trans [of _ 0] show False
       by (metis (no_types) mult_pos_pos neg_less_0_iff_less not_less_iff_gr_or_eq)
       next
        assume e * Re b = - (Im b * e)
        with b ‹0 < e› show False
          by (metis mult_pos_pos neg_less_0_iff_less not_less_iff_gr_or_eq)
       qed
     qed
     ultimately show ?thesis
      using interior_mono by blast
   qed
  } with assms show ?thesis by blast
qed

lemma wn_triangle2_0:
  assumes 0 ∈ interior(convex hull {a,b,c})
  shows
       0 < Im((b - a) * cnj (b)) ∧
        0 < Im((c - b) * cnj (c)) ∧
        0 < Im((a - c) * cnj (a))
        ∨
        Im((b - a) * cnj (b)) < 0 ∧
        0 < Im((b - c) * cnj (b)) ∧
        0 < Im((a - b) * cnj (a)) ∧
        0 < Im((c - a) * cnj (c))
proof -
  have [simp]: {b,c,a} = {a,b,c} {c,a,b} = {a,b,c} by auto
  show ?thesis
    using wn_triangle1 [OF assms] wn_triangle1 [of b c a] wn_triangle1 [of c a
b] assms
    by (auto simp: algebra_simps Im_complex_div_gt_0 Im_complex_div_lt_0
not_le not_less)
qed

lemma wn_triangle2:
  assumes z ∈ interior(convex hull {a,b,c})
   shows 0 < Im((b - a) * cnj (b - z)) ∧
         0 < Im((c - b) * cnj (c - z)) ∧
         0 < Im((a - c) * cnj (a - z))
         ∨
         Im((b - a) * cnj (b - z)) < 0 ∧
```

```
        0 < Im((b − c) * cnj (b − z)) ∧
        0 < Im((a − b) * cnj (a − z)) ∧
        0 < Im((c − a) * cnj (c − z))
proof −
  have 0: 0 ∈ interior(convex hull {a−z, b−z, c−z})
    using assms convex_hull_translation [of −z {a,b,c}]
          interior_translation [of −z]
    by (simp cong: image_cong_simp)
  show ?thesis using wn_triangle2_0 [OF 0]
    by simp
qed

lemma wn_triangle3:
  assumes z: z ∈ interior(convex hull {a,b,c})
      and 0 < Im((b−a) * cnj (b−z))
        0 < Im((c−b) * cnj (c−z))
        0 < Im((a−c) * cnj (a−z))
    shows winding_number (linepath a b +++ linepath b c +++ linepath c a) z
= 1
proof −
  have znot[simp]: z ∉ closed_segment a b z ∉ closed_segment b c z ∉ closed_segment
c a
    using z interior_of_triangle [of a b c]
    by (auto simp: closed_segment_def)
  have gt0: 0 < Re (winding_number (linepath a b +++ linepath b c +++ linepath
c a) z)
    using assms
   by (simp add: winding_number_linepath_pos_lt path_image_join winding_number_join_pos_combined)
  have lt2: Re (winding_number (linepath a b +++ linepath b c +++ linepath c
a) z) < 2
    using winding_number_lt_half_linepath [of _ a b]
    using winding_number_lt_half_linepath [of _ b c]
    using winding_number_lt_half_linepath [of _ c a] znot
    by (fastforce simp add: winding_number_join path_image_join)
  show ?thesis
    by (rule winding_number_eq_1) (simp_all add: path_image_join gt0 lt2)
qed

proposition winding_number_triangle:
  assumes z: z ∈ interior(convex hull {a,b,c})
    shows winding_number(linepath a b +++ linepath b c +++ linepath c a) z =
          (if 0 < Im((b − a) * cnj (b − z)) then 1 else −1)
proof −
  have [simp]: {a,c,b} = {a,b,c}  by auto
  have znot[simp]: z ∉ closed_segment a b z ∉ closed_segment b c z ∉ closed_segment
c a
    using z interior_of_triangle [of a b c]
    by (auto simp: closed_segment_def)
  then have [simp]: z ∉ closed_segment b a z ∉ closed_segment c b z ∉ closed_segment
```

```
a c
    using closed_segment_commute by blast+
  have *: winding_number (linepath a b +++ linepath b c +++ linepath c a) z =
            winding_number (reversepath (linepath a c +++ linepath c b +++
linepath b a)) z
   by (simp add: reversepath_joinpaths winding_number_join not_in_path_image_join)
  show ?thesis
    apply (rule disjE [OF wn_triangle2 [OF z]])
    subgoal
      by (simp add: wn_triangle3 z)
    subgoal
     by (simp add: path_image_join winding_number_reversepath * wn_triangle3
z)
    done
qed
```

3.8 Winding numbers for simple closed paths

```
lemma winding_number_from_innerpath:
  assumes simple_path c1 and c1: pathstart c1 = a pathfinish c1 = b
      and simple_path c2 and c2: pathstart c2 = a pathfinish c2 = b
      and simple_path c and c: pathstart c = a pathfinish c = b
      and c1c2: path_image c1 ∩ path_image c2 = {a,b}
      and c1c:  path_image c1 ∩ path_image c = {a,b}
      and c2c:  path_image c2 ∩ path_image c = {a,b}
      and ne_12: path_image c ∩ inside(path_image c1 ∪ path_image c2) ≠ {}
      and z: z ∈ inside(path_image c1 ∪ path_image c)
      and wn_d: winding_number (c1 +++ reversepath c) z = d
      and a ≠ b d ≠ 0
  obtains z ∈ inside(path_image c1 ∪ path_image c2) winding_number (c1 +++
reversepath c2) z = d
proof −
  obtain 0: inside(path_image c1 ∪ path_image c) ∩ inside(path_image c2 ∪
path_image c) = {}
      and 1: inside(path_image c1 ∪ path_image c) ∪ inside(path_image c2 ∪
path_image c) ∪
            (path_image c − {a,b}) = inside(path_image c1 ∪ path_image c2)
    by (rule split_inside_simple_closed_curve
            [OF ‹simple_path c1› c1 ‹simple_path c2› c2 ‹simple_path c› c ‹a ≠
b› c1c2 c1c c2c ne_12])
  have znot: z ∉ path_image c  z ∉ path_image c1 z ∉ path_image c2
    using union_with_outside z 1 by auto
  then have zout: z ∈ outside (path_image c ∪ path_image c2)
   by (metis 0 ComplI UnE disjoint_iff_not_equal sup.commute union_with_inside
z)
  have wn_cc2: winding_number (c +++ reversepath c2) z = 0
    by (rule winding_number_zero_in_outside; simp add: zout ‹simple_path c2›
c c2 ‹simple_path c› simple_path_imp_path path_image_join)
  show ?thesis
```

proof
show $z \in$ *inside* (*path_image c1* $\cup$ *path_image c2*)
using *1 z* **by** *blast*
have *winding_number c1 z* $-$ *winding_number c z* $= d$
using *assms znot*
by (*metis wn_d winding_number_join simple_path_imp_path winding_number_reversepath add.commute path_image_reversepath path_reversepath pathstart_reversepath uminus_add_conv_diff*)
then show *winding_number* (*c1 +++ reversepath c2*) *z* $= d$
using *wn_cc2* **by** (*simp add*: *winding_number_join simple_path_imp_path assms znot winding_number_reversepath*)
qed
qed

lemma *simple_closed_path_wn1*:
fixes *a*::*complex* **and** *e*::*real*
assumes $0 < e$
and *sp_pl*: *simple_path*(*p +++ linepath* $(a - e)$ $(a + e)$) (**is** *simple_path ?pae*)
and *psp*: *pathstart p* $= a + e$
and *pfp*: *pathfinish p* $= a - e$
and *disj*: *ball a e* $\cap$ *path_image p* $= \{\}$
obtains *z* **where** $z \in$ *inside* (*path_image ?pae*) *cmod* (*winding_number ?pae z*) $= 1$
proof $-$
have *arc p* **and** *arc_lp*: *arc* (*linepath* $(a - e)$ $(a + e)$)
and *pap*: *path_image p* $\cap$ *path_image* (*linepath* $(a - e)$ $(a + e)$) $\subseteq \{$*pathstart p*, $a-e\}$
using *simple_path_join_loop_eq* [*of linepath* $(a - e)$ $(a + e)$ *p*] *assms* **by** *auto*
have *mid_eq_a*: *midpoint* $(a - e)$ $(a + e) = a$
by (*simp add*: *midpoint_def*)
with *assms* **have** *a* $\in$ *path_image ?pae*
by (*simp add*: *assms path_image_join*) (*metis midpoint_in_closed_segment*)
then have *a* $\in$ *frontier*(*inside* (*path_image ?pae*))
using *assms* **by** (*simp add*: *Jordan_inside_outside*)
with ‹$0 < e$› **obtain** *c* **where** *c*: $c \in$ *inside* (*path_image ?pae*)
and *dac*: *dist a c* $< e$
by (*auto simp*: *frontier_straddle*)
then have $c \notin$ *path_image ?pae*
using *inside_no_overlap* **by** *blast*
then have $c \notin$ *path_image p* $c \notin$ *closed_segment* $(a - e)$ $(a + e)$
by (*simp_all add*: *assms path_image_join*)
with ‹$0 < e$› *dac* **have** $c \notin$ *affine hull* $\{a - e, a + e\}$
by (*simp add*: *segment_as_ball not_le*)
with ‹$0 < e$› **have** $*$: $\neg$ *collinear* $\{a - e, c, a + e\}$
using *collinear_3_affine_hull* [*of a*$-$*e a*+*e*] **by** (*auto simp*: *insert_commute*)
have *13*: $1/3 + 1/3 + 1/3 = (1$::*real*$)$ **by** *simp*
have $(1/3) *_R (a - of\_real\ e) + (1/3) *_R c + (1/3) *_R (a + of\_real\ e) \in$ *interior*(*convex hull* $\{a - e, c, a + e\}$)

```
    using interior_convex_hull_3_minimal [OF * DIM_complex]
    by clarsimp (metis 13 zero_less_divide_1_iff zero_less_numeral)
  then obtain z where z: z ∈ interior(convex hull {a − e, c, a + e}) by force
  have [simp]: z ∉ closed_segment (a − e) c
    by (metis DIM_complex Diff_iff IntD2 inf_sup_absorb interior_of_triangle
z)
  have [simp]: z ∉ closed_segment (a + e) (a − e)
    by (metis DIM_complex DiffD2 Un_iff interior_of_triangle z)
  have [simp]: z ∉ closed_segment c (a + e)
   by (metis (no_types, lifting) DIM_complex DiffD2 Un_insert_right inf_sup_aci(5)
insertCI interior_of_triangle mk_disjoint_insert z)
  show thesis
  proof
   have norm (winding_number (linepath (a − e) c +++ linepath c (a + e) +++
linepath (a + e) (a − e)) z) = 1
     using winding_number_triangle [OF z] by simp
   have zin: z ∈ inside (path_image (linepath (a + e) (a − e)) ∪ path_image p)
    and zeq: winding_number (linepath (a + e) (a − e) +++ reversepath p) z =
             winding_number (linepath (a − e) c +++ linepath c (a + e) +++
linepath (a + e) (a − e)) z
   proof (rule winding_number_from_innerpath
       [of linepath (a + e) (a − e) a+e a−e p
         linepath (a + e) c +++ linepath c (a − e) z
           winding_number (linepath (a − e)  c +++ linepath  c (a + e) +++
linepath (a + e) (a − e)) z])
     show sp_aec: simple_path (linepath (a + e) c +++ linepath c (a − e))
     proof (rule arc_imp_simple_path [OF arc_join])
       show arc (linepath (a + e) c)
        by (metis ‹c ∉ path_image p› arc_linepath pathstart_in_path_image psp)
       show arc (linepath c (a − e))
        by (metis ‹c ∉ path_image p› arc_linepath pathfinish_in_path_image pfp)
       show path_image (linepath (a + e) c) ∩ path_image (linepath c (a − e))
⊆ {pathstart (linepath c (a − e))}
         by clarsimp (metis * IntI Int_closed_segment closed_segment_commute
singleton_iff)
     qed auto
     show simple_path p
       using ‹arc p› arc_simple_path by blast
     show sp_ae2: simple_path (linepath (a + e) (a − e))
       using ‹arc p› arc_distinct_ends pfp psp by fastforce
     show pa: pathfinish (linepath (a + e) (a − e)) = a − e
         pathstart (linepath (a + e) c +++ linepath c (a − e)) = a + e
         pathfinish (linepath (a + e) c +++ linepath c (a − e)) = a − e
         pathstart p = a + e pathfinish p = a − e
         pathstart (linepath (a + e) (a − e)) = a + e
       by (simp_all add: assms)
     show 1: path_image (linepath (a + e) (a − e)) ∩ path_image p = {a + e,
a − e}
     proof
```

show *path_image* (*linepath* (*a* + *e*) (*a* − *e*)) ∩ *path_image p* ⊆ {*a* + *e*, *a* − *e*}
using *pap closed_segment_commute psp segment_convex_hull* **by** *fastforce*
show {*a* + *e*, *a* − *e*} ⊆ *path_image* (*linepath* (*a* + *e*) (*a* − *e*)) ∩ *path_image p*
using *pap pathfinish_in_path_image pathstart_in_path_image pfp psp* **by** *fastforce*
qed
show *2*: *path_image* (*linepath* (*a* + *e*) (*a* − *e*)) ∩ *path_image* (*linepath* (*a* + *e*) *c* +++ *linepath c* (*a* − *e*)) =
{*a* + *e*, *a* − *e*} (**is** *?lhs* = *?rhs*)
proof
have ¬ *collinear* {*c*, *a* + *e*, *a* − *e*}
using ∗ **by** (*simp add*: *insert_commute*)
then have *convex hull* {*a* + *e*, *a* − *e*} ∩ *convex hull* {*a* + *e*, *c*} = {*a* + *e*}
convex hull {*a* + *e*, *a* − *e*} ∩ *convex hull* {*c*, *a* − *e*} = {*a* − *e*}
by (*metis* (*full_types*) *Int_closed_segment insert_commute segment_convex_hull*)+
then show *?lhs* ⊆ *?rhs*
by (*metis Int_Un_distrib equalityD1 insert_is_Un path_image_join path_image_linepath path_join_eq path_linepath segment_convex_hull simple_path_def sp_aec*)
show *?rhs* ⊆ *?lhs*
using *segment_convex_hull* **by** (*simp add*: *path_image_join*)
qed
have *path_image p* ∩ *path_image* (*linepath* (*a* + *e*) *c*) ⊆ {*a* + *e*}
proof (*clarsimp simp*: *path_image_join*)
fix *x*
assume *x* ∈ *path_image p* **and** *x_ac*: *x* ∈ *closed_segment* (*a* + *e*) *c*
then have *dist x a* ≥ *e*
by (*metis IntI all_not_in_conv disj dist_commute mem_ball not_less*)
with *x_ac dac* ‹*e* > *0*› **show** *x* = *a* + *e*
by (*auto simp*: *norm_minus_commute dist_norm closed_segment_eq_open dest*: *open_segment_furthest_le* [**where** *y*=*a*])
qed
moreover
have *path_image p* ∩ *path_image* (*linepath c* (*a* − *e*)) ⊆ {*a* − *e*}
proof (*clarsimp simp*: *path_image_join*)
fix *x*
assume *x* ∈ *path_image p* **and** *x_ac*: *x* ∈ *closed_segment c* (*a* − *e*)
then have *dist x a* ≥ *e*
by (*metis IntI all_not_in_conv disj dist_commute mem_ball not_less*)
with *x_ac dac* ‹*e* > *0*› **show** *x* = *a* − *e*
by (*auto simp*: *norm_minus_commute dist_norm closed_segment_eq_open dest*: *open_segment_furthest_le* [**where** *y*=*a*])
qed
ultimately
have *path_image p* ∩ *path_image* (*linepath* (*a* + *e*) *c* +++ *linepath c* (*a* − *e*)) ⊆ {*a* + *e*, *a* − *e*}
by (*force simp*: *path_image_join*)

then show *3*: *path_image p ∩ path_image (linepath (a + e) c +++ linepath c (a − e)) = {a + e, a − e}*
using *1 2* **by** *blast*
show *4*: *path_image (linepath (a + e) c +++ linepath c (a − e)) ∩ inside (path_image (linepath (a + e) (a − e)) ∪ path_image p) ≠ {}*
apply (*clarsimp simp*: *path_image_join segment_convex_hull disjoint_iff_not_equal*)
by (*metis* (*no_types*, *opaque_lifting*) *UnI1 Un_commute c closed_segment_commute ends_in_segment*(*1*) *path_image_join path_image_linepath pathstart_linepath pfp segment_convex_hull*)
show *zin_inside*: *z ∈ inside (path_image (linepath (a + e) (a − e)) ∪ path_image (linepath (a + e) c +++ linepath c (a − e)))*
proof (*simp add*: *path_image_join*)
show *z ∈ inside (closed_segment (a + e) (a − e) ∪ (closed_segment (a + e) c ∪ closed_segment c (a − e)))*
by (*metis z inside_of_triangle DIM_complex Un_commute closed_segment_commute*)
qed
show *5*: *winding_number (linepath (a + e) (a − e) +++ reversepath (linepath (a + e) c +++ linepath c (a − e))) z = winding_number (linepath (a − e) c +++ linepath c (a + e) +++ linepath (a + e) (a − e)) z*
by (*simp add*: *reversepath_joinpaths path_image_join winding_number_join*)
show *6*: *winding_number (linepath (a − e) c +++ linepath c (a + e) +++ linepath (a + e) (a − e)) z ≠ 0*
by (*simp add*: *winding_number_triangle z*)
show *winding_number (linepath (a + e) (a − e) +++ reversepath p) z = winding_number (linepath (a − e) c +++ linepath c (a + e) +++ linepath (a + e) (a − e)) z*
by (*metis 1 2 3 4 5 6 pa sp_aec sp_ae2* ‹*arc p*› ‹*simple_path p*› *arc_distinct_ends winding_number_from_innerpath zin_inside*)
qed (*use assms* ‹*e > 0*› **in** *auto*)
show *z_inside*: *z ∈ inside (path_image ?pae)*
using *zin* **by** (*simp add*: *assms path_image_join Un_commute closed_segment_commute*)
have *cmod (winding_number ?pae z) = cmod ((winding_number*(*reversepath ?pae*) *z*))
proof (*subst winding_number_reversepath*)
show *path ?pae*
using *simple_path_imp_path sp_pl* **by** *blast*
show *z ∉ path_image ?pae*
by (*metis IntI emptyE inside_no_overlap z_inside*)
qed (*simp add*: *inside_def*)
also have *... = cmod (winding_number*(*linepath (a + e) (a − e) +++ reversepath p*) *z*)
by (*simp add*: *pfp reversepath_joinpaths*)
also have *... = cmod (winding_number (linepath (a − e) c +++ linepath c (a + e) +++ linepath (a + e) (a − e)) z)*
by (*simp add*: *zeq*)
also have *... = 1*

```
    using z by (simp add: interior_of_triangle winding_number_triangle)
   finally show cmod (winding_number ?pae z) = 1 .
 qed
qed

lemma simple_closed_path_wn2:
 fixes a::complex and d e::real
 assumes 0 < d 0 < e
   and sp_pl: simple_path(p +++ linepath (a - d) (a + e))
   and psp:   pathstart p = a + e
   and pfp:   pathfinish p = a - d
obtains z where z ∈ inside (path_image (p +++ linepath (a - d) (a + e)))
                cmod (winding_number (p +++ linepath (a - d) (a + e)) z) = 1
proof -
  have [simp]: a + of_real x ∈ closed_segment (a - α) (a - β) ⟷ x ∈
closed_segment (-α) (-β) for x α β::real
   using closed_segment_translation_eq [of a]
    by (metis (no_types, opaque_lifting) add_uminus_conv_diff of_real_minus
of_real_closed_segment)
  have [simp]: a - of_real x ∈ closed_segment (a + α) (a + β) ⟷ -x ∈
closed_segment α β for x α β::real
  by (metis closed_segment_translation_eq diff_conv_add_uminus of_real_closed_segment
of_real_minus)
 have arc p and arc_lp: arc (linepath (a - d) (a + e)) and path p
   and pap: path_image p ∩ closed_segment (a - d) (a + e) ⊆ {a+e, a-d}
    using simple_path_join_loop_eq [of linepath (a - d) (a + e) p] assms
arc_imp_path by auto
 have 0 ∈ closed_segment (-d) e
   using ‹0 < d› ‹0 < e› closed_segment_eq_real_ivl by auto
 then have a ∈ path_image (linepath (a - d) (a + e))
   using of_real_closed_segment [THEN iffD2]
    by (force dest: closed_segment_translation_eq [of a, THEN iffD2] simp del:
of_real_closed_segment)
 then have a ∉ path_image p
   using ‹0 < d› ‹0 < e› pap by auto
 then obtain k where 0 < k and k: ball a k ∩ (path_image p) = {}
   using ‹0 < e› ‹path p› not_on_path_ball by blast
 define kde where kde ≡ (min k (min d e)) / 2
 have 0 < kde kde < k kde < d kde < e
   using ‹0 < k› ‹0 < d› ‹0 < e› by (auto simp: kde_def)
 let ?q = linepath (a + kde) (a + e) +++ p +++ linepath (a - d) (a - kde)
 have - kde ∈ closed_segment (-d) e
   using ‹0 < kde› ‹kde < d› ‹kde < e› closed_segment_eq_real_ivl by auto
 then have a_diff_kde: a - kde ∈ closed_segment (a - d) (a + e)
   using of_real_closed_segment [THEN iffD2]
    by (force dest: closed_segment_translation_eq [of a, THEN iffD2] simp del:
of_real_closed_segment)
 then have clsub2: closed_segment (a - d) (a - kde) ⊆ closed_segment (a -
d) (a + e)
```

```
    by (simp add: subset_closed_segment)
  then have path_image p ∩ closed_segment (a − d) (a − kde) ⊆ {a + e, a −
d}
    using pap by force
  moreover
  have a + e ∉ path_image p ∩ closed_segment (a − d) (a − kde)
   using ‹0 < kde› ‹kde < d› ‹0 < e› by (auto simp: closed_segment_eq_real_ivl)
  ultimately have sub_a_diff_d: path_image p ∩ closed_segment (a − d) (a −
kde) ⊆ {a − d}
    by blast
  have kde ∈ closed_segment (−d) e
    using ‹0 < kde› ‹kde < d› ‹kde < e› closed_segment_eq_real_ivl by auto
  then have a_diff_kde: a + kde ∈ closed_segment (a − d) (a + e)
    using of_real_closed_segment [THEN iffD2]
    by (force dest: closed_segment_translation_eq [of a, THEN iffD2] simp del:
of_real_closed_segment)
  then have clsub1: closed_segment (a + kde) (a + e) ⊆ closed_segment (a −
d) (a + e)
    by (simp add: subset_closed_segment)
  then have closed_segment (a + kde) (a + e) ∩ path_image p ⊆ {a + e, a −
d}
    using pap by force
  moreover
  have closed_segment (a + kde) (a + e) ∩ closed_segment (a − d) (a − kde) =
{}
  proof (clarsimp intro!: equals0I)
    fix y
    assume y1: y ∈ closed_segment (a + kde) (a + e)
       and y2: y ∈ closed_segment (a − d) (a − kde)
    obtain u::real where u: y = a + u and 0 < u
    proof −
      obtain ξ where ξ: y = (1 − ξ) *R (a + kde) + ξ *R (a + e) and 0 ≤ ξ ξ
≤ 1
        using y1 by (auto simp: in_segment)
      show thesis
      proof
        show y = a + ((1 − ξ)*kde + ξ*e)
          using ξ by (auto simp: scaleR_conv_of_real algebra_simps)
        have (1 − ξ)*kde + ξ*e ≥ 0
          using ‹0 < kde› ‹0 ≤ ξ› ‹ξ ≤ 1› ‹0 < e› by auto
        moreover have (1 − ξ)*kde + ξ*e ≠ 0
       using ‹0 < kde› ‹0 ≤ ξ› ‹ξ ≤ 1› ‹0 < e› by (auto simp: add_nonneg_eq_0_iff)
        ultimately show (1 − ξ)*kde + ξ*e > 0 by simp
      qed
    qed
    moreover
    obtain v::real where v: y = a + v and v ≤ 0
    proof −
      obtain ξ where ξ: y = (1 − ξ) *R (a − d) + ξ *R (a − kde) and 0 ≤ ξ ξ
```

```
≤ 1
      using y2 by (auto simp: in_segment)
    show thesis
    proof
      show y = a + (- ((1 - ξ)*d + ξ*kde))
        using ξ by (force simp: scaleR_conv_of_real algebra_simps)
      show - ((1 - ξ)*d + ξ*kde) ≤ 0
        using ‹0 < kde› ‹0 ≤ ξ› ‹ξ ≤ 1› ‹0 < d›
     by (metis add.left_neutral add_nonneg_nonneg le_diff_eq less_eq_real_def
neg_le_0_iff_le split_mult_pos_le)

    qed
  qed
  ultimately show False
    by auto
 qed
 moreover have a - d ∉ closed_segment (a + kde) (a + e)
  using ‹0 < kde› ‹kde < d› ‹0 < e› by (auto simp: closed_segment_eq_real_ivl)
 ultimately have sub_a_plus_e:
  closed_segment (a + kde) (a + e) ∩ (path_image p ∪ closed_segment (a - d)
(a - kde)) ⊆ {a + e}
  by auto
 have kde ∈ closed_segment (-kde) e
  using ‹0 < kde› ‹kde < d› ‹kde < e› closed_segment_eq_real_ivl by auto
 then have a_add_kde: a + kde ∈ closed_segment (a - kde) (a + e)
  using of_real_closed_segment [THEN iffD2]
  by (force dest: closed_segment_translation_eq [of a, THEN iffD2] simp del:
of_real_closed_segment)
 have closed_segment (a - kde) (a + kde) ∩ closed_segment (a + kde) (a + e)
= {a + kde}
  by (metis a_add_kde Int_closed_segment)
 moreover
 have path_image p ∩ closed_segment (a - kde) (a + kde) = {}
 proof (rule equals0I, clarify)
  fix y assume y ∈ path_image p y ∈ closed_segment (a - kde) (a + kde)
  with equals0D [OF k, of y] ‹0 < kde› ‹kde < k› show False
   by (auto simp: dist_norm dest: dist_decreases_closed_segment [where c=a])
 qed
 moreover
 have - kde ∈ closed_segment (-d) kde
  using ‹0 < kde› ‹kde < d› ‹kde < e› closed_segment_eq_real_ivl by auto
 then have a_diff_kde': a - kde ∈ closed_segment (a - d) (a + kde)
  using of_real_closed_segment [THEN iffD2]
  by (force dest: closed_segment_translation_eq [of a, THEN iffD2] simp del:
of_real_closed_segment)
 then have closed_segment (a - d) (a - kde) ∩ closed_segment (a - kde) (a +
kde) = {a - kde}
  by (metis Int_closed_segment)
 ultimately
```

have *pa_subset_pm_kde*: *path_image ?q* ∩ *closed_segment* (*a* − *kde*) (*a* + *kde*) ⊆ {*a* − *kde*, *a* + *kde*}
  by (*auto simp*: *path_image_join assms*)
have *ge_kde1*: ∃ *y*. *x* = *a* + *of_real y* ∧ *y* ≥ *kde* **if** *x*: *x* ∈ *closed_segment* (*a* + *kde*) (*a* + *e*) **for** *x*
proof −
  obtain *u* **where** *0* ≤ *u u* ≤ *1* **and** *u*: *x* = (*1* − *u*) $*_R$ (*a* + *kde*) + *u* $*_R$ (*a* + *e*)
    using *x* **by** (*auto simp*: *in_segment*)
  then have *kde* ≤ (*1* − *u*) ∗ *kde* + *u* ∗ *e*
    using ‹*kde* < *e*› *segment_bound_lemma* **by** *auto*
  moreover have *x* = *a* + ((*1* − *u*) ∗ *kde* + *u* ∗ *e*)
    by (*fastforce simp*: *u algebra_simps scaleR_conv_of_real*)
  ultimately
  show *?thesis* **by** *blast*
qed
have *ge_kde2*: ∃ *y*. *x* = *a* + *of_real y* ∧ *y* ≤ −*kde* **if** *x*: *x* ∈ *closed_segment* (*a* − *d*) (*a* − *kde*) **for** *x*
proof −
  obtain *u* **where** *0* ≤ *u u* ≤ *1* **and** *u*: *x* = (*1* − *u*) $*_R$ (*a* − *d*) + *u* $*_R$ (*a* − *kde*)
    using *x* **by** (*auto simp*: *in_segment*)
  then have *kde* ≤ ((*1*−*u*)∗*d* + *u*∗*kde*)
    using ‹*kde* < *d*› *segment_bound_lemma* **by** *auto*
  moreover have *x* = *a* − ((*1*−*u*)∗*d* + *u*∗*kde*)
    by (*fastforce simp*: *u algebra_simps scaleR_conv_of_real*)
  ultimately show *?thesis*
    by (*metis add_uminus_conv_diff neg_le_iff_le of_real_minus*)
qed
have *notin_paq*: *x* ∉ *path_image ?q* **if** *dist a x* < *kde* **for** *x*
proof −
  have *x* ∉ *path_image p*
    using *k* ‹*kde* < *k*› *that* **by** *force*
  then show *?thesis*
    using *that assms* **by** (*auto simp*: *path_image_join dist_norm dest*!: *ge_kde1 ge_kde2*)
qed
obtain *z* **where** *zin*: *z* ∈ *inside* (*path_image* (*?q* +++ *linepath* (*a* − *kde*) (*a* + *kde*)))
    and *z1*: *cmod* (*winding_number* (*?q* +++ *linepath* (*a* − *kde*) (*a* + *kde*)) *z*) = *1*
proof (*rule simple_closed_path_wn1* [*of kde ?q a*])
  show *simple_path* (*?q* +++ *linepath* (*a* − *kde*) (*a* + *kde*))
  proof (*intro simple_path_join_loop conjI*)
    show *arc ?q*
    proof (*rule arc_join*)
      show *arc* (*linepath* (*a* + *kde*) (*a* + *e*))
        using ‹*kde* < *e*› ‹*arc p*› **by** (*force simp*: *pfp*)
      show *arc* (*p* +++ *linepath* (*a* − *d*) (*a* − *kde*))

```
      using ‹kde < d› ‹kde < e› ‹arc p› sub_a_diff_d by (force simp: pfp intro:
arc_join)
    qed (auto simp: psp pfp path_image_join sub_a_plus_e)
    show arc (linepath (a − kde) (a + kde))
      using ‹0 < kde› by auto
  qed (use pa_subset_pm_kde in auto)
 qed (use ‹0 < kde› notin_paq in auto)
 have eq: path_image (?q +++ linepath (a − kde) (a + kde)) = path_image (p
+++ linepath (a − d) (a + e))
        (is ?lhs = ?rhs)
 proof
  show ?lhs ⊆ ?rhs
    using clsub1 clsub2 apply (auto simp: path_image_join assms)
    by (meson subsetCE subset_closed_segment)
  show ?rhs ⊆ ?lhs
    apply (simp add: path_image_join assms Un_ac)
     by (metis Un_closed_segment Un_assoc a_diff_kde a_diff_kde' le_supI2
subset_refl)
  qed
 show thesis
 proof
  show zzin: z ∈ inside (path_image (p +++ linepath (a − d) (a + e)))
    by (metis eq zin)
  then have znotin: z ∉ path_image p
     by (metis ComplD Un_iff inside_Un_outside path_image_join pathfin-
ish_linepath pathstart_reversepath pfp reversepath_linepath)
  have znotin_d_kde: z ∉ closed_segment (a − d) (a + kde)
   by (metis ComplD Un_iff Un_closed_segment a_diff_kde inside_Un_outside
path_image_join path_image_linepath pathstart_linepath pfp zzin)
  have znotin_d_e: z ∉ closed_segment (a − d) (a + e)
   by (metis IntI UnCI emptyE inside_no_overlap path_image_join path_image_linepath
pathstart_linepath pfp zzin)
  have znotin_kde_e: z ∉ closed_segment (a + kde) (a + e) and znotin_d_kde':
z ∉ closed_segment (a − d) (a − kde)
    using clsub1 clsub2 zzin
    by (metis (no_types, opaque_lifting) IntI Un_iff emptyE inside_no_overlap
path_image_join path_image_linepath pathstart_linepath pfp subsetD)+
  have winding_number (linepath (a − d) (a + e)) z =
        winding_number (linepath (a − d) (a + kde)) z + winding_number
(linepath (a + kde) (a + e)) z
  proof (rule winding_number_split_linepath)
    show a + complex_of_real kde ∈ closed_segment (a − d) (a + e)
      by (simp add: a_diff_kde)
    show z ∉ closed_segment (a − d) (a + e)
    by (metis ComplD Un_iff inside_Un_outside path_image_join path_image_linepath
pathstart_linepath pfp zzin)
  qed
  also have ... = winding_number (linepath (a + kde) (a + e)) z +
            (winding_number (linepath (a − d) (a − kde)) z + winding_number
```

(*linepath* (*a* − *kde*) (*a* + *kde*)) *z*)
by (*simp add*: *winding_number_split_linepath* [*of a*−*kde*, *symmetric*] *znotin_d_kde a_diff_kde'*)
finally have *winding_number* (*p* +++ *linepath* (*a* − *d*) (*a* + *e*)) *z* =
winding_number p z + *winding_number* (*linepath* (*a* + *kde*) (*a* + *e*)) *z* +
(*winding_number* (*linepath* (*a* − *d*) (*a* − *kde*)) *z* +
winding_number (*linepath* (*a* − *kde*) (*a* + *kde*)) *z*)
by (*metis* (*no_types*, *lifting*) *ComplD Un_iff* ‹*arc p*› *add.assoc arc_imp_path eq path_image_join path_join_path_ends path_linepath simple_path_imp_path sp_pl union_with_outside winding_number_join zin*)
also have ... = *winding_number* (*linepath* (*a* + *kde*) (*a* + *e*)) *z*
+ *winding_number* (*p* +++ *linepath* (*a* − *d*) (*a* − *kde*)) *z*
+ *winding_number* (*linepath* (*a* − *kde*) (*a* + *kde*)) *z*
using ‹*path p*› *znotin assms*
by *simp* (*metis Un_iff Un_closed_segment a_diff_kde' path_image_linepath path_linepath pathstart_linepath winding_number_join znotin_d_kde*)
also have ... = *winding_number ?q z* + *winding_number* (*linepath* (*a* − *kde*) (*a* + *kde*)) *z*
using ‹*path p*› *znotin assms* **by** (*simp add*: *path_image_join winding_number_join znotin_kde_e znotin_d_kde'*)
also have ... = *winding_number* (*?q* +++ *linepath* (*a* − *kde*) (*a* + *kde*)) *z*
by (*metis* (*mono_tags*, *lifting*) *ComplD UnCI* ‹*path p*› *eq inside_outside path_image_join path_join_eq path_linepath pathfinish_join pathfinish_linepath pathstart_join pathstart_linepath pfp psp winding_number_join zzin*)
finally have *winding_number* (*p* +++ *linepath* (*a* − *d*) (*a* + *e*)) *z* =
winding_number (*?q* +++ *linepath* (*a* − *kde*) (*a* + *kde*)) *z* **.**
then show *cmod* (*winding_number* (*p* +++ *linepath* (*a* − *d*) (*a* + *e*)) *z*) = *1*
by (*simp add*: *z1*)
qed
qed

lemma *simple_closed_path_wn3*:
fixes *p* :: *real* ⇒ *complex*
assumes *simple_path p* **and** *loop*: *pathfinish p* = *pathstart p*
obtains *z* **where** *z* ∈ *inside* (*path_image p*) *cmod* (*winding_number p z*) = *1*
proof −
have *ins*: *inside*(*path_image p*) ≠ {} *open*(*inside*(*path_image p*))
connected(*inside*(*path_image p*))
and *out*: *outside*(*path_image p*) ≠ {} *open*(*outside*(*path_image p*))
connected(*outside*(*path_image p*))
and *bo*: *bounded*(*inside*(*path_image p*)) ¬ *bounded*(*outside*(*path_image p*))
and *ins_out*: *inside*(*path_image p*) ∩ *outside*(*path_image p*) = {}
inside(*path_image p*) ∪ *outside*(*path_image p*) = − *path_image p*
and *fro*: *frontier*(*inside*(*path_image p*)) = *path_image p*
frontier(*outside*(*path_image p*)) = *path_image p*
using *Jordan_inside_outside* [*OF assms*] **by** *auto*
obtain *a* **where** *a*: *a* ∈ *inside*(*path_image p*)
using ‹*inside* (*path_image p*) ≠ {}› **by** *blast*

obtain *d*::*real* **where** $0 < d$ **and** *d_fro*: $a - d \in$ *frontier* (*inside* (*path_image p*))
and *d_int*: $\bigwedge\varepsilon.\ [\![0 \le \varepsilon; \varepsilon < d]\!] \Longrightarrow (a - \varepsilon) \in$ *inside* (*path_image p*)
using *ray_to_frontier* [*of inside* (*path_image p*) *a* −*1*] *bo ins a interior_eq*
by (*metis ab_group_add_class.ab_diff_conv_add_uminus of_real_def scale_minus_right zero_neq_neg_one*)
obtain *e*::*real* **where** $0 < e$ **and** *e_fro*: $a + e \in$ *frontier* (*inside* (*path_image p*))
and *e_int*: $\bigwedge\varepsilon.\ [\![0 \le \varepsilon; \varepsilon < e]\!] \Longrightarrow (a + \varepsilon) \in$ *inside* (*path_image p*)
using *ray_to_frontier* [*of inside* (*path_image p*) *a 1*] *bo ins a interior_eq*
by (*metis of_real_def zero_neq_one*)
obtain *t0* **where** $0 \le t0\ t0 \le 1$ **and** *pt*: $p\ t0 = a - d$
using *a d_fro fro* **by** (*auto simp*: *path_image_def*)
obtain *q* **where** *simple_path q* **and** *q_ends*: *pathstart q* $= a - d$ *pathfinish q* $= a - d$
and *q_eq_p*: *path_image q* = *path_image p*
and *wn_q_eq_wn_p*: $\bigwedge z.\ z \in$ *inside*(*path_image p*) $\Longrightarrow$ *winding_number q z* = *winding_number p z*
proof
show *simple_path* (*shiftpath t0 p*)
by (*simp add*: *pathstart_shiftpath pathfinish_shiftpath*
simple_path_shiftpath path_image_shiftpath ‹*0* ≤ *t0*› ‹*t0* ≤ *1*› *assms*)
show *pathstart* (*shiftpath t0 p*) = *a* − *d*
using *pt* **by** (*simp add*: ‹*t0* ≤ *1*› *pathstart_shiftpath*)
show *pathfinish* (*shiftpath t0 p*) = *a* − *d*
by (*simp add*: ‹*0* ≤ *t0*› *loop pathfinish_shiftpath pt*)
show *path_image* (*shiftpath t0 p*) = *path_image p*
by (*simp add*: ‹*0* ≤ *t0*› ‹*t0* ≤ *1*› *loop path_image_shiftpath*)
show *winding_number* (*shiftpath t0 p*) *z* = *winding_number p z*
if *z* ∈ *inside* (*path_image p*) **for** *z*
by (*metis ComplD Un_iff* ‹*0* ≤ *t0*› ‹*t0* ≤ *1*› ‹*simple_path p*› *atLeastAtMost_iff inside_Un_outside*
loop simple_path_imp_path that winding_number_shiftpath)
qed
have *ad_not_ae*: $a - d \neq a + e$
by (*metis* ‹*0* < *d*› ‹*0* < *e*› *add.left_inverse add_left_cancel add_uminus_conv_diff*
le_add_same_cancel2 less_eq_real_def not_less of_real_add of_real_def of_real_eq_0_iff pt)
have *ad_ae_q*: $\{a - d, a + e\} \subseteq$ *path_image q*
using ‹*path_image q* = *path_image p*› *d_fro e_fro fro*(*1*) **by** *auto*
have *ada*: *open_segment* (*a* − *d*) *a* ⊆ *inside* (*path_image p*)
proof (*clarsimp simp*: *in_segment*)
fix *u*::*real* **assume** *0* < *u u* < *1*
with *d_int* **have** $a - (1 - u) * d \in$ *inside* (*path_image p*)
by (*metis* ‹*0* < *d*› *add.commute diff_add_cancel left_diff_distrib' less_add_same_cancel2 less_eq_real_def mult.left_neutral zero_less_mult_iff*)
then show (*1* − *u*) *R* (*a* − *d*) + *u R a* ∈ *inside* (*path_image p*)
by (*simp add*: *diff_add_eq of_real_def real_vector.scale_right_diff_distrib*)
qed

```
  have aae: open_segment a (a + e) ⊆ inside (path_image p)
  proof (clarsimp simp: in_segment)
    fix u::real assume 0 < u u < 1
    with e_int have a + u * e ∈ inside (path_image p)
        by (meson ‹0 < e› less_eq_real_def mult_less_cancel_right2 not_less
zero_less_mult_iff)
    then show (1 - u) *R a + u *R (a + e) ∈ inside (path_image p)
     by (metis (mono_tags, lifting) add.assoc of_real_mult pth_6 scaleR_collapse
scaleR_conv_of_real)
  qed
  have complex_of_real (d * d + (e * e + d * (e + e))) ≠ 0
    using ad_not_ae
     by (metis ‹0 < d› ‹0 < e› add_strict_left_mono less_add_same_cancel1
not_sum_squares_lt_zero
      of_real_eq_0_iff zero_less_double_add_iff_zero_less_single_add zero_less_mult_iff)
  moreover have ∃u>0. u < 1 ∧ d = u * d + u * e
    using ‹0 < d› ‹0 < e›
   by (rule_tac x=d / (d+e) in exI) (simp add: divide_simps scaleR_conv_of_real
flip: distrib_left)
  ultimately have a_in_de: a ∈ open_segment (a - d) (a + e)
   using ad_not_ae by (simp add: in_segment algebra_simps scaleR_conv_of_real
flip: of_real_add of_real_mult)
  then have open_segment (a - d) (a + e) ⊆ inside (path_image p)
    using ada a aae Un_open_segment [of a a-d a+e] by blast
  then have path_image q ∩ open_segment (a - d) (a + e) = {}
    using inside_no_overlap by (fastforce simp: q_eq_p)
  with ad_ae_q have paq_Int_cs: path_image q ∩ closed_segment (a - d) (a +
e) = {a - d, a + e}
    by (simp add: closed_segment_eq_open)
  obtain t where 0 ≤ t t ≤ 1 and qt: q t = a + e
    using a e_fro fro ad_ae_q by (auto simp: path_defs)
  then have t ≠ 0
    by (metis ad_not_ae pathstart_def q_ends(1))
  then have t ≠ 1
    by (metis ad_not_ae pathfinish_def q_ends(2) qt)
  have q01: q 0 = a - d q 1 = a - d
    using q_ends by (auto simp: pathstart_def pathfinish_def)
  obtain z where zin: z ∈ inside (path_image (subpath t 0 q +++ linepath (a -
d) (a + e)))
            and z1: cmod (winding_number (subpath t 0 q +++ linepath (a - d)
(a + e)) z) = 1
  proof (rule simple_closed_path_wn2 [of d e subpath t 0 q a], simp_all add: q01)
    show simple_path (subpath t 0 q +++ linepath (a - d) (a + e))
    proof (rule simple_path_join_loop, simp_all add: qt q01)
      have inj_on q (closed_segment t 0)
        using ‹0 ≤ t› ‹simple_path q› ‹t ≤ 1› ‹t ≠ 0› ‹t ≠ 1›
      by (fastforce simp: simple_path_def inj_on_def closed_segment_eq_real_ivl)
      then show arc (subpath t 0 q)
        using ‹0 ≤ t› ‹simple_path q› ‹t ≤ 1› ‹t ≠ 0›
```

by (*simp add*: *arc_subpath_eq simple_path_imp_path*)
show *arc* (*linepath* (*a* − *d*) (*a* + *e*))
by (*simp add*: *ad_not_ae*)
show *path_image* (*subpath t 0 q*) ∩ *closed_segment* (*a* − *d*) (*a* + *e*) ⊆ {*a* + *e*, *a* − *d*}
using *qt paq_Int_cs* ‹*simple_path q*› ‹0 ≤ *t*› ‹*t* ≤ 1›
by (*force simp*: *dest*: *rev_subsetD* [*OF* _ *path_image_subpath_subset*] *intro*: *simple_path_imp_path*)
qed
qed (*auto simp*: ‹0 < *d*› ‹0 < *e*› *qt*)
have *pa01_Un*: *path_image* (*subpath 0 t q*) ∪ *path_image* (*subpath 1 t q*) = *path_image q*
unfolding *path_image_subpath*
using ‹0 ≤ *t*› ‹*t* ≤ 1› **by** (*force simp*: *path_image_def image_iff*)
with *paq_Int_cs* **have** *pa_01q*:
(*path_image* (*subpath 0 t q*) ∪ *path_image* (*subpath 1 t q*)) ∩ *closed_segment* (*a* − *d*) (*a* + *e*) = {*a* − *d*, *a* + *e*}
by *metis*
have *z_notin_ed*: *z* ∉ *closed_segment* (*a* + *e*) (*a* − *d*)
using *zin q01* **by** (*simp add*: *path_image_join closed_segment_commute inside_def*)
have *z_notin_0t*: *z* ∉ *path_image* (*subpath 0 t q*)
by (*metis* (*no_types*, *opaque_lifting*) *IntI Un_upper1 subsetD empty_iff inside_no_overlap path_image_join*
path_image_subpath_commute pathfinish_subpath pathstart_def pathstart_linepath q_ends(1) *qt subpath_trivial zin*)
have [*simp*]: − *winding_number* (*subpath t 0 q*) *z* = *winding_number* (*subpath 0 t q*) *z*
by (*metis* ‹0 ≤ *t*› ‹*simple_path q*› ‹*t* ≤ 1› *atLeastAtMost_iff zero_le_one*
path_image_subpath_commute path_subpath real_eq_0_iff_le_ge_0
reversepath_subpath simple_path_imp_path winding_number_reversepath z_notin_0t)
obtain *z_in_q*: *z* ∈ *inside*(*path_image q*)
and *wn_q*: *winding_number* (*subpath 0 t q* +++ *subpath t 1 q*) *z* = − *winding_number* (*subpath t 0 q* +++ *linepath* (*a* − *d*) (*a* + *e*)) *z*
proof (*rule winding_number_from_innerpath*
[*of subpath 0 t q a−d a+e subpath 1 t q linepath* (*a* − *d*) (*a* + *e*)
z − *winding_number* (*subpath t 0 q* +++ *linepath* (*a* − *d*) (*a* + *e*)) *z*],
simp_all add: *q01 qt pa01_Un reversepath_subpath*)
show *simple_path* (*subpath 0 t q*) *simple_path* (*subpath 1 t q*)
by (*simp_all add*: ‹0 ≤ *t*› ‹*simple_path q*› ‹*t* ≤ 1› ‹*t* ≠ 0› ‹*t* ≠ 1› *simple_path_subpath*)
show *simple_path* (*linepath* (*a* − *d*) (*a* + *e*))
using *ad_not_ae* **by** *blast*
show *path_image* (*subpath 0 t q*) ∩ *path_image* (*subpath 1 t q*) = {*a* − *d*, *a* + *e*} (**is** *?lhs* = *?rhs*)
proof
show *?lhs* ⊆ *?rhs*
using ‹0 ≤ *t*› ‹*simple_path q*› ‹*t* ≤ 1› ‹*t* ≠ 1› *q_ends qt q01*

```
      by (force simp: pathfinish_def qt simple_path_def path_image_subpath)
    show ?rhs ⊆ ?lhs
      using ‹0 ≤ t› ‹t ≤ 1› q01 qt by (force simp: path_image_subpath)
  qed
  show path_image (subpath 0 t q) ∩ closed_segment (a − d) (a + e) = {a −
d, a + e} (is ?lhs = ?rhs)
  proof
    show ?lhs ⊆ ?rhs using paq_Int_cs pa01_Un by fastforce
  show ?rhs ⊆ ?lhs using ‹0 ≤ t› ‹t ≤ 1› q01 qt by (force simp: path_image_subpath)
  qed
  show path_image (subpath 1 t q) ∩ closed_segment (a − d) (a + e) = {a −
d, a + e} (is ?lhs = ?rhs)
  proof
    show ?lhs ⊆ ?rhs by (auto simp: pa_01q [symmetric])
  show ?rhs ⊆ ?lhs using ‹0 ≤ t› ‹t ≤ 1› q01 qt by (force simp: path_image_subpath)
  qed
  show closed_segment (a − d) (a + e) ∩ inside (path_image q) ≠ {}
    using a a_in_de open_closed_segment pa01_Un q_eq_p by fastforce
  show z ∈ inside (path_image (subpath 0 t q) ∪ closed_segment (a − d) (a +
e))
    by (metis path_image_join path_image_linepath path_image_subpath_commute
pathfinish_subpath pathstart_linepath q01(1) zin)
  show winding_number (subpath 0 t q +++ linepath (a + e) (a − d)) z =
    − winding_number (subpath t 0 q +++ linepath (a − d) (a + e)) z
    using z_notin_ed z_notin_0t ‹0 ≤ t› ‹simple_path q› ‹t ≤ 1›
    by (simp add: simple_path_imp_path qt q01 path_image_subpath_commute
closed_segment_commute winding_number_join winding_number_reversepath [symmetric])
  show − d ≠ e
    using ad_not_ae by auto
  show winding_number (subpath t 0 q +++ linepath (a − d) (a + e)) z ≠ 0
    using z1 by auto
 qed
 show ?thesis
 proof
  show z ∈ inside (path_image p)
    using q_eq_p z_in_q by auto
  then have [simp]: z ∉ path_image q
    by (metis disjoint_iff_not_equal inside_no_overlap q_eq_p)
  have [simp]: z ∉ path_image (subpath 1 t q)
    using inside_def pa01_Un z_in_q by fastforce
 have winding_number(subpath 0 t q +++ subpath t 1 q) z = winding_number(subpath
0 1 q) z
    using z_notin_0t ‹0 ≤ t› ‹simple_path q› ‹t ≤ 1›
   by (simp add: simple_path_imp_path qt path_image_subpath_commute winding_number_join winding_number_subpath_combine)
  with wn_q have winding_number (subpath t 0 q +++ linepath (a − d) (a +
e)) z = − winding_number q z
    by auto
  with z1 have cmod (winding_number q z) = 1
```

```
    by simp
  with z1 wn_q_eq_wn_p show cmod (winding_number p z) = 1
    using z1 wn_q_eq_wn_p  by (simp add: ‹z ∈ inside (path_image p)›)
  qed
qed

proposition simple_closed_path_winding_number_inside:
  assumes simple_path γ
  obtains ⋀z. z ∈ inside(path_image γ) ⟹ winding_number γ z = 1
        | ⋀z. z ∈ inside(path_image γ) ⟹ winding_number γ z = −1
proof (cases pathfinish γ = pathstart γ)
  case True
  have path γ
    by (simp add: assms simple_path_imp_path)
  then have const: winding_number γ constant_on inside(path_image γ)
  proof (rule winding_number_constant)
    show connected (inside(path_image γ))
      by (simp add: Jordan_inside_outside True assms)
  qed (use inside_no_overlap True in auto)
 obtain z where zin: z ∈ inside (path_image γ) and z1: cmod (winding_number
γ z) = 1
    using simple_closed_path_wn3 [of γ] True assms by blast
  have winding_number γ z ∈ ℤ
    using zin integer_winding_number [OF ‹path γ› True] inside_def by blast
  moreover have real_of_int x = − 1 ⟷ x = −1 for x
    by linarith
  ultimately consider winding_number γ z = 1 | winding_number γ z = −1
    using z1 by (auto simp: Ints_def abs_if split: if_split_asm)
  with that const zin show ?thesis
    unfolding constant_on_def by metis
next
  case False
  then show ?thesis
    using inside_simple_curve_imp_closed assms that(2) by blast
qed

lemma simple_closed_path_abs_winding_number_inside:
  assumes simple_path γ z ∈ inside(path_image γ)
    shows |Re (winding_number γ z)| = 1
 by (metis assms norm_minus_cancel norm_one one_complex.simps(1) real_norm_def
simple_closed_path_winding_number_inside uminus_complex.simps(1))

lemma simple_closed_path_norm_winding_number_inside:
  assumes simple_path γ z ∈ inside(path_image γ)
  shows norm (winding_number γ z) = 1
proof −
  have pathfinish γ = pathstart γ
    using assms inside_simple_curve_imp_closed by blast
  with assms integer_winding_number have winding_number γ z ∈ ℤ
```

by (*simp add*: *inside_def simple_path_def*)
then show *?thesis*
by (*metis assms norm_minus_cancel norm_one simple_closed_path_winding_number_inside*)
qed

lemma *simple_closed_path_winding_number_cases*:
assumes *simple_path* γ *pathfinish* γ = *pathstart* γ $z \notin$ *path_image* γ
shows *winding_number* γ $z \in \{-1,0,1\}$
proof −
consider $z \in$ *inside* (*path_image* γ) | $z \in$ *outside* (*path_image* γ)
by (*metis ComplI UnE assms*(3) *inside_Un_outside*)
then show *?thesis*
proof *cases*
case *1*
then show *?thesis*
using *assms simple_closed_path_winding_number_inside* **by** *auto*
next
case *2*
then show *?thesis*
using *assms simple_path_def winding_number_zero_in_outside* **by** *blast*
qed
qed

lemma *simple_closed_path_winding_number_pos*:
⟦*simple_path* γ; *pathfinish* γ = *pathstart* γ; $z \notin$ *path_image* γ; $0 <$ *Re*(*winding_number* γ z)⟧
$\Longrightarrow$ *winding_number* γ z = *1*
using *simple_closed_path_winding_number_cases*
by *fastforce*

3.9 Winding number for rectangular paths

proposition *winding_number_rectpath*:
assumes $z \in$ *box a1 a3*
shows *winding_number* (*rectpath a1 a3*) z = *1*
proof −
from *assms* **have** *less*: *Re a1* < *Re a3 Im a1* < *Im a3*
by (*auto simp*: *in_box_complex_iff*)
define *a2 a4* **where** *a2* = *Complex* (*Re a3*) (*Im a1*) **and** *a4* = *Complex* (*Re a1*) (*Im a3*)
let *?l1* = *linepath a1 a2* **and** *?l2* = *linepath a2 a3*
and *?l3* = *linepath a3 a4* **and** *?l4* = *linepath a4 a1*
from *assms* **and** *less* **have** $z \notin$ *path_image* (*rectpath a1 a3*)
by (*auto simp*: *path_image_rectpath_cbox_minus_box*)
also have *path_image* (*rectpath a1 a3*) =
path_image ?l1 ∪ *path_image ?l2* ∪ *path_image ?l3* ∪ *path_image ?l4*
by (*simp add*: *rectpath_def Let_def path_image_join Un_assoc a2_def a4_def*)
finally have $z \notin \ldots$ **.**

```
  moreover have ∀ l∈{?l1,?l2,?l3,?l4}. Re (winding_number l z) > 0
    unfolding ball_simps HOL.simp_thms a2_def a4_def
    by (intro conjI; (rule winding_number_linepath_pos_lt;
       (insert assms, auto simp: a2_def a4_def in_box_complex_iff mult_neg_neg))+)
  ultimately have Re (winding_number (rectpath a1 a3) z) > 0
     by (simp add: winding_number_join path_image_join rectpath_def Let_def
a2_def a4_def)
  thus winding_number (rectpath a1 a3) z = 1 using assms less
    by (intro simple_closed_path_winding_number_pos simple_path_rectpath)
       (auto simp: path_image_rectpath_cbox_minus_box)
qed

proposition winding_number_rectpath_outside:
  assumes Re a1 ≤ Re a3 Im a1 ≤ Im a3
  assumes z ∉ cbox a1 a3
  shows   winding_number (rectpath a1 a3) z = 0
  using assms by (intro winding_number_zero_outside[OF _ _ _ assms(3)]
                        path_image_rectpath_subset_cbox) simp_all
```

A per-function version for continuous logs, a kind of monodromy

```
proposition winding_number_compose_exp:
  assumes path p
  shows winding_number (exp ∘ p) 0 = (pathfinish p − pathstart p) / (2 ∗ of_real
pi ∗ ı)
proof −
  obtain e where 0 < e and e: ⋀t. t ∈ {0..1} ⟹ e ≤ norm(exp(p t))
  proof
    have closed (path_image (exp ∘ p))
       by (simp add: assms closed_path_image holomorphic_on_exp holomor-
phic_on_imp_continuous_on path_continuous_image)
    then show 0 < setdist {0} (path_image (exp ∘ p))
    by (metis exp_not_eq_zero imageE image_comp infdist_eq_setdist infdist_pos_not_in_closed
path_defs(4) path_image_nonempty)
  next
    fix t::real
    assume t ∈ {0..1}
    have setdist {0} (path_image (exp ∘ p)) ≤ dist 0 (exp (p t))
    proof (rule setdist_le_dist)
      show exp (p t) ∈ path_image (exp ∘ p)
        using ‹t ∈ {0..1}› path_image_def by fastforce+
    qed auto
    then show setdist {0} (path_image (exp ∘ p)) ≤ cmod (exp (p t))
      by simp
  qed
  have bounded (path_image p)
    by (simp add: assms bounded_path_image)
  then obtain B where 0 < B and B: path_image p ⊆ cball 0 B
    by (meson bounded_pos mem_cball_0 subsetI)
  let ?B = cball (0::complex) (B+1)
```

have *uniformly_continuous_on ?B exp*
using *holomorphic_on_exp holomorphic_on_imp_continuous_on*
by (*force intro*: *compact_uniformly_continuous*)
then obtain *d* **where** *d* > *0*
and *d*: ⋀*x x'*. ⟦*x*∈*?B*; *x'*∈*?B*; *dist x' x* < *d*⟧ ⟹ *norm* (*exp x'* − *exp x*) < *e*
using ‹*e* > *0*› **by** (*auto simp*: *uniformly_continuous_on_def dist_norm*)
then have *min 1 d* > *0*
by *force*
then obtain *g* **where** *pfg*: *polynomial_function g* **and** *g 0* = *p 0 g 1* = *p 1*
and *gless*: ⋀*t*. *t* ∈ {*0..1*} ⟹ *norm*(*g t* − *p t*) < *min 1 d*
using *path_approx_polynomial_function* [*OF* ‹*path p*›] ‹*d* > *0*› ‹*0* < *e*›
unfolding *pathfinish_def pathstart_def* **by** *blast*
have *winding_number* (*exp* ∘ *p*) *0* = *winding_number* (*exp* ∘ *g*) *0*
proof (*rule winding_number_nearby_paths_eq* [*symmetric*])
show *path* (*exp* ∘ *p*) *path* (*exp* ∘ *g*)
by (*simp_all add*: *pfg assms holomorphic_on_exp holomorphic_on_imp_continuous_on path_continuous_image path_polynomial_function*)
next
fix *t* :: *real*
assume *t*: *t* ∈ {*0..1*}
with *gless* **have** *norm*(*g t* − *p t*) < *1*
using *min_less_iff_conj* **by** *blast*
moreover have *ptB*: *norm* (*p t*) ≤ *B*
using *B t* **by** (*force simp*: *path_image_def*)
ultimately have *cmod* (*g t*) ≤ *B* + *1*
by (*meson add_mono_thms_linordered_field*(*4*) *le_less_trans less_imp_le norm_triangle_sub*)
with *ptB gless t* **have** *cmod* ((*exp* ∘ *g*) *t* − (*exp* ∘ *p*) *t*) < *e*
by (*auto simp*: *dist_norm d*)
with *e t* **show** *cmod* ((*exp* ∘ *g*) *t* − (*exp* ∘ *p*) *t*) < *cmod* ((*exp* ∘ *p*) *t* − *0*)
by *fastforce*
qed (*use* ‹*g 0* = *p 0*› ‹*g 1* = *p 1*› **in** ‹*auto simp*: *pathfinish_def pathstart_def*›)
also have ... = *1* / (*of_real* (*2* ∗ *pi*) ∗ i) ∗ *contour_integral* (*exp* ∘ *g*) (λ*w*. *1* / (*w* − *0*))
proof (*rule winding_number_valid_path*)
have *continuous_on* (*path_image g*) (*deriv exp*)
by (*metis DERIV_exp DERIV_imp_deriv continuous_on_cong holomorphic_on_exp holomorphic_on_imp_continuous_on*)
then show *valid_path* (*exp* ∘ *g*)
by (*simp add*: *field_differentiable_within_exp pfg valid_path_compose valid_path_polynomial_function*)
show *0* ∉ *path_image* (*exp* ∘ *g*)
by (*auto simp*: *path_image_def*)
qed
also have ... = *1* / (*of_real* (*2* ∗ *pi*) ∗ i) ∗ *integral* {*0..1*} (λ*x*. *vector_derivative g* (*at x*))
proof (*simp add*: *contour_integral_integral*, *rule integral_cong*)
fix *t* :: *real*
assume *t*: *t* ∈ {*0..1*}
show *vector_derivative* (*exp* ∘ *g*) (*at t*) / *exp* (*g t*) = *vector_derivative g* (*at t*)

```
    proof −
     have (exp ∘ g has_vector_derivative vector_derivative (exp ∘ g) (at t)) (at t)
           by (meson DERIV_exp differentiable_def field_vector_diff_chain_at
has_vector_derivative_def
         has_vector_derivative_polynomial_function pfg vector_derivative_works)
      moreover have (exp ∘ g has_vector_derivative vector_derivative g (at t) ∗
exp (g t)) (at t)
      by (metis DERIV_exp field_vector_diff_chain_at has_vector_derivative_polynomial_function
pfg vector_derivative_at)
      ultimately show ?thesis
        by (simp add: divide_simps, rule vector_derivative_unique_at)
    qed
  qed
  also have ... = (pathfinish p − pathstart p) / (2 ∗ of_real pi ∗ i)
  proof −
    have ((λx. vector_derivative g (at x)) has_integral g 1 − g 0) {0..1}
    by (meson differentiable_at_polynomial_function fundamental_theorem_of_calculus

                    has_vector_derivative_at_within pfg vector_derivative_works
zero_le_one)
    then show ?thesis
      unfolding pathfinish_def pathstart_def
      using ‹g 0 = p 0› ‹g 1 = p 1› by auto
  qed
  finally show ?thesis .
qed

end
```

4 Cauchy's Integral Formula

```
theory Cauchy_Integral_Formula
  imports Winding_Numbers
begin
```

4.1 Proof

```
lemma Cauchy_integral_formula_weak:
    assumes S: convex S and finite k and conf: continuous_on S f
        and fcd: (⋀x. x ∈ interior S − k ⟹ f field_differentiable at x)
        and z: z ∈ interior S − k and vpg: valid_path γ
        and pasz: path_image γ ⊆ S − {z} and loop: pathfinish γ = pathstart γ
     shows ((λw. f w / (w − z)) has_contour_integral (2∗pi ∗ i ∗ winding_number
γ z ∗ f z)) γ
proof −
  let ?fz = λw. (f w − f z)/(w − z)
  obtain f' where f': (f has_field_derivative f') (at z)
    using fcd [OF z] by (auto simp: field_differentiable_def)
   have pas: path_image γ ⊆ S and znotin: z ∉ path_image γ using pasz by
```

```
blast+
  have c: continuous (at x within S) (λw. if w = z then f' else (f w − f z) / (w − z)) if x ∈ S for x
  proof (cases x = z)
    case True then show ?thesis
    using LIM_equal [of z ?fz λw. if w = z then f' else ?fz w] has_field_derivativeD [OF f']
      by (force simp add: continuous_within Lim_at_imp_Lim_at_within)
  next
    case False
    then have dxz: dist x z > 0 by auto
    have cf: continuous (at x within S) f
      using conf continuous_on_eq_continuous_within that by blast
    have continuous (at x within S) (λw. (f w − f z) / (w − z))
      by (rule cf continuous_intros | simp add: False)+
    then show ?thesis
      apply (rule continuous_transform_within [OF _ dxz that, of ?fz])
      apply (force simp: dist_commute)
      done
  qed
  have fink': finite (insert z k) using ‹finite k› by blast
  have ∗: ((λw. if w = z then f' else ?fz w) has_contour_integral 0) γ
  proof (rule Cauchy_theorem_convex [OF _ S fink' _ vpg pas loop])
    show (λw. if w = z then f' else ?fz w) field_differentiable at w
      if w ∈ interior S − insert z k for w
    proof (rule field_differentiable_transform_within)
      show (λw. ?fz w) field_differentiable at w
        using that by (intro derivative_intros fcd; simp)
    qed (use that in ‹auto simp add: dist_pos_lt dist_commute›)
  qed (use c in ‹force simp: continuous_on_eq_continuous_within›)
  show ?thesis
    apply (rule has_contour_integral_eq)
    using znotin has_contour_integral_add [OF has_contour_integral_lmul [OF has_contour_integral_winding_number [OF vpg znotin], of f z] ∗]
    apply (auto simp: ac_simps divide_simps)
    done
qed

theorem Cauchy_integral_formula_convex_simple:
  assumes convex S and holf: f holomorphic_on S and z ∈ interior S valid_path γ path_image γ ⊆ S − {z}
      pathfinish γ = pathstart γ
    shows ((λw. f w / (w − z)) has_contour_integral (2∗pi ∗ i ∗ winding_number γ z ∗ f z)) γ
proof −
  have ⋀x. x ∈ interior S ⟹ f field_differentiable at x
    using holf at_within_interior holomorphic_onD interior_subset by fastforce
  then show ?thesis
    using assms
```

by (*intro Cauchy_integral_formula_weak* [**where** $k = \{\}$]) (*auto simp*: *holomorphic_on_imp_continuous_on*)
qed

Hence the Cauchy formula for points inside a circle.

theorem *Cauchy_integral_circlepath*:
assumes *contf*: *continuous_on* (*cball z r*) *f* **and** *holf*: *f holomorphic_on* (*ball z r*) **and** *wz*: $norm(w - z) < r$
shows $((\lambda u.\ f\ u/(u - w))$ *has_contour_integral* $(2 * of\_real\ pi * \mathrm{i} * f\ w))$
(*circlepath z r*)
proof −
have $r > 0$
using *assms le_less_trans norm_ge_zero* **by** *blast*
have $((\lambda u.\ f\ u\ /\ (u - w))$ *has_contour_integral* $(2 * pi) * \mathrm{i} *$ *winding_number* (*circlepath z r*) $w * f\ w)$
(*circlepath z r*)
proof (*rule Cauchy_integral_formula_weak* [**where** $S =$ *cball z r* **and** $k = \{\}$])
show $\bigwedge x.\ x \in$ *interior* (*cball z r*) $- \{\} \Longrightarrow$
f field_differentiable at x
using *holf holomorphic_on_imp_differentiable_at* **by** *auto*
have $w \notin$ *sphere z r*
by *simp* (*metis dist_commute dist_norm not_le order_refl wz*)
then show *path_image* (*circlepath z r*) $\subseteq$ *cball z r* $-$ $\{w\}$
using ‹$r > 0$› **by** (*auto simp add*: *cball_def sphere_def*)
qed (*use wz* **in** ‹*simp_all add*: *dist_norm norm_minus_commute contf*›)
then show *?thesis*
by (*simp add*: *winding_number_circlepath assms*)
qed

corollary *Cauchy_integral_circlepath_simple*:
assumes *f holomorphic_on cball z r* $norm(w - z) < r$
shows $((\lambda u.\ f\ u/(u - w))$ *has_contour_integral* $(2 * of\_real\ pi * \mathrm{i} * f\ w))$
(*circlepath z r*)
using *assms* **by** (*force simp*: *holomorphic_on_imp_continuous_on holomorphic_on_subset Cauchy_integral_circlepath*)

4.2 General stepping result for derivative formulas

lemma *Cauchy_next_derivative*:
assumes *continuous_on* (*path_image* γ) f'
and *leB*: $\bigwedge t.\ t \in \{0..1\} \Longrightarrow norm\ (vector\_derivative\ \gamma\ (at\ t)) \leq B$
and *int*: $\bigwedge w.\ w \in S - path\_image\ \gamma \Longrightarrow ((\lambda u.\ f'\ u\ /\ (u - w)\hat{}k)$ *has_contour_integral* $f\ w)\ \gamma$
and *k*: $k \neq 0$
and *open S*
and γ: *valid_path* γ
and *w*: $w \in S - path\_image\ \gamma$
shows $(\lambda u.\ f'\ u\ /\ (u - w)\hat{}(Suc\ k))$ *contour_integrable_on* γ
and (*f has_field_derivative* $(k *$ *contour_integral* $\gamma\ (\lambda u.\ f'\ u/(u - w)\hat{}(Suc$

```
k))))
        (at w)  (is ?thes2)
proof −
  have open (S − path_image γ) using ‹open S› closed_valid_path_image γ by
blast
  then obtain d where d>0 and d: ball w d ⊆ S − path_image γ using w
    using open_contains_ball by blast
  have [simp]: ⋀n. cmod (1 + of_nat n) = 1 + of_nat n
    by (metis norm_of_nat of_nat_Suc)
  have cint: ⋀x. ⟦x ≠ w; cmod (x − w) < d⟧
        ⟹ (λz. (f′ z / (z − x) ^ k − f′ z / (z − w) ^ k) / (x ∗ k − w ∗ k))
contour_integrable_on γ
    using int w d
   apply (intro contour_integrable_div contour_integrable_diff has_contour_integral_integrable)
    by (force simp: dist_norm norm_minus_commute)
  have 1: ∀F n in at w. (λx. f′ x ∗ (inverse (x − n) ^ k − inverse (x − w) ^ k)
/ (n − w) / of_nat k)
                        contour_integrable_on γ
    unfolding eventually_at
    apply (rule_tac x=d in exI)
    apply (simp add: ‹d > 0› dist_norm field_simps cint)
    done
  have bim_g: bounded (image f′ (path_image γ))
   by (simp add: compact_imp_bounded compact_continuous_image compact_valid_path_image
assms)
  then obtain C where C > 0 and C: ⋀x. ⟦0 ≤ x; x ≤ 1⟧ ⟹ cmod (f′ (γ x))
≤ C
    by (force simp: bounded_pos path_image_def)
  have twom: ∀F n in at w.
              ∀x∈path_image γ.
               cmod ((inverse (x − n) ^ k − inverse (x − w) ^ k) / (n − w) / k −
inverse (x − w) ^ Suc k) < e
        if 0 < e for e
  proof −
    have ∗: cmod ((inverse (x − u) ^ k − inverse (x − w) ^ k) / ((u − w) ∗ k)
− inverse (x − w) ^ Suc k)   < e
          if x: x ∈ path_image γ and u ≠ w and uwd: cmod (u − w) < d/2
             and uw_less: cmod (u − w) < e ∗ (d/2) ^(k+2) / (1 + real k)
           for u x
    proof −
      define ff where [abs_def]:
       ff n w =
         (if n = 0 then inverse(x − w)^k
          else if n = 1 then k / (x − w)^(Suc k)
          else (k ∗ of_real(Suc k)) / (x − w)^(k + 2)) for n :: nat and w
      have km1: ⋀z::complex. z ≠ 0 ⟹ z ^ (k − Suc 0) = z ^ k / z
       by (simp add: field_simps) (metis Suc_pred ‹k ≠ 0› neq0_conv power_Suc)
      have ff1: (ff i has_field_derivative ff (Suc i) z) (at z within ball w (d/2))
            if z ∈ ball w (d/2) i ≤ 1 for i z
```

```
    proof −
      have z ∉ path_image γ
        using ‹x ∈ path_image γ› d that ball_divide_subset_numeral by blast
      then have xz[simp]: x ≠ z using ‹x ∈ path_image γ› by blast
      then have neq: x * x + z * z ≠ x * (z * 2)
        by (blast intro: dest!: sum_sqs_eq)
       with xz have ⋀v. v ≠ 0 ⟹ (x * x + z * z) * v ≠ (x * (z * 2) * v) by
auto
      then have neqq: ⋀v. v ≠ 0 ⟹ x * (x * v) + z * (z * v) ≠ x * (z * (2 *
v))
        by (simp add: algebra_simps)
      show ?thesis using ‹i ≤ 1›
        apply (simp add: ff_def dist_norm Nat.le_Suc_eq km1, safe)
       apply (rule derivative_eq_intros | simp add: km1 | simp add: field_simps
neq neqq)+
        done
    qed
    { fix a::real and b::real assume ab: a > 0 b > 0
      then have k * (1 + real k) * (1 / a) ≤ k * (1 + real k) * (4 / b) ⟷ b
≤ 4 * a
        by (subst mult_le_cancel_left_pos)
          (use ‹k ≠ 0› in ‹auto simp: divide_simps›)
      with ab have real k * (1 + real k) / a ≤ (real k * 4 + real k * real k * 4)
/ b ⟷ b ≤ 4 * a
        by (simp add: field_simps)
    } note canc = this
    have ff2: cmod (ff (Suc 1) v) ≤ real (k * (k + 1)) / (d/2) ^(k + 2)
          if v ∈ ball w (d/2) for v
    proof −
      have lessd: ⋀z. cmod (γ z − v) < d/2 ⟹ cmod (w − γ z) < d
        by (metis that norm_minus_commute norm_triangle_half_r dist_norm
mem_ball)
      have d/2 ≤ cmod (x − v) using d x that
        using lessd d x
      by (auto simp add: dist_norm path_image_def ball_def not_less [symmetric]
del: divide_const_simps)
      then have d ≤ cmod (x − v) * 2
        by (simp add: field_split_simps)
      then have dpow_le: d ^(k+2) ≤ (cmod (x − v) * 2) ^(k+2)
        using ‹0 < d› order_less_imp_le power_mono by blast
      have x ≠ v using that
        using ‹x ∈ path_image γ› ball_divide_subset_numeral d by fastforce
      then show ?thesis
    using ‹d > 0› apply (simp add: ff_def norm_mult norm_divide norm_power
dist_norm canc)
      using dpow_le apply (simp add: field_split_simps)
      done
    qed
    have ub: u ∈ ball w (d/2)
```

```
      using uwd by (simp add: dist_commute dist_norm)
    have cmod (inverse (x - u) ^ k - (inverse (x - w) ^ k + of_nat k * (u -
w) / ((x - w) * (x - w) ^ k)))
                ≤ (real k * 4 + real k * real k * 4) * (cmod (u - w) * cmod (u -
w)) / (d * (d * (d/2) ^ k))
      using complex_Taylor [OF _ ff1 ff2 _ ub, of w, simplified]
      by (simp add: ff_def ‹0 < d›)
    then have cmod (inverse (x - u) ^ k - (inverse (x - w) ^ k + of_nat k *
(u - w) / ((x - w) * (x - w) ^ k)))
                ≤ (cmod (u - w) * real k) * (1 + real k) * cmod (u - w) / (d/2)
^ (k+2)
      by (simp add: field_simps)
    then have cmod (inverse (x - u) ^ k - (inverse (x - w) ^ k + of_nat k *
(u - w) / ((x - w) * (x - w) ^ k)))
              / (cmod (u - w) * real k)
              ≤ (1 + real k) * cmod (u - w) / (d/2) ^ (k+2)
        using ‹k ≠ 0› ‹u ≠ w› by (simp add: mult_ac zero_less_mult_iff
pos_divide_le_eq)
    also have ... < e
      using uw_less ‹0 < d› by (simp add: mult_ac divide_simps)
    finally have e: cmod (inverse (x-u)^k - (inverse (x-w)^k + of_nat k *
(u-w) / ((x-w) * (x-w)^k)))
                    / cmod ((u - w) * real k)  <  e
      by (simp add: norm_mult)
    have x ≠ u
    using uwd ‹0 < d› x d by (force simp: dist_norm ball_def norm_minus_commute)
    show ?thesis
      apply (rule le_less_trans [OF _ e])
      using ‹k ≠ 0› ‹x ≠ u› ‹u ≠ w›
      apply (simp add: field_simps norm_divide [symmetric])
      done
  qed
  show ?thesis
    unfolding eventually_at
    apply (rule_tac x = min (d/2) ((e*(d/2)^(k + 2))/(Suc k)) in exI)
    apply (force simp: ‹d > 0› dist_norm that simp del: power_Suc intro: *)
    done
 qed
 have 2: uniform_limit (path_image γ) (λn x. f' x * (inverse (x - n) ^ k -
inverse (x - w) ^ k) / (n - w) / of_nat k) (λx. f' x / (x - w) ^ Suc k) (at w)
  unfolding uniform_limit_iff dist_norm
 proof clarify
  fix e::real
  assume 0 < e
  have *: cmod (f' (γ x) * (inverse (γ x - u) ^ k - inverse (γ x - w) ^ k) /
((u - w) * k) -
                f' (γ x) / ((γ x - w) * (γ x - w) ^ k)) < e
        if ec: cmod ((inverse (γ x - u) ^ k - inverse (γ x - w) ^ k) / ((u -
w) * k) -
```

```
                inverse (γ x − w) ∗ inverse (γ x − w) ^ k) < e / C
            and x: 0 ≤ x x ≤ 1
          for u x
  proof (cases (f' (γ x)) = 0)
    case True then show ?thesis by (simp add: ‹0 < e›)
  next
    case False
    have cmod (f' (γ x)) ∗ (inverse (γ x − u) ^ k − inverse (γ x − w) ^ k) / ((u
− w) ∗ k) −
                  f' (γ x) / ((γ x − w) ∗ (γ x − w) ^ k)) =
        cmod (f' (γ x)) ∗ ((inverse (γ x − u) ^ k − inverse (γ x − w) ^ k) / ((u
− w) ∗ k) −
                        inverse (γ x − w) ∗ inverse (γ x − w) ^ k))
      by (simp add: field_simps)
    also have ... = cmod (f' (γ x)) ∗
                  cmod ((inverse (γ x − u) ^ k − inverse (γ x − w) ^ k) / ((u −
w) ∗ k) −
                        inverse (γ x − w) ∗ inverse (γ x − w) ^ k)
      by (simp add: norm_mult)
    also have ... < cmod (f' (γ x)) ∗ (e/C)
      using False mult_strict_left_mono [OF ec] by force
    also have ... ≤ e using C
    by (metis False ‹0 < e› frac_le less_eq_real_def mult.commute pos_le_divide_eq
x zero_less_norm_iff)
    finally show ?thesis .
  qed
  show ∀F n in at w.
         ∀x∈path_image γ.
          cmod (f' x ∗ (inverse (x − n) ^ k − inverse (x − w) ^ k) / (n − w)
/ of_nat k − f' x / (x − w) ^ Suc k) < e
      using twom [OF divide_pos_pos [OF ‹0 < e› ‹C > 0›]]   unfolding
path_image_def
    by (force intro: ∗ elim: eventually_mono)
 qed
 show (λu. f' u / (u − w) ^ (Suc k)) contour_integrable_on γ
   by (rule contour_integral_uniform_limit [OF 1 2 leB γ]) auto
 have ∗: (λn. contour_integral γ (λx. f' x ∗ (inverse (x − n) ^ k − inverse (x −
w) ^ k) / (n − w) / k))
        −w→ contour_integral γ (λu. f' u / (u − w) ^ (Suc k))
   by (rule contour_integral_uniform_limit [OF 1 2 leB γ]) auto
 have ∗∗: contour_integral γ (λx. f' x ∗ (inverse (x − u) ^ k − inverse (x − w)
^ k) / ((u − w) ∗ k)) =
         (f u − f w) / (u − w) / k
   if dist u w < d for u
 proof −
   have u: u ∈ S − path_image γ
     by (metis subsetD d dist_commute mem_ball that)
   have §: ((λx. f' x ∗ inverse (x − u) ^ k) has_contour_integral f u) γ
        ((λx. f' x ∗ inverse (x − w) ^ k) has_contour_integral f w) γ
```

```
    using u w by (simp_all add: field_simps int)
  show ?thesis
    apply (rule contour_integral_unique)
   apply (simp add: diff_divide_distrib algebra_simps § has_contour_integral_diff
has_contour_integral_div)
    done
 qed
 show ?thes2
  apply (simp add: has_field_derivative_iff del: power_Suc)
  apply (rule Lim_transform_within [OF tendsto_mult_left [OF *] ‹0 < d› ])
  apply (simp add: ‹k ≠ 0› **)
  done
qed

lemma Cauchy_next_derivative_circlepath:
 assumes contf: continuous_on (path_image (circlepath z r)) f
    and int: ⋀w. w ∈ ball z r ⟹ ((λu. f u / (u − w)^k) has_contour_integral
g w) (circlepath z r)
    and k: k ≠ 0
    and w: w ∈ ball z r
  shows (λu. f u / (u − w)^(Suc k)) contour_integrable_on (circlepath z r)
        (is ?thes1)
   and (g has_field_derivative (k ∗ contour_integral (circlepath z r) (λu. f u/(u
− w)^(Suc k)))) (at w)
        (is ?thes2)
proof −
 have r > 0 using w
  using ball_eq_empty by fastforce
 have wim: w ∈ ball z r − path_image (circlepath z r)
  using w by (auto simp: dist_norm)
 show ?thes1 ?thes2
  by (rule Cauchy_next_derivative [OF contf _ int k open_ball valid_path_circlepath
wim, where B = 2 ∗ pi ∗ |r|];
      auto simp: vector_derivative_circlepath norm_mult)+
qed
```

In particular, the first derivative formula.

```
lemma Cauchy_derivative_integral_circlepath:
 assumes contf: continuous_on (cball z r) f
    and holf: f holomorphic_on ball z r
    and w: w ∈ ball z r
  shows (λu. f u/(u − w)^2) contour_integrable_on (circlepath z r)
        (is ?thes1)
   and (f has_field_derivative (1 / (2 ∗ of_real pi ∗ i) ∗ contour_integral(circlepath
z r) (λu. f u / (u − w)^2))) (at w)
        (is ?thes2)
proof −
 have [simp]: r ≥ 0 using w
  using ball_eq_empty by fastforce
```

have *f*: *continuous_on* (*path_image* (*circlepath z r*)) *f*
by (*rule continuous_on_subset* [*OF contf*]) (*force simp*: *cball_def sphere_def*)
have *int*: $\bigwedge w.$ *dist z w* $<$ *r* $\Longrightarrow$
((λ*u*. *f u* / (*u* − *w*)) *has_contour_integral* (λ*x*. *2* ∗ *of_real pi* ∗ i ∗ *f x*) *w*) (*circlepath z r*)
by (*rule Cauchy_integral_circlepath* [*OF contf holf*]) (*simp add*: *dist_norm norm_minus_commute*)
show *?thes1*
apply (*simp add*: *power2_eq_square*)
apply (*rule Cauchy_next_derivative_circlepath* [*OF f _ _ w*, **where** *k=1*, *simplified*])
apply (*blast intro*: *int*)
done
have ((λ*x*. *2* ∗ *of_real pi* ∗ i ∗ *f x*) *has_field_derivative contour_integral* (*circlepath z r*) (λ*u*. *f u* / (*u* − *w*)^*2*)) (*at w*)
apply (*simp add*: *power2_eq_square*)
apply (*rule Cauchy_next_derivative_circlepath* [*OF f _ _ w*, **where** *k=1* **and** *g* = λ*x*. *2* ∗ *of_real pi* ∗ i ∗ *f x*, *simplified*])
apply (*blast intro*: *int*)
done
then have *fder*: (*f has_field_derivative contour_integral* (*circlepath z r*) (λ*u*. *f u* / (*u* − *w*)^*2*) / (*2* ∗ *of_real pi* ∗ i)) (*at w*)
by (*rule DERIV_cdivide* [**where** *f* = λ*x*. *2* ∗ *of_real pi* ∗ i ∗ *f x* **and** *c* = *2* ∗ *of_real pi* ∗ i, *simplified*])
show *?thes2*
by *simp* (*rule fder*)
qed

4.3 Existence of all higher derivatives

proposition *derivative_is_holomorphic*:
assumes *open S*
and *fder*: $\bigwedge z.$ *z* ∈ *S* $\Longrightarrow$ (*f has_field_derivative f′ z*) (*at z*)
shows *f′ holomorphic_on S*
proof −
have ∗: ∃*h*. (*f′ has_field_derivative h*) (*at z*) **if** *z* ∈ *S* **for** *z*
proof −
obtain *r* **where** *r* > *0* **and** *r*: *cball z r* ⊆ *S*
using *open_contains_cball* ‹*z* ∈ *S*› ‹*open S*› **by** *blast*
then have *holf_cball*: *f holomorphic_on cball z r*
unfolding *holomorphic_on_def*
using *field_differentiable_at_within field_differentiable_def fder* **by** *fastforce*
then have *continuous_on* (*path_image* (*circlepath z r*)) *f*
using ‹*r* > *0*› **by** (*force elim*: *holomorphic_on_subset* [*THEN holomorphic_on_imp_continuous_on*])
then have *contfpi*: *continuous_on* (*path_image* (*circlepath z r*)) (λ*x*. *1*/(*2* ∗ *of_real pi*∗i) ∗ *f x*)
by (*auto intro*: *continuous_intros*)+
have *contf_cball*: *continuous_on* (*cball z r*) *f* **using** *holf_cball*

```
    by (simp add: holomorphic_on_imp_continuous_on holomorphic_on_subset)
   have holf_ball: f holomorphic_on ball z r using holf_cball
    using ball_subset_cball holomorphic_on_subset by blast
   { fix w assume w: w ∈ ball z r
    have intf: (λu. f u / (u − w)²) contour_integrable_on circlepath z r
      by (blast intro: w Cauchy_derivative_integral_circlepath [OF contf_cball
holf_ball])
    have fder': (f has_field_derivative 1 / (2 * of_real pi * i) * contour_integral
(circlepath z r) (λu. f u / (u − w)²))
                (at w)
      by (blast intro: w Cauchy_derivative_integral_circlepath [OF contf_cball
holf_ball])
    have f'_eq: f' w = contour_integral (circlepath z r) (λu. f u / (u − w)²) /
(2 * of_real pi * i)
     using fder' ball_subset_cball r w by (force intro: DERIV_unique [OF fder])
    have ((λu. f u / (u − w)² / (2 * of_real pi * i)) has_contour_integral
          contour_integral (circlepath z r) (λu. f u / (u − w)²) / (2 * of_real
pi * i))
          (circlepath z r)
     by (rule has_contour_integral_div [OF has_contour_integral_integral [OF
intf]])
    then have ((λu. f u / (2 * of_real pi * i * (u − w)²)) has_contour_integral
          contour_integral (circlepath z r) (λu. f u / (u − w)²) / (2 * of_real
pi * i))
          (circlepath z r)
     by (simp add: algebra_simps)
    then have ((λu. f u / (2 * of_real pi * i * (u − w)²)) has_contour_integral
f' w) (circlepath z r)
      by (simp add: f'_eq)
   } note * = this
   show ?thesis
    using Cauchy_next_derivative_circlepath [OF contfpi, of 2 f'] ‹0 < r› *
    using centre_in_ball mem_ball by force
 qed
 show ?thesis
   by (simp add: holomorphic_on_open [OF ‹open S›] *)
qed

lemma holomorphic_deriv [holomorphic_intros]:
   ⟦f holomorphic_on S; open S⟧ ⟹ (deriv f) holomorphic_on S
by (metis DERIV_deriv_iff_field_differentiable at_within_open derivative_is_holomorphic
holomorphic_on_def)

lemma analytic_deriv [analytic_intros]: f analytic_on S ⟹ (deriv f) analytic_on
S
 using analytic_on_holomorphic holomorphic_deriv by auto

lemma holomorphic_higher_deriv [holomorphic_intros]: ⟦f holomorphic_on S; open
S⟧ ⟹ (deriv ^^ n) f holomorphic_on S
```

by (*induction n*) (*auto simp*: *holomorphic_deriv*)

lemma *analytic_higher_deriv* [*analytic_intros*]: *f analytic_on S* ⟹ (*deriv* ^^ *n*) *f analytic_on S*
unfolding *analytic_on_def* **using** *holomorphic_higher_deriv* **by** *blast*

lemma *has_field_derivative_higher_deriv*:
⟦*f holomorphic_on S*; *open S*; *x* ∈ *S*⟧
⟹ ((*deriv* ^^ *n*) *f has_field_derivative* (*deriv* ^^ (*Suc n*)) *f x*) (*at x*)
by (*metis* (*no_types*, *opaque_lifting*) *DERIV_deriv_iff_field_differentiable at_within_open comp_apply*
funpow.simps(*2*) *holomorphic_higher_deriv holomorphic_on_def*)

lemma *higher_deriv_cmult*:
assumes *f holomorphic_on A x* ∈ *A open A*
shows (*deriv* ^^ *j*) (λ*x*. *c* ∗ *f x*) *x* = *c* ∗ (*deriv* ^^ *j*) *f x*
using *assms*
proof (*induction j arbitrary*: *f x*)
case (*Suc j f x*)
have *deriv* ((*deriv* ^^ *j*) (λ*x*. *c* ∗ *f x*)) *x* = *deriv* (λ*x*. *c* ∗ (*deriv* ^^ *j*) *f x*) *x*
using *eventually_nhds_in_open*[*of A x*] *assms*(*2*,*3*) *Suc.prems*
by (*intro deriv_cong_ev refl*) (*auto elim*!: *eventually_mono simp*: *Suc.IH*)
also have ... = *c* ∗ *deriv* ((*deriv* ^^ *j*) *f*) *x* **using** *Suc.prems assms*(*2*,*3*)
by (*intro deriv_cmult holomorphic_on_imp_differentiable_at holomorphic_higher_deriv*) *auto*
finally show *?case* **by** *simp*
qed *simp_all*

lemma *valid_path_compose_holomorphic*:
assumes *valid_path g* **and** *holo*:*f holomorphic_on S* **and** *open S path_image g* ⊆ *S*
shows *valid_path* (*f* ∘ *g*)
proof (*rule valid_path_compose*[*OF* ‹*valid_path g*›])
fix *x* **assume** *x* ∈ *path_image g*
then show *f field_differentiable at x*
using *analytic_on_imp_differentiable_at analytic_on_open assms holo* **by** *blast*
next
have *deriv f holomorphic_on S*
using *holomorphic_deriv holo* ‹*open S*› **by** *auto*
then show *continuous_on* (*path_image g*) (*deriv f*)
using *assms*(*4*) *holomorphic_on_imp_continuous_on holomorphic_on_subset*
by *auto*
qed

4.4 Morera's theorem

lemma *Morera_local_triangle_ball*:
assumes ⋀*z*. *z* ∈ *S*

```
        ⟹ ∃e a. 0 < e ∧ z ∈ ball a e ∧ continuous_on (ball a e) f ∧
                (∀b c. closed_segment b c ⊆ ball a e
                      ⟶ contour_integral (linepath a b) f +
                        contour_integral (linepath b c) f +
                        contour_integral (linepath c a) f = 0)
  shows f analytic_on S
proof −
  { fix z  assume z ∈ S
    with assms obtain e a where
          0 < e and z: z ∈ ball a e and contf: continuous_on (ball a e) f
       and 0: ⋀b c. closed_segment b c ⊆ ball a e
                  ⟹ contour_integral (linepath a b) f +
                     contour_integral (linepath b c) f +
                     contour_integral (linepath c a) f = 0
      by blast
    have az: dist a z < e using mem_ball z by blast
    have ∃e>0. f holomorphic_on ball z e
    proof (intro exI conjI)
      show f holomorphic_on ball z (e − dist a z)
      proof (rule holomorphic_on_subset)
        show ball z (e − dist a z) ⊆ ball a e
          by (simp add: dist_commute ball_subset_ball_iff)
        have sub_ball: ⋀y. dist a y < e ⟹ closed_segment a y ⊆ ball a e
          by (meson ‹0 < e› centre_in_ball convex_ball convex_contains_segment
mem_ball)
        show f holomorphic_on ball a e
       using triangle_contour_integrals_starlike_primitive [OF contf _ open_ball,
of a]
             derivative_is_holomorphic[OF open_ball]
          by (force simp add: 0 ‹0 < e› sub_ball)
      qed
    qed (simp add: az)
  }
  then show ?thesis
    by (simp add: analytic_on_def)
qed

lemma Morera_local_triangle:
  assumes ⋀z. z ∈ S
        ⟹ ∃t. open t ∧ z ∈ t ∧ continuous_on t f ∧
              (∀a b c. convex hull {a,b,c} ⊆ t
                      ⟶ contour_integral (linepath a b) f +
                        contour_integral (linepath b c) f +
                        contour_integral (linepath c a) f = 0)
  shows f analytic_on S
proof −
  { fix z  assume z ∈ S
    with assms obtain t where
          open t and z: z ∈ t and contf: continuous_on t f
```

```
      and 0: ⋀a b c. convex hull {a,b,c} ⊆ t
                  ⟹ contour_integral (linepath a b) f +
                     contour_integral (linepath b c) f +
                     contour_integral (linepath c a) f = 0
    by force
  then obtain e where e>0 and e: ball z e ⊆ t
    using open_contains_ball by blast
  have [simp]: continuous_on (ball z e) f using contf
    using continuous_on_subset e by blast
  have eq0: ⋀b c. closed_segment b c ⊆ ball z e ⟹
                  contour_integral (linepath z b) f +
                  contour_integral (linepath b c) f +
                  contour_integral (linepath c z) f = 0
    by (meson 0 z ‹0 < e› centre_in_ball closed_segment_subset convex_ball
dual_order.trans e starlike_convex_subset)
  have ∃e a. 0 < e ∧ z ∈ ball a e ∧ continuous_on (ball a e) f ∧
          (∀b c. closed_segment b c ⊆ ball a e ⟶
                 contour_integral (linepath a b) f + contour_integral (linepath b
c) f + contour_integral (linepath c a) f = 0)
    using ‹e > 0› eq0 by force
 }
 then show ?thesis
  by (simp add: Morera_local_triangle_ball)
qed

proposition Morera_triangle:
  ⟦continuous_on S f; open S;
    ⋀a b c. convex hull {a,b,c} ⊆ S
          ⟶ contour_integral (linepath a b) f +
            contour_integral (linepath b c) f +
            contour_integral (linepath c a) f = 0⟧
  ⟹ f analytic_on S
 using Morera_local_triangle by blast
```

4.5 Combining theorems for higher derivatives including Leibniz rule

```
lemma higher_deriv_linear [simp]:
  (deriv ^^ n) (λw. c*w) = (λz. if n = 0 then c*z else if n = 1 then c else 0)
 by (induction n) auto

lemma higher_deriv_const [simp]: (deriv ^^ n) (λw. c) = (λw. if n=0 then c else
0)
 by (induction n) auto

lemma higher_deriv_ident [simp]:
  (deriv ^^ n) (λw. w) z = (if n = 0 then z else if n = 1 then 1 else 0)
proof (induction n)
 case (Suc n)
```

```
  then show ?case by (metis higher_deriv_linear lambda_one)
qed auto

lemma higher_deriv_id [simp]:
    (deriv ^^ n) id z = (if n = 0 then z else if n = 1 then 1 else 0)
  by (simp add: id_def)

lemma has_complex_derivative_funpow_1:
    ⟦(f has_field_derivative 1) (at z); f z = z⟧ ⟹ (f^^n has_field_derivative 1)
(at z)
proof (induction n)
  case 0
  then show ?case
    by (simp add: id_def)
next
  case (Suc n)
  then show ?case
    by (metis DERIV_chain funpow_Suc_right mult.right_neutral)
qed

lemma higher_deriv_uminus:
  assumes f holomorphic_on S open S and z: z ∈ S
    shows (deriv ^^ n) (λw. -(f w)) z = - ((deriv ^^ n) f z)
using z
proof (induction n arbitrary: z)
  case 0 then show ?case by simp
next
  case (Suc n z)
  have ∗: ((deriv ^^ n) f has_field_derivative deriv ((deriv ^^ n) f) z) (at z)
    using Suc.prems assms has_field_derivative_higher_deriv by auto
  have ⋀x. x ∈ S ⟹ - (deriv ^^ n) f x = (deriv ^^ n) (λw. - f w) x
    by (auto simp add: Suc)
  then have ((deriv ^^ n) (λw. - f w) has_field_derivative - deriv ((deriv ^^ n)
f) z) (at z)
    using has_field_derivative_transform_within_open [of λw. -((deriv ^^ n) f
w)]
    using ∗ DERIV_minus Suc.prems ‹open S› by blast
  then show ?case
    by (simp add: DERIV_imp_deriv)
qed

lemma higher_deriv_add:
  fixes z::complex
  assumes f holomorphic_on S g holomorphic_on S open S and z: z ∈ S
    shows (deriv ^^ n) (λw. f w + g w) z = (deriv ^^ n) f z + (deriv ^^ n) g z
using z
proof (induction n arbitrary: z)
  case 0 then show ?case by simp
next
```

```
  case (Suc n z)
  have *: ((deriv ^^ n) f has_field_derivative deriv ((deriv ^^ n) f) z) (at z)
        ((deriv ^^ n) g has_field_derivative deriv ((deriv ^^ n) g) z) (at z)
    using Suc.prems assms has_field_derivative_higher_deriv by auto
  have ⋀x. x ∈ S ⟹ (deriv ^^ n) f x + (deriv ^^ n) g x = (deriv ^^ n) (λw. f
w + g w) x
    by (auto simp add: Suc)
  then have ((deriv ^^ n) (λw. f w + g w) has_field_derivative
       deriv ((deriv ^^ n) f) z + deriv ((deriv ^^ n) g) z) (at z)
    using has_field_derivative_transform_within_open [of λw. (deriv ^^ n) f w
+ (deriv ^^ n) g w]
    using * Deriv.field_differentiable_add Suc.prems ‹open S› by blast
  then show ?case
    by (simp add: DERIV_imp_deriv)
qed

lemma higher_deriv_diff:
  fixes z::complex
  assumes f holomorphic_on S g holomorphic_on S open S z ∈ S
    shows (deriv ^^ n) (λw. f w − g w) z = (deriv ^^ n) f z − (deriv ^^ n) g z
  unfolding diff_conv_add_uminus higher_deriv_add
   using assms higher_deriv_add higher_deriv_uminus holomorphic_on_minus
by presburger

lemma bb: Suc n choose k = (n choose k) + (if k = 0 then 0 else (n choose (k −
1)))
  by (cases k) simp_all

lemma higher_deriv_mult:
  fixes z::complex
  assumes f holomorphic_on S g holomorphic_on S open S and z: z ∈ S
    shows (deriv ^^ n) (λw. f w * g w) z =
          (∑ i = 0..n. of_nat (n choose i) * (deriv ^^ i) f z * (deriv ^^ (n − i)) g
z)
using z
proof (induction n arbitrary: z)
  case 0 then show ?case by simp
next
  case (Suc n z)
  have *: ⋀n. ((deriv ^^ n) f has_field_derivative deriv ((deriv ^^ n) f) z) (at z)
        ⋀n. ((deriv ^^ n) g has_field_derivative deriv ((deriv ^^ n) g) z) (at z)
    using Suc.prems assms has_field_derivative_higher_deriv by auto
  have sumeq: (∑ i = 0..n.
              of_nat (n choose i) * (deriv ((deriv ^^ i) f) z * (deriv ^^ (n − i)) g
z + deriv ((deriv ^^ (n − i)) g) z * (deriv ^^ i) f z)) =
          g z * deriv ((deriv ^^ n) f) z + (∑ i = 0..n. (deriv ^^ i) f z * (of_nat
(Suc n choose i) * (deriv ^^ (Suc n − i)) g z))
    apply (simp add: bb algebra_simps sum.distrib)
    apply (subst (4) sum_Suc_reindex)
```

```
    apply (auto simp: algebra_simps Suc_diff_le intro: sum.cong)
    done
  have ((deriv ^^ n) (λw. f w * g w) has_field_derivative
        (∑ i = 0..Suc n. (Suc n choose i) * (deriv ^^ i) f z * (deriv ^^ (Suc n −
i)) g z))
        (at z)
    apply (rule has_field_derivative_transform_within_open
        [of λw. (∑ i = 0..n. of_nat (n choose i) * (deriv ^^ i) f w * (deriv ^^ (n
− i)) g w) _ _ S])
      apply (simp add: algebra_simps)
      apply (rule derivative_eq_intros | simp)+
       apply (auto intro: DERIV_mult * ‹open S› Suc.prems Suc.IH [symmetric])
    by (metis (no_types, lifting) mult.commute sum.cong sumeq)
  then show ?case
    unfolding funpow.simps o_apply
    by (simp add: DERIV_imp_deriv)
qed

lemma higher_deriv_transform_within_open:
  fixes z::complex
  assumes f holomorphic_on S g holomorphic_on S open S and z: z ∈ S
      and fg: ⋀w. w ∈ S ⟹ f w = g w
    shows (deriv ^^ i) f z = (deriv ^^ i) g z
using z
by (induction i arbitrary: z)
  (auto simp: fg intro: complex_derivative_transform_within_open holomorphic_higher_deriv
assms)

lemma higher_deriv_compose_linear:
  fixes z::complex
  assumes f: f holomorphic_on T and S: open S and T: open T and z: z ∈ S
      and fg: ⋀w. w ∈ S ⟹ u * w ∈ T
    shows (deriv ^^ n) (λw. f (u * w)) z = u^n * (deriv ^^ n) f (u * z)
using z
proof (induction n arbitrary: z)
  case 0 then show ?case by simp
next
  case (Suc n z)
  have holo0: f holomorphic_on (*) u ‘ S
    by (meson fg f holomorphic_on_subset image_subset_iff)
  have holo2: (deriv ^^ n) f holomorphic_on (*) u ‘ S
   by (meson f fg holomorphic_higher_deriv holomorphic_on_subset image_subset_iff
T)
  have holo3: (λz. u ^ n * (deriv ^^ n) f (u * z)) holomorphic_on S
    by (intro holo2 holomorphic_on_compose [where g=(deriv ^^ n) f, unfolded
o_def] holomorphic_intros)
  have (*) u holomorphic_on S f holomorphic_on (*) u ‘ S
    by (rule holo0 holomorphic_intros)+
  then have holo1: (λw. f (u * w)) holomorphic_on S
```

```
    by (rule holomorphic_on_compose [where g=f, unfolded o_def])
  have deriv ((deriv ^^ n) (λw. f (u * w))) z = deriv (λz. u^n * (deriv ^^ n) f
(u*z)) z
  proof (rule complex_derivative_transform_within_open [OF _ holo3 S Suc.prems])
    show (deriv ^^ n) (λw. f (u * w)) holomorphic_on S
      by (rule holomorphic_higher_deriv [OF holo1 S])
  qed (simp add: Suc.IH)
  also have ... = u^n * deriv (λz. (deriv ^^ n) f (u * z)) z
  proof -
    have (deriv ^^ n) f analytic_on T
      by (simp add: analytic_on_open f holomorphic_higher_deriv T)
    then have (λw. (deriv ^^ n) f (u * w)) analytic_on S
    proof -
      have (deriv ^^ n) f ∘ (*) u holomorphic_on S
        by (simp add: holo2 holomorphic_on_compose)
      then show ?thesis
        by (simp add: S analytic_on_open o_def)
    qed
    then show ?thesis
     by (intro deriv_cmult analytic_on_imp_differentiable_at [OF _ Suc.prems])
  qed
  also have ... = u * u ^ n * deriv ((deriv ^^ n) f) (u * z)
  proof -
    have (deriv ^^ n) f field_differentiable at (u * z)
    using Suc.prems T f fg holomorphic_higher_deriv holomorphic_on_imp_differentiable_at
by blast
    then show ?thesis
      by (simp add: deriv_compose_linear)
  qed
  finally show ?case
    by simp
qed

lemma higher_deriv_add_at:
  assumes f analytic_on {z} g analytic_on {z}
    shows (deriv ^^ n) (λw. f w + g w) z = (deriv ^^ n) f z + (deriv ^^ n) g z
proof -
  have f analytic_on {z} ∧ g analytic_on {z}
    using assms by blast
  with higher_deriv_add show ?thesis
    by (auto simp: analytic_at_two)
qed

lemma higher_deriv_diff_at:
  assumes f analytic_on {z} g analytic_on {z}
    shows (deriv ^^ n) (λw. f w - g w) z = (deriv ^^ n) f z - (deriv ^^ n) g z
proof -
  have f analytic_on {z} ∧ g analytic_on {z}
    using assms by blast
```

with *higher_deriv_diff* **show** *?thesis*
by (*auto simp*: *analytic_at_two*)
qed

lemma *higher_deriv_uminus_at*:
f analytic_on {*z*} ⟹ (*deriv* ^^ *n*) (λ*w*. −(*f w*)) *z* = − ((*deriv* ^^ *n*) *f z*)
using *higher_deriv_uminus*
by (*auto simp*: *analytic_at*)

lemma *higher_deriv_mult_at*:
assumes *f analytic_on* {*z*} *g analytic_on* {*z*}
shows (*deriv* ^^ *n*) (λ*w*. *f w* ∗ *g w*) *z* =
(∑ *i* = *0*..*n*. *of_nat* (*n choose i*) ∗ (*deriv* ^^ *i*) *f z* ∗ (*deriv* ^^ (*n* − *i*)) *g z*)
proof −
have *f analytic_on* {*z*} ∧ *g analytic_on* {*z*}
using *assms* **by** *blast*
with *higher_deriv_mult* **show** *?thesis*
by (*auto simp*: *analytic_at_two*)
qed

Nonexistence of isolated singularities and a stronger integral formula.

proposition *no_isolated_singularity*:
fixes *z*::*complex*
assumes *f*: *continuous_on S f* **and** *holf*: *f holomorphic_on* (*S* − *K*) **and** *S*: *open S* **and** *K*: *finite K*
shows *f holomorphic_on S*
proof −
{ fix *z*
assume *z* ∈ *S* **and** *cdf*: ⋀*x*. *x* ∈ *S* − *K* ⟹ *f field_differentiable at x*
have *f field_differentiable at z*
proof (*cases z* ∈ *K*)
case *False* **then show** *?thesis* **by** (*blast intro*: *cdf* ‹*z* ∈ *S*›)
next
case *True*
with *finite_set_avoid* [*OF K*, *of z*]
obtain *d* **where** *d*>*0* **and** *d*: ⋀*x*. ⟦*x*∈*K*; *x* ≠ *z*⟧ ⟹ *d* ≤ *dist z x*
by *blast*
obtain *e* **where** *e*>*0* **and** *e*: *ball z e* ⊆ *S*
using *S* ‹*z* ∈ *S*› **by** (*force simp*: *open_contains_ball*)
have *fde*: *continuous_on* (*ball z* (*min d e*)) *f*
by (*metis Int_iff ball_min_Int continuous_on_subset e f subsetI*)
have *cont*: {*a*,*b*,*c*} ⊆ *ball z* (*min d e*) ⟹ *continuous_on* (*convex hull* {*a*, *b*, *c*}) *f* **for** *a b c*
by (*simp add*: *hull_minimal continuous_on_subset* [*OF fde*])
have *fd*: ⟦{*a*,*b*,*c*} ⊆ *ball z* (*min d e*); *x* ∈ *interior* (*convex hull* {*a*, *b*, *c*}) − *K*⟧
⟹ *f field_differentiable at x* **for** *a b c x*
by (*metis cdf Diff_iff Int_iff ball_min_Int subsetD convex_ball e inte-*

```
rior_mono interior_subset subset_hull)
      obtain g where ⋀w. w ∈ ball z (min d e) ⟹ (g has_field_derivative f w)
(at w within ball z (min d e))
        apply (rule contour_integral_convex_primitive
                    [OF convex_ball fde Cauchy_theorem_triangle_cofinite [OF _
K]])
        using cont fd by auto
      then have f holomorphic_on ball z (min d e)
        by (metis open_ball at_within_open derivative_is_holomorphic)
      then show ?thesis
        unfolding holomorphic_on_def
          by (metis open_ball ‹0 < d› ‹0 < e› at_within_open centre_in_ball
min_less_iff_conj)
    qed
  }
  with holf S K show ?thesis
   by (simp add: holomorphic_on_open open_Diff finite_imp_closed field_differentiable_def
[symmetric])
qed

lemma no_isolated_singularity':
  fixes z::complex
  assumes f: ⋀z. z ∈ K ⟹ (f ⟶ f z) (at z within S)
     and holf: f holomorphic_on (S − K) and S: open S and K: finite K
    shows f holomorphic_on S
proof (rule no_isolated_singularity[OF _ assms(2−)])
  show continuous_on S f unfolding continuous_on_def
  proof
    fix z assume z: z ∈ S
    show (f ⟶ f z) (at z within S)
    proof (cases z ∈ K)
      case False
      from holf have continuous_on (S − K) f
        by (rule holomorphic_on_imp_continuous_on)
      with z False have (f ⟶ f z) (at z within (S − K))
        by (simp add: continuous_on_def)
      also from z K S False have at z within (S − K) = at z within S
        by (subst (1 2) at_within_open) (auto intro: finite_imp_closed)
      finally show (f ⟶ f z) (at z within S) .
    qed (insert assms z, simp_all)
  qed
qed

proposition Cauchy_integral_formula_convex:
  assumes S: convex S and K: finite K and contf: continuous_on S f
    and fcd: (⋀x. x ∈ interior S − K ⟹ f field_differentiable at x)
    and z: z ∈ interior S and vpg: valid_path γ
    and pasz: path_image γ ⊆ S − {z} and loop: pathfinish γ = pathstart γ
  shows ((λw. f w / (w − z)) has_contour_integral (2*pi * i * winding_number
```

$\gamma\ z * f\ z))\ \gamma$
proof −
have $*$: $\bigwedge x.\ x \in$ *interior* $S \Longrightarrow f$ *field_differentiable at* x
unfolding *holomorphic_on_open* [*symmetric*] *field_differentiable_def*
using *no_isolated_singularity* [**where** $S =$ *interior* S]
by (*meson K contf continuous_at_imp_continuous_on continuous_on_interior fcd*
field_differentiable_at_within field_differentiable_def holomorphic_onI
holomorphic_on_imp_differentiable_at open_interior)
show *?thesis*
by (*rule Cauchy_integral_formula_weak* [*OF S finite.emptyI contf*]) (*use* $*$ *assms* **in** *auto*)
qed

Formula for higher derivatives.

lemma *Cauchy_has_contour_integral_higher_derivative_circlepath*:
assumes *contf*: *continuous_on* (*cball z r*) *f*
and *holf*: *f holomorphic_on ball z r*
and *w*: $w \in$ *ball z r*
shows $((\lambda u.\ f\ u\ /\ (u - w)\ \hat{}\ (Suc\ k))$ *has_contour_integral* $((2 * pi *$ i$)\ /$ $(fact\ k) * (deriv\ \hat{}\hat{}\ k)\ f\ w))$
(*circlepath z r*)
using *w*
proof (*induction k arbitrary*: *w*)
case *0* **then show** *?case*
using *assms* **by** (*auto simp*: *Cauchy_integral_circlepath dist_commute dist_norm*)
next
case (*Suc k*)
have [*simp*]: $r > 0$ **using** *w*
using *ball_eq_empty* **by** *fastforce*
have *f*: *continuous_on* (*path_image* (*circlepath z r*)) *f*
by (*rule continuous_on_subset* [*OF contf*]) (*force simp*: *cball_def sphere_def less_imp_le*)
obtain *X* **where** *X*: $((\lambda u.\ f\ u\ /\ (u - w)\ \hat{}\ Suc\ (Suc\ k))$ *has_contour_integral* *X*) (*circlepath z r*)
using *Cauchy_next_derivative_circlepath*(*1*) [*OF f Suc.IH _ Suc.prems*]
by (*auto simp*: *contour_integrable_on_def*)
then have *con*: *contour_integral* (*circlepath z r*) $((\lambda u.\ f\ u\ /\ (u - w)\ \hat{}\ Suc\ (Suc\ k))) = X$
by (*rule contour_integral_unique*)
have $\bigwedge n.\ ((deriv\ \hat{}\hat{}\ n)\ f$ *has_field_derivative deriv* $((deriv\ \hat{}\hat{}\ n)\ f)\ w)$ (*at w*)
using *Suc.prems assms has_field_derivative_higher_deriv* **by** *auto*
then have *dnf_diff*: $\bigwedge n.$ (*deriv* $\hat{}\hat{}$ *n*) *f field_differentiable* (*at w*)
by (*force simp*: *field_differentiable_def*)
have *deriv* $(\lambda w.$ *complex_of_real* $(2 * pi) *$ i $/$ $(fact\ k) * (deriv\ \hat{}\hat{}\ k)\ f\ w)\ w =$
of_nat $(Suc\ k) *$ *contour_integral* (*circlepath z r*) $(\lambda u.\ f\ u\ /\ (u - w)\ \hat{}$ $Suc\ (Suc\ k))$
by (*force intro*!: *DERIV_imp_deriv Cauchy_next_derivative_circlepath* [*OF f Suc.IH _ Suc.prems*])

```
  also have ... = of_nat (Suc k) * X
    by (simp only: con)
  finally have deriv (λw. ((2 * pi) * i / (fact k)) * (deriv ^^ k) f w) w = of_nat
(Suc k) * X .
  then have ((2 * pi) * i / (fact k)) * deriv (λw. (deriv ^^ k) f w) w = of_nat
(Suc k) * X
    by (metis deriv_cmult dnf_diff)
  then have deriv (λw. (deriv ^^ k) f w) w = of_nat (Suc k) * X / ((2 * pi) * i
/ (fact k))
    by (simp add: field_simps)
  then show ?case
  using of_nat_eq_0_iff X by fastforce
qed

lemma Cauchy_higher_derivative_integral_circlepath:
  assumes contf: continuous_on (cball z r) f
      and holf: f holomorphic_on ball z r
      and w: w ∈ ball z r
    shows (λu. f u / (u - w)^(Suc k)) contour_integrable_on (circlepath z r)
          (is ?thes1)
      and (deriv ^^ k) f w = (fact k) / (2 * pi * i) * contour_integral(circlepath z
r) (λu. f u/(u - w)^(Suc k))
          (is ?thes2)
proof -
  have *: ((λu. f u / (u - w) ^ Suc k) has_contour_integral (2 * pi) * i / (fact
k) * (deriv ^^ k) f w)
          (circlepath z r)
    using Cauchy_has_contour_integral_higher_derivative_circlepath [OF assms]
    by simp
  show ?thes1 using *
    using contour_integrable_on_def by blast
  show ?thes2
    unfolding contour_integral_unique [OF *] by (simp add: field_split_simps)
qed

corollary Cauchy_contour_integral_circlepath:
  assumes continuous_on (cball z r) f f holomorphic_on ball z r w ∈ ball z r
    shows contour_integral(circlepath z r) (λu. f u/(u - w)^(Suc k)) = (2 * pi *
i) * (deriv ^^ k) f w / (fact k)
by (simp add: Cauchy_higher_derivative_integral_circlepath [OF assms])

lemma Cauchy_contour_integral_circlepath_2:
  assumes continuous_on (cball z r) f f holomorphic_on ball z r w ∈ ball z r
    shows contour_integral(circlepath z r) (λu. f u/(u - w)^2) = (2 * pi * i) *
deriv f w
  using Cauchy_contour_integral_circlepath [OF assms, of 1]
  by (simp add: power2_eq_square)
```

4.6 A holomorphic function is analytic, i.e. has local power series

theorem *holomorphic_power_series*:
assumes *holf*: *f holomorphic_on ball z r*
and *w*: *w* ∈ *ball z r*
shows ((λ*n*. (*deriv* ^^ *n*) *f z* / (*fact n*) ∗ (*w* − *z*)^*n*) *sums f w*)
proof −
— Replacing *r* and the original (weak) premises with stronger ones
obtain *r* **where** *r* > *0* **and** *holfc*: *f holomorphic_on cball z r* **and** *w*: *w* ∈ *ball z r*
proof
have *cball z* ((*r* + *dist w z*) / *2*) ⊆ *ball z r*
using *w* **by** (*simp add*: *dist_commute field_sum_of_halves subset_eq*)
then show *f holomorphic_on cball z* ((*r* + *dist w z*) / *2*)
by (*rule holomorphic_on_subset* [*OF holf*])
have *r* > *0*
using *w* **by** *clarsimp* (*metis dist_norm le_less_trans norm_ge_zero*)
then show *0* < (*r* + *dist w z*) / *2*
by *simp* (*use zero_le_dist* [*of w z*] **in** *linarith*)
qed (*use w* **in** ‹*auto simp*: *dist_commute*›)
then have *holf*: *f holomorphic_on ball z r*
using *ball_subset_cball holomorphic_on_subset* **by** *blast*
have *contf*: *continuous_on* (*cball z r*) *f*
by (*simp add*: *holfc holomorphic_on_imp_continuous_on*)
have *cint*: ⋀*k*. (λ*u*. *f u* / (*u* − *z*) ^ *Suc k*) *contour_integrable_on circlepath z r*
by (*rule Cauchy_higher_derivative_integral_circlepath* [*OF contf holf*]) (*simp add*: ‹*0* < *r*›)
obtain *B* **where** *0* < *B* **and** *B*: ⋀*u*. *u* ∈ *cball z r* ⟹ *norm*(*f u*) ≤ *B*
by (*metis* (*no_types*) *bounded_pos compact_cball compact_continuous_image compact_imp_bounded contf image_eqI*)
obtain *k* **where** *k*: *0* < *k k* ≤ *r* **and** *wz_eq*: *norm*(*w* − *z*) = *r* − *k*
and *kle*: ⋀*u*. *norm*(*u* − *z*) = *r* ⟹ *k* ≤ *norm*(*u* − *w*)
proof
show ⋀*u*. *cmod* (*u* − *z*) = *r* ⟹ *r* − *dist z w* ≤ *cmod* (*u* − *w*)
by (*metis add_diff_eq diff_add_cancel dist_norm norm_diff_ineq*)
qed (*use w* **in** ‹*auto simp*: *dist_norm norm_minus_commute*›)
have *ul*: *uniform_limit* (*sphere z r*) (λ*n x*. (∑ *k*<*n*. (*w* − *z*) ^ *k* ∗ (*f x* / (*x* − *z*) ^ *Suc k*))) (λ*x*. *f x* / (*x* − *w*)) *sequentially*
unfolding *uniform_limit_iff dist_norm*
proof *clarify*
fix *e*::*real*
assume *0* < *e*
have *rr*: *0* ≤ (*r* − *k*) / *r* (*r* − *k*) / *r* < *1* **using** *k* **by** *auto*
obtain *n* **where** *n*: ((*r* − *k*) / *r*) ^ *n* < *e* / *B* ∗ *k*
using *real_arch_pow_inv* [*of e*/*B*∗*k* (*r* − *k*)/*r*] ‹*0* < *e*› ‹*0* < *B*› *k* **by** *force*
have *norm* ((∑ *k*<*N*. (*w* − *z*) ^ *k* ∗ *f u* / (*u* − *z*) ^ *Suc k*) − *f u* / (*u* − *w*)) < *e*
if *n* ≤ *N* **and** *r*: *r* = *dist z u* **for** *N u*
proof −

```
    have N: ((r − k) / r) ^ N < e / B * k
      using le_less_trans [OF power_decreasing n]
      using ‹n ≤ N› k by auto
    have u [simp]: (u ≠ z) ∧ (u ≠ w)
      using ‹0 < r› r w by auto
    have wzu_not1: (w − z) / (u − z) ≠ 1
       by (metis (no_types) dist_norm divide_eq_1_iff less_irrefl mem_ball
norm_minus_commute r w)
    have norm ((∑k<N. (w − z) ^ k * f u / (u − z) ^ Suc k) * (u − w) − f u)
         = norm ((∑k<N. (((w − z) / (u − z)) ^ k)) * f u * (u − w) / (u −
z) − f u)
       unfolding sum_distrib_right sum_divide_distrib power_divide by (simp
add: algebra_simps)
    also have ... = norm ((((((w − z) / (u − z)) ^ N − 1) * (u − w) / (((w −
z) / (u − z) − 1) * (u − z)) − 1) * norm (f u)
      using ‹0 < B›
      apply (auto simp: geometric_sum [OF wzu_not1])
      apply (simp add: field_simps norm_mult [symmetric])
      done
    also have ... = norm ((u−z) ^ N * (w − u) − ((w − z) ^ N − (u−z) ^
N) * (u−w)) / (r ^ N * norm (u−w)) * norm (f u)
       using ‹0 < r› r by (simp add: divide_simps norm_mult norm_divide
norm_power dist_norm norm_minus_commute)
    also have ... = norm ((w − z) ^ N * (w − u)) / (r ^ N * norm (u − w))
* norm (f u)
      by (simp add: algebra_simps)
    also have ... = norm (w − z) ^ N * norm (f u) / r ^ N
      by (simp add: norm_mult norm_power norm_minus_commute)
    also have ... ≤ (((r − k)/r)^N) * B
      using ‹0 < r› w k
      by (simp add: B divide_simps mult_mono r wz_eq)
    also have ... < e * k
      using ‹0 < B› N by (simp add: divide_simps)
    also have ... ≤ e * norm (u − w)
      using r kle ‹0 < e› by (simp add: dist_commute dist_norm)
    finally show ?thesis
      by (simp add: field_split_simps norm_divide del: power_Suc)
  qed
  with ‹0 < r› show ∀F n in sequentially. ∀x∈sphere z r.
           norm ((∑k<n. (w − z) ^ k * (f x / (x − z) ^ Suc k)) − f x / (x −
w)) < e
    by (auto simp: mult_ac less_imp_le eventually_sequentially Ball_def)
 qed
 have §: ⋀x k. k∈ {..<x} ⟹
          (λu. (w − z) ^ k * (f u / (u − z) ^ Suc k)) contour_integrable_on
circlepath z r
   using contour_integrable_lmul [OF cint, of (w − z) ^ a for a] by (simp add:
field_simps)
 have eq: ∀F x in sequentially.
```

```
        contour_integral (circlepath z r) (λu. ∑k<x. (w − z) ^ k * (f u / (u
− z) ^ Suc k)) =
        (∑k<x. contour_integral (circlepath z r) (λu. f u / (u − z) ^ Suc k) *
(w − z) ^ k)
   apply (rule eventuallyI)
   apply (subst contour_integral_sum, simp)
   apply (simp_all only: § contour_integral_lmul cint algebra_simps)
   done
  have ⋀u k. k ∈ {..<u} ⟹ (λx. f x / (x − z) ^ Suc k) contour_integrable_on
circlepath z r
   using ‹0 < r› by (force intro!: Cauchy_higher_derivative_integral_circlepath
[OF contf holf])
  then have ⋀u. (λy. ∑k<u. (w − z) ^ k * (f y / (y − z) ^ Suc k)) con-
tour_integrable_on circlepath z r
   by (intro contour_integrable_sum contour_integrable_lmul, simp)
  then have (λk. contour_integral (circlepath z r) (λu. f u/(u − z)^(Suc k)) * (w
− z)^k)
      sums contour_integral (circlepath z r) (λu. f u/(u − w))
   unfolding sums_def using ‹0 < r›
  by (intro Lim_transform_eventually [OF _ eq] contour_integral_uniform_limit_circlepath
[OF eventuallyI ul]) auto
  then have (λk. contour_integral (circlepath z r) (λu. f u/(u − z)^(Suc k)) * (w
− z)^k)
         sums (2 * of_real pi * i * f w)
   using w by (auto simp: dist_commute dist_norm contour_integral_unique [OF
Cauchy_integral_circlepath_simple [OF holfc]])
  then have (λk. contour_integral (circlepath z r) (λu. f u / (u − z) ^ Suc k) *
(w − z)^k / (i * (of_real pi * 2)))
         sums ((2 * of_real pi * i * f w) / (i * (complex_of_real pi * 2)))
   by (rule sums_divide)
  then have (λn. (w − z) ^ n * contour_integral (circlepath z r) (λu. f u / (u −
z) ^ Suc n) / (i * (of_real pi * 2)))
         sums f w
   by (simp add: field_simps)
  then show ?thesis
   by (simp add: field_simps ‹0 < r› Cauchy_higher_derivative_integral_circlepath
[OF contf holf])
qed
```

4.7 The Liouville theorem and the Fundamental Theorem of Algebra

These weak Liouville versions don't even need the derivative formula.

```
lemma Liouville_weak_0:
  assumes holf: f holomorphic_on UNIV and inf: (f ⟶ 0) at_infinity
    shows f z = 0
proof (rule ccontr)
  assume fz: f z ≠ 0
  with inf [unfolded Lim_at_infinity, rule_format, of norm(f z)/2]
```

```
  obtain B where B: ⋀x. B ≤ cmod x ⟹ norm (f x) * 2 < cmod (f z)
    by (auto simp: dist_norm)
  define R where R = 1 + |B| + norm z
  have R > 0 unfolding R_def
  proof −
    have 0 ≤ cmod z + |B|
      by (metis (full_types) add_nonneg_nonneg norm_ge_zero real_norm_def)
    then show 0 < 1 + |B| + cmod z
      by linarith
  qed
  have ∗: ((λu. f u / (u − z)) has_contour_integral 2 ∗ complex_of_real pi ∗ i ∗
f z) (circlepath z R)
    using continuous_on_subset holf  holomorphic_on_subset ‹0 < R›
   by (force intro: holomorphic_on_imp_continuous_on Cauchy_integral_circlepath)
  have cmod (x − z) = R ⟹ cmod (f x) * 2 < cmod (f z) for x
    unfolding R_def
    by (rule B) (use norm_triangle_ineq4 [of x z] in auto)
  with ‹R > 0› fz show False
    using has_contour_integral_bound_circlepath [OF ∗, of norm(f z)/2/R]
    by (auto simp: less_imp_le norm_mult norm_divide field_split_simps)
qed

proposition Liouville_weak:
  assumes f holomorphic_on UNIV and (f ⟶ l) at_infinity
    shows f z = l
  using Liouville_weak_0 [of λz. f z − l]
  by (simp add: assms holomorphic_on_diff LIM_zero)

proposition Liouville_weak_inverse:
  assumes f holomorphic_on UNIV and unbounded: ⋀B. eventually (λx. norm
(f x) ≥ B) at_infinity
    obtains z where f z = 0
proof −
  { assume f: ⋀z. f z ≠ 0
    have 1: (λx. 1 / f x) holomorphic_on UNIV
      by (simp add: holomorphic_on_divide assms f)
    have 2: ((λx. 1 / f x) ⟶ 0) at_infinity
    proof (rule tendstoI [OF eventually_mono])
      fix e::real
      assume e > 0
      show eventually (λx. 2/e ≤ cmod (f x)) at_infinity
        by (rule_tac B=2/e in unbounded)
    qed (simp add: dist_norm norm_divide field_split_simps)
    have False
      using Liouville_weak_0 [OF 1 2] f by simp
  }
  then show ?thesis
    using that by blast
qed
```

In particular we get the Fundamental Theorem of Algebra.

theorem *fundamental_theorem_of_algebra*:
  fixes *a* :: *nat* ⇒ *complex*
 assumes *a 0* = *0* ∨ (∃ *i* ∈ {*1..n*}. *a i* ≠ *0*)
 obtains *z* **where** (∑ *i*≤*n*. *a i* ∗ *z*^*i*) = *0*
using *assms*
proof (*elim disjE bexE*)
 assume *a 0* = *0* **then show** *?thesis*
  by (*auto simp*: *that* [*of 0*])
next
 fix *i*
 assume *i*: *i* ∈ {*1..n*} **and** *nz*: *a i* ≠ *0*
 have *1*: (λ*z*. ∑ *i*≤*n*. *a i* ∗ *z*^*i*) *holomorphic_on UNIV*
  by (*rule holomorphic_intros*)+
 show *thesis*
 proof (*rule Liouville_weak_inverse* [*OF 1*])
  show ∀ *F x in at_infinity*. *B* ≤ *cmod* (∑ *i*≤*n*. *a i* ∗ *x* ^ *i*) **for** *B*
   using *i nz* **by** (*intro polyfun_extremal exI*[*of* _ *i*]) *auto*
 qed (*use that* **in** *auto*)
qed

4.8 Weierstrass convergence theorem

lemma *holomorphic_uniform_limit*:
 assumes *cont*: *eventually* (λ*n*. *continuous_on* (*cball z r*) (*f n*) ∧ (*f n*) *holomorphic_on ball z r*) *F*
   and *ulim*: *uniform_limit* (*cball z r*) *f g F*
   and *F*: ¬ *trivial_limit F*
 obtains *continuous_on* (*cball z r*) *g g holomorphic_on ball z r*
proof (*cases r 0*::*real rule*: *linorder_cases*)
 case *less* **then show** *?thesis* **by** (*force simp*: *ball_empty less_imp_le continuous_on_def holomorphic_on_def intro*: *that*)
next
 case *equal* **then show** *?thesis*
  by (*force simp*: *holomorphic_on_def intro*: *that*)
next
 case *greater*
 have *contg*: *continuous_on* (*cball z r*) *g*
  using *cont uniform_limit_theorem* [*OF eventually_mono ulim F*] **by** *blast*
 have *path_image* (*circlepath z r*) ⊆ *cball z r*
  using ‹*0* < *r*› **by** *auto*
 then have *1*: *continuous_on* (*path_image* (*circlepath z r*)) (λ*x*. *1* / (*2* ∗ *complex_of_real pi* ∗ i) ∗ *g x*)
  by (*intro continuous_intros continuous_on_subset* [*OF contg*])
 have *2*: ((λ*u*. *1* / (*2* ∗ *of_real pi* ∗ i) ∗ *g u* / (*u* − *w*) ^ *1*) *has_contour_integral g w*) (*circlepath z r*)
   if *w*: *w* ∈ *ball z r* **for** *w*
 proof −
  define *d* **where** *d* = (*r* − *norm*(*w* − *z*))

have $0 < d$ $d \le r$ **using** w **by** (*auto simp*: *norm_minus_commute d_def dist_norm*)
have *dle*: $\bigwedge u.\ cmod\ (z - u) = r \Longrightarrow d \le cmod\ (u - w)$
unfolding *d_def* **by** (*metis add_diff_eq diff_add_cancel norm_diff_ineq norm_minus_commute*)
have *ev_int*: $\forall_F\ n\ in\ F.\ (\lambda u.\ f\ n\ u\ /\ (u - w))\ contour\_integrable\_on\ circlepath\ z\ r$
using w
by (*auto intro*: *eventually_mono* [*OF cont*] *Cauchy_higher_derivative_integral_circlepath* [**where** *k*=*0*, *simplified*])
have $\bigwedge e.\ [\![0 < r;\ 0 < d;\ 0 < e]\!] \Longrightarrow \forall_F\ n\ in\ F.\ \forall x{\in}sphere\ z\ r.\ x \neq w \longrightarrow cmod\ (f\ n\ x - g\ x) < e * cmod\ (x - w)$
apply (*rule_tac e1*=$e * d$ **in** *eventually_mono* [*OF uniform_limitD* [*OF ulim*]])
apply (*force simp*: *dist_norm intro*: *dle mult_left_mono less_le_trans*)+
done
then have *ul_less*: *uniform_limit* (*sphere z r*) ($\lambda n\ x.\ f\ n\ x\ /\ (x - w)$) ($\lambda x.\ g\ x\ /\ (x - w)$) *F*
using *greater* ‹*0 < d*›
by (*auto simp add*: *uniform_limit_iff dist_norm norm_divide diff_divide_distrib* [*symmetric*] *divide_simps*)
have *g_cint*: ($\lambda u.\ g\ u/(u - w)$) *contour_integrable_on circlepath z r*
by (*rule contour_integral_uniform_limit_circlepath* [*OF ev_int ul_less F* ‹*0 < r*›])
have *cif_tends_cig*: (($\lambda n.$ *contour_integral*(*circlepath z r*) ($\lambda u.\ f\ n\ u\ /\ (u - w)$))) $\longrightarrow$ *contour_integral*(*circlepath z r*) ($\lambda u.\ g\ u/(u - w)$)) *F*
by (*rule contour_integral_uniform_limit_circlepath* [*OF ev_int ul_less F* ‹*0 < r*›])
have *f_tends_cig*: (($\lambda n.$ *2 * of_real pi* * i * *f n w*) $\longrightarrow$ *contour_integral* (*circlepath z r*) ($\lambda u.\ g\ u\ /\ (u - w)$))) *F*
proof (*rule Lim_transform_eventually*)
show $\forall_F\ x\ in\ F.$ *contour_integral* (*circlepath z r*) ($\lambda u.\ f\ x\ u\ /\ (u - w)$) = *2 * of_real pi* * i * *f x w*
using *w*‹*0 < d*› *d_def*
by (*auto intro*: *eventually_mono* [*OF cont contour_integral_unique* [*OF Cauchy_integral_circlepath*]])
qed (*auto simp*: *cif_tends_cig*)
have $\bigwedge e.\ 0 < e \Longrightarrow \forall_F\ n\ in\ F.\ dist\ (f\ n\ w)\ (g\ w) < e$
by (*rule eventually_mono* [*OF uniform_limitD* [*OF ulim*]]) (*use w* **in** *auto*)
then have (($\lambda n.$ *2 * of_real pi* * i * *f n w*) $\longrightarrow$ *2 * of_real pi* * i * *g w*) *F*
by (*rule tendsto_mult_left* [*OF tendstoI*])
then have (($\lambda u.\ g\ u\ /\ (u - w)$)) *has_contour_integral 2 * of_real pi* * i * *g w*) (*circlepath z r*)
using *has_contour_integral_integral* [*OF g_cint*] *tendsto_unique* [*OF F f_tends_cig*] *w*
by *fastforce*

```
    then have ((λu. g u / (2 * of_real pi * i * (u - w))) has_contour_integral g
w) (circlepath z r)
      using has_contour_integral_div [where c = 2 * of_real pi * i]
      by (force simp: field_simps)
    then show ?thesis
      by (simp add: dist_norm)
  qed
  show ?thesis
    using Cauchy_next_derivative_circlepath(2) [OF 1 2, simplified]
    by (fastforce simp add: holomorphic_on_open contg intro: that)
qed
```

Version showing that the limit is the limit of the derivatives.

```
proposition has_complex_derivative_uniform_limit:
  fixes z::complex
  assumes cont: eventually (λn. continuous_on (cball z r) (f n) ∧
                                (∀ w ∈ ball z r. ((f n) has_field_derivative (f' n w)) (at
w))) F
      and ulim: uniform_limit (cball z r) f g F
      and F: ¬ trivial_limit F and 0 < r
  obtains g' where
      continuous_on (cball z r) g
      ⋀w. w ∈ ball z r ⟹ (g has_field_derivative (g' w)) (at w) ∧ ((λn. f' n w)
⟶ g' w) F
proof −
  let ?conint = contour_integral (circlepath z r)
  have g: continuous_on (cball z r) g g holomorphic_on ball z r
    by (rule holomorphic_uniform_limit [OF eventually_mono [OF cont] ulim F];
          auto simp: holomorphic_on_open field_differentiable_def)+
  then obtain g' where g': ⋀x. x ∈ ball z r ⟹ (g has_field_derivative g' x) (at
x)
    using DERIV_deriv_iff_has_field_derivative
    by (fastforce simp add: holomorphic_on_open)
  then have derg: ⋀x. x ∈ ball z r ⟹ deriv g x = g' x
    by (simp add: DERIV_imp_deriv)
  have tends_f'n_g': ((λn. f' n w) ⟶ g' w) F if w: w ∈ ball z r for w
  proof −
    have eq_f': ?conint (λx. f n x / (x - w)^2) - ?conint (λx. g x / (x - w)^2) =
(f' n w - g' w) * (2 * of_real pi * i)
          if cont_fn: continuous_on (cball z r) (f n)
          and fnd: ⋀w. w ∈ ball z r ⟹ (f n has_field_derivative f' n w) (at w)
for n
    proof −
      have hol_fn: f n holomorphic_on ball z r
        using fnd by (force simp: holomorphic_on_open)
      have (f n has_field_derivative 1 / (2 * of_real pi * i) * ?conint (λu. f n u
/ (u - w)^2)) (at w)
        by (rule Cauchy_derivative_integral_circlepath [OF cont_fn hol_fn w])
      then have f': f' n w = 1 / (2 * of_real pi * i) * ?conint (λu. f n u / (u -
```

```
w)^2)
      using DERIV_unique [OF fnd] w by blast
    show ?thesis
     by (simp add: f' Cauchy_contour_integral_circlepath_2 [OF g w] derg [OF
w] field_split_simps)
  qed
  define d where d = (r - norm(w - z))^2
  have d > 0
    using w by (simp add: dist_commute dist_norm d_def)
  have dle: d ≤ cmod ((y - w)^2) if r = cmod (z - y) for y
  proof -
    have w ∈ ball z (cmod (z - y))
      using that w by fastforce
    then have cmod (w - z) ≤ cmod (z - y)
      by (simp add: dist_complex_def norm_minus_commute)
    moreover have cmod (z - y) - cmod (w - z) ≤ cmod (y - w)
    by (metis diff_add_cancel diff_add_eq_diff_diff_swap norm_minus_commute
norm_triangle_ineq2)
    ultimately show ?thesis
      using that by (simp add: d_def norm_power power_mono)
  qed
  have 1: ∀F n in F. (λx. f n x / (x - w)^2) contour_integrable_on circlepath z
r
   by (force simp: holomorphic_on_open intro: w Cauchy_derivative_integral_circlepath
eventually_mono [OF cont])
  have 2: uniform_limit (sphere z r) (λn x. f n x / (x - w)^2) (λx. g x / (x -
w)^2) F
    unfolding uniform_limit_iff
  proof clarify
    fix e::real
    assume e > 0
    with ‹r > 0›
    have ∀F n in F. ∀x. x ≠ w ⟶ cmod (z - x) = r ⟶ cmod (f n x - g x)
< e * cmod ((x - w)^2)
     by (force simp: ‹0 < d› dist_norm dle intro: less_le_trans eventually_mono
[OF uniform_limitD [OF ulim], of e*d])
    with ‹r > 0› ‹e > 0›
   show ∀F n in F. ∀x∈sphere z r. dist (f n x / (x - w)^2) (g x / (x - w)^2) < e
      by (simp add: norm_divide field_split_simps sphere_def dist_norm)
  qed
  have ((λn. contour_integral (circlepath z r) (λx. f n x / (x - w)^2))
        ⟶ contour_integral (circlepath z r) ((λx. g x / (x - w)^2))) F
    by (rule contour_integral_uniform_limit_circlepath [OF 1 2 F ‹0 < r›])
  then have tendsto_0: ((λn. 1 / (2 * of_real pi * i) * (?conint (λx. f n x / (x
- w)^2) - ?conint (λx. g x / (x - w)^2))) ⟶ 0) F
    using Lim_null by (force intro!: tendsto_mult_right_zero)
  have ((λn. f' n w - g' w) ⟶ 0) F
    apply (rule Lim_transform_eventually [OF tendsto_0])
    apply (force simp: divide_simps intro: eq_f' eventually_mono [OF cont])
```

```
    done
  then show ?thesis using Lim_null by blast
 qed
 obtain g' where ⋀w. w ∈ ball z r ⟹ (g has_field_derivative (g' w)) (at w) ∧
((λn. f' n w) ⟶ g' w) F
    by (blast intro: tends_f'n_g' g')
 then show ?thesis using g
  using that by blast
qed
```

4.9 Some more simple/convenient versions for applications

```
lemma holomorphic_uniform_sequence:
 assumes S: open S
    and hol_fn: ⋀n. (f n) holomorphic_on S
     and ulim_g: ⋀x. x ∈ S ⟹ ∃ d. 0 < d ∧ cball x d ⊆ S ∧ uniform_limit
(cball x d) f g sequentially
 shows g holomorphic_on S
proof −
 have ∃f'. (g has_field_derivative f') (at z) if z ∈ S for z
 proof −
  obtain r where 0 < r and r: cball z r ⊆ S
            and ul: uniform_limit (cball z r) f g sequentially
    using ulim_g [OF ‹z ∈ S›] by blast
   have ∗: ∀F n in sequentially. continuous_on (cball z r) (f n) ∧ f n holomor-
phic_on ball z r
   proof (intro eventuallyI conjI)
    show continuous_on (cball z r) (f x) for x
     using hol_fn holomorphic_on_imp_continuous_on holomorphic_on_subset
r by blast
    show f x holomorphic_on ball z r for x
     by (metis hol_fn holomorphic_on_subset interior_cball interior_subset r)
   qed
   show ?thesis
    using ‹0 < r› centre_in_ball ul
    by (auto simp: holomorphic_on_open intro: holomorphic_uniform_limit [OF
∗])
 qed
 with S show ?thesis
  by (simp add: holomorphic_on_open)
qed

lemma has_complex_derivative_uniform_sequence:
 fixes S :: complex set
 assumes S: open S
    and hfd: ⋀n x. x ∈ S ⟹ ((f n) has_field_derivative f' n x) (at x)
    and ulim_g: ⋀x. x ∈ S
        ⟹ ∃ d. 0 < d ∧ cball x d ⊆ S ∧ uniform_limit (cball x d) f g sequentially
 shows ∃ g'. ∀ x ∈ S. (g has_field_derivative g' x) (at x) ∧ ((λn. f' n x) ⟶
```

```
g' x) sequentially
proof −
  have y: ∃y. (g has_field_derivative y) (at z) ∧ (λn. f' n z) ⟶ y if z ∈ S
for z
  proof −
    obtain r where 0 < r and r: cball z r ⊆ S
              and ul: uniform_limit (cball z r) f g sequentially
      using ulim_g [OF ‹z ∈ S›] by blast
    have ∗: ∀F n in sequentially. continuous_on (cball z r) (f n) ∧
                                  (∀w ∈ ball z r. ((f n) has_field_derivative (f' n w))
(at w))
    proof (intro eventuallyI conjI ballI)
      show continuous_on (cball z r) (f x) for x
      by (meson S continuous_on_subset hfd holomorphic_on_imp_continuous_on
holomorphic_on_open r)
      show w ∈ ball z r ⟹ (f x has_field_derivative f' x w) (at w) for w x
        using ball_subset_cball hfd r by blast
    qed
    show ?thesis
      by (rule has_complex_derivative_uniform_limit [OF ∗, of g]) (use ‹0 < r›
ul in ‹force+›)
  qed
  show ?thesis
    by (rule bchoice) (blast intro: y)
qed
```

4.10 On analytic functions defined by a series

```
lemma series_and_derivative_comparison:
  fixes S :: complex set
  assumes S: open S
      and h: summable h
      and hfd: ⋀n x. x ∈ S ⟹ (f n has_field_derivative f' n x) (at x)
      and to_g: ∀F n in sequentially. ∀x∈S. norm (f n x) ≤ h n
  obtains g g' where ∀x ∈ S. ((λn. f n x) sums g x) ∧ ((λn. f' n x) sums g' x)
∧ (g has_field_derivative g' x) (at x)
proof −
  obtain g where g: uniform_limit S (λn x. ∑i<n. f i x) g sequentially
    using Weierstrass_m_test_ev [OF to_g h] by force
  have ∗: ∃d>0. cball x d ⊆ S ∧ uniform_limit (cball x d) (λn x. ∑i<n. f i x)
g sequentially
        if x ∈ S for x
  proof −
    obtain d where d>0 and d: cball x d ⊆ S
      using open_contains_cball [of S] ‹x ∈ S› S by blast
    show ?thesis
    proof (intro conjI exI)
      show uniform_limit (cball x d) (λn x. ∑i<n. f i x) g sequentially
          using d g uniform_limit_on_subset by (force simp: dist_norm eventu-
```

ally_sequentially)
  qed (*use* ‹$d > 0$› d **in** *auto*)
 qed
 have $\bigwedge x.\ x \in S \Longrightarrow (\lambda n.\ \sum i{<}n.\ f\ i\ x) \longrightarrow g\ x$
  by (*metis tendsto_uniform_limitI* [*OF g*])
 moreover have $\exists\, g'.\ \forall\, x{\in}S.\ (g\ has\_field\_derivative\ g'\ x)\ (at\ x) \wedge (\lambda n.\ \sum i{<}n.\ f'\ i\ x) \longrightarrow g'\ x$
  by (*rule has_complex_derivative_uniform_sequence* [*OF S*]) (*auto intro*: $*$ *hfd DERIV_sum*)+
 ultimately show *?thesis*
  by (*metis sums_def that*)
qed

A version where we only have local uniform/comparative convergence.

lemma *series_and_derivative_comparison_local*:
 fixes S :: *complex set*
 assumes S: *open S*
   and *hfd*: $\bigwedge n\ x.\ x \in S \Longrightarrow (f\ n\ has\_field\_derivative\ f'\ n\ x)\ (at\ x)$
   and *to_g*: $\bigwedge x.\ x \in S \Longrightarrow \exists\, d\ h.\ 0 < d \wedge summable\ h \wedge (\forall_F\ n\ in\ sequentially.\ \forall\, y{\in}ball\ x\ d \cap S.\ norm\ (f\ n\ y) \leq h\ n)$
 shows $\exists\, g\ g'.\ \forall\, x \in S.\ ((\lambda n.\ f\ n\ x)\ sums\ g\ x) \wedge ((\lambda n.\ f'\ n\ x)\ sums\ g'\ x) \wedge (g\ has\_field\_derivative\ g'\ x)\ (at\ x)$
proof −
 have $\exists\, y.\ (\lambda n.\ f\ n\ z)\ sums\ (\sum n.\ f\ n\ z) \wedge (\lambda n.\ f'\ n\ z)\ sums\ y \wedge ((\lambda x.\ \sum n.\ f\ n\ x)\ has\_field\_derivative\ y)\ (at\ z)$
   if $z \in S$ **for** z
 proof −
  obtain $d\ h$ **where** $0 < d$ *summable h* **and** *le_h*: $\forall_F\ n\ in\ sequentially.\ \forall\, y{\in}ball\ z\ d \cap S.\ norm\ (f\ n\ y) \leq h\ n$
   using *to_g* ‹$z \in S$› **by** *meson*
  then obtain r **where** $r{>}0$ **and** r: *ball z r* $\subseteq$ *ball z d* $\cap$ S **using** ‹$z \in S$› S
   by (*metis Int_iff open_ball centre_in_ball open_Int open_contains_ball_eq*)
  have *1*: *open* (*ball z d* $\cap$ S)
   by (*simp add*: *open_Int S*)
  have *2*: $\bigwedge n\ x.\ x \in ball\ z\ d \cap S \Longrightarrow (f\ n\ has\_field\_derivative\ f'\ n\ x)\ (at\ x)$
   by (*auto simp*: *hfd*)
  obtain $g\ g'$ **where** gg': $\forall\, x \in ball\ z\ d \cap S.\ ((\lambda n.\ f\ n\ x)\ sums\ g\ x) \wedge ((\lambda n.\ f'\ n\ x)\ sums\ g'\ x) \wedge (g\ has\_field\_derivative\ g'\ x)\ (at\ x)$
   by (*auto intro*: *le_h series_and_derivative_comparison* [*OF 1* ‹*summable h*› *hfd*])
  then have $(\lambda n.\ f'\ n\ z)\ sums\ g'\ z$
   by (*meson* ‹$0 < r$› *centre_in_ball contra_subsetD r*)
  moreover have $(\lambda n.\ f\ n\ z)\ sums\ (\sum n.\ f\ n\ z)$
   using *summable_sums centre_in_ball* ‹$0 < d$› ‹*summable h*› *le_h*
   by (*metis* (*full_types*) *Int_iff gg' summable_def that*)
  moreover have $((\lambda x.\ \sum n.\ f\ n\ x)\ has\_field\_derivative\ g'\ z)\ (at\ z)$
  proof (*rule has_field_derivative_transform_within*)
   show $\bigwedge x.\ dist\ x\ z < r \Longrightarrow g\ x = (\sum n.\ f\ n\ x)$

by (*metis subsetD dist_commute gg' mem_ball r sums_unique*)
qed (*use* ‹*0 < r*› *gg'* ‹*z ∈ S*› ‹*0 < d*› **in** *auto*)
ultimately show *?thesis* **by** *auto*
qed
then show *?thesis*
by (*rule_tac x*=*λx. suminf* (*λn. f n x*) **in** *exI*) *meson*
qed

Sometimes convenient to compare with a complex series of positive reals. (?)

lemma *series_and_derivative_comparison_complex*:
fixes *S* :: *complex set*
assumes *S*: *open S*
and *hfd*: ⋀*n x. x ∈ S* ⟹ (*f n has_field_derivative f' n x*) (*at x*)
and *to_g*: ⋀*x. x ∈ S* ⟹ ∃ *d h. 0 < d* ∧ *summable h* ∧ *range h* ⊆ ℝ≥0 ∧ (∀*F n in sequentially. ∀ y*∈*ball x d* ∩ *S. cmod*(*f n y*) ≤ *cmod* (*h n*))
shows ∃ *g g'*. ∀ *x* ∈ *S*. ((*λn. f n x*) *sums g x*) ∧ ((*λn. f' n x*) *sums g' x*) ∧ (*g has_field_derivative g' x*) (*at x*)
apply (*rule series_and_derivative_comparison_local* [*OF S hfd*], *assumption*)
apply (*rule ex_forward* [*OF to_g*], *assumption*)
apply (*erule exE*)
apply (*rule_tac x*=*Re* ∘ *h* **in** *exI*)
apply (*force simp*: *summable_Re o_def nonneg_Reals_cmod_eq_Re image_subset_iff*)
done

Sometimes convenient to compare with a complex series of positive reals. (?)

lemma *series_differentiable_comparison_complex*:
fixes *S* :: *complex set*
assumes *S*: *open S*
and *hfd*: ⋀*n x. x ∈ S* ⟹ *f n field_differentiable* (*at x*)
and *to_g*: ⋀*x. x ∈ S* ⟹ ∃ *d h. 0 < d* ∧ *summable h* ∧ *range h* ⊆ ℝ≥0 ∧ (∀*F n in sequentially. ∀ y*∈*ball x d* ∩ *S. cmod*(*f n y*) ≤ *cmod* (*h n*))
obtains *g* **where** ∀ *x* ∈ *S*. ((*λn. f n x*) *sums g x*) ∧ *g field_differentiable* (*at x*)
proof −
have *hfd'*: ⋀*n x. x ∈ S* ⟹ (*f n has_field_derivative deriv* (*f n*) *x*) (*at x*)
using *hfd field_differentiable_derivI* **by** *blast*
have ∃ *g g'*. ∀ *x* ∈ *S*. ((*λn. f n x*) *sums g x*) ∧ ((*λn. deriv* (*f n*) *x*) *sums g' x*) ∧ (*g has_field_derivative g' x*) (*at x*)
by (*metis series_and_derivative_comparison_complex* [*OF S hfd' to_g*])
then show *?thesis*
using *field_differentiable_def that* **by** *blast*
qed

In particular, a power series is analytic inside circle of convergence.

lemma *power_series_and_derivative_0*:
fixes *a* :: *nat* ⇒ *complex* **and** *r*::*real*
assumes *summable* (*λn. a n* ∗ *r*^*n*)

```
    shows ∃g g'. ∀z. cmod z < r ⟶
            ((λn. a n * z^n) sums g z) ∧ ((λn. of_nat n * a n * z^(n - 1)) sums
g' z) ∧ (g has_field_derivative g' z) (at z)
proof (cases 0 < r)
  case True
    have der: ⋀n z. ((λx. a n * x ^ n) has_field_derivative of_nat n * a n * z ^
(n - 1)) (at z)
      by (rule derivative_eq_intros | simp)+
    have y_le: cmod y ≤ cmod (of_real r + of_real (cmod z)) / 2
      if cmod (z - y) * 2 < r - cmod z for z y
    proof -
      have cmod y * 2 ≤ r + cmod z
        using norm_triangle_ineq2 [of y z] that
        by (simp only: diff_le_eq norm_minus_commute mult_2)
      then show ?thesis
        using ‹r > 0›
        by (auto simp: algebra_simps norm_mult norm_divide norm_power simp
flip: of_real_add)
    qed
    have summable (λn. a n * complex_of_real r ^ n)
      using assms ‹r > 0› by simp
    moreover have ⋀z. cmod z < r ⟹ cmod ((of_real r + of_real (cmod z)) /
2) < cmod (of_real r)
      using ‹r > 0›
      by (simp flip: of_real_add)
    ultimately have sum: ⋀z. cmod z < r ⟹ summable (λn. of_real (cmod (a
n)) * ((of_real r + complex_of_real (cmod z)) / 2) ^ n)
      by (rule power_series_conv_imp_absconv_weak)
    have ∃g g'. ∀z ∈ ball 0 r. (λn. (a n) * z ^ n) sums g z ∧
            (λn. of_nat n * (a n) * z ^ (n - 1)) sums g' z ∧ (g has_field_derivative
g' z) (at z)
     apply (rule series_and_derivative_comparison_complex [OF open_ball der])
      apply (rule_tac x=(r - norm z)/2 in exI)
      apply (rule_tac x=λn. of_real(norm(a n)*((r + norm z)/2)^n) in exI)
      using ‹r > 0›
        apply (auto simp: sum eventually_sequentially norm_mult norm_power
dist_norm intro!: mult_left_mono power_mono y_le)
      done
  then show ?thesis
    by (simp add: ball_def)
next
  case False then show ?thesis
    unfolding not_less
    using less_le_trans norm_not_less_zero by blast
qed

proposition power_series_and_derivative:
  fixes a :: nat ⇒ complex and r::real
  assumes summable (λn. a n * r^n)
```

```
    obtains g g' where ∀z ∈ ball w r.
            ((λn. a n * (z - w) ^ n) sums g z) ∧ ((λn. of_nat n * a n * (z - w) ^
(n - 1)) sums g' z) ∧
             (g has_field_derivative g' z) (at z)
  using power_series_and_derivative_0 [OF assms]
  apply clarify
  apply (rule_tac g=(λz. g(z - w)) in that)
  using DERIV_shift [where z=-w]
  apply (auto simp: norm_minus_commute Ball_def dist_norm)
  done

proposition power_series_holomorphic:
  assumes ⋀w. w ∈ ball z r ⟹ ((λn. a n*(w - z)^n) sums f w)
    shows f holomorphic_on ball z r
proof -
  have ∃f'. (f has_field_derivative f') (at w) if w: dist z w < r for w
  proof -
    have inb: z + complex_of_real ((dist z w + r) / 2) ∈ ball z r
    proof -
      have wz: cmod (w - z) < r using w
        by (auto simp: field_split_simps dist_norm norm_minus_commute)
      then have 0 ≤ r
        by (meson less_eq_real_def norm_ge_zero order_trans)
      show ?thesis
        using w by (simp add: dist_norm ‹0≤r› flip: of_real_add)
    qed
    have sum: summable (λn. a n * of_real (((cmod (z - w) + r) / 2) ^ n))
      using assms [OF inb] by (force simp: summable_def dist_norm)
    obtain g g' where gg': ⋀u. u ∈ ball z ((cmod (z - w) + r) / 2) ⟹
                           (λn. a n * (u - z) ^ n) sums g u ∧
                           (λn. of_nat n * a n * (u - z) ^ (n - 1)) sums g' u ∧ (g
has_field_derivative g' u) (at u)
      by (rule power_series_and_derivative [OF sum, of z]) fastforce
    have [simp]: g u = f u if cmod (u - w) < (r - cmod (z - w)) / 2 for u
    proof -
      have less: cmod (z - u) * 2 < cmod (z - w) + r
        using that dist_triangle2 [of z u w]
        by (simp add: dist_norm [symmetric] algebra_simps)
      have (λn. a n * (u - z) ^ n) sums g u (λn. a n * (u - z) ^ n) sums f u
        using gg' [of u] less w by (auto simp: assms dist_norm)
      then show ?thesis
        by (metis sums_unique2)
    qed
    have (f has_field_derivative g' w) (at w)
      by (rule has_field_derivative_transform_within [where d=(r - norm(z -
w))/2])
     (use w gg' [of w] in ‹(force simp: dist_norm)+›)
    then show ?thesis ..
  qed
```

then show *?thesis* **by** (*simp add*: *holomorphic_on_open*)
qed

corollary *holomorphic_iff_power_series*:
  f holomorphic_on ball z r ⟷
   (∀ *w* ∈ *ball z r*. (λ*n*. (*deriv* ^^ *n*) *f z* / (*fact n*) ∗ (*w* − *z*) ^ *n*) *sums f w*)
 apply (*intro iffI ballI holomorphic_power_series*, *assumption*+)
 apply (*force intro*: *power_series_holomorphic* [**where** *a* = λ*n*. (*deriv* ^^ *n*) *f z* / (*fact n*)])
 done

lemma *power_series_analytic*:
  (⋀*w*. *w* ∈ *ball z r* ⟹ (λ*n*. *a n*∗(*w* − *z*) ^ *n*) *sums f w*) ⟹ *f analytic_on ball z r*
 by (*force simp*: *analytic_on_open intro*!: *power_series_holomorphic*)

lemma *analytic_iff_power_series*:
  f analytic_on ball z r ⟷
   (∀ *w* ∈ *ball z r*. (λ*n*. (*deriv* ^^ *n*) *f z* / (*fact n*) ∗ (*w* − *z*) ^ *n*) *sums f w*)
 by (*simp add*: *analytic_on_open holomorphic_iff_power_series*)

4.11 Equality between holomorphic functions, on open ball then connected set

lemma *holomorphic_fun_eq_on_ball*:
 ⟦*f holomorphic_on ball z r*; *g holomorphic_on ball z r*;
  w ∈ *ball z r*;
  ⋀*n*. (*deriv* ^^ *n*) *f z* = (*deriv* ^^ *n*) *g z*⟧
  ⟹ *f w* = *g w*
 by (*auto simp*: *holomorphic_iff_power_series sums_unique2* [*of* λ*n*. (*deriv* ^^ *n*) *f z* / (*fact n*) ∗ (*w* − *z*) ^ *n*])

lemma *holomorphic_fun_eq_0_on_ball*:
 ⟦*f holomorphic_on ball z r*; *w* ∈ *ball z r*;
  ⋀*n*. (*deriv* ^^ *n*) *f z* = 0⟧
  ⟹ *f w* = 0
 by (*auto simp*: *holomorphic_iff_power_series sums_unique2* [*of* λ*n*. (*deriv* ^^ *n*) *f z* / (*fact n*) ∗ (*w* − *z*) ^ *n*])

lemma *holomorphic_fun_eq_0_on_connected*:
 assumes *holf*: *f holomorphic_on S* **and** *open S*
  and *cons*: *connected S*
  and *der*: ⋀*n*. (*deriv* ^^ *n*) *f z* = 0
  and *z* ∈ *S w* ∈ *S*
 shows *f w* = 0
proof −
 have ∗: *ball x e* ⊆ (⋂ *n*. {*w* ∈ *S*. (*deriv* ^^ *n*) *f w* = 0})
  if ∀ *u*. (*deriv* ^^ *u*) *f x* = 0 *ball x e* ⊆ *S* **for** *x e*
 proof −

```
    have (deriv ^^ m) ((deriv ^^ n) f) x = 0 for m n
      by (metis funpow_add o_apply that(1))
    then have ⋀x' n. dist x x' < e ⟹ (deriv ^^ n) f x' = 0
      using ‹open S›
      by (meson holf holomorphic_fun_eq_0_on_ball holomorphic_higher_deriv
holomorphic_on_subset mem_ball that(2))
    with that show ?thesis by auto
  qed
  obtain e where e>0 and e: ball w e ⊆ S using openE [OF ‹open S› ‹w ∈ S›]
.
  then have holfb: f holomorphic_on ball w e
    using holf holomorphic_on_subset by blast
  have open (⋂n. {w ∈ S. (deriv ^^ n) f w = 0})
    using ‹open S›
    apply (simp add: open_contains_ball Ball_def)
    apply (erule all_forward)
    using ∗ by auto blast+
  then have openin (top_of_set S) (⋂n. {w ∈ S. (deriv ^^ n) f w = 0})
    by (force intro: open_subset)
  moreover have closedin (top_of_set S) (⋂n. {w ∈ S. (deriv ^^ n) f w = 0})
    using assms
   by (auto intro: continuous_closedin_preimage_constant holomorphic_on_imp_continuous_on
holomorphic_higher_deriv)
  moreover have (⋂n. {w ∈ S. (deriv ^^ n) f w = 0}) = S ⟹ f w = 0
    using ‹e>0› e by (force intro: holomorphic_fun_eq_0_on_ball [OF holfb])
  ultimately show ?thesis
    using cons der ‹z ∈ S›
    by (auto simp add: connected_clopen)
qed

lemma holomorphic_fun_eq_on_connected:
  assumes f holomorphic_on S g holomorphic_on S and open S  connected S
      and ⋀n. (deriv ^^ n) f z = (deriv ^^ n) g z
      and z ∈ S w ∈ S
    shows f w = g w
proof (rule holomorphic_fun_eq_0_on_connected [of λx. f x − g x S z, simpli-
fied])
  show (λx. f x − g x) holomorphic_on S
    by (intro assms holomorphic_intros)
  show ⋀n. (deriv ^^ n) (λx. f x − g x) z = 0
    using assms higher_deriv_diff by auto
qed (use assms in auto)

lemma holomorphic_fun_eq_const_on_connected:
  assumes holf: f holomorphic_on S and open S
      and cons: connected S
      and der: ⋀n. 0 < n ⟹ (deriv ^^ n) f z = 0
      and z ∈ S w ∈ S
    shows f w = f z
```

proof (*rule holomorphic_fun_eq_0_on_connected* [*of* $\lambda w.\ f\ w - f\ z\ S\ z$, *simplified*])
show $(\lambda w.\ f\ w - f\ z)$ *holomorphic_on S*
by (*intro assms holomorphic_intros*)
show $\bigwedge n.\ (deriv\ \widehat{\ \ }\ n)\ (\lambda w.\ f\ w - f\ z)\ z = 0$
by (*subst higher_deriv_diff*) (*use assms* **in** ‹*auto intro*: *holomorphic_intros*›)
qed (*use assms* **in** *auto*)

4.12 Some basic lemmas about poles/singularities

lemma *pole_lemma*:
assumes *holf*: *f holomorphic_on S* **and** *a*: $a \in interior\ S$
shows $(\lambda z.\ if\ z = a\ then\ deriv\ f\ a$
else $(f\ z - f\ a)\ /\ (z - a))$ *holomorphic_on S* (**is** *?F holomorphic_on S*)
proof −
have *F1*: *?F field_differentiable* (*at u within S*) **if** $u \in S\ u \neq a$ **for** *u*
proof −
have *fcd*: *f field_differentiable at u within S*
using *holf holomorphic_on_def* **by** (*simp add*: ‹$u \in S$›)
have *cd*: $(\lambda z.\ (f\ z - f\ a)\ /\ (z - a))$ *field_differentiable at u within S*
by (*rule fcd derivative_intros* | *simp add*: *that*)+
have $0 < dist\ a\ u$ **using** *that dist_nz* **by** *blast*
then show *?thesis*
by (*rule field_differentiable_transform_within* [*OF _ _ _ cd*]) (*auto simp*: ‹$u \in S$›)
qed
have *F2*: *?F field_differentiable at a* **if** $0 < e\ ball\ a\ e \subseteq S$ **for** *e*
proof −
have *holfb*: *f holomorphic_on ball a e*
by (*rule holomorphic_on_subset* [*OF holf* ‹*ball a e* ⊆ *S*›])
have *2*: *?F holomorphic_on ball a e* − {*a*}
using *mem_ball that*
by (*auto simp add*: *holomorphic_on_def simp flip*: *field_differentiable_def intro*: *F1 field_differentiable_within_subset*)
have *isCont* $(\lambda z.\ if\ z = a\ then\ deriv\ f\ a\ else\ (f\ z - f\ a)\ /\ (z - a))\ x$
if $dist\ a\ x < e$ **for** *x*
proof (*cases x=a*)
case *True*
then have *f field_differentiable at a*
using *holfb* ‹$0 < e$› *holomorphic_on_imp_differentiable_at* **by** *auto*
with *True* **show** *?thesis*
by (*auto simp*: *continuous_at has_field_derivative_iff simp flip*: *DERIV_deriv_iff_field_differentiable*
elim: *rev_iffD1* [*OF _ LIM_equal*])
next
case *False* **with** *2 that* **show** *?thesis*
by (*force simp*: *holomorphic_on_open open_Diff field_differentiable_def* [*symmetric*] *field_differentiable_imp_continuous_at*)

```
    qed
    then have 1: continuous_on (ball a e) ?F
      by (clarsimp simp: continuous_on_eq_continuous_at)
    have ?F holomorphic_on ball a e
      by (auto intro: no_isolated_singularity [OF 1 2])
    with that show ?thesis
      by (simp add: holomorphic_on_open field_differentiable_def [symmetric]
                    field_differentiable_at_within)
  qed
  show ?thesis
  proof
    fix x assume x ∈ S show ?F field_differentiable at x within S
    proof (cases x=a)
      case True then show ?thesis
       using a by (auto simp: mem_interior intro: field_differentiable_at_within
F2)
    next
      case False with F1 ‹x ∈ S› 
      show ?thesis by blast
    qed
  qed
qed

lemma pole_theorem:
  assumes holg: g holomorphic_on S and a: a ∈ interior S
      and eq: ⋀z. z ∈ S − {a} ⟹ g z = (z − a) ∗ f z
    shows (λz. if z = a then deriv g a
               else f z − g a/(z − a)) holomorphic_on S
  using pole_lemma [OF holg a]
  by (rule holomorphic_transform) (simp add: eq field_split_simps)

lemma pole_lemma_open:
  assumes f holomorphic_on S open S
    shows (λz. if z = a then deriv f a else (f z − f a)/(z − a)) holomorphic_on S
proof (cases a ∈ S)
  case True with assms interior_eq pole_lemma
    show ?thesis by fastforce
next
  case False with assms show ?thesis
     apply (simp add: holomorphic_on_def field_differentiable_def [symmetric],
clarify)
     apply (rule field_differentiable_transform_within [where f = λz. (f z − f
a)/(z − a) and d = 1])
    apply (rule derivative_intros | force)+
    done
qed

lemma pole_theorem_open:
  assumes holg: g holomorphic_on S and S: open S
```

```
    and eq: ⋀z. z ∈ S − {a} ⟹ g z = (z − a) ∗ f z
  shows (λz. if z = a then deriv g a
             else f z − g a/(z − a)) holomorphic_on S
 using pole_lemma_open [OF holg S]
 by (rule holomorphic_transform) (auto simp: eq divide_simps)

lemma pole_theorem_0:
 assumes holg: g holomorphic_on S and a: a ∈ interior S
    and eq: ⋀z. z ∈ S − {a} ⟹ g z = (z − a) ∗ f z
    and [simp]: f a = deriv g a g a = 0
  shows f holomorphic_on S
 using pole_theorem [OF holg a eq]
 by (rule holomorphic_transform) (auto simp: eq field_split_simps)

lemma pole_theorem_open_0:
 assumes holg: g holomorphic_on S and S: open S
    and eq: ⋀z. z ∈ S − {a} ⟹ g z = (z − a) ∗ f z
    and [simp]: f a = deriv g a g a = 0
  shows f holomorphic_on S
 using pole_theorem_open [OF holg S eq]
 by (rule holomorphic_transform) (auto simp: eq field_split_simps)

lemma pole_theorem_analytic:
 assumes g: g analytic_on S
    and eq: ⋀z. z ∈ S
          ⟹ ∃d. 0 < d ∧ (∀w ∈ ball z d − {a}. g w = (w − a) ∗ f w)
   shows (λz. if z = a then deriv g a else f z − g a/(z − a)) analytic_on S (is
?F analytic_on S)
 unfolding analytic_on_def
proof
 fix x
 assume x ∈ S
 with g obtain e where 0 < e and e: g holomorphic_on ball x e
   by (auto simp add: analytic_on_def)
 obtain d where 0 < d and d: ⋀w. w ∈ ball x d − {a} ⟹ g w = (w − a) ∗ f
w
   using ‹x ∈ S› eq by blast
 have ?F holomorphic_on ball x (min d e)
  using d e ‹x ∈ S› by (fastforce simp: holomorphic_on_subset subset_ball intro!:
pole_theorem_open)
 then show ∃e>0. ?F holomorphic_on ball x e
   using ‹0 < d› ‹0 < e› not_le by fastforce
qed

lemma pole_theorem_analytic_0:
 assumes g: g analytic_on S
    and eq: ⋀z. z ∈ S ⟹ ∃d. 0 < d ∧ (∀w ∈ ball z d − {a}. g w = (w − a) ∗
f w)
    and [simp]: f a = deriv g a g a = 0
```

shows *f analytic_on S*
proof −
have [*simp*]: (λ*z*. *if z = a then deriv g a else f z − g a / (z − a)*) = *f*
by *auto*
show *?thesis*
using *pole_theorem_analytic* [*OF g eq*] **by** *simp*
qed

lemma *pole_theorem_analytic_open_superset*:
assumes *g*: *g analytic_on S* **and** *S* ⊆ *T open T*
and *eq*: ⋀*z*. *z* ∈ *T* − {*a*} ⟹ *g z* = (*z* − *a*) ∗ *f z*
shows (λ*z*. *if z = a then deriv g a*
else f z − g a/(z − a)) *analytic_on S*
proof (*rule pole_theorem_analytic* [*OF g*])
fix *z*
assume *z* ∈ *S*
then obtain *e* **where** *0 < e* **and** *e*: *ball z e* ⊆ *T*
using *assms openE* **by** *blast*
then show ∃ *d*>*0*. ∀ *w*∈*ball z d* − {*a*}. *g w* = (*w* − *a*) ∗ *f w*
using *eq* **by** *auto*
qed

lemma *pole_theorem_analytic_open_superset_0*:
assumes *g*: *g analytic_on S S* ⊆ *T open T* ⋀*z*. *z* ∈ *T* − {*a*} ⟹ *g z* = (*z* − *a*) ∗ *f z*
and [*simp*]: *f a* = *deriv g a g a* = *0*
shows *f analytic_on S*
proof −
have [*simp*]: (λ*z*. *if z = a then deriv g a else f z − g a / (z − a)*) = *f*
by *auto*
have (λ*z*. *if z = a then deriv g a else f z − g a/(z − a)*) *analytic_on S*
by (*rule pole_theorem_analytic_open_superset* [*OF g*])
then show *?thesis* **by** *simp*
qed

4.13 General, homology form of Cauchy's theorem

Proof is based on Dixon's, as presented in Lang's "Complex Analysis" book (page 147).

lemma *contour_integral_continuous_on_linepath_2D*:
assumes *open U* **and** *cont_dw*: ⋀*w*. *w* ∈ *U* ⟹ *F w contour_integrable_on* (*linepath a b*)
and *cond_uu*: *continuous_on* (*U* × *U*) (λ(*x*,*y*). *F x y*)
and *abu*: *closed_segment a b* ⊆ *U*
shows *continuous_on U* (λ*w*. *contour_integral* (*linepath a b*) (*F w*))
proof −
have ∗: ∃ *d*>*0*. ∀ *x*′∈*U*. *dist x*′ *w* < *d* ⟶
dist (*contour_integral* (*linepath a b*) (*F x*′))
(*contour_integral* (*linepath a b*) (*F w*)) ≤ ε

```
        if w ∈ U 0 < ε a ≠ b for w ε
  proof −
    obtain δ where δ>0 and δ: cball w δ ⊆ U using open_contains_cball ‹open
U› ‹w ∈ U› by force
    let ?TZ = cball w δ  × closed_segment a b
    have uniformly_continuous_on ?TZ (λ(x,y). F x y)
    proof (rule compact_uniformly_continuous)
      show continuous_on ?TZ (λ(x,y). F x y)
        by (rule continuous_on_subset[OF cond_uu]) (use SigmaE δ abu in blast)
      show compact ?TZ
        by (simp add: compact_Times)
    qed
    then obtain η where η>0
        and η: ⋀x x'. ⟦x∈?TZ; x'∈?TZ; dist x' x < η⟧ ⟹
                      dist ((λ(x,y). F x y) x') ((λ(x,y). F x y) x) < ε/norm(b − a)
      using ‹0 < ε› ‹a ≠ b›
      by (auto elim: uniformly_continuous_onE [where e = ε/norm(b − a)])
    have η: ⟦norm (w − x1) ≤ δ;  x2 ∈ closed_segment a b;
             norm (w − x1') ≤ δ;  x2' ∈ closed_segment a b; norm ((x1', x2') −
(x1, x2)) < η⟧
            ⟹ norm (F x1' x2' − F x1 x2) ≤ ε / cmod (b − a)
           for x1 x2 x1' x2'
      using η [of (x1,x2) (x1',x2')] by (force simp: dist_norm)
    have le_ee: cmod (contour_integral (linepath a b) (λx. F x' x − F w x)) ≤ ε
          if x' ∈ U cmod (x' − w) < δ cmod (x' − w) < η  for x'
    proof −
      have (λx. F x' x − F w x) contour_integrable_on linepath a b
        by (simp add: ‹w ∈ U› cont_dw contour_integrable_diff that)
      then have cmod (contour_integral (linepath a b) (λx. F x' x − F w x)) ≤
ε/norm(b − a) ∗ norm(b − a)
        apply (rule has_contour_integral_bound_linepath [OF has_contour_integral_integral
_ η])
         using ‹0 < ε› ‹0 < δ› that by (auto simp: norm_minus_commute)
      also have ... = ε using ‹a ≠ b› by simp
      finally show ?thesis .
    qed
    show ?thesis
      apply (rule_tac x=min δ η in exI)
      using ‹0 < δ› ‹0 < η›
       by (auto simp: dist_norm contour_integral_diff [OF cont_dw cont_dw,
symmetric] ‹w ∈ U› intro: le_ee)
  qed
  show ?thesis
  proof (cases a=b)
    case False
    show ?thesis
      by (rule continuous_onI) (use False in ‹auto intro: ∗›)
  qed auto
qed
```

This version has *polynomial_function* γ as an additional assumption.

lemma *Cauchy_integral_formula_global_weak*:
assumes *open U* **and** *holf*: *f holomorphic_on U*
and *z*: $z \in U$ **and** γ: *polynomial_function* γ
and *pasz*: *path_image* $\gamma \subseteq U - \{z\}$ **and** *loop*: *pathfinish* γ = *pathstart* γ
and *zero*: $\bigwedge w.\ w \notin U \Longrightarrow$ *winding_number* $\gamma\ w = 0$
shows $((\lambda w.\ f\ w\ /\ (w - z))$ *has_contour_integral* $(2{*}pi * \mathrm{i} *$ *winding_number* $\gamma\ z * f\ z))\ \gamma$
proof −
obtain γ' **where** *pfγ'*: *polynomial_function* γ' **and** γ': $\bigwedge x.$ (γ *has_vector_derivative* $(\gamma'\ x))$ (*at x*)
using *has_vector_derivative_polynomial_function* [*OF* γ] **by** *blast*
then have *bounded*(*path_image* γ')
by (*simp add*: *path_image_def compact_imp_bounded compact_continuous_image continuous_on_polymonial_function*)
then obtain *B* **where** *B*>*0* **and** *B*: $\bigwedge x.\ x \in$ *path_image* $\gamma' \Longrightarrow$ *norm* $x \leq B$
using *bounded_pos* **by** *force*
define *d* **where** [*abs_def*]: *d z w* = (*if w* = *z then deriv f z else* (*f w* − *f z*)/(*w* − *z*)) **for** *z w*
define *v* **where** *v* = {*w*. *w* ∉ *path_image* $\gamma \wedge$ *winding_number* $\gamma\ w = 0$}
have *path* γ *valid_path* γ **using** γ
by (*auto simp*: *path_polynomial_function valid_path_polynomial_function*)
then have *ov*: *open v*
by (*simp add*: *v_def open_winding_number_levelsets loop*)
have *uv_Un*: *U* ∪ *v* = *UNIV*
using *pasz zero* **by** (*auto simp*: *v_def*)
have *conf*: *continuous_on U f*
by (*metis holf holomorphic_on_imp_continuous_on*)
have *hol_d*: (*d y*) *holomorphic_on U* **if** *y* ∈ *U* **for** *y*
proof −
have ∗: (λ*c*. *if c* = *y then deriv f y else* (*f c* − *f y*) / (*c* − *y*)) *holomorphic_on U*
by (*simp add*: *holf pole_lemma_open* ‹*open U*›)
then have *isCont* (λ*x*. *if x* = *y then deriv f y else* (*f x* − *f y*) / (*x* − *y*)) *y*
using *at_within_open field_differentiable_imp_continuous_at holomorphic_on_def that* ‹*open U*› **by** *fastforce*
then have *continuous_on U* (*d y*)
using ∗ *d_def holomorphic_on_imp_continuous_on* **by** *auto*
moreover have *d y holomorphic_on U* − {*y*}
proof −
have (λ*w*. *if w* = *y then deriv f y else* (*f w* − *f y*) / (*w* − *y*)) *field_differentiable at w*
if *w* ∈ *U* − {*y*} **for** *w*
proof (*rule field_differentiable_transform_within*)
show (λ*w*. (*f w* − *f y*) / (*w* − *y*)) *field_differentiable at w*
using *that* ‹*open U*› *holf*
by (*auto intro*!: *holomorphic_on_imp_differentiable_at derivative_intros*)
show *dist w y* > *0*
using *that* **by** *auto*

qed (*auto simp*: *dist_commute*)
then show *?thesis*
unfolding *field_differentiable_def* **by** (*simp add*: *d_def holomorphic_on_open* ‹*open U*› *open_delete*)
qed
ultimately show *?thesis*
by (*rule no_isolated_singularity*) (*auto simp*: ‹*open U*›)
qed
have *cint_fxy*: (λ*x*. (*f x* − *f y*) / (*x* − *y*)) *contour_integrable_on* γ **if** *y* ∉ *path_image* γ **for** *y*
proof (*rule contour_integrable_holomorphic_simple* [**where** *S* = *U*−{*y*}])
show (λ*x*. (*f x* − *f y*) / (*x* − *y*)) *holomorphic_on U* − {*y*}
by (*force intro*: *holomorphic_intros holomorphic_on_subset* [*OF holf*])
show *path_image* γ ⊆ *U* − {*y*}
using *pasz that* **by** *blast*
qed (*auto simp*: ‹*open U*› *open_delete* ‹*valid_path* γ›)
define *h* **where**
h z = (*if z* ∈ *U then contour_integral* γ (*d z*) *else contour_integral* γ (λ*w*. *f w*/(*w* − *z*))) **for** *z*
have *U*: ((*d z*) *has_contour_integral h z*) γ **if** *z* ∈ *U* **for** *z*
proof −
have *d z holomorphic_on U*
by (*simp add*: *hol_d that*)
with *that* **show** *?thesis*
by (*metis Diff_subset* ‹*valid_path* γ› ‹*open U*› *contour_integrable_holomorphic_simple h_def has_contour_integral_integral pasz subset_trans*)
qed
have *V*: ((λ*w*. *f w* / (*w* − *z*)) *has_contour_integral h z*) γ **if** *z*: *z* ∈ *v* **for** *z*
proof −
have *0*: *0* = (*f z*) ∗ *2* ∗ *of_real* (*2* ∗ *pi*) ∗ i ∗ *winding_number* γ *z*
using *v_def z* **by** *auto*
then have ((λ*x*. *1* / (*x* − *z*)) *has_contour_integral 0*) γ
using *z v_def has_contour_integral_winding_number* [*OF* ‹*valid_path* γ›]
by *fastforce*
then have ((λ*x*. *f z* ∗ (*1* / (*x* − *z*))) *has_contour_integral 0*) γ
using *has_contour_integral_lmul* **by** *fastforce*
then have ((λ*x*. *f z* / (*x* − *z*)) *has_contour_integral 0*) γ
by (*simp add*: *field_split_simps*)
moreover have ((λ*x*. (*f x* − *f z*) / (*x* − *z*)) *has_contour_integral contour_integral* γ (*d z*)) γ
using *z*
apply (*simp add*: *v_def*)
apply (*metis* (*no_types*, *lifting*) *contour_integrable_eq d_def has_contour_integral_eq has_contour_integral_integral cint_fxy*)
done
ultimately have ∗: ((λ*x*. *f z* / (*x* − *z*) + (*f x* − *f z*) / (*x* − *z*)) *has_contour_integral* (*0* + *contour_integral* γ (*d z*))) γ
by (*rule has_contour_integral_add*)
have ((λ*w*. *f w* / (*w* − *z*)) *has_contour_integral contour_integral* γ (*d z*)) γ

```
      if z ∈ U
      using ∗ by (auto simp: divide_simps has_contour_integral_eq)
    moreover have ((λw. f w / (w − z)) has_contour_integral contour_integral
γ (λw. f w / (w − z))) γ
      if z ∉ U
   proof (rule has_contour_integral_integral [OF contour_integrable_holomorphic_simple
[where S=U]])
      show (λw. f w / (w − z)) holomorphic_on U
        by (rule holomorphic_intros assms | use that in force)+
    qed (use ‹open U› pasz ‹valid_path γ› in auto)
    ultimately show ?thesis
      using z by (simp add: h_def)
  qed
  have znot: z ∉ path_image γ
    using pasz by blast
  obtain d0 where d0>0 and d0: ⋀x y. x ∈ path_image γ ⟹ y ∈ − U ⟹ d0
≤ dist x y
    using separate_compact_closed [of path_image γ −U] pasz ‹open U› ‹path γ›
compact_path_image
    by blast
  obtain dd where 0 < dd and dd: {y + k | y k. y ∈ path_image γ ∧ k ∈ ball 0
dd} ⊆ U
  proof
    show 0 < d0 / 2 using ‹0 < d0› by auto
  qed (use ‹0 < d0› d0 in ‹force simp: dist_norm›)
  define T where T ≡ {y + k |y k. y ∈ path_image γ ∧ k ∈ cball 0 (dd / 2)}
  have ⋀x x′. ⟦x ∈ path_image γ; dist x x′ ∗ 2 < dd⟧ ⟹ ∃ y k. x′ = y + k ∧ y
∈ path_image γ ∧ dist 0 k ∗ 2 ≤ dd
    apply (rule_tac x=x in exI)
    apply (rule_tac x=x′−x in exI)
    apply (force simp: dist_norm)
    done
  then have subt: path_image γ ⊆ interior T
    using ‹0 < dd›
    apply (clarsimp simp add: mem_interior T_def)
    apply (rule_tac x=dd/2 in exI, auto)
    done
  have compact T
    unfolding T_def
   by (fastforce simp add: ‹valid_path γ› compact_valid_path_image intro!: com-
pact_sums)
  have T: T ⊆ U
    unfolding T_def using ‹0 < dd› dd by fastforce
  obtain L where L>0
        and L: ⋀f B. ⟦f holomorphic_on interior T; ⋀z. z∈interior T ⟹ cmod
(f z) ≤ B⟧ ⟹
                    cmod (contour_integral γ f) ≤ L ∗ B
    using contour_integral_bound_exists [OF open_interior ‹valid_path γ› subt]
     by blast
```

```
  have bounded(f ‘ T)
    by (meson ‹compact T› compact_continuous_image compact_imp_bounded
conf continuous_on_subset T)
  then obtain D where D>0 and D: ⋀x. x ∈ T ⟹ norm (f x) ≤ D
    by (auto simp: bounded_pos)
  obtain C where C>0 and C: ⋀x. x ∈ T ⟹ norm x ≤ C
    using ‹compact T› bounded_pos compact_imp_bounded by force
  have dist (h y) 0 ≤ e if 0 < e and le: D * L / e + C ≤ cmod y for e y
  proof −
    have D * L / e > 0 using ‹D>0› ‹L>0› ‹e>0› by simp
    with le have ybig: norm y > C by force
    with C have y ∉ T by force
    then have ynot: y ∉ path_image γ
      using subt interior_subset by blast
    have [simp]: winding_number γ y = 0
    proof (rule winding_number_zero_outside)
      show path_image γ ⊆ cball 0 C
        by (meson C interior_subset mem_cball_0 subset_eq subt)
    qed (use ybig loop ‹path γ› in auto)
    have [simp]: h y = contour_integral γ (λw. f w/(w − y))
      by (rule contour_integral_unique [symmetric]) (simp add: v_def ynot V)
    have holint: (λw. f w / (w − y)) holomorphic_on interior T
    proof (intro holomorphic_intros)
      show f holomorphic_on interior T
        using holf holomorphic_on_subset interior_subset T by blast
    qed (use ‹y ∉ T› interior_subset in auto)
    have leD: cmod (f z / (z − y)) ≤ D * (e / L / D) if z: z ∈ interior T for z
    proof −
      have D * L / e + cmod z ≤ cmod y
        using le C [of z] z using interior_subset by force
      then have DL2: D * L / e ≤ cmod (z − y)
       using norm_triangle_ineq2 [of y z] by (simp add: norm_minus_commute)
      have cmod (f z / (z − y)) = cmod (f z) * inverse (cmod (z − y))
      by (simp add: norm_mult norm_inverse Fields.field_class.field_divide_inverse)
      also have ... ≤ D * (e / L / D)
      proof (rule mult_mono)
        show cmod (f z) ≤ D
          using D interior_subset z by blast
        show inverse (cmod (z − y)) ≤ e / L / D D ≥ 0
       using ‹L>0› ‹e>0› ‹D>0› DL2 by (auto simp: norm_divide field_split_simps)
      qed auto
      finally show ?thesis .
    qed
    have dist (h y) 0 = cmod (contour_integral γ (λw. f w / (w − y)))
      by (simp add: dist_norm)
    also have ... ≤ L * (D * (e / L / D))
      by (rule L [OF holint leD])
    also have ... = e
      using ‹L>0› ‹0 < D› by auto
```

```
    finally show ?thesis .
  qed
  then have (h ⟶ 0) at_infinity
    by (meson Lim_at_infinityI)
  moreover have h holomorphic_on UNIV
  proof −
    have con_ff: continuous (at (x,z)) (λ(x,y). (f y − f x) / (y − x))
                 if x ∈ U z ∈ U x ≠ z for x z
      using that conf
      apply (simp add: split_def continuous_on_eq_continuous_at ‹open U›)
      apply (simp | rule continuous_intros continuous_within_compose2 [where
g=f])+
      done
    have con_fstsnd: continuous_on UNIV (λx. (fst x − snd x) ::complex)
      by (rule continuous_intros)+
    have open_uu_Id: open (U × U − Id)
    proof (rule open_Diff)
      show open (U × U)
        by (simp add: open_Times ‹open U›)
      show closed (Id :: complex rel)
       using continuous_closed_preimage_constant [OF con_fstsnd closed_UNIV,
of 0]
        by (auto simp: Id_fstsnd_eq algebra_simps)
    qed
    have con_derf: continuous (at z) (deriv f) if z ∈ U for z
    proof (rule continuous_on_interior)
      show continuous_on U (deriv f)
       by (simp add: holf holomorphic_deriv holomorphic_on_imp_continuous_on
‹open U›)
    qed (simp add: interior_open that ‹open U›)
    have tendsto_f': ((λ(x,y). if y = x then deriv f (x)
                               else (f (y) − f (x)) / (y − x)) ⟶ deriv f x)
                     (at (x, x) within U × U) if x ∈ U for x
    proof (rule Lim_withinI)
      fix e::real assume 0 < e
      obtain k1 where k1>0 and k1: ⋀x'. norm (x' − x) ≤ k1 ⟹ norm (deriv
f x' − deriv f x) < e
        using ‹0 < e› continuous_within_E [OF con_derf [OF ‹x ∈ U›]]
        by (metis UNIV_I dist_norm)
      obtain k2 where k2>0 and k2: ball x k2 ⊆ U
        by (blast intro: openE [OF ‹open U›] ‹x ∈ U›)
      have neq: norm ((f z' − f x') / (z' − x') − deriv f x) ≤ e
                  if z' ≠ x' and less_k1: norm (x'−x, z'−x) < k1 and less_k2:
norm (x'−x, z'−x) < k2
                  for x' z'
      proof −
        have cs_less: w ∈ closed_segment x' z' ⟹ cmod (w − x) ≤ norm (x'−x,
z'−x) for w
          using segment_furthest_le [of w x' z' x]
```

by (*metis* (*no_types*) *dist_commute dist_norm norm_fst_le norm_snd_le order_trans*)
have *derf_le*: $w \in$ *closed_segment* x' $z' \Longrightarrow z' \neq x' \Longrightarrow$ *cmod* (*deriv f w* − *deriv f x*) ≤ *e* **for** *w*
by (*blast intro*: *cs_less less_k1 k1* [*unfolded divide_const_simps dist_norm*] *less_imp_le le_less_trans*)
have *f_has_der*: $\bigwedge x.\ x \in U \Longrightarrow$ (*f has_field_derivative deriv f x*) (*at x within U*)
by (*metis DERIV_deriv_iff_field_differentiable at_within_open holf holomorphic_on_def* ‹*open U*›)
have *closed_segment* x' $z' \subseteq U$
by (*rule order_trans* [*OF _ k2*]) (*simp add*: *cs_less le_less_trans* [*OF _ less_k2*] *dist_complex_def norm_minus_commute subset_iff*)
then have *cint_derf*: (*deriv f has_contour_integral f z'* − *f x'*) (*linepath x' z'*)
using *contour_integral_primitive* [*OF f_has_der valid_path_linepath*] *pasz* **by** *simp*
then have ∗: ((λ*x*. *deriv f x* / (*z'* − *x'*)) *has_contour_integral* (*f z'* − *f x'*) / (*z'* − *x'*)) (*linepath x' z'*)
by (*rule has_contour_integral_div*)
have *norm* ((*f z'* − *f x'*) / (*z'* − *x'*) − *deriv f x*) ≤ *e*/*norm*(*z'* − *x'*) ∗ *norm*(*z'* − *x'*)
apply (*rule has_contour_integral_bound_linepath* [*OF has_contour_integral_diff* [*OF* ∗]])
using *has_contour_integral_div* [**where** *c* = *z'* − *x'*, *OF has_contour_integral_const_linepath* [*of deriv f x z' x'*]]
‹*e* > 0› ‹*z'* ≠ *x'*›
apply (*auto simp*: *norm_divide divide_simps derf_le*)
done
also have ... ≤ *e* **using** ‹0 < *e*› **by** *simp*
finally show *?thesis* **.**
qed
show ∃ *d*>0. ∀ *xa*∈*U* × *U*.
0 < *dist xa* (*x*, *x*) ∧ *dist xa* (*x*, *x*) < *d* ⟶
dist (*case xa of* (*x*, *y*) ⇒ *if y* = *x then deriv f x else* (*f y* − *f x*) / (*y* − *x*)) (*deriv f x*) ≤ *e*
apply (*rule_tac x*=*min k1 k2* **in** *exI*)
using ‹*k1*>0› ‹*k2*>0› ‹*e*>0›
by (*force simp*: *dist_norm neq intro*: *dual_order.strict_trans2 k1 less_imp_le norm_fst_le*)
qed
have *con_pa_f*: *continuous_on* (*path_image* γ) *f*
by (*meson holf holomorphic_on_imp_continuous_on holomorphic_on_subset interior_subset subt T*)
have *le_B*: $\bigwedge T.\ T \in \{0..1\} \Longrightarrow$ *cmod* (*vector_derivative* γ (*at T*)) ≤ *B*
using γ' *B* **by** (*simp add*: *path_image_def vector_derivative_at rev_image_eqI*)
have *f_has_cint*: $\bigwedge w.\ w \in v - path\_image\ \gamma \Longrightarrow$ ((λ*u*. *f u* / (*u* − *w*) ^ 1) *has_contour_integral h w*) γ
by (*simp add*: *V*)

have *cond_uu*: *continuous_on* $(U \times U)$ $(\lambda(x,y).\ d\ x\ y)$
apply (*simp add*: *continuous_on_eq_continuous_within d_def continuous_within tendsto_f'*)
apply (*simp add*: *tendsto_within_open_NO_MATCH open_Times* ‹*open U*›, *clarify*)
apply (*rule Lim_transform_within_open* [*OF* _ *open_uu_Id*, **where** $f = (\lambda(x,y).\ (f\ y - f\ x)\ /\ (y - x))$])
using *con_ff*
apply (*auto simp*: *continuous_within*)
done
have *hol_dw*: $(\lambda z.\ d\ z\ w)$ *holomorphic_on* U **if** $w \in U$ **for** w
proof −
have *continuous_on* U $((\lambda(x,y).\ d\ x\ y) \circ (\lambda z.\ (w,z)))$
by (*rule continuous_on_compose continuous_intros continuous_on_subset* [*OF cond_uu*] | *force intro*: *that*)+
then have ∗: *continuous_on* U $(\lambda z.\ \textit{if}\ w = z\ \textit{then deriv}\ f\ z\ \textit{else}\ (f\ w - f\ z)\ /\ (w - z))$
by (*rule rev_iffD1* [*OF* _ *continuous_on_cong* [*OF refl*]]) (*simp add*: *d_def field_simps*)
have ∗∗: $(\lambda z.\ \textit{if}\ w = z\ \textit{then deriv}\ f\ z\ \textit{else}\ (f\ w - f\ z)\ /\ (w - z))$ *field_differentiable at* x
if $x \in U\ x \neq w$ **for** x
proof (*rule_tac* $f = \lambda x.\ (f\ w - f\ x)/(w - x)$ **and** $d = \textit{dist}\ x\ w$ **in** *field_differentiable_transform_within*)
show $(\lambda x.\ (f\ w - f\ x)\ /\ (w - x))$ *field_differentiable at* x
using *that* ‹*open U*›
by (*intro derivative_intros holomorphic_on_imp_differentiable_at* [*OF holf*]; *force*)
qed (*use that* ‹*open U*› **in** ‹*auto simp*: *dist_commute*›)
show *?thesis*
unfolding *d_def*
proof (*rule no_isolated_singularity* [*OF* ∗ _ ‹*open U*›])
show $(\lambda z.\ \textit{if}\ w = z\ \textit{then deriv}\ f\ z\ \textit{else}\ (f\ w - f\ z)\ /\ (w - z))$ *holomorphic_on* $U - \{w\}$
by (*auto simp*: *field_differentiable_def* [*symmetric*] *holomorphic_on_open open_Diff* ‹*open U*› ∗∗)
qed *auto*
qed
{ fix $a\ b$
assume *abu*: *closed_segment* $a\ b \subseteq U$
have *cont_cint_d*: *continuous_on* U $(\lambda w.\ \textit{contour\_integral}\ (\textit{linepath}\ a\ b)\ (\lambda z.\ d\ z\ w))$
proof (*rule contour_integral_continuous_on_linepath_2D* [*OF* ‹*open U*› _ _ *abu*])
show $\bigwedge w.\ w \in U \Longrightarrow (\lambda z.\ d\ z\ w)$ *contour_integrable_on* $(\textit{linepath}\ a\ b)$
by (*metis abu hol_dw continuous_on_subset contour_integrable_continuous_linepath holomorphic_on_imp_continuous_on*)
show *continuous_on* $(U \times U)$ $(\lambda(x,\ y).\ d\ y\ x)$
by (*auto intro*: *continuous_on_swap_args cond_uu*)

```
    qed
    have cont_cint_dγ: continuous_on {0..1} ((λw. contour_integral (linepath
a b) (λz. d z w)) ∘ γ)
    proof (rule continuous_on_compose)
      show continuous_on {0..1} γ
        using ‹path γ› path_def by blast
      show continuous_on (γ ` {0..1}) (λw. contour_integral (linepath a b) (λz.
d z w))
        using pasz unfolding path_image_def
        by (auto intro!: continuous_on_subset [OF cont_cint_d])
    qed
    have continuous_on {0..1} (λx. vector_derivative γ (at x))
    using pfγ' by (simp add: continuous_on_polymonial_function vector_derivative_at
[OF γ'])
    then      have cint_cint: (λw. contour_integral (linepath a b) (λz. d z w))
contour_integrable_on γ
      apply (simp add: contour_integrable_on)
      apply (rule integrable_continuous_real)
      by (rule continuous_on_mult [OF cont_cint_dγ [unfolded o_def]])
    have contour_integral (linepath a b) h = contour_integral (linepath a b) (λz.
contour_integral γ (d z))
      using abu  by (force simp: h_def intro: contour_integral_eq)
    also have ... =  contour_integral γ (λw. contour_integral (linepath a b) (λz.
d z w))
    proof (rule contour_integral_swap)
      show continuous_on (path_image (linepath a b) × path_image γ) (λ(y1,
y2). d y1 y2)
        using abu pasz by (auto intro: continuous_on_subset [OF cond_uu])
      show continuous_on {0..1} (λt. vector_derivative (linepath a b) (at t))
        by (auto intro!: continuous_intros)
      show continuous_on {0..1} (λt. vector_derivative γ (at t))
        by (metis γ' continuous_on_eq path_def path_polynomial_function pfγ'
vector_derivative_at)
    qed (use ‹valid_path γ› in auto)
    finally have cint_h_eq:
      contour_integral (linepath a b) h =
            contour_integral γ (λw. contour_integral (linepath a b) (λz. d z
w)) .
    note cint_cint cint_h_eq
  } note cint_h = this
  have conthu: continuous_on U h
  proof (simp add: continuous_on_sequentially, clarify)
    fix a x
    assume x: x ∈ U and au: ∀ n. a n ∈ U and ax: a ⟶ x
    then have A1: ∀F n in sequentially. d (a n) contour_integrable_on γ
      by (meson U contour_integrable_on_def eventuallyI)
    obtain dd where dd>0 and dd: cball x dd ⊆ U using open_contains_cball
‹open U› x by force
    have A2: uniform_limit (path_image γ) (λn. d (a n)) (d x) sequentially
```

```
      unfolding uniform_limit_iff dist_norm
    proof clarify
      fix ee::real
      assume 0 < ee
      show ∀F n in sequentially. ∀ξ∈path_image γ. cmod (d (a n) ξ − d x ξ) <
ee
      proof −
        let ?ddpa = {(w,z) |w z. w ∈ cball x dd ∧ z ∈ path_image γ}
        have uniformly_continuous_on ?ddpa (λ(x,y). d x y)
      proof (rule compact_uniformly_continuous [OF continuous_on_subset[OF
cond_uu]])
          show compact {(w, z) |w z. w ∈ cball x dd ∧ z ∈ path_image γ}
            using ‹valid_path γ›
            by (auto simp: compact_Times compact_valid_path_image simp del:
mem_cball)
        qed (use dd pasz in auto)
        then obtain kk where kk>0
          and kk: ⋀x x'. ⟦x ∈ ?ddpa; x' ∈ ?ddpa; dist x' x < kk⟧ ⟹
                        dist ((λ(x,y). d x y) x') ((λ(x,y). d x y) x) < ee
          by (rule uniformly_continuous_onE [where e = ee]) (use ‹0 < ee› in
auto)
        have kk: ⟦norm (w − x) ≤ dd; z ∈ path_image γ; norm ((w, z) − (x, z))
< kk⟧ ⟹ norm (d w z − d x z) < ee
          for w z
          using ‹dd>0› kk [of (x,z) (w,z)] by (force simp: norm_minus_commute
dist_norm)
        obtain no where ∀n≥no. dist (a n) x < min dd kk
          using ax unfolding lim_sequentially
          by (meson ‹0 < dd› ‹0 < kk› min_less_iff_conj)
        then show ?thesis
          using ‹dd > 0› ‹kk > 0› by (fastforce simp: eventually_sequentially kk
dist_norm)
      qed
    qed
    have (λn. contour_integral γ (d (a n))) ⟶ contour_integral γ (d x)
      by (rule contour_integral_uniform_limit [OF A1 A2 le_B]) (auto simp:
‹valid_path γ›)
    then have tendsto_hx: (λn. contour_integral γ (d (a n))) ⟶ h x
      by (simp add: h_def x)
    then show (h ∘ a) ⟶ h x
      by (simp add: h_def x au o_def)
  qed
  show ?thesis
  proof (simp add: holomorphic_on_open field_differentiable_def [symmetric],
clarify)
    fix z0
    consider z0 ∈ v | z0 ∈ U using uv_Un by blast
    then show h field_differentiable at z0
    proof cases
```

```
      assume z0 ∈ v then show ?thesis
        using Cauchy_next_derivative [OF con_pa_f le_B f_has_cint _ ov] V
f_has_cint ‹valid_path γ›
        by (auto simp: field_differentiable_def v_def)
    next
      assume z0 ∈ U then
       obtain e where e>0 and e: ball z0 e ⊆ U by (blast intro: openE [OF
‹open U›])
      have ∗: contour_integral (linepath a b) h + contour_integral (linepath b c)
h + contour_integral (linepath c a) h = 0
            if abc_subset: convex hull {a, b, c} ⊆ ball z0 e  for a b c
      proof −
       have ∗: ⋀x1 x2 z. z ∈ U ⟹ closed_segment x1 x2 ⊆ U ⟹ (λw. d w z)
contour_integrable_on linepath x1 x2
         using  hol_dw holomorphic_on_imp_continuous_on ‹open U›
         by (auto intro!: contour_integrable_holomorphic_simple)
       have abc: closed_segment a b ⊆ U  closed_segment b c ⊆ U  closed_segment
c a ⊆ U
         using that e segments_subset_convex_hull by fastforce+
        have eq0: ⋀w. w ∈ U ⟹ contour_integral (linepath a b +++ linepath b
c +++ linepath c a) (λz. d z w) = 0
        proof (rule contour_integral_unique [OF Cauchy_theorem_triangle])
         show ⋀w. w ∈ U ⟹ (λz. d z w) holomorphic_on convex hull {a, b, c}
              using e abc_subset by (auto intro: holomorphic_on_subset [OF
hol_dw])
        qed
        have contour_integral γ
               (λx. contour_integral (linepath a b) (λz. d z x) +
                   (contour_integral (linepath b c) (λz. d z x) +
                    contour_integral (linepath c a) (λz. d z x))) = 0
         apply (rule contour_integral_eq_0)
         using abc pasz U
         apply (subst contour_integral_join [symmetric], auto intro: eq0 ∗)+
         done
        then show ?thesis
         by (simp add: cint_h abc contour_integrable_add contour_integral_add
[symmetric] add_ac)
      qed
      show ?thesis
       using e ‹e > 0›
       by (auto intro!: holomorphic_on_imp_differentiable_at [OF _ open_ball]
analytic_imp_holomorphic
                    Morera_triangle continuous_on_subset [OF conthu] ∗)
    qed
  qed
 qed
 ultimately have [simp]: h z = 0 for z
  by (meson Liouville_weak)
 have ((λw. 1 / (w − z)) has_contour_integral complex_of_real (2 ∗ pi) ∗ i ∗
```

winding_number γ *z*) γ
by (*rule has_contour_integral_winding_number* [*OF* ‹*valid_path* γ› *znot*])
then have ((λ*w*. *f z* ∗ (*1* / (*w* − *z*))) *has_contour_integral complex_of_real* (*2* ∗ *pi*) ∗ i ∗ *winding_number* γ *z* ∗ *f z*) γ
by (*metis mult.commute has_contour_integral_lmul*)
then have *1*: ((λ*w*. *f z* / (*w* − *z*)) *has_contour_integral complex_of_real* (*2* ∗ *pi*) ∗ i ∗ *winding_number* γ *z* ∗ *f z*) γ
by (*simp add*: *field_split_simps*)
moreover have *2*: ((λ*w*. (*f w* − *f z*) / (*w* − *z*)) *has_contour_integral 0*) γ
using *U* [*OF z*] *pasz d_def* **by** (*force elim*: *has_contour_integral_eq* [**where** *g* = λ*w*. (*f w* − *f z*)/(*w* − *z*)])
show *?thesis*
using *has_contour_integral_add* [*OF 1 2*] **by** (*simp add*: *diff_divide_distrib*)
qed

theorem *Cauchy_integral_formula_global*:
assumes *S*: *open S* **and** *holf*: *f holomorphic_on S*
and *z*: *z* ∈ *S* **and** *vpg*: *valid_path* γ
and *pasz*: *path_image* γ ⊆ *S* − {*z*} **and** *loop*: *pathfinish* γ = *pathstart* γ
and *zero*: ⋀*w*. *w* ∉ *S* ⟹ *winding_number* γ *w* = *0*
shows ((λ*w*. *f w* / (*w* − *z*)) *has_contour_integral* (*2*∗*pi* ∗ i ∗ *winding_number* γ *z* ∗ *f z*)) γ
proof −
have *path* γ **using** *vpg* **by** (*blast intro*: *valid_path_imp_path*)
have *hols*: (λ*w*. *f w* / (*w* − *z*)) *holomorphic_on S* − {*z*} (λ*w*. *1* / (*w* − *z*)) *holomorphic_on S* − {*z*}
by (*rule holomorphic_intros holomorphic_on_subset* [*OF holf*] | *force*)+
then have *cint_fw*: (λ*w*. *f w* / (*w* − *z*)) *contour_integrable_on* γ
by (*meson contour_integrable_holomorphic_simple holomorphic_on_imp_continuous_on open_delete S vpg pasz*)
obtain *d* **where** *d*>*0*
and *d*: ⋀*g h*. ⟦*valid_path g*; *valid_path h*; ∀ *t*∈{*0..1*}. *cmod* (*g t* − γ *t*) < *d* ∧ *cmod* (*h t* − γ *t*) < *d*;
pathstart h = *pathstart g* ∧ *pathfinish h* = *pathfinish g*⟧
⟹ *path_image h* ⊆ *S* − {*z*} ∧ (∀ *f*. *f holomorphic_on S* − {*z*} ⟶ *contour_integral h f* = *contour_integral g f*)
using *contour_integral_nearby_ends* [*OF* _ ‹*path* γ› *pasz*] *S* **by** (*simp add*: *open_Diff*) *metis*
obtain *p* **where** *polyp*: *polynomial_function p*
and *ps*: *pathstart p* = *pathstart* γ **and** *pf*: *pathfinish p* = *pathfinish* γ **and** *led*: ∀ *t*∈{*0..1*}. *cmod* (*p t* − γ *t*) < *d*
using *path_approx_polynomial_function* [*OF* ‹*path* γ› ‹*d* > *0*›] **by** *metis*
then have *ploop*: *pathfinish p* = *pathstart p* **using** *loop* **by** *auto*
have *vpp*: *valid_path p* **using** *polyp valid_path_polynomial_function* **by** *blast*
have [*simp*]: *z* ∉ *path_image* γ **using** *pasz* **by** *blast*
have *paps*: *path_image p* ⊆ *S* − {*z*} **and** *cint_eq*: (⋀*f*. *f holomorphic_on S* − {*z*} ⟹ *contour_integral p f* = *contour_integral* γ *f*)
using *pf ps led d* [*OF vpg vpp*] ‹*d* > *0*› **by** *auto*
have *wn_eq*: *winding_number p z* = *winding_number* γ *z*

```
    using vpp paps
    by (simp add: subset_Diff_insert vpg valid_path_polynomial_function wind-
ing_number_valid_path cint_eq hols)
  have winding_number p w = winding_number γ w if w ∉ S for w
  proof −
    have hol: (λv. 1 / (v − w)) holomorphic_on S − {z}
      using that by (force intro: holomorphic_intros holomorphic_on_subset [OF
holf])
    have w ∉ path_image p w ∉ path_image γ using paps pasz that by auto
    then show ?thesis
    using vpp vpg by (simp add: subset_Diff_insert valid_path_polynomial_function
winding_number_valid_path cint_eq [OF hol])
  qed
  then have wn0: ⋀w. w ∉ S ⟹ winding_number p w = 0
    by (simp add: zero)
  show ?thesis
     using Cauchy_integral_formula_global_weak [OF S holf z polyp paps ploop
wn0] hols
    by (metis wn_eq cint_eq has_contour_integral_eqpath cint_fw cint_eq)
qed

theorem Cauchy_theorem_global:
    assumes S: open S and holf: f holomorphic_on S
        and vpg: valid_path γ and loop: pathfinish γ = pathstart γ
        and pas: path_image γ ⊆ S
        and zero: ⋀w. w ∉ S ⟹ winding_number γ w = 0
      shows (f has_contour_integral 0) γ
proof −
  obtain z where z ∈ S and znot: z ∉ path_image γ
  proof −
    have compact (path_image γ)
      using compact_valid_path_image vpg by blast
    then have path_image γ ≠ S
      by (metis (no_types) compact_open path_image_nonempty S)
    with pas show ?thesis by (blast intro: that)
  qed
  then have pasz: path_image γ ⊆ S − {z} using pas by blast
  have hol: (λw. (w − z) ∗ f w) holomorphic_on S
    by (rule holomorphic_intros holf)+
  show ?thesis
    using Cauchy_integral_formula_global [OF S hol ‹z ∈ S› vpg pasz loop zero]
    by (auto simp: znot elim!: has_contour_integral_eq)
qed

corollary Cauchy_theorem_global_outside:
    assumes open S f holomorphic_on S valid_path γ  pathfinish γ = pathstart γ
path_image γ ⊆ S
          ⋀w. w ∉ S ⟹ w ∈ outside(path_image γ)
      shows (f has_contour_integral 0) γ
```

by (*metis Cauchy_theorem_global assms winding_number_zero_in_outside valid_path_imp_path*)

lemma *simply_connected_imp_winding_number_zero*:
assumes *simply_connected S path g*
path_image g ⊆ S pathfinish g = pathstart g z ∉ S
shows *winding_number g z = 0*
proof −
have *hom*: *homotopic_loops S g* (*linepath* (*pathstart g*) (*pathstart g*))
by (*meson assms homotopic_paths_imp_homotopic_loops pathfinish_linepath simply_connected_eq_contractible_path*)
then have *homotopic_paths* (− {*z*}) *g* (*linepath* (*pathstart g*) (*pathstart g*))
by (*meson ‹z ∉ S› homotopic_loops_imp_homotopic_paths_null homotopic_paths_subset subset_Compl_singleton*)
then have *winding_number g z = winding_number*(*linepath* (*pathstart g*) (*pathstart g*)) *z*
by (*rule winding_number_homotopic_paths*)
also have *... = 0*
using *assms* **by** (*force intro*: *winding_number_trivial*)
finally show *?thesis* **.**
qed

lemma *Cauchy_theorem_simply_connected*:
assumes *open S simply_connected S f holomorphic_on S valid_path g*
path_image g ⊆ S pathfinish g = pathstart g
shows (*f has_contour_integral 0*) *g*
using *assms*
apply (*simp add*: *simply_connected_eq_contractible_path*)
apply (*auto intro*!: *Cauchy_theorem_null_homotopic* [**where** *a = pathstart g*]
homotopic_paths_imp_homotopic_loops)
using *valid_path_imp_path* **by** *blast*

proposition *holomorphic_logarithm_exists*:
assumes *A*: *convex A open A*
and *f*: *f holomorphic_on A* ⋀*x*. *x ∈ A* ⟹ *f x ≠ 0*
and *z0*: *z0 ∈ A*
obtains *g* **where** *g holomorphic_on A* **and** ⋀*x*. *x ∈ A* ⟹ *exp* (*g x*) = *f x*
proof −
note *f′ = holomorphic_derivI* [*OF f*(*1*) *A*(*2*)]
obtain *g* **where** *g*: ⋀*x*. *x ∈ A* ⟹ (*g has_field_derivative deriv f x / f x*) (*at x*)
proof (*rule holomorphic_convex_primitive′* [*OF A*])
show (*λx*. *deriv f x / f x*) *holomorphic_on A*
by (*intro holomorphic_intros f A*)
qed (*auto simp*: *A at_within_open*[*of _ A*])
define *h* **where** *h* = (*λx*. −*g z0* + *ln* (*f z0*) + *g x*)
from *g* **and** *A* **have** *g_holo*: *g holomorphic_on A*
by (*auto simp*: *holomorphic_on_def at_within_open*[*of _ A*] *field_differentiable_def*)
hence *h_holo*: *h holomorphic_on A*
by (*auto simp*: *h_def intro*!: *holomorphic_intros*)
have ∃ *c*. ∀ *x*∈*A*. *f x / exp* (*h x*) − *1 = c*

```
  proof (rule has_field_derivative_zero_constant, goal_cases)
    case (2 x)
    note [simp] = at_within_open[OF _ ‹open A›]
    from 2 and z0 and f show ?case
      by (auto simp: h_def exp_diff field_simps intro!: derivative_eq_intros g f')
  qed fact+
  then obtain c where c: ⋀x. x ∈ A ⟹ f x / exp (h x) - 1 = c
    by blast
  from c[OF z0] and z0 and f have c = 0
    by (simp add: h_def)
  with c have ⋀x. x ∈ A ⟹ exp (h x) = f x by simp
  from that[OF h_holo this] show ?thesis .
qed
```

4.14 Cauchy's inequality and more versions of Liouville

```
lemma Cauchy_higher_deriv_bound:
    assumes holf: f holomorphic_on (ball z r)
        and contf: continuous_on (cball z r) f
        and fin : ⋀w. w ∈ ball z r ⟹ f w ∈ ball y B0
        and 0 < r and 0 < n
      shows norm ((deriv ^^ n) f z) ≤ (fact n) * B0 / r^n
proof -
  have 0 < B0 using ‹0 < r› fin [of z]
    by (metis ball_eq_empty ex_in_conv fin not_less)
  have le_B0: cmod (f w - y) ≤ B0 if cmod (w - z) ≤ r for w
  proof (rule continuous_on_closure_norm_le [of ball z r λw. f w - y], use ‹0
< r› in simp_all)
    show continuous_on (cball z r) (λw. f w - y)
      by (intro continuous_intros contf)
    show dist z w ≤ r
      by (simp add: dist_commute dist_norm that)
  qed (use fin in ‹auto simp: dist_norm less_eq_real_def norm_minus_commute›)
  have (deriv ^^ n) f z = (deriv ^^ n) (λw. f w) z - (deriv ^^ n) (λw. y) z
    using ‹0 < n› by simp
  also have ... = (deriv ^^ n) (λw. f w - y) z
    by (rule higher_deriv_diff [OF holf, symmetric]) (auto simp: ‹0 < r›)
  finally have (deriv ^^ n) f z = (deriv ^^ n) (λw. f w - y) z .
  have contf': continuous_on (cball z r) (λu. f u - y)
    by (rule contf continuous_intros)+
  have holf': (λu. (f u - y)) holomorphic_on (ball z r)
    by (simp add: holf holomorphic_on_diff)
  define a where a = (2 * pi)/(fact n)
  have 0 < a  by (simp add: a_def)
  have B0/r^(Suc n)*2 * pi * r = a*((fact n)*B0/r^n)
    using ‹0 < r› by (simp add: a_def field_split_simps)
  have der_dif: (deriv ^^ n) (λw. f w - y) z = (deriv ^^ n) f z
    using ‹0 < r› ‹0 < n›
    by (auto simp: higher_deriv_diff [OF holf holomorphic_on_const])
```

```
  have norm ((2 * of_real pi * i)/(fact n) * (deriv ^^ n) (λw. f w - y) z)
        ≤ (B0/r^(Suc n)) * (2 * pi * r)
    apply (rule has_contour_integral_bound_circlepath [of (λu. (f u - y)/(u - z)^(Suc n)) _ z])
    using Cauchy_has_contour_integral_higher_derivative_circlepath [OF contf' holf']
    using ‹0 < B0› ‹0 < r›
   apply (auto simp: norm_divide norm_mult norm_power divide_simps le_B0)
    done
  then show ?thesis
    using ‹0 < r›
     by (auto simp: norm_divide norm_mult norm_power field_simps der_dif le_B0)
qed

lemma Cauchy_inequality:
    assumes holf: f holomorphic_on (ball ξ r)
       and contf: continuous_on (cball ξ r) f
       and 0 < r
       and nof: ⋀x. norm(ξ-x) = r ⟹ norm(f x) ≤ B
     shows norm ((deriv ^^ n) f ξ) ≤ (fact n) * B / r^n
proof -
  obtain x where norm (ξ-x) = r
    by (metis abs_of_nonneg add_diff_cancel_left' ‹0 < r› diff_add_cancel
              dual_order.strict_implies_order norm_of_real)
  then have 0 ≤ B
    by (metis nof norm_not_less_zero not_le order_trans)
  have ξ ∈ ball ξ r
    using ‹0 < r› by simp
  then have ((λu. f u / (u-ξ) ^ Suc n) has_contour_integral (2 * pi) * i / fact n * (deriv ^^ n) f ξ)
        (circlepath ξ r)
   by (rule Cauchy_has_contour_integral_higher_derivative_circlepath [OF contf holf])
  have norm ((2 * pi * i)/(fact n) * (deriv ^^ n) f ξ) ≤ (B / r^(Suc n)) * (2 * pi * r)
  proof (rule has_contour_integral_bound_circlepath)
    have ξ ∈ ball ξ r
      using ‹0 < r› by simp
    then show ((λu. f u / (u-ξ) ^ Suc n) has_contour_integral (2 * pi) * i / fact n * (deriv ^^ n) f ξ)
         (circlepath ξ r)
       by (rule Cauchy_has_contour_integral_higher_derivative_circlepath [OF contf holf])
    show ⋀x. cmod (x-ξ) = r ⟹ cmod (f x / (x-ξ) ^ Suc n) ≤ B / r ^ Suc n
      using ‹0 ≤ B› ‹0 < r›
     by (simp add: norm_divide norm_power nof frac_le norm_minus_commute del: power_Suc)
  qed (use ‹0 ≤ B› ‹0 < r› in auto)
```

```
  then show ?thesis using ‹0 < r›
    by (simp add: norm_divide norm_mult field_simps)
qed

lemma Liouville_polynomial:
    assumes holf: f holomorphic_on UNIV
        and nof: ⋀z. A ≤ norm z ⟹ norm(f z) ≤ B * norm z ^ n
      shows f ξ = (∑ k≤n. (deriv^^k) f 0 / fact k * ξ ^ k)
proof (cases rule: le_less_linear [THEN disjE])
  assume B ≤ 0
  then have ⋀z. A ≤ norm z ⟹ norm(f z) = 0
     by (metis nof less_le_trans zero_less_mult_iff neqE norm_not_less_zero
norm_power not_le)
  then have f0: (f ⟶ 0) at_infinity
    using Lim_at_infinity by force
  then have [simp]: f = (λw. 0)
    using Liouville_weak [OF holf, of 0]
    by (simp add: eventually_at_infinity f0) meson
  show ?thesis by simp
next
  assume 0 < B
  have ((λk. (deriv ^^ k) f 0 / (fact k) * (ξ - 0)^k) sums f ξ)
  proof (rule holomorphic_power_series [where r = norm ξ + 1])
    show f holomorphic_on ball 0 (cmod ξ + 1) ξ ∈ ball 0 (cmod ξ + 1)
      using holf holomorphic_on_subset by auto
  qed
  then have sumsf: ((λk. (deriv ^^ k) f 0 / (fact k) * ξ^k) sums f ξ) by simp
  have (deriv ^^ k) f 0 / fact k * ξ ^ k = 0 if k>n for k
  proof (cases (deriv ^^ k) f 0 = 0)
    case True then show ?thesis by simp
  next
    case False
    define w where w = complex_of_real (fact k * B / cmod ((deriv ^^ k) f 0)
+ (|A| + 1))
    have 1 ≤ abs (fact k * B / cmod ((deriv ^^ k) f 0) + (|A| + 1))
      using ‹0 < B› by simp
    then have wge1: 1 ≤ norm w
      by (metis norm_of_real w_def)
    then have w ≠ 0 by auto
    have kB: 0 < fact k * B
      using ‹0 < B› by simp
    then have 0 ≤ fact k * B / cmod ((deriv ^^ k) f 0)
      by simp
    then have wgeA: A ≤ cmod w
      by (simp only: w_def norm_of_real)
   have fact k * B / cmod ((deriv ^^ k) f 0) < abs (fact k * B / cmod ((deriv ^^
k) f 0) + (|A| + 1))
      using ‹0 < B› by simp
    then have wge: fact k * B / cmod ((deriv ^^ k) f 0) < norm w
```

```
    by (metis norm_of_real w_def)
  then have fact k * B / norm w < cmod ((deriv ^^ k) f 0)
   using False by (simp add: field_split_simps mult.commute split: if_split_asm)
  also have ... ≤ fact k * (B * norm w ^ n) / norm w ^ k
  proof (rule Cauchy_inequality)
    show f holomorphic_on ball 0 (cmod w)
      using holf holomorphic_on_subset by force
    show continuous_on (cball 0 (cmod w)) f
      using holf holomorphic_on_imp_continuous_on holomorphic_on_subset
by blast
    show ⋀x. cmod (0 - x) = cmod w ⟹ cmod (f x) ≤ B * cmod w ^ n
      by (metis nof wgeA dist_0_norm dist_norm)
  qed (use ‹w ≠ 0› in auto)
  also have ... = fact k * (B * 1 / cmod w ^ (k-n))
  proof -
    have cmod w ^ n / cmod w ^ k = 1 / cmod w ^ (k - n)
      using ‹k>n› ‹w ≠ 0› ‹0 < B› by (simp add: field_split_simps semir-
ing_normalization_rules)
    then show ?thesis
      by (metis times_divide_eq_right)
  qed
  also have ... = fact k * B / cmod w ^ (k-n)
    by simp
  finally have fact k * B / cmod w < fact k * B / cmod w ^ (k - n) .
  then have 1 / cmod w < 1 / cmod w ^ (k - n)
    by (metis kB divide_inverse inverse_eq_divide mult_less_cancel_left_pos)
  then have cmod w ^ (k - n) < cmod w
   by (metis frac_le le_less_trans norm_ge_zero norm_one not_less order_refl
wge1 zero_less_one)
  with self_le_power [OF wge1] have False
    by (meson diff_is_0_eq not_gr0 not_le that)
  then show ?thesis by blast
 qed
 then have (deriv ^^ (k + Suc n)) f 0 / fact (k + Suc n) * ξ ^ (k + Suc n) =
0 for k
  using not_less_eq by blast
 then have (λi. (deriv ^^ (i + Suc n)) f 0 / fact (i + Suc n) * ξ ^ (i + Suc n))
sums 0
  by (rule sums_0)
 with sums_split_initial_segment [OF sumsf, where n = Suc n]
 show ?thesis
  using atLeast0AtMost lessThan_Suc_atMost sums_unique2 by fastforce
qed
```

Every bounded entire function is a constant function.

```
theorem Liouville_theorem:
  assumes holf: f holomorphic_on UNIV
    and bf: bounded (range f)
  shows f constant_on UNIV
```

```
proof −
  obtain B where ⋀z. cmod (f z) ≤ B
    by (meson bf bounded_pos rangeI)
  then show ?thesis
    using Liouville_polynomial [OF holf, of 0 B 0, simplified]
    by (meson constant_on_def)
qed
```

A holomorphic function f has only isolated zeros unless f is 0.

```
lemma powser_0_nonzero:
  fixes a :: nat ⇒ 'a::{real_normed_field,banach}
  assumes r: 0 < r
      and sm: ⋀x. norm (x−ξ) < r ⟹ (λn. a n * (x−ξ) ^ n) sums (f x)
      and [simp]: f ξ = 0
      and m0: a m ≠ 0 and m>0
  obtains s where 0 < s and ⋀z. z ∈ cball ξ s − {ξ} ⟹ f z ≠ 0
proof −
  have r ≤ conv_radius a
    using sm sums_summable by (auto simp: le_conv_radius_iff [where ξ=ξ])
  obtain m where am: a m ≠ 0 and az [simp]: (⋀n. n<m ⟹ a n = 0)
  proof
    show a (LEAST n. a n ≠ 0) ≠ 0
      by (metis (mono_tags, lifting) m0 LeastI)
  qed (fastforce dest!: not_less_Least)
  define b where b i = a (i+m) / a m for i
  define g where g x = suminf (λi. b i * (x−ξ) ^ i) for x
  have [simp]: b 0 = 1
    by (simp add: am b_def)
  { fix x::'a
    assume norm (x−ξ) < r
    then have (λn. (a m * (x−ξ)^m) * (b n * (x−ξ)^n)) sums (f x)
      using am az sm sums_zero_iff_shift [of m (λn. a n * (x−ξ) ^ n) f x]
      by (simp add: b_def monoid_mult_class.power_add algebra_simps)
    then have x ≠ ξ ⟹ (λn. b n * (x−ξ)^n) sums (f x / (a m * (x−ξ)^m))
      using am by (simp add: sums_mult_D)
  } note bsums = this
  then have  norm (x−ξ) < r ⟹ summable (λn. b n * (x−ξ)^n) for x
    using sums_summable by (cases x=ξ) auto
  then have r ≤ conv_radius b
    by (simp add: le_conv_radius_iff [where ξ=ξ])
  then have r/2 < conv_radius b
    using not_le order_trans r by fastforce
  then have continuous_on (cball ξ (r/2)) g
    using powser_continuous_suminf [of r/2 b ξ] by (simp add: g_def)
  then obtain s where s>0 ⋀x. ⟦norm (x−ξ) ≤ s; norm (x−ξ) ≤ r/2⟧ ⟹ dist
(g x) (g ξ) < 1/2
  proof (rule continuous_onE)
    show ξ ∈ cball ξ (r / 2) 1/2 > (0::real)
      using r by auto
```

```
  qed (auto simp: dist_commute dist_norm)
  moreover have g ξ = 1
    by (simp add: g_def)
  ultimately have gnz: ⋀x. ⟦norm (x−ξ) ≤ s; norm (x−ξ) ≤ r/2⟧ ⟹ (g x) ≠ 0
    by fastforce
  have f x ≠ 0 if x ≠ ξ norm (x−ξ) ≤ s norm (x−ξ) ≤ r/2 for x
    using bsums [of x] that gnz [of x] r sums_iff unfolding g_def by fastforce
  then show ?thesis
    apply (rule_tac s=min s (r/2) in that)
    using ‹0 < r› ‹0 < s› by (auto simp: dist_commute dist_norm)
qed
```

4.15 Complex functions and power series

The following defines the power series expansion of a complex function at a given point (assuming that it is analytic at that point).

```
definition fps_expansion :: (complex ⇒ complex) ⇒ complex ⇒ complex fps
where
  fps_expansion f z0 = Abs_fps (λn. (deriv ^^ n) f z0 / fact n)

lemma
  fixes r :: ereal
  assumes f holomorphic_on eball z0 r
  shows   conv_radius_fps_expansion: fps_conv_radius (fps_expansion f z0) ≥ r
    and   eval_fps_expansion: ⋀z. z ∈ eball z0 r ⟹ eval_fps (fps_expansion f
z0) (z − z0) = f z
    and   eval_fps_expansion': ⋀z. norm z < r ⟹ eval_fps (fps_expansion f z0)
z = f (z0 + z)
proof −
  have (λn. fps_nth (fps_expansion f z0) n ∗ (z − z0) ^ n) sums f z
    if z ∈ ball z0 r' ereal r' < r for z r'
  proof −
    from that(2) have ereal r' ≤ r by simp
    from assms(1) and this have f holomorphic_on ball z0 r'
      by (rule holomorphic_on_subset[OF _ ball_eball_mono])
    from holomorphic_power_series [OF this that(1)]
      show ?thesis by (simp add: fps_expansion_def)
  qed
  hence ∗: (λn. fps_nth (fps_expansion f z0) n ∗ (z − z0) ^ n) sums f z
    if z ∈ eball z0 r for z
    using that by (subst (asm) eball_conv_UNION_balls) blast
 show fps_conv_radius (fps_expansion f z0) ≥ r unfolding fps_conv_radius_def
  proof (rule conv_radius_geI_ex)
    fix r' :: real assume r': r' > 0 ereal r' < r
    thus ∃z. norm z = r' ∧ summable (λn. fps_nth (fps_expansion f z0) n ∗ z ^
n)
      using ∗[of z0 + of_real r']
      by (intro exI[of _ of_real r']) (auto simp: summable_def dist_norm)
  qed
```

```
  show eval_fps (fps_expansion f z0) (z − z0) = f z if z ∈ eball z0 r for z
    using ∗[OF that] by (simp add: eval_fps_def sums_iff)
  show eval_fps (fps_expansion f z0) z = f (z0 + z) if ereal (norm z) < r for z
    using ∗[of z0 + z] and that by (simp add: eval_fps_def sums_iff dist_norm)
qed
```

We can now show several more facts about power series expansions (at least in the complex case) with relative ease that would have been trickier without complex analysis.

```
lemma
  fixes f :: complex fps and r :: ereal
  assumes ⋀z. ereal (norm z) < r ⟹ eval_fps f z ≠ 0
 shows  fps_conv_radius_inverse: fps_conv_radius (inverse f) ≥ min r (fps_conv_radius
f)
    and    eval_fps_inverse: ⋀z. ereal (norm z) < fps_conv_radius f ⟹ ereal
(norm z) < r ⟹
                          eval_fps (inverse f) z = inverse (eval_fps f z)
proof −
  define R where R = min (fps_conv_radius f) r
  have ∗: fps_conv_radius (inverse f) ≥ min r (fps_conv_radius f) ∧
        (∀ z∈eball 0 (min (fps_conv_radius f) r). eval_fps (inverse f) z = inverse
(eval_fps f z))
  proof (cases min r (fps_conv_radius f) > 0)
    case True
    define f' where f' = fps_expansion (λz. inverse (eval_fps f z)) 0
   have holo: (λz. inverse (eval_fps f z)) holomorphic_on eball 0 (min r (fps_conv_radius
f))
      using assms by (intro holomorphic_intros) auto
    from holo have radius: fps_conv_radius f' ≥ min r (fps_conv_radius f)
      unfolding f'_def by (rule conv_radius_fps_expansion)
    have eval_f': eval_fps f' z = inverse (eval_fps f z)
      if norm z < fps_conv_radius f norm z < r for z
      using that unfolding f'_def by (subst eval_fps_expansion'[OF holo]) auto

    have f ∗ f' = 1
    proof (rule eval_fps_eqD)
     from radius and True have 0 < min (fps_conv_radius f) (fps_conv_radius
f')
        by (auto simp: min_def split: if_splits)
      also have ... ≤ fps_conv_radius (f ∗ f') by (rule fps_conv_radius_mult)
      finally show ... > 0 .
    next
      from True have R > 0 by (auto simp: R_def)
      hence eventually (λz. z ∈ eball 0 R) (nhds 0)
        by (intro eventually_nhds_in_open) (auto simp: zero_ereal_def)
      thus eventually (λz. eval_fps (f ∗ f') z = eval_fps 1 z) (nhds 0)
      proof eventually_elim
        case (elim z)
        hence eval_fps (f ∗ f') z = eval_fps f z ∗ eval_fps f' z
```

```
      using radius by (intro eval_fps_mult)
                      (auto simp: R_def min_def split: if_splits intro: less_trans)
    also have eval_fps f' z = inverse (eval_fps f z)
      using elim by (intro eval_f') (auto simp: R_def)
    also from elim have eval_fps f z ≠ 0
      by (intro assms) (auto simp: R_def)
    hence eval_fps f z * inverse (eval_fps f z) = eval_fps 1 z
      by simp
    finally show eval_fps (f * f') z = eval_fps 1 z .
  qed
 qed simp_all
 hence f' = inverse f
  by (intro fps_inverse_unique [symmetric]) (simp_all add: mult_ac)
 with eval_f' and radius show ?thesis by simp
next
 case False
 hence *: eball 0 R = {}
  by (intro eball_empty) (auto simp: R_def min_def split: if_splits)
 show ?thesis
 proof safe
  from False have min r (fps_conv_radius f) ≤ 0
   by (simp add: min_def)
  also have 0 ≤ fps_conv_radius (inverse f)
   by (simp add: fps_conv_radius_def conv_radius_nonneg)
  finally show min r (fps_conv_radius f) ≤ ... .
 qed (unfold * [unfolded R_def], auto)
qed

from * show fps_conv_radius (inverse f) ≥ min r (fps_conv_radius f) by blast
from * show eval_fps (inverse f) z = inverse (eval_fps f z)
 if ereal (norm z) < fps_conv_radius f ereal (norm z) < r for z
 using that by auto
qed

lemma
 fixes f g :: complex fps and r :: ereal
 defines R ≡ Min {r, fps_conv_radius f, fps_conv_radius g}
 assumes fps_conv_radius f > 0 fps_conv_radius g > 0 r > 0
 assumes nz: ⋀z. z ∈ eball 0 r ⟹ eval_fps g z ≠ 0
 shows   fps_conv_radius_divide': fps_conv_radius (f / g) ≥ R
  and    eval_fps_divide':
         ereal (norm z) < R ⟹ eval_fps (f / g) z = eval_fps f z / eval_fps g z
proof −
 from nz[of 0] and ‹r > 0› have nz': fps_nth g 0 ≠ 0
  by (auto simp: eval_fps_at_0 zero_ereal_def)
 have R ≤ min r (fps_conv_radius g)
  by (auto simp: R_def intro: min.coboundedI2)
 also have min r (fps_conv_radius g) ≤ fps_conv_radius (inverse g)
  by (intro fps_conv_radius_inverse assms) (auto simp: zero_ereal_def)
```

```
  finally have radius: fps_conv_radius (inverse g) ≥ R .
  have R ≤ min (fps_conv_radius f) (fps_conv_radius (inverse g))
    by (intro radius min.boundedI) (auto simp: R_def intro: min.coboundedI1
min.coboundedI2)
  also have ... ≤ fps_conv_radius (f * inverse g)
    by (rule fps_conv_radius_mult)
  also have f * inverse g = f / g
    by (intro fps_divide_unit [symmetric] nz')
  finally show fps_conv_radius (f / g) ≥ R .

  assume z: ereal (norm z) < R
  have eval_fps (f * inverse g) z = eval_fps f z * eval_fps (inverse g) z
    using radius by (intro eval_fps_mult less_le_trans[OF z])
                    (auto simp: R_def intro: min.coboundedI1 min.coboundedI2)
  also have eval_fps (inverse g) z = inverse (eval_fps g z) using ‹r > 0›
    by (intro eval_fps_inverse[where r = r] less_le_trans[OF z] nz)
      (auto simp: R_def intro: min.coboundedI1 min.coboundedI2)
  also have f * inverse g = f / g by fact
  finally show eval_fps (f / g) z = eval_fps f z / eval_fps g z by (simp add:
field_split_simps)
qed

lemma
  fixes f g :: complex fps and r :: ereal
  defines R ≡ Min {r, fps_conv_radius f, fps_conv_radius g}
  assumes subdegree g ≤ subdegree f
  assumes fps_conv_radius f > 0 fps_conv_radius g > 0 r > 0
  assumes ⋀z. z ∈ eball 0 r ⟹ z ≠ 0 ⟹ eval_fps g z ≠ 0
  shows   fps_conv_radius_divide: fps_conv_radius (f / g) ≥ R
    and   eval_fps_divide:
        ereal (norm z) < R ⟹ c = fps_nth f (subdegree g) / fps_nth g (subdegree
g) ⟹
            eval_fps (f / g) z = (if z = 0 then c else eval_fps f z / eval_fps g z)
proof −
  define f' g' where f' = fps_shift (subdegree g) f and g' = fps_shift (subdegree
g) g
  have f_eq: f = f' * fps_X ^ subdegree g and g_eq: g = g' * fps_X ^ subdegree
g
    unfolding f'_def g'_def by (rule subdegree_decompose' le_refl | fact)+
  have subdegree: subdegree f' = subdegree f − subdegree g subdegree g' = 0
    using assms(2) by (simp_all add: f'_def g'_def)
  have [simp]: fps_conv_radius f' = fps_conv_radius f fps_conv_radius g' =
fps_conv_radius g
    by (simp_all add: f'_def g'_def)
  have [simp]: fps_nth f' 0 = fps_nth f (subdegree g)
               fps_nth g' 0 = fps_nth g (subdegree g) by (simp_all add: f'_def
g'_def)
  have g_nz: g ≠ 0
  proof −
```

```
    define z :: complex where z = (if r = ∞ then 1 else of_real (real_of_ereal r
/ 2))
    from ‹r > 0› have z ∈ eball 0 r
      by (cases r) (auto simp: z_def eball_def)
    moreover have z ≠ 0 using ‹r > 0›
      by (cases r) (auto simp: z_def)
    ultimately have eval_fps g z ≠ 0 by (rule assms(6))
    thus g ≠ 0 by auto
  qed
  have fg: f / g = f' * inverse g'
    by (subst f_eq, subst (2) g_eq) (insert g_nz, simp add: fps_divide_unit)

  have g'_nz: eval_fps g' z ≠ 0 if z: norm z < min r (fps_conv_radius g) for z
  proof (cases z = 0)
    case False
    with assms and z have eval_fps g z ≠ 0 by auto
    also from z have eval_fps g z = eval_fps g' z * z ^ subdegree g
      by (subst g_eq) (auto simp: eval_fps_mult)
    finally show ?thesis by auto
  qed (insert ‹g ≠ 0›, auto simp: g'_def eval_fps_at_0)

  have R ≤ min (min r (fps_conv_radius g)) (fps_conv_radius g')
    by (auto simp: R_def min.coboundedI1 min.coboundedI2)
  also have ... ≤ fps_conv_radius (inverse g')
    using g'_nz by (rule fps_conv_radius_inverse)
  finally have conv_radius_inv: R ≤ fps_conv_radius (inverse g') .
  hence R ≤ fps_conv_radius (f' * inverse g')
    by (intro order.trans[OF _ fps_conv_radius_mult])
       (auto simp: R_def intro: min.coboundedI1 min.coboundedI2)
  thus fps_conv_radius (f / g) ≥ R by (simp add: fg)

  fix z c :: complex assume z: ereal (norm z) < R
  assume c: c = fps_nth f (subdegree g) / fps_nth g (subdegree g)
  show eval_fps (f / g) z = (if z = 0 then c else eval_fps f z / eval_fps g z)
  proof (cases z = 0)
    case False
   from z and conv_radius_inv have ereal (norm z) < fps_conv_radius (inverse
g')
      by simp
    with z have eval_fps (f / g) z = eval_fps f' z * eval_fps (inverse g') z
      unfolding fg by (subst eval_fps_mult) (auto simp: R_def)
    also have eval_fps (inverse g') z = inverse (eval_fps g' z)
      using z by (intro eval_fps_inverse[of min r (fps_conv_radius g')] g'_nz)
(auto simp: R_def)
    also have eval_fps f' z * ... = eval_fps f z / eval_fps g z
      using z False assms(2) by (simp add: f'_def g'_def eval_fps_shift R_def)
    finally show ?thesis using False by simp
  qed (simp_all add: eval_fps_at_0 fg field_simps c)
qed
```

lemma *has_fps_expansion_fps_expansion* [*intro*]:
assumes *open A 0 ∈ A f holomorphic_on A*
shows *f has_fps_expansion fps_expansion f 0*
proof −
from *assms*(*1*,*2*) **obtain** *r* **where** *r*: *r > 0 ball 0 r ⊆ A*
by (*auto simp*: *open_contains_ball*)
have *holo*: *f holomorphic_on eball 0* (*ereal r*)
using *r*(*2*) **and** *assms*(*3*) **by** *auto*
from *r*(*1*) **have** *0 < ereal r* **by** *simp*
also have *r ≤ fps_conv_radius* (*fps_expansion f 0*)
using *holo* **by** (*intro conv_radius_fps_expansion*) *auto*
finally have *... > 0* **.**
moreover have *eventually* (*λz. z ∈ ball 0 r*) (*nhds 0*)
using *r*(*1*) **by** (*intro eventually_nhds_in_open*) *auto*
hence *eventually* (*λz. eval_fps* (*fps_expansion f 0*) *z* = *f z*) (*nhds 0*)
by *eventually_elim* (*subst eval_fps_expansion'*[*OF holo*], *auto*)
ultimately show *?thesis* **using** *r*(*1*) **by** (*auto simp*: *has_fps_expansion_def*)
qed

lemma *fps_conv_radius_tan*:
fixes *c* :: *complex*
assumes *c ≠ 0*
shows *fps_conv_radius* (*fps_tan c*) ≥ *pi* / (*2* ∗ *norm c*)
proof −
have *fps_conv_radius* (*fps_tan c*) ≥
Min {*pi* / (*2* ∗ *norm c*), *fps_conv_radius* (*fps_sin c*), *fps_conv_radius* (*fps_cos c*)}
unfolding *fps_tan_def*
proof (*rule fps_conv_radius_divide*)
fix *z* :: *complex* **assume** *z ∈ eball 0* (*pi* / (*2* ∗ *norm c*))
with *cos_eq_zero_imp_norm_ge*[*of c*∗*z*] *assms*
show *eval_fps* (*fps_cos c*) *z ≠ 0* **by** (*auto simp*: *norm_mult field_simps*)
qed (*insert assms*, *auto*)
thus *?thesis* **by** (*simp add*: *min_def*)
qed

lemma *eval_fps_tan*:
fixes *c* :: *complex*
assumes *norm z < pi* / (*2* ∗ *norm c*)
shows *eval_fps* (*fps_tan c*) *z* = *tan* (*c* ∗ *z*)
proof (*cases c = 0*)
case *False*
show *?thesis* **unfolding** *fps_tan_def*
proof (*subst eval_fps_divide'*[**where** *r* = *pi* / (*2* ∗ *norm c*)])
fix *z* :: *complex* **assume** *z ∈ eball 0* (*pi* / (*2* ∗ *norm c*))
with *cos_eq_zero_imp_norm_ge*[*of c*∗*z*] *assms*
show *eval_fps* (*fps_cos c*) *z ≠ 0* **using** *False* **by** (*auto simp*: *norm_mult field_simps*)

qed (*use False assms* **in** ‹*auto simp*: *field_simps tan_def*›)
qed *simp_all*

end

5 Conformal Mappings and Consequences of Cauchy's Integral Theorem

By John Harrison et al. Ported from HOL Light by L C Paulson (2016)

Also Cauchy's residue theorem by Wenda Li (2016)

theory *Conformal_Mappings*
imports *Cauchy_Integral_Formula*

begin

5.1 Analytic continuation

proposition *isolated_zeros*:
assumes *holf*: *f holomorphic_on S*
and *open S connected S ξ ∈ S f ξ = 0 β ∈ S f β ≠ 0*
obtains *r* **where** *0 < r* **and** *ball ξ r ⊆ S* **and**
⋀*z*. *z ∈ ball ξ r − {ξ} ⟹ f z ≠ 0*
proof −
obtain *r* **where** *0 < r* **and** *r*: *ball ξ r ⊆ S*
using ‹*open S*› ‹*ξ ∈ S*› *open_contains_ball_eq* **by** *blast*
have *powf*: ((λ*n*. (*deriv* ^^ *n*) *f ξ* / (*fact n*) ∗ (*z* − *ξ*) ^ *n*) *sums f z*) **if** *z ∈ ball ξ r* **for** *z*
by (*intro holomorphic_power_series* [*OF* _ *that*] *holomorphic_on_subset* [*OF holf r*])
obtain *m* **where** *m*: (*deriv* ^^ *m*) *f ξ* / (*fact m*) ≠ *0*
using *holomorphic_fun_eq_0_on_connected* [*OF holf* ‹*open S*› ‹*connected S*› _ ‹*ξ ∈ S*› ‹*β ∈ S*›] ‹*f β ≠ 0*›
by *auto*
then have *m ≠ 0* **using** *assms*(*5*) *funpow_0* **by** *fastforce*
obtain *s* **where** *0 < s* **and** *s*: ⋀*z*. *z ∈ cball ξ s − {ξ} ⟹ f z ≠ 0*
using *powser_0_nonzero* [*OF* ‹*0 < r*› *powf* ‹*f ξ = 0*› *m*]
by (*metis* ‹*m ≠ 0*› *dist_norm mem_ball norm_minus_commute not_gr_zero*)
have *0 < min r s* **by** (*simp add*: ‹*0 < r*› ‹*0 < s*›)
then show *thesis*
apply (*rule that*)
using *r s* **by** *auto*
qed

proposition *analytic_continuation*:
assumes *holf*: *f holomorphic_on S*
and *open S* **and** *connected S*
and *U ⊆ S* **and** *ξ ∈ S*

```
    and ξ islimpt U
    and fU0 [simp]: ⋀z. z ∈ U ⟹ f z = 0
    and w ∈ S
  shows f w = 0
proof −
  obtain e where 0 < e and e: cball ξ e ⊆ S
    using ‹open S› ‹ξ ∈ S› open_contains_cball_eq by blast
  define T where T = cball ξ e ∩ U
  have contf: continuous_on (closure T) f
   by (metis T_def closed_cball closure_minimal e holf holomorphic_on_imp_continuous_on
          holomorphic_on_subset inf.cobounded1)
  have fT0 [simp]: ⋀x. x ∈ T ⟹ f x = 0
    by (simp add: T_def)
  have ⋀r. ⟦∀e>0. ∃x'∈U. x' ≠ ξ ∧ dist x' ξ < e; 0 < r⟧ ⟹ ∃x'∈cball ξ e ∩
U. x' ≠ ξ ∧ dist x' ξ < r
   by (metis ‹0 < e› IntI dist_commute less_eq_real_def mem_cball min_less_iff_conj)
  then have ξ islimpt T using ‹ξ islimpt U›
    by (auto simp: T_def islimpt_approachable)
  then have ξ ∈ closure T
    by (simp add: closure_def)
  then have f ξ = 0
    by (auto simp: continuous_constant_on_closure [OF contf])
  moreover have ⋀r. ⟦0 < r; ⋀z. z ∈ ball ξ r − {ξ} ⟹ f z ≠ 0⟧ ⟹ False
     by (metis open_ball ‹ξ islimpt T› centre_in_ball fT0 insertE insert_Diff
islimptE)
  ultimately show ?thesis
    by (metis ‹open S› ‹connected S› ‹ξ ∈ S› ‹w ∈ S› holf isolated_zeros)
qed

corollary analytic_continuation_open:
  assumes open s and open s' and s ≠ {} and connected s'
      and s ⊆ s'
  assumes f holomorphic_on s' and g holomorphic_on s'
      and ⋀z. z ∈ s ⟹ f z = g z
  assumes z ∈ s'
  shows   f z = g z
proof −
  from ‹s ≠ {}› obtain ξ where ξ ∈ s by auto
  with ‹open s› have ξ: ξ islimpt s
    by (intro interior_limit_point) (auto simp: interior_open)
  have f z − g z = 0
    by (rule analytic_continuation[of λz. f z − g z s' s ξ])
       (insert assms ‹ξ ∈ s› ξ, auto intro: holomorphic_intros)
  thus ?thesis by simp
qed

corollary analytic_continuation':
  assumes f holomorphic_on S open S connected S
      and U ⊆ S ξ ∈ S ξ islimpt U
```

```
    and f constant_on U
  shows f constant_on S
proof −
  obtain c where c: ⋀x. x ∈ U ⟹ f x − c = 0
    by (metis ‹f constant_on U› constant_on_def diff_self)
  have (λz. f z − c) holomorphic_on S
    using assms by (intro holomorphic_intros)
  with c analytic_continuation assms have ⋀x. x ∈ S ⟹ f x − c = 0
    by blast
  then show ?thesis
    unfolding constant_on_def by force
qed

lemma holomorphic_compact_finite_zeros:
  assumes S: f holomorphic_on S open S connected S
      and compact K K ⊆ S
      and ¬ f constant_on S
    shows finite {z∈K. f z = 0}
proof (rule ccontr)
  assume infinite {z∈K. f z = 0}
  then obtain z where z ∈ K and z: z islimpt {z∈K. f z = 0}
    using ‹compact K› by (auto simp: compact_eq_Bolzano_Weierstrass)
  moreover have {z∈K. f z = 0} ⊆ S
    using ‹K ⊆ S› by blast
    ultimately show False
    using assms analytic_continuation [OF S] unfolding constant_on_def
    by blast
qed
```

5.2 Open mapping theorem

```
lemma holomorphic_contract_to_zero:
  assumes contf: continuous_on (cball ξ r) f
      and holf: f holomorphic_on ball ξ r
      and 0 < r
      and norm_less: ⋀z. norm(ξ − z) = r ⟹ norm(f ξ) < norm(f z)
  obtains z where z ∈ ball ξ r f z = 0
proof −
  { assume fnz: ⋀w. w ∈ ball ξ r ⟹ f w ≠ 0
    then have 0 < norm (f ξ)
      by (simp add: ‹0 < r›)
    have fnz': ⋀w. w ∈ cball ξ r ⟹ f w ≠ 0
      by (metis norm_less dist_norm fnz less_eq_real_def mem_ball mem_cball
norm_not_less_zero norm_zero)
    have frontier(cball ξ r) ≠ {}
      using ‹0 < r› by simp
    define g where [abs_def]: g z = inverse (f z) for z
    have contg: continuous_on (cball ξ r) g
      unfolding g_def using contf continuous_on_inverse fnz' by blast
```

```
    have holg: g holomorphic_on ball ξ r
      unfolding g_def using fnz holf holomorphic_on_inverse by blast
    have frontier (cball ξ r) ⊆ cball ξ r
      by (simp add: subset_iff)
    then have contf': continuous_on (frontier (cball ξ r)) f
        and contg': continuous_on (frontier (cball ξ r)) g
      by (blast intro: contf contg continuous_on_subset)+
    have froc: frontier(cball ξ r) ≠ {}
      using ‹0 < r› by simp
    moreover have continuous_on (frontier (cball ξ r)) (norm o f)
      using contf' continuous_on_compose continuous_on_norm_id by blast
    ultimately obtain w where w: w ∈ frontier(cball ξ r)
                          and now: ⋀x. x ∈ frontier(cball ξ r) ⟹ norm (f w) ≤ norm
(f x)
      using continuous_attains_inf [OF compact_frontier [OF compact_cball]]
      by (metis comp_apply)
    then have fw: 0 < norm (f w)
      by (simp add: fnz')
    have continuous_on (frontier (cball ξ r)) (norm o g)
      using contg' continuous_on_compose continuous_on_norm_id by blast
    then obtain v where v: v ∈ frontier(cball ξ r)
                    and nov: ⋀x. x ∈ frontier(cball ξ r) ⟹ norm (g v) ≥ norm (g x)
      using continuous_attains_sup [OF compact_frontier [OF compact_cball]
froc] by force
    then have fv: 0 < norm (f v)
      by (simp add: fnz')
    have norm ((deriv ^^ 0) g ξ) ≤ fact 0 * norm (g v) / r ^ 0
      by (rule Cauchy_inequality [OF holg contg ‹0 < r›]) (simp add: dist_norm
nov)
    then have cmod (g ξ) ≤ cmod (g v)
      by simp
    moreover have cmod (ξ - w) = r
      by (metis (no_types) dist_norm frontier_cball mem_sphere w)
    ultimately obtain wr: norm (ξ - w) = r and nfw: norm (f w) ≤ norm (f ξ)
      unfolding g_def
      by (metis (no_types) ‹0 < cmod (f ξ)› less_imp_inverse_less norm_inverse
not_le now order_trans v)
    with fw have False
      using norm_less by force
  }
  with that show ?thesis by blast
qed

theorem open_mapping_thm:
  assumes holf: f holomorphic_on S
      and S: open S and connected S
      and open U and U ⊆ S
      and fne: ¬ f constant_on S
    shows open (f ‘ U)
```

```
proof −
  have ∗: open (f ‘ U)
          if U ≠ {} and U: open U connected U and f holomorphic_on U and
fneU: ⋀x. ∃ y ∈ U. f y ≠ x
          for U
  proof (clarsimp simp: open_contains_ball)
    fix ξ assume ξ: ξ ∈ U
    show ∃ e>0. ball (f ξ) e ⊆ f ‘ U
    proof −
      have hol: (λz. f z − f ξ) holomorphic_on U
        by (rule holomorphic_intros that)+
      obtain s where 0 < s and sbU: ball ξ s ⊆ U
                and sne: ⋀z. z ∈ ball ξ s − {ξ} ⟹ (λz. f z − f ξ) z ≠ 0
        using isolated_zeros [OF hol U ξ]  by (metis fneU right_minus_eq)
      obtain r where 0 < r and r: cball ξ r ⊆ ball ξ s
        using ‹0 < s› by (rule_tac r=s/2 in that) auto
      have cball ξ r ⊆ U
        using sbU r by blast
      then have frsbU: frontier (cball ξ r) ⊆ U
        using Diff_subset frontier_def order_trans by fastforce
      then have cof: compact (frontier(cball ξ r))
        by blast
      have frne: frontier (cball ξ r) ≠ {}
        using ‹0 < r› by auto
      have contfr: continuous_on (frontier (cball ξ r)) (λz. norm (f z − f ξ))
        by (metis continuous_on_norm continuous_on_subset frsbU hol holomor-
phic_on_imp_continuous_on)
      obtain w where norm (ξ − w) = r
                and w: (⋀z. norm (ξ − z) = r ⟹ norm (f w − f ξ) ≤ norm(f z −
f ξ))
        using continuous_attains_inf [OF cof frne contfr] by (auto simp: dist_norm)
      moreover define ε where ε ≡ norm (f w − f ξ) / 3
      ultimately have 0 < ε
        using ‹0 < r› dist_complex_def r sne by auto
      have ball (f ξ) ε ⊆ f ‘ U
      proof
        fix γ
        assume γ: γ ∈ ball (f ξ) ε
        have ∗: cmod (γ − f ξ) < cmod (γ − f z) if cmod (ξ − z) = r for z
        proof −
          have lt: cmod (f w − f ξ) / 3 < cmod (γ − f z)
            using w [OF that] γ
            using dist_triangle2 [of f ξ γ  f z] dist_triangle2 [of f ξ f z γ]
            by (simp add: ε_def dist_norm norm_minus_commute)
          show ?thesis
            by (metis ε_def dist_commute dist_norm less_trans lt mem_ball γ)
        qed
       have continuous_on (cball ξ r) (λz. γ − f z)
          using ‹cball ξ r ⊆ U› ‹f holomorphic_on U›
```

```
      by (force intro: continuous_intros continuous_on_subset holomorphic_on_imp_continuous_on)
      moreover have (λz. γ − f z) holomorphic_on ball ξ r
        using ‹cball ξ r ⊆ U› ball_subset_cball holomorphic_on_subset that(4)
        by (intro holomorphic_intros) blast
      ultimately obtain z where z ∈ ball ξ r γ − f z = 0
        using ∗ ‹0 < r› holomorphic_contract_to_zero by blast
      then show γ ∈ f ‘ U
        using ‹cball ξ r ⊆ U› by fastforce
    qed
    then show ?thesis using ‹0 < ε› by blast
  qed
 qed
 have open (f ‘ X) if X ∈ components U for X
 proof −
  have holfU: f holomorphic_on U
    using ‹U ⊆ S› holf holomorphic_on_subset by blast
  have X ≠ {}
    using that by (simp add: in_components_nonempty)
  moreover have open X
    using that ‹open U› open_components by auto
  moreover have connected X
    using that in_components_maximal by blast
  moreover have f holomorphic_on X
    by (meson that holfU holomorphic_on_subset in_components_maximal)
  moreover have ∃ y∈X. f y ≠ x for x
  proof (rule ccontr)
    assume not: ¬ (∃ y∈X. f y ≠ x)
    have X ⊆ S
      using ‹U ⊆ S› in_components_subset that by blast
    obtain w where w: w ∈ X using ‹X ≠ {}› by blast
    have wis: w islimpt X
      using w ‹open X› interior_eq by auto
    have hol: (λz. f z − x) holomorphic_on S
      by (simp add: holf holomorphic_on_diff)
    with fne [unfolded constant_on_def]
        analytic_continuation[OF hol S ‹connected S› ‹X ⊆ S› _ wis] not ‹X ⊆
S› w
    show False by auto
  qed
  ultimately show ?thesis
    by (rule ∗)
 qed
 then have open (f ‘ ⋃(components U))
  by (metis (no_types, lifting) imageE image_Union open_Union)
 then show ?thesis
  by force
qed
```

No need for S to be connected. But the nonconstant condition is stronger.

corollary *open_mapping_thm2*:
  assumes *holf*: *f holomorphic_on S*
    and *S*: *open S*
    and *open U U* $\subseteq$ *S*
    and *fnc*: $\bigwedge X.$ $[\![$*open X*; *X* $\subseteq$ *S*; *X* $\neq$ {}$]\!] \Longrightarrow \neg$ *f constant_on X*
   shows *open* (*f* ‘ *U*)
proof −
  have *S* = $\bigcup$(*components S*) **by** *simp*
  with ‹*U* $\subseteq$ *S*› **have** *U* = ($\bigcup$ *C* $\in$ *components S*. *C* $\cap$ *U*) **by** *auto*
  then have *f* ‘ *U* = ($\bigcup$ *C* $\in$ *components S*. *f* ‘ (*C* $\cap$ *U*))
    using *image_UN* **by** *fastforce*
  moreover
  { fix *C* **assume** *C* $\in$ *components S*
    with *S* ‹*C* $\in$ *components S*› *open_components in_components_connected*
    have *C*: *open C connected C* **by** *auto*
    have *C* $\subseteq$ *S*
      by (*metis* ‹*C* $\in$ *components S*› *in_components_maximal*)
    have *nf*: $\neg$ *f constant_on C*
      using ‹*open C*› ‹*C* $\in$ *components S*› ‹*C* $\subseteq$ *S*› *fnc in_components_nonempty*
by *blast*
    have *f holomorphic_on C*
      by (*metis holf holomorphic_on_subset* ‹*C* $\subseteq$ *S*›)
    then have *open* (*f* ‘ (*C* $\cap$ *U*))
      by (*meson C* ‹*open U*› *inf_le1 nf open_Int open_mapping_thm*)
  } ultimately show *?thesis*
    by *force*
qed

corollary *open_mapping_thm3*:
  assumes *holf*: *f holomorphic_on S*
    and *open S* **and** *injf*: *inj_on f S*
   shows *open* (*f* ‘ *S*)
proof (*rule open_mapping_thm2* [*OF holf*])
  show $\bigwedge X.$ $[\![$*open X*; *X* $\subseteq$ *S*; *X* $\neq$ {}$]\!] \Longrightarrow \neg$ *f constant_on X*
    using *inj_on_subset injective_not_constant injf* **by** *blast*
qed (*use assms* **in** *auto*)

5.3 Maximum modulus principle

If f is holomorphic, then its norm (modulus) cannot exhibit a true local maximum that is properly within the domain of f.

proposition *maximum_modulus_principle*:
  assumes *holf*: *f holomorphic_on S*
    and *S*: *open S* **and** *connected S*
    and *open U* **and** *U* $\subseteq$ *S* **and** $\xi \in U$
    and *no*: $\bigwedge z.$ *z* $\in$ *U* $\Longrightarrow$ *norm*(*f z*) $\leq$ *norm*(*f* ξ)
   shows *f constant_on S*
proof (*rule ccontr*)
  assume $\neg$ *f constant_on S*

```
  then have open (f ‘ U)
    using open_mapping_thm assms by blast
  moreover have ¬ open (f ‘ U)
  proof −
    have ∃t. cmod (f ξ − t) < e ∧ t ∉ f ‘ U if 0 < e for e
      using that
      apply (rule_tac x=if 0 < Re(f ξ) then f ξ + (e/2) else f ξ − (e/2) in exI)
      apply (simp add: dist_norm)
      apply (fastforce simp: cmod_Re_le_iff dest!: no dest: sym)
      done
    then show ?thesis
     unfolding open_contains_ball by (metis ‹ξ ∈ U› contra_subsetD dist_norm
imageI mem_ball)
  qed
  ultimately show False
    by blast
qed

proposition maximum_modulus_frontier:
  assumes holf: f holomorphic_on (interior S)
      and contf: continuous_on (closure S) f
      and bos: bounded S
      and leB: ⋀z. z ∈ frontier S ⟹ norm(f z) ≤ B
      and ξ ∈ S
    shows norm(f ξ) ≤ B
proof −
  have compact (closure S) using bos
    by (simp add: bounded_closure compact_eq_bounded_closed)
  moreover have continuous_on (closure S) (cmod ∘ f)
    using contf continuous_on_compose continuous_on_norm_id by blast
  ultimately obtain z where z ∈ closure S and z: ⋀y. y ∈ closure S ⟹ (cmod
∘ f) y ≤ (cmod ∘ f) z
    using continuous_attains_sup [of closure S norm o f] ‹ξ ∈ S› by auto
  then consider z ∈ frontier S | z ∈ interior S using frontier_def by auto
  then have norm(f z) ≤ B
  proof cases
    case 1 then show ?thesis using leB by blast
  next
    case 2
    have f constant_on (connected_component_set (interior S) z)
    proof (rule maximum_modulus_principle)
      show f holomorphic_on connected_component_set (interior S) z
        by (metis connected_component_subset holf holomorphic_on_subset)
      show zin: z ∈ connected_component_set (interior S) z
        by (simp add: 2)
      show ⋀W. W ∈ connected_component_set (interior S) z ⟹ cmod (f W)
≤ cmod (f z)
        using closure_def connected_component_subset z by fastforce
    qed (auto simp: open_connected_component)
```

then obtain c **where** c: $\bigwedge w.\ w \in$ *connected_component_set* (*interior* S) $z \Longrightarrow f\ w = c$
by (*auto simp*: *constant_on_def*)
have f ‘ *closure*(*connected_component_set* (*interior* S) z) $\subseteq \{c\}$
proof (*rule image_closure_subset*)
show *continuous_on* (*closure* (*connected_component_set* (*interior* S) z)) f
by (*meson closure_mono connected_component_subset contf continuous_on_subset interior_subset*)
qed (*use* c **in** *auto*)
then have cc: $\bigwedge w.\ w \in$ *closure*(*connected_component_set* (*interior* S) z) $\Longrightarrow f\ w = c$ **by** *blast*
have *connected_component* (*interior* S) z z
by (*simp add*: *2*)
moreover have *connected_component_set* (*interior* S) $z \neq$ *UNIV*
by (*metis bos bounded_interior connected_component_eq_UNIV not_bounded_UNIV*)
ultimately have *frontier*(*connected_component_set* (*interior* S) z) $\neq \{\}$
by (*meson 2 connected_component_eq_empty frontier_not_empty*)
then obtain w **where** w: $w \in$ *frontier*(*connected_component_set* (*interior* S) z)
by *auto*
then have *norm* (f z) = *norm* (f w) **by** (*simp add*: *2 c cc frontier_def*)
also have ... $\leq B$
using w *frontier_interior_subset frontier_of_connected_component_subset*
by (*blast intro*: *leB*)
finally show *?thesis* **.**
qed
then show *?thesis*
using z ‹$\xi \in S$› *closure_subset* **by** *fastforce*
qed

corollary *maximum_real_frontier*:
assumes *holf*: f *holomorphic_on* (*interior* S)
and *contf*: *continuous_on* (*closure* S) f
and *bos*: *bounded* S
and *leB*: $\bigwedge z.\ z \in$ *frontier* $S \Longrightarrow$ *Re*(f z) $\leq B$
and $\xi \in S$
shows *Re*(f ξ) $\leq B$
using *maximum_modulus_frontier* [*of exp o f S exp B*]
Transcendental.continuous_on_exp holomorphic_on_compose holomorphic_on_exp assms
by *auto*

5.4 Factoring out a zero according to its order

lemma *holomorphic_factor_order_of_zero*:
assumes *holf*: f *holomorphic_on* S
and *os*: *open* S
and $\xi \in S$ $0 < n$
and *dnz*: (*deriv* ^^ n) f $\xi \neq 0$

```
    and dfz: ⋀i. ⟦0 < i; i < n⟧ ⟹ (deriv ^^ i) f ξ = 0
  obtains g r where 0 < r
            g holomorphic_on ball ξ r
            ⋀w. w ∈ ball ξ r ⟹ f w - f ξ = (w - ξ)^n * g w
            ⋀w. w ∈ ball ξ r ⟹ g w ≠ 0
proof -
  obtain r where r>0 and r: ball ξ r ⊆ S using assms by (blast elim!: openE)
  then have holfb: f holomorphic_on ball ξ r
    using holf holomorphic_on_subset by blast
  define g where g w = suminf (λi. (deriv ^^ (i + n)) f ξ / (fact(i + n)) * (w
- ξ)^i) for w
  have sumsg: (λi. (deriv ^^ (i + n)) f ξ / (fact(i + n)) * (w - ξ)^i) sums g w
   and feq: f w - f ξ = (w - ξ)^n * g w
      if w: w ∈ ball ξ r for w
  proof -
    define powf where powf = (λi. (deriv ^^ i) f ξ/(fact i) * (w - ξ)^i)
    have [simp]: powf 0 = f ξ
      by (simp add: powf_def)
    have sing: {..<n} - {i. powf i = 0} = (if f ξ = 0 then {} else {0})
      unfolding powf_def using ‹0 < n› dfz by (auto simp: dfz; metis funpow_0
not_gr0)
    have powf sums f w
      unfolding powf_def by (rule holomorphic_power_series [OF holfb w])
    moreover have (∑ i<n. powf i) = f ξ
      by (subst sum.setdiff_irrelevant [symmetric]; simp add: dfz sing)
    ultimately have fsums: (λi. powf (i+n)) sums (f w - f ξ)
      using w sums_iff_shift' by metis
    then have *: summable (λi. (w - ξ) ^ n * ((deriv ^^ (i + n)) f ξ * (w - ξ)
^ i / fact (i + n)))
      unfolding powf_def using sums_summable
      by (auto simp: power_add mult_ac)
    have summable (λi. (deriv ^^ (i + n)) f ξ * (w - ξ) ^ i / fact (i + n))
    proof (cases w=ξ)
      case False then show ?thesis
        using summable_mult [OF *, of 1 / (w - ξ) ^ n] by simp
    next
      case True then show ?thesis
      by (auto simp: Power.semiring_1_class.power_0_left intro!: summable_finite
[of {0}]
                split: if_split_asm)
    qed
    then show sumsg: (λi. (deriv ^^ (i + n)) f ξ / (fact(i + n)) * (w - ξ)^i)
sums g w
      by (simp add: summable_sums_iff g_def)
    show f w - f ξ = (w - ξ)^n * g w
      using sums_mult [OF sumsg, of (w - ξ) ^ n]
    by (intro sums_unique2 [OF fsums]) (auto simp: power_add mult_ac powf_def)
  qed
  then have holg: g holomorphic_on ball ξ r
```

```
    by (meson sumsg power_series_holomorphic)
  then have contg: continuous_on (ball ξ r) g
    by (blast intro: holomorphic_on_imp_continuous_on)
  have g ξ ≠ 0
    using dnz unfolding g_def
    by (subst suminf_finite [of {0}]) auto
  obtain d where 0 < d and d: ⋀w. w ∈ ball ξ d ⟹ g w ≠ 0
    using ‹0 < r› continuous_on_avoid [OF contg _ ‹g ξ ≠ 0›]
    by (metis centre_in_ball le_cases mem_ball mem_ball_leI)
  show ?thesis
  proof
    show g holomorphic_on ball ξ (min r d)
      using holg by (auto simp: feq holomorphic_on_subset subset_ball d)
  qed (use ‹0 < r› ‹0 < d› in ‹auto simp: feq d›)
qed


lemma holomorphic_factor_order_of_zero_strong:
  assumes holf: f holomorphic_on S open S  ξ ∈ S 0 < n
      and (deriv ^^ n) f ξ ≠ 0
      and ⋀i. ⟦0 < i; i < n⟧ ⟹ (deriv ^^ i) f ξ = 0
   obtains g r where 0 < r
                     g holomorphic_on ball ξ r
                     ⋀w. w ∈ ball ξ r ⟹ f w - f ξ = ((w - ξ) * g w) ^ n
                     ⋀w. w ∈ ball ξ r ⟹ g w ≠ 0
proof -
  obtain g r where 0 < r
                and holg: g holomorphic_on ball ξ r
                and feq: ⋀w. w ∈ ball ξ r ⟹ f w - f ξ = (w - ξ)^n * g w
                and gne: ⋀w. w ∈ ball ξ r ⟹ g w ≠ 0
    by (auto intro: holomorphic_factor_order_of_zero [OF assms])
  have con: continuous_on (ball ξ r) (λz. deriv g z / g z)
    by (rule continuous_intros) (auto simp: gne holg holomorphic_deriv holomorphic_on_imp_continuous_on)
  have cd: (λz. deriv g z / g z) field_differentiable at x if dist ξ x < r for x
  proof (intro derivative_intros)
    show deriv g field_differentiable at x
        using that holg mem_ball by (blast intro: holomorphic_deriv holomorphic_on_imp_differentiable_at)
    show g field_differentiable at x
     by (metis that open_ball at_within_open holg holomorphic_on_def mem_ball)
    qed (simp add: gne that)
    obtain h where h: ⋀x. x ∈ ball ξ r ⟹ (h has_field_derivative deriv g x / g x) (at x)
      using holomorphic_convex_primitive [of ball ξ r {} λz. deriv g z / g z]
       by (metis (no_types, lifting) Diff_empty at_within_interior cd con convex_ball infinite_imp_nonempty interior_ball mem_ball)
  then have continuous_on (ball ξ r) h
   by (metis open_ball holomorphic_on_imp_continuous_on holomorphic_on_open)
```

```
  then have con: continuous_on (ball ξ r) (λx. exp (h x) / g x)
   by (auto intro!: continuous_intros simp add: holg holomorphic_on_imp_continuous_on
gne)
  have 0: dist ξ x < r ⟹ ((λx. exp (h x) / g x) has_field_derivative 0) (at x)
for x
   apply (rule h derivative_eq_intros DERIV_deriv_iff_field_differentiable [THEN
iffD2] | simp)+
    using holg by (auto simp: holomorphic_on_imp_differentiable_at gne h)
  obtain c where c: ⋀x. x ∈ ball ξ r ⟹ exp (h x) / g x = c
   by (rule DERIV_zero_connected_constant [of ball ξ r {} λx. exp(h x) / g x])
(auto simp: con 0)
  have hol: (λz. exp ((Ln (inverse c) + h z) / of_nat n)) holomorphic_on ball ξ r
   proof (intro holomorphic_intros holomorphic_on_compose [unfolded o_def,
where g = exp])
    show h holomorphic_on ball ξ r
     using h holomorphic_on_open by blast
  qed (use ‹0 < n› in auto)
  show ?thesis
  proof
    show ⋀w. w ∈ ball ξ r ⟹ f w - f ξ = ((w - ξ) * exp ((Ln (inverse c) + h
w) / of_nat n)) ^ n
     using ‹0 < n›
      by (auto simp: feq power_mult_distrib exp_divide_power_eq exp_add gne
simp flip: c)
  qed (use hol ‹0 < r› in auto)
qed


lemma
  fixes k :: 'a::wellorder
  assumes a_def: a == LEAST x. P x and P: P k
  shows def_LeastI: P a and def_Least_le: a ≤ k
unfolding a_def
by (rule LeastI Least_le; rule P)+

lemma holomorphic_factor_zero_nonconstant:
  assumes holf: f holomorphic_on S and S: open S connected S
     and ξ ∈ S f ξ = 0
     and nonconst: ¬ f constant_on S
   obtains g r n
     where 0 < n  0 < r  ball ξ r ⊆ S
           g holomorphic_on ball ξ r
           ⋀w. w ∈ ball ξ r ⟹ f w = (w - ξ) ^ n * g w
           ⋀w. w ∈ ball ξ r ⟹ g w ≠ 0
proof (cases ∀ n>0. (deriv ^^ n) f ξ = 0)
  case True then show ?thesis
    using holomorphic_fun_eq_const_on_connected [OF holf S _ ‹ξ ∈ S›] non-
const by (simp add: constant_on_def)
next
```

case *False*
then obtain *n0* **where** *n0* > *0* **and** *n0*: (*deriv* ^^ *n0*) *f* ξ ≠ *0* **by** *blast*
obtain *r0* **where** *r0* > *0* *ball* ξ *r0* ⊆ *S* **using** *S openE* ‹ξ ∈ *S*› **by** *auto*
define *n* **where** *n* ≡ *LEAST n*. (*deriv* ^^ *n*) *f* ξ ≠ *0*
have *n_ne*: (*deriv* ^^ *n*) *f* ξ ≠ *0*
by (*rule def_LeastI* [*OF n_def*]) (*rule n0*)
then have *0* < *n* **using** ‹*f* ξ = *0*›
using *funpow_0* **by** *fastforce*
have *n_min*: ⋀*k*. *k* < *n* ⟹ (*deriv* ^^ *k*) *f* ξ = *0*
using *def_Least_le* [*OF n_def*] *not_le* **by** *blast*
then obtain *g r1*
where *g*: *0* < *r1* *g holomorphic_on ball* ξ *r1*
and *geq*: ⋀*w*. *w* ∈ *ball* ξ *r1* ⟹ *f w* = (*w* − ξ) ^ *n* ∗ *g w*
and *g0*: ⋀*w*. *w* ∈ *ball* ξ *r1* ⟹ *g w* ≠ *0*
by (*auto intro*: *holomorphic_factor_order_of_zero* [*OF holf* ‹*open S*› ‹ξ ∈ *S*› ‹*n* > *0*› *n_ne*] *simp*: ‹*f* ξ = *0*›)
show *?thesis*
proof
show *g holomorphic_on ball* ξ (*min r0 r1*)
using *g* **by** *auto*
show ⋀*w*. *w* ∈ *ball* ξ (*min r0 r1*) ⟹ *f w* = (*w* − ξ) ^ *n* ∗ *g w*
by (*simp add*: *geq*)
qed (*use* ‹*0* < *n*› ‹*0* < *r0*› ‹*0* < *r1*› ‹*ball* ξ *r0* ⊆ *S*› *g0* **in** *auto*)
qed

lemma *holomorphic_lower_bound_difference*:
assumes *holf*: *f holomorphic_on S* **and** *S*: *open S connected S*
and ξ ∈ *S* **and** φ ∈ *S*
and *fne*: *f* φ ≠ *f* ξ
obtains *k n r*
where *0* < *k* *0* < *r*
ball ξ *r* ⊆ *S*
⋀*w*. *w* ∈ *ball* ξ *r* ⟹ *k* ∗ *norm*(*w* − ξ) ^ *n* ≤ *norm*(*f w* − *f* ξ)
proof −
define *n* **where** *n* = (*LEAST n*. *0* < *n* ∧ (*deriv* ^^ *n*) *f* ξ ≠ *0*)
obtain *n0* **where** *0* < *n0* **and** *n0*: (*deriv* ^^ *n0*) *f* ξ ≠ *0*
using *fne holomorphic_fun_eq_const_on_connected* [*OF holf S*] ‹ξ ∈ *S*› ‹φ ∈ *S*› **by** *blast*
then have *0* < *n* **and** *n_ne*: (*deriv* ^^ *n*) *f* ξ ≠ *0*
unfolding *n_def* **by** (*metis* (*mono_tags*, *lifting*) *LeastI*)+
have *n_min*: ⋀*k*. ⟦*0* < *k*; *k* < *n*⟧ ⟹ (*deriv* ^^ *k*) *f* ξ = *0*
unfolding *n_def* **by** (*blast dest*: *not_less_Least*)
then obtain *g r*
where *0* < *r* **and** *holg*: *g holomorphic_on ball* ξ *r*
and *fne*: ⋀*w*. *w* ∈ *ball* ξ *r* ⟹ *f w* − *f* ξ = (*w* − ξ) ^ *n* ∗ *g w*
and *gnz*: ⋀*w*. *w* ∈ *ball* ξ *r* ⟹ *g w* ≠ *0*
by (*auto intro*: *holomorphic_factor_order_of_zero* [*OF holf* ‹*open S*› ‹ξ ∈ *S*› ‹*n* > *0*› *n_ne*])

obtain e **where** $e{>}0$ **and** e: $ball\ \xi\ e \subseteq S$ **using** *assms* **by** (*blast elim*!: *openE*)
then have *holfb*: f *holomorphic_on ball* ξ e
using *holf holomorphic_on_subset* **by** *blast*
define d **where** $d = (min\ e\ r)\ /\ 2$
have $0 < d$ **using** ‹$0 < r$› ‹$0 < e$› **by** (*simp add*: *d_def*)
have $d < r$
using ‹$0 < r$› **by** (*auto simp*: *d_def*)
then have *cbb*: $cball\ \xi\ d \subseteq ball\ \xi\ r$
by (*auto simp*: *cball_subset_ball_iff*)
then have g *holomorphic_on cball* ξ d
by (*rule holomorphic_on_subset* [*OF holg*])
then have *closed* (g ‘ *cball* ξ d)
by (*simp add*: *compact_imp_closed compact_continuous_image holomorphic_on_imp_continuous_on*)
moreover have g ‘ *cball* ξ $d \neq \{\}$
using ‹$0 < d$› **by** *auto*
ultimately obtain x **where** x: $x \in g$ ‘ *cball* ξ d **and** $\bigwedge y.\ y \in g$ ‘ *cball* ξ $d \Longrightarrow$ *dist 0* $x \leq$ *dist 0* y
by (*rule distance_attains_inf*) *blast*
then have *leg*: $\bigwedge w.\ w \in cball\ \xi\ d \Longrightarrow norm\ x \leq norm\ (g\ w)$
by *auto*
have $ball\ \xi\ d \subseteq cball\ \xi\ d$ **by** *auto*
also have ... $\subseteq ball\ \xi\ e$ **using** ‹$0 < d$› *d_def* **by** *auto*
also have ... $\subseteq S$ **by** (*rule e*)
finally have *dS*: $ball\ \xi\ d \subseteq S$ **.**
have $x \neq 0$ **using** *gnz x* ‹$d < r$› **by** *auto*
show *thesis*
proof
show $\bigwedge w.\ w \in ball\ \xi\ d \Longrightarrow cmod\ x * cmod\ (w - \xi)\ \hat{}\ n \leq cmod\ (f\ w - f\ \xi)$
using ‹$d < r$› *leg* **by** (*auto simp*: *fne norm_mult norm_power algebra_simps mult_right_mono*)
qed (*use dS* ‹$x \neq 0$› ‹$d > 0$› **in** *auto*)
qed

lemma
assumes *holf*: f *holomorphic_on* $(S - \{\xi\})$ **and** ξ: $\xi \in interior\ S$
shows *holomorphic_on_extend_lim*:
$(\exists g.\ g\ holomorphic\_on\ S \wedge (\forall z \in S - \{\xi\}.\ g\ z = f\ z)) \longleftrightarrow$
$((\lambda z.\ (z - \xi) * f\ z) \longrightarrow 0)\ (at\ \xi)$
(**is** *?P* = *?Q*)
and *holomorphic_on_extend_bounded*:
$(\exists g.\ g\ holomorphic\_on\ S \wedge (\forall z \in S - \{\xi\}.\ g\ z = f\ z)) \longleftrightarrow$
$(\exists B.\ eventually\ (\lambda z.\ norm(f\ z) \leq B)\ (at\ \xi))$
(**is** *?P* = *?R*)
proof −
obtain δ **where** $0 < \delta$ **and** δ: $ball\ \xi\ \delta \subseteq S$
using ξ *mem_interior* **by** *blast*
have *?R* **if** *holg*: g *holomorphic_on* S **and** *gf*: $\bigwedge z.\ z \in S - \{\xi\} \Longrightarrow g\ z = f\ z$
for g
proof −

```
    have §: cmod (f x) ≤ cmod (g ξ) + 1 if x ≠ ξ dist x ξ < δ dist (g x) (g ξ) <
1 for x
    proof −
      have x ∈ S
        by (metis δ dist_commute mem_ball subsetD that(2))
      with that gf [of x] show ?thesis
        using norm_triangle_ineq2 [of f x g ξ] dist_complex_def by auto
    qed
    then have ∗: ∀F z in at ξ. dist (g z) (g ξ) < 1 ⟶ cmod (f z) ≤ cmod (g ξ)
+ 1
      using ‹0 < δ› eventually_at by blast
    have continuous_on (interior S) g
     by (meson continuous_on_subset holg holomorphic_on_imp_continuous_on
interior_subset)
    then have ⋀x. x ∈ interior S ⟹ (g ⟶⟶ g x) (at x)
     using continuous_on_interior continuous_within holg holomorphic_on_imp_continuous_on
by blast
    then have (g ⟶⟶ g ξ) (at ξ)
      by (simp add: ξ)
    then show ?thesis
      apply (rule_tac x=norm(g ξ) + 1 in exI)
      apply (rule eventually_mp [OF ∗ tendstoD [where e=1]], auto)
      done
  qed
  moreover have ?Q if ∀F z in at ξ. cmod (f z) ≤ B for B
    by (rule lim_null_mult_right_bounded [OF _ that]) (simp add: LIM_zero)
  moreover have ?P if (λz. (z − ξ) ∗ f z) −ξ→ 0
  proof −
    define h where [abs_def]: h z = (z − ξ)^2 ∗ f z for z
    have h0: (h has_field_derivative 0) (at ξ)
      apply (simp add: h_def has_field_derivative_iff)
     apply (auto simp: field_split_simps power2_eq_square Lim_transform_within
[OF that, of 1])
      done
    have holh: h holomorphic_on S
    proof (simp add: holomorphic_on_def, clarify)
      fix z assume z ∈ S
      show h field_differentiable at z within S
      proof (cases z = ξ)
        case True then show ?thesis
          using field_differentiable_at_within field_differentiable_def h0 by blast
      next
        case False
        then have f field_differentiable at z within S
          using holomorphic_onD [OF holf, of z] ‹z ∈ S›
          unfolding field_differentiable_def has_field_derivative_iff
          by (force intro: exI [where x=dist ξ z] elim: Lim_transform_within_set
[unfolded eventually_at])
        then show ?thesis
```

```
        by (simp add: h_def power2_eq_square derivative_intros)
      qed
    qed
    define g where [abs_def]: g z = (if z = ξ then deriv h ξ else (h z − h ξ) / (z
− ξ)) for z
    have holg: g holomorphic_on S
      unfolding g_def by (rule pole_lemma [OF holh ξ])
    have §: ∀z∈S − {ξ}. (g z − g ξ) / (z − ξ) = f z
     using h0 by (auto simp: g_def power2_eq_square divide_simps DERIV_imp_deriv
h_def)
    show ?thesis
      apply (intro exI conjI)
       apply (rule pole_lemma [OF holg ξ])
      apply (simp add: §)
      done
  qed
  ultimately show ?P = ?Q and ?P = ?R
    by meson+
qed

lemma pole_at_infinity:
 assumes holf: f holomorphic_on UNIV and lim: ((inverse o f) ⟶ l) at_infinity
  obtains a n where ⋀z. f z = (∑ i≤n. a i * z^i)
proof (cases l = 0)
  case False
  show thesis
  proof
    show f z = (∑ i≤0. inverse l * z ^ i) for z
      using False tendsto_inverse [OF lim] by (simp add: Liouville_weak [OF
holf])
  qed
next
  case True
  then have [simp]: l = 0 .
  show ?thesis
  proof (cases ∃r. 0 < r ∧ (∀z ∈ ball 0 r − {0}. f(inverse z) ≠ 0))
    case True
      then obtain r where 0 < r and r: ⋀z. z ∈ ball 0 r − {0} ⟹ f(inverse
z) ≠ 0
          by auto
     have 1: inverse ∘ f ∘ inverse holomorphic_on ball 0 r − {0}
     by (rule holomorphic_on_compose holomorphic_intros holomorphic_on_subset
[OF holf] | force simp: r)+
     have 2: 0 ∈ interior (ball 0 r)
       using ‹0 < r› by simp
     have ∃B. 0<B ∧ eventually (λz. cmod ((inverse ∘ f ∘ inverse) z) ≤ B) (at
0)
       apply (rule exI [where x=1])
       using tendstoD [OF lim [unfolded lim_at_infinity_0] zero_less_one]
```

```
    by (simp add: eventually_mono)
  with holomorphic_on_extend_bounded [OF 1 2]
  obtain g where holg: g holomorphic_on ball 0 r
          and geq: ⋀z. z ∈ ball 0 r − {0} ⟹ g z = (inverse ∘ f ∘ inverse) z
    by meson
  have ifi0: (inverse ∘ f ∘ inverse) −0→ 0
    using ‹l = 0› lim lim_at_infinity_0 by blast
  have g2g0: g −0→ g 0
  using ‹0 < r› centre_in_ball continuous_at continuous_on_eq_continuous_at
holg
    by (blast intro: holomorphic_on_imp_continuous_on)
  have g2g1: g −0→ 0
    apply (rule Lim_transform_within_open [OF ifi0 open_ball [of 0 r]])
    using ‹0 < r› by (auto simp: geq)
  have [simp]: g 0 = 0
    by (rule tendsto_unique [OF _ g2g0 g2g1]) simp
  have ball 0 r − {0::complex} ≠ {}
  using ‹0 < r› by (metis 2 Diff_iff insert_Diff interior_ball interior_singleton)
  then obtain w::complex where w ≠ 0 and w: norm w < r by force
  then have g w ≠ 0 by (simp add: geq r)
  obtain B n e where 0 < B 0 < e e ≤ r
                and leg: ⋀w. norm w < e ⟹ B * cmod w ^ n ≤ cmod (g w)
   proof (rule holomorphic_lower_bound_difference [OF holg open_ball con-
nected_ball])
    show g w ≠ g 0
      by (simp add: ‹g w ≠ 0›)
    show w ∈ ball 0 r
      using mem_ball_0 w by blast
  qed (use ‹0 < r› in ‹auto simp: ball_subset_ball_iff›)
  have §: cmod (f z) ≤ cmod z ^ n / B if 2/e ≤ cmod z for z
  proof −
    have ize: inverse z ∈ ball 0 e − {0} using that ‹0 < e›
      by (auto simp: norm_divide field_split_simps algebra_simps)
    then have [simp]: z ≠ 0 and izr: inverse z ∈ ball 0 r − {0} using ‹e ≤
r›
      by auto
    then have [simp]: f z ≠ 0
      using r [of inverse z] by simp
    have [simp]: f z = inverse (g (inverse z))
      using izr geq [of inverse z] by simp
    show ?thesis using ize leg [of inverse z] ‹0 < B› ‹0 < e›
      by (simp add: field_split_simps norm_divide algebra_simps)
  qed
  show thesis
  proof
    show f z = (∑ i≤n. (deriv ^^ i) f 0 / fact i * z ^ i) for z
      using § by (rule_tac A = 2/e and B = 1/B in Liouville_polynomial
[OF holf], simp)
  qed
```

```
  next
    case False
    then have fi0: ⋀r. r > 0 ⟹ ∃z∈ball 0 r − {0}. f (inverse z) = 0
      by simp
    have fz0: f z = 0 if 0 < r and lt1: ⋀x. x ≠ 0 ⟹ cmod x < r ⟹ inverse
(cmod (f (inverse x))) < 1
              for z r
    proof −
      have f0: (f ⟶ 0) at_infinity
      proof −
        have DIM_complex[intro]: 2 ≤ DIM(complex)  — should not be necessary!
          by simp
       have f (inverse x) ≠ 0 ⟹ x ≠ 0 ⟹ cmod x < r ⟹ 1 < cmod (f (inverse
x)) for x
          using lt1[of x] by (auto simp: field_simps)
        then have ∗∗: cmod (f (inverse x)) ≤ 1 ⟹ x ≠ 0 ⟹ cmod x < r ⟹ f
(inverse x) = 0 for x
          by force
        then have ∗: (f ∘ inverse) ‘ (ball 0 r − {0}) ⊆ {0} ∪ − ball 0 1
          by force
        have continuous_on (inverse ‘ (ball 0 r − {0})) f
          using continuous_on_subset holf holomorphic_on_imp_continuous_on
by blast
        then have connected ((f ∘ inverse) ‘ (ball 0 r − {0}))
          using connected_punctured_ball
          by (intro connected_continuous_image continuous_intros; force)
        then have {0} ∩ (f ∘ inverse) ‘ (ball 0 r − {0}) = {} ∨ − ball 0 1 ∩ (f ∘
inverse) ‘ (ball 0 r − {0}) = {}
          by (rule connected_closedD) (use ∗ in auto)
        then have f (inverse w) = 0 if w ≠ 0 cmod w < r for w
          using ∗∗[of w] fi0 ‹0 < r› that by force
        then show ?thesis
          unfolding lim_at_infinity_0
        using eventually_at ‹r > 0› by (force simp add: intro: tendsto_eventually)
      qed
      obtain w where w ∈ ball 0 r − {0} and f (inverse w) = 0
        using False ‹0 < r› by blast
      then show ?thesis
        by (auto simp: f0 Liouville_weak [OF holf, of 0])
    qed
    show thesis
    proof
      show ⋀z. f z = (∑ i≤0. 0 ∗ z ^ i)
        using lim
        apply (simp add: lim_at_infinity_0 Lim_at dist_norm norm_inverse)
        using fz0 zero_less_one by blast
    qed
  qed
qed
```

5.5 Entire proper functions are precisely the non-trivial polynomials

lemma *proper_map_polyfun*:
  fixes $c :: nat \Rightarrow 'a::\{real\_normed\_div\_algebra,heine\_borel\}$
  assumes *closed S* **and** *compact K* **and** *c*: $c\ i \neq 0\ 1 \leq i\ i \leq n$
  shows *compact* $(S \cap \{z.\ (\sum i{\leq}n.\ c\ i * z\hat{}i) \in K\})$
proof −
  obtain *B* **where** $B > 0$ **and** *B*: $\bigwedge x.\ x \in K \Longrightarrow norm\ x \leq B$
    by (*metis compact_imp_bounded* ‹*compact K*› *bounded_pos*)
  have ∗: $norm\ x \leq b$
      if $\bigwedge x.\ b \leq norm\ x \Longrightarrow B + 1 \leq norm\ (\sum i{\leq}n.\ c\ i * x\ \hat{}\ i)$
        $(\sum i{\leq}n.\ c\ i * x\ \hat{}\ i) \in K$ **for** *b x*
  proof −
    have $norm\ (\sum i{\leq}n.\ c\ i * x\ \hat{}\ i) \leq B$
      using *B that* **by** *blast*
    moreover have $\neg\ B + 1 \leq B$
      by *simp*
    ultimately show $norm\ x \leq b$
      using *that* **by** (*metis* (*no_types*) *less_eq_real_def not_less order_trans*)
  qed
  have *bounded* $\{z.\ (\sum i{\leq}n.\ c\ i * z\ \hat{}\ i) \in K\}$
    using *Limits.polyfun_extremal* [**where** $c{=}c$ **and** $B{=}B{+}1$, *OF c*]
    by (*auto simp*: *bounded_pos eventually_at_infinity_pos* ∗)
  moreover have *closed* $((\lambda z.\ (\sum i{\leq}n.\ c\ i * z\ \hat{}\ i)) -` K)$
    using ‹*compact K*› *compact_eq_bounded_closed*
    by (*intro allI continuous_closed_vimage continuous_intros*; *force*)
  ultimately show *?thesis*
    using *closed_Int_compact* [*OF* ‹*closed S*›] *compact_eq_bounded_closed*
    by (*auto simp add*: *vimage_def*)
qed

lemma *proper_map_polyfun_univ*:
  fixes $c :: nat \Rightarrow 'a::\{real\_normed\_div\_algebra,heine\_borel\}$
  assumes *compact K* $c\ i \neq 0\ 1 \leq i\ i \leq n$
  shows *compact* $(\{z.\ (\sum i{\leq}n.\ c\ i * z\hat{}i) \in K\})$
  using *proper_map_polyfun* [*of UNIV K c i n*] *assms* **by** *simp*

lemma *proper_map_polyfun_eq*:
  assumes *f holomorphic_on UNIV*
    shows $(\forall k.\ compact\ k \longrightarrow compact\ \{z.\ f\ z \in k\}) \longleftrightarrow$
        $(\exists c\ n.\ 0 < n \wedge (c\ n \neq 0) \wedge f = (\lambda z.\ \sum i{\leq}n.\ c\ i * z\hat{}i))$
       (**is** *?lhs* = *?rhs*)
proof
  assume *compf* [*rule_format*]: *?lhs*
  have *2*: $\exists k.\ 0 < k \wedge a\ k \neq 0 \wedge f = (\lambda z.\ \sum i \leq k.\ a\ i * z\ \hat{}\ i)$
      if $\bigwedge z.\ f\ z = (\sum i{\leq}n.\ a\ i * z\ \hat{}\ i)$ **for** *a n*
  proof (*cases* $\forall i{\leq}n.\ 0{<}i \longrightarrow a\ i = 0$)
    case *True*
    then have [*simp*]: $\bigwedge z.\ f\ z = a\ 0$

```
    by (simp add: that sum.atMost_shift)
   have False using compf [of {a 0}] by simp
   then show ?thesis ..
  next
   case False
   then obtain k where k: 0 < k k≤n a k ≠ 0 by force
   define m where m = (GREATEST k. k≤n ∧ a k ≠ 0)
   have m: m≤n ∧ a m ≠ 0
    unfolding m_def
    using GreatestI_nat [where b = n] k by (metis (mono_tags, lifting))
   have [simp]: a i = 0 if m < i i ≤ n for i
    using Greatest_le_nat [where b = n and P = λk. k≤n ∧ a k ≠ 0]
    using m_def not_le that by auto
   have k ≤ m
    unfolding m_def
    using Greatest_le_nat [where b = n] k by (metis (mono_tags, lifting))
   with k m show ?thesis
   by (rule_tac x=m in exI) (auto simp: that comm_monoid_add_class.sum.mono_neutral_right)
  qed
  have §: ((inverse ∘ f) ⟶ 0) at_infinity
  proof (rule Lim_at_infinityI)
   fix e::real assume 0 < e
   with compf [of cball 0 (inverse e)]
   show ∃ B. ∀ x. B ≤ cmod x ⟶ dist ((inverse ∘ f) x) 0 ≤ e
    apply simp
   apply (clarsimp simp add: compact_eq_bounded_closed bounded_pos norm_inverse)
   by (metis (no_types, opaque_lifting) inverse_inverse_eq le_less_trans less_eq_real_def
less_imp_inverse_less linordered_field_no_ub not_less)
  qed
  then obtain a n where ⋀z. f z = (∑ i≤n. a i * z^i)
   using assms pole_at_infinity by blast
  with § 2 show ?rhs by blast
next
  assume ?rhs
  then obtain c n where 0 < n c n ≠ 0 f = (λz. ∑ i≤n. c i * z ^ i) by blast
  then have compact {z. f z ∈ k} if compact k for k
   by (auto intro: proper_map_polyfun_univ [OF that])
  then show ?lhs by blast
qed
```

5.6 Relating invertibility and nonvanishing of derivative

```
lemma has_complex_derivative_locally_injective:
  assumes holf: f holomorphic_on S
      and S: ξ ∈ S open S
      and dnz: deriv f ξ ≠ 0
  obtains r where r > 0 ball ξ r ⊆ S inj_on f (ball ξ r)
proof −
  have ∗: ∃ d>0. ∀ x. dist ξ x < d ⟶ onorm (λv. deriv f x * v − deriv f ξ * v)
```

```
< e if e > 0 for e
  proof −
    have contdf: continuous_on S (deriv f)
      by (simp add: holf holomorphic_deriv holomorphic_on_imp_continuous_on
‹open S›)
    obtain δ where δ>0 and δ: ⋀x. ⟦x ∈ S; dist x ξ ≤ δ⟧ ⟹ cmod (deriv f x −
deriv f ξ) ≤ e/2
      using continuous_onE [OF contdf ‹ξ ∈ S›, of e/2] ‹0 < e›
      by (metis dist_complex_def half_gt_zero less_imp_le)
    have §: ⋀ζ z. ⟦ζ ∈ S; dist ξ ζ < δ⟧ ⟹ cmod (deriv f ζ − deriv f ξ) ∗ cmod
z ≤ e/2 ∗ cmod z
      by (intro mult_right_mono [OF δ]) (auto simp: dist_commute)
    obtain ε where ε>0 ball ξ ε ⊆ S
      by (metis openE [OF ‹open S› ‹ξ ∈ S›])
   with ‹δ>0› have ∃δ>0. ∀x. dist ξ x < δ ⟶ onorm (λv. deriv f x ∗ v − deriv
f ξ ∗ v) ≤ e/2
      using §
      apply (rule_tac x=min δ ε in exI)
      apply (intro conjI allI impI Operator_Norm.onorm_le)
     apply (force simp: mult_right_mono norm_mult [symmetric] left_diff_distrib
δ)+
      done
    with ‹e>0› show ?thesis by force
  qed
  have inj ((∗) (deriv f ξ))
    using dnz by simp
  then obtain g′ where g′: linear g′ g′ ∘ (∗) (deriv f ξ) = id
    using linear_injective_left_inverse [of (∗) (deriv f ξ)] by auto
  show ?thesis
    apply (rule has_derivative_locally_injective [OF S, where f=f and f′ = λz
h. deriv f z ∗ h and g′ = g′])
    using g′ ∗
    apply (simp_all add: linear_conv_bounded_linear that)
   using ‹open S› has_field_derivative_imp_has_derivative holf holomorphic_derivI
by blast
qed

lemma has_complex_derivative_locally_invertible:
  assumes holf: f holomorphic_on S
      and S: ξ ∈ S open S
      and dnz: deriv f ξ ≠ 0
  obtains r where r > 0 ball ξ r ⊆ S open (f ‘ (ball ξ r)) inj_on f (ball ξ r)
proof −
  obtain r where r > 0 ball ξ r ⊆ S inj_on f (ball ξ r)
    by (blast intro: that has_complex_derivative_locally_injective [OF assms])
  then have ξ: ξ ∈ ball ξ r by simp
  then have nc: ¬ f constant_on ball ξ r
    using ‹inj_on f (ball ξ r)› injective_not_constant by fastforce
  have holf′: f holomorphic_on ball ξ r
```

```
    using ‹ball ξ r ⊆ S› holf holomorphic_on_subset by blast
  have open (f ‘ ball ξ r)
    by (simp add: ‹inj_on f (ball ξ r)› holf′ open_mapping_thm3)
  then show ?thesis
    using ‹0 < r› ‹ball ξ r ⊆ S› ‹inj_on f (ball ξ r)› that  by blast
qed

lemma holomorphic_injective_imp_regular:
  assumes holf: f holomorphic_on S
      and open S and injf: inj_on f S
      and ξ ∈ S
    shows deriv f ξ ≠ 0
proof −
  obtain r where r>0 and r: ball ξ r ⊆ S using assms by (blast elim!: openE)
  have holf′: f holomorphic_on ball ξ r
    using ‹ball ξ r ⊆ S› holf holomorphic_on_subset by blast
  show ?thesis
  proof (cases ∀ n>0. (deriv ^^ n) f ξ = 0)
    case True
    have fcon: f w = f ξ if w ∈ ball ξ r for w
      by (meson open_ball True ‹0 < r› centre_in_ball connected_ball holf′
            holomorphic_fun_eq_const_on_connected that)
    have False
      using fcon [of ξ + r/2] ‹0 < r› r injf unfolding inj_on_def
     by (metis ‹ξ ∈ S› contra_subsetD dist_commute fcon mem_ball perfect_choose_dist)
    then show ?thesis ..
  next
    case False
    then obtain n0 where n0: n0 > 0 ∧ (deriv ^^ n0) f ξ ≠ 0 by blast
    define n where [abs_def]: n = (LEAST n. n > 0 ∧ (deriv ^^ n) f ξ ≠ 0)
    have n_ne: n > 0 (deriv ^^ n) f ξ ≠ 0
      using def_LeastI [OF n_def n0] by auto
    have n_min: ⋀k. 0 < k ⟹ k < n ⟹ (deriv ^^ k) f ξ = 0
      using def_Least_le [OF n_def] not_le by auto
    obtain g δ where 0 < δ
          and holg: g holomorphic_on ball ξ δ
          and fd: ⋀w. w ∈ ball ξ δ ⟹ f w − f ξ = ((w − ξ) ∗ g w) ^ n
          and gnz: ⋀w. w ∈ ball ξ δ ⟹ g w ≠ 0
      by (blast intro: n_min holomorphic_factor_order_of_zero_strong [OF holf
‹open S› ‹ξ ∈ S› n_ne])
    show ?thesis
    proof (cases n=1)
      case True
      with n_ne show ?thesis by auto
    next
      case False
      have g holomorphic_on ball ξ (min r δ)
        using holg by (simp add: holomorphic_on_subset subset_ball)
      then have holgw: (λw. (w − ξ) ∗ g w) holomorphic_on ball ξ (min r δ)
```

```
    by (intro holomorphic_intros)
  have gd: ⋀w. dist ξ w < δ ⟹ (g has_field_derivative deriv g w) (at w)
    using holg
    by (simp add: DERIV_deriv_iff_field_differentiable holomorphic_on_def
at_within_open_NO_MATCH)
  have ∗: ⋀w. w ∈ ball ξ (min r δ)
      ⟹ ((λw. (w − ξ) ∗ g w) has_field_derivative ((w − ξ) ∗ deriv g w + g
w))
          (at w)
    by (rule gd derivative_eq_intros | simp)+
  have [simp]: deriv (λw. (w − ξ) ∗ g w) ξ ≠ 0
    using ∗ [of ξ] ‹0 < δ› ‹0 < r› by (simp add: DERIV_imp_deriv gnz)
  obtain T where ξ ∈ T open T and Tsb: T ⊆ ball ξ (min r δ) and oimT:
open ((λw. (w − ξ) ∗ g w) ‘ T)
    using ‹0 < r› ‹0 < δ› has_complex_derivative_locally_invertible [OF
holgw, of ξ]
    by force
  define U where U = (λw. (w − ξ) ∗ g w) ‘ T
  have open U by (metis oimT U_def)
  moreover have 0 ∈ U
    using ‹ξ ∈ T› by (auto simp: U_def intro: image_eqI [where x = ξ])
  ultimately obtain ε where ε>0 and ε: cball 0 ε ⊆ U
    using ‹open U› open_contains_cball by blast
  then have ε ∗ exp(2 ∗ of_real pi ∗ i ∗ (0/n)) ∈ cball 0 ε
        ε ∗ exp(2 ∗ of_real pi ∗ i ∗ (1/n)) ∈ cball 0 ε
    by (auto simp: norm_mult)
  with ε have ε ∗ exp(2 ∗ of_real pi ∗ i ∗ (0/n)) ∈ U
          ε ∗ exp(2 ∗ of_real pi ∗ i ∗ (1/n)) ∈ U by blast+
  then obtain y0 y1 where y0 ∈ T and y0: (y0 − ξ) ∗ g y0 = ε ∗ exp(2 ∗
of_real pi ∗ i ∗ (0/n))
                    and y1 ∈ T and y1: (y1 − ξ) ∗ g y1 = ε ∗ exp(2 ∗ of_real
pi ∗ i ∗ (1/n))
    by (auto simp: U_def)
  then have y0 ∈ ball ξ δ y1 ∈ ball ξ δ using Tsb by auto
  then have f y0 − f ξ = ((y0 − ξ) ∗ g y0) ^ n f y1 − f ξ = ((y1 − ξ) ∗ g
y1) ^ n
    using fd by blast+
  moreover have y0 ≠ y1
    using y0 y1 ‹ε > 0› complex_root_unity_eq_1 [of n 1] ‹n > 0› False by
auto
  moreover have T ⊆ S
    by (meson Tsb min.cobounded1 order_trans r subset_ball)
  ultimately have False
    using inj_onD [OF injf, of y0 y1] ‹y0 ∈ T› ‹y1 ∈ T›
    using complex_root_unity [of n 1]
    apply (simp add: y0 y1 power_mult_distrib)
    apply (force simp: algebra_simps)
    done
  then show ?thesis ..
```

```
    qed
  qed
qed
```

Hence a nice clean inverse function theorem

```
lemma has_field_derivative_inverse_strong:
  fixes f :: 'a::{euclidean_space,real_normed_field} ⇒ 'a
  shows ⟦DERIV f x :> f'; f' ≠ 0; open S; x ∈ S; continuous_on S f;
         ⋀z. z ∈ S ⟹ g (f z) = z⟧
         ⟹ DERIV g (f x) :> inverse (f')
  unfolding has_field_derivative_def
  by (rule has_derivative_inverse_strong [of S x f g]) auto

lemma has_field_derivative_inverse_strong_x:
  fixes f :: 'a::{euclidean_space,real_normed_field} ⇒ 'a
  shows ⟦DERIV f (g y) :> f'; f' ≠ 0; open S; continuous_on S f; g y ∈ S; f(g
y) = y;
         ⋀z. z ∈ S ⟹ g (f z) = z⟧
         ⟹ DERIV g y :> inverse (f')
  unfolding has_field_derivative_def
  by (rule has_derivative_inverse_strong_x [of S g y f]) auto

proposition holomorphic_has_inverse:
  assumes holf: f holomorphic_on S
      and open S and injf: inj_on f S
  obtains g where g holomorphic_on (f ‘ S)
                  ⋀z. z ∈ S ⟹ deriv f z * deriv g (f z) = 1
                  ⋀z. z ∈ S ⟹ g(f z) = z
proof −
  have ofs: open (f ‘ S)
    by (rule open_mapping_thm3 [OF assms])
  have contf: continuous_on S f
    by (simp add: holf holomorphic_on_imp_continuous_on)
  have *: (the_inv_into S f has_field_derivative inverse (deriv f z)) (at (f z)) if
z ∈ S for z
  proof −
    have 1: (f has_field_derivative deriv f z) (at z)
      using DERIV_deriv_iff_field_differentiable ‹z ∈ S› ‹open S› holf holomor-
phic_on_imp_differentiable_at
      by blast
    have 2: deriv f z ≠ 0
     using ‹z ∈ S› ‹open S› holf holomorphic_injective_imp_regular injf by blast
    show ?thesis
    proof (rule has_field_derivative_inverse_strong [OF 1 2 ‹open S› ‹z ∈ S›])
      show continuous_on S f
        by (simp add: holf holomorphic_on_imp_continuous_on)
      show ⋀z. z ∈ S ⟹ the_inv_into S f (f z) = z
        by (simp add: injf the_inv_into_f_f)
    qed
```

```
  qed
  show ?thesis
    proof
      show the_inv_into S f holomorphic_on f ' S
        by (simp add: holomorphic_on_open ofs) (blast intro: *)
    next
      fix z assume z ∈ S
      have deriv f z ≠ 0
        using ‹z ∈ S› ‹open S› holf holomorphic_injective_imp_regular injf by
blast
      then show deriv f z * deriv (the_inv_into S f) (f z) = 1
        using * [OF ‹z ∈ S›]  by (simp add: DERIV_imp_deriv)
    next
      fix z assume z ∈ S
      show the_inv_into S f (f z) = z
        by (simp add: ‹z ∈ S› injf the_inv_into_f_f)
  qed
qed
```

5.7 The Schwarz Lemma

```
lemma Schwarz1:
  assumes holf: f holomorphic_on S
      and contf: continuous_on (closure S) f
      and S: open S connected S
      and boS: bounded S
      and S ≠ {}
  obtains w where w ∈ frontier S
       ⋀z. z ∈ closure S ⟹ norm (f z) ≤ norm (f w)
proof −
  have connf: continuous_on (closure S) (norm o f)
    using contf continuous_on_compose continuous_on_norm_id by blast
  have coc: compact (closure S)
    by (simp add: ‹bounded S› bounded_closure compact_eq_bounded_closed)
 then obtain x where x: x ∈ closure S and xmax: ⋀z. z ∈ closure S ⟹ norm(f
z) ≤ norm(f x)
    using continuous_attains_sup [OF _ _ connf] ‹S ≠ {}› by auto
  then show ?thesis
  proof (cases x ∈ frontier S)
    case True
    then show ?thesis using that xmax by blast
  next
    case False
    then have x ∈ S
      using ‹open S› frontier_def interior_eq x by auto
    then have f constant_on S
    proof (rule maximum_modulus_principle [OF holf S ‹open S› order_refl])
      show ⋀z. z ∈ S ⟹ cmod (f z) ≤ cmod (f x)
        using closure_subset by (blast intro: xmax)
```

```
    qed
    then have f constant_on (closure S)
      by (rule constant_on_closureI [OF _ contf])
    then obtain c where c: ⋀x. x ∈ closure S ⟹ f x = c
      by (meson constant_on_def)
    obtain w where w ∈ frontier S
      by (metis coc all_not_in_conv assms(6) closure_UNIV frontier_eq_empty
not_compact_UNIV)
    then show ?thesis
      by (simp add: c frontier_def that)
  qed
qed

lemma Schwarz2:
 ⟦f holomorphic_on ball 0 r;
    0 < s; ball w s ⊆ ball 0 r;
    ⋀z. norm (w−z) < s ⟹ norm(f z) ≤ norm(f w)⟧
    ⟹ f constant_on ball 0 r
by (rule maximum_modulus_principle [where U = ball w s and ξ = w]) (simp_all
add: dist_norm)

lemma Schwarz3:
  assumes holf: f holomorphic_on (ball 0 r) and [simp]: f 0 = 0
  obtains h where h holomorphic_on (ball 0 r) and ⋀z. norm z < r ⟹ f z =
z * (h z) and deriv f 0 = h 0
proof −
  define h where h z = (if z = 0 then deriv f 0 else f z / z) for z
  have d0: deriv f 0 = h 0
    by (simp add: h_def)
  moreover have h holomorphic_on (ball 0 r)
    by (rule pole_theorem_open_0 [OF holf, of 0]) (auto simp: h_def)
  moreover have norm z < r ⟹ f z = z * h z for z
    by (simp add: h_def)
  ultimately show ?thesis
    using that by blast
qed

proposition Schwarz_Lemma:
  assumes holf: f holomorphic_on (ball 0 1) and [simp]: f 0 = 0
     and no: ⋀z. norm z < 1 ⟹ norm (f z) < 1
     and ξ: norm ξ < 1
    shows norm (f ξ) ≤ norm ξ and norm(deriv f 0) ≤ 1
     and ((∃ z. norm z < 1 ∧ z ≠ 0 ∧ norm(f z) = norm z)
          ∨ norm(deriv f 0) = 1)
         ⟹ ∃ α. (∀ z. norm z < 1 ⟶ f z = α * z) ∧ norm α = 1
     (is ?P ⟹ ?Q)
proof −
  obtain h where holh: h holomorphic_on (ball 0 1)
           and fz_eq: ⋀z. norm z < 1 ⟹ f z = z * (h z) and df0: deriv f 0 = h
```

```
0
    by (rule Schwarz3 [OF holf]) auto
  have noh_le: norm (h z) ≤ 1 if z: norm z < 1 for z
  proof −
    have norm (h z) < a if a: 1 < a for a
    proof −
      have max (inverse a) (norm z) < 1
        using z a by (simp_all add: inverse_less_1_iff)
      then obtain r where r: max (inverse a) (norm z) < r and r < 1
        using Rats_dense_in_real by blast
      then have nzr: norm z < r and ira: inverse r < a
        using z a less_imp_inverse_less by force+
      then have 0 < r
        by (meson norm_not_less_zero not_le order.strict_trans2)
      have holh': h holomorphic_on ball 0 r
         by (meson holh ‹r < 1› holomorphic_on_subset less_eq_real_def sub-
set_ball)
      have conth': continuous_on (cball 0 r) h
      by (meson ‹r < 1› dual_order.trans holh holomorphic_on_imp_continuous_on
holomorphic_on_subset mem_ball_0 mem_cball_0 not_less subsetI)
      obtain w where w: norm w = r and lenw: ⋀z. norm z < r ⟹ norm(h z)
≤ norm(h w)
        apply (rule Schwarz1 [OF holh']) using conth' ‹0 < r› by auto
      have h w = f w / w using fz_eq ‹r < 1› nzr w by auto
      then have cmod (h z) < inverse r
        by (metis ‹0 < r› ‹r < 1› divide_strict_right_mono inverse_eq_divide
              le_less_trans lenw no norm_divide nzr w)
      then show ?thesis using ira by linarith
    qed
    then show norm (h z) ≤ 1
      using not_le by blast
  qed
  show cmod (f ξ) ≤ cmod ξ
  proof (cases ξ = 0)
    case True then show ?thesis by auto
  next
    case False
    then show ?thesis
      by (simp add: noh_le fz_eq ξ mult_left_le norm_mult)
  qed
  show no_df0: norm(deriv f 0) ≤ 1
    by (simp add: ‹⋀z. cmod z < 1 ⟹ cmod (h z) ≤ 1› df0)
  show ?Q if ?P
    using that
  proof
    assume ∃ z. cmod z < 1 ∧ z ≠ 0 ∧ cmod (f z) = cmod z
    then obtain γ where γ: cmod γ < 1 γ ≠ 0 cmod (f γ) = cmod γ by blast
    then have [simp]: norm (h γ) = 1
      by (simp add: fz_eq norm_mult)
```

```
    have §: ball γ (1 − cmod γ) ⊆ ball 0 1
      by (simp add: ball_subset_ball_iff)
    moreover have ⋀z. cmod (γ − z) < 1 − cmod γ ⟹ cmod (h z) ≤ cmod (h
γ)
       by (metis ‹cmod (h γ) = 1› § dist_0_norm dist_complex_def in_mono
mem_ball noh_le)
    ultimately obtain c where c: ⋀z. norm z < 1 ⟹ h z = c
      using Schwarz2 [OF holh, of 1 − norm γ γ, unfolded constant_on_def] γ
by auto
    then have norm c = 1
      using γ by force
    with c show ?thesis
      using fz_eq by auto
  next
    assume [simp]: cmod (deriv f 0) = 1
    then obtain c where c: ⋀z. norm z < 1 ⟹ h z = c
      using Schwarz2 [OF holh zero_less_one, of 0, unfolded constant_on_def]
df0 noh_le
      by auto
    moreover have norm c = 1  using df0 c by auto
    ultimately show ?thesis
      using fz_eq by auto
  qed
qed

corollary Schwarz_Lemma':
  assumes holf: f holomorphic_on (ball 0 1) and [simp]: f 0 = 0
      and no: ⋀z. norm z < 1 ⟹ norm (f z) < 1
    shows ((∀ξ. norm ξ < 1 ⟶ norm (f ξ) ≤ norm ξ)
           ∧ norm(deriv f 0) ≤ 1)
           ∧ (((∃z. norm z < 1 ∧ z ≠ 0 ∧ norm(f z) = norm z)
             ∨ norm(deriv f 0) = 1)
             ⟶ (∃α. (∀z. norm z < 1 ⟶ f z = α * z) ∧ norm α = 1))
  using Schwarz_Lemma [OF assms]
  by (metis (no_types) norm_eq_zero zero_less_one)
```

5.8 The Schwarz reflection principle

```
lemma hol_pal_lem0:
  assumes d · a ≤ k k ≤ d · b
  obtains c where
     c ∈ closed_segment a b d · c = k
     ⋀z. z ∈ closed_segment a c ⟹ d · z ≤ k
     ⋀z. z ∈ closed_segment c b ⟹ k ≤ d · z
proof −
  obtain c where cin: c ∈ closed_segment a b and keq: k = d · c
    using connected_ivt_hyperplane [of closed_segment a b a b d k]
    by (auto simp: assms)
  have closed_segment a c ⊆ {z. d · z ≤ k}  closed_segment c b ⊆ {z. k ≤ d · z}
```

```
    unfolding segment_convex_hull using assms keq
    by (auto simp: convex_halfspace_le convex_halfspace_ge hull_minimal)
  then show ?thesis using cin that by fastforce
qed

lemma hol_pal_lem1:
  assumes convex S open S
      and abc: a ∈ S b ∈ S c ∈ S
          d ≠ 0 and lek: d · a ≤ k d · b ≤ k d · c ≤ k
      and holf1: f holomorphic_on {z. z ∈ S ∧ d · z < k}
      and contf: continuous_on S f
    shows contour_integral (linepath a b) f +
          contour_integral (linepath b c) f +
          contour_integral (linepath c a) f = 0
proof −
  have interior (convex hull {a, b, c}) ⊆ interior(S ∩ {x. d · x ≤ k})
  proof (intro interior_mono hull_minimal)
    show {a, b, c} ⊆ S ∩ {x. d · x ≤ k}
      by (simp add: abc lek)
    show convex (S ∩ {x. d · x ≤ k})
      by (rule convex_Int [OF ‹convex S› convex_halfspace_le])
  qed
  also have ... ⊆ {z ∈ S. d · z < k}
    by (force simp: interior_open [OF ‹open S›] ‹d ≠ 0›)
  finally have ∗: interior (convex hull {a, b, c}) ⊆ {z ∈ S. d · z < k} .
  have continuous_on (convex hull {a,b,c}) f
    using ‹convex S› contf abc continuous_on_subset subset_hull
    by fastforce
  moreover have f holomorphic_on interior (convex hull {a,b,c})
    by (rule holomorphic_on_subset [OF holf1 ∗])
  ultimately show ?thesis
   using Cauchy_theorem_triangle_interior has_chain_integral_chain_integral3
     by blast
qed

lemma hol_pal_lem2:
  assumes S: convex S open S
      and abc: a ∈ S b ∈ S c ∈ S
      and d ≠ 0 and lek: d · a ≤ k d · b ≤ k
      and holf1: f holomorphic_on {z. z ∈ S ∧ d · z < k}
      and holf2: f holomorphic_on {z. z ∈ S ∧ k < d · z}
      and contf: continuous_on S f
    shows contour_integral (linepath a b) f +
          contour_integral (linepath b c) f +
          contour_integral (linepath c a) f = 0
proof (cases d · c ≤ k)
  case True show ?thesis
    by (rule hol_pal_lem1 [OF S abc ‹d ≠ 0› lek True holf1 contf])
next
```

case *False*
then have $d \cdot c > k$ **by** *force*
obtain a' **where** a': $a' \in$ *closed_segment* b c **and** $d \cdot a' = k$
and ba': $\bigwedge z.\ z \in$ *closed_segment* b $a' \Longrightarrow d \cdot z \leq k$
and $a'c$: $\bigwedge z.\ z \in$ *closed_segment* a' $c \Longrightarrow k \leq d \cdot z$
using *False hol_pal_lem0* [*of* d b k c, *OF* ‹$d \cdot b \leq k$›] **by** *auto*
obtain b' **where** b': $b' \in$ *closed_segment* a c **and** $d \cdot b' = k$
and ab': $\bigwedge z.\ z \in$ *closed_segment* a $b' \Longrightarrow d \cdot z \leq k$
and $b'c$: $\bigwedge z.\ z \in$ *closed_segment* b' $c \Longrightarrow k \leq d \cdot z$
using *False hol_pal_lem0* [*of* d a k c, *OF* ‹$d \cdot a \leq k$›] **by** *auto*
have $a'b'$: $a' \in S \wedge b' \in S$
using a' *abc* b' *convex_contains_segment* ‹*convex* S›) **by** *auto*
have *continuous_on* (*closed_segment* c a) f
by (*meson abc contf continuous_on_subset convex_contains_segment* ‹*convex* S›)
then have *1*: *contour_integral* (*linepath* c a) f =
contour_integral (*linepath* c b') f + *contour_integral* (*linepath* b' a) f
using b' *closed_segment_commute contour_integral_split_linepath* **by** *blast*
have *continuous_on* (*closed_segment* b c) f
by (*meson abc contf continuous_on_subset convex_contains_segment* ‹*convex* S›)
then have *2*: *contour_integral* (*linepath* b c) f =
contour_integral (*linepath* b a') f + *contour_integral* (*linepath* a' c) f
by (*rule contour_integral_split_linepath* [*OF* _ a'])
have *3*: *contour_integral* (*reversepath* (*linepath* b' a')) f =
$-$ *contour_integral* (*linepath* b' a') f
by (*rule contour_integral_reversepath* [*OF valid_path_linepath*])
have *fcd_le*: f *field_differentiable at* x
if $x \in$ *interior* $S \wedge x \in$ *interior* $\{x.\ d \cdot x \leq k\}$ **for** x
proof $-$
have f *holomorphic_on* $S \cap \{c.\ d \cdot c < k\}$
by (*metis* (*no_types*) *Collect_conj_eq Collect_mem_eq holf1*)
then have $\exists C\ D.\ x \in$ *interior* $C \cap$ *interior* $D \wedge f$ *holomorphic_on interior* $C \cap$ *interior* D
using *that*
by (*metis Collect_mem_eq Int_Collect* ‹$d \neq 0$› *interior_halfspace_le interior_open* ‹*open* S›)
then show f *field_differentiable at* x
by (*metis at_within_interior holomorphic_on_def interior_Int interior_interior*)
qed
have *ab_le*: $\bigwedge x.\ x \in$ *closed_segment* a $b \Longrightarrow d \cdot x \leq k$
proof $-$
fix x :: *complex*
assume $x \in$ *closed_segment* a b
then have $\bigwedge C.\ x \in C \vee b \notin C \vee a \notin C \vee \neg$ *convex* C
by (*meson contra_subsetD convex_contains_segment*)
then show $d \cdot x \leq k$
by (*metis lek convex_halfspace_le mem_Collect_eq*)
qed

have *cs*: *closed_segment a′ b′* ⊆ {*x*. *d* · *x* ≤ *k*} ∧ *closed_segment b′ a* ⊆ {*x*. *d* · *x* ≤ *k*}
by (*simp add*: ‹*d* · *a′* = *k*› ‹*d* · *b′* = *k*› *closed_segment_subset convex_halfspace_le lek*(*1*))
have *continuous_on* (*S* ∩ {*x*. *d* · *x* ≤ *k*}) *f* **using** *contf*
by (*simp add*: *continuous_on_subset*)
then have (*f has_contour_integral 0*)
(*linepath a b* +++ *linepath b a′* +++ *linepath a′ b′* +++ *linepath b′ a*)
apply (*rule Cauchy_theorem_convex* [**where** *K* = {}])
by (*simp_all add*: *path_image_join convex_Int convex_halfspace_le* ‹*convex S*› *fcd_le ab_le*
closed_segment_subset abc a′b′ ba′ cs)
then have *4*: *contour_integral* (*linepath a b*) *f* +
contour_integral (*linepath b a′*) *f* +
contour_integral (*linepath a′ b′*) *f* +
contour_integral (*linepath b′ a*) *f* = *0*
by (*rule has_chain_integral_chain_integral4*)
have *fcd_ge*: *f field_differentiable at x*
if *x* ∈ *interior S* ∧ *x* ∈ *interior* {*x*. *k* ≤ *d* · *x*} **for** *x*
proof −
have *f2*: *f holomorphic_on S* ∩ {*c*. *k* < *d* · *c*}
by (*metis* (*full_types*) *Collect_conj_eq Collect_mem_eq holf2*)
have *f3*: *interior S* = *S*
by (*simp add*: *interior_open* ‹*open S*›)
then have *x* ∈ *S* ∩ *interior* {*c*. *k* ≤ *d* · *c*}
using *that* **by** *simp*
then show *f field_differentiable at x*
using *f3 f2* **unfolding** *holomorphic_on_def*
by (*metis* (*no_types*) ‹*d* ≠ *0*› *at_within_interior interior_Int interior_halfspace_ge interior_interior*)
qed
have *cs*: *closed_segment c b′* ⊆ {*x*. *k* ≤ *d* · *x*} ∧ *closed_segment b′ a′* ⊆ {*x*. *k* ≤ *d* · *x*}
by (*simp add*: ‹*d* · *a′* = *k*› *b′c closed_segment_subset convex_halfspace_ge*)
have *continuous_on* (*S* ∩ {*x*. *k* ≤ *d* · *x*}) *f* **using** *contf*
by (*simp add*: *continuous_on_subset*)
then have (*f has_contour_integral 0*) (*linepath a′ c* +++ *linepath c b′* +++ *linepath b′ a′*)
apply (*rule Cauchy_theorem_convex* [**where** *K* = {}])
by (*simp_all add*: *path_image_join convex_Int convex_halfspace_ge* ‹*convex S*›
fcd_ge closed_segment_subset abc a′b′ a′c cs)
then have *5*: *contour_integral* (*linepath a′ c*) *f* + *contour_integral* (*linepath c b′*) *f* + *contour_integral* (*linepath b′ a′*) *f* = *0*
by (*rule has_chain_integral_chain_integral3*)
show *?thesis*
using *1 2 3 4 5* **by** (*metis add.assoc eq_neg_iff_add_eq_0 reversepath_linepath*)
qed

lemma *hol_pal_lem3*:
assumes *S*: *convex S open S*
and *abc*: $a \in S$ $b \in S$ $c \in S$
and $d \neq 0$ **and** *lek*: $d \cdot a \leq k$
and *holf1*: *f holomorphic_on* $\{z.\ z \in S \wedge d \cdot z < k\}$
and *holf2*: *f holomorphic_on* $\{z.\ z \in S \wedge k < d \cdot z\}$
and *contf*: *continuous_on S f*
shows *contour_integral* (*linepath a b*) *f* +
contour_integral (*linepath b c*) *f* +
contour_integral (*linepath c a*) *f* = *0*
proof (*cases* $d \cdot b \leq k$)
case *True* **show** *?thesis*
by (*rule hol_pal_lem2* [*OF S abc* ‹$d \neq 0$› *lek True holf1 holf2 contf*])
next
case *False*
show *?thesis*
proof (*cases* $d \cdot c \leq k$)
case *True*
have *contour_integral* (*linepath c a*) *f* +
contour_integral (*linepath a b*) *f* +
contour_integral (*linepath b c*) *f* = *0*
by (*rule hol_pal_lem2* [*OF S* ‹$c \in S$› ‹$a \in S$› ‹$b \in S$› ‹$d \neq 0$› ‹$d \cdot c \leq k$›
lek holf1 holf2 contf])
then show *?thesis*
by (*simp add*: *algebra_simps*)
next
case *False*
have *contour_integral* (*linepath b c*) *f* +
contour_integral (*linepath c a*) *f* +
contour_integral (*linepath a b*) *f* = *0*
using *hol_pal_lem2* [*OF S* ‹$b \in S$› ‹$c \in S$› ‹$a \in S$›, *of* $-d$ $-k$]
using ‹$d \neq 0$› ‹$\neg\ d \cdot b \leq k$› *False* **by** (*simp_all add*: *holf1 holf2 contf*)
then show *?thesis*
by (*simp add*: *algebra_simps*)
qed
qed

lemma *hol_pal_lem4*:
assumes *S*: *convex S open S*
and *abc*: $a \in S$ $b \in S$ $c \in S$ **and** $d \neq 0$
and *holf1*: *f holomorphic_on* $\{z.\ z \in S \wedge d \cdot z < k\}$
and *holf2*: *f holomorphic_on* $\{z.\ z \in S \wedge k < d \cdot z\}$
and *contf*: *continuous_on S f*
shows *contour_integral* (*linepath a b*) *f* +
contour_integral (*linepath b c*) *f* +
contour_integral (*linepath c a*) *f* = *0*
proof (*cases* $d \cdot a \leq k$)
case *True* **show** *?thesis*
by (*rule hol_pal_lem3* [*OF S abc* ‹$d \neq 0$› *True holf1 holf2 contf*])

```
next
  case False
  show ?thesis
    using ‹d ≠ 0› hol_pal_lem3 [OF S abc, of −d −k] False
    by (simp_all add: holf1 holf2 contf)
qed

lemma holomorphic_on_paste_across_line:
  assumes S: open S and d ≠ 0
      and holf1: f holomorphic_on (S ∩ {z. d • z < k})
      and holf2: f holomorphic_on (S ∩ {z. k < d • z})
      and contf: continuous_on S f
    shows f holomorphic_on S
proof −
  have ∗: ∃t. open t ∧ p ∈ t ∧ continuous_on t f ∧
              (∀a b c. convex hull {a, b, c} ⊆ t ⟶
                       contour_integral (linepath a b) f +
                       contour_integral (linepath b c) f +
                       contour_integral (linepath c a) f = 0)
         if p ∈ S for p
  proof −
    obtain e where e>0 and e: ball p e ⊆ S
      using ‹p ∈ S› openE S by blast
    then have continuous_on (ball p e) f
      using contf continuous_on_subset by blast
    moreover have f holomorphic_on {z. dist p z < e ∧ d • z < k}
      apply (rule holomorphic_on_subset [OF holf1])
      using e by auto
    moreover have f holomorphic_on {z. dist p z < e ∧ k < d • z}
      apply (rule holomorphic_on_subset [OF holf2])
      using e by auto
    ultimately show ?thesis
      apply (rule_tac x=ball p e in exI)
      using ‹e > 0› e ‹d ≠ 0› hol_pal_lem4 [of ball p e _ _ _ d _ k]
      by (force simp add: subset_hull)
  qed
  show ?thesis
    by (blast intro: ∗ Morera_local_triangle analytic_imp_holomorphic)
qed

proposition Schwarz_reflection:
  assumes open S and cnjs: cnj ‘ S ⊆ S
      and  holf: f holomorphic_on (S ∩ {z. 0 < Im z})
      and contf: continuous_on (S ∩ {z. 0 ≤ Im z}) f
      and f: ⋀z. ⟦z ∈ S; z ∈ ℝ⟧ ⟹ (f z) ∈ ℝ
    shows (λz. if 0 ≤ Im z then f z else cnj(f(cnj z))) holomorphic_on S
proof −
  have 1: (λz. if 0 ≤ Im z then f z else cnj (f (cnj z))) holomorphic_on (S ∩ {z.
0 < Im z})
```

by (*force intro*: *iffD1* [*OF holomorphic_cong* [*OF refl*] *holf*])
have *cont_cfc*: *continuous_on* (*S* ∩ {*z*. *Im z* ≤ *0*}) (*cnj o f o cnj*)
using *cnjs*
by (*intro continuous_intros continuous_on_compose continuous_on_subset* [*OF contf*]) *auto*
have *cnj* ∘ *f* ∘ *cnj field_differentiable at x within S* ∩ {*z*. *Im z* < *0*}
if *x* ∈ *S Im x* < *0 f field_differentiable at* (*cnj x*) *within S* ∩ {*z*. *0* < *Im z*}
for *x*
using *that*
apply (*clarsimp simp add*: *field_differentiable_def has_field_derivative_iff Lim_within dist_norm*)
apply (*rule_tac x*=*cnj f'* **in** *exI*)
apply (*elim all_forward ex_forward conj_forward imp_forward asm_rl*, *clarify*)
apply (*drule_tac x*=*cnj xa* **in** *bspec*)
using *cnjs* **apply** *force*
apply (*metis complex_cnj_cnj complex_cnj_diff complex_cnj_divide complex_mod_cnj*)
done
then have *hol_cfc*: (*cnj o f o cnj*) *holomorphic_on* (*S* ∩ {*z*. *Im z* < *0*})
using *holf cnjs*
by (*force simp*: *holomorphic_on_def*)
have *2*: (λ*z*. *if 0* ≤ *Im z then f z else cnj* (*f* (*cnj z*))) *holomorphic_on* (*S* ∩ {*z*. *Im z* < *0*})
apply (*rule iffD1* [*OF holomorphic_cong* [*OF refl*]])
using *hol_cfc* **by** *auto*
have [*simp*]: (*S* ∩ {*z*. *0* ≤ *Im z*}) ∪ (*S* ∩ {*z*. *Im z* ≤ *0*}) = *S*
by *force*
have *eq*: ⋀*z*. ⟦*z* ∈ *S*; *Im z* ≤ *0*; *0* ≤ *Im z*⟧ ⟹ *f z* = *cnj* (*f* (*cnj z*))
using *f Reals_cnj_iff complex_is_Real_iff* **by** *auto*
have *continuous_on* ((*S* ∩ {*z*. *0* ≤ *Im z*}) ∪ (*S* ∩ {*z*. *Im z* ≤ *0*}))
(λ*z*. *if 0* ≤ *Im z then f z else cnj* (*f* (*cnj z*)))
apply (*rule continuous_on_cases_local*)
using *cont_cfc contf*
by (*simp_all add*: *closedin_closed_Int closed_halfspace_Im_le closed_halfspace_Im_ge eq*)
then have *3*: *continuous_on S* (λ*z*. *if 0* ≤ *Im z then f z else cnj* (*f* (*cnj z*)))
by *force*
show *?thesis*
apply (*rule holomorphic_on_paste_across_line* [*OF* ‹*open S*›, *of* − i _ *0*])
using *1 2 3* **by** *auto*
qed

5.9 Bloch's theorem

lemma *Bloch_lemma_0*:
assumes *holf*: *f holomorphic_on cball 0 r* **and** *0* < *r*
and [*simp*]: *f 0* = *0*
and *le*: ⋀*z*. *norm z* < *r* ⟹ *norm*(*deriv f z*) ≤ *2* ∗ *norm*(*deriv f 0*)

```
    shows ball 0 ((3 - 2 * sqrt 2) * r * norm(deriv f 0)) ⊆ f ‘ ball 0 r
proof -
  have sqrt 2 < 3/2
    by (rule real_less_lsqrt) (auto simp: power2_eq_square)
  then have sq3: 0 < 3 - 2 * sqrt 2 by simp
  show ?thesis
  proof (cases deriv f 0 = 0)
    case True then show ?thesis by simp
  next
    case False
    define C where C = 2 * norm(deriv f 0)
    have 0 < C using False by (simp add: C_def)
    have holf': f holomorphic_on ball 0 r using holf
      using ball_subset_cball holomorphic_on_subset by blast
    then have holdf': deriv f holomorphic_on ball 0 r
      by (rule holomorphic_deriv [OF _ open_ball])
    have Le1: norm(deriv f z - deriv f 0) ≤ norm z / (r - norm z) * C
              if norm z < r for z
    proof -
      have T1: norm(deriv f z - deriv f 0) ≤ norm z / (R - norm z) * C
               if R: norm z < R R < r for R
      proof -
        have 0 < R using R
          by (metis less_trans norm_zero zero_less_norm_iff)
        have df_le: ⋀x. norm x < r ⟹ norm (deriv f x) ≤ C
          using le by (simp add: C_def)
        have hol_df: deriv f holomorphic_on cball 0 R
          using R holdf' holomorphic_on_subset by auto
        have *: ((λw. deriv f w / (w - z)) has_contour_integral 2 * pi * ı * deriv
f z) (circlepath 0 R)
                if norm z < R for z
          using ‹0 < R› that Cauchy_integral_formula_convex_simple [OF con-
vex_cball hol_df, of _ circlepath 0 R]
          by (force simp: winding_number_circlepath)
        have **: ((λx. deriv f x / (x - z) - deriv f x / x) has_contour_integral
                  of_real (2 * pi) * ı * (deriv f z - deriv f 0))
                 (circlepath 0 R)
          using has_contour_integral_diff [OF * [of z] * [of 0]] ‹0 < R› that
          by (simp add: algebra_simps)
        have [simp]: ⋀x. norm x = R ⟹ x ≠ z using that(1) by blast
        have norm (deriv f x / (x - z) - deriv f x / x)
                    ≤ C * norm z / (R * (R - norm z))
                  if norm x = R for x
        proof -
          have [simp]: norm (deriv f x * x - deriv f x * (x - z)) =
                       norm (deriv f x) * norm z
            by (simp add: norm_mult right_diff_distrib')
          show ?thesis
            using ‹0 < R› ‹0 < C› R that
```

```
            by (auto simp add: norm_mult norm_divide divide_simps df_le
mult_mono norm_triangle_ineq2)
      qed
      then show ?thesis
        using has_contour_integral_bound_circlepath
              [OF **, of C * norm z/(R*(R - norm z))]
             ‹0 < R› ‹0 < C› R
        apply (simp add: norm_mult norm_divide)
        apply (simp add: divide_simps mult.commute)
        done
    qed
    obtain r' where r': norm z < r' r' < r
      using Rats_dense_in_real [of norm z r] ‹norm z < r› by blast
    then have [simp]: closure {r'<..<r} = {r'..r} by simp
    show ?thesis
      apply (rule continuous_ge_on_closure
              [where f = λr. norm z / (r - norm z) * C and s = {r'<..<r},
               OF _ _ T1])
      using that r'
      by (auto simp: not_le intro!: continuous_intros)
  qed
  have *: (norm z - norm z^2/(r - norm z)) * norm(deriv f 0) ≤ norm(f z)
          if r: norm z < r for z
  proof -
    have 1: ⋀x. x ∈ ball 0 r ⟹
          ((λz. f z - deriv f 0 * z) has_field_derivative deriv f x - deriv f 0)
           (at x within ball 0 r)
      by (rule derivative_eq_intros holomorphic_derivI holf' | simp)+
    have 2: closed_segment 0 z ⊆ ball 0 r
     by (metis ‹0 < r› convex_ball convex_contains_segment dist_self mem_ball
mem_ball_0 that)
    have 4: norm (deriv f (x *R z) - deriv f 0) * norm z ≤ norm z * norm z *
x * C / (r - norm z)
          if x: 0 ≤ x x ≤ 1 for x
    proof -
      have [simp]: x * norm z < r
     using r x by (meson le_less_trans mult_le_cancel_right2 norm_not_less_zero)
      have norm (deriv f (x *R z) - deriv f 0) ≤ norm (x *R z) / (r - norm (x
*R z)) * C
        apply (rule Le1) using r x ‹0 < r› by simp
      also have ... ≤ norm (x *R z) / (r - norm z) * C
        using r x ‹0 < r›
        apply (simp add: field_split_simps)
      by (simp add: ‹0 < C› mult.assoc mult_left_le_one_le ordered_comm_semiring_class.comm_mult_le
      finally have norm (deriv f (x *R z) - deriv f 0) * norm z ≤ norm (x *R
z)  / (r - norm z) * C * norm z
        by (rule mult_right_mono) simp
      with x show ?thesis by (simp add: algebra_simps)
    qed
```

have *le_norm*: $abc \le norm\ d - e \Longrightarrow norm(f - d) \le e \Longrightarrow abc \le norm\ f$
for *abc d e* **and** *f*::*complex*
by (*metis add_diff_cancel_left' add_diff_eq diff_left_mono norm_diff_ineq order_trans*)
have *norm* (*integral* {*0..1*} (λ*x*. (*deriv f* (*x* *$_R$ *z*) − *deriv f 0*) ∗ *z*))
≤ *integral* {*0..1*} (λ*t*. (*norm z*)^2 ∗ *t* / (*r* − *norm z*) ∗ *C*)
proof (*rule integral_norm_bound_integral*)
show (λ*x*. (*deriv f* (*x* *$_R$ *z*) − *deriv f 0*) ∗ *z*) *integrable_on* {*0..1*}
using *contour_integral_primitive* [*OF 1*, *of linepath 0 z*] *2*
by (*simp add*: *has_contour_integral_linepath has_integral_integrable_integral*)
have (∗) ((*cmod z*)^2) *integrable_on* {*0..1*}
by (*metis ident_integrable_on integrable_0 integrable_eq integrable_on_cmult_iff lambda_zero*)
then show (λ*t*. (*norm z*)^2 ∗ *t* / (*r* − *norm z*) ∗ *C*) *integrable_on* {*0..1*}
using *integrable_on_cmult_right*[**where** *'b*=*real*, *simplified*] *integrable_on_cdivide*
[**where** *'b*=*real*, *simplified*]
by *blast*
qed (*simp add*: *norm_mult power2_eq_square 4*)
then have *int_le*: *norm* (*f z* − *deriv f 0* ∗ *z*) ≤ (*norm z*)^2 ∗ *norm*(*deriv f 0*)
/ ((*r* − *norm z*))
using *contour_integral_primitive* [*OF 1*, *of linepath 0 z*] *2*
by (*simp add*: *has_contour_integral_linepath has_integral_integrable_integral C_def*)
have *norm z* ∗ (*norm* (*deriv f 0*) ∗ (*r* − *norm z* − *norm z*)) ≤ *norm z* ∗
(*norm* (*deriv f 0*) ∗ (*r* − *norm z*) − *norm* (*deriv f 0*) ∗ *norm z*)
by (*simp add*: *algebra_simps*)
then have §: (*norm z* ∗ (*r* − *norm z*) − *norm z* ∗ *norm z*) ∗ *norm* (*deriv f*
0) ≤ *norm* (*deriv f 0*) ∗ *norm z* ∗ (*r* − *norm z*) − *norm z* ∗ *norm z* ∗ *norm* (*deriv*
f 0)
by (*simp add*: *algebra_simps*)
show *?thesis*
apply (*rule le_norm* [*OF _ int_le*])
using ‹*norm z* < *r*›
by (*simp add*: *power2_eq_square divide_simps C_def norm_mult* §)
qed
have *sq201* [*simp*]: *0* < (*1* − *sqrt 2* / *2*) (*1* − *sqrt 2* / *2*) < *1*
by (*auto simp*: *sqrt2_less_2*)
have *1*: *continuous_on* (*closure* (*ball 0* ((*1* − *sqrt 2* / *2*) ∗ *r*))) *f*
proof (*rule continuous_on_subset* [*OF holomorphic_on_imp_continuous_on*
[*OF holf*]])
show *closure* (*ball 0* ((*1* − *sqrt 2* / *2*) ∗ *r*)) ⊆ *cball 0 r*
proof −
have (*1* − *sqrt 2* / *2*) ∗ *r* ≤ *r*
by (*simp add*: ‹*0* < *r*›)
then show *?thesis*
by (*meson ball_subset_cball closed_cball closure_minimal dual_order.trans subset_ball*)
qed
qed

```
    have 2: open (f ‘ interior (ball 0 ((1 − sqrt 2 / 2) ∗ r)))
    proof (rule open_mapping_thm [OF holf′ open_ball connected_ball])
      show interior (ball 0 ((1 − sqrt 2 / 2) ∗ r)) ⊆ ball (0::complex) r
        using ‹0 < r› mult_pos_pos sq201 by (simp add: ball_subset_ball_iff)
      show ¬ f constant_on ball 0 r
       using False ‹0 < r› centre_in_ball holf′ holomorphic_nonconstant by blast
    qed auto
    have ball 0 ((3 − 2 ∗ sqrt 2) ∗ r ∗ norm (deriv f 0)) =
          ball (f 0) ((3 − 2 ∗ sqrt 2) ∗ r ∗ norm (deriv f 0))
      by simp
    also have ... ⊆ f ‘ ball 0 ((1 − sqrt 2 / 2) ∗ r)
    proof −
      have 3: (3 − 2 ∗ sqrt 2) ∗ r ∗ norm (deriv f 0) ≤ norm (f z)
        if norm z = (1 − sqrt 2 / 2) ∗ r for z
        apply (rule order_trans [OF _ ∗])
        using ‹0 < r›
         apply (simp_all add: field_simps power2_eq_square that)
        apply (simp add: mult.assoc [symmetric])
        done
      show ?thesis
        apply (rule ball_subset_open_map_image [OF 1 2 _ bounded_ball])
        using ‹0 < r› sq201 3 C_def ‹0 < C› sq3 by auto
    qed
    also have ... ⊆ f ‘ ball 0 r
    proof −
      have ⋀x. (1 − sqrt 2 / 2) ∗ r ≤ r
        using ‹0 < r› by (auto simp: field_simps)
      then show ?thesis
        by auto
    qed
    finally show ?thesis .
  qed
qed

lemma Bloch_lemma:
  assumes holf: f holomorphic_on cball a r and 0 < r
    and le: ⋀z. z ∈ ball a r ⟹ norm(deriv f z) ≤ 2 ∗ norm(deriv f a)
  shows ball (f a) ((3 − 2 ∗ sqrt 2) ∗ r ∗ norm(deriv f a)) ⊆ f ‘ ball a r (is ?lhs
⊆ ?rhs)
proof −
  have fz: (λz. f (a + z)) = f o (λz. (a + z))
    by (simp add: o_def)
  have hol0: (λz. f (a + z)) holomorphic_on cball 0 r
     unfolding fz by (intro holomorphic_intros holf holomorphic_on_compose |
simp)+
  then have [simp]: ⋀x. norm x < r ⟹ (λz. f (a + z)) field_differentiable at x
    by (metis open_ball at_within_open ball_subset_cball diff_0 dist_norm holo-
morphic_on_def holomorphic_on_subset mem_ball norm_minus_cancel)
  have [simp]: ⋀z. norm z < r ⟹ f field_differentiable at (a + z)
```

```
    by (metis holf open_ball add_diff_cancel_left' dist_complex_def holomor-
phic_on_imp_differentiable_at holomorphic_on_subset interior_cball interior_subset
mem_ball norm_minus_commute)
  then have [simp]: f field_differentiable at a
    by (metis add.comm_neutral ‹0 < r› norm_eq_zero)
  have hol1: (λz. f (a + z) − f a) holomorphic_on cball 0 r
    by (intro holomorphic_intros hol0)
  then have §: ball 0 ((3 − 2 ∗ sqrt 2) ∗ r ∗ norm (deriv (λz. f (a + z) − f a)
0))
              ⊆ (λz. f (a + z) − f a) ‘ ball 0 r
    apply (rule Bloch_lemma_0)
    using ‹0 < r›
      apply (simp_all add: ‹0 < r›)
    apply (simp add: fz deriv_chain dist_norm le)
    done
  show ?thesis
  proof clarify
    fix x
    assume x ∈ ?lhs
    with subsetD [OF §, of x − f a] show x ∈ ?rhs
    by (force simp: fz ‹0 < r› dist_norm deriv_chain field_differentiable_compose)
  qed
qed

proposition Bloch_unit:
  assumes holf: f holomorphic_on ball a 1 and [simp]: deriv f a = 1
  obtains b r where 1/12 < r and ball b r ⊆ f ‘ (ball a 1)
proof −
  define r :: real where r = 249/256
  have 0 < r r < 1 by (auto simp: r_def)
  define g where g z = deriv f z ∗ of_real(r − norm(z − a)) for z
  have deriv f holomorphic_on ball a 1
    by (rule holomorphic_deriv [OF holf open_ball])
  then have continuous_on (ball a 1) (deriv f)
    using holomorphic_on_imp_continuous_on by blast
  then have continuous_on (cball a r) (deriv f)
    by (rule continuous_on_subset) (simp add: cball_subset_ball_iff ‹r < 1›)
  then have continuous_on (cball a r) g
    by (simp add: g_def continuous_intros)
  then have 1: compact (g ‘ cball a r)
    by (rule compact_continuous_image [OF _ compact_cball])
  have 2: g ‘ cball a r ≠ {}
    using ‹r > 0› by auto
  obtain p where pr: p ∈ cball a r
            and pge: ⋀y. y ∈ cball a r ⟹ norm (g y) ≤ norm (g p)
    using distance_attains_sup [OF 1 2, of 0] by force
  define t where t = (r − norm(p − a)) / 2
  have norm (p − a) ≠ r
    using pge [of a] ‹r > 0› by (auto simp: g_def norm_mult)
```

```
  then have norm (p − a) < r using pr
    by (simp add: norm_minus_commute dist_norm)
  then have 0 < t
    by (simp add: t_def)
  have cpt: cball p t ⊆ ball a r
   using ‹0 < t› by (simp add: cball_subset_ball_iff dist_norm t_def field_simps)
  have gen_le_dfp: norm (deriv f y) * (r − norm (y − a)) / (r − norm (p − a))
≤ norm (deriv f p)
          if y ∈ cball a r for y
  proof −
    have [simp]: norm (y − a) ≤ r
      using that by (simp add: dist_norm norm_minus_commute)
    have norm (g y) ≤ norm (g p)
      using pge [OF that] by simp
    then have norm (deriv f y) * abs (r − norm (y − a)) ≤ norm (deriv f p) *
abs (r − norm (p − a))
      by (simp only: dist_norm g_def norm_mult norm_of_real)
    with that ‹norm (p − a) < r› show ?thesis
      by (simp add: dist_norm field_split_simps)
  qed
  have le_norm_dfp: r / (r − norm (p − a)) ≤ norm (deriv f p)
    using gen_le_dfp [of a] ‹r > 0› by auto
  have 1: f holomorphic_on cball p t
    using cpt ‹r < 1› order_subst1 subset_ball
    by (force simp add: intro!: holomorphic_on_subset [OF holf])
  have 2: norm (deriv f z) ≤ 2 * norm (deriv f p) if z ∈ ball p t for z
  proof −
    have z: z ∈ cball a r
      by (meson ball_subset_cball subsetD cpt that)
    then have norm(z − a) < r
    by (metis ball_subset_cball contra_subsetD cpt dist_norm mem_ball norm_minus_commute
that)
    have norm (deriv f z) * (r − norm (z − a)) / (r − norm (p − a)) ≤ norm
(deriv f p)
      using gen_le_dfp [OF z] by simp
    with ‹norm (z − a) < r› ‹norm (p − a) < r›
    have norm (deriv f z) ≤ (r − norm (p − a)) / (r − norm (z − a)) * norm
(deriv f p)
      by (simp add: field_simps)
    also have ... ≤ 2 * norm (deriv f p)
    proof (rule mult_right_mono)
      show (r − cmod (p − a)) / (r − cmod (z − a)) ≤ 2
       using that ‹norm (p − a) < r› ‹norm(z − a) < r› dist_triangle3 [of z a p]
        by (simp add: field_simps t_def dist_norm [symmetric])
    qed auto
    finally show ?thesis .
  qed
  have sqrt2: sqrt 2 < 2113/1494
    by (rule real_less_lsqrt) (auto simp: power2_eq_square)
```

```
  then have sq3: 0 < 3 - 2 * sqrt 2 by simp
  have 1 / 12 / ((3 - 2 * sqrt 2) / 2) < r
    using sq3 sqrt2 by (auto simp: field_simps r_def)
  also have ... ≤ cmod (deriv f p) * (r - cmod (p - a))
    using ‹norm (p - a) < r› le_norm_dfp   by (simp add: pos_divide_le_eq)
  finally have 1 / 12 < cmod (deriv f p) * (r - cmod (p - a)) * ((3 - 2 * sqrt
2) / 2)
    using pos_divide_less_eq half_gt_zero_iff sq3 by blast
  then have **: 1 / 12 < (3 - 2 * sqrt 2) * t * norm (deriv f p)
    using sq3 by (simp add: mult.commute t_def)
  have ball (f p) ((3 - 2 * sqrt 2) * t * norm (deriv f p)) ⊆ f ‘ ball p t
    by (rule Bloch_lemma [OF 1 ‹0 < t› 2])
  also have ... ⊆ f ‘ ball a 1
  proof -
    have ball a r ⊆ ball a 1
      using ‹0 < t› ‹r < 1› by (simp add: ball_subset_ball_iff dist_norm)
    then show ?thesis
      using ball_subset_cball cpt by blast
  qed
  finally have ball (f p) ((3 - 2 * sqrt 2) * t * norm (deriv f p)) ⊆ f ‘ ball a 1 .
  with ** show ?thesis
    by (rule that)
qed

theorem Bloch:
  assumes holf: f holomorphic_on ball a r and 0 < r
      and r': r' ≤ r * norm (deriv f a) / 12
  obtains b where ball b r' ⊆ f ‘ (ball a r)
proof (cases deriv f a = 0)
  case True with r' show ?thesis
    using ball_eq_empty that by fastforce
next
  case False
  define C where C = deriv f a
  have 0 < norm C using False by (simp add: C_def)
  have dfa: f field_differentiable at a
    using ‹0 < r› holomorphic_on_imp_differentiable_at [OF holf] by auto
  have fo: (λz. f (a + of_real r * z)) = f o (λz. (a + of_real r * z))
    by (simp add: o_def)
  have holf': f holomorphic_on (λz. a + complex_of_real r * z) ‘ ball 0 1
    using ‹0 < r› holomorphic_on_subset [OF holf] by (force simp: dist_norm
norm_mult)
  have 1: (λz. f (a + r * z) / (C * r)) holomorphic_on ball 0 1
    using ‹0 < r› ‹0 < norm C›
    by (intro holomorphic_intros holomorphic_on_compose holf'; simp add: fo)+
  have ((λz. f (a + of_real r * z) / (C * of_real r)) has_field_derivative
       (deriv f (a + of_real r * z) / C)) (at z)
      if norm z < 1 for z
  proof -
```

```
    have fd: f field_differentiable at (a + complex_of_real r * z)
     using ‹0 < r› by (simp_all add: dist_norm norm_mult holomorphic_on_imp_differentiable_at
[OF holf] that)
    have *: ((λx. f (a + of_real r * x)) has_field_derivative
          (deriv f (a + of_real r * z) * of_real r)) (at z)
     by (rule fd DERIV_chain [OF field_differentiable_derivI]derivative_eq_intros
| simp add: fo)+
    show ?thesis
      apply (rule derivative_eq_intros * | simp)+
      using ‹0 < r› by (auto simp: C_def False)
  qed
  have deriv (λz. f (a + of_real r * z) / (C * of_real r)) 0 = deriv (λz. f (a +
complex_of_real r * z)) 0 /
    (C * complex_of_real r)
    apply (rule deriv_cdivide_right)
   by (metis (no_types) DERIV_chain2 add.right_neutral dfa field_differentiable_add_const
field_differentiable_def field_differentiable_linear fo mult_zero_right)
  also have ... = 1
   using ‹0 < r› by (simp add: C_def False fo derivative_intros dfa deriv_chain)
  finally have 2: deriv (λz. f (a + of_real r * z) / (C * of_real r)) 0 = 1 .
  have sb1: (*) (C * r) ‘ (λz. f (a + of_real r * z) / (C * r)) ‘ ball 0 1
            ⊆ f ‘ ball a r
    using ‹0 < r› by (auto simp: dist_norm norm_mult C_def False)
  have sb2: ball (C * r * b) r' ⊆ (*) (C * r) ‘ ball b t
            if 1 / 12 < t for b t
  proof −
    have *: r * cmod (deriv f a) / 12 ≤ r * (t * cmod (deriv f a))
     using that ‹0 < r› less_eq_real_def mult.commute mult.right_neutral mult_left_mono
norm_ge_zero times_divide_eq_right
      by auto
    show ?thesis
      apply clarify
      apply (rule_tac x=x / (C * r) in image_eqI)
       using ‹0 < r› apply (simp_all add: dist_norm norm_mult norm_divide
C_def False field_simps)
      using * r' by linarith
  qed
  show ?thesis
    apply (rule Bloch_unit [OF 1 2])
    apply (rule_tac b=(C * of_real r) * b in that)
    using image_mono sb1 sb2 by fastforce
qed

corollary Bloch_general:
  assumes holf: f holomorphic_on S and a ∈ S
      and tle: ⋀z. z ∈ frontier S ⟹ t ≤ dist a z
      and rle: r ≤ t * norm(deriv f a) / 12
  obtains b where ball b r ⊆ f ‘ S
proof −
```

```
  consider r ≤ 0 | 0 < t * norm(deriv f a) / 12 using rle by force
  then show ?thesis
  proof cases
    case 1 then show ?thesis
      by (simp add: ball_empty that)
  next
    case 2
    show ?thesis
    proof (cases deriv f a = 0)
      case True then show ?thesis
        using rle by (simp add: ball_empty that)
    next
      case False
      then have t > 0
        using 2 by (force simp: zero_less_mult_iff)
      have ¬ ball a t ⊆ S ⟹ ball a t ∩ frontier S ≠ {}
       by (metis Diff_eq_empty_iff ‹0 < t› ‹a ∈ S› closure_Int_ball_not_empty
closure_subset connected_Int_frontier connected_ball inf.commute)
      with tle have *: ball a t ⊆ S by fastforce
      then have 1: f holomorphic_on ball a t
        using holf using holomorphic_on_subset by blast
      show ?thesis
        apply (rule Bloch [OF 1 ‹t > 0› rle])
        apply (rule_tac b=b in that)
        using * apply force
        done
    qed
  qed
qed

end
theory Complex_Singularities
  imports Conformal_Mappings
begin
```

5.10 Non-essential singular points

```
definition is_pole ::
  ('a::topological_space ⇒ 'b::real_normed_vector) ⇒ 'a ⇒ bool where
  is_pole f a = (LIM x (at a). f x :> at_infinity)

lemma is_pole_cong:
  assumes eventually (λx. f x = g x) (at a) a=b
  shows is_pole f a ⟷ is_pole g b
  unfolding is_pole_def using assms by (intro filterlim_cong,auto)

lemma is_pole_transform:
  assumes is_pole f a eventually (λx. f x = g x) (at a) a=b
  shows is_pole g b
```

using *is_pole_cong assms* **by** *auto*

lemma *is_pole_shift_iff*:
fixes $f :: ('a::real\_normed\_vector \Rightarrow 'b::real\_normed\_vector)$
shows $is\_pole\ (f \circ (+)\ d)\ a = is\_pole\ f\ (a + d)$
by (*metis add_diff_cancel_right' filterlim_shift_iff is_pole_def*)

lemma *is_pole_tendsto*:
fixes $f::('a::topological\_space \Rightarrow 'b::real\_normed\_div\_algebra)$
shows $is\_pole\ f\ x \Longrightarrow ((inverse\ o\ f) \longrightarrow 0)\ (at\ x)$
unfolding *is_pole_def*
by (*auto simp add:filterlim_inverse_at_iff*[*symmetric*] *comp_def filterlim_at*)

lemma *is_pole_inverse_holomorphic*:
assumes *open s*
and *f_holo:f holomorphic_on* $(s-\{z\})$
and *pole:is_pole f z*
and *non_z:*$\forall\, x{\in}s-\{z\}.\ f\ x{\neq}0$
shows $(\lambda x.\ if\ x{=}z\ then\ 0\ else\ inverse\ (f\ x))\ holomorphic\_on\ s$
proof −
define *g* **where** $g \equiv \lambda x.\ if\ x{=}z\ then\ 0\ else\ inverse\ (f\ x)$
have *isCont g z* **unfolding** *isCont_def* **using** *is_pole_tendsto*[*OF pole*]
by (*simp add*: *g_def cong*: *LIM_cong*)
moreover have *continuous_on* $(s-\{z\})\ f$ **using** *f_holo holomorphic_on_imp_continuous_on*
by *auto*
hence *continuous_on* $(s-\{z\})\ (inverse\ o\ f)$ **unfolding** *comp_def*
by (*auto elim*!:*continuous_on_inverse simp add:non_z*)
hence *continuous_on* $(s-\{z\})\ g$ **unfolding** *g_def*
apply (*subst continuous_on_cong*[**where** $t{=}s-\{z\}$ **and** $g{=}inverse\ o\ f$])
by *auto*
ultimately have *continuous_on s g* **using** *open_delete*[*OF* ‹*open s*›] ‹*open s*›
by (*auto simp add:continuous_on_eq_continuous_at*)
moreover have $(inverse\ o\ f)\ holomorphic\_on\ (s-\{z\})$
unfolding *comp_def* **using** *f_holo*
by (*auto elim*!:*holomorphic_on_inverse simp add:non_z*)
hence *g holomorphic_on* $(s-\{z\})$
apply (*subst holomorphic_cong*[**where** $t{=}s-\{z\}$ **and** $g{=}inverse\ o\ f$])
by (*auto simp add:g_def*)
ultimately show *?thesis* **unfolding** *g_def* **using** ‹*open s*›
by (*auto elim*!: *no_isolated_singularity*)
qed

lemma *not_is_pole_holomorphic*:
assumes *open A* $x \in A$ *f holomorphic_on A*
shows $\neg is\_pole\ f\ x$
proof −
have *continuous_on A f* **by** (*intro holomorphic_on_imp_continuous_on*) *fact*
with *assms* **have** *isCont f x* **by** (*simp add*: *continuous_on_eq_continuous_at*)
hence $f \longrightarrow f\ x$ **by** (*simp add*: *isCont_def*)

```
  thus ¬is_pole f x unfolding is_pole_def
    using not_tendsto_and_filterlim_at_infinity[of at x f f x] by auto
qed

lemma is_pole_inverse_power: n > 0 ⟹ is_pole (λz::complex. 1 / (z − a) ^
n) a
  unfolding is_pole_def inverse_eq_divide [symmetric]
  by (intro filterlim_compose[OF filterlim_inverse_at_infinity] tendsto_intros)
     (auto simp: filterlim_at eventually_at intro!: exI[of _ 1] tendsto_eq_intros)

lemma is_pole_inverse: is_pole (λz::complex. 1 / (z − a)) a
  using is_pole_inverse_power[of 1 a] by simp

lemma is_pole_divide:
  fixes f :: 'a :: t2_space ⇒ 'b :: real_normed_field
  assumes isCont f z filterlim g (at 0) (at z) f z ≠ 0
  shows   is_pole (λz. f z / g z) z
proof −
  have filterlim (λz. f z ∗ inverse (g z)) at_infinity (at z)
    by (intro tendsto_mult_filterlim_at_infinity[of _ f z]
              filterlim_compose[OF filterlim_inverse_at_infinity])+
       (insert assms, auto simp: isCont_def)
  thus ?thesis by (simp add: field_split_simps is_pole_def)
qed

lemma is_pole_basic:
  assumes f holomorphic_on A open A z ∈ A f z ≠ 0 n > 0
  shows   is_pole (λw. f w / (w − z) ^ n) z
proof (rule is_pole_divide)
  have continuous_on A f by (rule holomorphic_on_imp_continuous_on) fact
  with assms show isCont f z by (auto simp: continuous_on_eq_continuous_at)
  have filterlim (λw. (w − z) ^ n) (nhds 0) (at z)
    using assms by (auto intro!: tendsto_eq_intros)
  thus filterlim (λw. (w − z) ^ n) (at 0) (at z)
    by (intro filterlim_atI tendsto_eq_intros)
       (insert assms, auto simp: eventually_at_filter)
qed fact+

lemma is_pole_basic':
  assumes f holomorphic_on A open A 0 ∈ A f 0 ≠ 0 n > 0
  shows   is_pole (λw. f w / w ^ n) 0
  using is_pole_basic[of f A 0] assms by simp
```

The proposition $\exists\, x.\ f\ -z\to\ x\ \vee$ *is_pole f z* can be interpreted as the complex function f has a non-essential singularity at z (i.e. the singularity is either removable or a pole).

```
definition not_essential::[complex ⇒ complex, complex] ⇒ bool where
  not_essential f z = (∃ x. f−z→x ∨ is_pole f z)
```

definition *isolated_singularity_at*::[*complex* ⇒ *complex*, *complex*] ⇒ *bool* **where**
isolated_singularity_at f z = (∃ *r*>*0*. *f analytic_on ball z r*−{*z*})

named_theorems *singularity_intros introduction rules for singularities*

lemma *holomorphic_factor_unique*:
fixes *f*::*complex* ⇒ *complex* **and** *z*::*complex* **and** *r*::*real* **and** *m n*::*int*
assumes *r*>*0 g z*≠*0 h z*≠*0*
and *asm*:∀ *w*∈*ball z r*−{*z*}. *f w* = *g w* ∗ (*w*−*z*) *powr n* ∧ *g w*≠*0* ∧ *f w* = *h w* ∗ (*w* − *z*) *powr m* ∧ *h w*≠*0*
and *g_holo*:*g holomorphic_on ball z r* **and** *h_holo*:*h holomorphic_on ball z r*
shows *n*=*m*
proof −
have [*simp*]:*at z within ball z r* ≠ *bot* **using** ‹*r*>*0*›
by (*auto simp add*:*at_within_ball_bot_iff*)
have *False* **when** *n*>*m*
proof −
have (*h* ⟶ *0*) (*at z within ball z r*)
proof (*rule Lim_transform_within*[*OF* _ ‹*r*>*0*›, **where** *f*=λ*w*. (*w* − *z*) *powr* (*n* − *m*) ∗ *g w*])
have ∀ *w*∈*ball z r*−{*z*}. *h w* = (*w*−*z*)*powr*(*n*−*m*) ∗ *g w*
using ‹*n*>*m*› *asm* ‹*r*>*0*›
apply (*auto simp add*:*field_simps powr_diff*)
by *force*
then show ⟦*x*′ ∈ *ball z r*; *0* < *dist x*′ *z*;*dist x*′ *z* < *r*⟧
⟹ (*x*′ − *z*) *powr* (*n* − *m*) ∗ *g x*′ = *h x*′ **for** *x*′ **by** *auto*
next
define *F* **where** *F* ≡ *at z within ball z r*
define *f*′ **where** *f*′ ≡ λ*x*. (*x* − *z*) *powr* (*n*−*m*)
have *f*′ *z*=*0* **using** ‹*n*>*m*› **unfolding** *f*′*_def* **by** *auto*
moreover have *continuous F f*′ **unfolding** *f*′*_def F_def continuous_def*
apply (*subst Lim_ident_at*)
using ‹*n*>*m*› **by** (*auto intro*!:*tendsto_powr_complex_0 tendsto_eq_intros*)
ultimately have (*f*′ ⟶ *0*) *F* **unfolding** *F_def*
by (*simp add*: *continuous_within*)
moreover have (*g* ⟶ *g z*) *F*
using *holomorphic_on_imp_continuous_on*[*OF g_holo*,*unfolded continuous_on_def*] ‹*r*>*0*›
unfolding *F_def* **by** *auto*
ultimately show ((λ*w*. *f*′ *w* ∗ *g w*) ⟶ *0*) *F* **using** *tendsto_mult* **by** *fastforce*
qed
moreover have (*h* ⟶ *h z*) (*at z within ball z r*)
using *holomorphic_on_imp_continuous_on*[*OF h_holo*]
by (*auto simp add*:*continuous_on_def* ‹*r*>*0*›)
ultimately have *h z*=*0* **by** (*auto intro*!: *tendsto_unique*)
thus *False* **using** ‹*h z*≠*0*› **by** *auto*
qed
moreover have *False* **when** *m*>*n*

```
  proof −
    have (g ⟶ 0) (at z within ball z r)
    proof (rule Lim_transform_within[OF _ ‹r>0›, where f=λw. (w − z) powr
(m − n) ∗ h w])
      have ∀ w∈ball z r −{z}. g w = (w−z) powr (m−n) ∗ h w using ‹m>n› asm
        apply (auto simp add:field_simps powr_diff)
        by force
      then show ⟦x′ ∈ ball z r; 0 < dist x′ z;dist x′ z < r⟧
            ⟹ (x′ − z) powr (m − n) ∗ h x′ = g x′ for x′ by auto
    next
      define F where F ≡ at z within ball z r
      define f′ where f′ ≡λx. (x − z) powr (m−n)
      have f′ z=0 using ‹m>n› unfolding f′_def by auto
      moreover have continuous F f′ unfolding f′_def F_def continuous_def
        apply (subst Lim_ident_at)
        using ‹m>n› by (auto intro!:tendsto_powr_complex_0 tendsto_eq_intros)
      ultimately have (f′ ⟶ 0) F unfolding F_def
        by (simp add: continuous_within)
      moreover have (h ⟶ h z) F
        using holomorphic_on_imp_continuous_on[OF h_holo,unfolded continu-
ous_on_def] ‹r>0›
        unfolding F_def by auto
      ultimately show ((λw. f′ w ∗ h w) ⟶ 0) F using tendsto_mult by
fastforce
    qed
    moreover have (g ⟶ g z) (at z within ball z r)
      using holomorphic_on_imp_continuous_on[OF g_holo]
      by (auto simp add:continuous_on_def ‹r>0›)
    ultimately have g z=0 by (auto intro!: tendsto_unique)
    thus False using ‹g z≠0› by auto
  qed
  ultimately show n=m by fastforce
qed

lemma holomorphic_factor_puncture:
  assumes f_iso:isolated_singularity_at f z
      and not_essential f z — f has either a removable singularity or a pole at z
      and non_zero:∃ F w in (at z). f w≠0 — f will not be constantly zero in a
neighbour of z
  shows ∃!n::int. ∃ g r. 0 < r ∧ g holomorphic_on cball z r ∧ g z≠0
        ∧ (∀ w∈cball z r−{z}. f w = g w ∗ (w−z) powr n ∧ g w≠0)
proof −
  define P where P = (λf n g r. 0 < r ∧ g holomorphic_on cball z r ∧ g z≠0
        ∧ (∀ w∈cball z r − {z}. f w = g w ∗ (w−z) powr (of_int n) ∧ g w≠0))
  have imp_unique:∃!n::int. ∃ g r. P f n g r when ∃ n g r. P f n g r
  proof (rule ex_ex1I[OF that])
    fix n1 n2 :: int
    assume g1_asm:∃ g1 r1. P f n1 g1 r1 and g2_asm:∃ g2 r2. P f n2 g2 r2
    define fac where fac ≡ λn g r. ∀ w∈cball z r−{z}. f w = g w ∗ (w − z) powr
```

```
(of_int n) ∧ g w ≠ 0
    obtain g1 r1 where 0 < r1 and g1_holo: g1 holomorphic_on cball z r1 and
g1 z≠0
        and fac n1 g1 r1 using g1_asm unfolding P_def fac_def by auto
    obtain g2 r2 where 0 < r2 and g2_holo: g2 holomorphic_on cball z r2 and
g2 z≠0
        and fac n2 g2 r2 using g2_asm unfolding P_def fac_def by auto
    define r where r ≡ min r1 r2
    have r>0 using ‹r1>0› ‹r2>0› unfolding r_def by auto
    moreover have ∀ w∈ball z r−{z}. f w = g1 w * (w−z) powr n1 ∧ g1 w≠0
        ∧ f w = g2 w * (w − z) powr n2  ∧ g2 w≠0
      using ‹fac n1 g1 r1› ‹fac n2 g2 r2›   unfolding fac_def r_def
      by fastforce
    ultimately show n1=n2 using g1_holo g2_holo ‹g1 z≠0› ‹g2 z≠0›
      apply (elim holomorphic_factor_unique)
      by (auto simp add:r_def)
  qed

  have P_exist:∃ n g r. P h n g r when
      ∃ z'. (h ⟶ z') (at z) isolated_singularity_at h z  ∃ F w in (at z). h w≠0
    for h
  proof −
    from that(2) obtain r where r>0 h analytic_on ball z r − {z}
      unfolding isolated_singularity_at_def by auto
    obtain z' where (h ⟶ z') (at z) using ‹∃ z'. (h ⟶ z') (at z)› by auto
    define h' where h'=(λx. if x=z then z' else h x)
    have h' holomorphic_on ball z r
      apply (rule no_isolated_singularity'[of {z}])
       subgoal by (metis LIM_equal Lim_at_imp_Lim_at_within ‹h −z→ z'›
empty_iff h'_def insert_iff)
       subgoal using ‹h analytic_on ball z r − {z}› analytic_imp_holomorphic
h'_def holomorphic_transform
        by fastforce
      by auto
    have ?thesis when z'=0
    proof −
      have h' z=0 using that unfolding h'_def by auto
      moreover have ¬ h' constant_on ball z r
        using ‹∃ F w in (at z). h w≠0› unfolding constant_on_def frequently_def
eventually_at h'_def
        apply simp
        by (metis ‹0 < r› centre_in_ball dist_commute mem_ball that)
      moreover note ‹h' holomorphic_on ball z r›
      ultimately obtain g r1 n where 0 < n 0 < r1 ball z r1 ⊆ ball z r and
          g:g holomorphic_on ball z r1
          ⋀w. w ∈ ball z r1 ⟹ h' w = (w − z) ^ n * g w
          ⋀w. w ∈ ball z r1 ⟹ g w ≠ 0
       using holomorphic_factor_zero_nonconstant[of _ ball z r z thesis,simplified,
              OF ‹h' holomorphic_on ball z r› ‹r>0› ‹h' z=0› ‹¬ h' constant_on
```

```
ball z r›]
      by (auto simp add:dist_commute)
    define rr where rr=r1/2
    have P h' n g rr
      unfolding P_def rr_def
      using ‹n>0› ‹r1>0› g by (auto simp add:powr_nat)
    then have P h n g rr
      unfolding h'_def P_def by auto
    then show ?thesis unfolding P_def by blast
  qed
  moreover have ?thesis when z'≠0
  proof −
    have h' z≠0 using that unfolding h'_def by auto
    obtain r1 where r1>0 cball z r1 ⊆ ball z r ∀x∈cball z r1. h' x≠0
    proof −
      have isCont h' z h' z≠0
        by (auto simp add: Lim_cong_within ‹h −z→ z'› ‹z'≠0› continuous_at
h'_def)
      then obtain r2 where r2:r2>0 ∀x∈ball z r2. h' x≠0
        using continuous_at_avoid[of z h' 0 ] unfolding ball_def by auto
      define r1 where r1=min r2 r / 2
      have 0 < r1 cball z r1 ⊆ ball z r
        using ‹r2>0› ‹r>0› unfolding r1_def by auto
      moreover have ∀x∈cball z r1. h' x ≠ 0
        using r2 unfolding r1_def by simp
      ultimately show ?thesis using that by auto
    qed
   then have P h' 0 h' r1 using ‹h' holomorphic_on ball z r› unfolding P_def
by auto
    then have P h 0 h' r1 unfolding P_def h'_def by auto
    then show ?thesis unfolding P_def by blast
  qed
  ultimately show ?thesis by auto
 qed

 have ?thesis when ∃x. (f ⟶ x) (at z)
  apply (rule_tac imp_unique[unfolded P_def])
  using P_exist[OF that(1) f_iso non_zero] unfolding P_def .
 moreover have ?thesis when is_pole f z
 proof (rule imp_unique[unfolded P_def])
  obtain e where [simp]:e>0 and e_holo:f holomorphic_on ball z e − {z} and
e_nz: ∀x∈ball z e−{z}. f x≠0
  proof −
    have ∀F x in at z. f z ≠ 0
      using ‹is_pole f z› filterlim_at_infinity_imp_eventually_ne unfolding
is_pole_def
      by auto
    then obtain e1 where e1:e1>0 ∀x∈ball z e1−{z}. f x≠0
      using that eventually_at[of λx. f x≠0 z UNIV,simplified] by (auto simp
```

```
add:dist_commute)
    obtain e2 where e2:e2>0 f holomorphic_on ball z e2 - {z}
    using f_iso analytic_imp_holomorphic unfolding isolated_singularity_at_def
by auto
    define e where e=min e1 e2
    show ?thesis
      apply (rule that[of e])
      using  e1 e2 unfolding e_def by auto
  qed

  define h where h ≡ λx. inverse (f x)

  have ∃ n g r. P h n g r
  proof -
    have h -z→ 0
      using Lim_transform_within_open assms(2) h_def is_pole_tendsto that
by fastforce
    moreover have ∃ F w in (at z). h w≠0
      using non_zero
      apply (elim frequently_rev_mp)
      unfolding h_def eventually_at by (auto intro:exI[where x=1])
    moreover have isolated_singularity_at h z
      unfolding isolated_singularity_at_def h_def
      apply (rule exI[where x=e])
      using e_holo e_nz ‹e>0› by (metis open_ball analytic_on_open
        holomorphic_on_inverse open_delete)
    ultimately show ?thesis
      using P_exist[of h] by auto
  qed
  then obtain n g r
    where 0 < r and
        g_holo:g holomorphic_on cball z r and g z≠0 and
        g_fac:(∀ w∈cball z r-{z}. h w = g w * (w - z) powr of_int n  ∧ g w ≠
0)
    unfolding P_def by auto
  have P f (-n) (inverse o g) r
  proof -
    have f w = inverse (g w) * (w - z) powr of_int (- n) when w∈cball z r -
{z} for w
      using g_fac[rule_format,of w] that unfolding h_def
      apply (auto simp add:powr_minus )
      by (metis inverse_inverse_eq inverse_mult_distrib)
    then show ?thesis
      unfolding P_def comp_def
      using ‹r>0› g_holo g_fac ‹g z≠0› by (auto intro:holomorphic_intros)
  qed
  then show ∃ x g r. 0 < r ∧ g holomorphic_on cball z r ∧ g z ≠ 0
            ∧ (∀ w∈cball z r - {z}. f w = g w * (w - z) powr of_int x  ∧ g w
≠ 0)
```

```
    unfolding P_def by blast
  qed
  ultimately show ?thesis using ‹not_essential f z› unfolding not_essential_def
by presburger
qed

lemma not_essential_transform:
  assumes not_essential g z
  assumes ∀F w in (at z). g w = f w
  shows not_essential f z
  using assms unfolding not_essential_def
  by (simp add: filterlim_cong is_pole_cong)

lemma isolated_singularity_at_transform:
  assumes isolated_singularity_at g z
  assumes ∀F w in (at z). g w = f w
  shows isolated_singularity_at f z
proof −
  obtain r1 where r1>0 and r1:g analytic_on ball z r1 − {z}
    using assms(1) unfolding isolated_singularity_at_def by auto
  obtain r2 where r2>0 and r2: ∀x. x ≠ z ∧ dist x z < r2 ⟶ g x = f x
    using assms(2) unfolding eventually_at by auto
  define r3 where r3=min r1 r2
  have r3>0 unfolding r3_def using ‹r1>0› ‹r2>0› by auto
  moreover have f analytic_on ball z r3 − {z}
  proof −
    have g holomorphic_on ball z r3 − {z}
      using r1 unfolding r3_def by (subst (asm) analytic_on_open,auto)
    then have f holomorphic_on ball z r3 − {z}
      using r2 unfolding r3_def
      by (auto simp add:dist_commute elim!:holomorphic_transform)
    then show ?thesis by (subst analytic_on_open,auto)
  qed
  ultimately show ?thesis unfolding isolated_singularity_at_def by auto
qed

lemma not_essential_powr[singularity_intros]:
  assumes LIM w (at z). f w :> (at x)
  shows not_essential (λw. (f w) powr (of_int n)) z
proof −
  define fp where fp=(λw. (f w) powr (of_int n))
  have ?thesis when n>0
  proof −
    have (λw. (f w) ^ (nat n)) −z→ x ^ nat n
      using that assms unfolding filterlim_at by (auto intro!:tendsto_eq_intros)
    then have fp −z→ x ^ nat n unfolding fp_def
      apply (elim Lim_transform_within[where d=1],simp)
      by (metis less_le powr_0 powr_of_int that zero_less_nat_eq zero_power)
    then show ?thesis unfolding not_essential_def fp_def by auto
```

```
  qed
  moreover have ?thesis when n=0
  proof −
    have fp −z→ 1
      apply (subst tendsto_cong[where g=λ_.1])
      using that filterlim_at_within_not_equal[OF assms,of 0] unfolding fp_def
by auto
    then show ?thesis unfolding fp_def not_essential_def by auto
  qed
  moreover have ?thesis when n<0
  proof (cases x=0)
    case True
    have LIM w (at z). inverse ((f w) ^ (nat (−n))) :> at_infinity
      apply (subst filterlim_inverse_at_iff[symmetric],simp)
      apply (rule filterlim_atI)
     subgoal using assms True that unfolding filterlim_at by (auto intro!:tendsto_eq_intros)
      subgoal using filterlim_at_within_not_equal[OF assms,of 0]
        by (eventually_elim,insert that,auto)
      done
    then have LIM w (at z). fp w :> at_infinity
    proof (elim filterlim_mono_eventually)
      show ∀F x in at z. inverse (f x ^ nat (− n)) = fp x
        using filterlim_at_within_not_equal[OF assms,of 0] unfolding fp_def
        apply eventually_elim
        using powr_of_int that by auto
    qed auto
    then show ?thesis unfolding fp_def not_essential_def is_pole_def by auto
  next
    case False
    let ?xx= inverse (x ^ (nat (−n)))
    have (λw. inverse ((f w) ^ (nat (−n)))) −z→?xx
     using assms False unfolding filterlim_at by (auto intro!:tendsto_eq_intros)
    then have fp −z→?xx
      apply (elim Lim_transform_within[where d=1],simp)
       unfolding fp_def by (metis inverse_zero nat_mono_iff nat_zero_as_int
neg_0_less_iff_less
         not_le power_eq_0_iff powr_0 powr_of_int that)
    then show ?thesis unfolding fp_def not_essential_def by auto
  qed
  ultimately show ?thesis by linarith
qed

lemma isolated_singularity_at_powr[singularity_intros]:
  assumes isolated_singularity_at f z ∀F w in (at z). f w≠0
  shows isolated_singularity_at (λw. (f w) powr (of_int n)) z
proof −
  obtain r1 where r1>0 f analytic_on ball z r1 − {z}
    using assms(1) unfolding isolated_singularity_at_def by auto
  then have r1:f holomorphic_on ball z r1 − {z}
```

```
    using analytic_on_open[of ball z r1−{z} f] by blast
  obtain r2 where r2>0 and r2:∀ w. w ≠ z ∧ dist w z < r2 ⟶ f w ≠ 0
    using assms(2) unfolding eventually_at by auto
  define r3 where r3=min r1 r2
  have (λw. (f w) powr of_int n) holomorphic_on ball z r3 − {z}
    apply (rule holomorphic_on_powr_of_int)
    subgoal unfolding r3_def using r1 by auto
    subgoal unfolding r3_def using r2 by (auto simp add:dist_commute)
    done
  moreover have r3>0 unfolding r3_def using ‹0 < r1› ‹0 < r2› by linarith
  ultimately show ?thesis unfolding isolated_singularity_at_def
    apply (subst (asm) analytic_on_open[symmetric])
    by auto
qed

lemma non_zero_neighbour:
  assumes f_iso:isolated_singularity_at f z
      and f_ness:not_essential f z
      and f_nconst:∃ F w in (at z). f w≠0
    shows ∀ F w in (at z). f w≠0
proof −
  obtain fn fp fr where [simp]:fp z ≠ 0 and fr > 0
        and fr: fp holomorphic_on cball z fr
              ∀ w∈cball z fr − {z}. f w = fp w ∗ (w − z) powr of_int fn ∧ fp w
≠ 0
   using holomorphic_factor_puncture[OF f_iso f_ness f_nconst,THEN ex1_implies_ex]
by auto
  have f w ≠ 0 when  w ≠ z dist w z < fr for w
  proof −
    have f w = fp w ∗ (w − z) powr of_int fn fp w ≠ 0
      using fr(2)[rule_format, of w] using that by (auto simp add:dist_commute)
    moreover have (w − z) powr of_int fn ≠0
      unfolding powr_eq_0_iff using ‹w≠z› by auto
    ultimately show ?thesis by auto
  qed
  then show ?thesis using ‹fr>0› unfolding eventually_at by auto
qed

lemma non_zero_neighbour_pole:
  assumes is_pole f z
  shows ∀ F w in (at z). f w≠0
  using assms filterlim_at_infinity_imp_eventually_ne[of f at z 0]
  unfolding is_pole_def by auto

lemma non_zero_neighbour_alt:
  assumes holo: f holomorphic_on S
      and open S connected S z ∈ S  β ∈ S f β ≠ 0
    shows ∀ F w in (at z). f w≠0 ∧ w∈S
proof (cases f z = 0)
```

```
  case True
  from isolated_zeros[OF holo ‹open S› ‹connected S› ‹z ∈ S› True ‹β ∈ S› ‹f β
≠ 0›]
  obtain r where 0 < r ball z r ⊆ S ∀ w ∈ ball z r − {z}.f w ≠ 0 by metis
  then show ?thesis unfolding eventually_at
    apply (rule_tac x=r in exI)
    by (auto simp add:dist_commute)
next
  case False
  obtain r1 where r1:r1>0 ∀ y. dist z y < r1 ⟶ f y ≠ 0
   using continuous_at_avoid[of z f, OF _ False] assms(2,4) continuous_on_eq_continuous_at
      holo holomorphic_on_imp_continuous_on by blast
  obtain r2 where r2:r2>0 ball z r2 ⊆ S
    using assms(2) assms(4) openE by blast
  show ?thesis unfolding eventually_at
    apply (rule_tac x=min r1 r2 in exI)
    using r1 r2 by (auto simp add:dist_commute)
qed

lemma not_essential_times[singularity_intros]:
  assumes f_ness:not_essential f z and g_ness:not_essential g z
  assumes f_iso:isolated_singularity_at f z and g_iso:isolated_singularity_at g
z
  shows not_essential (λw. f w * g w) z
proof −
  define fg where fg = (λw. f w * g w)
  have ?thesis when ¬ ((∃ F w in (at z). f w≠0) ∧ (∃ F w in (at z). g w≠0))
  proof −
    have ∀ F w in (at z). fg w=0
      using that[unfolded frequently_def, simplified] unfolding fg_def
      by (auto elim: eventually_rev_mp)
    from tendsto_cong[OF this] have fg −z→0 by auto
    then show ?thesis unfolding not_essential_def fg_def by auto
  qed
 moreover have ?thesis when f_nconst:∃ F w in (at z). f w≠0 and g_nconst:∃ F w
in (at z). g w≠0
  proof −
    obtain fn fp fr where [simp]:fp z ≠ 0 and fr > 0
        and fr: fp holomorphic_on cball z fr
              ∀ w∈cball z fr − {z}. f w = fp w * (w − z) powr of_int fn ∧ fp w
≠ 0
     using holomorphic_factor_puncture[OF f_iso f_ness f_nconst,THEN ex1_implies_ex]
by auto
    obtain gn gp gr where [simp]:gp z ≠ 0 and gr > 0
        and gr: gp holomorphic_on cball z gr
              ∀ w∈cball z gr − {z}. g w = gp w * (w − z) powr of_int gn ∧ gp w
≠ 0
     using holomorphic_factor_puncture[OF g_iso g_ness g_nconst,THEN ex1_implies_ex]
by auto
```

```
    define r1 where r1=(min fr gr)
    have r1>0 unfolding r1_def using ‹fr>0› ‹gr>0› by auto
     have fg_times:fg w = (fp w * gp w) * (w - z) powr (of_int (fn+gn)) and
fgp_nz:fp w*gp w≠0
      when w∈ball z r1 - {z} for w
    proof -
      have f w = fp w * (w - z) powr of_int fn fp w≠0
        using fr(2)[rule_format,of w] that unfolding r1_def by auto
      moreover have g w = gp w * (w - z) powr of_int gn gp w ≠ 0
        using gr(2)[rule_format, of w] that unfolding r1_def by auto
      ultimately show fg w = (fp w * gp w) * (w - z) powr (of_int (fn+gn)) fp
w*gp w≠0
        unfolding fg_def by (auto simp add:powr_add)
    qed

    have [intro]: fp -z→fp z gp -z→gp z
        using fr(1) ‹fr>0› gr(1) ‹gr>0›
        by (meson open_ball ball_subset_cball centre_in_ball
         continuous_on_eq_continuous_at continuous_within holomorphic_on_imp_continuous_on
            holomorphic_on_subset)+
    have ?thesis when fn+gn>0
    proof -
      have (λw. (fp w * gp w) * (w - z) ^(nat (fn+gn))) -z→0
        using that by (auto intro!:tendsto_eq_intros)
      then have fg -z→ 0
        apply (elim Lim_transform_within[OF _ ‹r1>0›])
         by (metis (no_types, opaque_lifting) Diff_iff cball_trivial dist_commute
dist_self
           eq_iff_diff_eq_0 fg_times less_le linorder_not_le mem_ball mem_cball
powr_of_int
             that)
      then show ?thesis unfolding not_essential_def fg_def by auto
    qed
    moreover have ?thesis when fn+gn=0
    proof -
      have (λw. fp w * gp w) -z→fp z*gp z
        using that by (auto intro!:tendsto_eq_intros)
      then have fg -z→ fp z*gp z
        apply (elim Lim_transform_within[OF _ ‹r1>0›])
        apply (subst fg_times)
        by (auto simp add:dist_commute that)
      then show ?thesis unfolding not_essential_def fg_def by auto
    qed
    moreover have ?thesis when fn+gn<0
    proof -
      have LIM w (at z). fp w * gp w / (w-z)^nat (-(fn+gn)) :> at_infinity
        apply (rule filterlim_divide_at_infinity)
        apply (insert that, auto intro!:tendsto_eq_intros filterlim_atI)
```

```
      using eventually_at_topological by blast
    then have is_pole fg z unfolding is_pole_def
      apply (elim filterlim_transform_within[OF _ _ ‹r1>0›],simp)
      apply (subst fg_times,simp add:dist_commute)
      apply (subst powr_of_int)
      using that by (auto simp add:field_split_simps)
    then show ?thesis unfolding not_essential_def fg_def by auto
  qed
  ultimately show ?thesis unfolding not_essential_def fg_def by fastforce
 qed
 ultimately show ?thesis by auto
qed

lemma not_essential_inverse[singularity_intros]:
 assumes f_ness:not_essential f z
 assumes f_iso:isolated_singularity_at f z
 shows not_essential (λw. inverse (f w)) z
proof −
 define vf where vf = (λw. inverse (f w))
 have ?thesis when ¬(∃ F w in (at z). f w≠0)
 proof −
  have ∀ F w in (at z). f w=0
   using that[unfolded frequently_def, simplified] by (auto elim: eventually_rev_mp)
  then have ∀ F w in (at z). vf w=0
   unfolding vf_def by auto
  from tendsto_cong[OF this] have vf −z→0 unfolding vf_def by auto
  then show ?thesis unfolding not_essential_def vf_def by auto
 qed
 moreover have ?thesis when is_pole f z
 proof −
  have vf −z→0
   using that filterlim_at filterlim_inverse_at_iff unfolding is_pole_def vf_def
by blast
  then show ?thesis unfolding not_essential_def vf_def by auto
 qed
 moreover have ?thesis when ∃ x. f−z→x and f_nconst:∃ F w in (at z). f w≠0
 proof −
  from that obtain fz where fz:f−z→fz by auto
  have ?thesis when fz=0
  proof −
   have (λw. inverse (vf w)) −z→0
    using fz that unfolding vf_def by auto
   moreover have ∀ F w in at z. inverse (vf w) ≠ 0
    using non_zero_neighbour[OF f_iso f_ness f_nconst]
    unfolding vf_def by auto
   ultimately have is_pole vf z
    using filterlim_inverse_at_iff[of vf at z] unfolding filterlim_at is_pole_def
by auto
   then show ?thesis unfolding not_essential_def vf_def by auto
```

```
    qed
    moreover have ?thesis when fz≠0
    proof −
      have vf −z→inverse fz
        using fz that unfolding vf_def by (auto intro:tendsto_eq_intros)
      then show ?thesis unfolding not_essential_def vf_def by auto
    qed
    ultimately show ?thesis by auto
  qed
  ultimately show ?thesis using f_ness unfolding not_essential_def by auto
qed

lemma isolated_singularity_at_inverse[singularity_intros]:
  assumes f_iso:isolated_singularity_at f z
      and f_ness:not_essential f z
  shows isolated_singularity_at (λw. inverse (f w)) z
proof −
  define vf where vf = (λw. inverse (f w))
  have ?thesis when ¬(∃ F w in (at z). f w≠0)
  proof −
    have ∀ F w in (at z). f w=0
    using that[unfolded frequently_def, simplified] by (auto elim: eventually_rev_mp)
    then have ∀ F w in (at z). vf w=0
      unfolding vf_def by auto
    then obtain d1 where d1>0 and d1:∀ x. x ≠ z ∧ dist x z < d1 ⟶ vf x = 0
      unfolding eventually_at by auto
    then have vf holomorphic_on ball z d1−{z}
      apply (rule_tac holomorphic_transform[of λ_. 0])
      by (auto simp add:dist_commute)
    then have vf analytic_on ball z d1 − {z}
      by (simp add: analytic_on_open open_delete)
      then show ?thesis using ‹d1>0› unfolding isolated_singularity_at_def
vf_def by auto
  qed
  moreover have ?thesis when f_nconst:∃ F w in (at z). f w≠0
  proof −
     have ∀ F w in at z. f w ≠ 0 using non_zero_neighbour[OF f_iso f_ness
f_nconst] .
    then obtain d1 where d1:d1>0 ∀ x. x ≠ z ∧ dist x z < d1 ⟶ f x ≠ 0
      unfolding eventually_at by auto
    obtain d2 where d2>0 and d2:f analytic_on ball z d2 − {z}
      using f_iso unfolding isolated_singularity_at_def by auto
    define d3 where d3=min d1 d2
    have d3>0 unfolding d3_def using ‹d1>0› ‹d2>0› by auto
    moreover have vf analytic_on ball z d3 − {z}
      unfolding vf_def
      apply (rule analytic_on_inverse)
      subgoal using d2 unfolding d3_def by (elim analytic_on_subset) auto
     subgoal for w using d1 unfolding d3_def by (auto simp add:dist_commute)
```

```
    done
   ultimately show ?thesis unfolding isolated_singularity_at_def vf_def by
auto
 qed
 ultimately show ?thesis by auto
qed

lemma not_essential_divide[singularity_intros]:
 assumes f_ness:not_essential f z and g_ness:not_essential g z
 assumes f_iso:isolated_singularity_at f z and g_iso:isolated_singularity_at g
z
 shows not_essential (λw. f w / g w) z
proof −
 have not_essential (λw. f w ∗ inverse (g w)) z
  apply (rule not_essential_times[where g=λw. inverse (g w)])
  using assms by (auto intro: isolated_singularity_at_inverse not_essential_inverse)
 then show ?thesis by (simp add:field_simps)
qed

lemma
 assumes f_iso:isolated_singularity_at f z
    and g_iso:isolated_singularity_at g z
   shows isolated_singularity_at_times[singularity_intros]:
           isolated_singularity_at (λw. f w ∗ g w) z and
         isolated_singularity_at_add[singularity_intros]:
           isolated_singularity_at (λw. f w + g w) z
proof −
 obtain d1 d2 where d1>0 d2>0
    and d1:f analytic_on ball z d1 − {z} and d2:g analytic_on ball z d2 − {z}
   using f_iso g_iso unfolding isolated_singularity_at_def by auto
 define d3 where d3=min d1 d2
 have d3>0 unfolding d3_def using ‹d1>0› ‹d2>0› by auto

 have (λw. f w ∗ g w) analytic_on ball z d3 − {z}
  apply (rule analytic_on_mult)
  using d1 d2 unfolding d3_def by (auto elim:analytic_on_subset)
 then show isolated_singularity_at (λw. f w ∗ g w) z
  using ‹d3>0› unfolding isolated_singularity_at_def by auto
 have (λw. f w + g w) analytic_on ball z d3 − {z}
  apply (rule analytic_on_add)
  using d1 d2 unfolding d3_def by (auto elim:analytic_on_subset)
 then show isolated_singularity_at (λw. f w + g w) z
  using ‹d3>0› unfolding isolated_singularity_at_def by auto
qed

lemma isolated_singularity_at_uminus[singularity_intros]:
 assumes f_iso:isolated_singularity_at f z
 shows isolated_singularity_at (λw. − f w) z
 using assms unfolding isolated_singularity_at_def using analytic_on_neg by
```

blast

lemma *isolated_singularity_at_id*[*singularity_intros*]:
isolated_singularity_at (λ*w*. *w*) *z*
unfolding *isolated_singularity_at_def* **by** (*simp add*: *gt_ex*)

lemma *isolated_singularity_at_minus*[*singularity_intros*]:
assumes *f_iso*:*isolated_singularity_at f z*
and *g_iso*:*isolated_singularity_at g z*
shows *isolated_singularity_at* (λ*w*. *f w* − *g w*) *z*
using *isolated_singularity_at_uminus*[*THEN isolated_singularity_at_add*[*OF f_iso*,*of* λ*w*. − *g w*]
,*OF g_iso*] **by** *simp*

lemma *isolated_singularity_at_divide*[*singularity_intros*]:
assumes *f_iso*:*isolated_singularity_at f z*
and *g_iso*:*isolated_singularity_at g z*
and *g_ness*:*not_essential g z*
shows *isolated_singularity_at* (λ*w*. *f w* / *g w*) *z*
using *isolated_singularity_at_inverse*[*THEN isolated_singularity_at_times*[*OF f_iso*,
of λ*w*. *inverse* (*g w*)],*OF g_iso g_ness*] **by** (*simp add*:*field_simps*)

lemma *isolated_singularity_at_const*[*singularity_intros*]:
isolated_singularity_at (λ*w*. *c*) *z*
unfolding *isolated_singularity_at_def* **by** (*simp add*: *gt_ex*)

lemma *isolated_singularity_at_holomorphic*:
assumes *f holomorphic_on s*−{*z*} *open s z*∈*s*
shows *isolated_singularity_at f z*
using *assms* **unfolding** *isolated_singularity_at_def*
by (*metis analytic_on_holomorphic centre_in_ball insert_Diff openE open_delete subset_insert_iff*)

5.10.1 The order of non-essential singularities (i.e. removable singularities or poles)

definition *zorder* :: (*complex* ⇒ *complex*) ⇒ *complex* ⇒ *int* **where**
zorder f z = (*THE n*. (∃ *h r*. *r*>0 ∧ *h holomorphic_on cball z r* ∧ *h z*≠0
∧ (∀ *w*∈*cball z r* − {*z*}. *f w* = *h w* ∗ (*w*−*z*) *powr* (*of_int n*)
∧ *h w* ≠0)))

definition *zor_poly*
::[*complex* ⇒ *complex*, *complex*] ⇒ *complex* ⇒ *complex* **where**
zor_poly f z = (*SOME h*. ∃ *r*. *r* > 0 ∧ *h holomorphic_on cball z r* ∧ *h z* ≠ 0
∧ (∀ *w*∈*cball z r* − {*z*}. *f w* = *h w* ∗ (*w* − *z*) *powr* (*zorder f z*)
∧ *h w* ≠0))

lemma *zorder_exist*:

```
  fixes f::complex ⇒ complex and z::complex
  defines n≡zorder f z and g≡zor_poly f z
  assumes f_iso:isolated_singularity_at f z
      and f_ness:not_essential f z
      and f_nconst:∃ F w in (at z). f w≠0
  shows g z≠0 ∧ (∃ r. r>0 ∧ g holomorphic_on cball z r
    ∧ (∀ w∈cball z r − {z}. f w  = g w * (w−z) powr n  ∧ g w ≠0))
proof −
  define P where P = (λn g r. 0 < r ∧ g holomorphic_on cball z r ∧ g z≠0
        ∧ (∀ w∈cball z r − {z}. f w = g w * (w−z) powr (of_int n) ∧ g w≠0))
  have ∃!n. ∃ g r. P n g r
   using holomorphic_factor_puncture[OF assms(3−)] unfolding P_def by auto
  then have ∃ g r. P n g r
    unfolding n_def P_def zorder_def
    by (drule_tac theI',argo)
  then have ∃ r. P n g r
    unfolding P_def zor_poly_def g_def n_def
    by (drule_tac someI_ex,argo)
  then obtain r1 where P n g r1 by auto
  then show ?thesis unfolding P_def by auto
qed

lemma
  fixes f::complex ⇒ complex and z::complex
  assumes f_iso:isolated_singularity_at f z
      and f_ness:not_essential f z
      and f_nconst:∃ F w in (at z). f w≠0
    shows zorder_inverse: zorder (λw. inverse (f w)) z = − zorder f z
      and zor_poly_inverse: ∀ F w in (at z). zor_poly (λw. inverse (f w)) z w
                                          = inverse (zor_poly f z w)
proof −
  define vf where vf = (λw. inverse (f w))
  define fn vfn where
    fn = zorder f z  and vfn = zorder vf z
  define fp vfp where
    fp = zor_poly f z and vfp = zor_poly vf z

  obtain fr where [simp]:fp z ≠ 0 and fr > 0
        and fr: fp holomorphic_on cball z fr
              ∀ w∈cball z fr − {z}. f w = fp w * (w − z) powr of_int fn ∧ fp w
≠ 0
    using zorder_exist[OF f_iso f_ness f_nconst,folded fn_def fp_def]
    by auto
  have fr_inverse: vf w = (inverse (fp w)) * (w − z) powr (of_int (−fn))
      and fr_nz: inverse (fp w)≠0
    when w∈ball z fr − {z} for w
  proof −
    have f w = fp w * (w − z) powr of_int fn fp w≠0
      using fr(2)[rule_format,of w] that by auto
```

```
    then show vf w = (inverse (fp w)) * (w - z) powr (of_int (-fn)) inverse (fp
w)≠0
      unfolding vf_def by (auto simp add:powr_minus)
  qed
  obtain vfr where [simp]:vfp z ≠ 0 and vfr>0 and vfr:vfp holomorphic_on cball
z vfr
      (∀ w∈cball z vfr - {z}. vf w = vfp w * (w - z) powr of_int vfn ∧ vfp w ≠ 0)
  proof -
    have isolated_singularity_at vf z
      using isolated_singularity_at_inverse[OF f_iso f_ness] unfolding vf_def .
    moreover have not_essential vf z
      using not_essential_inverse[OF f_ness f_iso] unfolding vf_def .
    moreover have ∃F w in at z. vf w ≠ 0
      using f_nconst unfolding vf_def by (auto elim:frequently_elim1)
    ultimately show ?thesis using zorder_exist[of vf z, folded vfn_def vfp_def]
that by auto
  qed


  define r1 where r1 = min fr vfr
  have r1>0 using ‹fr>0› ‹vfr>0› unfolding r1_def by simp
  show vfn = - fn
    apply (rule holomorphic_factor_unique[of r1 vfp z λw. inverse (fp w) vf])
    subgoal using ‹r1>0› by simp
    subgoal by simp
    subgoal by simp
    subgoal
    proof (rule ballI)
      fix w assume w ∈ ball z r1 - {z}
      then have w ∈ ball z fr - {z} w ∈ cball z vfr - {z}  unfolding r1_def by
auto
      from fr_inverse[OF this(1)] fr_nz[OF this(1)] vfr(2)[rule_format,OF this(2)]
      show vf w = vfp w * (w - z) powr of_int vfn ∧ vfp w ≠ 0
            ∧ vf w = inverse (fp w) * (w - z) powr of_int (- fn) ∧ inverse (fp w)
≠ 0 by auto
    qed
    subgoal using vfr(1) unfolding r1_def by (auto intro!:holomorphic_intros)
    subgoal using fr unfolding r1_def by (auto intro!:holomorphic_intros)
    done

  have vfp w = inverse (fp w) when w∈ball z r1-{z} for w
  proof -
    have w ∈ ball z fr - {z} w ∈ cball z vfr - {z}  w≠z using that unfolding
r1_def by auto
    from fr_inverse[OF this(1)] fr_nz[OF this(1)] vfr(2)[rule_format,OF this(2)]
‹vfn = - fn› ‹w≠z›
    show ?thesis by auto
  qed
  then show ∀F w in (at z). vfp w = inverse (fp w)
```

```
    unfolding eventually_at using ‹r1>0›
    apply (rule_tac x=r1 in exI)
    by (auto simp add:dist_commute)
qed

lemma
  fixes f g::complex ⇒ complex and z::complex
  assumes f_iso:isolated_singularity_at f z and g_iso:isolated_singularity_at g
z
      and f_ness:not_essential f z and g_ness:not_essential g z
      and fg_nconst: ∃ F w in (at z). f w * g w≠ 0
  shows zorder_times:zorder (λw. f w * g w) z = zorder f z + zorder g z and
        zor_poly_times:∀ F w in (at z). zor_poly (λw. f w * g w) z w
                                          = zor_poly f z w *zor_poly g z w
proof −
  define fg where fg = (λw. f w * g w)
  define fn gn fgn where
    fn = zorder f z and gn = zorder g z and fgn = zorder fg z
  define fp gp fgp where
    fp = zor_poly f z and gp = zor_poly g z and fgp = zor_poly fg z
  have f_nconst:∃ F w in (at z). f w ≠ 0 and g_nconst:∃ F w in (at z).g w≠ 0
    using fg_nconst by (auto elim!:frequently_elim1)
  obtain fr where [simp]:fp z ≠ 0 and fr > 0
        and fr: fp holomorphic_on cball z fr
              ∀ w∈cball z fr − {z}. f w = fp w * (w − z) powr of_int fn ∧ fp w
≠ 0
    using zorder_exist[OF f_iso f_ness f_nconst,folded fp_def fn_def] by auto
  obtain gr where [simp]:gp z ≠ 0 and gr > 0
        and gr: gp holomorphic_on cball z gr
              ∀ w∈cball z gr − {z}. g w = gp w * (w − z) powr of_int gn ∧ gp w
≠ 0
    using zorder_exist[OF g_iso g_ness g_nconst,folded gn_def gp_def] by auto
  define r1 where r1=min fr gr
  have r1>0 unfolding r1_def using ‹fr>0› ‹gr>0› by auto
  have fg_times:fg w = (fp w * gp w) * (w − z) powr (of_int (fn+gn)) and
fgp_nz:fp w*gp w≠0
    when w∈ball z r1 − {z} for w
  proof −
    have f w = fp w * (w − z) powr of_int fn fp w≠0
      using fr(2)[rule_format,of w] that unfolding r1_def by auto
    moreover have g w = gp w * (w − z) powr of_int gn gp w ≠ 0
      using gr(2)[rule_format, of w] that unfolding r1_def by auto
    ultimately show fg w = (fp w * gp w) * (w − z) powr (of_int (fn+gn)) fp
w*gp w≠0
      unfolding fg_def by (auto simp add:powr_add)
  qed

  obtain fgr where [simp]:fgp z ≠ 0 and fgr > 0
        and fgr: fgp holomorphic_on cball z fgr
```

```
                ∀ w∈cball z fgr − {z}. fg w = fgp w * (w − z) powr of_int fgn ∧
fgp w ≠ 0
  proof −
    have fgp z ≠ 0 ∧ (∃ r>0. fgp holomorphic_on cball z r
          ∧ (∀ w∈cball z r − {z}. fg w = fgp w * (w − z) powr of_int fgn ∧ fgp w
≠ 0))
      apply (rule zorder_exist[of fg z, folded fgn_def fgp_def])
       subgoal unfolding fg_def using isolated_singularity_at_times[OF f_iso
g_iso] .
       subgoal unfolding fg_def using not_essential_times[OF f_ness g_ness
f_iso g_iso] .
      subgoal unfolding fg_def using fg_nconst .
      done
    then show ?thesis using that by blast
  qed
  define r2 where r2 = min fgr r1
  have r2>0 using ‹r1>0› ‹fgr>0› unfolding r2_def by simp
  show fgn = fn + gn
    apply (rule holomorphic_factor_unique[of r2 fgp z λw. fp w * gp w fg])
    subgoal using ‹r2>0› by simp
    subgoal by simp
    subgoal by simp
    subgoal
    proof (rule ballI)
      fix w assume w ∈ ball z r2 − {z}
      then have w ∈ ball z r1 − {z} w ∈ cball z fgr − {z}  unfolding r2_def by
auto
     from fg_times[OF this(1)] fgp_nz[OF this(1)] fgr(2)[rule_format,OF this(2)]
      show fg w = fgp w * (w − z) powr of_int fgn ∧ fgp w ≠ 0
            ∧ fg w = fp w * gp w * (w − z) powr of_int (fn + gn) ∧ fp w * gp w
≠ 0 by auto
    qed
   subgoal using fgr(1) unfolding r2_def r1_def by (auto intro!:holomorphic_intros)
   subgoal using fr(1) gr(1) unfolding r2_def r1_def by (auto intro!:holomorphic_intros)
    done

  have fgp w = fp w *gp w when w∈ball z r2−{z} for w
  proof −
    have w ∈ ball z r1 − {z} w ∈ cball z fgr − {z} w≠z using that  unfolding
r2_def by auto
   from fg_times[OF this(1)] fgp_nz[OF this(1)] fgr(2)[rule_format,OF this(2)]
‹fgn = fn + gn› ‹w≠z›
    show ?thesis by auto
  qed
  then show ∀ F w in (at z). fgp w = fp w * gp w
    using ‹r2>0› unfolding eventually_at by (auto simp add:dist_commute)
qed

lemma
```

fixes *f g*::*complex* ⇒ *complex* **and** *z*::*complex*
assumes *f_iso*:*isolated_singularity_at f z* **and** *g_iso*:*isolated_singularity_at g z*
and *f_ness*:*not_essential f z* **and** *g_ness*:*not_essential g z*
and *fg_nconst*: ∃ *F w in* (*at z*). *f w* ∗ *g w*≠ *0*
shows *zorder_divide*:*zorder* (λ*w*. *f w* / *g w*) *z* = *zorder f z* − *zorder g z* **and**
zor_poly_divide:∀ *F w in* (*at z*). *zor_poly* (λ*w*. *f w* / *g w*) *z w*
= *zor_poly f z w* / *zor_poly g z w*
proof −
have *f_nconst*:∃ *F w in* (*at z*). *f w* ≠ *0* **and** *g_nconst*:∃ *F w in* (*at z*).*g w*≠ *0*
using *fg_nconst* **by** (*auto elim*!:*frequently_elim1*)
define *vg* **where** *vg*=(λ*w*. *inverse* (*g w*))
have *zorder* (λ*w*. *f w* ∗ *vg w*) *z* = *zorder f z* + *zorder vg z*
apply (*rule zorder_times*[*OF f_iso _ f_ness*,*of vg*])
subgoal unfolding *vg_def* **using** *isolated_singularity_at_inverse*[*OF g_iso g_ness*] .
subgoal unfolding *vg_def* **using** *not_essential_inverse*[*OF g_ness g_iso*] .
subgoal unfolding *vg_def* **using** *fg_nconst* **by** (*auto elim*!:*frequently_elim1*)
done
then show *zorder* (λ*w*. *f w* / *g w*) *z* = *zorder f z* − *zorder g z*
using *zorder_inverse*[*OF g_iso g_ness g_nconst*,*folded vg_def*] **unfolding** *vg_def*
by (*auto simp add*:*field_simps*)

have ∀ *F w in at z*. *zor_poly* (λ*w*. *f w* ∗ *vg w*) *z w* = *zor_poly f z w* ∗ *zor_poly vg z w*
apply (*rule zor_poly_times*[*OF f_iso _ f_ness*,*of vg*])
subgoal unfolding *vg_def* **using** *isolated_singularity_at_inverse*[*OF g_iso g_ness*] .
subgoal unfolding *vg_def* **using** *not_essential_inverse*[*OF g_ness g_iso*] .
subgoal unfolding *vg_def* **using** *fg_nconst* **by** (*auto elim*!:*frequently_elim1*)
done
then show ∀ *F w in* (*at z*). *zor_poly* (λ*w*. *f w* / *g w*) *z w* = *zor_poly f z w* / *zor_poly g z w*
using *zor_poly_inverse*[*OF g_iso g_ness g_nconst*,*folded vg_def*] **unfolding** *vg_def*
apply *eventually_elim*
by (*auto simp add*:*field_simps*)
qed

lemma *zorder_exist_zero*:
fixes *f*::*complex* ⇒ *complex* **and** *z*::*complex*
defines *n*≡*zorder f z* **and** *g*≡*zor_poly f z*
assumes *holo*: *f holomorphic_on s* **and**
open s connected s z∈*s*
and *non_const*: ∃ *w*∈*s*. *f w* ≠ *0*
shows (*if f z*=*0 then n* > *0 else n*=*0*) ∧ (∃ *r*. *r*>*0* ∧ *cball z r* ⊆ *s* ∧ *g holomorphic_on cball z r*
∧ (∀ *w*∈*cball z r*. *f w* = *g w* ∗ (*w*−*z*) ^ *nat n* ∧ *g w* ≠*0*))

proof −
obtain *r* **where** *g z* ≠ *0* **and** *r*: *r*>*0 cball z r* ⊆ *s g holomorphic_on cball z r*
(∀ *w*∈*cball z r* − {*z*}. *f w* = *g w* ∗ (*w* − *z*) *powr of_int n* ∧ *g w* ≠ *0*)
proof −
have *g z* ≠ *0* ∧ (∃ *r*>*0*. *g holomorphic_on cball z r*
∧ (∀ *w*∈*cball z r* − {*z*}. *f w* = *g w* ∗ (*w* − *z*) *powr of_int n* ∧ *g w* ≠ *0*))
proof (*rule zorder_exist*[*of f z*,*folded g_def n_def*])
show *isolated_singularity_at f z* **unfolding** *isolated_singularity_at_def*
using *holo assms*(*4*,*6*)
by (*meson Diff_subset open_ball analytic_on_holomorphic holomorphic_on_subset openE*)
show *not_essential f z* **unfolding** *not_essential_def*
using *assms*(*4*,*6*) *at_within_open continuous_on holo holomorphic_on_imp_continuous_on*
by *fastforce*
have ∀ *F w in at z*. *f w* ≠ *0* ∧ *w*∈*s*
proof −
obtain *w* **where** *w*∈*s f w*≠*0* **using** *non_const* **by** *auto*
then show *?thesis*
by (*rule non_zero_neighbour_alt*[*OF holo* ‹*open s*› ‹*connected s*› ‹*z*∈*s*›])
qed
then show ∃ *F w in at z*. *f w* ≠ *0*
apply (*elim eventually_frequentlyE*)
by *auto*
qed
then obtain *r1* **where** *g z* ≠ *0 r1*>*0* **and** *r1*:*g holomorphic_on cball z r1*
(∀ *w*∈*cball z r1* − {*z*}. *f w* = *g w* ∗ (*w* − *z*) *powr of_int n* ∧ *g w* ≠ *0*)
by *auto*
obtain *r2* **where** *r2*: *r2*>*0 cball z r2* ⊆ *s*
using *assms*(*4*,*6*) *open_contains_cball_eq* **by** *blast*
define *r3* **where** *r3*=*min r1 r2*
have *r3*>*0 cball z r3* ⊆ *s* **using** ‹*r1*>*0*› *r2* **unfolding** *r3_def* **by** *auto*
moreover have *g holomorphic_on cball z r3*
using *r1*(*1*) **unfolding** *r3_def* **by** *auto*
moreover have (∀ *w*∈*cball z r3* − {*z*}. *f w* = *g w* ∗ (*w* − *z*) *powr of_int n* ∧
g w ≠ *0*)
using *r1*(*2*) **unfolding** *r3_def* **by** *auto*
ultimately show *?thesis* **using** *that*[*of r3*] ‹*g z*≠*0*› **by** *auto*
qed

have *if_0*:*if f z*=*0 then n* > *0 else n*=*0*
proof −
have *f*− *z* → *f z*
by (*metis assms*(*4*,*6*,*7*) *at_within_open continuous_on holo holomorphic_on_imp_continuous_on*)
then have (λ*w*. *g w* ∗ (*w* − *z*) *powr of_int n*) −*z*→ *f z*
apply (*elim Lim_transform_within_open*[**where** *s*=*ball z r*])
using *r* **by** *auto*
moreover have *g* −*z*→*g z*
by (*metis* (*mono_tags*, *lifting*) *open_ball at_within_open_subset*
ball_subset_cball centre_in_ball continuous_on holomorphic_on_imp_continuous_on

```
r(1,3) subsetCE)
  ultimately have (λw. (g w * (w − z) powr of_int n) / g w) −z→ f z/g z
    apply (rule_tac tendsto_divide)
    using ‹g z≠0› by auto
  then have powr_tendsto:(λw. (w − z) powr of_int n) −z→ f z/g z
    apply (elim Lim_transform_within_open[where s=ball z r])
    using r by auto

  have ?thesis when n≥0 f z=0
  proof −
    have (λw. (w − z) ^ nat n) −z→ f z/g z
      using powr_tendsto
      apply (elim Lim_transform_within[where d=r])
      by (auto simp add: powr_of_int ‹n≥0› ‹r>0›)
    then have ∗:(λw. (w − z) ^ nat n) −z→ 0 using ‹f z=0› by simp
    moreover have False when n=0
    proof −
      have (λw. (w − z) ^ nat n) −z→ 1
        using ‹n=0› by auto
      then show False using ∗ using LIM_unique zero_neq_one by blast
    qed
    ultimately show ?thesis using that by fastforce
  qed
  moreover have ?thesis when n≥0 f z≠0
  proof −
    have False when n>0
    proof −
      have (λw. (w − z) ^ nat n) −z→ f z/g z
        using powr_tendsto
        apply (elim Lim_transform_within[where d=r])
        by (auto simp add: powr_of_int ‹n≥0› ‹r>0›)
      moreover have (λw. (w − z) ^ nat n) −z→ 0
        using ‹n>0› by (auto intro!:tendsto_eq_intros)
        ultimately show False using ‹f z≠0› ‹g z≠0› using LIM_unique di-
vide_eq_0_iff by blast
    qed
    then show ?thesis using that by force
  qed
  moreover have False when n<0
  proof −
    have (λw. inverse ((w − z) ^ nat (− n))) −z→ f z/g z
        (λw.((w − z) ^ nat (− n))) −z→ 0
      subgoal  using powr_tendsto powr_of_int that
        by (elim Lim_transform_within_open[where s=UNIV],auto)
      subgoal using that by (auto intro!:tendsto_eq_intros)
      done
    from tendsto_mult[OF this,simplified]
    have (λx. inverse ((x − z) ^ nat (− n)) * (x − z) ^ nat (− n)) −z→ 0 .
    then have (λx. 1::complex) −z→ 0
```

```
      by (elim Lim_transform_within_open[where s=UNIV],auto)
    then show False using LIM_const_eq by fastforce
   qed
   ultimately show ?thesis by fastforce
  qed
  moreover have f w = g w * (w−z) ^ nat n ∧ g w ≠0 when w∈cball z r for w
  proof (cases w=z)
   case True
   then have f −z→f w
    using assms(4,6) at_within_open continuous_on holo holomorphic_on_imp_continuous_on
by fastforce
   then have (λw. g w * (w−z) ^ nat n) −z→f w
   proof (elim Lim_transform_within[OF _ ‹r>0›])
    fix x assume 0 < dist x z dist x z < r
    then have x ∈ cball z r − {z} x≠z
     unfolding cball_def by (auto simp add: dist_commute)
    then have f x = g x * (x − z) powr of_int n
     using r(4)[rule_format,of x] by simp
    also have ... = g x * (x − z) ^ nat n
     apply (subst powr_of_int)
     using if_0 ‹x≠z› by (auto split:if_splits)
    finally show f x = g x * (x − z) ^ nat n .
   qed
   moreover have (λw. g w * (w−z) ^ nat n) −z→ g w * (w−z) ^ nat n
    using True apply (auto intro!:tendsto_eq_intros)
    by (metis open_ball at_within_open_subset ball_subset_cball centre_in_ball
        continuous_on holomorphic_on_imp_continuous_on r(1) r(3) that)
   ultimately have f w = g w * (w−z) ^ nat n using LIM_unique by blast
   then show ?thesis using ‹g z≠0› True by auto
  next
   case False
   then have f w = g w * (w − z) powr of_int n ∧ g w ≠ 0
    using r(4) that by auto
   then show ?thesis using False if_0 powr_of_int by (auto split:if_splits)
  qed
  ultimately show ?thesis using r by auto
qed

lemma zorder_exist_pole:
  fixes f::complex ⇒ complex and z::complex
  defines n≡zorder f z and g≡zor_poly f z
  assumes holo: f holomorphic_on s−{z} and
          open s z∈s
      and is_pole f z
  shows n < 0 ∧ g z≠0 ∧ (∃r. r>0 ∧ cball z r ⊆ s ∧ g holomorphic_on cball z r
    ∧ (∀w∈cball z r − {z}. f w = g w / (w−z) ^ nat (− n) ∧ g w ≠0))
proof −
  obtain r where g z ≠ 0 and r: r>0 cball z r ⊆ s g holomorphic_on cball z r
          (∀w∈cball z r − {z}. f w = g w * (w − z) powr of_int n ∧ g w ≠ 0)
```

```
  proof −
    have g z ≠ 0 ∧ (∃ r>0. g holomorphic_on cball z r
          ∧ (∀ w∈cball z r − {z}. f w = g w ∗ (w − z) powr of_int n ∧ g w ≠ 0))
    proof (rule zorder_exist[of f z,folded g_def n_def])
      show isolated_singularity_at f z unfolding isolated_singularity_at_def
        using holo assms(4,5)
          by (metis analytic_on_holomorphic centre_in_ball insert_Diff openE
open_delete subset_insert_iff)
      show not_essential f z unfolding not_essential_def
      using assms(4,6) at_within_open continuous_on holo holomorphic_on_imp_continuous_on
        by fastforce
      from non_zero_neighbour_pole[OF ‹is_pole f z›] show ∃ F w in at z. f w ≠
0
        apply (elim eventually_frequentlyE)
        by auto
    qed
    then obtain r1 where g z ≠ 0 r1>0 and r1:g holomorphic_on cball z r1
          (∀ w∈cball z r1 − {z}. f w = g w ∗ (w − z) powr of_int n ∧ g w ≠ 0)
      by auto
    obtain r2 where r2: r2>0 cball z r2 ⊆ s
      using assms(4,5) open_contains_cball_eq by metis
    define r3 where r3=min r1 r2
    have r3>0 cball z r3 ⊆ s using ‹r1>0› r2 unfolding r3_def by auto
    moreover have g holomorphic_on cball z r3
      using r1(1) unfolding r3_def by auto
    moreover have (∀ w∈cball z r3 − {z}. f w = g w ∗ (w − z) powr of_int n ∧
g w ≠ 0)
      using r1(2) unfolding r3_def by auto
    ultimately show ?thesis using that[of r3] ‹g z≠0› by auto
  qed

  have n<0
  proof (rule ccontr)
    assume ¬ n < 0
    define c where c=(if n=0 then g z else 0)
    have [simp]:g −z→ g z
      by (metis open_ball at_within_open ball_subset_cball centre_in_ball
        continuous_on holomorphic_on_imp_continuous_on holomorphic_on_subset
r(1) r(3) )
    have ∀ F x in at z. f x = g x ∗ (x − z) ^ nat n
      unfolding eventually_at_topological
      apply (rule_tac exI[where x=ball z r])
      using r powr_of_int ‹¬ n < 0› by auto
    moreover have (λx. g x ∗ (x − z) ^ nat n) −z→c
    proof (cases n=0)
      case True
      then show ?thesis unfolding c_def by simp
    next
      case False
```

```
    then have (λx. (x − z) ^ nat n) −z→ 0 using ‹¬ n < 0›
      by (auto intro!:tendsto_eq_intros)
    from tendsto_mult[OF _ this,of g g z,simplified]
    show ?thesis unfolding c_def using False by simp
  qed
  ultimately have f −z→c using tendsto_cong by fast
 then show False using ‹is_pole f z› at_neq_bot not_tendsto_and_filterlim_at_infinity
    unfolding is_pole_def by blast
 qed
 moreover have ∀ w∈cball z r − {z}. f w = g w / (w−z) ^ nat (− n) ∧ g w ≠0
  using r(4) ‹n<0› powr_of_int
  by (metis Diff_iff divide_inverse eq_iff_diff_eq_0 insert_iff linorder_not_le)
 ultimately show ?thesis using r(1−3) ‹g z≠0› by auto
qed

lemma zorder_eqI:
 assumes open s z ∈ s g holomorphic_on s g z ≠ 0
 assumes fg_eq:⋀w. ⟦w ∈ s;w≠z⟧ ⟹ f w = g w * (w − z) powr n
 shows   zorder f z = n
proof −
 have continuous_on s g by (rule holomorphic_on_imp_continuous_on) fact
 moreover have open (−{0::complex}) by auto
 ultimately have open ((g −‘ (−{0})) ∩ s)
  unfolding continuous_on_open_vimage[OF ‹open s›] by blast
 moreover from assms have z ∈ (g −‘ (−{0})) ∩ s by auto
 ultimately obtain r where r: r > 0 cball z r ⊆  s ∩ (g −‘ (−{0}))
  unfolding open_contains_cball by blast

 let ?gg= (λw. g w * (w − z) powr n)
 define P where P = (λn g r. 0 < r ∧ g holomorphic_on cball z r ∧ g z≠0
       ∧ (∀ w∈cball z r − {z}. f w = g w * (w−z) powr (of_int n) ∧ g w≠0))
 have P n g r
  unfolding P_def using r assms(3,4,5) by auto
 then have ∃ g r. P n g r by auto
 moreover have unique: ∃!n. ∃ g r. P n g r unfolding P_def
 proof (rule holomorphic_factor_puncture)
  have ball z r−{z} ⊆ s using r using ball_subset_cball by blast
  then have ?gg holomorphic_on ball z r−{z}
   using ‹g holomorphic_on s› r by (auto intro!: holomorphic_intros)
  then have f holomorphic_on ball z r − {z}
   apply (elim holomorphic_transform)
   using fg_eq ‹ball z r−{z} ⊆ s› by auto
  then show isolated_singularity_at f z unfolding isolated_singularity_at_def
   using analytic_on_open open_delete r(1) by blast
 next
  have not_essential ?gg z
  proof (intro singularity_intros)
   show not_essential g z
   by (meson ‹continuous_on s g› assms(1) assms(2) continuous_on_eq_continuous_at
```

```
        isCont_def not_essential_def)
    show ∀F w in at z. w − z ≠ 0 by (simp add: eventually_at_filter)
    then show LIM w at z. w − z :> at 0
      unfolding filterlim_at by (auto intro:tendsto_eq_intros)
    show isolated_singularity_at g z
      by (meson Diff_subset open_ball analytic_on_holomorphic
        assms(1,2,3) holomorphic_on_subset isolated_singularity_at_def openE)
  qed
  then show not_essential f z
    apply (elim not_essential_transform)
    unfolding eventually_at using assms(1,2) assms(5)[symmetric]
    by (metis dist_commute mem_ball openE subsetCE)
  show ∃F w in at z. f w ≠ 0 unfolding frequently_at
  proof (rule,rule)
    fix d::real assume 0 < d
    define z' where z'=z+min d r / 2
    have z' ≠ z  dist z' z < d
      unfolding z'_def using ‹d>0› ‹r>0›
      by (auto simp add:dist_norm)
    moreover have f z' ≠ 0
    proof (subst fg_eq[OF _ ‹z'≠z›])
      have z' ∈ cball z r unfolding z'_def using ‹r>0› ‹d>0› by (auto simp
add:dist_norm)
      then show  z' ∈ s using r(2) by blast
      show g z' * (z' − z) powr of_int n ≠ 0
        using P_def ‹P n g r› ‹z' ∈ cball z r› calculation(1) by auto
    qed
    ultimately show ∃x∈UNIV. x ≠ z ∧ dist x z < d ∧ f x ≠ 0 by auto
  qed
 qed
 ultimately have (THE n. ∃ g r. P n g r) = n
  by (rule_tac the1_equality)
 then show ?thesis unfolding zorder_def P_def by blast
qed

lemma simple_zeroI:
 assumes open s z ∈ s g holomorphic_on s g z ≠ 0
 assumes ⋀w. w ∈ s ⟹ f w = g w * (w − z)
 shows   zorder f z = 1
 using assms(1−4) by (rule zorder_eqI) (use assms(5) in auto)

lemma higher_deriv_power:
 shows   (deriv ^^ j) (λw. (w − z) ^ n) w =
          pochhammer (of_nat (Suc n − j)) j * (w − z) ^ (n − j)
proof (induction j arbitrary: w)
 case 0
 thus ?case by auto
next
 case (Suc j w)
```

```
  have (deriv ^^ Suc j) (λw. (w − z) ^ n) w = deriv ((deriv ^^ j) (λw. (w − z)
^ n)) w
    by simp
  also have (deriv ^^ j) (λw. (w − z) ^ n) =
              (λw. pochhammer (of_nat (Suc n − j)) j ∗ (w − z) ^ (n − j))
    using Suc by (intro Suc.IH ext)
  also {
    have (... has_field_derivative of_nat (n − j) ∗
              pochhammer (of_nat (Suc n − j)) j ∗ (w − z) ^ (n − Suc j)) (at w)
      using Suc.prems by (auto intro!: derivative_eq_intros)
    also have of_nat (n − j) ∗ pochhammer (of_nat (Suc n − j)) j =
                pochhammer (of_nat (Suc n − Suc j)) (Suc j)
      by (cases Suc j ≤ n, subst pochhammer_rec)
         (insert Suc.prems, simp_all add: algebra_simps Suc_diff_le pochham-
mer_0_left)
    finally have deriv (λw. pochhammer (of_nat (Suc n − j)) j ∗ (w − z) ^ (n −
j)) w =
                    ... ∗ (w − z) ^ (n − Suc j)
      by (rule DERIV_imp_deriv)
  }
  finally show ?case .
qed

lemma zorder_zero_eqI:
  assumes f_holo:f holomorphic_on s and open s z ∈ s
  assumes zero: ⋀i. i < nat n ⟹ (deriv ^^ i) f z = 0
  assumes nz: (deriv ^^ nat n) f z ≠ 0 and n≥0
  shows   zorder f z = n
proof −
  obtain r where [simp]:r>0 and ball z r ⊆ s
    using ‹open s› ‹z∈s› openE by blast
  have nz':∃ w∈ball z r. f w ≠ 0
  proof (rule ccontr)
    assume ¬ (∃ w∈ball z r. f w ≠ 0)
    then have eventually (λu. f u = 0) (nhds z)
      using ‹r>0› unfolding eventually_nhds
      apply (rule_tac x=ball z r in exI)
      by auto
    then have (deriv ^^ nat n) f z = (deriv ^^ nat n) (λ_. 0) z
      by (intro higher_deriv_cong_ev) auto
    also have (deriv ^^ nat n) (λ_. 0) z = 0
      by (induction n) simp_all
    finally show False using nz by contradiction
  qed

  define zn g where zn = zorder f z and g = zor_poly f z
  obtain e where e_if:if f z = 0 then 0 < zn else zn = 0 and
          [simp]:e>0 and cball z e ⊆ ball z r and
          g_holo:g holomorphic_on cball z e and
```

```
        e_fac:(∀ w∈cball z e. f w = g w * (w - z) ^ nat zn ∧ g w ≠ 0)
  proof -
    have f holomorphic_on ball z r
      using f_holo ‹ball z r ⊆ s› by auto
    from that zorder_exist_zero[of f ball z r z,simplified,OF this nz',folded zn_def
g_def]
    show ?thesis by blast
  qed
  from this(1,2,5) have zn≥0 g z≠0
    subgoal by (auto split:if_splits)
    subgoal using ‹0 < e› ball_subset_cball centre_in_ball e_fac by blast
    done

  define A where A = (λi. of_nat (i choose (nat zn)) * fact (nat zn) * (deriv ^^
(i - nat zn)) g z)
  have deriv_A:(deriv ^^ i) f z = (if zn ≤ int i then A i else 0) for i
  proof -
    have eventually (λw. w ∈ ball z e) (nhds z)
      using ‹cball z e ⊆ ball z r› ‹e>0› by (intro eventually_nhds_in_open) auto
    hence eventually (λw. f w = (w - z) ^ (nat zn) * g w) (nhds z)
      apply eventually_elim
      by (use e_fac in auto)
    hence (deriv ^^ i) f z = (deriv ^^ i) (λw. (w - z) ^ nat zn * g w) z
      by (intro higher_deriv_cong_ev) auto
    also have ... = (∑j=0..i. of_nat (i choose j) *
                    (deriv ^^ j) (λw. (w - z) ^ nat zn) z * (deriv ^^ (i - j)) g z)
      using g_holo ‹e>0›
      by (intro higher_deriv_mult[of _ ball z e]) (auto intro!: holomorphic_intros)
    also have ... = (∑j=0..i. if j = nat zn then
                  of_nat (i choose nat zn) * fact (nat zn) * (deriv ^^ (i - nat zn))
g z else 0)
    proof (intro sum.cong refl, goal_cases)
      case (1 j)
      have (deriv ^^ j) (λw. (w - z) ^ nat zn) z =
            pochhammer (of_nat (Suc (nat zn) - j)) j * 0 ^ (nat zn - j)
        by (subst higher_deriv_power) auto
      also have ... = (if j = nat zn then fact j else 0)
        by (auto simp: not_less pochhammer_0_left pochhammer_fact)
      also have of_nat (i choose j) * ... * (deriv ^^ (i - j)) g z =
                  (if j = nat zn then of_nat (i choose (nat zn)) * fact (nat zn)
                      * (deriv ^^ (i - nat zn)) g z else 0)
        by simp
      finally show ?case .
    qed
    also have ... = (if i ≥ zn then A i else 0)
      by (auto simp: A_def)
    finally show (deriv ^^ i) f z = ... .
  qed
```

```
  have False when n<zn
  proof −
    have (deriv ^^ nat n) f z = 0
      using deriv_A[of nat n] that ‹n≥0› by auto
    with nz show False by auto
  qed
  moreover have n≤zn
  proof −
    have g z ≠ 0 using e_fac[rule_format,of z] ‹e>0› by simp
    then have (deriv ^^ nat zn) f z ≠ 0
      using deriv_A[of nat zn] by(auto simp add:A_def)
    then have nat zn ≥ nat n using zero[of nat zn] by linarith
    moreover have zn≥0 using e_if by (auto split:if_splits)
    ultimately show ?thesis using nat_le_eq_zle by blast
  qed
  ultimately show ?thesis unfolding zn_def by fastforce
qed

lemma
  assumes eventually (λz. f z = g z) (at z) z = z'
  shows zorder_cong:zorder f z = zorder g z' and zor_poly_cong:zor_poly f z =
zor_poly g z'
proof −
  define P where P = (λff n h r. 0 < r ∧ h holomorphic_on cball z r ∧ h z≠0
                ∧ (∀ w∈cball z r − {z}. ff w = h w * (w−z) powr (of_int n) ∧ h
w≠0))
  have (∃ r. P f n h r) = (∃ r. P g n h r) for n h
  proof −
    have ∗: ∃ r. P g n h r if ∃ r. P f n h r and eventually (λx. f x = g x) (at z)
for f g
    proof −
      from that(1) obtain r1 where r1_P:P f n h r1 by auto
      from that(2) obtain r2 where r2>0 and r2_dist:∀ x. x ≠ z ∧ dist x z ≤
r2 ⟶ f x = g x
        unfolding eventually_at_le by auto
      define r where r=min r1 r2
      have r>0 h z≠0 using r1_P ‹r2>0› unfolding r_def P_def by auto
      moreover have h holomorphic_on cball z r
        using r1_P unfolding P_def r_def by auto
     moreover have g w = h w * (w − z) powr of_int n ∧ h w ≠ 0 when w∈cball
z r − {z} for w
      proof −
        have f w = h w * (w − z) powr of_int n ∧ h w ≠ 0
          using r1_P that unfolding P_def r_def by auto
        moreover have f w=g w using r2_dist[rule_format,of w] that unfolding
r_def
          by (simp add: dist_commute)
        ultimately show ?thesis by simp
      qed
```

```
    ultimately show ?thesis unfolding P_def by auto
  qed
  from assms have eq': eventually (λz. g z = f z) (at z)
    by (simp add: eq_commute)
  show ?thesis
    by (rule iffI[OF *[OF _ assms(1)] *[OF _ eq']])
 qed
 then show zorder f z = zorder g z' zor_poly f z = zor_poly g z'
    using ‹z=z'› unfolding P_def zorder_def zor_poly_def by auto
qed

lemma zorder_nonzero_div_power:
 assumes open s z ∈ s f holomorphic_on s f z ≠ 0 n > 0
 shows  zorder (λw. f w / (w − z) ^ n) z = − n
 apply (rule zorder_eqI[OF ‹open s› ‹z∈s› ‹f holomorphic_on s› ‹f z≠0›])
 apply (subst powr_of_int)
 using ‹n>0› by (auto simp add:field_simps)

lemma zor_poly_eq:
 assumes isolated_singularity_at f z not_essential f z ∃ F w in at z. f w ≠ 0
 shows eventually (λw. zor_poly f z w = f w * (w − z) powr − zorder f z) (at z)
proof −
 obtain r where r:r>0
      (∀ w∈cball z r − {z}. f w = zor_poly f z w * (w − z) powr of_int (zorder f
z))
    using zorder_exist[OF assms] by blast
 then have *: ∀ w∈ball z r − {z}. zor_poly f z w = f w * (w − z) powr − zorder
f z
    by (auto simp: field_simps powr_minus)
 have eventually (λw. w ∈ ball z r − {z}) (at z)
    using r eventually_at_ball'[of r z UNIV] by auto
 thus ?thesis by eventually_elim (insert *, auto)
qed

lemma zor_poly_zero_eq:
 assumes f holomorphic_on s open s connected s z ∈ s ∃ w∈s. f w ≠ 0
 shows eventually (λw. zor_poly f z w = f w / (w − z) ^ nat (zorder f z)) (at z)
proof −
 obtain r where r:r>0
      (∀ w∈cball z r − {z}. f w = zor_poly f z w * (w − z) ^ nat (zorder f z))
    using zorder_exist_zero[OF assms] by auto
 then have *: ∀ w∈ball z r − {z}. zor_poly f z w = f w / (w − z) ^ nat (zorder
f z)
    by (auto simp: field_simps powr_minus)
 have eventually (λw. w ∈ ball z r − {z}) (at z)
    using r eventually_at_ball'[of r z UNIV] by auto
 thus ?thesis by eventually_elim (insert *, auto)
qed
```

lemma *zor_poly_pole_eq*:
assumes *f_iso*:*isolated_singularity_at f z is_pole f z*
shows *eventually* (λ*w*. *zor_poly f z w* = *f w* ∗ (*w* − *z*) ^ *nat* (− *zorder f z*)) (*at z*)
proof −
obtain *e* **where** [*simp*]:*e*>*0* **and** *f_holo*:*f holomorphic_on ball z e* − {*z*}
using *f_iso analytic_imp_holomorphic* **unfolding** *isolated_singularity_at_def*
by *blast*
obtain *r* **where** *r*:*r*>*0*
(∀ *w*∈*cball z r* − {*z*}. *f w* = *zor_poly f z w* / (*w* − *z*) ^ *nat* (− *zorder f z*))
using *zorder_exist_pole*[*OF f_holo*,*simplified*,*OF* ‹*is_pole f z*›] **by** *auto*
then have ∗: ∀ *w*∈*ball z r* − {*z*}. *zor_poly f z w* = *f w* ∗ (*w* − *z*) ^ *nat* (− *zorder f z*)
by (*auto simp*: *field_simps*)
have *eventually* (λ*w*. *w* ∈ *ball z r* − {*z*}) (*at z*)
using *r eventually_at_ball'*[*of r z UNIV*] **by** *auto*
thus *?thesis* **by** *eventually_elim* (*insert* ∗, *auto*)
qed

lemma *zor_poly_eqI*:
fixes *f* :: *complex* ⇒ *complex* **and** *z0* :: *complex*
defines *n* ≡ *zorder f z0*
assumes *isolated_singularity_at f z0 not_essential f z0* ∃ *F w in at z0*. *f w* ≠ *0*
assumes *lim*: ((λ*x*. *f* (*g x*) ∗ (*g x* − *z0*) *powr* − *n*) ⟶ *c*) *F*
assumes *g*: *filterlim g* (*at z0*) *F* **and** *F* ≠ *bot*
shows *zor_poly f z0 z0* = *c*
proof −
from *zorder_exist*[*OF assms*(*2*−*4*)] **obtain** *r* **where**
r: *r* > *0 zor_poly f z0 holomorphic_on cball z0 r*
⋀*w*. *w* ∈ *cball z0 r* − {*z0*} ⟹ *f w* = *zor_poly f z0 w* ∗ (*w* − *z0*) *powr n*
unfolding *n_def* **by** *blast*
from *r*(*1*) **have** *eventually* (λ*w*. *w* ∈ *ball z0 r* ∧ *w* ≠ *z0*) (*at z0*)
using *eventually_at_ball'*[*of r z0 UNIV*] **by** *auto*
hence *eventually* (λ*w*. *zor_poly f z0 w* = *f w* ∗ (*w* − *z0*) *powr* − *n*) (*at z0*)
by *eventually_elim* (*insert r*, *auto simp*: *field_simps powr_minus*)
moreover have *continuous_on* (*ball z0 r*) (*zor_poly f z0*)
using *r* **by** (*intro holomorphic_on_imp_continuous_on*) *auto*
with *r*(*1*,*2*) **have** *isCont* (*zor_poly f z0*) *z0*
by (*auto simp*: *continuous_on_eq_continuous_at*)
hence (*zor_poly f z0* ⟶ *zor_poly f z0 z0*) (*at z0*)
unfolding *isCont_def* **.**
ultimately have ((λ*w*. *f w* ∗ (*w* − *z0*) *powr* − *n*) ⟶ *zor_poly f z0 z0*) (*at z0*)
by (*blast intro*: *Lim_transform_eventually*)
hence ((λ*x*. *f* (*g x*) ∗ (*g x* − *z0*) *powr* − *n*) ⟶ *zor_poly f z0 z0*) *F*
by (*rule filterlim_compose*[*OF* _ *g*])
from *tendsto_unique*[*OF* ‹*F* ≠ *bot*› *this lim*] **show** *?thesis* **.**
qed

lemma *zor_poly_zero_eqI*:
fixes $f :: complex \Rightarrow complex$ **and** $z0 :: complex$
defines $n \equiv zorder\ f\ z0$
assumes $f\ holomorphic\_on\ A\ open\ A\ connected\ A\ z0 \in A\ \exists z{\in}A.\ f\ z \neq 0$
assumes $lim\colon ((\lambda x.\ f\ (g\ x)\ /\ (g\ x - z0)\ \hat{}\ nat\ n) \longrightarrow c)\ F$
assumes $g\colon filterlim\ g\ (at\ z0)\ F$ **and** $F \neq bot$
shows $zor\_poly\ f\ z0\ z0 = c$
proof −
from $zorder\_exist\_zero[OF\ assms(2{-}6)]$ **obtain** r **where**
$r\colon r > 0\ cball\ z0\ r \subseteq A\ zor\_poly\ f\ z0\ holomorphic\_on\ cball\ z0\ r$
$\bigwedge w.\ w \in cball\ z0\ r \Longrightarrow f\ w = zor\_poly\ f\ z0\ w * (w - z0)\ \hat{}\ nat\ n$
unfolding n_def **by** $blast$
from $r(1)$ **have** $eventually\ (\lambda w.\ w \in ball\ z0\ r \wedge w \neq z0)\ (at\ z0)$
using $eventually\_at\_ball'[of\ r\ z0\ UNIV]$ **by** $auto$
hence $eventually\ (\lambda w.\ zor\_poly\ f\ z0\ w = f\ w\ /\ (w - z0)\ \hat{}\ nat\ n)\ (at\ z0)$
by $eventually\_elim\ (insert\ r,\ auto\ simp\colon field\_simps)$
moreover have $continuous\_on\ (ball\ z0\ r)\ (zor\_poly\ f\ z0)$
using r **by** $(intro\ holomorphic\_on\_imp\_continuous\_on)\ auto$
with $r(1,2)$ **have** $isCont\ (zor\_poly\ f\ z0)\ z0$
by $(auto\ simp\colon continuous\_on\_eq\_continuous\_at)$
hence $(zor\_poly\ f\ z0 \longrightarrow zor\_poly\ f\ z0\ z0)\ (at\ z0)$
unfolding $isCont\_def$ **.**
ultimately have $((\lambda w.\ f\ w\ /\ (w - z0)\ \hat{}\ nat\ n) \longrightarrow zor\_poly\ f\ z0\ z0)\ (at\ z0)$
by $(blast\ intro\colon Lim\_transform\_eventually)$
hence $((\lambda x.\ f\ (g\ x)\ /\ (g\ x - z0)\ \hat{}\ nat\ n) \longrightarrow zor\_poly\ f\ z0\ z0)\ F$
by $(rule\ filterlim\_compose[OF\ \_\ g])$
from $tendsto\_unique[OF\ \langle F \neq bot\rangle\ this\ lim]$ **show** $?thesis$ **.**
qed

lemma *zor_poly_pole_eqI*:
fixes $f :: complex \Rightarrow complex$ **and** $z0 :: complex$
defines $n \equiv zorder\ f\ z0$
assumes $f\_iso\colon isolated\_singularity\_at\ f\ z0$ **and** $is\_pole\ f\ z0$
assumes $lim\colon ((\lambda x.\ f\ (g\ x) * (g\ x - z0)\ \hat{}\ nat\ (-n)) \longrightarrow c)\ F$
assumes $g\colon filterlim\ g\ (at\ z0)\ F$ **and** $F \neq bot$
shows $zor\_poly\ f\ z0\ z0 = c$
proof −
obtain r **where** $r\colon r > 0\ \ zor\_poly\ f\ z0\ holomorphic\_on\ cball\ z0\ r$
proof −
have $\exists_F\ w\ in\ at\ z0.\ f\ w \neq 0$
using $non\_zero\_neighbour\_pole[OF\ \langle is\_pole\ f\ z0\rangle]$ **by** $(auto\ elim\colon eventually\_frequentlyE)$
moreover have $not\_essential\ f\ z0$ **unfolding** $not\_essential\_def$ **using** $\langle is\_pole\ f\ z0\rangle$ **by** $simp$
ultimately show $?thesis$ **using** $that\ zorder\_exist[OF\ f\_iso,folded\ n\_def]$ **by** $auto$
qed
from $r(1)$ **have** $eventually\ (\lambda w.\ w \in ball\ z0\ r \wedge w \neq z0)\ (at\ z0)$
using $eventually\_at\_ball'[of\ r\ z0\ UNIV]$ **by** $auto$
have $eventually\ (\lambda w.\ zor\_poly\ f\ z0\ w = f\ w * (w - z0)\ \hat{}\ nat\ (-n))\ (at\ z0)$

```
    using zor_poly_pole_eq[OF f_iso ‹is_pole f z0›] unfolding n_def .
  moreover have continuous_on (ball z0 r) (zor_poly f z0)
    using r by (intro holomorphic_on_imp_continuous_on) auto
  with r(1,2) have isCont (zor_poly f z0) z0
    by (auto simp: continuous_on_eq_continuous_at)
  hence (zor_poly f z0 ⟶ zor_poly f z0 z0) (at z0)
    unfolding isCont_def .
  ultimately have ((λw. f w * (w - z0) ^ nat (-n)) ⟶ zor_poly f z0 z0) (at
z0)
    by (blast intro: Lim_transform_eventually)
  hence ((λx. f (g x) * (g x - z0) ^ nat (-n)) ⟶ zor_poly f z0 z0) F
    by (rule filterlim_compose[OF _ g])
  from tendsto_unique[OF ‹F ≠ bot› this lim] show ?thesis .
qed

end
theory Complex_Residues
  imports Complex_Singularities
begin
```

5.11 Definition of residues

Wenda Li and LC Paulson (2016). A Formal Proof of Cauchy's Residue Theorem. Interactive Theorem Proving

```
definition residue :: (complex ⇒ complex) ⇒ complex ⇒ complex where
  residue f z = (SOME int. ∃e>0. ∀ε>0. ε<e
    ⟶ (f has_contour_integral 2*pi* i *int) (circlepath z ε))

lemma residue_cong:
  assumes eq: eventually (λz. f z = g z) (at z) and z = z'
  shows   residue f z = residue g z'
proof -
  from assms have eq': eventually (λz. g z = f z) (at z)
    by (simp add: eq_commute)
  let ?P = λf c e. (∀ε>0. ε < e ⟶
   (f has_contour_integral of_real (2 * pi) * i * c) (circlepath z ε))
  have residue f z = residue g z unfolding residue_def
  proof (rule Eps_cong)
    fix c :: complex
    have ∃e>0. ?P g c e
      if ∃e>0. ?P f c e and eventually (λz. f z = g z) (at z) for f g
    proof -
      from that(1) obtain e where e: e > 0 ?P f c e
        by blast
      from that(2) obtain e' where e': e' > 0 ⋀z'. z' ≠ z ⟹ dist z' z < e' ⟹
f z' = g z'
        unfolding eventually_at by blast
      have ?P g c (min e e')
      proof (intro allI exI impI, goal_cases)
```

```
      case (1 ε)
      hence (f has_contour_integral of_real (2 * pi) * i * c) (circlepath z ε)
        using e(2) by auto
      thus ?case
      proof (rule has_contour_integral_eq)
        fix z' assume z' ∈ path_image (circlepath z ε)
        hence dist z' z < e' and z' ≠ z
          using 1 by (auto simp: dist_commute)
        with e'(2)[of z'] show f z' = g z' by simp
      qed
    qed
    moreover from e and e' have min e e' > 0 by auto
    ultimately show ?thesis by blast
  qed
  from this[OF _ eq] and this[OF _ eq']
    show (∃e>0. ?P f c e) ⟷ (∃e>0. ?P g c e)
    by blast
 qed
 with assms show ?thesis by simp
qed

lemma contour_integral_circlepath_eq:
 assumes open s and f_holo:f holomorphic_on (s-{z}) and 0<e1 e1≤e2
   and e2_cball:cball z e2 ⊆ s
 shows
   f contour_integrable_on circlepath z e1
   f contour_integrable_on circlepath z e2
   contour_integral (circlepath z e2) f = contour_integral (circlepath z e1) f
proof -
 define l where l ≡ linepath (z+e2) (z+e1)
 have [simp]:valid_path l pathstart l=z+e2 pathfinish l=z+e1 unfolding l_def
by auto
 have e2>0 using ‹e1>0› ‹e1≤e2› by auto
 have zl_img:z∉path_image l
   proof
     assume z ∈ path_image l
     then have e2 ≤ cmod (e2 - e1)
      using segment_furthest_le[of z z+e2 z+e1 z+e2,simplified] ‹e1>0› ‹e2>0›
unfolding l_def
       by (auto simp add:closed_segment_commute)
     thus False using ‹e2>0› ‹e1>0› ‹e1≤e2›
       apply (subst (asm) norm_of_real)
       by auto
   qed
 define g where g ≡ circlepath z e2 +++ l +++ reversepath (circlepath z e1)
+++ reversepath l
  show [simp]: f contour_integrable_on circlepath z e2 f contour_integrable_on
(circlepath z e1)
   proof -
```

```
      show f contour_integrable_on circlepath z e2
        apply (intro contour_integrable_continuous_circlepath[OF
              continuous_on_subset[OF holomorphic_on_imp_continuous_on[OF
f_holo]]])
        using ‹e2>0› e2_cball by auto
      show f contour_integrable_on (circlepath z e1)
        apply (intro contour_integrable_continuous_circlepath[OF
              continuous_on_subset[OF holomorphic_on_imp_continuous_on[OF
f_holo]]])
        using ‹e1>0› ‹e1≤e2› e2_cball by auto
    qed
  have [simp]:f contour_integrable_on l
    proof −
      have closed_segment (z + e2) (z + e1) ⊆ cball z e2 using ‹e2>0› ‹e1>0›
‹e1≤e2›
        by (intro closed_segment_subset,auto simp add:dist_norm)
      hence closed_segment (z + e2) (z + e1) ⊆ s − {z} using zl_img e2_cball
unfolding l_def
        by auto
      then show f contour_integrable_on l unfolding l_def
        apply (intro contour_integrable_continuous_linepath[OF
              continuous_on_subset[OF holomorphic_on_imp_continuous_on[OF
f_holo]]])
        by auto
    qed
  let ?ig=λg. contour_integral g f
  have (f has_contour_integral 0) g
    proof (rule Cauchy_theorem_global[OF _ f_holo])
      show open (s − {z}) using ‹open s› by auto
      show valid_path g unfolding g_def l_def by auto
      show pathfinish g = pathstart g unfolding g_def l_def by auto
    next
      have path_img:path_image g ⊆ cball z e2
        proof −
        have closed_segment (z + e2) (z + e1) ⊆ cball z e2 using ‹e2>0› ‹e1>0›
‹e1≤e2›
          by (intro closed_segment_subset,auto simp add:dist_norm)
        moreover have sphere z |e1| ⊆ cball z e2 using ‹e2>0› ‹e1≤e2› ‹e1>0›
by auto
          ultimately show ?thesis unfolding g_def l_def using ‹e2>0›
            by (simp add: path_image_join closed_segment_commute)
        qed
      show path_image g ⊆ s − {z}
        proof −
          have z∉path_image g using zl_img
         unfolding g_def l_def by (auto simp add: path_image_join closed_segment_commute)
          moreover note ‹cball z e2 ⊆ s› and path_img
          ultimately show ?thesis by auto
        qed
```

```
    show winding_number g w = 0 whenw ∉ s − {z} for w
     proof −
       have winding_number g w = 0 when w∉s using that e2_cball
        apply (intro winding_number_zero_outside[OF _ _ _ _ path_img])
        by (auto simp add:g_def l_def)
       moreover have winding_number g z=0
        proof −
          let ?Wz=λg. winding_number g z
            have ?Wz g = ?Wz (circlepath z e2) + ?Wz l + ?Wz (reversepath
(circlepath z e1))
              + ?Wz (reversepath l)
            using ‹e2>0› ‹e1>0› zl_img unfolding g_def l_def
              by (subst winding_number_join,auto simp add:path_image_join
closed_segment_commute)+
           also have ... = ?Wz (circlepath z e2) + ?Wz (reversepath (circlepath
z e1))
            using zl_img
            apply (subst (2) winding_number_reversepath)
            by (auto simp add:l_def closed_segment_commute)
          also have ... = 0
           proof −
            have ?Wz (circlepath z e2) = 1 using ‹e2>0›
             by (auto intro: winding_number_circlepath_centre)
             moreover have ?Wz (reversepath (circlepath z e1)) = −1 using
‹e1>0›
              apply (subst winding_number_reversepath)
              by (auto intro: winding_number_circlepath_centre)
            ultimately show ?thesis by auto
           qed
          finally show ?thesis .
         qed
        ultimately show ?thesis using that by auto
       qed
    qed
  then have 0 = ?ig g using contour_integral_unique by simp
  also have ... = ?ig (circlepath z e2) + ?ig l + ?ig (reversepath (circlepath z e1))
     + ?ig (reversepath l)
    unfolding g_def
    by (auto simp add:contour_integrable_reversepath_eq)
  also have ... = ?ig (circlepath z e2)  − ?ig (circlepath z e1)
    by (auto simp add:contour_integral_reversepath)
  finally show contour_integral (circlepath z e2) f = contour_integral (circlepath
z e1) f
    by simp
qed

lemma base_residue:
  assumes open s z∈s r>0 and f_holo:f holomorphic_on (s − {z})
    and r_cball:cball z r ⊆ s
```

```
  shows (f has_contour_integral 2 * pi * i * (residue f z)) (circlepath z r)
proof −
  obtain e where e>0 and e_cball:cball z e ⊆ s
    using open_contains_cball[of s] ‹open s› ‹z∈s› by auto
  define c where c ≡ 2 * pi * i
  define i where i ≡ contour_integral (circlepath z e) f / c
  have (f has_contour_integral c*i) (circlepath z ε) when ε>0 ε<e for ε
    proof −
      have contour_integral (circlepath z e) f = contour_integral (circlepath z ε) f
          f contour_integrable_on circlepath z ε
          f contour_integrable_on circlepath z e
        using ‹ε<e›
          by (intro contour_integral_circlepath_eq[OF ‹open s› f_holo ‹ε>0› _
e_cball],auto)+
      then show ?thesis unfolding i_def c_def
        by (auto intro:has_contour_integral_integral)
    qed
  then have ∃ e>0. ∀ε>0. ε<e ⟶ (f has_contour_integral c * (residue f z))
(circlepath z ε)
    unfolding residue_def c_def
    apply (rule_tac someI[of _ i],intro  exI[where x=e])
    by (auto simp add:‹e>0› c_def)
  then obtain e' where e'>0
       and e'_def:∀ε>0. ε<e' ⟶ (f has_contour_integral c * (residue f z))
(circlepath z ε)
    by auto
  let ?int=λe. contour_integral (circlepath z e) f
  define  ε where ε ≡ Min {r,e'} / 2
  have ε>0 ε≤r ε<e' using ‹r>0› ‹e'>0› unfolding ε_def by auto
  have (f has_contour_integral c * (residue f z)) (circlepath z ε)
    using e'_def[rule_format,OF ‹ε>0› ‹ε<e'›] .
  then show ?thesis unfolding c_def
   using contour_integral_circlepath_eq[OF ‹open s› f_holo ‹ε>0› ‹ε≤r› r_cball]
    by (auto elim: has_contour_integral_eqpath[of _ _ circlepath z ε circlepath z
r])
qed

lemma residue_holo:
  assumes open s z ∈ s and f_holo: f holomorphic_on s
  shows residue f z = 0
proof −
  define c where c ≡ 2 * pi * i
  obtain e where e>0 and e_cball:cball z e ⊆ s using ‹open s› ‹z∈s›
    using open_contains_cball_eq by blast
  have (f has_contour_integral c*residue f z) (circlepath z e)
    using f_holo
   by (auto intro: base_residue[OF ‹open s› ‹z∈s› ‹e>0› _ e_cball,folded c_def])
  moreover have (f has_contour_integral 0) (circlepath z e)
    using f_holo e_cball ‹e>0›
```

```
    by (auto intro: Cauchy_theorem_convex_simple[of _ cball z e])
  ultimately have c*residue f z =0
    using has_contour_integral_unique by blast
  thus ?thesis unfolding c_def  by auto
qed

lemma residue_const:residue (λ_. c) z = 0
  by (intro residue_holo[of UNIV::complex set],auto intro:holomorphic_intros)

lemma residue_add:
  assumes open s z ∈ s and f_holo: f holomorphic_on s − {z}
      and g_holo:g holomorphic_on s − {z}
  shows residue (λz. f z + g z) z= residue f z + residue g z
proof −
  define c where c ≡ 2 ∗ pi ∗ i
  define fg where fg ≡ (λz. f z+g z)
  obtain e where e>0 and e_cball:cball z e ⊆ s using ‹open s› ‹z∈s›
    using open_contains_cball_eq by blast
  have (fg has_contour_integral c ∗ residue fg z) (circlepath z e)
    unfolding fg_def using f_holo g_holo
    apply (intro base_residue[OF ‹open s› ‹z∈s› ‹e>0› _ e_cball,folded c_def])
    by (auto intro:holomorphic_intros)
 moreover have (fg has_contour_integral c∗residue f z + c∗ residue g z) (circlepath
z e)
    unfolding fg_def using f_holo g_holo
     by (auto intro: has_contour_integral_add base_residue[OF ‹open s› ‹z∈s›
‹e>0› _ e_cball,folded c_def])
  ultimately have c∗(residue f z + residue g z) = c ∗ residue fg z
    using has_contour_integral_unique by (auto simp add:distrib_left)
  thus ?thesis unfolding fg_def
    by (auto simp add:c_def)
qed

lemma residue_lmul:
  assumes open s z ∈ s and f_holo: f holomorphic_on s − {z}
  shows residue (λz. c ∗ (f z)) z= c ∗ residue f z
proof (cases c=0)
  case True
  thus ?thesis using residue_const by auto
next
  case False
  define c' where c' ≡ 2 ∗ pi ∗ i
  define f' where f' ≡ (λz. c ∗ (f z))
  obtain e where e>0 and e_cball:cball z e ⊆ s using ‹open s› ‹z∈s›
    using open_contains_cball_eq by blast
  have (f' has_contour_integral c' ∗ residue f' z) (circlepath z e)
    unfolding f'_def using f_holo
    apply (intro base_residue[OF ‹open s› ‹z∈s› ‹e>0› _ e_cball,folded c'_def])
    by (auto intro:holomorphic_intros)
```

```
  moreover have (f' has_contour_integral c * (c' * residue f z)) (circlepath z e)
    unfolding f'_def using f_holo
    by (auto intro: has_contour_integral_lmul
      base_residue[OF ‹open s› ‹z∈s› ‹e>0› _ e_cball,folded c'_def])
  ultimately have c' * residue f' z = c * (c' * residue f z)
    using has_contour_integral_unique by auto
  thus ?thesis unfolding f'_def c'_def using False
    by (auto simp add:field_simps)
qed

lemma residue_rmul:
  assumes open s z ∈ s and f_holo: f holomorphic_on s - {z}
  shows residue (λz. (f z) * c) z= residue f z * c
using residue_lmul[OF assms,of c] by (auto simp add:algebra_simps)

lemma residue_div:
  assumes open s z ∈ s and f_holo: f holomorphic_on s - {z}
  shows residue (λz. (f z) / c) z= residue f z / c
using residue_lmul[OF assms,of 1/c] by (auto simp add:algebra_simps)

lemma residue_neg:
  assumes open s z ∈ s and f_holo: f holomorphic_on s - {z}
  shows residue (λz. - (f z)) z= - residue f z
using residue_lmul[OF assms,of -1] by auto

lemma residue_diff:
  assumes open s z ∈ s and f_holo: f holomorphic_on s - {z}
     and g_holo:g holomorphic_on s - {z}
  shows residue (λz. f z - g z) z= residue f z - residue g z
using residue_add[OF assms(1,2,3),of λz. - g z] residue_neg[OF assms(1,2,4)]
by (auto intro:holomorphic_intros g_holo)

lemma residue_simple:
  assumes open s z∈s and f_holo:f holomorphic_on s
  shows residue (λw. f w / (w - z)) z = f z
proof -
  define c where c ≡ 2 * pi * ii
  define f' where f' ≡ λw. f w / (w - z)
  obtain e where e>0 and e_cball:cball z e ⊆ s using ‹open s› ‹z∈s›
    using open_contains_cball_eq by blast
  have (f' has_contour_integral c * f z) (circlepath z e)
    unfolding f'_def c_def using ‹e>0› f_holo e_cball
    by (auto intro!: Cauchy_integral_circlepath_simple holomorphic_intros)
  moreover have (f' has_contour_integral c * residue f' z) (circlepath z e)
    unfolding f'_def using f_holo
    apply (intro base_residue[OF ‹open s› ‹z∈s› ‹e>0› _ e_cball,folded c_def])
    by (auto intro!:holomorphic_intros)
  ultimately have c * f z = c * residue f' z
    using has_contour_integral_unique by blast
```

```
  thus ?thesis unfolding c_def f'_def  by auto
qed

lemma residue_simple':
  assumes s: open s z ∈ s and holo: f holomorphic_on (s − {z})
      and lim: ((λw. f w ∗ (w − z)) ⟶ c) (at z)
  shows   residue f z = c
proof −
  define g where g = (λw. if w = z then c else f w ∗ (w − z))
  from holo have (λw. f w ∗ (w − z)) holomorphic_on (s − {z}) (is ?P)
    by (force intro: holomorphic_intros)
  also have ?P ⟷ g holomorphic_on (s − {z})
    by (intro holomorphic_cong refl) (simp_all add: g_def)
  finally have ∗: g holomorphic_on (s − {z}) .

  note lim
  also have (λw. f w ∗ (w − z)) −z→ c ⟷ g −z→ g z
   by (intro filterlim_cong refl) (simp_all add: g_def [abs_def] eventually_at_filter)
  finally have ∗∗: g −z→ g z .

  have g_holo: g holomorphic_on s
    by (rule no_isolated_singularity'[where K = {z}])
       (insert assms ∗ ∗∗, simp_all add: at_within_open_NO_MATCH)
  from s and this have residue (λw. g w / (w − z)) z = g z
    by (rule residue_simple)
  also have ∀F za in at z. g za / (za − z) = f za
    unfolding eventually_at by (auto intro!: exI[of _ 1] simp: field_simps g_def)
  hence residue (λw. g w / (w − z)) z = residue f z
    by (intro residue_cong refl)
  finally show ?thesis
    by (simp add: g_def)
qed

lemma residue_holomorphic_over_power:
  assumes open A z0 ∈ A f holomorphic_on A
  shows   residue (λz. f z / (z − z0) ^ Suc n) z0 = (deriv ^^ n) f z0 / fact n
proof −
  let ?f = λz. f z / (z − z0) ^ Suc n
  from assms(1,2) obtain r where r: r > 0 cball z0 r ⊆ A
    by (auto simp: open_contains_cball)
  have (?f has_contour_integral 2 ∗ pi ∗ i ∗ residue ?f z0) (circlepath z0 r)
    using r assms by (intro base_residue[of A]) (auto intro!: holomorphic_intros)
  moreover have (?f has_contour_integral 2 ∗ pi ∗ i / fact n ∗ (deriv ^^ n) f
z0) (circlepath z0 r)
    using assms r
    by (intro Cauchy_has_contour_integral_higher_derivative_circlepath)
     (auto intro!: holomorphic_on_subset[OF assms(3)] holomorphic_on_imp_continuous_on)
  ultimately have 2 ∗ pi ∗ i ∗ residue ?f z0 = 2 ∗ pi ∗ i / fact n ∗ (deriv ^^ n)
f z0
```

```
    by (rule has_contour_integral_unique)
  thus ?thesis by (simp add: field_simps)
qed

lemma residue_holomorphic_over_power':
  assumes open A 0 ∈ A f holomorphic_on A
  shows   residue (λz. f z / z ^ Suc n) 0 = (deriv ^^ n) f 0 / fact n
  using residue_holomorphic_over_power[OF assms] by simp

theorem residue_fps_expansion_over_power_at_0:
  assumes f has_fps_expansion F
  shows   residue (λz. f z / z ^ Suc n) 0 = fps_nth F n
proof −
  from has_fps_expansion_imp_holomorphic[OF assms] obtain s
    where open s 0 ∈ s f holomorphic_on s ⋀z. z ∈ s ⟹ f z = eval_fps F z
    by auto
  with assms have residue (λz. f z / (z − 0) ^ Suc n) 0 = (deriv ^^ n) f 0 / fact
n
    unfolding has_fps_expansion_def
    by (intro residue_holomorphic_over_power[of s]) (auto simp: zero_ereal_def)
  also from assms have ... = fps_nth F n
    by (subst fps_nth_fps_expansion) auto
  finally show ?thesis by simp
qed

lemma residue_pole_order:
  fixes f::complex ⇒ complex and z::complex
  defines n ≡ nat (− zorder f z) and h ≡ zor_poly f z
  assumes f_iso:isolated_singularity_at f z
    and pole:is_pole f z
  shows residue f z = ((deriv ^^ (n − 1)) h z / fact (n−1))
proof −
  define g where g ≡ λx. if x=z then 0 else inverse (f x)
  obtain e where [simp]:e>0 and f_holo:f holomorphic_on ball z e − {z}
   using f_iso analytic_imp_holomorphic unfolding isolated_singularity_at_def
by blast
  obtain r where 0 < n 0 < r and r_cball:cball z r ⊆ ball z e and h_holo: h
holomorphic_on cball z r
     and h_divide:(∀ w∈cball z r. (w≠z ⟶ f w = h w / (w − z) ^ n) ∧ h w ≠ 0)
  proof −
    obtain r where r:zorder f z < 0 h z ≠ 0 r>0 cball z r ⊆ ball z e h holomor-
phic_on cball z r
        (∀ w∈cball z r − {z}. f w = h w / (w − z) ^ n ∧ h w ≠ 0)
      using zorder_exist_pole[OF f_holo,simplified,OF ‹is_pole f z›,folded n_def
h_def] by auto
    have n>0 using ‹zorder f z < 0› unfolding n_def by simp
    moreover have (∀ w∈cball z r. (w≠z ⟶ f w = h w / (w − z) ^ n) ∧ h w ≠
0)
      using ‹h z≠0› r(6) by blast
```

```
    ultimately show ?thesis using r(3,4,5) that by blast
  qed
  have r_nonzero:⋀w. w ∈ ball z r − {z} ⟹ f w ≠ 0
    using h_divide by simp
  define c where c ≡ 2 * pi * i
  define der_f where der_f ≡ ((deriv ^^ (n − 1)) h z / fact (n−1))
  define h' where h' ≡ λu. h u / (u − z) ^ n
   have (h' has_contour_integral c / fact (n − 1) * (deriv ^^ (n − 1)) h z)
(circlepath z r)
    unfolding h'_def
    proof (rule Cauchy_has_contour_integral_higher_derivative_circlepath[of z r
h z n−1,
        folded c_def Suc_pred'[OF ‹n>0›]])
     show continuous_on (cball z r) h using holomorphic_on_imp_continuous_on
h_holo by simp
      show h holomorphic_on ball z r using h_holo by auto
      show  z ∈ ball z r using ‹r>0› by auto
    qed
   then have (h' has_contour_integral c * der_f) (circlepath z r) unfolding
der_f_def by auto
  then have (f has_contour_integral c * der_f) (circlepath z r)
    proof (elim has_contour_integral_eq)
      fix x assume x ∈ path_image (circlepath z r)
      hence x∈cball z r − {z} using ‹r>0› by auto
      then show h' x = f x using h_divide unfolding h'_def by auto
    qed
  moreover have (f has_contour_integral c * residue f z) (circlepath z r)
     using base_residue[of ‹ball z e› z,simplified,OF ‹r>0› f_holo r_cball,folded
c_def]
    unfolding c_def by simp
 ultimately have c * der_f =  c * residue f z using has_contour_integral_unique
by blast
  hence der_f = residue f z unfolding c_def by auto
  thus ?thesis unfolding der_f_def by auto
qed

lemma residue_simple_pole:
  assumes isolated_singularity_at f z0
  assumes is_pole f z0 zorder f z0 = − 1
  shows   residue f z0 = zor_poly f z0 z0
  using assms by (subst residue_pole_order) simp_all

lemma residue_simple_pole_limit:
  assumes isolated_singularity_at f z0
  assumes is_pole f z0 zorder f z0 = − 1
  assumes ((λx. f (g x) * (g x − z0)) ⟶ c) F
  assumes filterlim g (at z0) F F ≠ bot
  shows   residue f z0 = c
proof −
```

```
  have residue f z0 = zor_poly f z0 z0
    by (rule residue_simple_pole assms)+
  also have ... = c
    apply (rule zor_poly_pole_eqI)
    using assms by auto
  finally show ?thesis .
qed

lemma
  assumes f_holo:f holomorphic_on s and g_holo:g holomorphic_on s
        and open s connected s z ∈ s
  assumes g_deriv:(g has_field_derivative g') (at z)
  assumes f z ≠ 0 g z = 0 g' ≠ 0
  shows   porder_simple_pole_deriv: zorder (λw. f w / g w) z = - 1
    and   residue_simple_pole_deriv: residue (λw. f w / g w) z = f z / g'
proof -
  have [simp]:isolated_singularity_at f z isolated_singularity_at g z
   using isolated_singularity_at_holomorphic[OF _ ‹open s› ‹z∈s›] f_holo g_holo
    by (meson Diff_subset holomorphic_on_subset)+
  have [simp]:not_essential f z not_essential g z
    unfolding not_essential_def using f_holo g_holo assms(3,5)
   by (meson continuous_on_eq_continuous_at continuous_within holomorphic_on_imp_continuous_on)+
  have g_nconst:∃F w in at z. g w ≠0
  proof (rule ccontr)
    assume ¬ (∃F w in at z. g w ≠ 0)
    then have ∀F w in nhds z. g w = 0
      unfolding eventually_at eventually_nhds frequently_at using ‹g z = 0›
      by (metis open_ball UNIV_I centre_in_ball dist_commute mem_ball)
    then have deriv g z = deriv (λ_. 0) z
      by (intro deriv_cong_ev) auto
    then have deriv g z = 0 by auto
    then have g' = 0 using g_deriv DERIV_imp_deriv by blast
    then show False using ‹g'≠0› by auto
  qed

  have zorder (λw. f w / g w) z = zorder f z - zorder g z
  proof -
    have ∀F w in at z. f w ≠0 ∧ w∈s
      apply (rule non_zero_neighbour_alt)
      using assms by auto
    with g_nconst have ∃F w in at z. f w * g w ≠ 0
      by (elim frequently_rev_mp eventually_rev_mp,auto)
    then show ?thesis using zorder_divide[of f z g] by auto
  qed
  moreover have zorder f z=0
    apply (rule zorder_zero_eqI[OF f_holo ‹open s› ‹z∈s›])
    using ‹f z≠0› by auto
  moreover have zorder g z=1
    apply (rule zorder_zero_eqI[OF g_holo ‹open s› ‹z∈s›])
```

```
    subgoal using assms(8) by auto
    subgoal using DERIV_imp_deriv assms(9) g_deriv by auto
    subgoal by simp
    done
  ultimately show zorder (λw. f w / g w) z = − 1 by auto

  show residue (λw. f w / g w) z = f z / g'
  proof (rule residue_simple_pole_limit[where g=id and F=at z,simplified])
    show zorder (λw. f w / g w) z = − 1 by fact
    show isolated_singularity_at (λw. f w / g w) z
      by (auto intro: singularity_intros)
    show is_pole (λw. f w / g w) z
    proof (rule is_pole_divide)
      have ∀F x in at z. g x ≠ 0
        apply (rule non_zero_neighbour)
        using g_nconst by auto
      moreover have g −z→ 0
        using DERIV_isCont assms(8) continuous_at g_deriv by force
      ultimately show filterlim g (at 0) (at z) unfolding filterlim_at by simp
      show isCont f z
           using assms(3,5) continuous_on_eq_continuous_at f_holo holomor-
phic_on_imp_continuous_on
        by auto
      show f z ≠ 0 by fact
    qed
    show filterlim id (at z) (at z) by (simp add: filterlim_iff)
    have ((λw. (f w ∗ (w − z)) / g w) ⟶ f z / g') (at z)
    proof (rule lhopital_complex_simple)
      show ((λw. f w ∗ (w − z)) has_field_derivative f z) (at z)
        using assms by (auto intro!: derivative_eq_intros holomorphic_derivI[OF
f_holo])
      show (g has_field_derivative g') (at z) by fact
    qed (insert assms, auto)
    then show ((λw. (f w / g w) ∗ (w − z)) ⟶ f z / g') (at z)
      by (simp add: field_split_simps)
  qed
qed
```

5.12 Poles and residues of some well-known functions

```
lemma is_pole_Gamma: is_pole Gamma (−of_nat n)
  unfolding is_pole_def using Gamma_poles .

lemma Gamma_residue:
  residue Gamma (−of_nat n) = (−1) ^ n / fact n
proof (rule residue_simple')
  show open (− (ℤ≤0 − {−of_nat n}) :: complex set)
    by (intro open_Compl closed_subset_Ints) auto
  show Gamma holomorphic_on (− (ℤ≤0 − {−of_nat n}) − {− of_nat n})
```

by (*rule holomorphic_Gamma*) *auto*
show (λ*w*. *Gamma w* ∗ (*w* − (−*of_nat n*))) −(−*of_nat n*)→ (− *1*) ^ *n* / *fact n*
using *Gamma_residues*[*of n*] **by** *simp*
qed *auto*

end

6 The Residue Theorem, the Argument Principle and Rouché's Theorem

theory *Residue_Theorem*
imports *Complex_Residues HOL−Library.Landau_Symbols*
begin

6.1 Cauchy's residue theorem

lemma *get_integrable_path*:
assumes *open s connected* (*s−pts*) *finite pts f holomorphic_on* (*s−pts*) *a∈s−pts b∈s−pts*
obtains *g* **where** *valid_path g pathstart g* = *a pathfinish g* = *b*
path_image g ⊆ *s−pts f contour_integrable_on g* **using** *assms*
proof (*induct arbitrary*:*s thesis a rule*:*finite_induct*[*OF* ‹*finite pts*›])
case *1*
obtain *g* **where** *valid_path g path_image g* ⊆ *s pathstart g* = *a pathfinish g* = *b*
using *connected_open_polynomial_connected*[*OF* ‹*open s*›,*of a b*] ‹*connected* (*s* − {})›
valid_path_polynomial_function 1.prems(*6*) *1.prems*(*7*) **by** *auto*
moreover have *f contour_integrable_on g*
using *contour_integrable_holomorphic_simple*[*OF* _ ‹*open s*› ‹*valid_path g*› ‹*path_image g* ⊆ *s*›,*of f*]
‹*f holomorphic_on s* − {}›
by *auto*
ultimately show *?case* **using** *1*(*1*)[*of g*] **by** *auto*
next
case *idt*:(*2 p pts*)
obtain *e* **where** *e*>*0* **and** *e*:∀ *w*∈*ball a e*. *w* ∈ *s* ∧ (*w* ≠ *a* ⟶ *w* ∉ *insert p pts*)
using *finite_ball_avoid*[*OF* ‹*open s*› ‹*finite* (*insert p pts*)›, *of a*]
‹*a* ∈ *s* − *insert p pts*›
by *auto*
define *a′* **where** *a′* ≡ *a*+*e*/*2*
have *a′*∈*s*−{*p*} −*pts* **using** *e*[*rule_format*,*of a*+*e*/*2*] ‹*e*>*0*›
by (*auto simp add*:*dist_complex_def a′_def*)
then obtain *g′* **where** *g′*[*simp*]:*valid_path g′ pathstart g′* = *a′ pathfinish g′* = *b*
path_image g′ ⊆ *s* − {*p*} − *pts f contour_integrable_on g′*
using *idt.hyps*(*3*)[*of a′ s*−{*p*}] *idt.prems idt.hyps*(*1*)
by (*metis Diff_insert2 open_delete*)
define *g* **where** *g* ≡ *linepath a a′* +++ *g′*
have *valid_path g* **unfolding** *g_def* **by** (*auto intro*: *valid_path_join*)

```
  moreover have pathstart g = a and  pathfinish g = b unfolding g_def by auto
  moreover have path_image g ⊆ s − insert p pts unfolding g_def
    proof (rule subset_path_image_join)
      have closed_segment a a' ⊆ ball a e using ‹e>0›
      by (auto dest!:segment_bound1 simp:a'_def dist_complex_def norm_minus_commute)
      then show path_image (linepath a a') ⊆ s − insert p pts using e idt(9)
        by auto
    next
      show path_image g' ⊆ s − insert p pts using g'(4) by blast
    qed
  moreover have f contour_integrable_on g
    proof −
      have closed_segment a a' ⊆ ball a e using ‹e>0›
      by (auto dest!:segment_bound1 simp:a'_def dist_complex_def norm_minus_commute)
      then have continuous_on (closed_segment a a') f
      using e idt.prems(6) holomorphic_on_imp_continuous_on[OF idt.prems(5)]
        apply (elim continuous_on_subset)
        by auto
      then have f contour_integrable_on linepath a a'
        using contour_integrable_continuous_linepath by auto
      then show ?thesis unfolding g_def
        apply (rule contour_integrable_joinI)
        by (auto simp add: ‹e>0›)
    qed
  ultimately show ?case using idt.prems(1)[of g] by auto
qed

lemma Cauchy_theorem_aux:
  assumes open s connected (s−pts) finite pts pts ⊆ s f holomorphic_on s−pts
          valid_path g pathfinish g = pathstart g path_image g ⊆ s−pts
          ∀ z. (z ∉ s) ⟶ winding_number g z  = 0
          ∀ p∈s. h p>0 ∧ (∀ w∈cball p (h p). w∈s ∧ (w≠p ⟶ w ∉ pts))
  shows contour_integral g f = (∑ p∈pts. winding_number g p ∗ contour_integral
(circlepath p (h p)) f)
    using assms
proof (induct arbitrary:s g rule:finite_induct[OF ‹finite pts›])
  case 1
 then show ?case by (simp add: Cauchy_theorem_global contour_integral_unique)
next
  case (2 p pts)
  note fin[simp] = ‹finite (insert p pts)›
    and connected = ‹connected (s − insert p pts)›
    and valid[simp] = ‹valid_path g›
    and g_loop[simp] = ‹pathfinish g = pathstart g›
    and holo[simp]= ‹f holomorphic_on s − insert p pts›
    and path_img = ‹path_image g ⊆ s − insert p pts›
    and winding = ‹∀ z. z ∉ s ⟶ winding_number g z = 0›
    and h = ‹∀ pa∈s. 0 < h pa ∧ (∀ w∈cball pa (h pa). w ∈ s ∧ (w ≠ pa ⟶ w ∉
insert p pts))›
```

have *h p>0* **and** *p∈s*
and *h_p*: ∀ *w∈cball p (h p). w ∈ s ∧ (w ≠ p ⟶ w ∉ insert p pts)*
using *h ‹insert p pts ⊆ s›* **by** *auto*
obtain *pg* **where** *pg[simp]*: *valid_path pg pathstart pg = pathstart g pathfinish pg=p+h p*
path_image pg ⊆ s−insert p pts f contour_integrable_on pg
proof −
have *p + h p∈cball p (h p)* **using** *h[rule_format,of p]*
by (*simp add*: *‹p ∈ s› dist_norm*)
then have *p + h p ∈ s − insert p pts* **using** *h[rule_format,of p] ‹insert p pts ⊆ s›*
by *fastforce*
moreover have *pathstart g ∈ s − insert p pts* **using** *path_img* **by** *auto*
ultimately show *?thesis*
using *get_integrable_path[OF ‹open s› connected fin holo,of pathstart g p+h p] that*
by *blast*
qed
obtain *n*::*int* **where** *n=winding_number g p*
using *integer_winding_number[OF _ g_loop,of p] valid path_img*
by (*metis DiffD2 Ints_cases insertI1 subset_eq valid_path_imp_path*)
define *p_circ* **where** *p_circ ≡ circlepath p (h p)*
define *p_circ_pt* **where** *p_circ_pt ≡ linepath (p+h p) (p+h p)*
define *n_circ* **where** *n_circ ≡ λn. ((+++) p_circ ^^ n) p_circ_pt*
define *cp* **where** *cp ≡ if n≥0 then reversepath (n_circ (nat n)) else n_circ (nat (− n))*
have *n_circ*:*valid_path (n_circ k)*
winding_number (n_circ k) p = k
pathstart (n_circ k) = p + h p pathfinish (n_circ k) = p + h p
path_image (n_circ k) = (if k=0 then {p + h p} else sphere p (h p))
p ∉ path_image (n_circ k)
⋀*p′. p′∉s − pts ⟹ winding_number (n_circ k) p′=0 ∧ p′∉path_image (n_circ k)*
f contour_integrable_on (n_circ k)
contour_integral (n_circ k) f = k ∗ contour_integral p_circ f
for *k*
proof (*induct k*)
case *0*
show *valid_path (n_circ 0)*
and *path_image (n_circ 0) = (if 0=0 then {p + h p} else sphere p (h p))*
and *winding_number (n_circ 0) p = of_nat 0*
and *pathstart (n_circ 0) = p + h p*
and *pathfinish (n_circ 0) = p + h p*
and *p ∉ path_image (n_circ 0)*
unfolding *n_circ_def p_circ_pt_def* **using** *‹h p > 0›*
by (*auto simp add*: *dist_norm*)
show *winding_number (n_circ 0) p′=0 ∧ p′∉path_image (n_circ 0)* **when** *p′∉s− pts* **for** *p′*
unfolding *n_circ_def p_circ_pt_def*

```
      apply (auto intro!:winding_number_trivial)
       by (metis Diff_iff pathfinish_in_path_image pg(3) pg(4) subsetCE sub-
set_insertI that)+
     show f contour_integrable_on (n_circ 0)
      unfolding n_circ_def p_circ_pt_def
     by (auto intro!:contour_integrable_continuous_linepath simp add:continuous_on_sing)
     show contour_integral (n_circ 0) f = of_nat 0 * contour_integral p_circ f
      unfolding n_circ_def p_circ_pt_def by auto
   next
     case (Suc k)
     have n_Suc:n_circ (Suc k) = p_circ +++ n_circ k unfolding n_circ_def
by auto
      have pcirc:p ∉ path_image p_circ valid_path p_circ pathfinish p_circ =
pathstart (n_circ k)
      using Suc(3) unfolding p_circ_def using ‹h p > 0› by (auto simp add:
p_circ_def)
     have pcirc_image:path_image p_circ ⊆ s − insert p pts
      proof −
       have path_image p_circ ⊆ cball p (h p) using ‹0 < h p› p_circ_def by
auto
       then show ?thesis using h_p pcirc(1) by auto
      qed
     have pcirc_integrable:f contour_integrable_on p_circ
      by (auto simp add:p_circ_def intro!: pcirc_image[unfolded p_circ_def]
     contour_integrable_continuous_circlepath holomorphic_on_imp_continuous_on
       holomorphic_on_subset[OF holo])
     show valid_path (n_circ (Suc k))
      using valid_path_join[OF pcirc(2) Suc(1) pcirc(3)] unfolding n_circ_def
by auto
     show path_image (n_circ (Suc k))
       = (if Suc k = 0 then {p + complex_of_real (h p)} else sphere p (h p))
      proof −
       have path_image p_circ = sphere p (h p)
        unfolding p_circ_def using ‹0 < h p› by auto
       then show ?thesis unfolding n_Suc using Suc.hyps(5) ‹h p>0›
        by (auto simp add: path_image_join[OF pcirc(3)] dist_norm)
      qed
     then show p ∉ path_image (n_circ (Suc k)) using ‹h p>0› by auto
     show winding_number (n_circ (Suc k)) p = of_nat (Suc k)
      proof −
       have winding_number p_circ p = 1
        by (simp add: ‹h p > 0› p_circ_def winding_number_circlepath_centre)
        moreover have p ∉ path_image (n_circ k) using Suc(5) ‹h p>0› by
auto
       then have winding_number (p_circ +++ n_circ k) p
         = winding_number p_circ p + winding_number (n_circ k) p
        using valid_path_imp_path Suc.hyps(1) Suc.hyps(2) pcirc
        apply (intro winding_number_join)
        by auto
```

```
        ultimately show ?thesis using Suc(2) unfolding n_circ_def
          by auto
      qed
    show pathstart (n_circ (Suc k)) = p + h p
      by (simp add: n_circ_def p_circ_def)
    show pathfinish (n_circ (Suc k)) = p + h p
      using Suc(4) unfolding n_circ_def by auto
     show winding_number (n_circ (Suc k)) p'=0 ∧  p'∉path_image (n_circ
(Suc k)) when p'∉s-pts for p'
      proof -
        have  p' ∉ path_image p_circ using ‹p ∈ s› h p_circ_def that using
pcirc_image by blast
        moreover have p' ∉ path_image (n_circ k)
          using Suc.hyps(7) that by blast
        moreover have winding_number p_circ p' = 0
          proof -
            have path_image p_circ ⊆ cball p (h p)
              using h unfolding p_circ_def using ‹p ∈ s› by fastforce
             moreover have p'∉cball p (h p) using ‹p ∈ s› h that 2.hyps(2) by
fastforce
            ultimately show ?thesis unfolding p_circ_def
              apply (intro winding_number_zero_outside)
              by auto
          qed
        ultimately show ?thesis
          unfolding n_Suc
          apply (subst winding_number_join)
       by (auto simp: valid_path_imp_path pcirc Suc that not_in_path_image_join
Suc.hyps(7)[OF that])
      qed
    show f contour_integrable_on (n_circ (Suc k))
      unfolding n_Suc
        by (rule contour_integrable_joinI[OF pcirc_integrable Suc(8) pcirc(2)
Suc(1)])
      show contour_integral (n_circ (Suc k)) f = (Suc k) *  contour_integral
p_circ f
      unfolding n_Suc
    by (auto simp add:contour_integral_join[OF pcirc_integrable Suc(8) pcirc(2)
Suc(1)]
        Suc(9) algebra_simps)
  qed
 have cp[simp]:pathstart cp = p + h p  pathfinish cp = p + h p
      valid_path cp path_image cp ⊆ s - insert p pts
      winding_number cp p = - n
      ⋀p'. p'∉s - pts ⟹ winding_number cp p'=0 ∧ p' ∉ path_image cp
      f contour_integrable_on cp
      contour_integral cp f = - n * contour_integral p_circ f
  proof -
    show pathstart cp = p + h p and pathfinish cp = p + h p and valid_path cp
```

```
      using n_circ unfolding cp_def by auto
    next
      have sphere p (h p) ⊆  s − insert p pts
        using h[rule_format,of p] ‹insert p pts ⊆ s› by force
      moreover  have p + complex_of_real (h p) ∈ s − insert p pts
        using pg(3) pg(4) by (metis pathfinish_in_path_image subsetCE)
      ultimately show path_image cp ⊆  s − insert p pts unfolding cp_def
        using n_circ(5)  by auto
    next
      show winding_number cp p = − n
        unfolding cp_def using winding_number_reversepath n_circ ‹h p>0›
        by (auto simp: valid_path_imp_path)
    next
      show winding_number cp p'=0 ∧ p' ∉ path_image cp when p'∉s − pts for
p'
        unfolding cp_def
        apply (auto)
        apply (subst winding_number_reversepath)
        by (auto simp add: valid_path_imp_path n_circ(7)[OF that] n_circ(1))
    next
      show f contour_integrable_on cp unfolding cp_def
        using contour_integrable_reversepath_eq n_circ(1,8) by auto
    next
      show contour_integral cp f = − n ∗ contour_integral p_circ f
          unfolding cp_def using contour_integral_reversepath[OF n_circ(1)]
n_circ(9)
        by auto
    qed
  define g' where g' ≡ g +++ pg +++ cp +++ (reversepath pg)
  have contour_integral g' f = (∑ p∈pts. winding_number g' p ∗ contour_integral
(circlepath p (h p)) f)
    proof (rule 2.hyps(3)[of s−{p} g',OF _ _ ‹finite pts› ])
      show connected (s − {p} − pts) using connected by (metis Diff_insert2)
      show open (s − {p}) using ‹open s› by auto
      show  pts ⊆ s − {p} using ‹insert p pts ⊆ s› ‹ p ∉ pts›  by blast
        show f holomorphic_on s − {p} − pts using holo ‹p ∉ pts› by (metis
Diff_insert2)
      show valid_path g'
        unfolding g'_def cp_def using n_circ valid pg g_loop
        by (auto intro!:valid_path_join )
      show pathfinish g' = pathstart g'
        unfolding g'_def cp_def using pg(2) by simp
      show path_image g' ⊆ s − {p} − pts
        proof −
          define s' where s' ≡ s − {p} − pts
          have s':s' = s−insert p pts  unfolding s'_def by auto
          then show ?thesis using path_img pg(4) cp(4)
            unfolding g'_def
            apply (fold s'_def s')
```

```
        apply (intro subset_path_image_join)
        by auto
      qed
    note path_join_imp[simp]
    show ∀ z. z ∉ s − {p} ⟶ winding_number g' z = 0
      proof clarify
        fix z assume z:z∉s − {p}
        have winding_number (g +++ pg +++ cp +++ reversepath pg) z =
winding_number g z
          + winding_number (pg +++ cp +++ (reversepath pg)) z
        proof (rule winding_number_join)
        show path g using ‹valid_path g› by (simp add: valid_path_imp_path)
          show z ∉ path_image g using z path_img by auto
          show path (pg +++ cp +++ reversepath pg) using pg(3) cp
            by (simp add: valid_path_imp_path)
        next
         have path_image (pg +++ cp +++ reversepath pg) ⊆ s − insert p pts
           using pg(4) cp(4) by (auto simp:subset_path_image_join)
          then show z ∉ path_image (pg +++ cp +++ reversepath pg) using
z by auto
        next
         show pathfinish g = pathstart (pg +++ cp +++ reversepath pg) using
g_loop by auto
        qed
       also have ... = winding_number g z + (winding_number pg z
          + winding_number (cp +++ (reversepath pg)) z)
        proof (subst add_left_cancel,rule winding_number_join)
          show path pg and path (cp +++ reversepath pg)
           and pathfinish pg = pathstart (cp +++ reversepath pg)
            by (auto simp add: valid_path_imp_path)
          show z ∉ path_image pg using pg(4) z by blast
          show z ∉ path_image (cp +++ reversepath pg) using z
          by (metis Diff_iff ‹z ∉ path_image pg› contra_subsetD cp(4) insertI1
              not_in_path_image_join path_image_reversepath singletonD)
        qed
       also have ... = winding_number g z + (winding_number pg z
          + (winding_number cp z + winding_number (reversepath pg) z))
        apply (auto intro!:winding_number_join simp: valid_path_imp_path)
        apply (metis Diff_iff contra_subsetD cp(4) insertI1 singletonD z)
        by (metis Diff_insert2 Diff_subset contra_subsetD pg(4) z)
       also have ... = winding_number g z + winding_number cp z
        apply (subst winding_number_reversepath)
        apply (auto simp: valid_path_imp_path)
        by (metis Diff_iff contra_subsetD insertI1 pg(4) singletonD z)
         finally have winding_number g' z = winding_number g z + wind-
ing_number cp z
        unfolding g'_def .
       moreover have winding_number g z + winding_number cp z = 0
        using winding z ‹n=winding_number g p› by auto
```

```
      ultimately show winding_number g' z = 0 unfolding g'_def by auto
    qed
   show ∀pa∈s − {p}. 0 < h pa ∧ (∀w∈cball pa (h pa). w ∈ s − {p} ∧ (w ≠
pa ⟶ w ∉ pts))
    using h by fastforce
  qed
 moreover have contour_integral g' f = contour_integral g f
   − winding_number g p ∗ contour_integral p_circ f
  proof −
   have contour_integral g' f =  contour_integral g f
    + contour_integral (pg +++ cp +++ reversepath pg) f
    unfolding g'_def
    apply (subst contour_integral_join)
    by (auto simp add:open_Diff[OF ‹open s›,OF finite_imp_closed[OF fin]]
     intro!: contour_integrable_holomorphic_simple[OF holo _ _ path_img]
     contour_integrable_reversepath)
   also have ... = contour_integral g f + contour_integral pg f
     + contour_integral (cp +++ reversepath pg) f
    apply (subst contour_integral_join)
    by (auto simp add:contour_integrable_reversepath)
   also have ... = contour_integral g f + contour_integral pg f
     + contour_integral cp f + contour_integral (reversepath pg) f
    apply (subst contour_integral_join)
    by (auto simp add:contour_integrable_reversepath)
   also have ... = contour_integral g f + contour_integral cp f
    using contour_integral_reversepath
    by (auto simp add:contour_integrable_reversepath)
  also have ... = contour_integral g f − winding_number g p ∗ contour_integral
p_circ f
    using ‹n=winding_number g p› by auto
   finally show ?thesis .
  qed
 moreover have winding_number g' p' = winding_number g p' when p'∈pts for
p'
  proof −
   have [simp]: p' ∉ path_image g p' ∉ path_image pg p'∉path_image cp
    using 2.prems(8) that
    apply blast
    apply (metis Diff_iff Diff_insert2 contra_subsetD pg(4) that)
    by (meson DiffD2 cp(4) rev_subsetD subset_insertI that)
   have winding_number g' p' = winding_number g p'
    + winding_number (pg +++ cp +++ reversepath pg) p' unfolding g'_def
    apply (subst winding_number_join)
    apply (simp_all add: valid_path_imp_path)
    apply (intro not_in_path_image_join)
    by auto
   also have ... = winding_number g p' + winding_number pg p'
     + winding_number (cp +++ reversepath pg) p'
    apply (subst winding_number_join)
```

apply (*simp_all add*: *valid_path_imp_path*)
apply (*intro not_in_path_image_join*)
by *auto*
also have ... = *winding_number g p′* + *winding_number pg p′*+ *winding_number cp p′*
+ *winding_number* (*reversepath pg*) *p′*
apply (*subst winding_number_join*)
by (*simp_all add*: *valid_path_imp_path*)
also have ... = *winding_number g p′* + *winding_number cp p′*
apply (*subst winding_number_reversepath*)
by (*simp_all add*: *valid_path_imp_path*)
also have ... = *winding_number g p′* **using** *that* **by** *auto*
finally show *?thesis* **.**
qed
ultimately show *?case* **unfolding** *p_circ_def*
apply (*subst* (*asm*) *sum.cong*[*OF refl*,
of pts _ *λp. winding_number g p* ∗ *contour_integral* (*circlepath p* (*h p*)) *f*])
by (*auto simp add*:*sum.insert*[*OF* ‹*finite pts*› ‹*p*∉*pts*›] *algebra_simps*)
qed

lemma *Cauchy_theorem_singularities*:
assumes *open s connected s finite pts* **and**
holo:*f holomorphic_on s*−*pts* **and**
valid_path g **and**
loop:*pathfinish g* = *pathstart g* **and**
path_image g ⊆ *s*−*pts* **and**
homo:∀ *z*. (*z* ∉ *s*) ⟶ *winding_number g z* = *0* **and**
avoid:∀ *p*∈*s*. *h p*>*0* ∧ (∀ *w*∈*cball p* (*h p*). *w*∈*s* ∧ (*w*≠*p* ⟶ *w* ∉ *pts*))
shows *contour_integral g f* = (∑ *p*∈*pts*. *winding_number g p* ∗ *contour_integral* (*circlepath p* (*h p*)) *f*)
(**is** *?L*=*?R*)
proof −
define *circ* **where** *circ* ≡ *λp. winding_number g p* ∗ *contour_integral* (*circlepath p* (*h p*)) *f*
define *pts1* **where** *pts1* ≡ *pts* ∩ *s*
define *pts2* **where** *pts2* ≡ *pts* − *pts1*
have *pts*=*pts1* ∪ *pts2 pts1* ∩ *pts2* = {} *pts2* ∩ *s*={} *pts1*⊆*s*
unfolding *pts1_def pts2_def* **by** *auto*
have *contour_integral g f* = (∑ *p*∈*pts1*. *circ p*) **unfolding** *circ_def*
proof (*rule Cauchy_theorem_aux*[*OF* ‹*open s*› _ _ ‹*pts1*⊆*s*› _ ‹*valid_path g*› *loop* _ *homo*])
have *finite pts1* **unfolding** *pts1_def* **using** ‹*finite pts*› **by** *auto*
then show *connected* (*s* − *pts1*)
using ‹*open s*› ‹*connected s*› *connected_open_delete_finite*[*of s*] **by** *auto*
next
show *finite pts1* **using** ‹*pts* = *pts1* ∪ *pts2*› *assms*(*3*) **by** *auto*
show *f holomorphic_on s* − *pts1* **by** (*metis Diff_Int2 Int_absorb holo pts1_def*)
show *path_image g* ⊆ *s* − *pts1* **using** *assms*(*7*) *pts1_def* **by** *auto*

```
    show ∀ p∈s. 0 < h p ∧ (∀ w∈cball p (h p). w ∈ s ∧ (w ≠ p ⟶ w ∉ pts1))
      by (simp add: avoid pts1_def)
  qed
  moreover have sum circ pts2=0
  proof −
    have winding_number g p=0 when p∈pts2 for p
      using ‹pts2 ∩ s={}› that homo[rule_format,of p] by auto
    thus ?thesis unfolding circ_def
      apply (intro sum.neutral)
      by auto
  qed
  moreover have ?R=sum circ pts1 + sum circ pts2
  unfolding circ_def
  using sum.union_disjoint[OF _ _ ‹pts1 ∩ pts2 = {}›] ‹finite pts› ‹pts=pts1
∪ pts2›
  by blast
  ultimately show ?thesis
  apply (fold circ_def)
  by auto
qed

theorem Residue_theorem:
  fixes s pts::complex set and f::complex ⇒ complex
  and g::real ⇒ complex
  assumes open s connected s finite pts and
        holo:f holomorphic_on s−pts and
        valid_path g and
        loop:pathfinish g = pathstart g and
        path_image g ⊆ s−pts and
        homo:∀ z. (z ∉ s) ⟶ winding_number g z = 0
  shows contour_integral g f = 2 * pi * i *(∑ p∈pts. winding_number g p *
residue f p)
proof −
  define c where c ≡ 2 * pi * i
  obtain h where avoid:∀ p∈s. h p>0 ∧ (∀ w∈cball p (h p). w∈s ∧ (w≠p ⟶ w
∉ pts))
  using finite_cball_avoid[OF ‹open s› ‹finite pts›] by metis
  have contour_integral g f
    = (∑ p∈pts. winding_number g p * contour_integral (circlepath p (h p)) f)
  using Cauchy_theorem_singularities[OF assms avoid] .
  also have ... = (∑ p∈pts. c * winding_number g p * residue f p)
  proof (intro sum.cong)
    show pts = pts by simp
  next
    fix x assume x ∈ pts
    show winding_number g x * contour_integral (circlepath x (h x)) f
      = c * winding_number g x * residue f x
    proof (cases x∈s)
      case False
```

```
        then have winding_number g x=0 using homo by auto
        thus ?thesis by auto
      next
        case True
        have contour_integral (circlepath x (h x)) f = c* residue f x
          using ‹x∈pts› ‹finite pts› avoid[rule_format,OF True]
       apply (intro base_residue[of s−(pts−{x}),THEN contour_integral_unique,folded
c_def])
         by (auto intro:holomorphic_on_subset[OF holo] open_Diff[OF ‹open s›
finite_imp_closed])
        then show ?thesis by auto
      qed
  qed
 also have ... = c * (∑ p∈pts. winding_number g p * residue f p)
  by (simp add: sum_distrib_left algebra_simps)
 finally show ?thesis unfolding c_def .
qed
```

6.2 The argument principle

```
theorem argument_principle:
 fixes f::complex ⇒ complex and poles s:: complex set
 defines pz ≡ {w. f w = 0 ∨ w ∈ poles} — pz is the set of poles and zeros
 assumes open s and
        connected s and
        f_holo:f holomorphic_on s−poles and
        h_holo:h holomorphic_on s and
        valid_path g and
        loop:pathfinish g = pathstart g and
        path_img:path_image g ⊆ s − pz and
        homo:∀ z. (z ∉ s) ⟶ winding_number g z = 0 and
        finite:finite pz and
        poles:∀ p∈poles. is_pole f p
 shows contour_integral g (λx. deriv f x * h x / f x) = 2 * pi * i *
        (∑ p∈pz. winding_number g p * h p * zorder f p)
   (is ?L=?R)
proof −
 define c where c ≡ 2 * complex_of_real pi * i
 define ff where ff ≡ (λx. deriv f x * h x / f x)
 define cont where cont ≡ λff p e. (ff has_contour_integral c * zorder f p * h p
) (circlepath p e)
 define avoid where avoid ≡ λp e. ∀ w∈cball p e. w ∈ s ∧ (w ≠ p ⟶ w ∉ pz)

 have ∃ e>0. avoid p e ∧ (p∈pz ⟶ cont ff p e) when p∈s for p
 proof −
   obtain e1 where e1>0 and e1_avoid:avoid p e1
     using finite_cball_avoid[OF ‹open s› finite] ‹p∈s› unfolding avoid_def by
auto
   have ∃ e2>0. cball p e2 ⊆ ball p e1 ∧ cont ff p e2 when p∈pz
```

```
  proof −
    define po where po ≡ zorder f p
    define pp where pp ≡ zor_poly f p
    define f' where f' ≡ λw. pp w * (w − p) powr po
    define ff' where ff' ≡ (λx. deriv f' x * h x / f' x)
    obtain r where pp p≠0 r>0 and
      r<e1 and
      pp_holo:pp holomorphic_on cball p r and
      pp_po:(∀ w∈cball p r−{p}. f w = pp w * (w − p) powr po ∧ pp w ≠ 0)
    proof −
      have isolated_singularity_at f p
      proof −
        have f holomorphic_on ball p e1 − {p}
          apply (intro holomorphic_on_subset[OF f_holo])
          using e1_avoid ‹p∈pz› unfolding avoid_def pz_def by force
        then show ?thesis unfolding isolated_singularity_at_def
          using ‹e1>0› analytic_on_open open_delete by blast
      qed
      moreover have not_essential f p
      proof (cases is_pole f p)
        case True
        then show ?thesis unfolding not_essential_def by auto
      next
        case False
        then have p∈s−poles using ‹p∈s› poles unfolding pz_def by auto
        moreover have open (s−poles)
          using ‹open s›
          apply (elim open_Diff)
          apply (rule finite_imp_closed)
          using finite unfolding pz_def by simp
        ultimately have isCont f p
              using holomorphic_on_imp_continuous_on[OF f_holo] continuous_on_eq_continuous_at
          by auto
        then show ?thesis unfolding isCont_def not_essential_def by auto
      qed
      moreover have ∃F w in at p. f w ≠ 0
      proof (rule ccontr)
        assume ¬ (∃F w in at p. f w ≠ 0)
        then have ∀F w in at p. f w= 0 unfolding frequently_def by auto
        then obtain rr where rr>0 ∀ w∈ball p rr − {p}. f w =0
          unfolding eventually_at by (auto simp add:dist_commute)
        then have ball p rr − {p} ⊆ {w∈ball p rr−{p}. f w=0} by blast
          moreover have infinite (ball p rr − {p}) using ‹rr>0› using finite_imp_not_open by fastforce
        ultimately have infinite {w∈ball p rr−{p}. f w=0} using infinite_super by blast
        then have infinite pz
          unfolding pz_def infinite_super by auto
```

```
      then show False using ‹finite pz› by auto
    qed
   ultimately obtain r where pp p ≠ 0 and r:r>0 pp holomorphic_on cball
p r
          (∀ w∈cball p r − {p}. f w = pp w ∗ (w − p) powr of_int po ∧ pp w
≠ 0)
      using zorder_exist[of f p,folded po_def pp_def] by auto
    define r1 where r1=min r e1 / 2
    have r1<e1 unfolding r1_def using ‹e1>0› ‹r>0› by auto
    moreover have r1>0 pp holomorphic_on cball p r1
          (∀ w∈cball p r1 − {p}. f w = pp w ∗ (w − p) powr of_int po ∧ pp
w ≠ 0)
      unfolding r1_def using ‹e1>0› r by auto
    ultimately show ?thesis using that ‹pp p≠0› by auto
  qed

  define e2 where e2 ≡ r/2
  have e2>0 using ‹r>0› unfolding e2_def by auto
  define anal where anal ≡ λw. deriv pp w ∗ h w / pp w
  define prin where prin ≡ λw. po ∗ h w / (w − p)
  have ((λw.  prin w + anal w) has_contour_integral c ∗ po ∗ h p) (circlepath
p e2)
  proof (rule has_contour_integral_add[of _ _ _ _ 0,simplified])
    have ball p r ⊆ s
        using ‹r<e1› avoid_def ball_subset_cball e1_avoid by (simp add:
subset_eq)
    then have cball p e2 ⊆ s
      using ‹r>0› unfolding e2_def by auto
    then have (λw. po ∗ h w) holomorphic_on cball p e2
      using h_holo by (auto intro!: holomorphic_intros)
    then show (prin has_contour_integral c ∗ po ∗ h p ) (circlepath p e2)
      using Cauchy_integral_circlepath_simple[folded c_def, of λw. po ∗ h w]
‹e2>0›
      unfolding prin_def by (auto simp add: mult.assoc)
    have anal holomorphic_on ball p r unfolding anal_def
      using pp_holo h_holo pp_po ‹ball p r ⊆ s› ‹pp p≠0›
      by (auto intro!: holomorphic_intros)
    then show (anal has_contour_integral 0) (circlepath p e2)
      using e2_def ‹r>0›
      by (auto elim!: Cauchy_theorem_disc_simple)
  qed
  then have cont ff′ p e2 unfolding cont_def po_def
  proof (elim has_contour_integral_eq)
    fix w assume w ∈ path_image (circlepath p e2)
    then have w∈ball p r and w≠p unfolding e2_def using ‹r>0› by auto
    define wp where wp ≡ w−p
    have wp≠0 and pp w ≠0
      unfolding wp_def using ‹w≠p› ‹w∈ball p r› pp_po by auto
    moreover have der_f′:deriv f′ w = po ∗ pp w ∗ (w−p) powr (po − 1) +
```

```
deriv pp w * (w−p) powr po
      proof (rule DERIV_imp_deriv)
        have (pp has_field_derivative (deriv pp w)) (at w)
          using DERIV_deriv_iff_has_field_derivative pp_holo ‹w≠p›
         by (meson open_ball ‹w ∈ ball p r› ball_subset_cball holomorphic_derivI
holomorphic_on_subset)
          then show  (f' has_field_derivative of_int po * pp w * (w − p) powr
of_int (po − 1)
                + deriv pp w * (w − p) powr of_int po) (at w)
          unfolding f'_def using ‹w≠p›
       by (auto intro!: derivative_eq_intros DERIV_cong[OF has_field_derivative_powr_of_int])
      qed
      ultimately show prin w + anal w = ff' w
        unfolding ff'_def prin_def anal_def
        apply simp
        apply (unfold f'_def)
        apply (fold wp_def)
        apply (auto simp add:field_simps)
          by (metis (no_types, lifting) diff_add_cancel mult.commute powr_add
powr_to_1)
    qed
    then have cont ff p e2 unfolding cont_def
    proof (elim has_contour_integral_eq)
      fix w assume w ∈ path_image (circlepath p e2)
      then have w∈ball p r and w≠p unfolding e2_def using ‹r>0› by auto
      have deriv f' w =  deriv f w
      proof (rule complex_derivative_transform_within_open[where s=ball p r
− {p}])
       show f' holomorphic_on ball p r − {p} unfolding f'_def using pp_holo
          by (auto intro!: holomorphic_intros)
      next
        have ball p e1 − {p} ⊆ s − poles
         using ball_subset_cball e1_avoid[unfolded avoid_def] unfolding pz_def
          by auto
        then have ball p r − {p} ⊆ s − poles
          apply (elim dual_order.trans)
          using ‹r<e1› by auto
        then show f holomorphic_on ball p r − {p} using f_holo
          by auto
      next
        show open (ball p r − {p}) by auto
        show w ∈ ball p r − {p} using ‹w∈ball p r› ‹w≠p› by auto
      next
        fix x assume x ∈ ball p r − {p}
        then show f' x = f x
          using pp_po unfolding f'_def by auto
      qed
      moreover have  f' w  =  f w
        using ‹w ∈ ball p r› ball_subset_cball subset_iff pp_po ‹w≠p›
```

```
        unfolding f'_def by auto
      ultimately show ff' w = ff w
        unfolding ff'_def ff_def by simp
    qed
    moreover have cball p e2 ⊆ ball p e1
      using ‹0 < r› ‹r<e1› e2_def by auto
    ultimately show ?thesis using ‹e2>0› by auto
  qed
  then obtain e2 where e2:p∈pz ⟶ e2>0 ∧ cball p e2 ⊆ ball p e1 ∧ cont ff
p e2
    by auto
  define e4 where e4 ≡ if p∈pz then e2 else  e1
  have e4>0 using e2 ‹e1>0› unfolding e4_def by auto
   moreover have avoid p e4 using e2 ‹e1>0› e1_avoid unfolding e4_def
avoid_def by auto
  moreover have p∈pz ⟶ cont ff p e4
    by (auto simp add: e2 e4_def)
  ultimately show ?thesis by auto
 qed
 then obtain get_e where get_e:∀ p∈s. get_e p>0 ∧ avoid p (get_e p)
    ∧ (p∈pz ⟶ cont ff p (get_e p))
  by metis
 define ci where ci ≡ λp. contour_integral (circlepath p (get_e p)) ff
 define w where w ≡ λp. winding_number g p
 have contour_integral g ff = (∑ p∈pz. w p * ci p) unfolding ci_def w_def
 proof (rule Cauchy_theorem_singularities[OF ‹open s› ‹connected s› finite _
‹valid_path g› loop
      path_img homo])
  have open (s - pz) using open_Diff[OF _ finite_imp_closed[OF finite]] ‹open
s› by auto
  then show ff holomorphic_on s - pz unfolding ff_def using f_holo h_holo
    by (auto intro!: holomorphic_intros simp add:pz_def)
 next
  show ∀ p∈s. 0 < get_e p ∧ (∀ w∈cball p (get_e p). w ∈ s ∧ (w ≠ p ⟶ w ∉
pz))
    using get_e using avoid_def by blast
 qed
 also have ... = (∑ p∈pz. c * w p * h p * zorder f p)
 proof (rule sum.cong[of pz pz,simplified])
  fix p assume p ∈ pz
  show w p * ci p = c * w p * h p * (zorder f p)
  proof (cases p∈s)
    assume p ∈ s
    have ci p = c * h p * (zorder f p) unfolding ci_def
      apply (rule contour_integral_unique)
        using get_e ‹p∈s› ‹p∈pz› unfolding cont_def by (metis mult.assoc
mult.commute)
    thus ?thesis by auto
  next
```

```
    assume p∉s
    then have w p=0 using homo unfolding w_def by auto
    then show ?thesis by auto
   qed
  qed
  also have ... = c*(∑ p∈pz. w p * h p * zorder f p)
   unfolding sum_distrib_left by (simp add:algebra_simps)
  finally have contour_integral g ff = c * (∑ p∈pz. w p * h p * of_int (zorder f
p)) .
  then show ?thesis unfolding ff_def c_def w_def by simp
qed
```

6.3 Coefficient asymptotics for generating functions

For a formal power series that has a meromorphic continuation on some disc in the context plane, we can use the Residue Theorem to extract precise asymptotic information from the residues at the poles. This can be used to derive the asymptotic behaviour of the coefficients ($a_n \sim \ldots$). If the function is meromorphic on the entire complex plane, one can even derive a full asymptotic expansion.

We will first show the relationship between the coefficients and the sum over the residues with an explicit remainder term (the contour integral along the circle used in the Residue theorem).

```
theorem
  fixes f :: complex ⇒ complex and n :: nat and r :: real
  defines g ≡ (λw. f w / w ^ Suc n) and γ ≡ circlepath 0 r
  assumes open A connected A cball 0 r ⊆ A r > 0
  assumes f holomorphic_on A − S S ⊆ ball 0 r finite S 0 ∉ S
  shows   fps_coeff_conv_residues:
          (deriv ^^ n) f 0 / fact n =
            contour_integral γ g / (2 * pi * i) − (∑ z∈S. residue g z) (is ?thesis1)
    and   fps_coeff_residues_bound:
          (⋀z. norm z = r ⟹ z ∉ k ⟹ norm (f z) ≤ C) ⟹ C ≥ 0 ⟹ finite
k ⟹
              norm ((deriv ^^ n) f 0 / fact n + (∑ z∈S. residue g z)) ≤ C / r ^ n
proof −
  have holo: g holomorphic_on A − insert 0 S
   unfolding g_def using assms by (auto intro!: holomorphic_intros)
  have contour_integral γ g = 2 * pi * i * (∑ z∈insert 0 S. winding_number γ
z * residue g z)
  proof (rule Residue_theorem)
   show g holomorphic_on A − insert 0 S by fact
   from assms show ∀ z. z ∉ A ⟶ winding_number γ z = 0
     unfolding γ_def by (intro allI impI winding_number_zero_outside[of _
cball 0 r]) auto
  qed (insert assms, auto simp: γ_def)
  also have winding_number γ z = 1 if z ∈ insert 0 S for z
   unfolding γ_def using assms that by (intro winding_number_circlepath) auto
```

hence $(\sum z{\in}insert\ 0\ S.\ winding\_number\ \gamma\ z * residue\ g\ z) = (\sum z{\in}insert\ 0\ S.\ residue\ g\ z)$
by (*intro sum.cong*) *simp_all*
also have $\ldots = residue\ g\ 0 + (\sum z{\in}S.\ residue\ g\ z)$
using ‹$0 \notin S$› **and** ‹*finite S*› **by** (*subst sum.insert*) *auto*
also from ‹$r > 0$› **have** $0 \in cball\ 0\ r$ **by** *simp*
with *assms* **have** $0 \in A - S$ **by** *blast*
with *assms* **have** $residue\ g\ 0 = (deriv\ \widehat{\ }\widehat{\ }\ n)\ f\ 0\ /\ fact\ n$
unfolding *g_def* **by** (*subst residue_holomorphic_over_power'*[*of* $A - S$])
(*auto simp*: *finite_imp_closed*)
finally show *?thesis1*
by (*simp add*: *field_simps*)

assume *C*: $\bigwedge z.\ norm\ z = r \Longrightarrow z \notin k \Longrightarrow norm\ (f\ z) \leq C\ C \geq 0$ **and** *k*: *finite k*
have $(deriv\ \widehat{\ }\widehat{\ }\ n)\ f\ 0\ /\ fact\ n + (\sum z{\in}S.\ residue\ g\ z) = contour\_integral\ \gamma\ g\ /\ (2 * pi * \mathrm{i})$
using ‹*?thesis1*› **by** (*simp add*: *algebra_simps*)
also have $norm\ \ldots = norm\ (contour\_integral\ \gamma\ g)\ /\ (2 * pi)$
by (*simp add*: *norm_divide norm_mult*)
also have $norm\ (contour\_integral\ \gamma\ g) \leq C\ /\ r\ \widehat{\ }\ Suc\ n * (2 * pi * r)$
proof (*rule has_contour_integral_bound_circlepath_strong*)
from ‹*open A*› **and** ‹*finite S*› **have** *open* $(A - insert\ 0\ S)$
by (*blast intro*: *finite_imp_closed*)
with *assms* **show** (*g has_contour_integral contour_integral* γ *g*) (*circlepath 0 r*)
unfolding γ*_def*
by (*intro has_contour_integral_integral contour_integrable_holomorphic_simple* [*OF holo*]) *auto*
next
fix *z* **assume** *z*: $norm\ (z - 0) = r\ z \notin k$
hence $norm\ (g\ z) = norm\ (f\ z)\ /\ r\ \widehat{\ }\ Suc\ n$
by (*simp add*: *norm_divide g_def norm_mult norm_power*)
also have $\ldots \leq C\ /\ r\ \widehat{\ }\ Suc\ n$
using *k* **and** ‹$r > 0$› **and** *z* **by** (*intro divide_right_mono C zero_le_power*) *auto*
finally show $norm\ (g\ z) \leq C\ /\ r\ \widehat{\ }\ Suc\ n$ **.**
qed (*insert C*(*2*) *k* ‹$r > 0$›, *auto*)
also from ‹$r > 0$› **have** $C\ /\ r\ \widehat{\ }\ Suc\ n * (2 * pi * r)\ /\ (2 * pi) = C\ /\ r\ \widehat{\ }\ n$
by *simp*
finally show $norm\ ((deriv\ \widehat{\ }\widehat{\ }\ n)\ f\ 0\ /\ fact\ n + (\sum z{\in}S.\ residue\ g\ z)) \leq \ldots$
by − (*simp_all add*: *divide_right_mono*)
qed

Since the circle is fixed, we can get an upper bound on the values of the generating function on the circle and therefore show that the integral is $O(r^{-n})$.

corollary *fps_coeff_residues_bigo*:
fixes *f* :: *complex* ⇒ *complex* **and** *r* :: *real*

assumes *open A connected A cball 0 r ⊆ A r > 0*
assumes *f holomorphic_on A − S S ⊆ ball 0 r finite S 0 ∉ S*
assumes *g*: *eventually* (λ*n*. *g n* = −(∑ *z*∈*S*. *residue* (λ*z*. *f z* / *z* ^ *Suc n*) *z*)) *sequentially*
(**is** *eventually* (λ*n*. _ = −*?g*′ *n*) _)
shows (λ*n*. (*deriv* ^^ *n*) *f* 0 / *fact n* − *g n*) ∈ *O*(λ*n*. 1 / *r* ^ *n*) (**is** (λ*n*. *?c n* − _) ∈ *O*(_))
proof −
from *assms* **have** *compact* (*f* ‘ *sphere 0 r*)
by (*intro compact_continuous_image holomorphic_on_imp_continuous_on holomorphic_on_subset*[*OF* ‹*f holomorphic_on A − S*›]) *auto*
hence *bounded* (*f* ‘ *sphere 0 r*) **by** (*rule compact_imp_bounded*)
then obtain *C* **where** *C*: ⋀*z*. *z* ∈ *sphere 0 r* ⟹ *norm* (*f z*) ≤ *C*
by (*auto simp*: *bounded_iff sphere_def*)
have 0 ≤ *norm* (*f* (*of_real r*)) **by** *simp*
also from *C*[*of of_real r*] **and** ‹*r* > 0› **have** ... ≤ *C* **by** *simp*
finally have *C_nonneg*: *C* ≥ 0 **.**

have (λ*n*. *?c n* + *?g*′ *n*) ∈ *O*(λ*n*. *of_real* (1 / *r* ^ *n*))
proof (*intro bigoI*[*of* _ *C*] *always_eventually allI*)
fix *n* :: *nat*
from *assms* **and** *C* **and** *C_nonneg* **have** *norm* (*?c n* + *?g*′ *n*) ≤ *C* / *r* ^ *n*
by (*intro fps_coeff_residues_bound*[**where** *A* = *A* **and** *k* = {}]) *auto*
also have ... = *C* ∗ *norm* (*complex_of_real* (1 / *r* ^ *n*))
using ‹*r* > 0› **by** (*simp add*: *norm_divide norm_power*)
finally show *norm* (*?c n* + *?g*′ *n*) ≤ ... **.**
qed
also have *?this* ⟷ (λ*n*. *?c n* − *g n*) ∈ *O*(λ*n*. *of_real* (1 / *r* ^ *n*))
by (*intro landau_o.big.in_cong eventually_mono*[*OF g*]) *simp_all*
finally show *?thesis* **.**
qed

corollary *fps_coeff_residues_bigo*′:
fixes *f* :: *complex* ⇒ *complex* **and** *r* :: *real*
assumes *exp*: *f has_fps_expansion F*
assumes *open A connected A cball 0 r ⊆ A r > 0*
assumes *f holomorphic_on A − S S ⊆ ball 0 r finite S 0 ∉ S*
assumes *eventually* (λ*n*. *g n* = −(∑ *z*∈*S*. *residue* (λ*z*. *f z* / *z* ^ *Suc n*) *z*)) *sequentially*
(**is** *eventually* (λ*n*. _ = −*?g*′ *n*) _)
shows (λ*n*. *fps_nth F n* − *g n*) ∈ *O*(λ*n*. 1 / *r* ^ *n*) (**is** (λ*n*. *?c n* − _) ∈ *O*(_))
proof −
have *fps_nth F* = (λ*n*. (*deriv* ^^ *n*) *f* 0 / *fact n*)
using *fps_nth_fps_expansion*[*OF exp*] **by** (*intro ext*) *simp_all*
with *fps_coeff_residues_bigo*[*OF assms*(2−)] **show** *?thesis* **by** *simp*
qed

6.4 Rouche's theorem

theorem *Rouche_theorem*:
fixes *f g*::*complex* ⇒ *complex* **and** *s*:: *complex set*
defines *fg*≡(λ*p*. *f p* + *g p*)
defines *zeros_fg*≡{*p*. *fg p* = *0*} **and** *zeros_f*≡{*p*. *f p* = *0*}
assumes
open s **and** *connected s* **and**
finite zeros_fg **and**
finite zeros_f **and**
f_holo:*f holomorphic_on s* **and**
g_holo:*g holomorphic_on s* **and**
valid_path γ **and**
loop:*pathfinish* γ = *pathstart* γ **and**
path_img:*path_image* γ ⊆ *s* **and**
path_less:∀ *z*∈*path_image* γ. *cmod*(*f z*) > *cmod*(*g z*) **and**
homo:∀ *z*. (*z* ∉ *s*) ⟶ *winding_number* γ *z* = *0*
shows (∑ *p*∈*zeros_fg*. *winding_number* γ *p* ∗ *zorder fg p*)
= (∑ *p*∈*zeros_f*. *winding_number* γ *p* ∗ *zorder f p*)
proof −
have *path_fg*:*path_image* γ ⊆ *s* − *zeros_fg*
proof −
have *False* **when** *z*∈*path_image* γ **and** *f z* + *g z*=*0* **for** *z*
proof −
have *cmod* (*f z*) > *cmod* (*g z*) **using** ‹*z*∈*path_image* γ› *path_less* **by** *auto*
moreover have *f z* = − *g z* **using** ‹*f z* + *g z* =*0*› **by** (*simp add*: *eq_neg_iff_add_eq_0*)
then have *cmod* (*f z*) = *cmod* (*g z*) **by** *auto*
ultimately show *False* **by** *auto*
qed
then show *?thesis* **unfolding** *zeros_fg_def fg_def* **using** *path_img* **by** *auto*
qed
have *path_f*:*path_image* γ ⊆ *s* − *zeros_f*
proof −
have *False* **when** *z*∈*path_image* γ **and** *f z* =*0* **for** *z*
proof −
have *cmod* (*g z*) < *cmod* (*f z*) **using** ‹*z*∈*path_image* γ› *path_less* **by** *auto*
then have *cmod* (*g z*) < *0* **using** ‹*f z*=*0*› **by** *auto*
then show *False* **by** *auto*
qed
then show *?thesis* **unfolding** *zeros_f_def* **using** *path_img* **by** *auto*
qed
define *w* **where** *w* ≡ λ*p*. *winding_number* γ *p*
define *c* **where** *c* ≡ *2* ∗ *complex_of_real pi* ∗ i
define *h* **where** *h* ≡ λ*p*. *g p* / *f p* + *1*
obtain *spikes*
where *finite spikes* **and** *spikes*: ∀ *x*∈{*0..1*} − *spikes*. γ *differentiable at x*
using ‹*valid_path* γ›
by (*auto simp*: *valid_path_def piecewise_C1_differentiable_on_def C1_differentiable_on_eq*)
have *h_contour*:((λ*x*. *deriv h x* / *h x*) *has_contour_integral 0*) γ

```
  proof −
    have outside_img:0 ∈ outside (path_image (h o γ))
    proof −
      have h p ∈ ball 1 1 when p∈path_image γ for p
      proof −
        have cmod (g p/f p) <1 using path_less[rule_format,OF that]
          apply (cases cmod (f p) = 0)
          by (auto simp add: norm_divide)
       then show ?thesis unfolding h_def by (auto simp add:dist_complex_def)
      qed
      then have path_image (h o γ) ⊆ ball 1 1
        by (simp add: image_subset_iff path_image_compose)
      moreover have (0::complex) ∉ ball 1 1 by (simp add: dist_norm)
      ultimately show ?thesis
        using convex_in_outside[of ball 1 1 0] outside_mono by blast
    qed
    have valid_h:valid_path (h ∘ γ)
   proof (rule valid_path_compose_holomorphic[OF ‹valid_path γ› _ _ path_f])
      show h holomorphic_on s − zeros_f
        unfolding h_def using f_holo g_holo
        by (auto intro!: holomorphic_intros simp add:zeros_f_def)
    next
      show open (s − zeros_f) using ‹finite zeros_f› ‹open s› finite_imp_closed
        by auto
    qed
    have ((λz. 1/z) has_contour_integral 0) (h ∘ γ)
    proof −
     have 0 ∉ path_image (h ∘ γ) using outside_img by (simp add: outside_def)
      then have ((λz. 1/z) has_contour_integral c ∗ winding_number (h ∘ γ) 0)
(h ∘ γ)
       using has_contour_integral_winding_number[of h o γ 0,simplified] valid_h
        unfolding c_def by auto
      moreover have winding_number (h o γ) 0 = 0
      proof −
        have 0 ∈ outside (path_image (h ∘ γ)) using outside_img .
        moreover have path (h o γ)
          using valid_h by (simp add: valid_path_imp_path)
        moreover have pathfinish (h o γ) = pathstart (h o γ)
          by (simp add: loop pathfinish_compose pathstart_compose)
       ultimately show ?thesis using winding_number_zero_in_outside by auto
      qed
      ultimately show ?thesis by auto
    qed
    moreover have vector_derivative (h ∘ γ) (at x) = vector_derivative γ (at x)
∗ deriv h (γ x)
      when x∈{0..1} − spikes for x
    proof (rule vector_derivative_chain_at_general)
      show γ differentiable at x using that ‹valid_path γ› spikes by auto
    next
```

```
    define der where der ≡ λp. (deriv g p * f p − g p * deriv f p)/(f p * f p)
    define t where t ≡ γ x
    have f t≠0 unfolding zeros_f_def t_def
     by (metis DiffD1 image_eqI norm_not_less_zero norm_zero path_defs(4)
path_less that)
    moreover have t∈s
     using contra_subsetD path_image_def path_fg t_def that by fastforce
    ultimately have (h has_field_derivative der t) (at t)
     unfolding h_def der_def using g_holo f_holo ‹open s›
     by (auto intro!: holomorphic_derivI derivative_eq_intros)
    then show h field_differentiable at (γ x)
     unfolding t_def field_differentiable_def by blast
   qed
   then have ((/) 1 has_contour_integral 0) (h ∘ γ)
              = ((λx. deriv h x / h x) has_contour_integral 0) γ
    unfolding has_contour_integral
    apply (intro has_integral_spike_eq[OF negligible_finite, OF ‹finite spikes›])
    by auto
   ultimately show ?thesis by auto
  qed
  then have contour_integral γ (λx. deriv h x / h x) = 0
   using contour_integral_unique by simp
  moreover have contour_integral γ (λx. deriv fg x / fg x) = contour_integral γ
(λx. deriv f x / f x)
     + contour_integral γ (λp. deriv h p / h p)
  proof −
   have (λp. deriv f p / f p) contour_integrable_on γ
    proof (rule contour_integrable_holomorphic_simple[OF _ _ ‹valid_path γ›
path_f])
     show open (s − zeros_f) using finite_imp_closed[OF ‹finite zeros_f›] ‹open
s›
      by auto
     then show (λp. deriv f p / f p) holomorphic_on s − zeros_f
      using f_holo
      by (auto intro!: holomorphic_intros simp add:zeros_f_def)
   qed
   moreover have (λp. deriv h p / h p) contour_integrable_on γ
    using h_contour
    by (simp add: has_contour_integral_integrable)
   ultimately have contour_integral γ (λx. deriv f x / f x + deriv h x / h x) =
                    contour_integral γ (λp. deriv f p / f p) + contour_integral γ
(λp. deriv h p / h p)
    using contour_integral_add[of (λp. deriv f p / f p) γ (λp. deriv h p / h p) ]
    by auto
   moreover have deriv fg p / fg p =  deriv f p / f p + deriv h p / h p
                when p∈ path_image γ for p
   proof −
    have fg p≠0 and f p≠0 using path_f path_fg that unfolding zeros_f_def
zeros_fg_def
```

```
      by auto
    have h p≠0
    proof (rule ccontr)
      assume ¬ h p ≠ 0
     then have g p / f p= −1 unfolding h_def by (simp add: add_eq_0_iff2)
      then have cmod (g p/f p) = 1 by auto
      moreover have cmod (g p/f p) <1 using path_less[rule_format,OF that]
        apply (cases cmod (f p) = 0)
        by (auto simp add: norm_divide)
      ultimately show False by auto
    qed
    have der_fg:deriv fg p =  deriv f p + deriv g p unfolding fg_def
      using f_holo g_holo holomorphic_on_imp_differentiable_at[OF _  ‹open
s›] path_img that
      by auto
    have der_h:deriv h p = (deriv g p * f p − g p * deriv f p)/(f p * f p)
    proof −
      define der where der ≡ λp. (deriv g p * f p − g p * deriv f p)/(f p * f p)
      have p∈s using path_img that by auto
      then have (h has_field_derivative der p) (at p)
        unfolding h_def der_def using g_holo f_holo ‹open s› ‹f p≠0›
        by (auto intro!: derivative_eq_intros holomorphic_derivI)
      then show ?thesis unfolding der_def using DERIV_imp_deriv by auto
    qed
    show ?thesis
      apply (simp only:der_fg der_h)
      apply (auto simp add:field_simps ‹h p≠0› ‹f p≠0› ‹fg p≠0›)
      by (auto simp add:field_simps h_def ‹f p≠0› fg_def)
  qed
  then have contour_integral γ (λp. deriv fg p / fg p)
              = contour_integral γ (λp. deriv f p / f p + deriv h p / h p)
    by (elim contour_integral_eq)
  ultimately show ?thesis by auto
 qed
 moreover have contour_integral γ (λx. deriv fg x / fg x) = c * (∑ p∈zeros_fg.
w p * zorder fg p)
  unfolding c_def zeros_fg_def w_def
 proof (rule argument_principle[OF ‹open s› ‹connected s› _ _ ‹valid_path γ›
loop _ homo
      , of _ {} λ_. 1,simplified])
   show fg holomorphic_on s unfolding fg_def using f_holo g_holo holomor-
phic_on_add by auto
  show path_image γ ⊆ s − {p. fg p = 0} using path_fg unfolding zeros_fg_def
.
   show  finite {p. fg p = 0} using ‹finite zeros_fg› unfolding zeros_fg_def .
 qed
 moreover have contour_integral γ (λx. deriv f x / f x) = c * (∑ p∈zeros_f. w
p * zorder f p)
  unfolding c_def zeros_f_def w_def
```

proof (*rule argument_principle*[*OF* ‹*open s*› ‹*connected s*› _ _ ‹*valid_path γ*› *loop* _ *homo*
, *of* _ {} λ_. *1*,*simplified*])
show *f holomorphic_on s* **using** *f_holo g_holo holomorphic_on_add* **by** *auto*
show *path_image γ* ⊆ *s* − {*p*. *f p* = *0*} **using** *path_f* **unfolding** *zeros_f_def*
.
show *finite* {*p*. *f p* = *0*} **using** ‹*finite zeros_f*› **unfolding** *zeros_f_def* .
qed
ultimately have *c** (∑ *p*∈*zeros_fg*. *w p* * (*zorder fg p*)) = *c** (∑ *p*∈*zeros_f*. *w p* * (*zorder f p*))
by *auto*
then show *?thesis* **unfolding** *c_def* **using** *w_def* **by** *auto*
qed

end

7 The Great Picard Theorem and its Applications

Ported from HOL Light (cauchy.ml) by L C Paulson, 2017

theory *Great_Picard*
imports *Conformal_Mappings*
begin

7.1 Schottky's theorem

lemma *Schottky_lemma0*:
assumes *holf*: *f holomorphic_on S* **and** *cons*: *contractible S* **and** *a* ∈ *S*
and *f*: ⋀*z*. *z* ∈ *S* ⟹ *f z* ≠ *1* ∧ *f z* ≠ −*1*
obtains *g* **where** *g holomorphic_on S*
norm(*g a*) ≤ *1* + *norm*(*f a*) / *3*
⋀*z*. *z* ∈ *S* ⟹ *f z* = *cos*(*of_real pi* * *g z*)
proof −
obtain *g* **where** *holg*: *g holomorphic_on S* **and** *g*: *norm*(*g a*) ≤ *pi* + *norm*(*f a*)
and *f_eq_cos*: ⋀*z*. *z* ∈ *S* ⟹ *f z* = *cos*(*g z*)
using *contractible_imp_holomorphic_arccos_bounded* [*OF assms*]
by *blast*
show *?thesis*
proof
show (λ*z*. *g z* / *pi*) *holomorphic_on S*
by (*auto intro*: *holomorphic_intros holg*)
have *3* ≤ *pi*
using *pi_approx* **by** *force*
have *3* * *norm*(*g a*) ≤ *3* * (*pi* + *norm*(*f a*))
using *g* **by** *auto*
also have ... ≤ *pi* * *3* + *pi* * *cmod* (*f a*)
using ‹*3* ≤ *pi*› **by** (*simp add*: *mult_right_mono algebra_simps*)
finally show *cmod* (*g a* / *complex_of_real pi*) ≤ *1* + *cmod* (*f a*) / *3*
by (*simp add*: *field_simps norm_divide*)

show $\bigwedge z.\ z \in S \Longrightarrow f\ z = cos\ (complex\_of\_real\ pi * (g\ z\ /\ complex\_of\_real\ pi))$
by (*simp add*: *f_eq_cos*)
qed
qed

lemma *Schottky_lemma1*:
fixes *n*::*nat*
assumes $0 < n$
shows $0 < n + sqrt(real\ n\ \hat{}\ 2 - 1)$
proof −
have $0 < n * n$
by (*simp add*: *assms*)
then show *?thesis*
by (*metis add.commute add.right_neutral add_pos_nonneg assms diff_ge_0_iff_ge nat_less_real_le of_nat_0 of_nat_0_less_iff of_nat_power power2_eq_square real_sqrt_ge_0_iff*)
qed

lemma *Schottky_lemma2*:
fixes *x*::*real*
assumes $0 \le x$
obtains *n* **where** $0 < n\ |x - ln\ (real\ n + sqrt\ ((real\ n)^2 - 1))\ /\ pi| < 1/2$
proof −
obtain *n0*::*nat* **where** $0 < n0\ ln(n0 + sqrt(real\ n0\ \hat{}\ 2 - 1))\ /\ pi \le x$
proof
show $ln(real\ 1 + sqrt(real\ 1\ \hat{}\ 2 - 1))/pi \le x$
by (*auto simp*: *assms*)
qed *auto*
moreover
obtain *M*::*nat* **where** $\bigwedge n.\ [\![0 < n;\ ln(n + sqrt(real\ n\ \hat{}\ 2 - 1))\ /\ pi \le x]\!] \Longrightarrow n \le M$
proof
fix *n*::*nat*
assume $0 < n\ ln\ (n + sqrt\ ((real\ n)^2 - 1))\ /\ pi \le x$
then have $ln\ (n + sqrt\ ((real\ n)^2 - 1)) \le x * pi$
by (*simp add*: *field_split_simps*)
then have $*$: $exp\ (ln\ (n + sqrt\ ((real\ n)^2 - 1))) \le exp\ (x * pi)$
by *blast*
have *0*: $0 \le sqrt\ ((real\ n)^2 - 1)$
using ‹$0 < n$› **by** *auto*
have $n + sqrt\ ((real\ n)^2 - 1) = exp\ (ln\ (n + sqrt\ ((real\ n)^2 - 1)))$
by (*simp add*: *Suc_leI* ‹$0 < n$› *add_pos_nonneg real_of_nat_ge_one_iff*)
also have $... \le exp\ (x * pi)$
using $*$ **by** *blast*
finally have $real\ n \le exp\ (x * pi)$
using *0* **by** *linarith*

```
    then show n ≤ nat (ceiling (exp(x * pi)))
      by linarith
  qed
  ultimately obtain n where
     0 < n and le_x: ln(n + sqrt(real n ^ 2 - 1)) / pi ≤ x
           and le_n: ⋀k. ⟦0 < k; ln(k + sqrt(real k ^ 2 - 1)) / pi ≤ x⟧ ⟹ k ≤
n
    using bounded_Max_nat [of λn. 0<n ∧ ln (n + sqrt ((real n)^2 - 1)) / pi ≤
x] by metis
  define a where a ≡ ln(n + sqrt(real n ^ 2 - 1)) / pi
  define b where b ≡ ln (1 + real n + sqrt ((1 + real n)^2 - 1)) / pi
  have le_xa: a ≤ x
   and le_na: ⋀k. ⟦0 < k; ln(k + sqrt(real k ^ 2 - 1)) / pi ≤ x⟧ ⟹ k ≤ n
    using le_x le_n by (auto simp: a_def)
  moreover have x < b
    using le_n [of Suc n] by (force simp: b_def)
  moreover have b - a < 1
  proof -
    have ln (1 + real n + sqrt ((1 + real n)^2 - 1)) - ln (real n + sqrt ((real n)^2
- 1)) =
          ln ((1 + real n + sqrt ((1 + real n)^2 - 1)) / (real n + sqrt ((real n)^2 -
1)))
      by (simp add: ‹0 < n› Schottky_lemma1 add_pos_nonneg ln_div [symmetric])
    also have ... ≤ 3
    proof (cases n = 1)
      case True
      have sqrt 3 ≤ 2
        by (simp add: real_le_lsqrt)
      then have (2 + sqrt 3) ≤ 4
        by simp
      also have ... ≤ exp 3
        using exp_ge_add_one_self [of 3::real] by simp
      finally have ln (2 + sqrt 3) ≤ 3
          by (metis add_nonneg_nonneg add_pos_nonneg dbl_def dbl_inc_def
dbl_inc_simps(3)
           dbl_simps(3) exp_gt_zero ln_exp ln_le_cancel_iff real_sqrt_ge_0_iff
zero_le_one zero_less_one)
      then show ?thesis
        by (simp add: True)
    next
      case False with ‹0 < n› have 1 < n 2 ≤ n
        by linarith+
      then have 1: 1 ≤ real n * real n
      by (metis less_imp_le_nat one_le_power power2_eq_square real_of_nat_ge_one_iff)
      have *: 4 + (m+2) * 2 ≤ (m+2) * ((m+2) * 3) for m::nat
        by simp
      have 4 + n * 2 ≤ n * (n * 3)
        using * [of n-2]  ‹2 ≤ n›
        by (metis le_add_diff_inverse2)
```

```
    then have **: 4 + real n * 2 ≤ real n * (real n * 3)
    by (metis (mono_tags, opaque_lifting) of_nat_le_iff of_nat_add of_nat_mult
of_nat_numeral)
    have sqrt ((1 + real n)^2 - 1) ≤ 2 * sqrt ((real n)^2 - 1)
      by (auto simp: real_le_lsqrt power2_eq_square algebra_simps 1 **)
    then
    have ((1 + real n + sqrt ((1 + real n)^2 - 1)) / (real n + sqrt ((real n)^2 -
1))) ≤ 2
      using Schottky_lemma1 ‹0 < n› by (simp add: field_split_simps)
   then have ln ((1 + real n + sqrt ((1 + real n)^2 - 1)) / (real n + sqrt ((real
n)^2 - 1))) ≤ ln 2
      using Schottky_lemma1 [of n] ‹0 < n›
      by (simp add: field_split_simps add_pos_nonneg)
    also have ... ≤ 3
      using ln_add_one_self_le_self [of 1] by auto
    finally show ?thesis .
  qed
  also have ... < pi
    using pi_approx by simp
  finally show ?thesis
    by (simp add: a_def b_def field_split_simps)
 qed
 ultimately have |x - a| < 1/2 ∨ |x - b| < 1/2
  by (auto simp: abs_if)
 then show thesis
 proof
  assume |x - a| < 1/2
  then show ?thesis
    by (rule_tac n=n in that) (auto simp: a_def ‹0 < n›)
 next
  assume |x - b| < 1/2
  then show ?thesis
    by (rule_tac n=Suc n in that) (auto simp: b_def ‹0 < n›)
 qed
qed


lemma Schottky_lemma3:
 fixes z::complex
 assumes z ∈ (⋃m ∈ Ints. ⋃n ∈ {0<..}. {Complex m (ln(n + sqrt(real n ^ 2
- 1)) / pi)})
          ∪ (⋃m ∈ Ints. ⋃n ∈ {0<..}. {Complex m (-ln(n + sqrt(real n ^ 2 -
1)) / pi)})
 shows cos(pi * cos(pi * z)) = 1 ∨ cos(pi * cos(pi * z)) = -1
proof -
 have sqrt2 [simp]: complex_of_real (sqrt x) * complex_of_real (sqrt x) = x if x
≥ 0 for x::real
   by (metis abs_of_nonneg of_real_mult real_sqrt_mult_self that)
 define plusi where plusi (e::complex) ≡ e + inverse e for e
```

have *1*: ∃ *k*. *plusi* (*exp* (i ∗ (*of_int m* ∗ *complex_of_real pi*) − *ln* (*real n* + *sqrt* ((*real n*)^2 − *1*)))) = *of_int k* ∗ *2*
(**is** ∃ *k*. *?Φ k*)
if *n* > *0* **for** *m n*
proof −
have *eeq*: *e* ≠ *0* ⟹ *plusi e* = *n* ⟷ (*inverse e*) ^ *2* = *n*/*e* − *1* **for** *n e*::*complex*
by (*auto simp*: *plusi_def field_simps power2_eq_square*)
have [*simp*]: *1* ≤ *real n* ∗ *real n*
using *nat_0_less_mult_iff nat_less_real_le that* **by** *force*
consider *odd m* | *even m*
by *blast*
then have ∃ *k*. *?Φ k*
proof *cases*
case *1*
then have *?Φ* (− *n*)
using *Schottky_lemma1* [*OF that*]
by (*simp add*: *eeq*) (*simp add*: *power2_eq_square exp_diff exp_Euler exp_of_real algebra_simps sin_int_times_real cos_int_times_real*)
then show *?thesis* **..**
next
case *2*
then have *?Φ n*
using *Schottky_lemma1* [*OF that*]
by (*simp add*: *eeq*) (*simp add*: *power2_eq_square exp_diff exp_Euler exp_of_real algebra_simps*)
then show *?thesis* **..**
qed
then show *?thesis* **by** *blast*
qed
have *2*: ∃ *k*. *plusi* (*exp* (i ∗ (*of_int m* ∗ *complex_of_real pi*) +
(*ln* (*real n* + *sqrt* ((*real n*)^2 − *1*))))) = *of_int k* ∗ *2*
(**is** ∃ *k*. *?Φ k*)
if *n* > *0* **for** *m n*
proof −
have *eeq*: *e* ≠ *0* ⟹ *plusi e* = *n* ⟷ *e*^*2* − *n*∗*e* + *1* = *0* **for** *n e*::*complex*
by (*auto simp*: *plusi_def field_simps power2_eq_square*)
have [*simp*]: *1* ≤ *real n* ∗ *real n*
by (*metis One_nat_def add.commute nat_less_real_le of_nat_1 of_nat_Suc one_le_power power2_eq_square that*)
consider *odd m* | *even m*
by *blast*
then have ∃ *k*. *?Φ k*
proof *cases*
case *1*
then have *?Φ* (− *n*)
using *Schottky_lemma1* [*OF that*]
by (*simp add*: *eeq*) (*simp add*: *power2_eq_square exp_add exp_Euler exp_of_real algebra_simps sin_int_times_real cos_int_times_real*)
then show *?thesis* **..**

```
    next
      case 2
      then have ?Φ n
        using Schottky_lemma1 [OF that]
          by (simp add: eeq) (simp add: power2_eq_square exp_add exp_Euler
exp_of_real algebra_simps)
      then show ?thesis ..
    qed
    then show ?thesis by blast
  qed
  have ∃x. cos (complex_of_real pi * z) = of_int x
    using assms
     apply (auto simp: Ints_def cos_exp_eq exp_minus Complex_eq simp flip:
plusi_def)
     apply (auto simp: algebra_simps dest: 1 2)
    done
  then have sin(pi * cos(pi * z)) ^ 2 = 0
    by (simp add: Complex_Transcendental.sin_eq_0)
  then have 1 - cos(pi * cos(pi * z)) ^ 2 = 0
    by (simp add: sin_squared_eq)
  then show ?thesis
    using power2_eq_1_iff by auto
qed


theorem Schottky:
  assumes holf: f holomorphic_on cball 0 1
      and nof0: norm(f 0) ≤ r
      and not01: ⋀z. z ∈ cball 0 1 ⟹ ¬(f z = 0 ∨ f z = 1)
      and 0 < t t < 1 norm z ≤ t
    shows norm(f z) ≤ exp(pi * exp(pi * (2 + 2 * r + 12 * t / (1 - t))))
proof -
  obtain h where holf: h holomorphic_on cball 0 1
            and nh0: norm (h 0) ≤ 1 + norm(2 * f 0 - 1) / 3
            and h:   ⋀z. z ∈ cball 0 1 ⟹ 2 * f z - 1 = cos(of_real pi * h z)
  proof (rule Schottky_lemma0 [of λz. 2 * f z - 1 cball 0 1 0])
    show (λz. 2 * f z - 1) holomorphic_on cball 0 1
      by (intro holomorphic_intros holf)
    show contractible (cball (0::complex) 1)
      by (auto simp: convex_imp_contractible)
    show ⋀z. z ∈ cball 0 1 ⟹ 2 * f z - 1 ≠ 1 ∧ 2 * f z - 1 ≠ - 1
      using not01 by force
  qed auto
  obtain g where holg: g holomorphic_on cball 0 1
            and ng0:  norm(g 0) ≤ 1 + norm(h 0) / 3
            and g:    ⋀z. z ∈ cball 0 1 ⟹ h z = cos(of_real pi * g z)
  proof (rule Schottky_lemma0 [OF holf convex_imp_contractible, of 0])
    show ⋀z. z ∈ cball 0 1 ⟹ h z ≠ 1 ∧ h z ≠ - 1
      using h not01 by fastforce+
```

```
  qed auto
  have g0_2_f0: norm(g 0) ≤ 2 + norm(f 0)
  proof −
    have cmod (2 * f 0 − 1) ≤ cmod (2 * f 0) + 1
      by (metis norm_one norm_triangle_ineq4)
    also have ... ≤ 6 + 9 * cmod (f 0)
      by auto
    finally have 1 + norm(2 * f 0 − 1) / 3 ≤ (2 + norm(f 0) − 1) * 3
      by (simp add: divide_simps)
    with nh0 have norm(h 0) ≤ (2 + norm(f 0) − 1) * 3
      by linarith
    then have 1 + norm(h 0) / 3 ≤ 2 + norm(f 0)
      by simp
    with ng0 show ?thesis
      by auto
  qed
  have z ∈ ball 0 1
    using assms by auto
  have norm_g_12: norm(g z − g 0) ≤ (12 * t) / (1 − t)
  proof −
    obtain g' where g': ⋀x. x ∈ cball 0 1 ⟹ (g has_field_derivative g' x) (at x
within cball 0 1)
      using holg [unfolded holomorphic_on_def field_differentiable_def] by metis
    have int_g': (g' has_contour_integral g z − g 0) (linepath 0 z)
      using contour_integral_primitive [OF g' valid_path_linepath, of 0 z]
      using ‹z ∈ ball 0 1› segment_bound1 by fastforce
    have cmod (g' w) ≤ 12 / (1 − t) if w ∈ closed_segment 0 z for w
    proof −
      have w: w ∈ ball 0 1
        using segment_bound [OF that] ‹z ∈ ball 0 1› by simp
      have *: ⟦⋀b. (∃w ∈ T ∪ U. w ∈ ball b 1); ⋀x. x ∈ D ⟹ g x ∉ T ∪ U⟧
⟹ ∄b. ball b 1 ⊆ g ‘ D for T U D
        by force
      have ttt: 1 − t ≤ dist w u if cmod u = 1 for u
          using ‹norm z ≤ t› segment_bound1 [OF ‹w ∈ closed_segment 0 z›]
norm_triangle_ineq2 [of u w] that
        by (simp add: dist_norm norm_minus_commute)
      have ∄b. ball b 1 ⊆ g ‘ cball 0 1
      proof (rule *)
        show (∃w ∈ (⋃m ∈ Ints. ⋃n ∈ {0<..}. {Complex m (ln(n + sqrt(real n
^ 2 − 1)) / pi)}) ∪
                  (⋃m ∈ Ints. ⋃n ∈ {0<..}. {Complex m (−ln(n + sqrt(real n ^ 2
− 1)) / pi)}). w ∈ ball b 1) for b
        proof −
          obtain m where m: m ∈ ℤ |Re b − m| ≤ 1/2
            by (metis Ints_of_int abs_minus_commute of_int_round_abs_le)
          show ?thesis
          proof (cases 0::real Im b rule: le_cases)
            case le
```

```
        then obtain n where 0 < n and n: |Im b − ln (n + sqrt ((real n)^2 −
1)) / pi| < 1/2
          using Schottky_lemma2 [of Im b] by blast
        have dist b (Complex m (Im b)) ≤ 1/2
          by (metis cancel_comm_monoid_add_class.diff_cancel cmod_eq_Re
complex.sel(1) complex.sel(2) dist_norm m(2) minus_complex.code)
        moreover
         have dist (Complex m (Im b)) (Complex m (ln (n + sqrt ((real n)^2 −
1)) / pi)) < 1/2
            using n by (simp add: complex_norm cmod_eq_Re complex_diff
dist_norm del: Complex_eq)
        ultimately have dist b (Complex m (ln (real n + sqrt ((real n)^2 − 1))
/ pi)) < 1
         by (simp add: dist_triangle_lt [of b Complex m (Im b)] dist_commute)
        with le m ‹0 < n› show ?thesis
          apply (rule_tac x = Complex m (ln (real n + sqrt ((real n)^2 − 1)) /
pi) in bexI)
           by (force simp del: Complex_eq greaterThan_0)+
      next
        case ge
        then obtain n where 0 < n and n: |− Im b − ln (real n + sqrt ((real
n)^2 − 1)) / pi| < 1/2
          using Schottky_lemma2 [of − Im b] by auto
        have dist b (Complex m (Im b)) ≤ 1/2
          by (metis cancel_comm_monoid_add_class.diff_cancel cmod_eq_Re
complex.sel(1) complex.sel(2) dist_norm m(2) minus_complex.code)
        moreover
        have dist (Complex m (− ln (n + sqrt ((real n)^2 − 1)) / pi)) (Complex
m (Im b))
            = | − Im b − ln (real n + sqrt ((real n)^2 − 1)) / pi|
          by (simp add: complex_norm dist_norm cmod_eq_Re complex_diff)
         ultimately have dist b (Complex m (− ln (real n + sqrt ((real n)^2 −
1)) / pi)) < 1
          using n by (simp add: dist_triangle_lt [of b Complex m (Im b)]
dist_commute)
        with ge m ‹0 < n› show ?thesis
          by (rule_tac x = Complex m (− ln (real n + sqrt ((real n)^2 − 1)) /
pi) in bexI) auto
      qed
    qed
    show g v ∉ (⋃m ∈ Ints. ⋃n ∈ {0<..}. {Complex m (ln(n + sqrt(real n ^
2 − 1)) / pi)}) ∪
              (⋃m ∈ Ints. ⋃n ∈ {0<..}. {Complex m (−ln(n + sqrt(real n ^ 2
− 1)) / pi)})
        if v ∈ cball 0 1 for v
      using not01 [OF that]
        by (force simp: g [OF that, symmetric] h [OF that, symmetric] dest!:
Schottky_lemma3 [of g v])
  qed
```

```
      then have 12: (1 − t) ∗ cmod (deriv g w) / 12 < 1
        using Bloch_general [OF holg _ ttt, of 1] w by force
      have g field_differentiable at w within cball 0 1
        using holg w by (simp add: holomorphic_on_def)
      then have g field_differentiable at w within ball 0 1
        using ball_subset_cball field_differentiable_within_subset by blast
      with w have der_gw: (g has_field_derivative deriv g w) (at w)
      by (simp add: field_differentiable_within_open [of _ ball 0 1] field_differentiable_derivI)
      with DERIV_unique [OF der_gw] g' w have deriv g w = g' w
      by (metis open_ball at_within_open ball_subset_cball has_field_derivative_subset
subsetCE)
      then show cmod (g' w) ≤ 12 / (1 − t)
        using g' 12 ‹t < 1› by (simp add: field_simps)
    qed
    then have cmod (g z − g 0) ≤ 12 / (1 − t) ∗ cmod z
     using has_contour_integral_bound_linepath [OF int_g', of 12/(1 − t)] assms
      by simp
    with ‹cmod z ≤ t› ‹t < 1› show ?thesis
      by (simp add: field_split_simps)
  qed
  have fz: f z = (1 + cos(of_real pi ∗ h z)) / 2
    using h ‹z ∈ ball 0 1› by (auto simp: field_simps)
  have cmod (f z) ≤ exp (cmod (complex_of_real pi ∗ h z))
    by (simp add: fz mult.commute norm_cos_plus1_le)
  also have ... ≤ exp (pi ∗ exp (pi ∗ (2 + 2 ∗ r + 12 ∗ t / (1 − t))))
  proof (simp add: norm_mult)
    have cmod (g z − g 0) ≤ 12 ∗ t / (1 − t)
      using norm_g_12 ‹t < 1› by (simp add: norm_mult)
    then have cmod (g z) − cmod (g 0) ≤ 12 ∗ t / (1 − t)
      using norm_triangle_ineq2 order_trans by blast
    then have ∗: cmod (g z) ≤ 2 + 2 ∗ r + 12 ∗ t / (1 − t)
      using g0_2_f0 norm_ge_zero [of f 0] nof0
        by linarith
    have cmod (h z) ≤ exp (cmod (complex_of_real pi ∗ g z))
      using ‹z ∈ ball 0 1› by (simp add: g norm_cos_le)
    also have ... ≤ exp (pi ∗ (2 + 2 ∗ r + 12 ∗ t / (1 − t)))
      using ‹t < 1› nof0 ∗ by (simp add: norm_mult)
    finally show cmod (h z) ≤ exp (pi ∗ (2 + 2 ∗ r + 12 ∗ t / (1 − t))) .
  qed
  finally show ?thesis .
qed
```

7.2 The Little Picard Theorem

```
theorem Landau_Picard:
  obtains R
    where ⋀z. 0 < R z
          ⋀f. ⟦f holomorphic_on cball 0 (R(f 0));
                ⋀z. norm z ≤ R(f 0) ⟹ f z ≠ 0 ∧ f z ≠ 1⟧ ⟹ norm(deriv f 0)
```

```
< 1
proof −
  define R where R ≡ λz. 3 * exp(pi * exp(pi*(2 + 2 * cmod z + 12)))
  show ?thesis
  proof
    show Rpos: ⋀z. 0 < R z
      by (auto simp: R_def)
    show norm(deriv f 0) < 1
        if holf: f holomorphic_on cball 0 (R(f 0))
        and Rf:  ⋀z. norm z ≤ R(f 0) ⟹ f z ≠ 0 ∧ f z ≠ 1 for f
    proof −
      let ?r = R(f 0)
      define g where g ≡ f ∘ (λz. of_real ?r * z)
      have 0 < ?r
        using Rpos by blast
      have holg: g holomorphic_on cball 0 1
        unfolding g_def
     proof (intro holomorphic_intros holomorphic_on_compose holomorphic_on_subset
[OF holf])
        show (*) (complex_of_real (R (f 0))) ` cball 0 1 ⊆ cball 0 (R (f 0))
          using Rpos by (auto simp: less_imp_le norm_mult)
      qed
      have *: norm(g z) ≤ exp(pi * exp(pi * (2 + 2 * norm (f 0) + 12 * t / (1
− t))))
          if 0 < t t < 1 norm z ≤ t for t z
      proof (rule Schottky [OF holg])
        show cmod (g 0) ≤ cmod (f 0)
          by (simp add: g_def)
        show ⋀z. z ∈ cball 0 1 ⟹ ¬ (g z = 0 ∨ g z = 1)
          using Rpos by (simp add: g_def less_imp_le norm_mult Rf)
      qed (auto simp: that)
      have C1: g holomorphic_on ball 0 (1/2)
        by (rule holomorphic_on_subset [OF holg]) auto
      have C2: continuous_on (cball 0 (1/2)) g
      by (meson cball_divide_subset_numeral holg holomorphic_on_imp_continuous_on
holomorphic_on_subset)
      have C3: cmod (g z) ≤ R (f 0) / 3 if cmod (0 − z) = 1/2 for z
      proof −
        have norm(g z) ≤ exp(pi * exp(pi * (2 + 2 * norm (f 0) + 12)))
          using * [of 1/2] that by simp
        also have ... = ?r / 3
          by (simp add: R_def)
        finally show ?thesis .
      qed
      then have cmod_g'_le: cmod (deriv g 0) * 3 ≤ R (f 0) * 2
        using Cauchy_inequality [OF C1 C2 _ C3, of 1] by simp
      have holf': f holomorphic_on ball 0 (R(f 0))
        by (rule holomorphic_on_subset [OF holf]) auto
      then have fd0: f field_differentiable at 0
```

```
      by (rule holomorphic_on_imp_differentiable_at [OF _ open_ball])
        (auto simp: Rpos [of f 0])
    have g_eq: deriv g 0 = of_real ?r * deriv f 0
      unfolding g_def
    by (metis DERIV_imp_deriv DERIV_chain DERIV_cmult_Id fd0 field_differentiable_derivI
mult.commute mult_zero_right)
    show ?thesis
      using cmod_g'_le Rpos [of f 0]  by (simp add: g_eq norm_mult)
  qed
 qed
qed

lemma little_Picard_01:
 assumes holf: f holomorphic_on UNIV and f01: ⋀z. f z ≠ 0 ∧ f z ≠ 1
 obtains c where f = (λx. c)
proof −
 obtain R
   where Rpos: ⋀z. 0 < R z
     and R:    ⋀h. [[h holomorphic_on cball 0 (R(h 0));
                     ⋀z. norm z ≤ R(h 0) ⟹ h z ≠ 0 ∧ h z ≠ 1]] ⟹ norm(deriv
h 0) < 1
   using Landau_Picard by metis
 have contf: continuous_on UNIV f
   by (simp add: holf holomorphic_on_imp_continuous_on)
 show ?thesis
 proof (cases ∀x. deriv f x = 0)
   case True
   have (f has_field_derivative 0) (at x) for x
     by (metis True UNIV_I holf holomorphic_derivI open_UNIV)
   then obtain c where ⋀x. f(x) = c
    by (meson UNIV_I DERIV_zero_connected_constant [OF connected_UNIV
open_UNIV finite.emptyI contf])
   then show ?thesis
     using that by auto
 next
   case False
   then obtain w where w: deriv f w ≠ 0 by auto
   define fw where fw ≡ (f ∘ (λz. w + z / deriv f w))
   have norm_let1: norm(deriv fw 0) < 1
   proof (rule R)
     show fw holomorphic_on cball 0 (R (fw 0))
       unfolding fw_def
     by (intro holomorphic_intros holomorphic_on_compose w holomorphic_on_subset
[OF holf] subset_UNIV)
     show fw z ≠ 0 ∧ fw z ≠ 1 if cmod z ≤ R (fw 0) for z
       using f01 by (simp add: fw_def)
   qed
   have (fw has_field_derivative deriv f w * inverse (deriv f w)) (at 0)
     unfolding fw_def
```

```
    apply (intro DERIV_chain derivative_eq_intros w)+
    using holf holomorphic_derivI by (force simp: field_simps)+
  then show ?thesis
    using norm_let1 w by (simp add: DERIV_imp_deriv)
 qed
qed


theorem little_Picard:
 assumes holf: f holomorphic_on UNIV
     and a ≠ b range f ∩ {a,b} = {}
   obtains c where f = (λx. c)
proof −
 let ?g = λx. 1/(b − a)∗(f x − b) + 1
 obtain c where ?g = (λx. c)
 proof (rule little_Picard_01)
   show ?g holomorphic_on UNIV
     by (intro holomorphic_intros holf)
   show ⋀z. ?g z ≠ 0 ∧ ?g z ≠ 1
     using assms by (auto simp: field_simps)
 qed auto
 then have ?g x = c for x
   by meson
 then have f x = c ∗ (b−a) + a for x
   using assms by (auto simp: field_simps)
 then show ?thesis
   using that by blast
qed
```

A couple of little applications of Little Picard

```
lemma holomorphic_periodic_fixpoint:
 assumes holf: f holomorphic_on UNIV
     and p ≠ 0 and per: ⋀z. f(z + p) = f z
 obtains x where f x = x
proof −
 have False if non: ⋀x. f x ≠ x
 proof −
   obtain c where (λz. f z − z) = (λz. c)
   proof (rule little_Picard)
     show (λz. f z − z) holomorphic_on UNIV
       by (simp add: holf holomorphic_on_diff)
     show range (λz. f z − z) ∩ {p,0} = {}
       using assms non by auto (metis add.commute diff_eq_eq)
     qed (auto simp: assms)
   with per show False
     by (metis add.commute add_cancel_left_left ‹p ≠ 0› diff_add_cancel)
 qed
 then show ?thesis
   using that by blast
```

qed

lemma *holomorphic_involution_point*:
assumes *holfU*: *f holomorphic_on UNIV* **and** *non*: $\bigwedge a.\ f \neq (\lambda x.\ a + x)$
obtains *x* **where** $f(f\ x) = x$
proof −
{ assume *non_ff* [*simp*]: $\bigwedge x.\ f(f\ x) \neq x$
then have *non_fp* [*simp*]: $f\ z \neq z$ **for** *z*
by *metis*
have *holf*: *f holomorphic_on X* **for** *X*
using *assms holomorphic_on_subset* **by** *blast*
obtain *c* **where** *c*: $(\lambda x.\ (f(f\ x) - x)/(f\ x - x)) = (\lambda x.\ c)$
proof (*rule little_Picard_01*)
show $(\lambda x.\ (f(f\ x) - x)/(f\ x - x))$ *holomorphic_on UNIV*
using *non_fp*
by (*intro holomorphic_intros holf holomorphic_on_compose* [*unfolded o_def*, *OF holf*]) *auto*
qed *auto*
then obtain $c \neq 0\ c \neq 1$
by (*metis* (*no_types*, *opaque_lifting*) *non_ff diff_zero divide_eq_0_iff right_inverse_eq*)
have *eq*: $f(f\ x) - c * f\ x = x*(1 - c)$ **for** *x*
using *fun_cong* [*OF c*, *of x*] **by** (*simp add*: *field_simps*)
have *df_times_dff*: $deriv\ f\ z * (deriv\ f\ (f\ z) - c) = 1 * (1 - c)$ **for** *z*
proof (*rule DERIV_unique*)
show $((\lambda x.\ f\ (f\ x) - c * f\ x)$ *has_field_derivative*
$deriv\ f\ z * (deriv\ f\ (f\ z) - c))\ (at\ z)$
by (*rule derivative_eq_intros holomorphic_derivI* [*OF holfU*]
DERIV_chain [*unfolded o_def*, **where** *f*=*f* **and** *g*=*f*] | *simp add*: *algebra_simps*)+
show $((\lambda x.\ f\ (f\ x) - c * f\ x)$ *has_field_derivative* $1 * (1 - c))\ (at\ z)$
by (*simp add*: *eq mult_commute_abs*)
qed
{ fix *z*::*complex*
obtain *k* **where** *k*: $deriv\ f \circ f = (\lambda x.\ k)$
proof (*rule little_Picard*)
show $(deriv\ f \circ f)$ *holomorphic_on UNIV*
by (*meson holfU holomorphic_deriv holomorphic_on_compose holomorphic_on_subset open_UNIV subset_UNIV*)
obtain $deriv\ f\ (f\ x) \neq 0\ deriv\ f\ (f\ x) \neq c$ **for** *x*
using *df_times_dff* ‹$c \neq 1$› *eq_iff_diff_eq_0*
by (*metis lambda_one mult_zero_left mult_zero_right*)
then show $range\ (deriv\ f \circ f) \cap \{0,c\} = \{\}$
by *force*
qed (*use* ‹$c \neq 0$› **in** *auto*)
have ¬ *f constant_on UNIV*
by (*meson UNIV_I non_ff constant_on_def*)
with *holf open_mapping_thm* **have** *open*(*range f*)
by *blast*

```
      obtain l where l: ⋀x. f x − k ∗ x = l
     proof (rule DERIV_zero_connected_constant [of UNIV {} λx. f x − k ∗ x],
simp_all)
       have deriv f w − k = 0 for w
         proof (rule analytic_continuation [OF _ open_UNIV connected_UNIV
subset_UNIV, of λz. deriv f z − k f z range f w])
         show (λz. deriv f z − k) holomorphic_on UNIV
           by (intro holomorphic_intros holf open_UNIV)
         show f z islimpt range f
               by (metis (no_types, lifting) IntI UNIV_I ‹open (range f)› im-
age_eqI inf.absorb_iff2 inf_aci(1) islimpt_UNIV islimpt_eq_acc_point open_Int
top_greatest)
         show ⋀z. z ∈ range f ⟹ deriv f z − k = 0
           by (metis comp_def diff_self image_iff k)
       qed auto
       moreover
       have ((λx. f x − k ∗ x) has_field_derivative deriv f x − k) (at x) for x
            by (metis DERIV_cmult_Id Deriv.field_differentiable_diff UNIV_I
field_differentiable_derivI holf holomorphic_on_def)
       ultimately
       show ∀x. ((λx. f x − k ∗ x) has_field_derivative 0) (at x)
         by auto
       show continuous_on UNIV (λx. f x − k ∗ x)
      by (simp add: continuous_on_diff holf holomorphic_on_imp_continuous_on)
     qed (auto simp: connected_UNIV)
     have False
     proof (cases k=1)
       case True
       then have ∃x. k ∗ x + l ≠ a + x for a
         using l non [of a] ext [of f (+) a]
         by (metis add.commute diff_eq_eq)
       with True show ?thesis by auto
     next
       case False
       have ⋀x. (1 − k) ∗ x ≠ f 0
         using l [of 0]
           by (simp add: algebra_simps) (metis diff_add_cancel l mult.commute
non_fp)
       then show False
      by (metis False eq_iff_diff_eq_0 mult.commute nonzero_mult_div_cancel_right
times_divide_eq_right)
     qed
   }
  }
  then show thesis
    using that by blast
qed
```

7.3 The Arzelà–Ascoli theorem

lemma *subsequence_diagonalization_lemma*:
fixes $P :: nat \Rightarrow (nat \Rightarrow 'a) \Rightarrow bool$
assumes *sub*: $\bigwedge i\ r.\ \exists k.\ strict\_mono\ (k :: nat \Rightarrow nat) \wedge P\ i\ (r \circ k)$
and *P_P*: $\bigwedge i\ r{::}nat \Rightarrow 'a.\ \bigwedge k1\ k2\ N.$
$[\![P\ i\ (r \circ k1);\ \bigwedge j.\ N \le j \Longrightarrow \exists j'.\ j \le j' \wedge k2\ j = k1\ j']\!] \Longrightarrow P\ i\ (r \circ k2)$
obtains *k* **where** *strict_mono* $(k :: nat \Rightarrow nat)$ $\bigwedge i.\ P\ i\ (r \circ k)$
proof −
obtain *kk* **where** $\bigwedge i\ r.\ strict\_mono\ (kk\ i\ r :: nat \Rightarrow nat) \wedge P\ i\ (r \circ (kk\ i\ r))$
using *sub* **by** *metis*
then have *sub_kk*: $\bigwedge i\ r.\ strict\_mono\ (kk\ i\ r)$ **and** *P_kk*: $\bigwedge i\ r.\ P\ i\ (r \circ (kk\ i\ r))$
by *auto*
define *rr* **where** $rr \equiv rec\_nat\ (kk\ 0\ r)\ (\lambda n\ x.\ x \circ kk\ (Suc\ n)\ (r \circ x))$
then have [*simp*]: $rr\ 0 = kk\ 0\ r$ $\bigwedge n.\ rr(Suc\ n) = rr\ n \circ kk\ (Suc\ n)\ (r \circ rr\ n)$
by *auto*
show *thesis*
proof
have *sub_rr*: *strict_mono* $(rr\ i)$ **for** *i*
using *sub_kk* **by** (*induction i*) (*auto simp*: *strict_mono_def o_def*)
have *P_rr*: $P\ i\ (r \circ rr\ i)$ **for** *i*
using *P_kk* **by** (*induction i*) (*auto simp*: *o_def*)
have $i \le i+d \Longrightarrow rr\ i\ n \le rr\ (i+d)\ n$ **for** *d i n*
proof (*induction d*)
case *0* **then show** *?case*
by *simp*
next
case (*Suc d*) **then show** *?case*
using *seq_suble* [*OF sub_kk*] *strict_mono_less_eq* [*OF sub_rr*]
by (*simp add*: *order_subst1*)
qed
then have $\bigwedge i\ j\ n.\ i \le j \Longrightarrow rr\ i\ n \le rr\ j\ n$
by (*metis le_iff_add*)
show *strict_mono* $(\lambda n.\ rr\ n\ n)$
unfolding *strict_mono_Suc_iff*
by (*simp add*: *Suc_le_lessD strict_monoD strict_mono_imp_increasing sub_kk sub_rr*)
have $\exists j.\ i \le j \wedge rr\ (n+d)\ i = rr\ n\ j$ **for** *d n i*
proof (*induction d arbitrary*: *i*)
case (*Suc d*)
then show *?case*
using *seq_suble* [*OF sub_kk*] **by** *simp* (*meson order_trans*)
qed *auto*
then have $\bigwedge m\ n\ i.\ n \le m \Longrightarrow \exists j.\ i \le j \wedge rr\ m\ i = rr\ n\ j$
by (*metis le_iff_add*)
then show $P\ i\ (r \circ (\lambda n.\ rr\ n\ n))$ **for** *i*
by (*meson P_rr P_P*)
qed

qed

lemma *function_convergent_subsequence*:
fixes $f :: [nat,'a] \Rightarrow 'b::\{real\_normed\_vector,heine\_borel\}$
assumes *countable* S **and** M: $\bigwedge n::nat.\ \bigwedge x.\ x \in S \Longrightarrow norm(f\ n\ x) \leq M$
obtains k **where** *strict_mono* $(k::nat\Rightarrow nat)$ $\bigwedge x.\ x \in S \Longrightarrow \exists l.\ (\lambda n.\ f\ (k\ n)\ x) \longrightarrow l$
proof (*cases* $S = \{\}$)
case *True*
then show *?thesis*
using *strict_mono_id that* **by** *fastforce*
next
case *False*
with ‹*countable S*› **obtain** $\sigma :: nat \Rightarrow 'a$ **where** σ: $S = range\ \sigma$
using *uncountable_def* **by** *blast*
obtain k **where** *strict_mono* k **and** k: $\bigwedge i.\ \exists l.\ (\lambda n.\ (f \circ k)\ n\ (\sigma\ i)) \longrightarrow l$
proof (*rule subsequence_diagonalization_lemma*
[*of* $\lambda i\ r.\ \exists l.\ ((\lambda n.\ (f \circ r)\ n\ (\sigma\ i)) \longrightarrow l)$ *sequentially id*])
show $\exists k::nat\Rightarrow nat.\ strict\_mono\ k \wedge (\exists l.\ (\lambda n.\ (f \circ (r \circ k))\ n\ (\sigma\ i)) \longrightarrow l)$
for $i\ r$
proof −
have $f\ (r\ n)\ (\sigma\ i) \in cball\ 0\ M$ **for** n
by (*simp add*: $\sigma\ M$)
then show *?thesis*
using *compact_def* [*of cball* $(0::'b)$ M] **by** (*force simp*: *o_def*)
qed
show $\exists l.\ (\lambda n.\ (f \circ (r \circ k2))\ n\ (\sigma\ i)) \longrightarrow l$
if $\exists l.\ (\lambda n.\ (f \circ (r \circ k1))\ n\ (\sigma\ i)) \longrightarrow l\ \bigwedge j.\ N \leq j \Longrightarrow \exists j'\geq j.\ k2\ j = k1\ j'$
for $i\ N$ **and** $r\ k1\ k2 :: nat\Rightarrow nat$
using *that*
by (*simp add*: *lim_sequentially*) (*metis* (*no_types*, *opaque_lifting*) *le_cases order_trans*)
qed *auto*
with σ *that* **show** *?thesis*
by *force*
qed

theorem *Arzela_Ascoli*:
fixes $\mathcal{F} :: [nat,'a::euclidean\_space] \Rightarrow 'b::\{real\_normed\_vector,heine\_borel\}$
assumes *compact* S
and M: $\bigwedge n\ x.\ x \in S \Longrightarrow norm(\mathcal{F}\ n\ x) \leq M$
and *equicont*:
$\bigwedge x\ e.\ [\![x \in S;\ 0 < e]\!] \Longrightarrow \exists d.\ 0 < d \wedge (\forall n\ y.\ y \in S \wedge norm(x - y) < d \longrightarrow norm(\mathcal{F}\ n\ x - \mathcal{F}\ n\ y) < e)$
obtains $g\ k$ **where** *continuous_on* $S\ g$ *strict_mono* $(k :: nat \Rightarrow nat)$
$\bigwedge e.\ 0 < e \Longrightarrow \exists N.\ \forall n\ x.\ n \geq N \wedge x \in S \longrightarrow norm(\mathcal{F}(k\ n)\ x - g\ x) < e$

proof −
have *UEQ*: $\bigwedge e.\ 0 < e \Longrightarrow \exists d.\ 0 < d \wedge (\forall n.\ \forall x \in S.\ \forall x' \in S.\ dist\ x'\ x < d \longrightarrow dist\ (\mathcal{F}\ n\ x')\ (\mathcal{F}\ n\ x) < e)$
apply (*rule compact_uniformly_equicontinuous* [*OF* ‹*compact S*›, *of range* $\mathcal{F}$])
using *equicont* **by** (*force simp*: *dist_commute dist_norm*)+
have *continuous_on S g*
if $\bigwedge e.\ 0 < e \Longrightarrow \exists N.\ \forall n\ x.\ n \geq N \wedge x \in S \longrightarrow norm(\mathcal{F}(r\ n)\ x - g\ x) < e$
for $g :: 'a \Rightarrow 'b$ **and** $r :: nat \Rightarrow nat$
proof (*rule uniform_limit_theorem* [*of* _ $\mathcal{F} \circ r$])
have *continuous_on S* $(\mathcal{F}\ (r\ n))$ **for** *n*
using *UEQ* **by** (*force simp*: *continuous_on_iff*)
then show $\forall_F\ n$ *in sequentially*. *continuous_on S* $((\mathcal{F} \circ r)\ n)$
by (*simp add*: *eventually_sequentially*)
show *uniform_limit S* $(\mathcal{F} \circ r)$ *g sequentially*
using *that* **by** (*metis* (*mono_tags*, *opaque_lifting*) *comp_apply dist_norm uniform_limit_sequentially_iff*)
qed *auto*
moreover
obtain *R* **where** *countable R* $R \subseteq S$ **and** *SR*: $S \subseteq closure\ R$
by (*metis separable that*)
obtain *k* **where** *strict_mono k* **and** *k*: $\bigwedge x.\ x \in R \Longrightarrow \exists l.\ (\lambda n.\ \mathcal{F}\ (k\ n)\ x) \longrightarrow l$
using ‹$R \subseteq S$› **by** (*force intro*: *function_convergent_subsequence* [*OF* ‹*countable R*› *M*])
then have *Cauchy*: *Cauchy* $((\lambda n.\ \mathcal{F}\ (k\ n)\ x))$ **if** $x \in R$ **for** *x*
using *convergent_eq_Cauchy that* **by** *blast*
have $\exists N.\ \forall m\ n\ x.\ N \leq m \wedge N \leq n \wedge x \in S \longrightarrow dist\ ((\mathcal{F} \circ k)\ m\ x)\ ((\mathcal{F} \circ k)\ n\ x) < e$
if $0 < e$ **for** *e*
proof −
obtain *d* **where** $0 < d$
and *d*: $\bigwedge n.\ \forall x \in S.\ \forall x' \in S.\ dist\ x'\ x < d \longrightarrow dist\ (\mathcal{F}\ n\ x')\ (\mathcal{F}\ n\ x) < e/3$
by (*metis UEQ* ‹$0 < e$› *divide_pos_pos zero_less_numeral*)
obtain *T* **where** $T \subseteq R$ **and** *finite T* **and** *T*: $S \subseteq (\bigcup c \in T.\ ball\ c\ d)$
proof (*rule compactE_image* [*OF* ‹*compact S*›, *of R* $(\lambda x.\ ball\ x\ d)$])
have *closure* $R \subseteq (\bigcup c \in R.\ ball\ c\ d)$
using ‹$0 < d$› **by** (*auto simp*: *closure_approachable*)
with *SR* **show** $S \subseteq (\bigcup c \in R.\ ball\ c\ d)$
by *auto*
qed *auto*
have $\exists M.\ \forall m \geq M.\ \forall n \geq M.\ dist\ (\mathcal{F}\ (k\ m)\ x)\ (\mathcal{F}\ (k\ n)\ x) < e/3$ **if** $x \in R$ **for** *x*
using *Cauchy* ‹$0 < e$› *that* **unfolding** *Cauchy_def*
by (*metis less_divide_eq_numeral1*(*1*) *mult_zero_left*)
then obtain *MF* **where** *MF*: $\bigwedge x\ m\ n.\ [\![x \in R;\ m \geq MF\ x;\ n \geq MF\ x]\!] \Longrightarrow norm\ (\mathcal{F}\ (k\ m)\ x - \mathcal{F}\ (k\ n)\ x) < e/3$
using *dist_norm* **by** *metis*
have *dist* $((\mathcal{F} \circ k)\ m\ x)\ ((\mathcal{F} \circ k)\ n\ x) < e$
if *m*: *Max* $(MF\ `\ T) \leq m$ **and** *n*: *Max* $(MF\ `\ T) \leq n\ x \in S$ **for** *m n x*
proof −

obtain t **where** $t \in T$ **and** t: $x \in \mathit{ball}\ t\ d$
using ‹$x \in S$› T **by** *auto*
have $\mathit{norm}(\mathcal{F}\ (k\ m)\ t - \mathcal{F}\ (k\ m)\ x) < e\ /\ 3$
by (*metis* ‹$R \subseteq S$› ‹$T \subseteq R$› ‹$t \in T$› d *dist_norm mem_ball subset_iff* t ‹$x \in S$›)
moreover
have $\mathit{norm}(\mathcal{F}\ (k\ n)\ t - \mathcal{F}\ (k\ n)\ x) < e\ /\ 3$
by (*metis* ‹$R \subseteq S$› ‹$T \subseteq R$› ‹$t \in T$› *subsetD* d *dist_norm mem_ball* t ‹$x \in S$›)
moreover
have $\mathit{norm}(\mathcal{F}\ (k\ m)\ t - \mathcal{F}\ (k\ n)\ t) < e\ /\ 3$
proof (*rule MF*)
show $t \in R$
using ‹$T \subseteq R$› ‹$t \in T$› **by** *blast*
show $\mathit{MF}\ t \le m\ \mathit{MF}\ t \le n$
by (*meson Max_ge* ‹*finite* T› ‹$t \in T$› *finite_imageI imageI le_trans* m n)+
qed
ultimately
show *?thesis*
unfolding *dist_norm* [*symmetric*] *o_def*
by (*metis dist_triangle_third dist_commute*)
qed
then show *?thesis*
by *force*
qed
then obtain g **where** $\forall e{>}0.\ \exists N.\ \forall n\ x.\ N \le n \wedge x \in S \longrightarrow \mathit{norm}\ ((\mathcal{F} \circ k)\ n\ x - g\ x) < e$
using *uniformly_convergent_eq_cauchy* [*of* $\lambda x.\ x \in S\ \mathcal{F} \circ k$] **by** (*auto simp add*: *dist_norm*)
ultimately show *thesis*
by (*metis* ‹*strict_mono* k› *comp_apply that*)
qed

7.3.1 Montel's theorem

a sequence of holomorphic functions uniformly bounded on compact subsets of an open set S has a subsequence that converges to a holomorphic function, and converges *uniformly* on compact subsets of S.

theorem *Montel*:
fixes $\mathcal{F}$ :: [*nat*,*complex*] ⇒ *complex*
assumes *open* S
and $\mathcal{H}$: $\bigwedge h.\ h \in \mathcal{H} \Longrightarrow h$ *holomorphic_on* S
and *bounded*: $\bigwedge K.\ [\![\mathit{compact}\ K;\ K \subseteq S]\!] \Longrightarrow \exists B.\ \forall h \in \mathcal{H}.\ \forall\ z \in K.\ \mathit{norm}(h\ z) \le B$
and *rng_f*: *range* $\mathcal{F} \subseteq \mathcal{H}$
obtains $g\ r$
where g *holomorphic_on* S *strict_mono* (r :: *nat* ⇒ *nat*)
$\bigwedge x.\ x \in S \Longrightarrow ((\lambda n.\ \mathcal{F}\ (r\ n)\ x) \longrightarrow g\ x)$ *sequentially*

$\bigwedge K.\ [\![compact\ K;\ K \subseteq S]\!] \Longrightarrow uniform\_limit\ K\ (\mathcal{F} \circ r)\ g\ sequentially$

proof −
obtain K **where** $comK$: $\bigwedge n.\ compact(K\ n)$ **and** KS: $\bigwedge n{::}nat.\ K\ n \subseteq S$
and $subK$: $\bigwedge X.\ [\![compact\ X;\ X \subseteq S]\!] \Longrightarrow \exists N.\ \forall n{\geq}N.\ X \subseteq K\ n$
using *open_Union_compact_subsets* [*OF* ‹*open S*›] **by** *metis*
then have $\bigwedge i.\ \exists B.\ \forall h \in \mathcal{H}.\ \forall\ z \in K\ i.\ norm(h\ z) \leq B$
by (*simp add*: *bounded*)
then obtain B **where** B: $\bigwedge i\ h\ z.\ [\![h \in \mathcal{H};\ z \in K\ i]\!] \Longrightarrow norm(h\ z) \leq B\ i$
by *metis*
have $*$: $\exists r\ g.\ strict\_mono\ (r{::}nat{\Rightarrow}nat) \wedge (\forall e > 0.\ \exists N.\ \forall n{\geq}N.\ \forall x \in K\ i.\ norm((\mathcal{F} \circ r)\ n\ x - g\ x) < e)$
if $\bigwedge n.\ \mathcal{F}\ n \in \mathcal{H}$ **for** $\mathcal{F}\ i$
proof −
obtain $g\ k$ **where** *continuous_on* $(K\ i)\ g$ *strict_mono* $(k{::}nat{\Rightarrow}nat)$
$\bigwedge e.\ 0 < e \Longrightarrow \exists N.\ \forall n{\geq}N.\ \forall x \in K\ i.\ norm(\mathcal{F}(k\ n)\ x - g\ x) < e$
proof (*rule Arzela_Ascoli* [*of K i* $\mathcal{F}$ *B i*])
show $\exists d{>}0.\ \forall n\ y.\ y \in K\ i \wedge cmod\ (z - y) < d \longrightarrow cmod\ (\mathcal{F}\ n\ z - \mathcal{F}\ n\ y) < e$
if z: $z \in K\ i$ **and** $0 < e$ **for** $z\ e$
proof −
obtain r **where** $0 < r$ **and** r: *cball z r* $\subseteq S$
using z KS [*of i*] ‹*open S*› **by** (*force simp*: *open_contains_cball*)
have *cball z* $(2/3 * r) \subseteq$ *cball z r*
using ‹$0 < r$› **by** (*simp add*: *cball_subset_cball_iff*)
then have *z23S*: *cball z* $(2/3 * r) \subseteq S$
using r **by** *blast*
obtain M **where** $0 < M$ **and** M: $\bigwedge n\ w.\ dist\ z\ w \leq 2/3 * r \Longrightarrow norm(\mathcal{F}\ n\ w) \leq M$
proof −
obtain N **where** N: $\forall n{\geq}N.$ *cball z* $(2/3 * r) \subseteq K\ n$
using *subK compact_cball* [*of z* $(2/3 * r)$] *z23S* **by** *force*
have *cmod* $(\mathcal{F}\ n\ w) \leq |B\ N| + 1$ **if** $dist\ z\ w \leq 2/3 * r$ **for** $n\ w$
proof −
have $w \in K\ N$
using N *mem_cball that* **by** *blast*
then have *cmod* $(\mathcal{F}\ n\ w) \leq B\ N$
using B ‹$\bigwedge n.\ \mathcal{F}\ n \in \mathcal{H}$› **by** *blast*
also have $... \leq |B\ N| + 1$
by *simp*
finally show *?thesis* **.**
qed
then show *?thesis*
by (*rule_tac* $M{=}|B\ N| + 1$ **in** *that*) *auto*
qed
have *cmod* $(\mathcal{F}\ n\ z - \mathcal{F}\ n\ y) < e$
if $y \in K\ i$ **and** *y_near_z*: *cmod* $(z - y) < r/3$ *cmod* $(z - y) < e * r / (6 * M)$
for $n\ y$

```
      proof −
        have ((λw. ℱ n w / (w − ξ)) has_contour_integral
                (2 * pi) * i * winding_number (circlepath z (2/3 * r)) ξ * ℱ n ξ)
              (circlepath z (2/3 * r))
          if dist ξ z < (2/3 * r) for ξ
        proof (rule Cauchy_integral_formula_convex_simple)
          have ℱ n holomorphic_on S
            by (simp add: ℋ ‹⋀n. ℱ n ∈ ℋ›)
          with z23S show ℱ n holomorphic_on cball z (2/3 * r)
            using holomorphic_on_subset by blast
        qed (use that ‹0 < r› in ‹auto simp: dist_commute›)
        then have *: ((λw. ℱ n w / (w − ξ)) has_contour_integral (2 * pi) * i
* ℱ n ξ)
                    (circlepath z (2/3 * r))
          if dist ξ z < (2/3 * r) for ξ
          using that by (simp add: winding_number_circlepath dist_norm)
        have y: ((λw. ℱ n w / (w − y)) has_contour_integral (2 * pi) * i * ℱ
n y)
                  (circlepath z (2/3 * r))
        proof (rule *)
          show dist y z < 2/3 * r
          using that ‹0 < r› by (simp only: dist_norm norm_minus_commute)
        qed
        have z: ((λw. ℱ n w / (w − z)) has_contour_integral (2 * pi) * i * ℱ n
z)
                (circlepath z (2/3 * r))
          using ‹0 < r› by (force intro!: *)
        have le_er: cmod (ℱ n x / (x − y) − ℱ n x / (x − z)) ≤ e / r
            if cmod (x − z) = r/3 + r/3 for x
        proof −
          have ¬ (cmod (x − y) < r/3)
            using y_near_z(1) that ‹M > 0› ‹r > 0›
          by (metis (full_types) norm_diff_triangle_less norm_minus_commute
order_less_irrefl)
          then have r4_le_xy: r/4 ≤ cmod (x − y)
            using ‹r > 0› by simp
          then have neq: x ≠ y x ≠ z
        using that ‹r > 0› by (auto simp: field_split_simps norm_minus_commute)
          have leM: cmod (ℱ n x) ≤ M
            by (simp add: M dist_commute dist_norm that)
           have cmod (ℱ n x / (x − y) − ℱ n x / (x − z)) = cmod (ℱ n x) *
cmod (1 / (x − y) − 1 / (x − z))
          by (metis (no_types, lifting) divide_inverse mult.left_neutral norm_mult
right_diff_distrib')
          also have ... = cmod (ℱ n x) * cmod ((y − z) / ((x − y) * (x − z)))
            using neq by (simp add: field_split_simps)
          also have ... = cmod (ℱ n x) * (cmod (y − z) / (cmod(x − y) * (2/3
* r)))
            by (simp add: norm_mult norm_divide that)
```

```
          also have ... ≤ M * (cmod (y − z) / (cmod(x − y) * (2/3 * r)))
            using ‹r > 0› ‹M > 0› by (intro mult_mono [OF leM]) auto
          also have ... < M * ((e * r / (6 * M)) / (cmod(x − y) * (2/3 * r)))
            unfolding mult_less_cancel_left
            using y_near_z(2) ‹M > 0› ‹r > 0› neq
            by (simp add: field_simps mult_less_0_iff norm_minus_commute)
          also have ... ≤ e/r
            using ‹e > 0› ‹r > 0› r4_le_xy by (simp add: field_split_simps)
          finally show ?thesis by simp
        qed
        have (2 * pi) * cmod (F n y − F n z) = cmod ((2 * pi) * i * F n y −
(2 * pi) * i * F n z)
          by (simp add: right_diff_distrib [symmetric] norm_mult)
        also have cmod ((2 * pi) * i * F n y − (2 * pi) * i * F n z) ≤ e / r *
(2 * pi * (2/3 * r))

      proof (rule has_contour_integral_bound_circlepath [OF has_contour_integral_diff
[OF y z]])
        show ⋀x. cmod (x − z) = 2/3 * r ⟹ cmod (F n x / (x − y) − F n
x / (x − z)) ≤ e / r
          using le_er by auto
        qed (use ‹e > 0› ‹r > 0› in auto)
        also have ... = (2 * pi) * e * ((2/3))
          using ‹r > 0› by (simp add: field_split_simps)
        finally have cmod (F n y − F n z) ≤ e * (2/3)
          by simp
        also have ... < e
          using ‹e > 0› by simp
        finally show ?thesis by (simp add: norm_minus_commute)
      qed
      then show ?thesis
        apply (rule_tac x=min (r/3) ((e * r)/(6 * M)) in exI)
        using ‹0 < e› ‹0 < r› ‹0 < M› by simp
    qed
    show ⋀n x.  x ∈ K i ⟹ cmod (F n x) ≤ B i
      using B ‹⋀n. F n ∈ ℋ› by blast
  next
    fix g :: complex ⇒ complex and k :: nat ⇒ nat
    assume *: ⋀(g::complex⇒complex) (k::nat⇒nat). continuous_on (K i) g ⟹
                strict_mono k ⟹
                (⋀e. 0 < e ⟹ ∃N. ∀n≥N. ∀x∈K i. cmod (F (k n) x − g x) <
e) ⟹ thesis
        continuous_on (K i) g
        strict_mono k
        ⋀e. 0 < e ⟹ ∃N. ∀n x. N ≤ n ∧ x ∈ K i ⟶ cmod (F (k n) x − g
x) < e
    show ?thesis
      by (rule *(1)[OF *(2,3)], drule *(4)) auto
  qed (use comK in simp_all)
```

```
    then show ?thesis
      by auto
  qed
  define Φ where Φ ≡ λg i r. λk::nat⇒nat. ∀e>0. ∃N. ∀n≥N. ∀x∈K i. cmod
((ℱ ∘ (r ∘ k)) n x − g x) < e
  obtain k :: nat ⇒ nat where strict_mono k and k: ⋀i. ∃g. Φ g i id k
  proof (rule subsequence_diagonalization_lemma [where r=id])
    show ∃g. Φ g i id (r ∘ k2)
      if ex: ∃g. Φ g i id (r ∘ k1) and ⋀j. N ≤ j ⟹ ∃j'≥j. k2 j = k1 j'
      for i k1 k2 N and r::nat⇒nat
    proof −
      obtain g where Φ g i id (r ∘ k1)
        using ex by blast
      then have Φ g i id (r ∘ k2)
        using that
        by (simp add: Φ_def) (metis (no_types, opaque_lifting) le_trans linear)
      then show ?thesis
        by metis
    qed
    have ∃k g. strict_mono (k::nat⇒nat) ∧ Φ g i id (r ∘ k) for i r
      unfolding Φ_def o_assoc using rng_f by (force intro!: ∗)
    then show ⋀i r. ∃k. strict_mono (k::nat⇒nat) ∧ (∃g. Φ g i id (r ∘ k))
      by force
  qed fastforce
  have ∃l. ∀e>0. ∃N. ∀n≥N. norm(ℱ (k n) z − l) < e if z ∈ S for z
  proof −
    obtain G where G: ⋀i e. e > 0 ⟹ ∃M. ∀n≥M. ∀x∈K i. cmod ((ℱ ∘ k) n
x − G i x) < e
      using k unfolding Φ_def by (metis id_comp)
    obtain N where ⋀n. n ≥ N ⟹ z ∈ K n
      using subK [of {z}] that ‹z ∈ S› by auto
    moreover have ⋀e. e > 0 ⟹ ∃M. ∀n≥M. ∀x∈K N. cmod ((ℱ ∘ k) n x −
G N x) < e
      using G by auto
    ultimately show ?thesis
      by (metis comp_apply order_refl)
  qed
  then obtain g where g: ⋀z e. ⟦z ∈ S; e > 0⟧ ⟹ ∃N. ∀n≥N. norm(ℱ (k n)
z − g z) < e
    by metis
  show ?thesis
  proof
    show g_lim: ⋀x. x ∈ S ⟹ (λn. ℱ (k n) x) ⟶ g x
      by (simp add: lim_sequentially g dist_norm)
    have dg_le_e: ∃N. ∀n≥N. ∀x∈T. cmod (ℱ (k n) x − g x) < e
      if T: compact T T ⊆ S and 0 < e for T e
    proof −
      obtain N where N: ⋀n. n ≥ N ⟹ T ⊆ K n
        using subK [OF T] by blast
```

```
    obtain h where h: ⋀e. e>0 ⟹ ∃M. ∀n≥M. ∀x∈K N. cmod ((ℱ ∘ k) n x
− h x) < e
      using k unfolding Φ_def by (metis id_comp)
    have geq: g w = h w if w ∈ T for w
    proof (rule LIMSEQ_unique)
      show (λn. ℱ (k n) w) ⟶ g w
        using ‹T ⊆ S› g_lim that by blast
      show (λn. ℱ (k n) w) ⟶ h w
        using h N that by (force simp: lim_sequentially dist_norm)
    qed
    show ?thesis
      using T h N ‹0 < e› by (fastforce simp add: geq)
  qed
  then show ⋀K. ⟦compact K; K ⊆ S⟧
      ⟹ uniform_limit K (ℱ ∘ k) g sequentially
    by (simp add: uniform_limit_iff dist_norm eventually_sequentially)
  show g holomorphic_on S
  proof (rule holomorphic_uniform_sequence [OF ‹open S› ℋ])
    show ⋀n. (ℱ ∘ k) n ∈ ℋ
      by (simp add: range_subsetD rng_f)
    show ∃d>0. cball z d ⊆ S ∧ uniform_limit (cball z d) (λn. (ℱ ∘ k) n) g
sequentially
      if z ∈ S for z
    proof −
      obtain d where d: d>0 cball z d ⊆ S
        using ‹open S› ‹z ∈ S› open_contains_cball by blast
      then have uniform_limit (cball z d) (ℱ ∘ k) g sequentially
        using dg_le_e compact_cball by (auto simp: uniform_limit_iff eventu-
ally_sequentially dist_norm)
      with d show ?thesis by blast
    qed
  qed
 qed (auto simp: ‹strict_mono k›)
qed
```

7.4 Some simple but useful cases of Hurwitz's theorem

```
proposition Hurwitz_no_zeros:
 assumes S: open S connected S
    and holf: ⋀n::nat. ℱ n holomorphic_on S
    and holg: g holomorphic_on S
    and ul_g: ⋀K. ⟦compact K; K ⊆ S⟧ ⟹ uniform_limit K ℱ g sequentially
    and nonconst: ¬ g constant_on S
    and nz: ⋀n z. z ∈ S ⟹ ℱ n z ≠ 0
    and z0 ∈ S
    shows g z0 ≠ 0
proof
 assume g0: g z0 = 0
 obtain h r m
```

where $0 < m\ 0 < r$ **and** *subS*: $ball\ z0\ r \subseteq S$
and *holh*: h *holomorphic_on* $ball\ z0\ r$
and *geq*: $\bigwedge w.\ w \in ball\ z0\ r \Longrightarrow g\ w = (w - z0)\hat{}m * h\ w$
and *hnz*: $\bigwedge w.\ w \in ball\ z0\ r \Longrightarrow h\ w \neq 0$
by (*blast intro*: *holomorphic_factor_zero_nonconstant* [*OF holg S* ‹$z0 \in S$› *g0 nonconst*])
then have *holf0*: $\mathcal{F}\ n$ *holomorphic_on* $ball\ z0\ r$ **for** n
by (*meson holf holomorphic_on_subset*)
have $*$: $((\lambda z.\ deriv\ (\mathcal{F}\ n)\ z\ /\ \mathcal{F}\ n\ z)\ has\_contour\_integral\ 0)\ (circlepath\ z0\ (r/2))$ **for** n
proof (*rule Cauchy_theorem_disc_simple*)
show $(\lambda z.\ deriv\ (\mathcal{F}\ n)\ z\ /\ \mathcal{F}\ n\ z)$ *holomorphic_on* $ball\ z0\ r$
by (*metis* (*no_types*) ‹*open S*› *holf holomorphic_deriv holomorphic_on_divide holomorphic_on_subset nz subS*)
qed (*use* ‹$0 < r$› **in** *auto*)
have *hol_dg*: *deriv g holomorphic_on S*
by (*simp add*: ‹*open S*› *holg holomorphic_deriv*)
have *continuous_on* (*sphere z0* $(r/2)$) (*deriv g*)
using ‹$0 < r$› *subS*
by (*intro holomorphic_on_imp_continuous_on holomorphic_on_subset* [*OF hol_dg*]) *auto*
then have *compact* (*deriv g* ‘ (*sphere z0* $(r/2)$))
by (*rule compact_continuous_image* [*OF _ compact_sphere*])
then have *bo_dg*: *bounded* (*deriv g* ‘ (*sphere z0* $(r/2)$))
using *compact_imp_bounded* **by** *blast*
have *continuous_on* (*sphere z0* $(r/2)$) $(cmod \circ g)$
using ‹$0 < r$› *subS*
by (*intro continuous_intros holomorphic_on_imp_continuous_on holomorphic_on_subset* [*OF holg*]) *auto*
then have *compact* $((cmod \circ g)$ ‘ *sphere z0* $(r/2))$
by (*rule compact_continuous_image* [*OF _ compact_sphere*])
moreover have $(cmod \circ g)$ ‘ *sphere z0* $(r/2) \neq \{\}$
using ‹$0 < r$› **by** *auto*
ultimately obtain b **where** b: $b \in (cmod \circ g)$ ‘ *sphere z0* $(r/2)$
$\bigwedge t.\ t \in (cmod \circ g)$ ‘ *sphere z0* $(r/2) \Longrightarrow b \leq t$
using *compact_attains_inf* [*of* $(norm \circ g)$ ‘ (*sphere z0* $(r/2)$)] **by** *blast*
have $(\lambda n.\ contour\_integral\ (circlepath\ z0\ (r/2))\ (\lambda z.\ deriv\ (\mathcal{F}\ n)\ z\ /\ \mathcal{F}\ n\ z))$
$\longrightarrow$
contour_integral (*circlepath z0* $(r/2)$) $(\lambda z.\ deriv\ g\ z\ /\ g\ z)$
proof (*rule contour_integral_uniform_limit_circlepath*)
show $\forall_F\ n$ *in sequentially*. $(\lambda z.\ deriv\ (\mathcal{F}\ n)\ z\ /\ \mathcal{F}\ n\ z)$ *contour_integrable_on circlepath z0* $(r/2)$
using $*$ *contour_integrable_on_def eventually_sequentiallyI* **by** *meson*
show *uniform_limit* (*sphere z0* $(r/2)$) $(\lambda n\ z.\ deriv\ (\mathcal{F}\ n)\ z\ /\ \mathcal{F}\ n\ z)$ $(\lambda z.\ deriv\ g\ z\ /\ g\ z)$ *sequentially*
proof (*rule uniform_lim_divide* [*OF _ _ bo_dg*])
show *uniform_limit* (*sphere z0* $(r/2)$) $(\lambda a.\ deriv\ (\mathcal{F}\ a))$ (*deriv g*) *sequentially*
proof (*rule uniform_limitI*)
fix *e*::*real*

```
      assume 0 < e

       show ∀F n in sequentially. ∀x ∈ sphere z0 (r/2). dist (deriv (F n) x)
(deriv g x) < e
      proof −
       have dist (deriv (F n) w) (deriv g w) < e
        if e8: ⋀x. dist z0 x ≤ 3 * r / 4 ⟹ dist (F n x) (g x) * 8 < r * e
          and w: w ∈ sphere z0 (r/2)  for n w
       proof −
        have ball w (r/4) ⊆ ball z0 r  cball w (r/4) ⊆ ball z0 r
       using ‹0 < r› w by (simp_all add: ball_subset_ball_iff cball_subset_ball_iff
dist_commute)
        with subS have wr4_sub: ball w (r/4) ⊆ S cball w (r/4) ⊆ S by force+
        moreover
        have (λz. F n z − g z) holomorphic_on S
          by (intro holomorphic_intros holf holg)
        ultimately have hol: (λz. F n z − g z) holomorphic_on ball w (r/4)
          and cont: continuous_on (cball w (r / 4)) (λz. F n z − g z)
       using holomorphic_on_subset by (blast intro: holomorphic_on_imp_continuous_on)+
        have w ∈ S
          using ‹0 < r› wr4_sub by auto
        have dist z0 y ≤ 3 * r / 4 if dist w y < r/4 for y
        proof (rule dist_triangle_le [where z=w])
          show dist z0 w + dist y w ≤ 3 * r / 4
            using w that by (simp add: dist_commute)
        qed
        with e8 have in_ball: ⋀y. y ∈ ball w (r/4) ⟹ F n y − g y ∈ ball 0
(r/4 * e/2)
          by (simp add: dist_norm [symmetric])
        have F n field_differentiable at w
          by (metis holomorphic_on_imp_differentiable_at ‹w ∈ S› holf ‹open
S›)
        moreover
        have g field_differentiable at w
          using ‹w ∈ S› ‹open S› holg holomorphic_on_imp_differentiable_at
by auto
        moreover
        have cmod (deriv (λw. F n w − g w) w) * 2 ≤ e
          using Cauchy_higher_deriv_bound [OF hol cont in_ball, of 1] ‹r >
0› by auto
        ultimately have dist (deriv (F n) w) (deriv g w) ≤ e/2
          by (simp add: dist_norm)
        then show ?thesis
          using ‹e > 0› by auto
       qed
       moreover
       have cball z0 (3 * r / 4) ⊆ ball z0 r
        by (simp add: cball_subset_ball_iff ‹0 < r›)
       with subS have uniform_limit (cball z0 (3 * r/4)) F g sequentially
```

```
          by (force intro: ul_g)
        then have ∀F n in sequentially. ∀x∈cball z0 (3 * r / 4). dist (ℱ n x) (g
x) < r / 4 * e / 2
          using ‹0 < e› ‹0 < r› by (force simp: intro!: uniform_limitD)
        ultimately show ?thesis
          by (force simp add: eventually_sequentially)
      qed
    qed
    show uniform_limit (sphere z0 (r/2)) ℱ g sequentially
    proof (rule uniform_limitI)
      fix e::real
      assume 0 < e
      have sphere z0 (r/2) ⊆ ball z0 r
        using ‹0 < r› by auto
      with subS have uniform_limit (sphere z0 (r/2)) ℱ g sequentially
        by (force intro: ul_g)
      then show ∀F n in sequentially. ∀x ∈ sphere z0 (r/2). dist (ℱ n x) (g x)
< e
        using ‹0 < e› uniform_limit_iff by blast
    qed
    show b > 0 ⋀x. x ∈ sphere z0 (r/2) ⟹ b ≤ cmod (g x)
      using b ‹0 < r› by (fastforce simp: geq hnz)+
  qed
 qed (use ‹0 < r› in auto)
 then have (λn. 0) ⟶ contour_integral (circlepath z0 (r/2)) (λz. deriv g z
/ g z)
  by (simp add: contour_integral_unique [OF *])
 then have contour_integral (circlepath z0 (r/2)) (λz. deriv g z / g z) = 0
  by (simp add: LIMSEQ_const_iff)
 moreover
 have contour_integral (circlepath z0 (r/2)) (λz. deriv g z / g z) =
      contour_integral (circlepath z0 (r/2)) (λz. m / (z - z0) + deriv h z / h z)
 proof (rule contour_integral_eq, use ‹0 < r› in simp)
  fix w
  assume w: dist z0 w * 2 = r
  then have w_inb: w ∈ ball z0 r
   using ‹0 < r› by auto
  have h_der: (h has_field_derivative deriv h w) (at w)
   using holh holomorphic_derivI w_inb by blast
  have deriv g w = ((of_nat m * h w + deriv h w * (w - z0)) * (w - z0) ^ m)
/ (w - z0)
   if r = dist z0 w * 2 w ≠ z0
  proof -
   have ((λw. (w - z0) ^ m * h w) has_field_derivative
        (m * h w + deriv h w * (w - z0)) * (w - z0) ^ m / (w - z0)) (at w)
     apply (rule derivative_eq_intros h_der refl)+
     using that ‹m > 0› ‹0 < r› apply (simp add: divide_simps distrib_right)
     by (metis Suc_pred mult.commute power_Suc)
   then show ?thesis
```

```
    proof (rule DERIV_imp_deriv [OF has_field_derivative_transform_within_open])
      show ⋀x. x ∈ ball z0 r ⟹ (x − z0) ^ m * h x = g x
        by (simp add: hnz geq)
    qed (use that ‹m > 0› ‹0 < r› in auto)
   qed
   with ‹0 < r› ‹0 < m› w w_inb show deriv g w / g w = of_nat m / (w − z0)
+ deriv h w / h w
    by (auto simp: geq field_split_simps hnz)
  qed
  moreover
  have contour_integral (circlepath z0 (r/2)) (λz. m / (z − z0) + deriv h z / h
z) =
       2 * of_real pi * i * m + 0
  proof (rule contour_integral_unique [OF has_contour_integral_add])
    show ((λx. m / (x − z0)) has_contour_integral 2 * of_real pi * i * m)
(circlepath z0 (r/2))
     by (force simp: ‹0 < r› intro: Cauchy_integral_circlepath_simple)
   show ((λx. deriv h x / h x) has_contour_integral 0) (circlepath z0 (r/2))
     using hnz holh holomorphic_deriv holomorphic_on_divide ‹0 < r›
     by (fastforce intro!: Cauchy_theorem_disc_simple [of _ z0 r])
  qed
  ultimately show False using ‹0 < m› by auto
qed

corollary Hurwitz_injective:
  assumes S: open S connected S
     and holf: ⋀n::nat. F n holomorphic_on S
     and holg: g holomorphic_on S
     and ul_g: ⋀K. ⟦compact K; K ⊆ S⟧ ⟹ uniform_limit K F g sequentially
     and nonconst: ¬ g constant_on S
     and inj: ⋀n. inj_on (F n) S
   shows inj_on g S
proof −
  have False if z12: z1 ∈ S z2 ∈ S z1 ≠ z2 g z2 = g z1 for z1 z2
  proof −
   obtain z0 where z0 ∈ S and z0: g z0 ≠ g z2
     using constant_on_def nonconst by blast
   have (λz. g z − g z1) holomorphic_on S
     by (intro holomorphic_intros holg)
   then obtain r where 0 < r ball z2 r ⊆ S ⋀z. dist z2 z < r ∧ z ≠ z2 ⟹ g
z ≠ g z1
     apply (rule isolated_zeros [of λz. g z − g z1 S z2 z0])
     using S ‹z0 ∈ S› z0 z12 by auto
   have g z2 − g z1 ≠ 0
   proof (rule Hurwitz_no_zeros [of S − {z1} λn z. F n z − F n z1 λz. g z −
g z1])
     show open (S − {z1})
       by (simp add: S open_delete)
     show connected (S − {z1})
```

```
      by (simp add: connected_open_delete [OF S])
    show ⋀n. (λz. F n z − F n z1) holomorphic_on S − {z1}
      by (intro holomorphic_intros holomorphic_on_subset [OF holf]) blast
    show (λz. g z − g z1) holomorphic_on S − {z1}
      by (intro holomorphic_intros holomorphic_on_subset [OF holg]) blast
    show uniform_limit K (λn z. F n z − F n z1) (λz. g z − g z1) sequentially
        if compact K K ⊆ S − {z1} for K
    proof (rule uniform_limitI)
      fix e::real
      assume e > 0
      have uniform_limit K F g sequentially
        using that ul_g by fastforce
      then have K: ∀F n in sequentially. ∀x ∈ K. dist (F n x) (g x) < e/2
        using ‹0 < e› by (force simp: intro!: uniform_limitD)
      have uniform_limit {z1} F g sequentially
        by (simp add: ul_g z12)
      then have ∀F n in sequentially. ∀x ∈ {z1}. dist (F n x) (g x) < e/2
        using ‹0 < e› by (force simp: intro!: uniform_limitD)
      then have z1: ∀F n in sequentially. dist (F n z1) (g z1) < e/2
        by simp
     show ∀F n in sequentially. ∀x∈K. dist (F n x − F n z1) (g x − g z1) < e
        apply (rule eventually_mono [OF  eventually_conj [OF K z1]])
     by (metis (no_types, opaque_lifting) diff_add_eq diff_diff_eq2 dist_commute
dist_norm dist_triangle_add_half)
    qed
    show ¬ (λz. g z − g z1) constant_on S − {z1}
      unfolding constant_on_def
      by (metis Diff_iff ‹z0 ∈ S› empty_iff insert_iff right_minus_eq z0 z12)
    show ⋀n z. z ∈ S − {z1} ⟹ F n z − F n z1 ≠ 0
      by (metis DiffD1 DiffD2 eq_iff_diff_eq_0 inj inj_onD insertI1 ‹z1 ∈ S›)
    show z2 ∈ S − {z1}
      using ‹z2 ∈ S› ‹z1 ≠ z2› by auto
  qed
  with z12 show False by auto
 qed
 then show ?thesis by (auto simp: inj_on_def)
qed
```

7.5 The Great Picard theorem

```
lemma GPicard1:
 assumes S: open S connected S and w ∈ S 0 < r Y ⊆ X
    and holX: ⋀h. h ∈ X ⟹ h holomorphic_on S
    and X01: ⋀h z. ⟦h ∈ X; z ∈ S⟧ ⟹ h z ≠ 0 ∧ h z ≠ 1
    and r:   ⋀h. h ∈ Y ⟹ norm(h w) ≤ r
 obtains B Z where 0 < B open Z w ∈ Z Z ⊆ S ⋀h z. ⟦h ∈ Y; z ∈ Z⟧ ⟹
norm(h z) ≤ B
proof −
 obtain e where e > 0 and e: cball w e ⊆ S
```

```
    using assms open_contains_cball_eq by blast
  show ?thesis
  proof
    show 0 < exp(pi * exp(pi * (2 + 2 * r + 12)))
      by simp
    show ball w (e / 2) ⊆ S
      using e ball_divide_subset_numeral ball_subset_cball by blast
    show cmod (h z) ≤ exp (pi * exp (pi * (2 + 2 * r + 12)))
        if h ∈ Y z ∈ ball w (e / 2) for h z
    proof −
      have h ∈ X
        using ‹Y ⊆ X› ‹h ∈ Y›  by blast
      have hol_h_o: (h ∘ (λz. (w + of_real e * z))) holomorphic_on cball 0 1
      proof (intro holomorphic_intros holomorphic_on_compose)
        have h holomorphic_on S
          using holX ‹h ∈ X› by auto
        then have h holomorphic_on cball w e
          by (metis e holomorphic_on_subset)
        moreover have (λz. w + complex_of_real e * z) ‘ cball 0 1 ⊆ cball w e
          using that ‹e > 0› by (auto simp: dist_norm norm_mult)
        ultimately show h holomorphic_on (λz. w + complex_of_real e * z) ‘
cball 0 1
          by (rule holomorphic_on_subset)
      qed
      have norm_le_r: cmod ((h ∘ (λz. w + complex_of_real e * z)) 0) ≤ r
        by (auto simp: r ‹h ∈ Y›)
      have le12: norm (of_real(inverse e) * (z − w)) ≤ 1/2
      using that ‹e > 0› by (simp add: inverse_eq_divide dist_norm norm_minus_commute
norm_divide)
       have non01: h (w + e * z) ≠ 0 ∧ h (w + e * z) ≠ 1 if z ∈ cball 0 1 for
z::complex
      proof (rule X01 [OF ‹h ∈ X›])
        have w + complex_of_real e * z ∈ cball w e
          using ‹0 < e› that by (auto simp: dist_norm norm_mult)
        then show w + complex_of_real e * z ∈ S
          by (rule subsetD [OF e])
      qed
      have cmod (h z) ≤ cmod (h (w + of_real e * (inverse e * (z − w))))
        using ‹0 < e› by (simp add: field_split_simps)
      also have ... ≤ exp (pi * exp (pi * (14 + 2 * r)))
       using r [OF ‹h ∈ Y›] Schottky [OF hol_h_o norm_le_r _ _ _ le12] non01
by auto
      finally
      show ?thesis by simp
    qed
  qed (use ‹e > 0› in auto)
qed

lemma GPicard2:
```

```
  assumes S ⊆ T connected T S ≠ {} open S ⋀x. ⟦x islimpt S; x ∈ T⟧ ⟹ x ∈ S
    shows S = T
  by (metis assms open_subset connected_clopen closedin_limpt)


lemma GPicard3:
  assumes S: open S connected S w ∈ S and Y ⊆ X
      and holX: ⋀h. h ∈ X ⟹ h holomorphic_on S
      and X01:  ⋀h z. ⟦h ∈ X; z ∈ S⟧ ⟹ h z ≠ 0 ∧ h z ≠ 1
      and no_hw_le1: ⋀h. h ∈ Y ⟹ norm(h w) ≤ 1
      and compact K K ⊆ S
  obtains B where ⋀h z. ⟦h ∈ Y; z ∈ K⟧ ⟹ norm(h z) ≤ B
proof −
  define U where U ≡ {z ∈ S. ∃B Z. 0 < B ∧ open Z ∧ z ∈ Z ∧ Z ⊆ S ∧
                                (∀h z'. h ∈ Y ∧ z' ∈ Z ⟶ norm(h z') ≤ B)}
  then have U ⊆ S by blast
  have U = S
  proof (rule GPicard2 [OF ‹U ⊆ S› ‹connected S›])
    show U ≠ {}
    proof −
      obtain B Z where 0 < B open Z w ∈ Z Z ⊆ S
        and  ⋀h z. ⟦h ∈ Y; z ∈ Z⟧ ⟹ norm(h z) ≤ B
        using GPicard1 [OF S zero_less_one ‹Y ⊆ X› holX] X01 no_hw_le1 by
blast
      then show ?thesis
        unfolding U_def using ‹w ∈ S› by blast
    qed
    show open U
      unfolding open_subopen [of U] by (auto simp: U_def)
    fix v
    assume v: v islimpt U v ∈ S
    have ¬ (∀r>0. ∃h∈Y. r < cmod (h v))
    proof
      assume ∀r>0. ∃h∈Y. r < cmod (h v)
      then have ∀n. ∃h∈Y. Suc n < cmod (h v)
        by simp
      then obtain ℱ where FY: ⋀n. ℱ n ∈ Y and ltF: ⋀n. Suc n < cmod (ℱ
n v)
        by metis
      define 𝒢 where 𝒢 ≡ λn z. inverse(ℱ n z)
      have hol𝒢: 𝒢 n holomorphic_on S for n
      proof (simp add: 𝒢_def)
        show (λz. inverse (ℱ n z)) holomorphic_on S
          using FY X01 ‹Y ⊆ X› holX by (blast intro: holomorphic_on_inverse)
      qed
      have 𝒢not0: 𝒢 n z ≠ 0 and 𝒢not1: 𝒢 n z ≠ 1 if z ∈ S for n z
        using FY X01 ‹Y ⊆ X› that by (force simp: 𝒢_def)+
      have 𝒢_le1: cmod (𝒢 n v) ≤ 1 for n
        using less_le_trans linear ltF
```

```
      by (fastforce simp add: 𝒢_def norm_inverse inverse_le_1_iff)
    define W where W ≡ {h. h holomorphic_on S ∧ (∀z∈S. h z ≠ 0 ∧ h z
≠ 1)}
    obtain B Z where 0 < B open Z v ∈ Z Z ⊆ S
                and B: ⋀h z. ⟦h ∈ range 𝒢; z ∈ Z⟧ ⟹ norm(h z) ≤ B
      apply (rule GPicard1 [OF ‹open S› ‹connected S› ‹v ∈ S› zero_less_one,
of range 𝒢 W])
      using hol𝒢 𝒢not0 𝒢not1 𝒢_le1 by (force simp: W_def)+
    then obtain e where e > 0 and e: ball v e ⊆ Z
      by (meson open_contains_ball)
    obtain h j where holh: h holomorphic_on ball v e and strict_mono j
                and lim:  ⋀x. x ∈ ball v e ⟹ (λn. 𝒢 (j n) x) ⟶ h x
                and ulim: ⋀K. ⟦compact K; K ⊆ ball v e⟧
                          ⟹ uniform_limit K (𝒢 ∘ j) h sequentially
    proof (rule Montel)
      show ⋀h. h ∈ range 𝒢 ⟹ h holomorphic_on ball v e
        by (metis ‹Z ⊆ S› e hol𝒢 holomorphic_on_subset imageE)
      show ⋀K. ⟦compact K; K ⊆ ball v e⟧ ⟹ ∃B. ∀h∈range 𝒢. ∀z∈K. cmod
(h z) ≤ B
        using B e by blast
    qed auto
    have h v = 0
    proof (rule LIMSEQ_unique)
      show (λn. 𝒢 (j n) v) ⟶ h v
        using ‹e > 0› lim by simp
      have lt_Fj: real x ≤ cmod (ℱ (j x) v) for x
        by (metis of_nat_Suc ltF ‹strict_mono j› add.commute less_eq_real_def
less_le_trans nat_le_real_less seq_suble)
      show (λn. 𝒢 (j n) v) ⟶ 0
    proof (rule Lim_null_comparison [OF eventually_sequentiallyI lim_inverse_n])
        show cmod (𝒢 (j x) v) ≤ inverse (real x) if 1 ≤ x for x
          using that by (simp add: 𝒢_def norm_inverse_le_norm [OF lt_Fj])
      qed
    qed
    have h v ≠ 0
    proof (rule Hurwitz_no_zeros [of ball v e 𝒢 ∘ j h])
      show ⋀n. (𝒢 ∘ j) n holomorphic_on ball v e
        using ‹Z ⊆ S› e hol𝒢 by force
      show ⋀n z. z ∈ ball v e ⟹ (𝒢 ∘ j) n z ≠ 0
        using 𝒢not0 ‹Z ⊆ S› e by fastforce
      show ¬ h constant_on ball v e
      proof (clarsimp simp: constant_on_def)
        fix c
        have False if ⋀z. dist v z < e ⟹ h z = c
        proof −
          have h v = c
            by (simp add: ‹0 < e› that)
          obtain y where y ∈ U y ≠ v and y: dist y v < e
            using v ‹e > 0› by (auto simp: islimpt_approachable)
```

```
        then obtain C T where y ∈ S C > 0 open T y ∈ T T ⊆ S
          and ⋀h z'. ⟦h ∈ Y; z' ∈ T⟧ ⟹ cmod (h z') ≤ C
          using ‹y ∈ U› by (auto simp: U_def)
        then have le_C: ⋀n. cmod (ℱ n y) ≤ C
          using FY by blast

        have ∀⇩F n in sequentially. dist (𝒢 (j n) y) (h y) < inverse C
          using uniform_limitD [OF ulim [of {y}], of inverse C] ‹C > 0› y
          by (simp add: dist_commute)
        then obtain n where dist (𝒢 (j n) y) (h y) < inverse C
          by (meson eventually_at_top_linorder order_refl)
        moreover
        have h y = h v
          by (metis ‹h v = c› dist_commute that y)
        ultimately have cmod (inverse (ℱ (j n) y)) < inverse C
          by (simp add: ‹h v = 0› 𝒢_def)
        then have C < norm (ℱ (j n) y)
          by (metis 𝒢_def 𝒢not0 ‹y ∈ S› inverse_less_imp_less inverse_zero
norm_inverse zero_less_norm_iff)
        show False
          using ‹C < cmod (ℱ (j n) y)› le_C not_less by blast
      qed
      then show ∃x∈ball v e. h x ≠ c by force
    qed
    show h holomorphic_on ball v e
      by (simp add: holh)
      show ⋀K. ⟦compact K; K ⊆ ball v e⟧ ⟹ uniform_limit K (𝒢 ∘ j) h
sequentially
      by (simp add: ulim)
  qed (use ‹e > 0› in auto)
  with ‹h v = 0› show False by blast
 qed
 then obtain r where 0 < r and r: ⋀h. h ∈ Y ⟹ cmod (h v) ≤ r
  by (metis not_le)
 moreover
 obtain B Z where 0 < B open Z v ∈ Z Z ⊆ S ⋀h z. ⟦h ∈ Y; z ∈ Z⟧ ⟹
norm(h z) ≤ B
  using X01
  by (auto simp: r intro: GPicard1[OF ‹open S› ‹connected S› ‹v ∈ S› ‹r>0›
‹Y ⊆ X› holX] X01)
 ultimately show v ∈ U
  using v by (simp add: U_def) meson
qed
have ⋀x. x ∈ K ⟶ x ∈ U
 using ‹U = S› ‹K ⊆ S› by blast
then have ⋀x. x ∈ K ⟶ (∃B Z. 0 < B ∧ open Z ∧ x ∈ Z ∧
                        (∀h z'. h ∈ Y ∧ z' ∈ Z ⟶ norm(h z') ≤ B))
 unfolding U_def by blast
then obtain F Z where F: ⋀x. x ∈ K ⟹ open (Z x) ∧ x ∈ Z x ∧
```

```
                    (∀h z'. h ∈ Y ∧ z' ∈ Z x ⟶ norm(h z') ≤ F x)
    by metis
  then obtain L where L ⊆ K finite L and L: K ⊆ (⋃c∈L. Z c)
    by (auto intro: compactE_image [OF ‹compact K›, of K Z])
  then have ∗: ⋀x h z'. ⟦x ∈ L; h ∈ Y ∧ z' ∈ Z x⟧ ⟹ cmod (h z') ≤ F x
    using F by blast
  have ∃B. ∀h z. h ∈ Y ∧ z ∈ K ⟶ norm(h z) ≤ B
  proof (cases L = {})
    case True with L show ?thesis by simp
  next
    case False
    then have ∀h z. h ∈ Y ∧ z ∈ K ⟶ (∃x∈L. cmod (h z) ≤ F x)
      by (metis ∗ L UN_E subset_iff)
    with False ‹finite L› show ?thesis
     by (rule_tac x = Max (F ‘ L) in exI) (simp add: linorder_class.Max_ge_iff)
  qed
  with that show ?thesis by metis
qed


lemma GPicard4:
  assumes 0 < k and holf: f holomorphic_on (ball 0 k − {0})
      and AE: ⋀e. ⟦0 < e; e < k⟧ ⟹ ∃d. 0 < d ∧ d < e ∧ (∀z ∈ sphere 0 d.
norm(f z) ≤ B)
  obtains ε where 0 < ε ε < k ⋀z. z ∈ ball 0 ε − {0} ⟹ norm(f z) ≤ B
proof −
  obtain ε where 0 < ε ε < k/2 and ε: ⋀z. norm z = ε ⟹ norm(f z) ≤ B
    using AE [of k/2] ‹0 < k› by auto
  show ?thesis
  proof
    show ε < k
      using ‹0 < k› ‹ε < k/2› by auto
    show cmod (f ξ) ≤ B if ξ: ξ ∈ ball 0 ε − {0} for ξ
    proof −
      obtain d where 0 < d d < norm ξ and d: ⋀z. norm z = d ⟹ norm(f z)
≤ B
        using AE [of norm ξ] ‹ε < k› ξ by auto
      have [simp]: closure (cball 0 ε − ball 0 d) = cball 0 ε − ball 0 d
        by (blast intro!: closure_closed)
      have [simp]: interior (cball 0 ε − ball 0 d) = ball 0 ε − cball (0::complex) d
        using ‹0 < ε› ‹0 < d› by (simp add: interior_diff)
      have ∗: norm(f w) ≤ B if w ∈ cball 0 ε − ball 0 d for w
      proof (rule maximum_modulus_frontier [of f cball 0 ε − ball 0 d])
        show f holomorphic_on interior (cball 0 ε − ball 0 d)
          using ‹ε < k› ‹0 < d› that by (auto intro: holomorphic_on_subset [OF
holf])
        show continuous_on (closure (cball 0 ε − ball 0 d)) f
       proof (intro holomorphic_on_imp_continuous_on holomorphic_on_subset
[OF holf])
```

```
      show closure (cball 0 ε − ball 0 d) ⊆ ball 0 k − {0}
        using ‹0 < d› ‹ε < k› by auto
    qed
    show ⋀z. z ∈ frontier (cball 0 ε − ball 0 d) ⟹ cmod (f z) ≤ B
      unfolding frontier_def
      using ε d less_eq_real_def by force
  qed (use that in auto)
  show ?thesis
    using ∗ ‹d < cmod ξ› that by auto
 qed
qed (use ‹0 < ε› in auto)
qed


lemma GPicard5:
 assumes holf: f holomorphic_on (ball 0 1 − {0})
    and f01: ⋀z. z ∈ ball 0 1 − {0} ⟹ f z ≠ 0 ∧ f z ≠ 1
 obtains e B where 0 < e e < 1 0 < B
                   (∀ z ∈ ball 0 e − {0}. norm(f z) ≤ B) ∨
                   (∀ z ∈ ball 0 e − {0}. norm(f z) ≥ B)
proof −
 have [simp]: 1 + of_nat n ≠ (0::complex) for n
   using of_nat_eq_0_iff by fastforce
 have [simp]: cmod (1 + of_nat n) = 1 + of_nat n for n
   by (metis norm_of_nat of_nat_Suc)
 have ∗: (λx::complex. x / of_nat (Suc n)) ‘ (ball 0 1 − {0}) ⊆ ball 0 1 − {0}
for n
   by (auto simp: norm_divide field_split_simps split: if_split_asm)
 define h where h ≡ λn z::complex. f (z / (Suc n))
 have holh: (h n) holomorphic_on ball 0 1 − {0} for n
   unfolding h_def
 proof (rule holomorphic_on_compose_gen [unfolded o_def, OF _ holf ∗])
   show (λx. x / of_nat (Suc n)) holomorphic_on ball 0 1 − {0}
     by (intro holomorphic_intros) auto
 qed
 have h01: ⋀n z. z ∈ ball 0 1 − {0} ⟹ h n z ≠ 0 ∧ h n z ≠ 1
   unfolding h_def
   using ∗ by (force intro!: f01)
 obtain w where w: w ∈ ball 0 1 − {0::complex}
   by (rule_tac w = 1/2 in that) auto
 consider infinite {n. norm(h n w) ≤ 1} | infinite {n. 1 ≤ norm(h n w)}
  by (metis (mono_tags, lifting) infinite_nat_iff_unbounded_le le_cases mem_Collect_eq)
 then show ?thesis
 proof cases
   case 1
   with infinite_enumerate obtain r :: nat ⇒ nat
     where strict_mono r and r: ⋀n. r n ∈ {n. norm(h n w) ≤ 1}
     by blast
   obtain B where B: ⋀j z. ⟦norm z = 1/2; j ∈ range (h ∘ r)⟧ ⟹ norm(j z)
```

```
≤ B
    proof (rule GPicard3 [OF _ _ w, where K = sphere 0 (1/2)])
      show range (h ∘ r) ⊆
            {g. g holomorphic_on ball 0 1 − {0} ∧ (∀z ∈ ball 0 1 − {0}. g z ≠ 0
∧ g z ≠ 1)}
        using h01 by (auto intro: holomorphic_intros holomorphic_on_compose
holh)
      show connected (ball 0 1 − {0::complex})
        by (simp add: connected_open_delete)
    qed (use r in auto)
    have normf_le_B: cmod(f z) ≤ B if norm z = 1 / (2 * (1 + of_nat (r n)))
for z n
    proof −
      have *: ⋀w. norm w = 1/2 ⟹ cmod((f (w / (1 + of_nat (r n))))) ≤ B
        using B by (auto simp: h_def o_def)
      have half: norm (z * (1 + of_nat (r n))) = 1/2
        by (simp add: norm_mult divide_simps that)
      show ?thesis
        using * [OF half] by simp
    qed
    obtain ε where 0 < ε ε < 1 ⋀z. z ∈ ball 0 ε − {0} ⟹ cmod(f z) ≤ B
    proof (rule GPicard4 [OF zero_less_one holf, of B])
      fix e::real
      assume 0 < e e < 1
      obtain n where (1/e − 2) / 2 < real n
        using reals_Archimedean2 by blast
      also have ... ≤ r n
        using ‹strict_mono r› by (simp add: seq_suble)
      finally have (1/e − 2) / 2 < real (r n) .
      with ‹0 < e› have e: e > 1 / (2 + 2 * real (r n))
        by (simp add: field_simps)
      show ∃d>0. d < e ∧ (∀z∈sphere 0 d. cmod (f z) ≤ B)
        apply (rule_tac x=1 / (2 * (1 + of_nat (r n))) in exI)
        using normf_le_B by (simp add: e)
    qed blast
    then have ε: cmod (f z) ≤ |B| + 1 if cmod z < ε z ≠ 0 for z
      using that by fastforce
    have 0 < |B| + 1
      by simp
    then show ?thesis
      using ε by (force intro!: that [OF ‹0 < ε› ‹ε < 1›])
  next
    case 2
    with infinite_enumerate obtain r :: nat ⇒ nat
      where strict_mono r and r: ⋀n. r n ∈ {n. norm(h n w) ≥ 1}
      by blast
    obtain B where B: ⋀j z. ⟦norm z = 1/2; j ∈ range (λn. inverse ∘ h (r n))⟧
⟹ norm(j z) ≤ B
    proof (rule GPicard3 [OF _ _ w, where K = sphere 0 (1/2)])
```

```
    show range (λn. inverse ∘ h (r n)) ⊆
        {g. g holomorphic_on ball 0 1 − {0} ∧ (∀z∈ball 0 1 − {0}. g z ≠ 0 ∧
g z ≠ 1)}
    using h01 by (auto intro!: holomorphic_intros holomorphic_on_compose_gen
[unfolded o_def, OF _ holh] holomorphic_on_compose)
    show connected (ball 0 1 − {0::complex})
      by (simp add: connected_open_delete)
    show ⋀j. j ∈ range (λn. inverse ∘ h (r n)) ⟹ cmod (j w) ≤ 1
      using r norm_inverse_le_norm by fastforce
  qed (use r in auto)
  have norm_if_le_B: cmod(inverse (f z)) ≤ B if norm z = 1 / (2 * (1 +
of_nat (r n))) for z n
  proof −
    have ∗: inverse (cmod((f (z / (1 + of_nat (r n)))))) ≤ B if norm z = 1/2
for z
      using B [OF that] by (force simp: norm_inverse h_def)
    have half: norm (z * (1 + of_nat (r n))) = 1/2
      by (simp add: norm_mult divide_simps that)
    show ?thesis
      using ∗ [OF half] by (simp add: norm_inverse)
  qed
  have hol_if: (inverse ∘ f) holomorphic_on (ball 0 1 − {0})
    by (metis (no_types, lifting) holf comp_apply f01 holomorphic_on_inverse
holomorphic_transform)
  obtain ε where 0 < ε ε < 1 and leB: ⋀z. z ∈ ball 0 ε − {0} ⟹ cmod((inverse
∘ f) z) ≤ B
  proof (rule GPicard4 [OF zero_less_one hol_if, of B])
    fix e::real
    assume 0 < e e < 1
    obtain n where (1/e − 2) / 2 < real n
      using reals_Archimedean2 by blast
    also have ... ≤ r n
      using ‹strict_mono r› by (simp add: seq_suble)
    finally have (1/e − 2) / 2 < real (r n) .
    with ‹0 < e› have e: e > 1 / (2 + 2 * real (r n))
      by (simp add: field_simps)
    show ∃d>0. d < e ∧ (∀z∈sphere 0 d. cmod ((inverse ∘ f) z) ≤ B)
      apply (rule_tac x=1 / (2 * (1 + of_nat (r n))) in exI)
      using norm_if_le_B by (simp add: e)
  qed blast
  have ε: cmod (f z) ≥ inverse B and B > 0 if cmod z < ε z ≠ 0 for z
  proof −
    have inverse (cmod (f z)) ≤ B
      using leB that by (simp add: norm_inverse)
    moreover
    have f z ≠ 0
      using ‹ε < 1› f01 that by auto
    ultimately show cmod (f z) ≥ inverse B
      by (simp add: norm_inverse inverse_le_imp_le)
```

```
    show B > 0
      using ‹f z ≠ 0› ‹inverse (cmod (f z)) ≤ B› not_le order.trans by fastforce
  qed
  then have B > 0
    by (metis ‹0 < ε› dense leI order.asym vector_choose_size)
  then have inverse B > 0
    by (simp add: field_split_simps)
  then show ?thesis
    using ε that [OF ‹0 < ε› ‹ε < 1›]
    by (metis Diff_iff dist_0_norm insert_iff mem_ball)
 qed
qed


lemma GPicard6:
 assumes open M z ∈ M a ≠ 0 and holf: f holomorphic_on (M − {z})
    and f0a: ⋀w. w ∈ M − {z} ⟹ f w ≠ 0 ∧ f w ≠ a
 obtains r where 0 < r ball z r ⊆ M
                 bounded(f ‘ (ball z r − {z})) ∨
                 bounded((inverse ∘ f) ‘ (ball z r − {z}))
proof −
 obtain r where 0 < r and r: ball z r ⊆ M
   using assms openE by blast
 let ?g = λw. f (z + of_real r ∗ w) / a
 obtain e B where 0 < e e < 1 0 < B
    and B: (∀z ∈ ball 0 e − {0}. norm(?g z) ≤ B) ∨ (∀z ∈ ball 0 e − {0}.
norm(?g z) ≥ B)
 proof (rule GPicard5)
   show ?g holomorphic_on ball 0 1 − {0}
  proof (intro holomorphic_intros holomorphic_on_compose_gen [unfolded o_def,
OF _ holf])
    show (λx. z + complex_of_real r ∗ x) ‘ (ball 0 1 − {0}) ⊆ M − {z}
      using ‹0 < r› r
      by (auto simp: dist_norm norm_mult subset_eq)
   qed (use ‹a ≠ 0› in auto)
   show ⋀w. w ∈ ball 0 1 − {0} ⟹ f (z + of_real r ∗ w) / a ≠ 0 ∧ f (z +
of_real r ∗ w) / a ≠ 1
    using f0a ‹0 < r› ‹a ≠ 0› r
    by (auto simp: field_split_simps dist_norm norm_mult subset_eq)
 qed
 show ?thesis
 proof
   show 0 < e∗r
    by (simp add: ‹0 < e› ‹0 < r›)
   have ball z (e ∗ r) ⊆ ball z r
    by (simp add: ‹0 < r› ‹e < 1› order.strict_implies_order subset_ball)
   then show ball z (e ∗ r) ⊆ M
    using r by blast
  consider ⋀z. z ∈ ball 0 e − {0} ⟹ norm(?g z) ≤ B | ⋀z. z ∈ ball 0 e − {0}
```

```
⟹ norm(?g z) ≥ B
      using B by blast
    then show bounded (f ‘ (ball z (e * r) - {z})) ∨
          bounded ((inverse ∘ f) ‘ (ball z (e * r) - {z}))
    proof cases
      case 1
      have ⟦dist z w < e * r; w ≠ z⟧ ⟹ cmod (f w) ≤ B * norm a for w
        using ‹a ≠ 0› ‹0 < r› 1 [of (w - z) / r]
        by (simp add: norm_divide dist_norm field_split_simps)
      then show ?thesis
        by (force simp: intro!: boundedI)
    next
      case 2
      have ⟦dist z w < e * r; w ≠ z⟧ ⟹ cmod (f w) ≥ B * norm a for w
        using ‹a ≠ 0› ‹0 < r› 2 [of (w - z) / r]
        by (simp add: norm_divide dist_norm field_split_simps)
      then have ⟦dist z w < e * r; w ≠ z⟧ ⟹ inverse (cmod (f w)) ≤ inverse (B
* norm a) for w
      by (metis ‹0 < B› ‹a ≠ 0› mult_pos_pos norm_inverse norm_inverse_le_norm
zero_less_norm_iff)
      then show ?thesis
        by (force simp: norm_inverse intro!: boundedI)
    qed
  qed
qed


theorem great_Picard:
  assumes open M z ∈ M a ≠ b and holf: f holomorphic_on (M - {z})
      and fab: ⋀w. w ∈ M - {z} ⟹ f w ≠ a ∧ f w ≠ b
  obtains l where (f ⟶ l) (at z) ∨ ((inverse ∘ f) ⟶ l) (at z)
proof -
  obtain r where 0 < r and zrM: ball z r ⊆ M
            and r: bounded((λz. f z - a) ‘ (ball z r - {z})) ∨
                   bounded((inverse ∘ (λz. f z - a)) ‘ (ball z r - {z}))
  proof (rule GPicard6 [OF ‹open M› ‹z ∈ M›])
    show b - a ≠ 0
      using assms by auto
    show (λz. f z - a) holomorphic_on M - {z}
      by (intro holomorphic_intros holf)
  qed (use fab in auto)
  have holfb: f holomorphic_on ball z r - {z}
    using zrM by (auto intro: holomorphic_on_subset [OF holf])
  have holfb_i: (λz. inverse(f z - a)) holomorphic_on ball z r - {z}
    using fab zrM by (fastforce intro!: holomorphic_intros holfb)
  show ?thesis
    using r
  proof
    assume bounded ((λz. f z - a) ‘ (ball z r - {z}))
```

then obtain B **where** B: $\bigwedge w.\ w \in (\lambda z.\ f\ z - a)\ `\ (ball\ z\ r - \{z\}) \Longrightarrow norm\ w \leq B$
by (*force simp*: *bounded_iff*)
then have $\forall x.\ x \neq z \wedge dist\ x\ z < r \longrightarrow cmod\ (f\ x - a) \leq B$
by (*simp add*: *dist_commute*)
with ‹$0 < r$› **have** $\forall_F\ w\ in\ at\ z.\ cmod\ (f\ w - a) \leq B$
by (*force simp add*: *eventually_at*)
moreover have $\bigwedge x.\ cmod\ (f\ x - a) \leq B \Longrightarrow cmod\ (f\ x) \leq B + cmod\ a$
by (*metis add.commute add_le_cancel_right norm_triangle_sub order.trans*)
ultimately have $\exists B.\ \forall_F\ w\ in\ at\ z.\ cmod\ (f\ w) \leq B$
by (*metis* (*mono_tags*, *lifting*) *eventually_at*)
then obtain g **where** *holg*: $g\ holomorphic\_on\ ball\ z\ r$ **and** *gf*: $\bigwedge w.\ w \in ball\ z\ r - \{z\} \Longrightarrow g\ w = f\ w$
using ‹$0 < r$› *holomorphic_on_extend_bounded* [*OF holfb*] **by** *auto*
then have $g\ -z\rightarrow\ g\ z$
unfolding *continuous_at* [*symmetric*]
using ‹$0 < r$› *centre_in_ball field_differentiable_imp_continuous_at*
holomorphic_on_imp_differentiable_at **by** *blast*
then have $(f \longrightarrow g\ z)\ (at\ z)$
using *Lim_transform_within_open* [*of g g z z*]
using ‹$0 < r$› *centre_in_ball gf* **by** *blast*
then show *?thesis*
using *that* **by** *blast*
next
assume *bounded*((*inverse* ∘ $(\lambda z.\ f\ z - a))\ `\ (ball\ z\ r - \{z\})$)
then obtain B **where** B: $\bigwedge w.\ w \in (inverse \circ (\lambda z.\ f\ z - a))\ `\ (ball\ z\ r - \{z\}) \Longrightarrow norm\ w \leq B$
by (*force simp*: *bounded_iff*)
then have $\forall x.\ x \neq z \wedge dist\ x\ z < r \longrightarrow cmod\ (inverse\ (f\ x - a)) \leq B$
by (*simp add*: *dist_commute*)
with ‹$0 < r$› **have** $\forall_F\ w\ in\ at\ z.\ cmod\ (inverse\ (f\ w - a)) \leq B$
by (*auto simp add*: *eventually_at*)
then have $\exists B.\ \forall_F\ z\ in\ at\ z.\ cmod\ (inverse\ (f\ z - a)) \leq B$
by *blast*
then obtain g **where** *holg*: $g\ holomorphic\_on\ ball\ z\ r$ **and** *gf*: $\bigwedge w.\ w \in ball\ z\ r - \{z\} \Longrightarrow g\ w = inverse\ (f\ w - a)$
using ‹$0 < r$› *holomorphic_on_extend_bounded* [*OF holfb_i*] **by** *auto*
then have *gz*: $g\ -z\rightarrow\ g\ z$
unfolding *continuous_at* [*symmetric*]
using ‹$0 < r$› *centre_in_ball field_differentiable_imp_continuous_at*
holomorphic_on_imp_differentiable_at **by** *blast*
have *gnz*: $\bigwedge w.\ w \in ball\ z\ r - \{z\} \Longrightarrow g\ w \neq 0$
using *gf fab zrM* **by** *fastforce*
show *?thesis*
proof (*cases* $g\ z = 0$)
case *True*
have $*$: $[\![g \neq 0;\ inverse\ g = f - a]\!] \Longrightarrow g\ /\ (1 + a * g) = inverse\ f$ **for** *f g*::*complex*
by (*auto simp*: *field_simps*)

```
    have (inverse ∘ f) −z→ 0
    proof (rule Lim_transform_within_open [of λw. g w / (1 + a * g w) _ _
UNIV ball z r])
      show (λw. g w / (1 + a * g w)) −z→ 0
        using True by (auto simp: intro!: tendsto_eq_intros gz)
      show ⋀x. ⟦x ∈ ball z r; x ≠ z⟧ ⟹ g x / (1 + a * g x) = (inverse ∘ f) x
        using * gf gnz by simp
    qed (use ‹0 < r› in auto)
    with that show ?thesis by blast
  next
    case False
    show ?thesis
    proof (cases 1 + a * g z = 0)
      case True
      have (f ⟶ 0) (at z)
      proof (rule Lim_transform_within_open [of λw. (1 + a * g w) / g w _ _
_ ball z r])
        show (λw. (1 + a * g w) / g w) −z→ 0
          by (rule tendsto_eq_intros refl gz ‹g z ≠ 0› | simp add: True)+
        show ⋀x. ⟦x ∈ ball z r; x ≠ z⟧ ⟹ (1 + a * g x) / g x = f x
          using fab fab zrM by (fastforce simp add: gf field_split_simps)
      qed (use ‹0 < r› in auto)
      then show ?thesis
        using that by blast
    next
      case False
      have *: ⟦g ≠ 0; inverse g = f − a⟧ ⟹ g / (1 + a * g) = inverse f for f
g::complex
        by (auto simp: field_simps)
      have (inverse ∘ f) −z→ g z / (1 + a * g z)
      proof (rule Lim_transform_within_open [of λw. g w / (1 + a * g w) _ _
UNIV ball z r])
        show (λw. g w / (1 + a * g w)) −z→ g z / (1 + a * g z)
          using False by (auto simp: False intro!: tendsto_eq_intros gz)
        show ⋀x. ⟦x ∈ ball z r; x ≠ z⟧ ⟹ g x / (1 + a * g x) = (inverse ∘ f) x
          using * gf gnz by simp
      qed (use ‹0 < r› in auto)
      with that show ?thesis by blast
    qed
  qed
 qed
qed


corollary great_Picard_alt:
  assumes M: open M z ∈ M and holf: f holomorphic_on (M − {z})
    and non: ⋀l. ¬ (f ⟶ l) (at z) ⋀l. ¬ ((inverse ∘ f) ⟶ l) (at z)
  obtains a where − {a} ⊆ f ‘ (M − {z})
unfolding subset_iff image_iff
```

by (*metis great_Picard* [*OF M _ holf*] *non Compl_iff insertI1*)

corollary *great_Picard_infinite*:
assumes *M*: *open M z* ∈ *M* **and** *holf*: *f holomorphic_on* (*M* − {*z*})
and *non*: ⋀*l*. ¬ (*f* ⟶ *l*) (*at z*) ⋀*l*. ¬ ((*inverse* ∘ *f*) ⟶ *l*) (*at z*)
obtains *a* **where** ⋀*w*. *w* ≠ *a* ⟹ *infinite* {*x*. *x* ∈ *M* − {*z*} ∧ *f x* = *w*}
proof −
have *False* **if** *a* ≠ *b* **and** *ab*: *finite* {*x*. *x* ∈ *M* − {*z*} ∧ *f x* = *a*} *finite* {*x*. *x* ∈ *M* − {*z*} ∧ *f x* = *b*} **for** *a b*
proof −
have *finab*: *finite* {*x*. *x* ∈ *M* − {*z*} ∧ *f x* ∈ {*a*,*b*}}
using *finite_UnI* [*OF ab*] **unfolding** *mem_Collect_eq insert_iff empty_iff*
by (*simp add*: *conj_disj_distribL*)
obtain *r* **where** 0 < *r* **and** *zrM*: *ball z r* ⊆ *M* **and** *r*: ⋀*x*. ⟦*x* ∈ *M* − {*z*}; *f x* ∈ {*a*,*b*}⟧ ⟹ *x* ∉ *ball z r*
proof −
obtain *e* **where** *e* > 0 **and** *e*: *ball z e* ⊆ *M*
using *assms openE* **by** *blast*
show *?thesis*
proof (*cases* {*x* ∈ *M* − {*z*}. *f x* ∈ {*a*, *b*}} = {})
case *True*
then show *?thesis*
using *e* ‹*e* > 0› *that* **by** *fastforce*
next
case *False*
let *?r* = *min e* (*Min* (*dist z* ‘ {*x* ∈ *M* − {*z*}. *f x* ∈ {*a*,*b*}}))
show *?thesis*
proof
show 0 < *?r*
using *min_less_iff_conj Min_gr_iff finab False* ‹0 < *e*› **by** *auto*
have *ball z ?r* ⊆ *ball z e*
by (*simp add*: *subset_ball*)
with *e* **show** *ball z ?r* ⊆ *M* **by** *blast*
show ⋀*x*. ⟦*x* ∈ *M* − {*z*}; *f x* ∈ {*a*, *b*}⟧ ⟹ *x* ∉ *ball z ?r*
using *min_less_iff_conj Min_gr_iff finab False* ‹0 < *e*› **by** *auto*
qed
qed
qed
have *holfb*: *f holomorphic_on* (*ball z r* − {*z*})
apply (*rule holomorphic_on_subset* [*OF holf*])
using *zrM* **by** *auto*
show *?thesis*
apply (*rule great_Picard* [*OF open_ball _* ‹*a* ≠ *b*› *holfb*])
using *non* ‹0 < *r*› *r zrM* **by** *auto*
qed
with *that* **show** *thesis*
by *meson*
qed

theorem *Casorati_Weierstrass*:
assumes *open M z* ∈ *M f holomorphic_on* (*M* − {*z*})
and ⋀*l*. ¬ (*f* ⟶ *l*) (*at z*) ⋀*l*. ¬ ((*inverse* ∘ *f*) ⟶ *l*) (*at z*)
shows *closure*(*f* ‘ (*M* − {*z*})) = *UNIV*
proof −
obtain *a* **where** *a*: − {*a*} ⊆ *f* ‘ (*M* − {*z*})
using *great_Picard_alt* [*OF assms*] **.**
have *UNIV* = *closure*(− {*a*})
by (*simp add*: *closure_interior*)
also have ... ⊆ *closure*(*f* ‘ (*M* − {*z*}))
by (*simp add*: *a closure_mono*)
finally show *?thesis*
by *blast*
qed

end

8 Moebius functions, Equivalents of Simply Connected Sets, Riemann Mapping Theorem

theory *Riemann_Mapping*
imports *Great_Picard*
begin

8.1 Moebius functions are biholomorphisms of the unit disc

definition *Moebius_function* :: [*real*,*complex*,*complex*] ⇒ *complex* **where**
Moebius_function ≡ λ*t w z*. *exp*(i ∗ *of_real t*) ∗ (*z* − *w*) / (*1* − *cnj w* ∗ *z*)

lemma *Moebius_function_simple*:
Moebius_function 0 w z = (*z* − *w*) / (*1* − *cnj w* ∗ *z*)
by (*simp add*: *Moebius_function_def*)

lemma *Moebius_function_eq_zero*:
Moebius_function t w w = *0*
by (*simp add*: *Moebius_function_def*)

lemma *Moebius_function_of_zero*:
Moebius_function t w 0 = − *exp*(i ∗ *of_real t*) ∗ *w*
by (*simp add*: *Moebius_function_def*)

lemma *Moebius_function_norm_lt_1*:
assumes *w1*: *norm w* < *1* **and** *z1*: *norm z* < *1*
shows *norm* (*Moebius_function t w z*) < *1*
proof −
have *1* − *cnj w* ∗ *z* ≠ *0*
by (*metis complex_cnj_cnj complex_mod_sqrt_Re_mult_cnj mult.commute*

mult_less_cancel_right1 norm_ge_zero norm_mult norm_one order.asym right_minus_eq w1 z1)
then have *VV*: *1 − w ∗ cnj z ≠ 0*
by (*metis complex_cnj_cnj complex_cnj_mult complex_cnj_one right_minus_eq*)
then have *1 − norm (Moebius_function t w z) ^ 2 =*
((1 − norm w ^ 2) / (norm (1 − cnj w ∗ z) ^ 2)) ∗ (1 − norm z ^ 2)
apply (*cases w*)
apply (*cases z*)
apply (*simp add*: *Moebius_function_def divide_simps norm_divide norm_mult*)
apply (*simp add*: *complex_norm complex_diff complex_mult one_complex.code complex_cnj*)
apply (*auto simp*: *algebra_simps power2_eq_square*)
done
then have *1 − (cmod (Moebius_function t w z))² = (1 − cmod (w ∗ w)) / (cmod (1 − cnj w ∗ z))² ∗ (1 − cmod (z ∗ z))*
by (*simp add*: *norm_mult power2_eq_square*)
moreover have *0 < 1 − cmod (z ∗ z)*
by (*metis* (*no_types*) *z1 diff_gt_0_iff_gt mult.left_neutral norm_mult_less*)
ultimately have *0 < 1 − norm (Moebius_function t w z) ^ 2*
using ‹*1 − cnj w ∗ z ≠ 0*› *w1 norm_mult_less* **by** *fastforce*
then show *?thesis*
using *linorder_not_less* **by** *fastforce*
qed

lemma *Moebius_function_holomorphic*:
assumes *norm w < 1*
shows *Moebius_function t w holomorphic_on ball 0 1*
proof −
have ∗: *1 − z ∗ w ≠ 0* **if** *norm z < 1* **for** *z*
proof −
have *norm (1*::*complex) ≠ norm (z ∗ w)*
using *assms that norm_mult_less* **by** *fastforce*
then show *?thesis* **by** *auto*
qed
show *?thesis*
apply (*simp add*: *Moebius_function_def*)
apply (*intro holomorphic_intros*)
using *assms* ∗
by (*metis complex_cnj_cnj complex_cnj_mult complex_cnj_one complex_mod_cnj mem_ball_0 mult.commute right_minus_eq*)
qed

lemma *Moebius_function_compose*:
assumes *meq*: *−w1 = w2* **and** *norm w1 < 1 norm z < 1*
shows *Moebius_function 0 w1 (Moebius_function 0 w2 z) = z*
proof −
have *norm w2 < 1*
using *assms* **by** *auto*
then have *−w1 = z* **if** *cnj w2 ∗ z = 1*

by (*metis assms*(3) *complex_mod_cnj less_irrefl mult.right_neutral norm_mult_less norm_one that*)
moreover have *z=0* **if** *1 − cnj w2 ∗ z = cnj w1 ∗ (z − w2)*
proof −
have *w2 ∗ cnj w2 = 1*
using *that meq* **by** (*auto simp*: *algebra_simps*)
then show *z = 0*
by (*metis* (*no_types*) ‹*cmod w2 < 1*› *complex_mod_cnj less_irrefl mult.right_neutral norm_mult_less norm_one*)
qed
moreover have *z − w2 − w1 ∗ (1 − cnj w2 ∗ z) = z ∗ (1 − cnj w2 ∗ z − cnj w1 ∗ (z − w2))*
using *meq* **by** (*fastforce simp*: *algebra_simps*)
ultimately
show *?thesis*
by (*simp add*: *Moebius_function_def divide_simps norm_divide norm_mult*)
qed

lemma *ball_biholomorphism_exists*:
assumes *a ∈ ball 0 1*
obtains *f g* **where** *f a = 0*
f holomorphic_on ball 0 1 f ‘ ball 0 1 ⊆ ball 0 1
g holomorphic_on ball 0 1 g ‘ ball 0 1 ⊆ ball 0 1
⋀z. z ∈ ball 0 1 ⟹ f (g z) = z
⋀z. z ∈ ball 0 1 ⟹ g (f z) = z
proof
show *Moebius_function 0 a holomorphic_on ball 0 1 Moebius_function 0 (−a) holomorphic_on ball 0 1*
using *Moebius_function_holomorphic assms mem_ball_0* **by** *auto*
show *Moebius_function 0 a a = 0*
by (*simp add*: *Moebius_function_eq_zero*)
show *Moebius_function 0 a ‘ ball 0 1 ⊆ ball 0 1*
Moebius_function 0 (− a) ‘ ball 0 1 ⊆ ball 0 1
using *Moebius_function_norm_lt_1 assms* **by** *auto*
show *Moebius_function 0 a (Moebius_function 0 (− a) z) = z*
Moebius_function 0 (− a) (Moebius_function 0 a z) = z **if** *z ∈ ball 0 1* **for** *z*
using *Moebius_function_compose assms that* **by** *auto*
qed

8.2 A big chain of equivalents of simple connectedness for an open set

lemma *biholomorphic_to_disc_aux*:
assumes *open S connected S 0 ∈ S* **and** *S01*: *S ⊆ ball 0 1*
and *prev*: *⋀f. ⟦f holomorphic_on S*; *⋀z. z ∈ S ⟹ f z ≠ 0*; *inj_on f S⟧*
⟹ ∃ g. g holomorphic_on S ∧ (∀ z ∈ S. f z = (g z)²)
shows *∃ f g. f holomorphic_on S ∧ g holomorphic_on ball 0 1 ∧*
(∀ z ∈ S. f z ∈ ball 0 1 ∧ g(f z) = z) ∧

```
            (∀ z ∈ ball 0 1. g z ∈ S ∧ f(g z) = z)
proof −
  define F where F ≡ {h. h holomorphic_on S ∧ h ‘ S ⊆ ball 0 1 ∧ h 0 = 0 ∧
inj_on h S}
  have idF: id ∈ F
    using S01 by (auto simp: F_def)
  then have F ≠ {}
    by blast
  have imF_ne: ((λh. norm(deriv h 0)) ‘ F) ≠ {}
    using idF by auto
  have holF: ⋀h. h ∈ F ⟹ h holomorphic_on S
    by (auto simp: F_def)
 obtain f where f ∈ F and normf: ⋀h. h ∈ F ⟹ norm(deriv h 0) ≤ norm(deriv
f 0)
  proof −
    obtain r where r > 0 and r: ball 0 r ⊆ S
      using ‹open S› ‹0 ∈ S› openE by auto
    have bdd: bdd_above ((λh. norm(deriv h 0)) ‘ F)
    proof (intro bdd_aboveI exI ballI, clarify)
      show norm (deriv f 0) ≤ 1 / r if f ∈ F for f
      proof −
        have r01: (∗) (complex_of_real r) ‘ ball 0 1 ⊆ S
          using that ‹r > 0› by (auto simp: norm_mult r [THEN subsetD])
        then have f holomorphic_on (∗) (complex_of_real r) ‘ ball 0 1
          using holomorphic_on_subset [OF holF] by (simp add: that)
        then have holf: f ∘ (λz. (r ∗ z)) holomorphic_on (ball 0 1)
          by (intro holomorphic_intros holomorphic_on_compose)
        have f0: (f ∘ (∗) (complex_of_real r)) 0 = 0
          using F_def that by auto
        have f ‘ S ⊆ ball 0 1
          using F_def that by blast
        with r01 have fr1: ⋀z. norm z < 1 ⟹ norm ((f ∘ (∗)(of_real r))z) < 1
          by force
        have ∗: ((λw. f (r ∗ w)) has_field_derivative deriv f (r ∗ z) ∗ r) (at z)
          if z ∈ ball 0 1 for z::complex
        proof (rule DERIV_chain' [where g=f])
          show (f has_field_derivative deriv f (of_real r ∗ z)) (at (of_real r ∗ z))
            apply (rule holomorphic_derivI [OF holF ‹open S›])
             apply (rule ‹f ∈ F›)
            by (meson imageI r01 subset_iff that)
        qed simp
        have df0: ((λw. f (r ∗ w)) has_field_derivative deriv f 0 ∗ r) (at 0)
          using ∗ [of 0] by simp
          have deq: deriv (λx. f (complex_of_real r ∗ x)) 0 = deriv f 0 ∗ com-
plex_of_real r
          using DERIV_imp_deriv df0 by blast
        have norm (deriv (f ∘ (∗) (complex_of_real r)) 0) ≤ 1
          by (auto intro: Schwarz_Lemma [OF holf f0 fr1, of 0])
        with ‹r > 0› show ?thesis
```

```
      by (simp add: deq norm_mult divide_simps o_def)
    qed
  qed
  define l where l ≡ SUP h∈F. norm (deriv h 0)
  have eql: norm (deriv f 0) = l if le: l ≤ norm (deriv f 0) and f ∈ F for f
    apply (rule order_antisym [OF _ le])
    using ‹f ∈ F› bdd cSUP_upper by (fastforce simp: l_def)
  obtain ℱ where ℱin: ⋀n. ℱ n ∈ F and ℱlim: (λn. norm (deriv (ℱ n) 0))
⟶ l
  proof −
    have ∃f. f ∈ F ∧ |norm (deriv f 0) − l| < 1 / (Suc n) for n
    proof −
      obtain f where f ∈ F and f: l < norm (deriv f 0) + 1/(Suc n)
        using cSup_least [OF imF_ne, of l − 1/(Suc n)] by (fastforce simp add:
l_def)
      then have |norm (deriv f 0) − l| < 1 / (Suc n)
        by (fastforce simp add: abs_if not_less eql)
      with ‹f ∈ F› show ?thesis
        by blast
    qed
    then obtain ℱ where fF: ⋀n. (ℱ n) ∈ F
      and fless: ⋀n. |norm (deriv (ℱ n) 0) − l| < 1 / (Suc n)
      by metis
    have (λn. norm (deriv (ℱ n) 0)) ⟶ l
    proof (rule metric_LIMSEQ_I)
      fix e::real
      assume e > 0
      then obtain N::nat where N: e > 1/(Suc N)
        using nat_approx_posE by blast
      show ∃N. ∀n≥N. dist (norm (deriv (ℱ n) 0)) l < e
      proof (intro exI allI impI)
        fix n assume N ≤ n
        have dist (norm (deriv (ℱ n) 0)) l < 1 / (Suc n)
          using fless by (simp add: dist_norm)
        also have ... < e
          using N ‹N ≤ n› inverse_of_nat_le le_less_trans by blast
        finally show dist (norm (deriv (ℱ n) 0)) l < e .
      qed
    qed
    with fF show ?thesis
      using that by blast
  qed
  have ⋀K. ⟦compact K; K ⊆ S⟧ ⟹ ∃B. ∀h∈F. ∀z∈K. norm (h z) ≤ B
    by (rule_tac x=1 in exI) (force simp: F_def)
  moreover have range ℱ ⊆ F
    using ‹⋀n. ℱ n ∈ F› by blast
  ultimately obtain f and r :: nat ⇒ nat
    where holf: f holomorphic_on S and r: strict_mono r
      and limf: ⋀x. x ∈ S ⟹ (λn. ℱ (r n) x) ⟶ f x
```

```
      and ulimf: ⋀K. ⟦compact K; K ⊆ S⟧ ⟹ uniform_limit K (ℱ ∘ r) f
sequentially
    using Montel [of S F ℱ, OF ‹open S› holF] by auto+
  have der: ⋀n x. x ∈ S ⟹ ((ℱ ∘ r) n has_field_derivative ((λn. deriv (ℱ n))
∘ r) n x) (at x)
    using ‹⋀n. ℱ n ∈ F› ‹open S› holF holomorphic_derivI by fastforce
  have ulim: ⋀x. x ∈ S ⟹ ∃d>0. cball x d ⊆ S ∧ uniform_limit (cball x d)
(ℱ ∘ r) f sequentially
    by (meson ulimf ‹open S› compact_cball open_contains_cball)
  obtain f' :: complex⇒complex where f': (f has_field_derivative f' 0) (at 0)
    and tof'0: (λn. ((λn. deriv (ℱ n)) ∘ r) n 0) ⟶ f' 0
    using has_complex_derivative_uniform_sequence [OF ‹open S› der ulim] ‹0
∈ S› by metis
  then have derf0: deriv f 0 = f' 0
    by (simp add: DERIV_imp_deriv)
  have f field_differentiable (at 0)
    using field_differentiable_def f' by blast
  have (λx. (norm (deriv (ℱ (r x)) 0))) ⟶ norm (deriv f 0)
   using isCont_tendsto_compose [OF continuous_norm [OF continuous_ident]
tof'0] derf0 by auto
  with LIMSEQ_subseq_LIMSEQ [OF ℱlim r] have no_df0: norm(deriv f 0)
= l
    by (force simp: o_def intro: tendsto_unique)
  have nonconstf: ¬ f constant_on S
  proof −
    have False if ⋀x. x ∈ S ⟹ f x = c for c
    proof −
      have deriv f 0 = 0
     by (metis that ‹open S› ‹0 ∈ S› DERIV_imp_deriv [OF has_field_derivative_transform_within_open
[OF DERIV_const]])
      with no_df0 have l = 0
        by auto
      with eql [OF _ idF] show False by auto
    qed
    then show ?thesis
      by (meson constant_on_def)
  qed
  show ?thesis
  proof
    show f ∈ F
      unfolding F_def
    proof (intro CollectI conjI holf)
      have norm(f z) ≤ 1 if z ∈ S for z
      proof (intro Lim_norm_ubound [OF _ limf] always_eventually allI that)
        fix n
        have ℱ (r n) ∈ F
          by (simp add: ℱin)
        then show norm (ℱ (r n) z) ≤ 1
          using that by (auto simp: F_def)
```

```
      qed simp
      then have flless1: norm(f z) < 1 if z ∈ S for z
       using maximum_modulus_principle [OF holf ‹open S› ‹connected S› ‹open
S›] nonconstf that
        by fastforce
      then show f ‘ S ⊆ ball 0 1
        by auto
      have (λn. F (r n) 0) ⟶ 0
        using Fin by (auto simp: F_def)
      then show f 0 = 0
       using tendsto_unique [OF _ limf ] ‹0 ∈ S› trivial_limit_sequentially by
blast
      show inj_on f S
      proof (rule Hurwitz_injective [OF ‹open S› ‹connected S› _ holf])
        show ⋀n. (F ∘ r) n holomorphic_on S
          by (simp add: Fin holF)
       show ⋀K. ⟦compact K; K ⊆ S⟧ ⟹ uniform_limit K(F ∘ r) f sequentially
          by (metis ulimf)
        show ¬ f constant_on S
          using nonconstf by auto
        show ⋀n. inj_on ((F ∘ r) n) S
          using Fin by (auto simp: F_def)
      qed
    qed
    show ⋀h. h ∈ F ⟹ norm (deriv h 0) ≤ norm (deriv f 0)
      by (metis eql le_cases no_df0)
  qed
 qed
 have holf: f holomorphic_on S and injf: inj_on f S and f01: f ‘ S ⊆ ball 0 1
  using ‹f ∈ F› by (auto simp: F_def)
 obtain g where holg: g holomorphic_on (f ‘ S)
           and derg: ⋀z. z ∈ S ⟹ deriv f z ∗ deriv g (f z) = 1
           and gf: ⋀z. z ∈ S ⟹ g(f z) = z
  using holomorphic_has_inverse [OF holf ‹open S› injf] by metis
 have ball 0 1 ⊆ f ‘ S
 proof
  fix a::complex
  assume a: a ∈ ball 0 1
  have False if ⋀x. x ∈ S ⟹ f x ≠ a
  proof −
   obtain h k where h a = 0
    and holh: h holomorphic_on ball 0 1 and h01: h ‘ ball 0 1 ⊆ ball 0 1
    and holk: k holomorphic_on ball 0 1 and k01: k ‘ ball 0 1 ⊆ ball 0 1
    and hk: ⋀z. z ∈ ball 0 1 ⟹ h (k z) = z
    and kh: ⋀z. z ∈ ball 0 1 ⟹ k (h z) = z
    using ball_biholomorphism_exists [OF a] by blast
   have nf1: ⋀z. z ∈ S ⟹ norm(f z) < 1
    using ‹f ∈ F› by (auto simp: F_def)
   have 1: h ∘ f holomorphic_on S
```

using *F_def* ‹*f* ∈ *F*› *holh holomorphic_on_compose holomorphic_on_subset*
by *blast*
have *2*: ⋀*z*. *z* ∈ *S* ⟹ (*h* ∘ *f*) *z* ≠ *0*
by (*metis* ‹*h a* = *0*› *a comp_eq_dest_lhs nf1 kh mem_ball_0 that*)
have *3*: *inj_on* (*h* ∘ *f*) *S*
by (*metis* (*no_types*, *lifting*) *F_def* ‹*f* ∈ *F*› *comp_inj_on inj_on_inverseI*
injf kh mem_Collect_eq subset_inj_on)
obtain *ψ* **where** *holψ*: *ψ holomorphic_on* ((*h* ∘ *f*) ‘ *S*)
and *ψ2*: ⋀*z*. *z* ∈ *S* ⟹ *ψ*(*h* (*f z*)) ^ *2* = *h*(*f z*)
proof (*rule exE* [*OF prev* [*OF 1 2 3*]], *safe*)
fix *ϑ*
assume *holϑ*: *ϑ holomorphic_on S* **and** *ϑ2*: (∀*z*∈*S*. (*h* ∘ *f*) *z* = (*ϑ z*)^2)
show *thesis*
proof
show (*ϑ* ∘ *g* ∘ *k*) *holomorphic_on* (*h* ∘ *f*) ‘ *S*
proof (*intro holomorphic_on_compose*)
show *k holomorphic_on* (*h* ∘ *f*) ‘ *S*
apply (*rule holomorphic_on_subset* [*OF holk*])
using *f01 h01* **by** *force*
show *g holomorphic_on k* ‘ (*h* ∘ *f*) ‘ *S*
apply (*rule holomorphic_on_subset* [*OF holg*])
by (*auto simp*: *kh nf1*)
show *ϑ holomorphic_on g* ‘ *k* ‘ (*h* ∘ *f*) ‘ *S*
apply (*rule holomorphic_on_subset* [*OF holϑ*])
by (*auto simp*: *gf kh nf1*)
qed
show ((*ϑ* ∘ *g* ∘ *k*) (*h* (*f z*)))^2 = *h* (*f z*) **if** *z* ∈ *S* **for** *z*
proof −
have *f z* ∈ *ball 0 1*
by (*simp add*: *nf1 that*)
then have (*ϑ* (*g* (*k* (*h* (*f z*)))))^2 = (*ϑ* (*g* (*f z*)))^2
by (*metis kh*)
also have ... = *h* (*f z*)
using *ϑ2 gf that* **by** *auto*
finally show *?thesis*
by (*simp add*: *o_def*)
qed
qed
qed
have *normψ1*: *norm*(*ψ* (*h* (*f z*))) < *1* **if** *z* ∈ *S* **for** *z*
proof −
have *norm* (*ψ* (*h* (*f z*)) ^ *2*) < *1*
by (*metis* (*no_types*) *that DIM_complex ψ2 h01 image_subset_iff*
mem_ball_0 nf1)
then show *?thesis*
by (*metis le_less_trans mult_less_cancel_left2 norm_ge_zero norm_power*
not_le power2_eq_square)
qed
then have *ψ01*: *ψ* (*h* (*f 0*)) ∈ *ball 0 1*

```
    by (simp add: ‹0 ∈ S›)
  obtain p q where p0: p (ψ (h (f 0))) = 0
    and holp: p holomorphic_on ball 0 1 and p01: p ‘ ball 0 1 ⊆ ball 0 1
    and holq: q holomorphic_on ball 0 1 and q01: q ‘ ball 0 1 ⊆ ball 0 1
    and pq: ⋀z. z ∈ ball 0 1 ⟹ p (q z) = z
    and qp: ⋀z. z ∈ ball 0 1 ⟹ q (p z) = z
    using ball_biholomorphism_exists [OF ψ01] by metis
  have p ∘ ψ ∘ h ∘ f ∈ F
    unfolding F_def
  proof (intro CollectI conjI holf)
    show p ∘ ψ ∘ h ∘ f holomorphic_on S
    proof (intro holomorphic_on_compose holf)
      show h holomorphic_on f ‘ S
        apply (rule holomorphic_on_subset [OF holh])
        using f01 by force
      show ψ holomorphic_on h ‘ f ‘ S
        apply (rule holomorphic_on_subset [OF holψ])
        by auto
      show p holomorphic_on ψ ‘ h ‘ f ‘ S
        apply (rule holomorphic_on_subset [OF holp])
        by (auto simp: normψ1)
    qed
    show (p ∘ ψ ∘ h ∘ f) ‘ S ⊆ ball 0 1
      apply clarsimp
      by (meson normψ1 p01 image_subset_iff mem_ball_0)
    show (p ∘ ψ ∘ h ∘ f) 0 = 0
      by (simp add: ‹p (ψ (h (f 0))) = 0›)
    show inj_on (p ∘ ψ ∘ h ∘ f) S
      unfolding inj_on_def o_def
      by (metis ψ2 dist_0_norm gf kh mem_ball nf1 normψ1 qp)
  qed
  then have le_norm_df0: norm (deriv (p ∘ ψ ∘ h ∘ f) 0) ≤ norm (deriv f 0)
    by (rule normf)
  have 1: k ∘ power2 ∘ q holomorphic_on ball 0 1
  proof (intro holomorphic_on_compose holq)
    show power2 holomorphic_on q ‘ ball 0 1
      using holomorphic_on_subset holomorphic_on_power
      by (blast intro: holomorphic_on_ident)
    show k holomorphic_on power2 ‘ q ‘ ball 0 1
      apply (rule holomorphic_on_subset [OF holk])
      using q01 by (auto simp: norm_power abs_square_less_1)
  qed
  have 2: (k ∘ power2 ∘ q) 0 = 0
    using p0 F_def ‹f ∈ F› ψ01 ψ2 ‹0 ∈ S› kh qp by force
  have 3: norm ((k ∘ power2 ∘ q) z) < 1 if norm z < 1 for z
  proof −
    have norm ((power2 ∘ q) z) < 1
      using that q01 by (force simp: norm_power abs_square_less_1)
    with k01 show ?thesis
```

```
      by fastforce
    qed
    have False if c: ∀z. norm z < 1 ⟶ (k ∘ power2 ∘ q) z = c * z and norm
c = 1 for c
    proof −
      have c ≠ 0 using that by auto
      have norm (p(1/2)) < 1 norm (p(−1/2)) < 1
        using p01 by force+
        then have (k ∘ power2 ∘ q) (p(1/2)) = c * p(1/2) (k ∘ power2 ∘ q)
(p(−1/2)) = c * p(−1/2)
        using c by force+
      then have p (1/2) = p (− (1/2))
        by (auto simp: ‹c ≠ 0› qp o_def)
      then have q (p (1/2)) = q (p (− (1/2)))
        by simp
      then have 1/2 = − (1/2::complex)
        by (auto simp: qp)
      then show False
        by simp
    qed
    moreover
    have False if norm (deriv (k ∘ power2 ∘ q) 0) ≠ 1 norm (deriv (k ∘ power2
∘ q) 0) ≤ 1
      and le: ⋀ξ. norm ξ < 1 ⟹ norm ((k ∘ power2 ∘ q) ξ) ≤ norm ξ
    proof −
      have norm (deriv (k ∘ power2 ∘ q) 0) < 1
        using that by simp
      moreover have eq: deriv f 0 = deriv (k ∘ (λz. z ^ 2) ∘ q) 0 * deriv (p ∘
ψ ∘ h ∘ f) 0
    proof (intro DERIV_imp_deriv has_field_derivative_transform_within_open
[OF DERIV_chain])
        show (k ∘ power2 ∘ q has_field_derivative deriv (k ∘ power2 ∘ q) 0) (at
((p ∘ ψ ∘ h ∘ f) 0))
          using 1 holomorphic_derivI p0 by auto
        show (p ∘ ψ ∘ h ∘ f has_field_derivative deriv (p ∘ ψ ∘ h ∘ f) 0) (at 0)
          using ‹p ∘ ψ ∘ h ∘ f ∈ F› ‹open S› ‹0 ∈ S› holF holomorphic_derivI
by blast
        show ⋀x. x ∈ S ⟹ (k ∘ power2 ∘ q ∘ (p ∘ ψ ∘ h ∘ f)) x = f x
          using ψ2 f01 kh normψ1 qp by auto
      qed (use assms in simp_all)
      ultimately have cmod (deriv (p ∘ ψ ∘ h ∘ f) 0) ≤ 0
        using le_norm_df0
     by (metis linorder_not_le mult.commute mult_less_cancel_left2 norm_mult)
      moreover have 1 ≤ norm (deriv f 0)
        using normf [of id] by (simp add: idF)
      ultimately show False
        by (simp add: eq)
    qed
    ultimately show ?thesis
```

```
      using Schwarz_Lemma [OF 1 2 3] norm_one by blast
    qed
    then show a ∈ f ‘ S
      by blast
  qed
  then have f ‘ S = ball 0 1
    using F_def ‹f ∈ F› by blast
  then show ?thesis
    apply (rule_tac x=f in exI)
    apply (rule_tac x=g in exI)
    using holf holg derg gf by safe force+
qed


locale SC_Chain =
  fixes S :: complex set
  assumes openS: open S
begin

lemma winding_number_zero:
  assumes simply_connected S
  shows connected S ∧
        (∀γ z. path γ ∧ path_image γ ⊆ S ∧
               pathfinish γ = pathstart γ ∧ z ∉ S ⟶ winding_number γ z = 0)
  using assms
 by (auto simp: simply_connected_imp_connected simply_connected_imp_winding_number_zero)

lemma contour_integral_zero:
  assumes valid_path g path_image g ⊆ S pathfinish g = pathstart g f holomor-
phic_on S
        ⋀γ z. ⟦path γ; path_image γ ⊆ S; pathfinish γ = pathstart γ; z ∉ S⟧ ⟹
winding_number γ z = 0
  shows (f has_contour_integral 0) g
  using assms by (meson Cauchy_theorem_global openS valid_path_imp_path)

lemma global_primitive:
  assumes connected S and holf: f holomorphic_on S
  and prev: ⋀γ f. ⟦valid_path γ; path_image γ ⊆ S; pathfinish γ = pathstart γ; f
holomorphic_on S⟧ ⟹ (f has_contour_integral 0) γ
  shows ∃h. ∀z ∈ S. (h has_field_derivative f z) (at z)
proof (cases S = {})
case True then show ?thesis
    by simp
next
  case False
  then obtain a where a ∈ S
    by blast
  show ?thesis
  proof (intro exI ballI)
```

fix x **assume** $x \in S$
then obtain d **where** $d > 0$ **and** d: $cball\ x\ d \subseteq S$
using *openS open_contains_cball_eq* **by** *blast*
let $?g = \lambda z.\ (SOME\ g.\ polynomial\_function\ g \wedge path\_image\ g \subseteq S \wedge pathstart\ g = a \wedge pathfinish\ g = z)$
show $((\lambda z.\ contour\_integral\ (?g\ z)\ f)\ has\_field\_derivative\ f\ x)\ (at\ x)$
proof (*simp add*: *has_field_derivative_def has_derivative_at2 bounded_linear_mult_right*, *rule Lim_transform*)
show $(\lambda y.\ inverse(norm(y - x)) *_R (contour\_integral(linepath\ x\ y)\ f - f\ x * (y - x))) -x\to 0$
proof (*clarsimp simp add*: *Lim_at*)
fix e::*real* **assume** $e > 0$
moreover have $continuous\ (at\ x)\ f$
using *openS* ‹$x \in S$› *holf continuous_on_eq_continuous_at holomorphic_on_imp_continuous_on* **by** *auto*
ultimately obtain $d1$ **where** $d1 > 0$
and $d1$: $\bigwedge x'.\ dist\ x'\ x < d1 \Longrightarrow dist\ (f\ x')\ (f\ x) < e/2$
unfolding *continuous_at_eps_delta*
by (*metis less_divide_eq_numeral1*(*1*) *mult_zero_left*)
obtain $d2$ **where** $d2 > 0$ **and** $d2$: $ball\ x\ d2 \subseteq S$
using *openS* ‹$x \in S$› *open_contains_ball_eq* **by** *blast*
have $inverse\ (norm\ (y - x)) * norm\ (contour\_integral\ (linepath\ x\ y)\ f - f\ x * (y - x)) < e$
if $0 < d1\ 0 < d2\ y \neq x\ dist\ y\ x < d1\ dist\ y\ x < d2$ **for** y
proof −
have $f\ contour\_integrable\_on\ linepath\ x\ y$
proof (*rule contour_integrable_continuous_linepath* [*OF continuous_on_subset*])
show $continuous\_on\ S\ f$
by (*simp add*: *holf holomorphic_on_imp_continuous_on*)
have $closed\_segment\ x\ y \subseteq ball\ x\ d2$
by (*meson dist_commute_lessI dist_in_closed_segment le_less_trans mem_ball subsetI that*(*5*))
with $d2$ **show** $closed\_segment\ x\ y \subseteq S$
by *blast*
qed
then obtain z **where** z: $(f\ has\_contour\_integral\ z)\ (linepath\ x\ y)$
by (*force simp*: *contour_integrable_on_def*)
have *con*: $((\lambda w.\ f\ x)\ has\_contour\_integral\ f\ x * (y - x))\ (linepath\ x\ y)$
using *has_contour_integral_const_linepath* [*of f x y x*] **by** *metis*
have $norm\ (z - f\ x * (y - x)) \leq (e/2) * norm\ (y - x)$
proof (*rule has_contour_integral_bound_linepath*)
show $((\lambda w.\ f\ w - f\ x)\ has\_contour\_integral\ z - f\ x * (y - x))\ (linepath\ x\ y)$
by (*rule has_contour_integral_diff* [*OF z con*])
show $\bigwedge w.\ w \in closed\_segment\ x\ y \Longrightarrow norm\ (f\ w - f\ x) \leq e/2$
by (*metis d1 dist_norm less_le_trans not_less not_less_iff_gr_or_eq segment_bound1 that*(*4*))
qed (*use* ‹$e > 0$› **in** *auto*)

```
        with ‹e > 0› have inverse (norm (y − x)) ∗ norm (z − f x ∗ (y − x))
≤ e/2
          by (simp add: field_split_simps)
        also have ... < e
          using ‹e > 0› by simp
        finally show ?thesis
          by (simp add: contour_integral_unique [OF z])
      qed
      with ‹d1 > 0› ‹d2 > 0›
      show ∃ d>0. ∀ z. z ≠ x ∧ dist z x < d ⟶
            inverse (norm (z − x)) ∗ norm (contour_integral (linepath x z) f −
f x ∗ (z − x)) < e
        by (rule_tac x=min d1 d2 in exI) auto
    qed
  next
    have ∗: (1 / norm (y − x)) ∗R (contour_integral (?g y) f −
          (contour_integral (?g x) f + f x ∗ (y − x))) =
          (contour_integral (linepath x y) f − f x ∗ (y − x)) /R norm (y − x)
      if 0 < d y ≠ x and yx: dist y x < d for y
    proof −
      have y ∈ S
        by (metis subsetD d dist_commute less_eq_real_def mem_cball yx)
      have gxy: polynomial_function (?g x) ∧ path_image (?g x) ⊆ S ∧ pathstart
(?g x) = a ∧ pathfinish (?g x) = x
            polynomial_function (?g y) ∧ path_image (?g y) ⊆ S ∧ pathstart
(?g y) = a ∧ pathfinish (?g y) = y
        using someI_ex [OF connected_open_polynomial_connected [OF openS
‹connected S› ‹a ∈ S›]] ‹x ∈ S› ‹y ∈ S›
        by meson+
      then have vp: valid_path (?g x) valid_path (?g y)
        by (simp_all add: valid_path_polynomial_function)
        have f0: (f has_contour_integral 0) ((?g x) +++ linepath x y +++
reversepath (?g y))
      proof (rule prev)
        show valid_path ((?g x) +++ linepath x y +++ reversepath (?g y))
          using gxy vp by (auto simp: valid_path_join)
        have closed_segment x y ⊆ cball x d
          using yx by (auto simp: dist_commute dest!: dist_in_closed_segment)
        then have closed_segment x y ⊆ S
          using d by blast
        then show path_image ((?g x) +++ linepath x y +++ reversepath (?g
y)) ⊆ S
          using gxy by (auto simp: path_image_join)
      qed (use gxy holf in auto)
      then have fintxy: f contour_integrable_on linepath x y
      by (metis (no_types, lifting) contour_integrable_joinD1 contour_integrable_joinD2
gxy(2) has_contour_integral_integrable pathfinish_linepath pathstart_reversepath
valid_path_imp_reverse valid_path_join valid_path_linepath vp(2))
      have fintgx: f contour_integrable_on (?g x) f contour_integrable_on (?g y)
```

```
      using openS contour_integrable_holomorphic_simple gxy holf vp by blast+
      show ?thesis
        apply (clarsimp simp add: divide_simps)
        using contour_integral_unique [OF f0]
      apply (simp add: fintxy gxy contour_integrable_reversepath contour_integral_reversepath
fintgx vp)
        apply (simp add: algebra_simps)
        done
    qed
    show (λz. (1 / norm (z − x)) *R
              (contour_integral (?g z) f − (contour_integral (?g x) f + f x * (z
− x))) −
              (contour_integral (linepath x z) f − f x * (z − x)) /R norm (z − x))
         −x→ 0
      apply (rule tendsto_eventually)
      apply (simp add: eventually_at)
      apply (rule_tac x=d in exI)
      using ‹d > 0› * by simp
  qed
 qed
qed

lemma holomorphic_log:
 assumes connected S and holf: f holomorphic_on S and nz: ⋀z. z ∈ S ⟹ f z
≠ 0
 and prev: ⋀f. f holomorphic_on S ⟹ ∃h. ∀z ∈ S. (h has_field_derivative f z)
(at z)
 shows ∃g. g holomorphic_on S ∧ (∀z ∈ S. f z = exp(g z))
proof −
 have (λz. deriv f z / f z) holomorphic_on S
   by (simp add: openS holf holomorphic_deriv holomorphic_on_divide nz)
 then obtain g where g: ⋀z. z ∈ S ⟹ (g has_field_derivative deriv f z / f z)
(at z)
   using prev [of λz. deriv f z / f z] by metis
 have hfd: ⋀x. x ∈ S ⟹ ((λz. exp (g z) / f z) has_field_derivative 0) (at x)
   apply (rule derivative_eq_intros g| simp)+
     apply (subst DERIV_deriv_iff_field_differentiable)
   using openS holf holomorphic_on_imp_differentiable_at nz apply auto
   done
 obtain c where c: ⋀x. x ∈ S ⟹ exp (g x) / f x = c
 proof (rule DERIV_zero_connected_constant[OF ‹connected S› openS finite.emptyI])
   show continuous_on S (λz. exp (g z) / f z)
    by (metis (full_types) openS g continuous_on_divide continuous_on_exp holf
holomorphic_on_imp_continuous_on holomorphic_on_open nz)
   then show ∀x∈S − {}. ((λz. exp (g z) / f z) has_field_derivative 0) (at x)
    using hfd by (blast intro: DERIV_zero_connected_constant [OF ‹connected
S› openS finite.emptyI, of λz. exp(g z) / f z])
 qed auto
 show ?thesis
```

```
  proof (intro exI ballI conjI)
    show (λz. Ln(inverse c) + g z) holomorphic_on S
      apply (intro holomorphic_intros)
      using openS g holomorphic_on_open by blast
    fix z :: complex
    assume z ∈ S
    then have exp (g z) / c = f z
    by (metis c divide_divide_eq_right exp_not_eq_zero nonzero_mult_div_cancel_left)
    moreover have 1 / c ≠ 0
      using ‹z ∈ S› c nz by fastforce
    ultimately show f z = exp (Ln (inverse c) + g z)
      by (simp add: exp_add inverse_eq_divide)
  qed
qed

lemma holomorphic_sqrt:
  assumes holf: f holomorphic_on S and nz: ⋀z. z ∈ S ⟹ f z ≠ 0
  and prev: ⋀f. ⟦f holomorphic_on S; ∀ z ∈ S. f z ≠ 0⟧ ⟹ ∃ g. g holomorphic_on
S ∧ (∀ z ∈ S. f z = exp(g z))
  shows ∃ g. g holomorphic_on S ∧ (∀ z ∈ S. f z = (g z)²)
proof −
  obtain g where holg: g holomorphic_on S and g: ⋀z. z ∈ S ⟹ f z = exp (g
z)
    using prev [of f] holf nz by metis
  show ?thesis
  proof (intro exI ballI conjI)
    show (λz. exp(g z/2)) holomorphic_on S
      by (intro holomorphic_intros) (auto simp: holg)
    show ⋀z. z ∈ S ⟹ f z = (exp (g z/2))²
    by (metis (no_types) g exp_double nonzero_mult_div_cancel_left times_divide_eq_right
zero_neq_numeral)
  qed
qed

lemma biholomorphic_to_disc:
  assumes connected S and S: S ≠ {} S ≠ UNIV
  and prev: ⋀f. ⟦f holomorphic_on S; ∀ z ∈ S. f z ≠ 0⟧ ⟹ ∃ g. g holomorphic_on
S ∧ (∀ z ∈ S. f z = (g z)²)
  shows ∃ f g. f holomorphic_on S ∧ g holomorphic_on ball 0 1 ∧
                (∀ z ∈ S. f z ∈ ball 0 1 ∧ g(f z) = z) ∧
                (∀ z ∈ ball 0 1. g z ∈ S ∧ f(g z) = z)
proof −
  obtain a b where a ∈ S b ∉ S
    using S by blast
  then obtain δ where δ > 0 and δ: ball a δ ⊆ S
    using openS openE by blast
  obtain g where holg: g holomorphic_on S and eqg: ⋀z. z ∈ S ⟹ z − b = (g
z)²
  proof (rule exE [OF prev [of λz. z − b]])
```

```
    show (λz. z − b) holomorphic_on S
      by (intro holomorphic_intros)
  qed (use ‹b ∉ S› in auto)
  have ¬ g constant_on S
  proof −
    have (a + δ/2) ∈ ball a δ a + (δ/2) ≠ a
      using ‹δ > 0› by (simp_all add: dist_norm)
    then show ?thesis
      unfolding constant_on_def
      using eqg [of a] eqg [of a + δ/2] ‹a ∈ S› δ
      by (metis diff_add_cancel subset_eq)
  qed
  then have open (g ‘ ball a δ)
    using open_mapping_thm [of g S ball a δ, OF holg openS ‹connected S›] δ by
blast
  then obtain r where r > 0 and r: ball (g a) r ⊆ (g ‘ ball a δ)
    by (metis ‹0 < δ› centre_in_ball imageI openE)
  have g_not_r: g z ∉ ball (−(g a)) r if z ∈ S for z
  proof
    assume g z ∈ ball (−(g a)) r
    then have − g z ∈ ball (g a) r
      by (metis add.inverse_inverse dist_minus mem_ball)
    with r have − g z ∈ (g ‘ ball a δ)
      by blast
    then obtain w where w: − g z = g w dist a w < δ
      by auto
    then have w ∈ ball a δ
      by simp
    then have w ∈ S
      using δ by blast
    then have w = z
      by (metis diff_add_cancel eqg power_minus_Bit0 that w(1))
    then have g z = 0
      using ‹− g z = g w› by auto
    with eqg [OF that] have z = b
      by auto
    with that ‹b ∉ S› show False
      by simp
  qed
  then have nz: ⋀z. z ∈ S ⟹ g z + g a ≠ 0
    by (metis ‹0 < r› add.commute add_diff_cancel_left' centre_in_ball diff_0)
  let ?f = λz. (r/3) / (g z + g a) − (r/3) / (g a + g a)
  obtain h where holh: h holomorphic_on S and h a = 0 and h01: h ‘ S ⊆ ball
0 1 and inj_on h S
  proof
    show ?f holomorphic_on S
      by (intro holomorphic_intros holg nz)
   have 3: ⟦norm x ≤ 1/3; norm y ≤ 1/3⟧ ⟹ norm(x − y) < 1 for x y::complex
      using norm_triangle_ineq4 [of x y] by simp
```

```
    have norm ((r/3) / (g z + g a) - (r/3) / (g a + g a)) < 1 if z ∈ S for z
      apply (rule 3)
      unfolding norm_divide
      using ‹r > 0› g_not_r [OF ‹z ∈ S›] g_not_r [OF ‹a ∈ S›]
      by (simp_all add: field_split_simps dist_commute dist_norm)
  then show ?f ‘ S ⊆ ball 0 1
    by auto
    show inj_on ?f S
      using ‹r > 0› eqg apply (clarsimp simp: inj_on_def)
      by (metis diff_add_cancel)
  qed auto
  obtain k where holk: k holomorphic_on (h ‘ S)
            and derk: ⋀z. z ∈ S ⟹ deriv h z * deriv k (h z) = 1
            and kh: ⋀z. z ∈ S ⟹ k(h z) = z
    using holomorphic_has_inverse [OF holh openS ‹inj_on h S›] by metis

  have 1: open (h ‘ S)
    by (simp add: ‹inj_on h S› holh openS open_mapping_thm3)
  have 2: connected (h ‘ S)
   by (simp add: connected_continuous_image ‹connected S› holh holomorphic_on_imp_continuous_on)
  have 3: 0 ∈ h ‘ S
    using ‹a ∈ S› ‹h a = 0› by auto
  have 4: ∃ g. g holomorphic_on h ‘ S ∧ (∀ z∈h ‘ S. f z = (g z)²)
    if holf: f holomorphic_on h ‘ S and nz: ⋀z. z ∈ h ‘ S ⟹ f z ≠ 0 inj_on f (h
‘ S) for f
  proof −
    obtain g where holg: g holomorphic_on S and eqg: ⋀z. z ∈ S ⟹ (f ∘ h) z
= (g z)²
    proof −
      have f ∘ h holomorphic_on S
        by (simp add: holh holomorphic_on_compose holf)
      moreover have ∀ z∈S. (f ∘ h) z ≠ 0
        by (simp add: nz)
      ultimately show thesis
        using prev that by blast
    qed
    show ?thesis
    proof (intro exI conjI)
      show g ∘ k holomorphic_on h ‘ S
      proof −
        have k ‘ h ‘ S ⊆ S
          by (simp add: ‹⋀z. z ∈ S ⟹ k (h z) = z› image_subset_iff)
        then show ?thesis
          by (meson holg holk holomorphic_on_compose holomorphic_on_subset)
      qed
      show ∀ z∈h ‘ S. f z = ((g ∘ k) z)²
        using eqg kh by auto
    qed
  qed
```

```
  obtain f g where f: f holomorphic_on h ‘ S and g: g holomorphic_on ball 0 1
      and gf: ∀ z∈h ‘ S. f z ∈ ball 0 1 ∧ g (f z) = z and fg:∀ z∈ball 0 1. g z ∈ h
‘ S ∧ f (g z) = z
    using biholomorphic_to_disc_aux [OF 1 2 3 h01 4] by blast
  show ?thesis
  proof (intro exI conjI)
    show f ∘ h holomorphic_on S
      by (simp add: f holh holomorphic_on_compose)
    show k ∘ g holomorphic_on ball 0 1
        by (metis holomorphic_on_subset image_subset_iff fg holk g holomor-
phic_on_compose)
  qed (use fg gf kh in auto)
qed

lemma homeomorphic_to_disc:
  assumes S: S ≠ {}
    and prev: S = UNIV ∨
            (∃f g. f holomorphic_on S ∧ g holomorphic_on ball 0 1 ∧
                 (∀ z ∈ S. f z ∈ ball 0 1 ∧ g(f z) = z) ∧
                 (∀ z ∈ ball 0 1. g z ∈ S ∧ f(g z) = z)) (is _ ∨ ?P)
  shows S homeomorphic ball (0::complex) 1
  using prev
proof
  assume S = UNIV then show ?thesis
    using homeomorphic_ball01_UNIV homeomorphic_sym by blast
next
  assume ?P
  then show ?thesis
    unfolding homeomorphic_minimal
    using holomorphic_on_imp_continuous_on by blast
qed

lemma homeomorphic_to_disc_imp_simply_connected:
  assumes S = {} ∨ S homeomorphic ball (0::complex) 1
  shows simply_connected S
 using assms homeomorphic_simply_connected_eq convex_imp_simply_connected
by auto

end

proposition
  assumes open S
  shows simply_connected_eq_winding_number_zero:
        simply_connected S ⟷
          connected S ∧
          (∀ g z. path g ∧ path_image g ⊆ S ∧
                pathfinish g = pathstart g ∧ (z ∉ S)
                ⟶ winding_number g z = 0) (is ?wn0)
    and simply_connected_eq_contour_integral_zero:
```

```
        simply_connected S ⟷
          connected S ∧
          (∀ g f. valid_path g ∧ path_image g ⊆ S ∧
                pathfinish g = pathstart g ∧ f holomorphic_on S
              ⟶ (f has_contour_integral 0) g) (is ?ci0)
    and simply_connected_eq_global_primitive:
        simply_connected S ⟷
          connected S ∧
          (∀ f. f holomorphic_on S ⟶
              (∃ h. ∀ z. z ∈ S ⟶ (h has_field_derivative f z) (at z))) (is ?gp)
    and simply_connected_eq_holomorphic_log:
        simply_connected S ⟷
          connected S ∧
          (∀ f. f holomorphic_on S ∧ (∀ z ∈ S. f z ≠ 0)
              ⟶ (∃ g. g holomorphic_on S ∧ (∀ z ∈ S. f z = exp(g z)))) (is ?log)
    and simply_connected_eq_holomorphic_sqrt:
        simply_connected S ⟷
          connected S ∧
          (∀ f. f holomorphic_on S ∧ (∀ z ∈ S. f z ≠ 0)
              ⟶ (∃ g. g holomorphic_on S ∧ (∀ z ∈ S.  f z = (g z)^2))) (is ?sqrt)
    and simply_connected_eq_biholomorphic_to_disc:
        simply_connected S ⟷
          S = {} ∨ S = UNIV ∨
          (∃ f g. f holomorphic_on S ∧ g holomorphic_on ball 0 1 ∧
              (∀ z ∈ S. f z ∈ ball 0 1 ∧ g(f z) = z) ∧
              (∀ z ∈ ball 0 1. g z ∈ S ∧ f(g z) = z)) (is ?bih)
    and simply_connected_eq_homeomorphic_to_disc:
        simply_connected S ⟷ S = {} ∨ S homeomorphic ball (0::complex) 1
(is ?disc)
proof −
  interpret SC_Chain
    using assms by (simp add: SC_Chain_def)
  have ?wn0 ∧ ?ci0 ∧ ?gp ∧ ?log ∧ ?sqrt ∧ ?bih ∧ ?disc
proof −
  have ∗: ⟦α ⟹ β; β ⟹ γ; γ ⟹ δ; δ ⟹ ζ; ζ ⟹ η; η ⟹ ϑ; ϑ ⟹ ξ; ξ ⟹
α⟧
      ⟹ (α ⟷ β) ∧ (α ⟷ γ) ∧ (α ⟷ δ) ∧ (α ⟷ ζ) ∧
         (α ⟷ η) ∧ (α ⟷ ϑ) ∧ (α ⟷ ξ) for α β γ δ ζ η ϑ ξ
    by blast
  show ?thesis
    apply (rule ∗)
    using winding_number_zero apply metis
    using contour_integral_zero apply metis
    using global_primitive apply metis
    using holomorphic_log apply metis
    using holomorphic_sqrt apply simp
    using biholomorphic_to_disc apply blast
    using homeomorphic_to_disc apply blast
    using homeomorphic_to_disc_imp_simply_connected apply blast
```

```
    done
qed
 then show ?wn0 ?ci0 ?gp ?log ?sqrt ?bih ?disc
   by safe
qed

corollary contractible_eq_simply_connected_2d:
 fixes S :: complex set
 shows open S ⟹ (contractible S ⟷ simply_connected S)
 apply safe
  apply (simp add: contractible_imp_simply_connected)
using convex_imp_contractible homeomorphic_contractible_eq simply_connected_eq_homeomorphic_to_di
by auto
```

8.3 A further chain of equivalences about components of the complement of a simply connected set

(following 1.35 in Burckel'S book)

```
context SC_Chain
begin

lemma frontier_properties:
 assumes simply_connected S
 shows if bounded S then connected(frontier S)
       else ∀ C ∈ components(frontier S). ¬ bounded C
proof −
 have S = {} ∨ S homeomorphic ball (0::complex) 1
   using simply_connected_eq_homeomorphic_to_disc assms openS by blast
 then show ?thesis
 proof
   assume S = {}
   then have bounded S
     by simp
   with ‹S = {}› show ?thesis
     by simp
 next
   assume S01: S homeomorphic ball (0::complex) 1
   then obtain g f
     where gim: g ‘ S = ball 0 1 and fg: ⋀x. x ∈ S ⟹ f(g x) = x
       and fim: f ‘ ball 0 1 = S and gf: ⋀y. cmod y < 1 ⟹ g(f y) = y
       and contg: continuous_on S g and contf: continuous_on (ball 0 1) f
     by (fastforce simp: homeomorphism_def homeomorphic_def)
   define D where D ≡ λn. ball (0::complex) (1 − 1/(of_nat n + 2))
    define A where A ≡ λn. {z::complex. 1 − 1/(of_nat n + 2) < norm z ∧
norm z < 1}
   define X where X ≡ λn::nat. closure(f ‘ A n)
   have D01: D n ⊆ ball 0 1 for n
     by (simp add: D_def ball_subset_ball_iff)
   have A01: A n ⊆ ball 0 1 for n
```

```
    by (auto simp: A_def)
  have cloX: closed(X n) for n
    by (simp add: X_def)
  have Xsubclo: X n ⊆ closure S for n
    unfolding X_def by (metis A01 closure_mono fim image_mono)
  have connX: connected(X n) for n
    unfolding X_def
    apply (rule connected_imp_connected_closure)
    apply (rule connected_continuous_image)
    apply (simp add: continuous_on_subset [OF contf A01])
    using connected_annulus [of _ 0::complex] by (simp add: A_def)
  have nestX: X n ⊆ X m if m ≤ n for m n
  proof −
    have 1 − 1 / (real m + 2) ≤ 1 − 1 / (real n + 2)
      using that by (auto simp: field_simps)
    then show ?thesis
      by (auto simp: X_def A_def intro!: closure_mono)
  qed
  have closure S − S ⊆ (⋂ n. X n)
  proof
    fix x
    assume x ∈ closure S − S
    then have x ∈ closure S x ∉ S by auto
    show x ∈ (⋂ n. X n)
    proof
      fix n
      have ball 0 1 = closure (D n) ∪ A n
        by (auto simp: D_def A_def le_less_trans)
      with fim have Seq: S = f ‘ (closure (D n)) ∪ f ‘ (A n)
        by (simp add: image_Un)
      have continuous_on (closure (D n)) f
        by (simp add: D_def cball_subset_ball_iff continuous_on_subset [OF
contf])
      moreover have compact (closure (D n))
        by (simp add: D_def)
      ultimately have clo_fim: closed (f ‘ closure (D n))
        using compact_continuous_image compact_imp_closed by blast
      have ∗: (f ‘ cball 0 (1 − 1 / (real n + 2))) ⊆ S
        by (force simp: D_def Seq)
      show x ∈ X n
        using ‹x ∈ closure S› unfolding X_def Seq
        using ‹x ∉ S› ∗ D_def clo_fim by auto
    qed
  qed
  moreover have (⋂ n. X n) ⊆ closure S − S
  proof −
    have (⋂ n. X n) ⊆ closure S
    proof −
      have (⋂ n. X n) ⊆ X 0
```

```
      by blast
    also have ... ⊆ closure S
      apply (simp add: X_def fim [symmetric])
      apply (rule closure_mono)
      by (auto simp: A_def)
    finally show (⋂ n. X n) ⊆ closure S .
  qed
  moreover have (⋂ n. X n) ∩ S ⊆ {}
  proof (clarify, clarsimp simp: X_def fim [symmetric])
    fix x assume x [rule_format]: ∀ n. f x ∈ closure (f ‘ A n) and cmod x < 1
    then obtain n where n: 1 / (1 − norm x) < of_nat n
      using reals_Archimedean2 by blast
    with ‹cmod x < 1› gr0I have XX: 1 / of_nat n < 1 − norm x and n > 0
      by (fastforce simp: field_split_simps algebra_simps)+
    have f x ∈ f ‘ (D n)
      using n ‹cmod x < 1› by (auto simp: field_split_simps algebra_simps
D_def)
    moreover have  f ‘ D n ∩ closure (f ‘ A n) = {}
    proof −
      have op_fDn: open(f ‘ (D n))
      proof (rule invariance_of_domain)
        show continuous_on (D n) f
          by (rule continuous_on_subset [OF contf D01])
        show open (D n)
          by (simp add: D_def)
        show inj_on f (D n)
        unfolding inj_on_def using D01 by (metis gf mem_ball_0 subsetCE)
      qed
      have injf: inj_on f (ball 0 1)
        by (metis mem_ball_0 inj_on_def gf)
      have D n ∪ A n ⊆ ball 0 1
        using D01 A01 by simp
      moreover have D n ∩ A n = {}
        by (auto simp: D_def A_def)
      ultimately have f ‘ D n ∩ f ‘ A n = {}
        by (metis A01 D01 image_is_empty inj_on_image_Int injf)
      then show ?thesis
        by (simp add: open_Int_closure_eq_empty [OF op_fDn])
    qed
    ultimately show False
      using x [of n] by blast
  qed
  ultimately
  show (⋂ n. X n) ⊆ closure S − S
    using closure_subset disjoint_iff_not_equal by blast
qed
ultimately have closure S − S = (⋂ n. X n) by blast
then have frontierS: frontier S = (⋂ n. X n)
  by (simp add: frontier_def openS interior_open)
```

```
    show ?thesis
    proof (cases bounded S)
      case True
      have bouX: bounded (X n) for n
        apply (simp add: X_def)
        apply (rule bounded_closure)
        by (metis A01 fim image_mono bounded_subset [OF True])
      have compaX: compact (X n) for n
        apply (simp add: compact_eq_bounded_closed bouX)
        apply (auto simp: X_def)
        done
      have connected (⋂ n. X n)
        by (metis nestX compaX connX connected_nest)
      then show ?thesis
        by (simp add: True ‹frontier S = (⋂ n. X n)›)
    next
      case False
      have unboundedX: ¬ bounded(X n) for n
      proof
        assume bXn: bounded(X n)
        have continuous_on (cball 0 (1 - 1 / (2 + real n))) f
          by (simp add: cball_subset_ball_iff continuous_on_subset [OF contf])
        then have bounded (f ‘ cball 0 (1 - 1 / (2 + real n)))
          by (simp add: compact_imp_bounded [OF compact_continuous_image])
        moreover have bounded (f ‘ A n)
          by (auto simp: X_def closure_subset image_subset_iff bounded_subset
[OF bXn])
        ultimately have bounded (f ‘ (cball 0 (1 - 1/(2 + real n)) ∪ A n))
          by (simp add: image_Un)
        then have bounded (f ‘ ball 0 1)
          apply (rule bounded_subset)
          apply (auto simp: A_def algebra_simps)
          done
        then show False
          using False by (simp add: fim [symmetric])
      qed
      have clo_INTX: closed(⋂(range X))
        by (metis cloX closed_INT)
      then have lcX: locally compact (⋂(range X))
        by (metis closed_imp_locally_compact)
      have False if C: C ∈ components (frontier S) and boC: bounded C for C
      proof −
        have closed C
          by (metis C closed_components frontier_closed)
        then have compact C
          by (metis boC compact_eq_bounded_closed)
        have Cco: C ∈ components (⋂(range X))
          by (metis frontierS C)
        obtain K where C ⊆ K compact K
```

```
            and Ksub: K ⊆ ⋂(range X) and clo: closed(⋂(range X) − K)
      proof (cases {k. C ⊆ k ∧ compact k ∧ openin (top_of_set (⋂(range X)))
k} = {})
        case True
        then show ?thesis
          using Sura_Bura [OF lcX Cco ‹compact C›] boC
          by (simp add: True)
      next
        case False
          then obtain L where compact L C ⊆ L and K: openin (top_of_set
(⋂x. X x)) L
          by blast
        show ?thesis
        proof
          show L ⊆ ⋂(range X)
            by (metis K openin_imp_subset)
          show closed (⋂(range X) − L)
              by (metis closedin_diff closedin_self closedin_closed_trans [OF _
clo_INTX] K)
        qed (use ‹compact L› ‹C ⊆ L› in auto)
      qed
      obtain U V where open U and compact (closure U) and open V K ⊆ U
                and V: ⋂(range X) − K ⊆ V and U ∩ V = {}
        using separation_normal_compact [OF ‹compact K› clo] by blast
      then have U ∩ (⋂ (range X) − K) = {}
        by blast
      have (closure U − U) ∩ (⋂n. X n ∩ closure U) ≠ {}
      proof (rule compact_imp_fip)
        show compact (closure U − U)
          by (metis ‹compact (closure U)› ‹open U› compact_diff)
        show ⋀T. T ∈ range (λn. X n ∩ closure U) ⟹ closed T
          by clarify (metis cloX closed_Int closed_closure)
        show (closure U − U) ∩ ⋂F ≠ {}
          if finite F and F: F ⊆ range (λn. X n ∩ closure U) for F
        proof
          assume empty: (closure U − U) ∩ ⋂F = {}
          obtain J where finite J and J: F = (λn. X n ∩ closure U) ‘ J
            using finite_subset_image [OF ‹finite F› F] by auto
          show False
          proof (cases J = {})
            case True
            with J empty have closed U
              by (simp add: closure_subset_eq)
            have C ≠ {}
              using C in_components_nonempty by blast
            then have U ≠ {}
              using ‹K ⊆ U› ‹C ⊆ K› by blast
            moreover have U ≠ UNIV
              using ‹compact (closure U)› by auto
```

```
            ultimately show False
              using ‹open U› ‹closed U› clopen by blast
          next
            case False
            define j where j ≡ Max J
            have j ∈ J
              by (simp add: False ‹finite J› j_def)
            have jmax: ⋀m. m ∈ J ⟹ m ≤ j
              by (simp add: j_def ‹finite J›)
            have ⋂ ((λn. X n ∩ closure U) ‘ J) = X j ∩ closure U
              using False jmax nestX ‹j ∈ J› by auto
            then have X j ∩ closure U = X j ∩ U
              apply safe
              using DiffI J empty apply auto[1]
              using closure_subset by blast
            then have openin (top_of_set (X j)) (X j ∩ closure U)
              by (simp add: openin_open_Int ‹open U›)
            moreover have closedin (top_of_set (X j)) (X j ∩ closure U)
              by (simp add: closedin_closed_Int)
            moreover have X j ∩ closure U ≠ X j
                  by (metis unboundedX ‹compact (closure U)› bounded_subset
compact_eq_bounded_closed inf.order_iff)
            moreover have X j ∩ closure U ≠ {}
            proof −
              have C ≠ {}
                using C in_components_nonempty by blast
              moreover have C ⊆ X j ∩ closure U
                using ‹C ⊆ K› ‹K ⊆ U› Ksub closure_subset by blast
              ultimately show ?thesis by blast
            qed
            ultimately show False
              using connX [of j] by (force simp: connected_clopen)
          qed
        qed
      qed
      moreover have (⋂n. X n ∩ closure U) = (⋂n. X n) ∩ closure U
        by blast
      moreover have x ∈ U if ⋀n. x ∈ X n x ∈ closure U for x
      proof −
        have x ∉ V
          using ‹U ∩ V = {}› ‹open V› closure_iff_nhds_not_empty that(2) by
blast
        then show ?thesis
        by (metis (no_types) Diff_iff INT_I V ‹K ⊆ U› contra_subsetD that(1))
      qed
      ultimately show False
        by (auto simp: open_Int_closure_eq_empty [OF ‹open V›, of U])
    qed
    then show ?thesis
```

```
      by (auto simp: False)
    qed
  qed
qed


lemma unbounded_complement_components:
  assumes C: C ∈ components (− S) and S: connected S
    and prev: if bounded S then connected(frontier S)
              else ∀ C ∈ components(frontier S). ¬ bounded C
  shows ¬ bounded C
proof (cases bounded S)
  case True
  with prev have S ≠ UNIV and confr: connected(frontier S)
    by auto
  obtain w where C_ccsw: C = connected_component_set (− S) w and w ∉ S
    using C by (auto simp: components_def)
  show ?thesis
  proof (cases S = {})
    case True with C show ?thesis by auto
  next
    case False
    show ?thesis
    proof
      assume bounded C
      then have outside C ≠ {}
        using outside_bounded_nonempty by metis
       then obtain z where z: ¬ bounded (connected_component_set (− C) z)
and z ∉ C
        by (auto simp: outside_def)
      have clo_ccs: closed (connected_component_set (− S) x) for x
        by (simp add: closed_Compl closed_connected_component openS)
      have connected_component_set (− S) w = connected_component_set (− S)
z
      proof (rule joinable_connected_component_eq [OF confr])
        show frontier S ⊆ − S
          using openS by (auto simp: frontier_def interior_open)
        have False if connected_component_set (− S) w ∩ frontier (− S) = {}
        proof −
          have C ∩ frontier S = {}
            using that by (simp add: C_ccsw)
          then show False
             by (metis C_ccsw ComplI Compl_eq_Compl_iff Diff_subset False ‹w
∉ S› clo_ccs closure_closed compl_bot_eq connected_component_eq_UNIV con-
nected_component_eq_empty empty_subsetI frontier_complement frontier_def fron-
tier_not_empty frontier_of_connected_component_subset le_inf_iff subset_antisym)
        qed
        then show connected_component_set (− S) w ∩ frontier S ≠ {}
          by auto
```

```
      have ∗: ⟦frontier C ⊆ C; frontier C ⊆ F; frontier C ≠ {}⟧ ⟹ C ∩ F ≠
{} for C F::complex set
        by blast
      have connected_component_set (− S) z ∩ frontier (− S) ≠ {}
      proof (rule ∗)
     show frontier (connected_component_set (− S) z) ⊆ connected_component_set
(− S) z
         by (auto simp: closed_Compl closed_connected_component frontier_def
openS)
       show frontier (connected_component_set (− S) z) ⊆ frontier (− S)
        using frontier_of_connected_component_subset by fastforce
       have ¬ bounded (−S)
        by (simp add: True cobounded_imp_unbounded)
       then have connected_component_set (− S) z ≠ {}
        apply (simp only: connected_component_eq_empty)
        using confr openS ‹bounded C› ‹w ∉ S›
        apply (simp add: frontier_def interior_open C_ccsw)
       by (metis ComplI Compl_eq_Diff_UNIV connected_UNIV closed_closure
closure_subset connected_component_eq_self
                 connected_diff_open_from_closed subset_UNIV)
       then show frontier (connected_component_set (− S) z) ≠ {}
        apply (simp add: frontier_eq_empty connected_component_eq_UNIV)
        apply (metis False compl_top_eq double_compl)
        done
      qed
      then show connected_component_set (− S) z ∩ frontier S ≠ {}
       by auto
    qed
    then show False
    by (metis C_ccsw Compl_iff ‹w ∉ S› ‹z ∉ C› connected_component_eq_empty
connected_component_idemp)
   qed
 qed
next
 case False
 obtain w where C_ccsw: C = connected_component_set (− S) w and w ∉ S
   using C by (auto simp: components_def)
 have frontier (connected_component_set (− S) w) ⊆ connected_component_set
(− S) w
   by (simp add: closed_Compl closed_connected_component frontier_subset_eq
openS)
 moreover have frontier (connected_component_set (− S) w) ⊆ frontier S
   using frontier_complement frontier_of_connected_component_subset by blast
 moreover have frontier (connected_component_set (− S) w) ≠ {}
  by (metis C C_ccsw False bounded_empty compl_top_eq connected_component_eq_UNIV
double_compl frontier_not_empty in_components_nonempty)
 ultimately obtain z where zin: z ∈ frontier S and z: z ∈ connected_component_set
(− S) w
   by blast
```

have $*$: *connected_component_set* (*frontier S*) *z* $\in$ *components*(*frontier S*)
by (*simp add*: ‹*z* $\in$ *frontier S*› *componentsI*)
with *prev False* **have** $\neg$ *bounded* (*connected_component_set* (*frontier S*) *z*)
by *simp*
moreover have *connected_component* ($-$ *S*) *w* = *connected_component* ($-$ *S*) *z*
using *connected_component_eq* [*OF z*] **by** *force*
ultimately show *?thesis*
by (*metis C_ccsw* $*$ *zin bounded_subset closed_Compl closure_closed connected_component_maximal*
connected_component_refl connected_connected_component frontier_closures in_components_subset le_inf_iff mem_Collect_eq openS)
qed

lemma *empty_inside*:
assumes *connected S* $\bigwedge C$. *C* $\in$ *components* ($-$ *S*) $\Longrightarrow$ $\neg$ *bounded C*
shows *inside S* = {}
using *assms* **by** (*auto simp*: *components_def inside_def*)

lemma *empty_inside_imp_simply_connected*:
⟦*connected S*; *inside S* = {}⟧ $\Longrightarrow$ *simply_connected S*
by (*metis ComplI inside_Un_outside openS outside_mono simply_connected_eq_winding_number_zero subsetCE sup_bot.left_neutral winding_number_zero_in_outside*)

end

proposition
fixes *S* :: *complex set*
assumes *open S*
shows *simply_connected_eq_frontier_properties*:
simply_connected S $\longleftrightarrow$
connected S $\wedge$
(*if bounded S then connected*(*frontier S*)
else ($\forall$ *C* $\in$ *components*(*frontier S*). $\neg$*bounded C*)) (**is** *?fp*)
and *simply_connected_eq_unbounded_complement_components*:
simply_connected S $\longleftrightarrow$
connected S $\wedge$ ($\forall$ *C* $\in$ *components*($-$ *S*). $\neg$*bounded C*) (**is** *?ucc*)
and *simply_connected_eq_empty_inside*:
simply_connected S $\longleftrightarrow$
connected S $\wedge$ *inside S* = {} (**is** *?ei*)
proof $-$
interpret *SC_Chain*
using *assms* **by** (*simp add*: *SC_Chain_def*)
have *?fp* $\wedge$ *?ucc* $\wedge$ *?ei*
proof $-$
have $*$: ⟦$\alpha \Longrightarrow \beta$; $\beta \Longrightarrow \gamma$; $\gamma \Longrightarrow \delta$; $\delta \Longrightarrow \alpha$⟧
$\Longrightarrow$ ($\alpha \longleftrightarrow \beta$) $\wedge$ ($\alpha \longleftrightarrow \gamma$) $\wedge$ ($\alpha \longleftrightarrow \delta$) **for** α β γ δ
by *blast*
show *?thesis*

apply (*rule* ∗)
using *frontier_properties simply_connected_imp_connected* **apply** *blast*
apply *clarify*
using *unbounded_complement_components simply_connected_imp_connected*
apply *blast*
using *empty_inside* **apply** *blast*
using *empty_inside_imp_simply_connected* **apply** *blast*
done
qed
then show *?fp ?ucc ?ei*
by *safe*
qed

lemma *simply_connected_iff_simple*:
fixes *S* :: *complex set*
assumes *open S bounded S*
shows *simply_connected S* ⟷ *connected S* ∧ *connected*(− *S*)
apply (*simp add*: *simply_connected_eq_unbounded_complement_components assms*, *safe*)
apply (*metis DIM_complex assms*(*2*) *cobounded_has_bounded_component double_compl order_refl*)
by (*meson assms inside_bounded_complement_connected_empty simply_connected_eq_empty_inside simply_connected_eq_unbounded_complement_components*)

8.4 Further equivalences based on continuous logs and sqrts

context *SC_Chain*
begin

lemma *continuous_log*:
fixes *f* :: *complex*⇒*complex*
assumes *S*: *simply_connected S*
and *contf*: *continuous_on S f* **and** *nz*: ⋀*z*. *z* ∈ *S* ⟹ *f z* ≠ *0*
shows ∃ *g*. *continuous_on S g* ∧ (∀ *z* ∈ *S*. *f z* = *exp*(*g z*))
proof −
consider *S* = {} | *S homeomorphic ball* (*0*::*complex*) *1*
using *simply_connected_eq_homeomorphic_to_disc* [*OF openS*] *S* **by** *metis*
then show *?thesis*
proof *cases*
case *1* **then show** *?thesis*
by *simp*
next
case *2*
then obtain *h k* :: *complex*⇒*complex*
where *kh*: ⋀*x*. *x* ∈ *S* ⟹ *k*(*h x*) = *x* **and** *him*: *h* ‘ *S* = *ball 0 1*
and *conth*: *continuous_on S h*
and *hk*: ⋀*y*. *y* ∈ *ball 0 1* ⟹ *h*(*k y*) = *y* **and** *kim*: *k* ‘ *ball 0 1* = *S*
and *contk*: *continuous_on* (*ball 0 1*) *k*
unfolding *homeomorphism_def homeomorphic_def* **by** *metis*

```
    obtain g where contg: continuous_on (ball 0 1) g
            and expg: ⋀z. z ∈ ball 0 1 ⟹ (f ∘ k) z = exp (g z)
    proof (rule continuous_logarithm_on_ball)
      show continuous_on (ball 0 1) (f ∘ k)
        apply (rule continuous_on_compose [OF contk])
        using kim continuous_on_subset [OF contf]
        by blast
      show ⋀z. z ∈ ball 0 1 ⟹ (f ∘ k) z ≠ 0
        using kim nz by auto
    qed auto
    then show ?thesis
      by (metis comp_apply conth continuous_on_compose him imageI kh)
  qed
qed

lemma continuous_sqrt:
  fixes f :: complex⇒complex
  assumes contf: continuous_on S f and nz: ⋀z. z ∈ S ⟹ f z ≠ 0
  and prev: ⋀f::complex⇒complex.
              ⟦continuous_on S f; ⋀z. z ∈ S ⟹ f z ≠ 0⟧
                ⟹ ∃g. continuous_on S g ∧ (∀z ∈ S. f z = exp(g z))
  shows ∃g. continuous_on S g ∧ (∀z ∈ S. f z = (g z)^2)
proof −
  obtain g where contg: continuous_on S g and geq: ⋀z. z ∈ S ⟹ f z = exp(g
z)
    using contf nz prev by metis
  show ?thesis
proof (intro exI ballI conjI)
  show continuous_on S (λz. exp(g z/2))
     by (intro continuous_intros) (auto simp: contg)
    show ⋀z. z ∈ S ⟹ f z = (exp (g z/2))^2
    by (metis (no_types, lifting) divide_inverse exp_double geq mult.left_commute
mult.right_neutral right_inverse zero_neq_numeral)
  qed
qed

lemma continuous_sqrt_imp_simply_connected:
  assumes connected S
    and prev: ⋀f::complex⇒complex. ⟦continuous_on S f; ∀z ∈ S. f z ≠ 0⟧
            ⟹ ∃g. continuous_on S g ∧ (∀z ∈ S. f z = (g z)^2)
  shows simply_connected S
proof (clarsimp simp add: simply_connected_eq_holomorphic_sqrt [OF openS]
‹connected S›)
  fix f
  assume f holomorphic_on S and nz: ∀z∈S. f z ≠ 0
  then obtain g where contg: continuous_on S g and geq: ⋀z. z ∈ S ⟹ f z =
(g z)^2
    by (metis holomorphic_on_imp_continuous_on prev)
  show ∃g. g holomorphic_on S ∧ (∀z∈S. f z = (g z)^2)
```

```
  proof (intro exI ballI conjI)
    show g holomorphic_on S
    proof (clarsimp simp add: holomorphic_on_open [OF openS])
      fix z
      assume z ∈ S
      with nz geq have g z ≠ 0
        by auto
      obtain δ where 0 < δ ⋀w. ⟦w ∈ S; dist w z < δ⟧ ⟹ dist (g w) (g z) <
cmod (g z)
        using contg [unfolded continuous_on_iff] by (metis ‹g z ≠ 0› ‹z ∈ S›
zero_less_norm_iff)
      then have δ: ⋀w. ⟦w ∈ S; w ∈ ball z δ⟧ ⟹ g w + g z ≠ 0
        apply (clarsimp simp: dist_norm)
          by (metis ‹g z ≠ 0› add_diff_cancel_left' diff_0_right norm_eq_zero
norm_increases_online norm_minus_commute norm_not_less_zero not_less_iff_gr_or_eq)
      have ∗: (λx. (f x − f z) / (x − z) / (g x + g z)) −z→ deriv f z / (g z + g z)
        apply (intro tendsto_intros)
        using SC_Chain.openS SC_Chain_axioms ‹f holomorphic_on S› ‹z ∈ S›
has_field_derivativeD holomorphic_derivI apply fastforce
          using ‹z ∈ S› contg continuous_on_eq_continuous_at isCont_def openS
apply blast
        by (simp add: ‹g z ≠ 0›)
      then have (g has_field_derivative deriv f z / (g z + g z)) (at z)
        unfolding has_field_derivative_iff
      proof (rule Lim_transform_within_open)
        show open (ball z δ ∩ S)
          by (simp add: openS open_Int)
        show z ∈ ball z δ ∩ S
          using ‹z ∈ S› ‹0 < δ› by simp
        show ⋀x. ⟦x ∈ ball z δ ∩ S; x ≠ z⟧
                ⟹ (f x − f z) / (x − z) / (g x + g z) = (g x − g z) / (x − z)
          using δ
          apply (simp add: geq ‹z ∈ S› divide_simps)
          apply (auto simp: algebra_simps power2_eq_square)
          done
      qed
      then show ∃f'. (g has_field_derivative f') (at z) ..
    qed
  qed (use geq in auto)
qed

end

proposition
  fixes S :: complex set
  assumes open S
  shows simply_connected_eq_continuous_log:
        simply_connected S ⟷
         connected S ∧
```

```
        (∀f::complex⇒complex. continuous_on S f ∧ (∀z ∈ S. f z ≠ 0)
          ⟶ (∃g. continuous_on S g ∧ (∀z ∈ S. f z = exp (g z)))) (is ?log)
    and simply_connected_eq_continuous_sqrt:
        simply_connected S ⟷
        connected S ∧
        (∀f::complex⇒complex. continuous_on S f ∧ (∀z ∈ S. f z ≠ 0)
          ⟶ (∃g. continuous_on S g ∧ (∀z ∈ S. f z = (g z)^2))) (is ?sqrt)
proof −
  interpret SC_Chain
    using assms by (simp add: SC_Chain_def)
  have ?log ∧ ?sqrt
proof −
  have ∗: ⟦α ⟹ β; β ⟹ γ; γ ⟹ α⟧
          ⟹ (α ⟷ β) ∧ (α ⟷ γ) for α β γ
    by blast
  show ?thesis
    apply (rule ∗)
    apply (simp add: local.continuous_log winding_number_zero)
    apply (simp add: continuous_sqrt)
    apply (simp add: continuous_sqrt_imp_simply_connected)
    done
qed
  then show ?log ?sqrt
    by safe
qed
```

8.5 More Borsukian results

```
lemma Borsukian_componentwise_eq:
  fixes S :: 'a::euclidean_space set
  assumes S: locally connected S ∨ compact S
  shows Borsukian S ⟷ (∀C ∈ components S. Borsukian C)
proof −
  have ∗: ANR(−{0::complex})
    by (simp add: ANR_delete open_Compl open_imp_ANR)
  show ?thesis
    using cohomotopically_trivial_on_components [OF assms ∗] by (auto simp:
Borsukian_alt)
qed

lemma Borsukian_componentwise:
  fixes S :: 'a::euclidean_space set
 assumes locally connected S ∨ compact S ⋀C. C ∈ components S ⟹ Borsukian
C
  shows Borsukian S
  by (metis Borsukian_componentwise_eq assms)

lemma simply_connected_eq_Borsukian:
  fixes S :: complex set
```

shows *open S* $\Longrightarrow$ (*simply_connected S* $\longleftrightarrow$ *connected S* $\wedge$ *Borsukian S*)
by (*auto simp*: *simply_connected_eq_continuous_log Borsukian_continuous_logarithm*)

lemma *Borsukian_eq_simply_connected*:
fixes *S* :: *complex set*
shows *open S* $\Longrightarrow$ *Borsukian S* $\longleftrightarrow$ ($\forall$ *C* $\in$ *components S*. *simply_connected C*)
apply (*auto simp*: *Borsukian_componentwise_eq open_imp_locally_connected*)
using *in_components_connected open_components simply_connected_eq_Borsukian*
apply *blast*
using *open_components simply_connected_eq_Borsukian* **by** *blast*

lemma *Borsukian_separation_open_closed*:
fixes *S* :: *complex set*
assumes *S*: *open S* $\vee$ *closed S* **and** *bounded S*
shows *Borsukian S* $\longleftrightarrow$ *connected*(− *S*)
using *S*
proof
assume *open S*
show *?thesis*
unfolding *Borsukian_eq_simply_connected* [*OF* ‹*open S*›]
by (*meson* ‹*open S*› ‹*bounded S*› *bounded_subset in_components_connected in_components_subset nonseparation_by_component_eq open_components simply_connected_iff_simple*)
next
assume *closed S*
with ‹*bounded S*› **show** *?thesis*
by (*simp add*: *Borsukian_separation_compact compact_eq_bounded_closed*)
qed

8.6 Finally, the Riemann Mapping Theorem

theorem *Riemann_mapping_theorem*:
open S $\wedge$ *simply_connected S* $\longleftrightarrow$
S = {} $\vee$ *S* = *UNIV* $\vee$
($\exists$ *f g*. *f holomorphic_on S* $\wedge$ *g holomorphic_on ball 0 1* $\wedge$
($\forall$ *z* $\in$ *S*. *f z* $\in$ *ball 0 1* $\wedge$ *g*(*f z*) = *z*) $\wedge$
($\forall$ *z* $\in$ *ball 0 1*. *g z* $\in$ *S* $\wedge$ *f*(*g z*) = *z*))
(**is** _ = *?rhs*)
proof −
have *simply_connected S* $\longleftrightarrow$ *?rhs* **if** *open S*
by (*simp add*: *simply_connected_eq_biholomorphic_to_disc that*)
moreover have *open S* **if** *?rhs*
proof −
{ **fix** *f g*
assume *g*: *g holomorphic_on ball 0 1* $\forall$ *z*$\in$*ball 0 1*. *g z* $\in$ *S* $\wedge$ *f* (*g z*) = *z*
and $\forall$ *z*$\in$*S*. *cmod* (*f z*) < *1* $\wedge$ *g* (*f z*) = *z*
then have *S* = *g* ‘ (*ball 0 1*)
by (*force simp*:)
then have *open S*

by (*metis open_ball g inj_on_def open_mapping_thm3*)
}
with *that* **show** *open S* **by** *auto*
qed
ultimately show *?thesis* **by** *metis*
qed

8.7 Applications to Winding Numbers

lemma *simply_connected_inside_simple_path*:
fixes *p* :: *real* ⇒ *complex*
shows *simple_path p* ⟹ *simply_connected*(*inside*(*path_image p*))
using *Jordan_inside_outside connected_simple_path_image inside_simple_curve_imp_closed simply_connected_eq_frontier_properties*
by *fastforce*

lemma *simply_connected_Int*:
fixes *S* :: *complex set*
assumes *open S open T simply_connected S simply_connected T connected* (*S* ∩ *T*)
shows *simply_connected* (*S* ∩ *T*)
using *assms* **by** (*force simp*: *simply_connected_eq_winding_number_zero open_Int*)

8.8 The winding number defines a continuous logarithm for the path itself

lemma *winding_number_as_continuous_log*:
assumes *path p* **and** *ζ*: *ζ* ∉ *path_image p*
obtains *q* **where** *path q*
pathfinish q − *pathstart q* = *2* ∗ *of_real pi* ∗ i ∗ *winding_number p ζ*
⋀*t*. *t* ∈ {*0..1*} ⟹ *p t* = *ζ* + *exp*(*q t*)
proof −
let *?q* = λ*t*. *2* ∗ *of_real pi* ∗ i ∗ *winding_number*(*subpath 0 t p*) *ζ* + *Ln*(*pathstart p* − *ζ*)
show *?thesis*
proof
have ∗: *continuous* (*at t within* {*0..1*}) (λ*x*. *winding_number* (*subpath 0 x p*) *ζ*)
if *t*: *t* ∈ {*0..1*} **for** *t*
proof −
let *?B* = *ball* (*p t*) (*norm*(*p t* − *ζ*))
have *p t* ≠ *ζ*
using *path_image_def that ζ* **by** *blast*
then have *simply_connected ?B*
by (*simp add*: *convex_imp_simply_connected*)
then have ⋀*f*::*complex*⇒*complex*. *continuous_on ?B f* ∧ (∀ *ζ* ∈ *?B*. *f ζ* ≠ *0*)
⟶ (∃ *g*. *continuous_on ?B g* ∧ (∀ *ζ* ∈ *?B*. *f ζ* = *exp* (*g ζ*)))
by (*simp add*: *simply_connected_eq_continuous_log*)

```
    moreover have continuous_on ?B (λw. w − ζ)
      by (intro continuous_intros)
    moreover have (∀ z ∈ ?B. z − ζ ≠ 0)
      by (auto simp: dist_norm)
    ultimately obtain g where contg: continuous_on ?B g
      and geq: ⋀z. z ∈ ?B ⟹ z − ζ = exp (g z) by blast
    obtain d where 0 < d and d:
      ⋀x. ⟦x ∈ {0..1}; dist x t < d⟧ ⟹ dist (p x) (p t) < cmod (p t − ζ)
      using ‹path p› t unfolding path_def continuous_on_iff
      by (metis ‹p t ≠ ζ› right_minus_eq zero_less_norm_iff)
    have ((λx. winding_number (λw. subpath 0 x p w − ζ) 0 −
              winding_number (λw. subpath 0 t p w − ζ) 0) ⟶ 0)
         (at t within {0..1})
    proof (rule Lim_transform_within [OF _ ‹d > 0›])
      have continuous (at t within {0..1}) (g o p)
      proof (rule continuous_within_compose)
        show continuous (at t within {0..1}) p
          using ‹path p› continuous_on_eq_continuous_within path_def that by
blast
        show continuous (at (p t) within p ‘ {0..1}) g
         by (metis (no_types, lifting) open_ball UNIV_I ‹p t ≠ ζ› centre_in_ball
contg continuous_on_eq_continuous_at continuous_within_topological right_minus_eq
zero_less_norm_iff)
      qed
       with LIM_zero have ((λu. (g (subpath t u p 1) − g (subpath t u p 0)))
⟶ 0) (at t within {0..1})
        by (auto simp: subpath_def continuous_within o_def)
      then show ((λu. (g (subpath t u p 1) − g (subpath t u p 0)) / (2 ∗ of_real
pi ∗ i)) ⟶ 0)
         (at t within {0..1})
        by (simp add: tendsto_divide_zero)
      show (g (subpath t u p 1) − g (subpath t u p 0)) / (2 ∗ of_real pi ∗ i) =
           winding_number (λw. subpath 0 u p w − ζ) 0 − winding_number (λw.
subpath 0 t p w − ζ) 0
        if u ∈ {0..1} 0 < dist u t dist u t < d for u
      proof −
        have closed_segment t u ⊆ {0..1}
          using closed_segment_eq_real_ivl t that by auto
        then have piB: path_image(subpath t u p) ⊆ ?B
          apply (clarsimp simp add: path_image_subpath_gen)
              by (metis subsetD le_less_trans ‹dist u t < d› d dist_commute
dist_in_closed_segment)
        have ∗: path (g ∘ subpath t u p)
          apply (rule path_continuous_image)
          using ‹path p› t that apply auto[1]
          using piB contg continuous_on_subset by blast
        have (g (subpath t u p 1) − g (subpath t u p 0)) / (2 ∗ of_real pi ∗ i)
           =  winding_number (exp ∘ g ∘ subpath t u p) 0
          using winding_number_compose_exp [OF ∗]
```

```
          by (simp add: pathfinish_def pathstart_def o_assoc)
        also have ... = winding_number (λw. subpath t u p w − ζ) 0
        proof (rule winding_number_cong)
          have exp(g y) = y − ζ if y ∈ (subpath t u p) ` {0..1} for y
            by (metis that geq path_image_def piB subset_eq)
         then show ⋀x. ⟦0 ≤ x; x ≤ 1⟧ ⟹ (exp ∘ g ∘ subpath t u p) x = subpath
t u p x − ζ
            by auto
        qed
        also have ... = winding_number (λw. subpath 0 u p w − ζ) 0 −
                        winding_number (λw. subpath 0 t p w − ζ) 0
          apply (simp add: winding_number_offset [symmetric])
          using winding_number_subpath_combine [OF ‹path p› ζ, of 0 t u] ‹t ∈
{0..1}› ‹u ∈ {0..1}›
          by (simp add: add.commute eq_diff_eq)
        finally show ?thesis .
      qed
    qed
    then show ?thesis
    by (subst winding_number_offset) (simp add: continuous_within LIM_zero_iff)
  qed
  show path ?q
    unfolding path_def
   by (intro continuous_intros) (simp add: continuous_on_eq_continuous_within
∗)

  have ζ ≠ p 0
    by (metis ζ pathstart_def pathstart_in_path_image)
  then show pathfinish ?q − pathstart ?q = 2 ∗ of_real pi ∗ i ∗ winding_number
p ζ
    by (simp add: pathfinish_def pathstart_def)
  show p t = ζ + exp (?q t) if t ∈ {0..1} for t
  proof −
    have path (subpath 0 t p)
      using ‹path p› that by auto
    moreover
    have ζ ∉ path_image (subpath 0 t p)
    using ζ [unfolded path_image_def] that by (auto simp: path_image_subpath)
    ultimately show ?thesis
      using winding_number_exp_2pi [of subpath 0 t p ζ] ‹ζ ≠ p 0›
      by (auto simp: exp_add algebra_simps pathfinish_def pathstart_def sub-
path_def)
  qed
 qed
qed
```

8.9 Winding number equality is the same as path/loop homotopy in C - 0

lemma *winding_number_homotopic_loops_null_eq*:
assumes *path p* **and** *ζ*: *ζ* ∉ *path_image p*
shows *winding_number p ζ = 0* ⟷ (∃ *a*. *homotopic_loops* (−{*ζ*}) *p* (λ*t*. *a*))
(**is** *?lhs = ?rhs*)
proof
assume [*simp*]: *?lhs*
obtain *q* **where** *path q*
and *qeq*: *pathfinish q − pathstart q = 2 ∗ of_real pi ∗* i *∗ winding_number p ζ*
and *peq*: ⋀*t*. *t* ∈ {*0..1*} ⟹ *p t = ζ + exp*(*q t*)
using *winding_number_as_continuous_log* [*OF assms*] **by** *blast*
have ∗: *homotopic_with_canon* (λ*r*. *pathfinish r = pathstart r*)
{*0..1*} (−{*ζ*}) ((λ*w*. *ζ + exp w*) ∘ *q*) ((λ*w*. *ζ + exp w*) ∘ (λ*t*. *0*))
proof (*rule homotopic_with_compose_continuous_left*)
show *homotopic_with_canon* (λ*f*. *pathfinish* ((λ*w*. *ζ + exp w*) ∘ *f*) = *pathstart* ((λ*w*. *ζ + exp w*) ∘ *f*))
{*0..1*} *UNIV q* (λ*t*. *0*)
proof (*rule homotopic_with_mono*, *simp_all add*: *pathfinish_def pathstart_def*)
have *homotopic_loops UNIV q* (λ*t*. *0*)
by (*rule homotopic_loops_linear*) (*use qeq* ‹*path q*› **in** ‹*auto simp*: *path_defs*›)
then have *homotopic_with* (λ*r*. *r 1 = r 0*) (*top_of_set* {*0..1*}) *euclidean q* (λ*t*. *0*)
by (*simp add*: *homotopic_loops_def pathfinish_def pathstart_def*)
then show *homotopic_with* (λ*h*. *exp* (*h 1*) = *exp* (*h 0*)) (*top_of_set* {*0..1*}) *euclidean q* (λ*t*. *0*)
by (*rule homotopic_with_mono*) *simp*
qed
show *continuous_on UNIV* (λ*w*. *ζ + exp w*)
by (*rule continuous_intros*)+
show *range* (λ*w*. *ζ + exp w*) ⊆ −{*ζ*}
by *auto*
qed
then have *homotopic_with_canon* (λ*r*. *pathfinish r = pathstart r*) {*0..1*} (−{*ζ*}) *p* (λ*x*. *ζ + 1*)
by (*rule homotopic_with_eq*) (*auto simp*: *o_def peq pathfinish_def pathstart_def*)
then have *homotopic_loops* (−{*ζ*}) *p* (λ*t*. *ζ + 1*)
by (*simp add*: *homotopic_loops_def*)
then show *?rhs* **..**
next
assume *?rhs*
then obtain *a* **where** *homotopic_loops* (−{*ζ*}) *p* (λ*t*. *a*) **..**
then have *winding_number p ζ = winding_number* (λ*t*. *a*) *ζ a* ≠ *ζ*
using *winding_number_homotopic_loops homotopic_loops_imp_subset* **by** (*force simp*:)+
moreover have *winding_number* (λ*t*. *a*) *ζ = 0*
by (*metis winding_number_zero_const* ‹*a* ≠ *ζ*›)
ultimately show *?lhs* **by** *metis*

qed

lemma *winding_number_homotopic_paths_null_explicit_eq*:
assumes *path p* **and** *ζ*: *ζ* ∉ *path_image p*
shows *winding_number p ζ = 0* ⟷ *homotopic_paths* (−{*ζ*}) *p* (*linepath* (*pathstart p*) (*pathstart p*))
(**is** *?lhs = ?rhs*)
proof
assume *?lhs*
then show *?rhs*
apply (*auto simp*: *winding_number_homotopic_loops_null_eq* [*OF assms*])
apply (*rule homotopic_loops_imp_homotopic_paths_null*)
apply (*simp add*: *linepath_refl*)
done
next
assume *?rhs*
then show *?lhs*
by (*metis ζ pathstart_in_path_image winding_number_homotopic_paths winding_number_trivial*)
qed

lemma *winding_number_homotopic_paths_null_eq*:
assumes *path p* **and** *ζ*: *ζ* ∉ *path_image p*
shows *winding_number p ζ = 0* ⟷ (∃ *a*. *homotopic_paths* (−{*ζ*}) *p* (λ*t*. *a*))
(**is** *?lhs = ?rhs*)
proof
assume *?lhs*
then show *?rhs*
by (*auto simp*: *winding_number_homotopic_paths_null_explicit_eq* [*OF assms*] *linepath_refl*)
next
assume *?rhs*
then show *?lhs*
by (*metis ζ homotopic_paths_imp_pathfinish pathfinish_def pathfinish_in_path_image winding_number_homotopic_paths winding_number_zero_const*)
qed

proposition *winding_number_homotopic_paths_eq*:
assumes *path p* **and** *ζp*: *ζ* ∉ *path_image p*
and *path q* **and** *ζq*: *ζ* ∉ *path_image q*
and *qp*: *pathstart q = pathstart p pathfinish q = pathfinish p*
shows *winding_number p ζ = winding_number q ζ* ⟷ *homotopic_paths* (−{*ζ*}) *p q*
(**is** *?lhs = ?rhs*)
proof
assume *?lhs*
then have *winding_number* (*p +++ reversepath q*) *ζ = 0*
using *assms* **by** (*simp add*: *winding_number_join winding_number_reversepath*)
moreover

have *path* $(p +\!\!+\!\!+ \mathit{reversepath}\ q)$ $\zeta \notin$ *path_image* $(p +\!\!+\!\!+ \mathit{reversepath}\ q)$
using *assms* **by** (*auto simp*: *not_in_path_image_join*)
ultimately obtain *a* **where** *homotopic_paths* $(-\{\zeta\})$ $(p +\!\!+\!\!+ \mathit{reversepath}\ q)$ (*linepath a a*)
using *winding_number_homotopic_paths_null_explicit_eq* **by** *blast*
then show *?rhs*
using *homotopic_paths_imp_pathstart assms*
by (*fastforce simp add*: *dest*: *homotopic_paths_imp_homotopic_loops homotopic_paths_loop_parts*)
next
assume *?rhs*
then show *?lhs*
by (*simp add*: *winding_number_homotopic_paths*)
qed

lemma *winding_number_homotopic_loops_eq*:
assumes *path p* **and** *ζp*: $\zeta \notin$ *path_image p*
and *path q* **and** *ζq*: $\zeta \notin$ *path_image q*
and *loops*: *pathfinish p* = *pathstart p pathfinish q* = *pathstart q*
shows *winding_number p ζ* = *winding_number q ζ* $\longleftrightarrow$ *homotopic_loops* $(-\{\zeta\})$ *p q*
(**is** *?lhs* = *?rhs*)
proof
assume *L*: *?lhs*
have *pathstart p* $\neq \zeta$ *pathstart q* $\neq \zeta$
using *ζp ζq* **by** *blast+*
moreover have *path_connected* $(-\{\zeta\})$
by (*simp add*: *path_connected_punctured_universe*)
ultimately obtain *r* **where** *path r* **and** *rim*: *path_image r* $\subseteq -\{\zeta\}$
and *pas*: *pathstart r* = *pathstart p* **and** *paf*: *pathfinish r* = *pathstart q*
by (*auto simp*: *path_connected_def*)
then have *pathstart r* $\neq \zeta$ **by** *blast*
have *homotopic_loops* $(-\{\zeta\})$ *p* $(r +\!\!+\!\!+ q +\!\!+\!\!+ \mathit{reversepath}\ r)$
proof (*rule homotopic_paths_imp_homotopic_loops*)
show *homotopic_paths* $(-\{\zeta\})$ *p* $(r +\!\!+\!\!+ q +\!\!+\!\!+ \mathit{reversepath}\ r)$
by (*metis* (*mono_tags*, *opaque_lifting*) ‹*path r*› *L ζp ζq* ‹*path p*› ‹*path q*› *homotopic_loops_conjugate loops not_in_path_image_join paf pas path_image_reversepath path_imp_reversepath path_join_eq pathfinish_join pathfinish_reversepath pathstart_join pathstart_reversepath rim subset_Compl_singleton winding_number_homotopic_loops winding_number_homotopic_paths_eq*)
qed (*use loops pas* **in** *auto*)
moreover have *homotopic_loops* $(-\{\zeta\})$ $(r +\!\!+\!\!+ q +\!\!+\!\!+ \mathit{reversepath}\ r)$ *q*
using *rim ζq* **by** (*auto simp*: *homotopic_loops_conjugate paf* ‹*path q*› ‹*path r*› *loops*)
ultimately show *?rhs*
using *homotopic_loops_trans* **by** *metis*
next
assume *?rhs*

then show *?lhs*
by (*simp add*: *winding_number_homotopic_loops*)
qed

end
theory *Complex_Analysis*
imports
Residue_Theorem
Riemann_Mapping
begin

end

References

[1]