The Isoperimetric Inequality for Rectifiable Simple Closed Curves

Lawrence C. Paulson

Abstract

This entry formalises the planar isoperimetric inequality for rectifiable simple closed curves. If a curve has length L, the area of its inside is at most $L^2/(4\pi)$; equality implies that its image is a circle. The development proves a Wirtinger inequality, a special area formula for convex curves, and a convexification argument. It builds on the `Fourier` AFP entry and analysis material in Isabelle2026.

Contents

1 Proof outline and provenance **1**

2 Part 1: The Wirtinger inequality **1**

3 A very special case of Green's theorem for a convex area **26**
3.1 Area under an arc. 26
3.2 Area above an arc. 29
3.3 Lemmas for Green's theorem 32
3.4 Green's theorem special case at zero 49
3.5 Conclusion of Green's theorem and the signed area formula for a convex closed curve. 57

4 Isoperimetric theorem for convex curves **59**
4.1 Convexification 78
4.2 The isoperimetric theorem 112

1 Proof outline and provenance

The proof first establishes the inequality for convex curves. Arc-length reparametrisation and the Wirtinger inequality supply its analytic core; a special case of Green's area formula relates the resulting integral to the enclosed area. Convexification then reduces a general rectifiable simple closed curve to the convex case and accounts for equality.

This development was ported from John Harrison's HOL Light formalisation `100/isoperimetric.ml`.[1] The arc-length material also follows `Multivariate/integration.ml`.[2] Claude assisted with the translation; the Isabelle proofs required substantial manual revision. There were no additional textual sources. The Wirtinger proof uses the AFP entry *Fourier Series*.[3]

2 Part 1: The Wirtinger inequality

theory *Wirtinger*
imports *HOL−Analysis.Rectifiable-Path Fourier.Square-Integrable*

begin

Convex curve lemmas: Switching between views of a convex simple closed curve.

lemma *convex-hull-eq-closure-inside*:
fixes *g* :: *real* ⇒ *complex*
assumes *g*: *simple-path g pathfinish g* = *pathstart g*
and *conv*: *convex* (*inside* (*path-image g*))
shows *convex hull* (*path-image g*) = *closure* (*inside* (*path-image g*))
proof (*rule equalityI*)
have *bounded-inside*: *bounded* (*inside* (*path-image g*))
and *frontier-inside*: *frontier* (*inside* (*path-image g*)) = *path-image g*
using *Jordan-inside-outside g* **by** *blast+*
show *convex hull* (*path-image g*) ⊆ *closure* (*inside* (*path-image g*))
by (*metis Diff-subset conv convex-closure frontier-def frontier-inside hull-minimal*)
have *compact* (*closure* (*inside* (*path-image g*)))
using *compact-closure local.bounded-inside* **by** *blast*
then show *closure* (*inside* (*path-image g*)) ⊆ *convex hull* (*path-image g*)
by (*metis* (*no-types*, *lifting*) *Krein-Milman-frontier conv closure-closure convex-closure*
convex-interior-closure frontier-def frontier-inside order.refl)
qed

lemma *frontier-convex-hull-eq-path-image*:
fixes *g* :: *real* ⇒ *complex*
assumes *simple-path g pathfinish g* = *pathstart g convex* (*inside* (*path-image g*))
shows *frontier* (*convex hull* (*path-image g*)) = *path-image g*
using *Jordan-inside-outside*
by (*simp add*: *assms convex-hull-eq-closure-inside convex-interior-closure frontier-def*)

[1] https://github.com/jrh13/hol-light/blob/master/100/isoperimetric.ml
[2] https://github.com/jrh13/hol-light/blob/master/Multivariate/integration.ml
[3] https://isa-afp.org/entries/Fourier.html

lemma *frontier-convex-hull-subset-path-image*:
fixes $g :: real \Rightarrow complex$
assumes *simple-path g pathfinish g = pathstart g*
path-image g ⊆ frontier (convex hull (path-image g))
shows *frontier (convex hull path-image g) ⊆ path-image g*
proof −
— The interior of the convex hull is connected, bounded, and disjoint from *path-image g*
have *interior (convex hull (path-image g)) ∩ path-image g = {}*
using *assms(3) frontier-def* **by** *auto*
moreover have *connected (interior (convex hull (path-image g)))*
by (*simp add*: *convex-connected*)
moreover have *inside (path-image g) ⊆ convex hull path-image g*
by (*metis Un-subset-iff compl-le-swap2 convex-convex-hull hull-subset outside-subset-convex union-with-inside*)
ultimately have *int-inside*: *interior (convex hull (path-image g)) ⊆ inside (path-image g)*
using *Jordan-inside-outside*[*OF* ‹*simple-path g*›] *assms*
by (*smt* (*verit*) *connected-Int-frontier diff-shunt inf.commute interior-Int interior-eq le-iff-inf*)
have − *(convex hull (path-image g)) ⊆ outside (path-image g)*
by (*simp add*: *hull-subset outside-subset-convex*)
hence *inside-sub*: *inside (path-image g) ⊆ convex hull (path-image g)*
by (*metis Un-subset-iff compl-le-swap2 union-with-inside*)
— Since *inside (path-image g)* is open and contained in the convex hull, it lies in the interior of the hull
have *inside (path-image g) ⊆ interior (convex hull (path-image g))*
by (*simp add*: *Jordan-inside-outside assms inside-sub interior-maximal*)
with *assms* **show** *?thesis*
using *frontier-convex-hull-eq-path-image int-inside* **by** *auto*
qed

locale *W* =
fixes $f\ f' :: real \Rightarrow real$ **and** *a*::*real*
assumes $f'hsd$: $\bigwedge x.\ x \in \{0..2*pi\} \Longrightarrow (f'\ has\text{-}integral\ (f\ x - f\ 0))\ \{0..x\}$
and *feq*: $f\ (2*pi) = f\ 0$
and *f0*: $(f\ has\text{-}integral\ 0)\ \{0..2*pi\}$
and $f'2$: $(\lambda x.\ (f'\ x)^2)\ integrable\text{-}on\ \{0..2*pi\}$
and *a*: $0 \le a\ a < pi\ f\ (a + pi) = f\ a$

begin

definition *g* **where** $g \equiv \lambda x.\ (f\ x - f\ a)^2\ /\ tan\ (x - a)$
definition g' **where** $g' \equiv \lambda x.\ (f'\ x)^2 - (f\ x - f\ a)^2 - (f'\ x - (f\ x - f\ a)\ /\ tan\ (x - a))^2$

lemma f': ‹f' *integrable-on* $\{0..2*pi\}$›
using $f'hsd$ [*of* ‹$2*pi$›] **by** *fastforce*

lemma *f'abs*: ‹*f' absolutely-integrable-on {0..2∗pi}*›
proof (*intro f' absolutely-integrable-integrable-bound*)
show ‹*norm* (*f' x*) ≤ *1* + (*f' x*)²› **for** *x*
proof −
have ‹*0* ≤ (*1* − *f' x*)²› **and** ‹*0* ≤ (*1* + *f' x*)²›
by (*auto simp*: *power2-eq-square*)
then show *?thesis*
by (*auto simp*: *power2-eq-square abs-le-iff algebra-simps*)
qed
show ‹(λ*x*. *1* + (*f' x*)²) *integrable-on* {*0..2∗pi*}›
using *integrable-add* [*OF integrable-const-ivl f'2*] **by** *simp*
qed

lemma *contf*: ‹*continuous-on* {*0..2∗pi*} *f*›
proof (*rule continuous-on-eq*)
show ‹*continuous-on* {*0..2∗pi*} (λ*x*. *integral* {*0..x*} *f'* + *f* *0*)›
by (*intro continuous-on-add indefinite-integral-continuous-1* [*OF f'*] *continuous-on-const*)
show ‹⋀*x*. *x* ∈ {*0..2∗pi*} ⟹ *integral* {*0..x*} *f'* + *f* *0* = *f x*›
using *f'hsd* **by** (*auto simp*: *has-integral-integrable-integral*)
qed

The integral over completely trouble-free intervals.

lemma *trouble-free*: (*g'* *has-integral* *g d* − *g c*) {*c..d*}
if *c* ≤ *d*
and *sub-cd*: {*c..d*} ⊆ {*0..2∗pi*}
and *sin-nz*: ⋀*x*. *x* ∈ {*c..d*} ⟹ *sin* (*x* − *a*) ≠ *0*
for *c d*
proof −
have *f'-int*: ((λ*t*. *2* ∗ (*f t* − *f a*) ∗ *f' t*) *has-integral*
((*f x* − *f a*)² − (*f c* − *f a*)²)) {*c..x*}
if *xcd*: *x* ∈ {*c..d*} **for** *x*
proof −
have *c* ≤ *x* **and** *xd*: *x* ≤ *d* **and** *sub-cx*: {*c..x*} ⊆ {*0..2∗pi*}
using *xcd* ‹*c* ≤ *d*› *sub-cd* **by** *auto*
have *ac-f*: *absolutely-continuous-on* {*0..2∗pi*} *f*
using *absolute-integral-absolutely-continuous-derivative-eq f'abs f'hsd* **by** *blast*
have *ac-sq*: *absolutely-continuous-on* {*c..x*} (λ*t*. (*f t* − *f a*)²)
unfolding *power2-eq-square* **using** *absolutely-continuous-on-subset ac-f sub-cx*

by(*intro continuous-intros*) *fastforce+*
obtain *k* **where** *negk*: *negligible k*
and *derivf*: ⋀*t*. *t* ∈ {*0..2∗pi*} − *k* ⟹
((λ*u*. *integral* {*0..u*} *f'*) *has-vector-derivative f' t*) (*at t within* {*0..2∗pi*})
using *f' has-vector-derivative-indefinite-integral* **by** *blast*
have *deriv-sq*: ((λ*t*. (*f t* − *f a*)²) *has-vector-derivative* *2* ∗ (*f t* − *f a*) ∗ *f' t*) (*at t within* {*c..x*})
if *t* ∈ {*c..x*} − *k* **for** *t*
proof −

```
    have hvd-int: ((λu. integral {0..u} f') has-vector-derivative f' t) (at t within
{0..2*pi})
      using derivf that sub-cx by auto
    have ((λu. f u − f 0) has-vector-derivative f' t) (at t within {0..2*pi})
    proof (rule has-vector-derivative-transform-within[OF hvd-int])
      fix u assume u ∈ {0..2*pi} dist u t < 1
      then show integral {0..u} f' = f u − f 0
        using f'hsd by blast
    qed (use that sub-cx in auto)
    then have fderiv: (f has-vector-derivative f' t) (at t within {c..x})
        using has-vector-derivative-diff-const has-vector-derivative-within-subset
sub-cx by blast
    then show ?thesis
      unfolding power2-eq-square has-vector-derivative-def
      by − (rule derivative-eq-intros | simp add: algebra-simps)+
  qed
  show ?thesis
    using fundamental-theorem-of-calculus-absolutely-continuous [OF negk ‹c ≤
x› ac-sq deriv-sq] by simp
 qed
```

Apply integration by parts with

- $\lambda x.\ (f\ x - f\ a)š$ and its derivative $\lambda x.\ 2 * (f\ x - f\ a) * f'\ x$
- $\lambda x.\ inverse\ (tan\ (x - a))$ and its derivative $\lambda x.\ -\ inverse\ (sin\ (x - a))š$

```
 have ibp-int: ((λx. (f x − f a)^2 * (− inverse ((sin (x − a))^2)) +
    2 * (f x − f a) * f' x * inverse (tan (x − a)))
    has-integral ((f y − f a)^2 * inverse (tan (y − a)) −
                  (f c − f a)^2 * inverse (tan (c − a)))) {c..y}
  if y ∈ {c..d} for y
 proof (intro f'-int absolute-real-integration-by-parts-sum(2))
  have f'-abs-cd: f' absolutely-integrable-on {c..d}
    using absolutely-integrable-on-subinterval[OF f'abs sub-cd] .
  have cont-ffa: continuous-on {c..d} (λx. 2 * (f x − f a))
    using sub-cd by (intro continuous-intros continuous-on-subset [OF contf])
auto
  have meas: (λx. 2 * (f x − f a)) ∈ borel-measurable (lebesgue-on {c..d})
    using cont-ffa by (intro continuous-imp-measurable-on-sets-lebesgue) auto
  have bdd: bounded ((λx. 2 * (f x − f a)) ‘ {c..d})
    using cont-ffa compact-Icc compact-continuous-image compact-imp-bounded
by blast
  show (λx. 2 * (f x − f a) * f' x) absolutely-integrable-on {c..d}
   using absolutely-integrable-bounded-measurable-product-real bdd f'-abs-cd meas
by auto
  show (λx. − inverse ((sin (x − a))^2)) absolutely-integrable-on {c..d}
    by (intro absolutely-integrable-continuous-real continuous-intros) (use sin-nz
in auto)
```

```
    show ((λt. − inverse ((sin (t−a))²)) has-integral
          (inverse (tan (x − a)) − inverse (tan (c − a)))) {c..x}
      if x ∈ {c..d} for x
    proof −
      have c ≤ x and sub-cx: {c..x} ⊆ {c..d}
        using that by auto
      have inv-tan-eq: inverse (tan (t−a)) = cos (t−a) / sin (t−a)
        if t ∈ {c..x} for t
        by (simp add: Multiseries-Expansion.tan-conv-sin-cos)
      have deriv: ((λt. cos (t−a) / sin (t−a)) has-vector-derivative
                   − inverse ((sin (t−a))²)) (at t within {c..x})
        if t ∈ {c..x} for t
      proof −
        have sin-nz-t: sin (t−a) ≠ 0 using sin-nz that sub-cx by auto
        have ((λt. cos (t−a) / sin (t−a)) has-real-derivative
              (− sin (t−a) * sin (t−a) − cos (t−a) * cos (t−a)) / (sin (t−a) * sin
(t−a)))
              (at t within {c..x})
          by (intro derivative-eq-intros | simp add: sin-nz-t)+
        also have (− sin (t−a) * sin (t−a) − cos (t−a) * cos (t−a)) / (sin (t−a)
* sin (t−a))
                  = − inverse ((sin (t−a))²)
          using sin-cos-squared-add3 [of t−a]
          by (simp (no-asm-simp) add: divide-simps power2-eq-square)
        finally show ?thesis
          by (simp add: has-real-derivative-iff-has-vector-derivative)
      qed
        — Apply FTC
      show ?thesis
        using fundamental-theorem-of-calculus[OF - deriv] inv-tan-eq inv-tan-eq ‹c
≤ x›
        by simp
    qed
    show y ∈ {c..d} using that .
  qed (use ‹c ≤ d› in auto)
  have integrand-eq: (f x − f a)² * (− inverse ((sin (x − a))²)) +
      2 * (f x − f a) * f′ x * inverse (tan (x − a)) = g′ x
    if x ∈ {c..d} for x
  proof −
    have snz: sin (x − a) ≠ 0 (sin (x − a))² ≠ 0 using sin-nz that by auto
    let ?F = f x − f a
    let ?s = sin (x − a)
    let ?c = cos (x − a)
    have sc1: ?s² + ?c² = 1
      using sin-cos-squared-add[of x − a] by simp
    have ((f x − f a)² * (− inverse (?s²)) + 2 * (f x − f a) * f′ x * inverse (tan
(x − a))) * ?s²
          = − ?F² + 2 * ?F * f′ x * ?c * ?s
      using snz by (simp add: tan-def field-simps power2-eq-square)
```

```
    moreover have g' x * ?s² = ((f' x)² - ?F²) * ?s² - (f' x * ?s - ?F * ?c)²
      using snz by (simp add: g'-def tan-def field-simps)
    moreover have ...  = - ?F² + 2 * ?F * f' x * ?c * ?s
      using sc1 by algebra
    ultimately show ?thesis using snz
      by (metis mult-right-cancel)
  qed
    — Combine using has-integral-eq
  show ?thesis
    using has-integral-eq ibp-int integrand-eq unfolding g-def divide-inverse
    by (metis (no-types, lifting) atLeastAtMost-iff order.refl that(1))
qed
```

Continuity of g.

```
lemma g-cont: continuous-on {0..2*pi} g
  unfolding continuous-on-eq-continuous-within
proof
  fix c assume c-in: c ∈ {0..2*pi}
  show continuous (at c within {0..2*pi}) g
  proof (cases sin (c - a) = 0)
    case False
      — When sin(c − a) ≠ 0, g is a quotient of continuous functions.
    have g-eq: g x = (f x - f a)² * cos (x - a) / sin (x - a) for x
      unfolding g-def tan-def by (simp add: field-simps)
    have continuous (at c within {0..2*pi}) f
      using contf c-in continuous-on-eq-continuous-within by blast
    then show ?thesis unfolding g-eq
      using False by (auto simp: continuous-intros)
  next
    case True
      — When sin(c − a) = 0, we have g(c) = 0 and need to show g(x) → 0.
    then obtain n :: int where npi: c - a = of-int n * pi
      using sin-zero-iff-int2 by auto
    have of-int n ≥ -a / pi of-int n ≤ (2 * pi - a) / pi
      using a npi c-in pi-gt-zero by (simp-all add: field-simps)
    moreover have -a / pi > -1 using a pi-gt-zero by (simp add: field-simps)
    moreover have (2 * pi - a) / pi < 3
      using a pi-gt-zero by (auto simp: divide-simps)
    ultimately have n = 0 ∨ n = 1 ∨ n = 2
      by linarith
    then have fca: f c = f a
    proof (elim disjE)
      assume n = 2
      then have c = a + 2 * pi using npi by (simp add: algebra-simps)
      with c-in a pi-gt-zero have a = 0 by auto
      thus f c = f a using ‹c = a + 2 * pi› feq by simp
    qed (use npi a in ‹auto simp: algebra-simps›)
    have gc0: g c = 0
      unfolding g-def using fca by simp
```

— Derive $\tan(x-a) = \tan(x-c)$ from $\sin(c-a) = 0$.
from *True* **obtain** *n* :: *int* **where** *npi*: *c − a = of-int n ∗ pi*
using *sin-zero-iff-int2* **by** *auto*
have *tan-eq*: *tan (x − a) = tan (x − c)* **for** *x*
by (*metis npi diff-add-cancel diff-diff-eq2 tan-periodic-int*)
have *g-eq2*: *g x = (f x − f c)² ∗ cos (x − c) / sin (x − c)* **for** *x*
unfolding *g-def* **by** (*metis fca divide-divide-eq-right local.tan-eq tan-def*)
— Cauchy–Schwarz bound: $(f(x)-f(c))^2 \le |x-c| \cdot \int_c^x (f')^2$.
have *cs-bound*: *(f x − f c)² ≤ |x − c| ∗ integral {min c x..max c x} (λt. (f′ t)²)*
if *xin*: *x ∈ {0..2∗pi}* **for** *x*
proof −
have *f′-int-sub*: *f′ integrable-on {a..b}* **if** *{a..b} ⊆ {0..2∗pi}* **for** *a b*
using *integrable-subinterval-real*[*OF set-lebesgue-integral-eq-integral*(*1*)[*OF f′abs*] *that*] **.**
have *f′2-int-sub*: *(λt. (f′ t)²) integrable-on {a..b}* **if** *{a..b} ⊆ {0..2∗pi}* **for** *a b*
using *integrable-subinterval-real*[*OF f′2 that*] **.**
— Helper: the FTC gives $f(b)-f(a) = \int_a^b f'$ for $a, b \in \{0..2\pi\}$
have *ftc-sub*: *f b − f a = integral {a..b} f′*
if *a ∈ {0..2∗pi} b ∈ {0..2∗pi} a ≤ b* **for** *a b*
proof −
have *integral {0..a} f′ + integral {a..b} f′ = integral {0..b} f′*
by (*meson Henstock-Kurzweil-Integration.integral-combine atLeastAtMost-iff f′hsd*
has-integral-integrable that)
moreover have *integral {0..a} f′ = f a − f 0* **and** *integral {0..b} f′ = f b − f 0*
using *f′hsd that* **by** (*auto simp*: *has-integral-integrable-integral*)
ultimately show *?thesis* **by** *linarith*
qed
— Helper: Cauchy–Schwarz $\left(\int_I f'\right)^2 \le (b-a)\cdot\int_I (f')^2$ for $I = \{a..b\} \subseteq \{0..2\pi\}$
have *cs-sub*: *(integral {a..b} f′)² ≤ (b−a) ∗ integral {a..b} (λt. (f′ t)²)*
if *sub*: *{a..b} ⊆ {0..2∗pi}* **and** *ab*: *a < b* **for** *a b*
proof −
define *μ* **where** *μ ≡ integral {a..b} f′ / (b−a)*
have *f′2I*: *(λt. (f′ t)²) integrable-on {a..b}* **by** (*rule f′2-int-sub*[*OF sub*])
have *int1*: *(λt. − (2 ∗ μ) ∗ f′ t) integrable-on {a..b}*
using *f′-int-sub integrable-on-mult-right sub* **by** *blast*
have *0 ≤ integral {a..b} (λt. (f′ t − μ)²)*
by (*metis eq-integralD has-integral-nonneg order-refl zero-le-power2*)
also have *integral {a..b} (λt. (f′ t − μ)²) =*
integral {a..b} (λt. (f′ t)²) + (− (2 ∗ μ) ∗ integral {a..b} f′ + μ² ∗ (b−a))
proof −
have *integral {a..b} (λt. (f′ t − μ)²) =*
integral {a..b} (λt. (f′ t)² + (− (2 ∗ μ) ∗ f′ t + μ²))
by (*simp add*: *power2-eq-square algebra-simps*)
also have *... = integral {a..b} (λt. (f′ t)²) + integral {a..b} (λt. − (2 ∗*

```
μ) * f' t + μ^2)
        by (rule integral-add[OF f'2I integrable-add[OF int1 integrable-const-ivl]])
        also have integral {a..b} (λt. − (2 * μ) * f' t + μ^2) =
              integral {a..b} (λt. − (2 * μ) * f' t) + integral {a..b} (λt. μ^2)
          by (rule integral-add[OF int1 integrable-const-ivl])
        finally show ?thesis using ab by simp
      qed
      also have − (2 * μ) * integral {a..b} f' + μ^2 * (b−a) = − (integral {a..b}
f')^2 / (b−a)
        using ab unfolding μ-def by (simp add: power2-eq-square divide-simps)
     finally show ?thesis using ab by (simp add: pos-divide-le-eq mult.commute)
    qed
    show ?thesis
    proof (cases c ≤ x)
      case True
      show ?thesis using cs-sub ‹c ≤ x› c-in xin ftc-sub by fastforce
    next
      case False
      hence §: x < c {x..c} ⊆ {0..2*pi} using c-in xin by auto
      then show ?thesis
        by (simp add: cs-sub ftc-sub power2-commute)
    qed
  qed
    — The integral of (f')^2 over a shrinking interval tends to 0.
  define F where F ≡ λx. integral {0..x} (λt. (f' t)^2)
  have F-cont: continuous-on {0..2*pi} F
    unfolding F-def by (rule indefinite-integral-continuous-1[OF f'2])
  have F-eq: integral {min c x..max c x} (λt. (f' t)^2) = |F x − F c|
    if x ∈ {0..2*pi} for x
  proof (cases c ≤ x)
    case True
    have sub: {c..x} ⊆ {0..2*pi} using c-in that True by auto
    have integral {0..c} (λt. (f' t)^2) + integral {c..x} (λt. (f' t)^2) =
            integral {0..x} (λt. (f' t)^2)
      by (metis Henstock-Kurzweil-Integration.integral-combine True atLeastat-
Most-subset-iff f'2
          integrable-on-subinterval order-refl sub)
    hence integral {c..x} (λt. (f' t)^2) = F x − F c
      unfolding F-def by linarith
    moreover have 0 ≤ integral {c..x} (λt. (f' t)^2)
      by (metis integral-nonneg not-integrable-integral order.refl zero-le-power2)
    ultimately show ?thesis using True by (simp add: min-def max-def)
  next
    case False
    hence xc: x ≤ c by simp
    have sub: {x..c} ⊆ {0..2*pi} using c-in that xc by auto
    have integral {0..x} (λt. (f' t)^2) + integral {x..c} (λt. (f' t)^2)
            = integral {0..c} (λt. (f' t)^2)
    by (metis Henstock-Kurzweil-Integration.integral-combine atLeastatMost-subset-iff
```

```
f'2
        integrable-subinterval-real order.refl sub xc)
    moreover have 0 ≤ integral {x..c} (λt. (f' t)²)
      by (metis integral-nonneg not-integrable-integral order.refl zero-le-power2)
    ultimately show ?thesis using xc by (simp add: F-def min-def max-def)
  qed
  have ((λx. |F x − F c|) ⟶ 0) (at c within {0..2*pi})
    by (metis F-cont LIM-zero-iff c-in continuous-on-def tendsto-rabs-zero)
  then have f'2-int-tends-0: ((λx. integral {min c x..max c x} (λt. (f' t)²)) ⟶
0) (at c within {0..2*pi})
    by (smt (verit, best) F-eq Lim-cong-within)
  have ∀F x in at c. |x − c| / |sin (x − c)| < 2
    by real-asymp
  then have sinc-ratio-bounded:
    ∀F x in at c within {0..2*pi}. |x − c| / |sin (x − c)| ≤ 2
    by (metis (no-types, lifting) UNIV-I eventually-at-topological less-imp-le)
      — Now combine everything.
  show ?thesis
    unfolding continuous-within gc0
  proof (rule Lim-null-comparison[where g = λx. 2 * integral {min c x..max c
x} (λt. (f' t)²)])
    have mem: ∀F x in at c within {0..2*pi}. x ∈ {0..2*pi}
      unfolding at-within-def eventually-inf-principal by simp
   show ∀F x in at c within {0..2*pi}. norm (g x) ≤ 2 * integral {min c x..max
c x} (λt. (f' t)²)
  proof (rule eventually-mono[OF eventually-conj[OF sinc-ratio-bounded mem]])
    fix x assume H: |x − c| / |sin (x − c)| ≤ 2 ∧ x ∈ {0..2*pi}
    have |g x| = (f x − f c)² * |cos (x − c)| / |sin (x − c)|
      using g-eq2 by (simp add: abs-mult)
    also have ... ≤ (f x − f c)² * 1 / |sin (x − c)|
      by (meson abs-cos-le-one abs-ge-zero divide-right-mono
          ordered-comm-semiring-class.comm-mult-left-mono zero-le-power2)
    also have ... = (f x − f c)² / |sin (x − c)| by simp
    also have ... ≤ 2 * integral {min c x..max c x} (λt. (f' t)²)
    proof −
      have sub: {min c x..max c x} ⊆ {0..2*pi} using c-in H by auto
      have f'2I: (λt. (f' t)²) integrable-on {min c x..max c x}
        by (rule integrable-subinterval-real[OF f'2 sub])
      show ?thesis
      proof (cases sin (x − c) = 0)
        case False
        define I where I = integral {min c x..max c x} (λt. (f' t)²)
        have Ige: I ≥ 0
          unfolding I-def by (rule integral-nonneg[OF f'2I]) auto
        have (f x − f c)² / |sin (x − c)| ≤ |x − c| * I / |sin (x − c)|
          using H by (simp add: I-def cs-bound divide-right-mono)
        also have ... = (|x − c| / |sin (x − c)|) * I
          by (simp add: field-simps)
        also have ... ≤ 2 * I
```

```
        by (meson H Ige mult-mono order.refl zero-le-numeral)
       finally show ?thesis unfolding I-def .
     qed (use integral-nonneg[OF f'2I] in auto)
   qed
   finally show norm (g x) ≤ 2 * integral {min c x..max c x} (λt. (f' t)²) by
simp
  qed
  show ((λx. 2 * integral {min c x..max c x} (λt. (f' t)²)) ⟶ 0) (at c within
{0..2*pi})
   using tendsto-mult-right-zero[OF f'2-int-tends-0] by simp
 qed
qed
qed
```

The integral over mainly trouble-free intervals: we only need $sin(x - a) \neq 0$ on the open interior, allowing zeros at the endpoints.

```
lemma mainly-trouble-free:
 assumes c ≤ d and cd-sub: {c..d} ⊆ {0..2*pi} and sin-nz: ⋀x. x ∈ {c<..<d}
⟹ sin (x - a) ≠ 0
 shows (g' has-integral g d - g c) {c..d}
proof -
 have g' absolutely-integrable-on {c..d}
 proof -
  have f'2-abs: (λx. (f' x)²) absolutely-integrable-on {0..2*pi}
   by (simp add: absolutely-integrable-on-def f'2)
  have g'-int-sub: g' integrable-on {u..v} if {u..v} ⊆ {c<..<d} for u v
  proof (cases u ≤ v)
   case True
   show ?thesis
    using has-integral-integrable[OF trouble-free[OF True]] sin-nz that cd-sub by
force
  qed (simp add: not-le integrable-on-empty)
  have g'-int: g' integrable-on {c'..d'} if {c'..d'} ⊆ {c<..<d} for c' d'
   using ‹{c'..d'} ⊆ {c<..<d}› g'-int-sub by blast
  have abs-g-cont: ‹continuous-on {0..2 * pi} (λx. |g x|)›
   by (intro continuous-intros g-cont)
  obtain h where h-abs: h absolutely-integrable-on {c..d}
   and h-bounded: (∀x∈{c..d}. g' x ≤ h x) ∨ (∀x∈{c..d}. h x ≤ g' x)
    using absolutely-integrable-on-subinterval[OF f'2-abs cd-sub] by (simp add:
g'-def)
  show ?thesis
  proof (intro g'-int absolutely-integrable-improper [of c d , unfolded box-real])
   obtain w where 0 ≤ w w ≤ 2*pi and w: ∀y. 0≤y ⟶ y ≤ 2*pi ⟶ |g y|
≤ |g w|
    using continuous-attains-sup [of ‹{0..2*pi}› ‹λx. |g x|›]
   by (metis Arg2pi abs-g-cont atLeastAtMost-iff compact-Icc empty-iff less-eq-real-def)
   show bounded {integral {c'..d'} g' |c' d'. {c'..d'} ⊆ {c<..<d}}
   proof (rule boundedI)
    fix x assume x ∈ {integral {c'..d'} g' |c' d'. {c'..d'} ⊆ {c<..<d}}
```

```
      then obtain c' d' where cd': {c'..d'} ⊆ {c<..<d} and xeq: x = integral
{c'..d'} g'
        by auto
      show norm x ≤ 2 * |g w|
      proof (cases c' ≤ d')
        case True
        have sub-2pi: {c'..d'} ⊆ {0..2*pi}
          using cd' cd-sub greaterThanLessThan-subseteq-atLeastAtMost-iff by
blast
        have sin (t-a) ≠ 0 if t ∈ {c'..d'} for t
       using that cd' sin-nz by (meson greaterThanLessThan-subseteq-atLeastAtMost-iff
subsetD)
        then have integral {c'..d'} g' = g d' - g c'
          using True sub-2pi trouble-free by blast
        then have |x| ≤ |g d'| + |g c'|
          using xeq by linarith
        also have ... ≤ |g w| + |g w|
         by (metis True w add-mono atLeastatMost-subset-iff order-trans sub-2pi)
        also have ... = 2 * |g w| by algebra
        finally show ?thesis by (simp add: xeq)
      qed (simp add: xeq)
    qed
  qed (use h-abs h-bounded in auto)
 qed
 have g'-int: g' integrable-on {c..d}
   using ‹g' absolutely-integrable-on {c..d}› set-lebesgue-integral-eq-integral by
blast
 have g-cont-cd: continuous-on {c..d} g
  using continuous-on-subset[OF g-cont cd-sub] .
 have integral {c..d} g' = g d - g c
 proof (cases c < d)
  case False with ‹c ≤ d› show ?thesis by simp
 next
  case True
   — Pick sequences c_n → c and d_n → d from inside (c, d)
  define c-n where c-n ≡ λn. c + (d - c) / (real n + 2)
  define d-n where d-n ≡ λn. d - (d - c) / (real n + 2)
  have pos: 0 < (d - c) / (real n + 2) for n
   using True by auto
  have lt-dc: (d - c) / (real n + 2) < d - c for n
   using True by (simp add: divide-less-eq)
  have c-n-le-d-n: c-n n ≤ d-n n for n
  proof -
   have c * real n ≤ d * real n
    using True by (intro mult-right-mono) auto
   then have 2 * ((d - c) / (real n + 2)) ≤ d - c
    using True by (simp add: field-simps)
   then show ?thesis unfolding c-n-def d-n-def by linarith
  qed
```

have *frac-lim*: $(\lambda n.\ (d - c) / (real\ n + 2)) \longrightarrow 0$
proof (*rule real-tendsto-sandwich*)
show $\forall_F\ n$ *in sequentially.* $0 \leq (d - c) / (real\ n + 2)$
using *True* **by** (*intro always-eventually allI*) (*auto simp*: *field-simps*)
show $\forall_F\ n$ *in sequentially.* $(d - c) / (real\ n + 2) \leq (d - c) * (1 / real\ n)$
using *True* **by** (*intro eventually-sequentiallyI*[*of 1*]) (*auto simp*: *field-simps*)
show $(\lambda\text{-}.\ (0::real)) \longrightarrow 0$ **by** *simp*
show $(\lambda n.\ (d - c) * (1 / real\ n)) \longrightarrow 0$
using *tendsto-mult-right-zero*[*OF lim-inverse-n'*] **by** *simp*
qed
have *c-n-lim*: *c-n* $\longrightarrow c$
unfolding *c-n-def* **using** *tendsto-add*[*OF tendsto-const frac-lim*] **by** *simp*
have *d-n-lim*: *d-n* $\longrightarrow d$
unfolding *d-n-def* **using** *tendsto-diff*[*OF tendsto-const frac-lim*] **by** *simp*
— On each $\{c_n..d_n\}$, *trouble-free* applies
have *c-n-in*: *c-n* $n \in \{c<..<d\}$ **and** *d-n-in*: *d-n* $n \in \{c<..<d\}$ **for** n
using *pos*[*of n*] *lt-dc*[*of n*] **unfolding** *c-n-def d-n-def* **by** *auto*
have *sub-n*: $\{c\text{-}n\ n..d\text{-}n\ n\} \subseteq \{c<..<d\}$ **for** n
using *c-n-in*[*of n*] *d-n-in*[*of n*] *c-n-le-d-n*[*of n*] **by** *auto*
have *sub-2pi-n*: $\{c\text{-}n\ n..d\text{-}n\ n\} \subseteq \{0..2*pi\}$ **for** n
using *sub-n*[*of n*] *cd-sub greaterThanLessThan-subseteq-atLeastAtMost-iff* **by** *blast*
have *sin-nz-n*: $sin\ (x - a) \neq 0$ **if** $x \in \{c\text{-}n\ n..d\text{-}n\ n\}$ **for** $n\ x$
using *that sub-n*[*of n*] *sin-nz*
by (*meson greaterThanLessThan-subseteq-atLeastAtMost-iff subsetD*)
have *tf-n*: $(g'\ has\text{-}integral\ g\ (d\text{-}n\ n) - g\ (c\text{-}n\ n))\ \{c\text{-}n\ n..d\text{-}n\ n\}$ **for** n
using *trouble-free*[*OF c-n-le-d-n sub-2pi-n sin-nz-n*] **.**
have *int-n*: $integral\ \{c\text{-}n\ n..d\text{-}n\ n\}\ g' = g\ (d\text{-}n\ n) - g\ (c\text{-}n\ n)$ **for** n
using *tf-n*[*of n*] **by** (*rule integral-unique*)
have *int-lim*: $(\lambda n.\ integral\ \{c\text{-}n\ n..d\text{-}n\ n\}\ g') \longrightarrow integral\ \{c..d\}\ g'$
proof −
have *indef-cont*: *continuous-on* $\{c..d\}$ $(\lambda x.\ integral\ \{c..x\}\ g')$
by (*rule indefinite-integral-continuous-1*[*OF g'-int*])
have *c-n-cd*: *c-n* $n \in \{c..d\}$ **for** n
using *c-n-in*[*of n*] **by** (*meson atLeastAtMost-iff greaterThanLessThan-iff less-imp-le*)
have *d-n-cd*: *d-n* $n \in \{c..d\}$ **for** n
using *d-n-in*[*of n*] **by** (*meson atLeastAtMost-iff greaterThanLessThan-iff less-imp-le*)
have *split*: $integral\ \{c\text{-}n\ n..d\text{-}n\ n\}\ g' = integral\ \{c..d\text{-}n\ n\}\ g' - integral\ \{c..c\text{-}n\ n\}\ g'$ **for** n
proof −
have *cn-le*: $c \leq c\text{-}n\ n$ **using** *c-n-in*[*of n*] **by** *auto*
have *int-cdn*: g' *integrable-on* $\{c..d\text{-}n\ n\}$
by (*rule integrable-subinterval-real*[*OF g'-int*]) (*use d-n-cd*[*of n*] ‹$c \leq d$› **in** *auto*)
have $integral\ \{c..c\text{-}n\ n\}\ g' + integral\ \{c\text{-}n\ n..d\text{-}n\ n\}\ g' = integral\ \{c..d\text{-}n\ n\}\ g'$
by (*rule Henstock-Kurzweil-Integration.integral-combine*[*OF cn-le c-n-le-d-n*

int-cdn])
then show *?thesis* **by** *linarith*
qed
have ($\lambda n.$ *integral* $\{c..d\text{-}n\ n\}$ g') $\longrightarrow$ *integral* $\{c..d\}$ g'
by (*rule continuous-on-tendsto-compose*[*OF indef-cont d-n-lim*])
(*use d-n-cd* ‹$c \leq d$› **in** ‹*auto intro*: *always-eventually*›)
moreover have ($\lambda n.$ *integral* $\{c..c\text{-}n\ n\}$ g') $\longrightarrow$ *integral* $\{c..c\}$ g'
by (*rule continuous-on-tendsto-compose*[*OF indef-cont c-n-lim*])
(*use c-n-cd* ‹$c \leq d$› **in** ‹*auto intro*: *always-eventually*›)
moreover have *integral* $\{c..c\}$ $g' = 0$ **by** *simp*
ultimately have ($\lambda n.$ *integral* $\{c..d\text{-}n\ n\}$ g' − *integral* $\{c..c\text{-}n\ n\}$ g') $\longrightarrow$ *integral* $\{c..d\}$ g' − 0
by (*intro tendsto-diff*) *simp-all*
then show *?thesis* **using** *split* **by** *simp*
qed
moreover have ($\lambda n.$ g ($d\text{-}n$ n) − g ($c\text{-}n$ n)) $\longrightarrow$ $g\ d - g\ c$
proof (*intro tendsto-diff*)
obtain *d-n-cd*: $d\text{-}n\ n \in \{c..d\}$ **and** *c-n-cd*: $c\text{-}n\ n \in \{c..d\}$ **for** n
using *c-n-in d-n-in less-eq-real-def* **by** *force*
show ($\lambda n.$ g ($d\text{-}n$ n)) $\longrightarrow$ $g\ d$
by (*rule continuous-on-tendsto-compose*[*OF g-cont-cd d-n-lim*])
(*use d-n-cd* ‹$c \leq d$› **in** ‹*auto intro*: *always-eventually*›)
show ($\lambda n.$ g ($c\text{-}n$ n)) $\longrightarrow$ $g\ c$
by (*rule continuous-on-tendsto-compose*[*OF g-cont-cd c-n-lim*])
(*use c-n-cd* ‹$c \leq d$› **in** ‹*auto intro*: *always-eventually*›)
qed
ultimately show *?thesis*
using *int-n LIMSEQ-unique* **by** *auto*
qed
then show *?thesis*
using *integrable-integral*[*OF g'-int*] **by** *auto*
qed

end

theorem *Wirtinger-inequality*:
fixes $f\ f'$ :: *real* ⇒ *real*
assumes *f'hsd*: $\bigwedge x.\ x \in \{0..2*pi\} \Longrightarrow (f'$ *has-integral* $(f\ x - f\ 0))\ \{0..x\}$
and *feq*: $f\ (2*pi) = f\ 0$
and *f0*: $(f$ *has-integral* $0)\ \{0..2*pi\}$
and *f'2*: $(\lambda x.\ (f'\ x)^2)$ *integrable-on* $\{0..2*pi\}$
shows $(\lambda x.\ (f\ x)^2)$ *integrable-on* $\{0..2*pi\}$
and *integral* $\{0..2*pi\}$ $(\lambda x.\ (f\ x)^2) \leq$ *integral* $\{0..2*pi\}$ $(\lambda x.\ (f'\ x)^2)$
and *integral* $\{0..2*pi\}$ $(\lambda x.\ (f\ x)^2) =$ *integral* $\{0..2*pi\}$ $(\lambda x.\ (f'\ x)^2) \Longrightarrow$
$\exists\, c\ a.\ \forall\, x \in \{0..2*pi\}.\ f\ x = c * sin\ (x - a)$
proof −
obtain a **where** a: $0 \leq a\ a < pi\ f\ (a + pi) = f\ a$
proof −
have *contf*: ‹*continuous-on* $\{0..2*pi\}$ f›

```
    proof (rule continuous-on-eq)
      show ‹continuous-on {0..2*pi} (λx. integral {0..x} f' + f 0)›
        using f'hsd [of ‹2*pi›]
      by (intro continuous-on-add indefinite-integral-continuous-1  continuous-on-const)
auto
      show ‹⋀x. x ∈ {0..2*pi} ⟹ integral {0..x} f' + f 0 = f x›
        using f'hsd by (auto simp: has-integral-integrable-integral)
    qed
    define h where h ≡ λx. f (x + pi) − f x
    have hcont: continuous-on {0..pi} h
      unfolding h-def
      by (intro continuous-intros continuous-on-compose2 [OF contf]) auto
    have heq: h 0 + h pi = 0
      unfolding h-def using feq by simp
    have iv: is-interval (h ‘ {0..pi})
        using is-interval-connected-1 connected-continuous-image [OF hcont con-
nected-Icc]
      by blast
    haveh 0 ∈ h ‘ {0..pi}  h pi ∈ h ‘ {0..pi}
      using pi-gt-zero by auto
    with heq obtain a where a ∈ {0..pi} h a = 0
      by (smt (verit, best) imageE is-interval-1 iv)
    show thesis
    proof (cases a = pi)
      case True
      then have h 0 = 0 using heq ‹h a = 0› by auto
      then show thesis using that [of 0] pi-gt-zero by (auto simp: h-def)
    next
      case False
      then show thesis using that [of a] ‹a ∈ {0..pi}› ‹h a = 0›
        by (auto simp: h-def)
    qed
  qed

  interpret W f f' a
    using W.intro a assms by blast

  show (λx. (f x)^2) integrable-on {0..2*pi}
    by (intro integrable-continuous-interval continuous-on-power contf)

— Zeros of sine on this interval occur at a and a plus pi.
  have sin-nz-1: sin (x − a) ≠ 0 if a + pi < x x < 2*pi for x
    by (smt (verit) ‹0 ≤ a› sin-lt-zero that)
  have sin-nz-2: sin (x − a) ≠ 0 if a < x x < a + pi for x
    by (smt (verit, ccfv-threshold) sin-gt-zero that)
  have sin-nz-3: sin (x − a) ≠ 0 if 0 < x x < a for x
    using ‹a < pi› sin-zero-pi-iff that by auto
      — Apply mainly-trouble-free on three intervals.
  have int1: (g' has-integral g (2*pi) − g (a + pi)) {a + pi..2*pi}
```

by (*rule mainly-trouble-free*) (*use* ‹*0* ≤ *a*› ‹*a* < *pi*› *sin-nz-1* **in** *auto*)
have *int2*: (*g′ has-integral g* (*a* + *pi*) − *g a*) {*a*..*a* + *pi*}
by (*rule mainly-trouble-free*) (*use* ‹*0* ≤ *a*› ‹*a* < *pi*› *sin-nz-2* **in** *auto*)
have *int3*: (*g′ has-integral g a* − *g 0*) {*0*..*a*}
by (*rule mainly-trouble-free*) (*use* ‹*0* ≤ *a*› ‹*a* < *pi*› *sin-nz-3* **in** *auto*)
have *api-le*: *a* ≤ *a* + *pi* **and** *api-le2*: *a* + *pi* ≤ *2∗pi*
using ‹*0* ≤ *a*› ‹*a* < *pi*› **by** *auto*
have *a-le-2pi*: *a* ≤ *2∗pi* **using** ‹*0* ≤ *a*› ‹*a* < *pi*› **by** *auto*
have *int12*: (*g′ has-integral* (*g* (*a* + *pi*) − *g a*) + (*g* (*2∗pi*) − *g* (*a* + *pi*))) {*a*..*2∗pi*}
by (*rule has-integral-combine*[*OF api-le api-le2 int2 int1*])
have *g* (*2∗pi*) = *g 0*
unfolding *g-def* **using** *feq* **by** (*simp add*: *tan-def*)
hence *g′-zero*: (*g′ has-integral 0*) {*0*..*2∗pi*}
using *has-integral-combine*[*OF* ‹*0* ≤ *a*› *a-le-2pi int3 int12*] **by** (*simp add*: *algebra-simps*)
— Extract the inequality from $\int g' = 0$.
have *ffa-int*: (λ*x*. (*f x* − *f a*)²) *integrable-on* {*0*..*2∗pi*}
by (*intro integrable-continuous-interval continuous-intros contf*)
have *diff-int*: ((λ*x*. (*f′ x*)² − *g′ x*) *has-integral integral* {*0*..*2∗pi*} (λ*x*. (*f′ x*)²) − *0*) {*0*..*2∗pi*}
by (*rule has-integral-diff*[*OF integrable-integral*[*OF f′2*] *g′-zero*])
have *diff-ge*: (*f′ x*)² − *g′ x* ≥ (*f x* − *f a*)² **for** *x*
by (*simp add*: *g′-def zero-le-power2*)
have *integral* {*0*..*2∗pi*} (λ*x*. (*f x* − *f a*)²) ≤ *integral* {*0*..*2∗pi*} (λ*x*. (*f′ x*)² − *g′ x*)
by (*rule integral-le*[*OF ffa-int*]) (*use diff-int has-integral-integrable-integral* **in** ‹*auto intro*: *diff-ge*›)
also have *int-eq*: ... = *integral* {*0*..*2∗pi*} (λ*x*. (*f′ x*)²)
using *diff-int has-integral-integrable-integral* **by** *auto*
finally have *ineq-ffa*: *integral* {*0*..*2∗pi*} (λ*x*. (*f x* − *f a*)²) ≤ *integral* {*0*..*2∗pi*} (λ*x*. (*f′ x*)²) **.**
— Show $\int f(x)^2 \le \int (f(x) - f(a))^2$ using $\int f = 0$.
have (*f x*)² ≤ (*f x* − *f a*)² + *2* ∗ *f a* ∗ *f x* − (*f a*)² **for** *x*
by (*simp add*: *power2-eq-square algebra-simps*)
have *fx2-int*: (λ*x*. (*f x*)²) *integrable-on* {*0*..*2∗pi*}
by (*intro integrable-continuous-interval continuous-intros contf*)
have *ffa-2fa-int*: (λ*x*. *2* ∗ *f a* ∗ *f x*) *integrable-on* {*0*..*2∗pi*}
using *integrable-continuous-interval*[*OF contf*] *integrable-on-mult-right* **by** *blast*
have *integral* {*0*..*2∗pi*} (λ*x*. (*f x* − *f a*)²) = *integral* {*0*..*2∗pi*} (λ*x*. (*f x*)² − *2* ∗ *f a* ∗ *f x* + (*f a*)²)
by (*simp add*: *power2-eq-square algebra-simps*)
also have ... = *integral* {*0*..*2∗pi*} (λ*x*. (*f x*)² − *2* ∗ *f a* ∗ *f x*) + *integral* {*0*..*2∗pi*} (λ*x*. (*f a*)²)
by (*rule Henstock-Kurzweil-Integration.integral-add*)
(*auto intro*: *integrable-diff fx2-int ffa-2fa-int*)
also have *integral* {*0*..*2∗pi*} (λ*x*. (*f x*)² − *2* ∗ *f a* ∗ *f x*) =
integral {*0*..*2∗pi*} (λ*x*. (*f x*)²) − *integral* {*0*..*2∗pi*} (λ*x*. *2* ∗ *f a* ∗ *f x*)
by (*rule Henstock-Kurzweil-Integration.integral-diff*[*OF fx2-int ffa-2fa-int*])

```
  also have integral {0..2*pi} (λx. 2 * f a * f x) = 0
    using integral-cmul f0 by (simp add: has-integral-integrable-integral)
  also have integral {0..2*pi} (λx. (f a)²) = (f a)² * (2*pi)
    by simp
  finally have ffa-eq: integral {0..2*pi} (λx. (f x − f a)²)
                  = integral {0..2*pi} (λx. (f x)²) + (f a)² * (2*pi)
    by linarith
  then have ffa-ineq': integral {0..2*pi} (λx. (f x)²) ≤ integral {0..2*pi} (λx. (f
x − f a)²)
    by auto
  thus integral {0..2*pi} (λx. (f x)²) ≤ integral {0..2*pi} (λx. (f' x)²)
    using ineq-ffa by linarith
  show ∃ c a. ∀ x ∈ {0..2*pi}. f x = c * sin (x − a)
    if integral {0..2*pi} (λx. (f x)²) = integral {0..2*pi} (λx. (f' x)²)
  proof −
    — From the equality, all intermediate inequalities are equalities.
    have fa0: f a = 0
     by (smt (verit, best) ffa-ineq' a ffa-eq ineq-ffa mult-eq-0-iff power-eq-0-iff that)
    — The "rest" term integrates to 0.
    define rest where rest ≡ λx. f' x − (f x − f a) / tan (x − a)
    have diff-int: (λx. (f' x)² − g' x) integrable-on {0..2*pi}
      using has-integral-diff[OF integrable-integral[OF f'2] g'-zero]
        has-integral-integrable by blast
    have rest-sq-int: (λx. (rest x)²) integrable-on {0..2*pi}
      using integrable-diff[OF diff-int ffa-int]
      by (simp add: g'-def rest-def)
    have rest-sq-zero: integral {0..2*pi} (λx. (rest x)²) = 0
      using integral-add[OF ffa-int rest-sq-int] that g'-def int-eq
      by (fastforce simp: rest-def fa0)
    — Integral of c sin(x − a) via the fundamental theorem of calculus.
    have csin-integral: integral {u..v} (λx. c * sin (x − a)) =
        c * (cos (u − a) − cos (v − a)) if u ≤ v for u v c
    proof −
      have ((λx. − (c * cos (x − a))) has-real-derivative c * sin (x − a)) (at x)
for x
        by (auto intro!: derivative-eq-intros simp: algebra-simps)
      hence hvd: ((λx. − (c * cos (x − a))) has-vector-derivative c * sin (x − a))
        (at x within {u..v}) for x
      by (meson has-real-derivative-iff-has-vector-derivative has-vector-derivative-at-within)
      hence ((λx. c * sin (x − a)) has-integral
        (− (c * cos (v − a)) − (− (c * cos (u − a))))) {u..v}
        using that by (intro fundamental-theorem-of-calculus) auto
      thus ?thesis
        by (simp add: has-integral-integrable-integral algebra-simps)
    qed
    — Key fact: on intervals where sin(x − a) ≠ 0, f equals c sin(x − a).
    have key-fact: ∃ c. ∀ x∈{u..v}. f x = c * sin (x − a)
      if huv: 0 ≤ u u < v v ≤ 2*pi
        and hsin: ⋀x. x ∈ {u<..<v} ⟹ sin (x − a) ≠ 0
```

for u v
proof −
— Open-interval version (to be proved later).
— Step 1: $\int_u^v rest^2 = 0$ from $\int_0^{2\pi} rest^2 = 0$ and nonnegativity.
have *rest-sq-sub*: $(\lambda x.\ (rest\ x)^2)$ *integrable-on* $\{u..v\}$
using *integrable-on-subinterval rest-sq-int that* **by** *fastforce*
have *integral* $\{u..v\}$ $(\lambda x.\ (rest\ x)^2) \leq$ *integral* $\{0..2{*}pi\}$ $(\lambda x.\ (rest\ x)^2)$
by (*simp add*: *huv integral-subset-le rest-sq-int rest-sq-sub*)
moreover have $0 \leq$ *integral* $\{u..v\}$ $(\lambda x.\ (rest\ x)^2)$
by (*simp add*: *Henstock-Kurzweil-Integration.integral-nonneg rest-sq-sub*)
ultimately have *rest-sq-sub-zero*: *integral* $\{u..v\}$ $(\lambda x.\ (rest\ x)^2) = 0$
using *rest-sq-zero* **by** *linarith*
— Step 2: $rest = 0$ a.e. on $\{u..v\}$ via Lebesgue theory.
have *rest-sq-abs*: $(\lambda x.\ (rest\ x)^2)$ *absolutely-integrable-on* $\{u..v\}$
by (*simp add*: *absolutely-integrable-on-iff-nonneg rest-sq-sub*)
have *rest-sq-leb*: *integrable* (*lebesgue-on* $\{u..v\}$) $(\lambda x.\ (rest\ x)^2)$
by (*simp add*: *absolutely-integrable-imp-integrable rest-sq-abs*)
have $integral^L$ (*lebesgue-on* $\{u..v\}$) $(\lambda x.\ (rest\ x)^2) =$ *integral* $\{u..v\}$ $(\lambda x.\ (rest\ x)^2)$
using *lebesgue-integral-eq-integral lmeasurable-interval*(*1*) *rest-sq-leb* **by** *blast*
hence $integral^L$ (*lebesgue-on* $\{u..v\}$) $(\lambda x.\ (rest\ x)^2) = 0$
using *rest-sq-sub-zero* **by** *simp*
hence *rest-ae-zero*: *AE x in lebesgue-on* $\{u..v\}$. $rest\ x = 0$
using *integral-nonneg-eq-0-iff-AE*[*OF rest-sq-leb*] **by** *simp*
— Step 3: $h(x) = f(x)/\sin(x-a)$ is constant on (u, v). For any $\{s..t\} \subseteq (u, v)$, h is absolutely continuous and $h' = rest/\sin$ a.e., so $h(t) - h(s) = \int_s^t rest/\sin = 0$.
have *h-const*: $f\ s\ /\ sin\ (s - a) = f\ t\ /\ sin\ (t-a)$
if *hst*: $s \in \{u<..<v\}$ $t \in \{u<..<v\}$ **for** s t
proof (*cases* $s = t$)
case *True* **thus** *?thesis* **by** *simp*
next
case *False*
— WLOG s < t
define s' **where** $s' = min\ s\ t$
define t' **where** $t' = max\ s\ t$
have *st'*: $u < s'$ $t' < v$ $s' < t'$
using *hst False* **unfolding** *s'-def t'-def* **by** *auto*
have *st'-sub*: $\{s'..t'\} \subseteq \{u<..<v\}$ **and** *st'-sub2*: $\{s'..t'\} \subseteq \{0..2{*}pi\}$
using *st' huv* **by** *auto*
— $\sin(x - a) \neq 0$ on $\{s'..t'\}$
have *sin-nz-st*: $sin\ (x - a) \neq 0$ **if** $x \in \{s'..t'\}$ **for** x
using *hsin st'-sub that* **by** *auto*
— $h = f/\sin$ is absolutely continuous on $\{s'..t'\}$
define h **where** $h \equiv \lambda x.\ f\ x\ /\ sin\ (x - a)$
have *ac-f*: *absolutely-continuous-on* $\{0..2{*}pi\}$ f
using *absolute-integral-absolutely-continuous-derivative-eq f'abs f'hsd* **by** *blast*
have *ac-f-st*: *absolutely-continuous-on* $\{s'..t'\}$ f

using *absolutely-continuous-on-subset*[*OF ac-f st′-sub2*] **.**
— $1/\sin(x-a)$ is absolutely continuous on $\{s'..t'\}$ via a Lipschitz bound
— The derivative $-\cos/\sin^2$ is bounded on $\{s'..t'\}$ since sin is bounded away from 0
define *deriv* **where** *deriv* ≡ λ*x*::*real*. − *cos* (*x* − *a*) / (*sin* (*x* − *a*))^2
have *cont-deriv*: *continuous-on* {*s′..t′*} *deriv*
unfolding *deriv-def*
by (*intro continuous-intros*) (*use sin-nz-st* **in** *auto*)
have *bdd*: *bounded* (*deriv* ‘ {*s′..t′*})
using *compact-continuous-image compact-imp-bounded cont-deriv* **by** *blast*
then obtain *B* **where** *B*: ⋀*x*. *x* ∈ {*s′..t′*} ⟹ |*deriv x*| ≤ *B*
by (*meson bounded-real imageI*)
have *lipschitz*: |*inverse* (*sin* (*x* − *a*)) − *inverse* (*sin* (*y* − *a*))| ≤ *B* ∗ |*x* − *y*|
if *hx*: *s′* ≤ *x x* ≤ *t′* **and** *hy*: *s′* ≤ *y y* ≤ *t′* **for** *x y*
proof −
have *deriv-at*: ((λ*x*. *inverse* (*sin* (*x* − *a*))) *has-real-derivative deriv z*)
(*at z within* {*s′..t′*})
if *hz*: *z* ∈ {*s′..t′*} **for** *z*
proof −
have ((λ*x*. *sin* (*x* − *a*)) *has-real-derivative cos* (*z* − *a*)) (*at z within* {*s′..t′*})
by (*intro derivative-eq-intros* | *simp*)+
moreover have − (*cos* (*z* − *a*) ∗ *inverse* (*sin* (*z* − *a*) ^ *Suc* (*Suc 0*))) = *deriv z*
unfolding *deriv-def power2-eq-square*
by (*simp add*: *field-simps*)
ultimately show *?thesis*
by (*metis DERIV-inverse-fun sin-nz-st*[*OF hz*])
qed
have *norm* (*inverse* (*sin* (*x* − *a*)) − *inverse* (*sin* (*y* − *a*))) ≤ *B* ∗ *norm* (*x* − *y*)
using *hx hy B deriv-at*
by (*intro field-differentiable-bound*[*OF convex-real-interval*(*5*)]) *auto*
then show *?thesis* **by** (*simp add*: *real-norm-def*)
qed
then have *ac-inv-sin*: *absolutely-continuous-on* {*s′..t′*} (λ*x*. *inverse* (*sin* (*x* − *a*)))
by (*intro Lipschitz-imp-absolutely-continuous strip*; *auto*)
— $h = f \cdot (1/\sin)$ is AC on $\{s'..t'\}$
have *ac-h*: *absolutely-continuous-on* {*s′..t′*} *h*
using *absolutely-continuous-on-mul*[*OF ac-f-st ac-inv-sin*]
by (*simp add*: *divide-real-def h-def*)
— *h* has derivative *rest*/sin a.e. on {*s′..t′*}
obtain *k* **where** *negk*: *negligible k*
and *derivf*: ⋀*t*. *t* ∈ {*0..2∗pi*} − *k* ⟹
((λ*u*. *integral* {*0..u*} *f′*) *has-vector-derivative f′ t*)
(*at t within* {*0..2∗pi*})
using *f′ has-vector-derivative-indefinite-integral* **by** *blast*

```
    have f-eq: f t = f 0 + integral {0..t} f' if t ∈ {0..2*pi} for t
      using f'hsd[OF that] by (auto simp: has-integral-integrable-integral)
    have fderiv: (f has-vector-derivative f' t) (at t within {s'..t'})
      if t ∈ {s'..t'} - k for t
    proof -
      have t02: t ∈ {0..2*pi} and t ∈ {0..2*pi} - k
        using that st'-sub2 by auto
      then have ((λu. f u - f 0) has-vector-derivative f' t)
                  (at t within {0..2*pi})
        using derivf has-vector-derivative-transform-within t02
        by (smt (verit, best) f-eq has-vector-derivative-transform)
      then have (f has-vector-derivative f' t) (at t within {0..2*pi})
        using has-vector-derivative-diff-const by blast
      then show ?thesis
        by (rule has-vector-derivative-within-subset) (use st'-sub2 in auto)
    qed
      — Derivative of h = f / sin via the quotient rule
    have hderiv: (h has-vector-derivative (f' t * sin (t-a) - f t * cos (t-a)) /
(sin (t-a))^2)
          (at t within {s'..t'})
      if t ∈ {s'..t'} - k for t
    proof -
      have fd: (f has-real-derivative f' t) (at t within {s'..t'})
     using fderiv that by (simp add: has-real-derivative-iff-has-vector-derivative)
      have sd: ((λx. sin (x - a)) has-real-derivative cos (t-a))
                (at t within {s'..t'})
        by (auto intro!: derivative-eq-intros)
      have ((λx. f x / sin (x - a)) has-real-derivative
              (f' t * sin (t-a) - f t * cos (t-a)) / (sin (t-a))^2)
             (at t within {s'..t'})
        using DERIV-quotient[OF fd sd] sin-nz-st that
        by (simp add: power2-eq-square algebra-simps)
      then show ?thesis unfolding h-def
        by (simp add: has-real-derivative-iff-has-vector-derivative)
    qed
      — The derivative of h equals rest / sin
    have hderiv-eq: (f' t * sin (t-a) - f t * cos (t-a)) / (sin (t-a))^2
                    = rest t / sin (t-a)
      if t ∈ {s'..t'} for t
      using that unfolding rest-def fa0
   by (simp add: power2-eq-square divide-simps Multiseries-Expansion.tan-conv-sin-cos)
   have hderiv': (h has-vector-derivative rest t / sin (t-a)) (at t within {s'..t'})
      if t ∈ {s'..t'} - k for t
      using hderiv[OF that] hderiv-eq[of t] that by auto
      — rest = 0 a.e. on {u..v}, so obtain a negligible set N
    obtain N where negN: negligible N and restN: ⋀x. x ∈ {u..v} - N ⟹
rest x = 0
    proof -
      from rest-ae-zero[unfolded eventually-ae-filter[of - lebesgue-on {u..v}]]
```

```
      obtain N0 where N0: N0 ∈ null-sets (lebesgue-on {u..v})
        and sub: {x ∈ space (lebesgue-on {u..v}). rest x ≠ 0} ⊆ N0
        by auto
      have negligible N0
        using null-sets-restrict-space[of {u..v}] N0 negligible-iff-null-sets
        by auto
      moreover have rest x = 0 if x ∈ {u..v} − N0 for x
        using sub that by (auto simp: space-lebesgue-on)
      ultimately show ?thesis using that by blast
    qed
      — h has derivative 0 a.e. on {s'..t'}
    have hderiv-zero: (h has-vector-derivative 0) (at t within {s'..t'})
      if t ∈ {s'..t'} − (k ∪ N) for t
      using restN[of t] that st'-sub hderiv' using st'(2) by fastforce
    have neg-kN: negligible (k ∪ N)
      using negk negN by (rule negligible-Un)
        — By the FTC for AC functions: h(t') − h(s') = ∫ 0 = 0
    have h t' − h s' = integral {s'..t'} (λx. 0::real)
      using fundamental-theorem-of-calculus-absolutely-continuous [OF neg-kN
- ac-h hderiv-zero]
      using st' by auto
    then have h s' = h t' by simp
        — Translate back to f / sin
    then show ?thesis
      unfolding h-def s'-def t'-def by (auto split: if-splits)
  qed
  obtain x where x ∈ {u<..<v}
    using huv(2) dense by (metis greaterThanLessThan-iff)
  with eq-divide-eq hsin h-const that  have open-ver: ∃ c. ∀ x∈{u<..<v}. f x =
c * sin (x − a)
    by metis
  then obtain c where hc: ∀ x∈{u<..<v}. f x = c * sin (x − a)
    by auto
      — Extend to the closed interval by continuity.
  have f x = c * sin (x − a) if x ∈ {u..v} for x
  proof −
    have f x − c * sin (x − a) = 0
     proof (rule continuous-constant-on-closure[of {u<..<v} λx. f x − c * sin
(x − a) 0])
      show continuous-on (closure {u<..<v}) (λx. f x − c * sin (x − a))
        unfolding closure-greaterThanLessThan[OF huv(2)]
        by (intro continuous-intros continuous-on-subset[OF contf])
          (use huv in auto)
      show ⋀y. y ∈ {u<..<v} ⟹ f y − c * sin (y − a) = 0
        using hc by simp
      show x ∈ closure {u<..<v}
       unfolding closure-greaterThanLessThan[OF huv(2)] using that by auto
    qed
    thus ?thesis by simp
```

```
    qed
    thus ?thesis by auto
  qed
  show ?thesis
  proof (cases a=0)
    case True
    obtain c1 where c1: ∀ x∈{0..pi}. f x = c1 * sin (x − a)
      using key-fact[of 0 pi] sin-nz-2 True pi-gt-zero by auto
    obtain c2 where c2: ∀ x∈{pi..2*pi}. f x = c2 * sin (x − a)
      using key-fact[of pi 2*pi] sin-nz-1 True pi-gt-zero by auto
      — Use ∫ f = 0 and csin-integral to show c1 = c2.
    have eq1: integral {0..pi} f = c1 * (cos (0 − a) − cos (pi − a))
      by (metis (lifting) integral-cong True add-0 api-le c1 csin-integral)
    have eq2: integral {pi..2*pi} f = c2 * (cos (pi − a) − cos (2*pi − a))
      by (metis (lifting) integral-cong True add-0 api-le2 c2 csin-integral)
    have int-split: integral {0..2*pi} f = integral {0..pi} f + integral {pi..2*pi}
f
      using Henstock-Kurzweil-Integration.integral-combine[OF pi-ge-zero]
      by (metis True add-cancel-left-left api-le2 assms(3) integrable-on-def)
    have integral {0..2*pi} f = 0
      using f0 by (simp add: has-integral-integrable-integral)
    hence c1 * (cos (0 − a) − cos (pi − a)) + c2 * (cos (pi − a) − cos (2*pi
− a)) = 0
      using int-split eq1 eq2 by linarith
    hence c1 = c2 using True
      by (simp add: cos-two-pi cos-pi)
    then show ?thesis
      by (metis atLeastAtMost-iff c1 c2 nle-le)
  next
    case False
    with ‹0 ≤ a› have a-pos: 0 < a by auto
      — Three intervals where sin(x − a) ≠ 0
    obtain c1 where c1: ∀ x∈{0..a}. f x = c1 * sin (x − a)
      using key-fact[of 0 a] sin-nz-3 a-pos ‹a < pi› by auto
    obtain c2 where c2: ∀ x∈{a..a+pi}. f x = c2 * sin (x − a)
      using key-fact[of a a+pi] sin-nz-2 a-pos ‹0 ≤ a› ‹a < pi› by auto
    obtain c3 where c3: ∀ x∈{a+pi..2*pi}. f x = c3 * sin (x − a)
      using key-fact[of a+pi 2*pi] sin-nz-1 ‹0 ≤ a› ‹a < pi› by auto
      — Show c1 = c3 using f(2π) = f(0)
    have sin-a-nz: sin a ≠ 0
      using sin-gt-zero[OF a-pos ‹a < pi›] by (simp add: less-imp-le)
    have f0-eq: f 0 = c1 * sin (0 − a)
      using c1 ‹0 ≤ a› by auto
    have f2pi-eq: f (2*pi) = c3 * sin (2*pi − a)
      using c3 ‹0 ≤ a› ‹a < pi› by auto
      — Compute integrals on each interval
    have eq1: integral {0..a} f = c1 * (cos (0 − a) − cos (a − a))
      by (metis (no-types, lifting) integral-cong ‹0 ≤ a› c1 csin-integral)
    have eq2: integral {a..a+pi} f = c2 * (cos (a − a) − cos ((a+pi) − a))
```

```
      by (metis (no-types, lifting) api-le integral-cong c2 csin-integral)
    have eq3: integral {a+pi..2*pi} f = c3 * (cos ((a+pi) - a) - cos (2*pi -
a))
      by (metis (mono-tags, lifting) integral-cong api-le2 c3 csin-integral)
       — Split the integral into three parts
    have f-int: f integrable-on {0..2*pi}
      using f0 has-integral-integrable by blast
    have a-le: a ≤ a + pi using pi-gt-zero by linarith
    have api-le: a + pi ≤ 2 * pi using ‹a < pi› by linarith
    have int-split: integral {0..2*pi} f =
       integral {0..a} f + integral {a..a+pi} f + integral {a+pi..2*pi} f
    proof −
      have integral {0..2*pi} f = integral {0..a+pi} f + integral {a+pi..2*pi} f
        using Henstock-Kurzweil-Integration.integral-combine [OF - api-le f-int]
          a-pos pi-gt-zero by auto
      moreover have f integrable-on {0..a+pi}
        using integrable-subinterval-real[OF f-int] a-pos api-le by auto
      ultimately show ?thesis
        by (metis Henstock-Kurzweil-Integration.integral-combine ‹0 ≤ a› a-le)
    qed
      — Use ∫ f = 0 to show c1 = c2
    have integral {0..2*pi} f = 0
      using f0 by (simp add: has-integral-integrable-integral)
    hence sum-eq: c1 * (cos (0 − a) − cos (a − a)) + c2 * (cos (a − a) − cos
((a+pi) − a)) +
        c3 * (cos ((a+pi) − a) − cos (2*pi − a)) = 0
      using int-split eq1 eq2 eq3 by linarith
    have c1 * (cos a − 1) + 2 * c2 + c1 * (− 1 − cos a) = 0
      using f0-eq f2pi-eq feq sin-a-nz sum-eq by fastforce
    hence c12-eq: c1 = c2
      by (simp add: algebra-simps)
    show ?thesis
      using f0-eq feq sin-a-nz c1 c2 c3 c12-eq by fastforce
  qed
 qed
qed

theorem scaled-Wirtinger-inequality:
 fixes f f′ :: real ⇒ real
 assumes f′: ⋀x. x ∈ {0..1} ⟹ (f′ has-integral (f x − f 0)) {0..x}
   and f 1 = f 0
   and f-int: (f has-integral 0) {0..1}
   and f′-int: (λx. (f′ x)²) integrable-on {0..1}
 shows (λx. (f x)²) integrable-on {0..1}
   and integral {0..1} (λx. (2*pi * f x)²) ≤ integral {0..1} (λx. (f′ x)²)
   and integral {0..1} (λx. (2*pi * f x)²) = integral {0..1} (λx. (f′ x)²) ⟹
    ∃ c a. ∀ x ∈ {0..1}. f x = c * sin (2*pi*x − a)
proof −
 define g where g ≡ λx. f (x / (2*pi))
```

define g' **where** $g' \equiv \lambda x.\ (1/(2{*}pi)) * f'\ (x\ /\ (2{*}pi))$
have *twopi-pos*: $2 * pi > 0$ **and** *twopi-nz*: $2 * pi \neq 0$
and *inv-twopi-pos*: $1/(2{*}pi) > 0$ **and** *inv-twopi-nz*: $1/(2{*}pi) \neq (0::real)$
using *pi-gt-zero* **by** *auto*
have *img*: $(\lambda x.\ x\ /\ (1/(2{*}pi)))\ `\ \{0..1\} = \{0..2{*}pi\}$
using *image-divide-atLeastAtMost*[*OF inv-twopi-pos*] **by** *simp*
have *prec1*: $(g'$ *has-integral* $(g\ x - g\ 0))\ \{0..x\}$ **if** x: $x \in \{0..2{*}pi\}$ **for** x
proof −
have $*$: $((\lambda s.\ f'\ (1/(2{*}pi) * s))$ *has-integral* $(2{*}pi) *_R (f\ (x/(2{*}pi)) - f\ 0))$
$((\lambda s.\ s\ /\ (1/(2{*}pi)))\ `\ \{0..x/(2{*}pi)\})$
using *x has-integral-stretch-real*[*OF f' inv-twopi-nz*] *inv-twopi-pos* **by** *simp*
have $**$: $((\lambda s.\ f'\ (s/(2{*}pi)))$ *has-integral* $(2{*}pi) * (f\ (x/(2{*}pi)) - f\ 0))\ \{0..x\}$
using $*$ *image-divide-atLeastAtMost*[*OF inv-twopi-pos*, *of* $0\ x/(2{*}pi)$]
using *twopi-pos* **by** (*simp add*: *field-simps*)
have *val*: $1/(2{*}pi) * ((2{*}pi) * (f\ (x/(2{*}pi)) - f\ 0)) = f\ (x/(2{*}pi)) - f\ 0$
using *twopi-nz* **by** *simp*
show *?thesis*
using *has-integral-mult-right*[*OF* $**$, *of* $1/(2{*}pi)$] *twopi-nz*
unfolding *g'-def g-def val* **by** (*simp add*: *field-simps*)
qed
have *prec2*: $g\ (2{*}pi) = g\ 0$
unfolding *g-def* **using** *assms*(*2*) **by** *simp*
have *prec3*: $(g$ *has-integral* $0)\ \{0..2{*}pi\}$
using *has-integral-stretch-real-iff*[*OF inv-twopi-nz*, *of f 0 0 1*] *f-int g-def img* **by** *auto*
have *int*: $(\lambda x.\ (f'\ (x/(2{*}pi)))^2)$ *integrable-on* $\{0..2{*}pi\}$
using *f'-int integrable-stretch-real*[*OF - inv-twopi-nz*, *of* $\lambda x.\ (f'\ x)^2\ 0\ 1$] *img*
by (*simp add*: *field-simps*)
then have *prec4*: $(\lambda x.\ (g'\ x)^2)$ *integrable-on* $\{0..2{*}pi\}$
unfolding *g'-def power-mult-distrib*
using *integrable-on-cmult-left*[*OF int*, *of* $(1/(2{*}pi))^2$] **by** (*simp add*: *algebra-simps*)

Apply unscaled Wirtinger inequality

have *W1*: $(\lambda x.\ (g\ x)^2)$ *integrable-on* $\{0..2{*}pi\}$
and *W2*: *integral* $\{0..2{*}pi\}\ (\lambda x.\ (g\ x)^2) \leq$ *integral* $\{0..2{*}pi\}\ (\lambda x.\ (g'\ x)^2)$
and *W3*: *integral* $\{0..2{*}pi\}\ (\lambda x.\ (g\ x)^2) =$ *integral* $\{0..2{*}pi\}\ (\lambda x.\ (g'\ x)^2) \Longrightarrow$
$\exists\ c\ a.\ \forall\ x \in \{0..2{*}pi\}.\ g\ x = c * sin\ (x - a)$
using *Wirtinger-inequality*[*OF prec1 prec2 prec3 prec4*] **by** *auto*

Transfer conclusions back to scaled domain

have *g-unfold*: $\bigwedge x.\ (g\ x)^2 = (f\ (1/(2{*}pi) * x))^2$
unfolding *g-def* **by** (*simp add*: *field-simps*)
have *g'-unfold*: $\bigwedge x.\ (g'\ x)^2 = (1/(2{*}pi))^2 * (f'\ (1/(2{*}pi) * x))^2$
unfolding *g'-def* **by** (*simp add*: *power-mult-distrib field-simps*)

Show 1: integrability of $f(x)^2$ on $\{0..1\}$

show *int-f2*: $(\lambda x.\ (f\ x)^2)$ *integrable-on* $\{0..1\}$
using *integrable-stretch-real-iff*[*OF inv-twopi-nz*, *of* $\lambda x.\ (f\ x)^2\ 0\ 1$] *W1 g-def img* **by** *force*

Show 2: the scaled inequality

show *integral* $\{0..1\}$ $(\lambda x.\ (2{*}pi * f\ x)^2) \leq$ *integral* $\{0..1\}$ $(\lambda x.\ (f'\ x)^2)$
proof −
have *lhs-stretch*: *integral* $((\lambda x.\ x\ /\ (1/(2{*}pi)))\ `\ \{0..1\})$ $(\lambda x.\ (f\ (1/(2{*}pi) * x))^2)$
$= (1\ /\ |1/(2{*}pi)|) *_R$ *integral* $\{0..1\}$ $(\lambda x.\ (f\ x)^2)$
using *integral-stretch-real*[*OF inv-twopi-nz, of 0 1* $\lambda x.\ (f\ x)^2$] **by** *simp*
have *lhs-val*: *integral* $\{0..2{*}pi\}$ $(\lambda x.\ (g\ x)^2) = 2{*}pi *$ *integral* $\{0..1\}$ $(\lambda x.\ (f\ x)^2)$
using *lhs-stretch img inv-twopi-pos* **by** (*simp add*: *g-unfold*)
have *rhs-stretch*: *integral* $((\lambda x.\ x\ /\ (1/(2{*}pi)))\ `\ \{0..1\})$ $(\lambda x.\ (1/(2{*}pi))^2 * (f'\ (1/(2{*}pi) * x))^2)$
$= (1\ /\ |1/(2{*}pi)|) *_R$ *integral* $\{0..1\}$ $(\lambda x.\ (1/(2{*}pi))^2 * (f'\ x)^2)$
using *integral-stretch-real*[*OF inv-twopi-nz, of 0 1* $\lambda x.\ (1/(2{*}pi))^2 * (f'\ x)^2$] **by** *simp*
have *factor-out*: *integral* $\{0..1\}$ $(\lambda x.\ (1/(2{*}pi))^2 * (f'\ x)^2) = (1/(2{*}pi))^2 *$ *integral* $\{0..1\}$ $(\lambda x.\ (f'\ x)^2)$
by (*simp add*: *integral-mult-right*)
have *integral* $\{0..2{*}pi\}$ $(\lambda x.\ (g'\ x)^2) = (1\ /\ |1/(2{*}pi)|) *_R$ *integral* $\{0..1\}$ $(\lambda x.\ (1/(2{*}pi))^2 * (f'\ x)^2)$
using *img rhs-stretch* **by** (*simp add*: *g′-unfold*)
also have $\ldots = 2{*}pi * ((1/(2{*}pi))^2 *$ *integral* $\{0..1\}$ $(\lambda x.\ (f'\ x)^2))$
using *inv-twopi-pos factor-out* **by** *simp*
finally have *integral* $\{0..2{*}pi\}$ $(\lambda x.\ (g'\ x)^2) = 2{*}pi * ((1/(2{*}pi))^2 *$ *integral* $\{0..1\}$ $(\lambda x.\ (f'\ x)^2))$ **.**
with *W2 lhs-val*
have $2{*}pi *$ *integral* $\{0..1\}$ $(\lambda x.\ (f\ x)^2) \leq (1/(2{*}pi)) *$ *integral* $\{0..1\}$ $(\lambda x.\ (f'\ x)^2)$
by (*simp add*: *power2-eq-square*)
then have $(2{*}pi)^2 *$ *integral* $\{0..1\}$ $(\lambda x.\ (f\ x)^2) \leq$ *integral* $\{0..1\}$ $(\lambda x.\ (f'\ x)^2)$
using *twopi-pos* **by** (*simp add*: *power2-eq-square field-simps*)
then show *?thesis*
by (*simp add*: *power-mult-distrib*)
qed

Show 3: the equality case

show *integral* $\{0..1\}$ $(\lambda x.\ (2{*}pi * f\ x)^2) =$ *integral* $\{0..1\}$ $(\lambda x.\ (f'\ x)^2) \Longrightarrow$
$\exists\, c\ a.\ \forall\, x \in \{0..1\}.\ f\ x = c * sin\ (2{*}pi{*}x - a)$
proof −
assume *eq*: *integral* $\{0..1\}$ $(\lambda x.\ (2{*}pi * f\ x)^2) =$ *integral* $\{0..1\}$ $(\lambda x.\ (f'\ x)^2)$
have *integral* $\{0..2{*}pi\}$ $(\lambda x.\ (g\ x)^2) = (1\ /\ |1/(2{*}pi)|) *_R$ *integral* $\{0..1\}$ $(\lambda x.\ (f\ x)^2)$
using *img integral-stretch-real*[*OF inv-twopi-nz, of 0 1* $\lambda x.\ (f\ x)^2$] **by** (*simp add*: *g-unfold*)
also have $\ldots = 2{*}pi *$ *integral* $\{0..1\}$ $(\lambda x.\ (f\ x)^2)$
using *inv-twopi-pos* **by** *simp*
finally have *lhs*: *integral* $\{0..2{*}pi\}$ $(\lambda x.\ (g\ x)^2) = 2{*}pi *$ *integral* $\{0..1\}$ $(\lambda x.\ (f\ x)^2)$ **.**
have *rhs*: *integral* $\{0..2{*}pi\}$ $(\lambda x.\ (g'\ x)^2)$

```
          = (1/(2*pi)) * integral {0..1} (λx. (f' x)²)
    proof −
      have integral {0..2*pi} (λx. (g' x)²)
         = (1 / |1/(2*pi)|) *R integral {0..1} (λx. (1/(2*pi))² * (f' x)²)
       using integral-stretch-real[OF inv-twopi-nz, of 0 1 λx. (1/(2*pi))² * (f' x)²]
img
        by (simp add: g'-unfold)
      also have ... = (1/(2*pi)) * integral {0..1} (λx. (f' x)²)
        using twopi-pos by (simp add: power2-eq-square field-simps)
      finally show ?thesis .
    qed
    have (2*pi)² * integral {0..1} (λx. (f x)²) = integral {0..1} (λx. (f' x)²)
      using eq by (simp add: power-mult-distrib)
    then have 2*pi * integral {0..1} (λx. (f x)²) = (1/(2*pi)) * integral {0..1}
(λx. (f' x)²)
      using twopi-pos by (simp add: power2-eq-square field-simps)
    then have weq: integral {0..2*pi} (λx. (g x)²) = integral {0..2*pi} (λx. (g'
x)²)
      using lhs rhs by linarith
    from W3[OF weq] obtain c a where ca: ∀x ∈ {0..2*pi}. g x = c * sin (x −
a) by auto
    have f x = c * sin (2*pi*x − a) if x ∈ {0..1} for x
    proof −
      have 2*pi*x ∈ {0..2*pi} using twopi-pos
        using that by auto
      with ca show f x = c * sin (2*pi*x − a)
        by (metis g-def nonzero-mult-div-cancel-left twopi-nz)
    qed
    then show ∃c a. ∀x ∈ {0..1}. f x = c * sin (2*pi*x − a) by auto
  qed
qed

end
```

3 A very special case of Green's theorem for a convex area

```
theory Green-Variant
  imports Wirtinger

begin
```

3.1 Area under an arc.

```
locale Area =
  fixes g :: real ⇒ complex and g' :: real ⇒ complex and u v S
  assumes uv: u ≤ v
    and Re-g-le: Re (g u) ≤ Re (g v)
```

and *acont-g*: *absolutely-continuous-on* $\{u..v\}$ *g*
and *gim*: g ‘ $\{u..v\} \subseteq \{z.\ Im\ z \geq 0\}$
and *inj-g*: *inj-on g* $\{u..v\}$
and *inj-Re*: *inj-on Re* (g ‘ $\{u..v\}$)
and *negS*: *negligible S*
and *gder*: $\bigwedge t.\ t \in \{u..v\} - S \Longrightarrow (g\ has\text{-}vector\text{-}derivative\ g'\ t)\ (at\ t)$

begin

lemma *below-arclet*:
shows $(\lambda t.\ Re\ (g'\ t) * Im\ (g\ t))$ *absolutely-integrable-on* $\{u..v\}$ (**is** *?th1*)
and *integral* $\{u..v\}$ $(\lambda t.\ Re\ (g'\ t) * Im\ (g\ t)) =$
measure lebesgue $\{z.\ \exists w \in g$ ‘ $\{u..v\}.\ Re\ w = Re\ z \wedge 0 \leq Im\ z \wedge Im\ z \leq Im\ w\}$ (**is** *?th2*)
proof −
obtain *h* **where** *h*: $\bigwedge x.\ x \in \{u..v\} \Longrightarrow h\ (Re\ (g\ x)) = x$
by (*metis inj-g inj-Re comp-def comp-inj-on inv-into-f-f*)
define *ax* **where** $ax \equiv (\lambda t.\ Re\ (g\ t))$ ‘ $\{u..v\}$
have *cont-h*: *continuous-on ax h*
unfolding *ax-def*
by (*simp add*: *absolutely-continuous-on-imp-continuous acont-g continuous-on-Re continuous-on-inv h*)
show *?th1*
proof −
have *cont-g*: *continuous-on* $\{u..v\}$ *g*
by (*simp add*: *absolutely-continuous-on-imp-continuous acont-g*)
have *gp-ai*: g' *absolutely-integrable-on* $\{u..v\}$
by (*meson absolutely-integrable-absolutely-continuous-derivative acont-g gder has-vector-derivative-at-within negS*)
have *Re-gp-ai*: $(\lambda t.\ Re\ (g'\ t))$ *absolutely-integrable-on* $\{u..v\}$
using *Re-absolutely-integrable-on gp-ai* **by** *blast*
have *Im-g-cont*: *continuous-on* $\{u..v\}$ $(\lambda t.\ Im\ (g\ t))$
by (*intro continuous-intros cont-g*)
have *Im-g-bdd*: *bounded* $((\lambda t.\ Im\ (g\ t))$ ‘ $\{u..v\})$
by (*intro compact-imp-bounded compact-continuous-image*[*OF Im-g-cont compact-Icc*])
have *Im-g-meas*: $(\lambda t.\ Im\ (g\ t)) \in$ *borel-measurable* (*lebesgue-on* $\{u..v\}$)
using *continuous-imp-measurable-on-sets-lebesgue*[*OF Im-g-cont*] *atLeastAtMost-borel lborelD*
by (*metis sets-completionI-sets*)
show *?thesis*
using *absolutely-integrable-bounded-measurable-product-real* [*OF Im-g-meas - Im-g-bdd Re-gp-ai*]
by (*simp add*: *mult.commute*)
qed
have *cont-g*: *continuous-on* $\{u..v\}$ *g*
using *acont-g absolutely-continuous-on-imp-continuous is-interval-cc* **by** *blast*
have *cont-Reg*: *continuous-on* $\{u..v\}$ $(\lambda t.\ Re\ (g\ t))$
by (*intro continuous-intros cont-g*)

have *inj-Reg*: *inj-on* ($\lambda t.$ *Re* (*g t*)) {*u..v*}
using *comp-inj-on*[*OF inj-g inj-Re*] **by** (*simp add*: *o-def*)
have ∗: *strict-mono-on* {*u..v*} ($\lambda t.$ *Re* (*g t*)) ∨ *strict-antimono-on* {*u..v*} ($\lambda t.$ *Re* (*g t*))
using *injective-eq-monotone-map*[*OF is-interval-cc cont-Reg*] *inj-Reg* **by** *auto*
have *ax*: *ax* = {*Re* (*g u*)..*Re* (*g v*)}
proof (*rule antisym*)
show *ax* ⊆ {*Re* (*g u*)..*Re* (*g v*)}
proof (*cases u* = *v*)
case *False*
with ∗ *uv* **have** *mono*: *strict-mono-on* {*u..v*} ($\lambda t.$ *Re* (*g t*))
by (*smt* (*verit*, *ccfv-threshold*) *Re-g-le atLeastAtMost-iff monotone-onD*)
show *?thesis*
using *mono* **by** (*auto simp*: *monotone-on-def ax-def less-eq-real-def*)
qed (*auto simp*: *ax-def*)
next
show {*Re* (*g u*)..*Re* (*g v*)} ⊆ *ax*
using *ivt-increasing-component-on-1*[*OF uv cont-g*, *of 1*] **by** (*force simp*: *ax-def*)
qed
show *?th2*
proof −
define *f* **where** *f* ≡ ($\lambda x.$ *Im* (*g* (*h x*)))
have *h-range*: *h* ‘ *ax* ⊆ {*u..v*}
using *h* **by** (*force simp*: *ax-def*)
have *cont-f*: *continuous-on ax f*
using *cont-g cont-h continuous-on-Im continuous-on-compose2 f-def h-range* **by** *blast*
have *f-nonneg*: $\bigwedge x.$ *x* ∈ *ax* ⟹ *f x* ≥ *0*
unfolding *f-def* **using** *gim h-range* **by** *blast*
have *mono-Reg*: *mono-on* {*u..v*} ($\lambda t.$ *Re* (*g t*))
using ∗ *Re-g-le atLeastAtMost-iff leD order-le-less uv*
unfolding *monotone-on-def* **by** *metis*
have *acont-Reg*: *absolutely-continuous-on* {*u..v*} ($\lambda t.$ *Re* (*g t*))
using *absolutely-continuous-on-compose-linear*[*OF acont-g bounded-linear-Re*[*THEN bounded-linear.linear*]]
by (*simp add*: *o-def*)
have *deriv-Reg*: $\bigwedge t.$ *t* ∈ {*u..v*} − *S* ⟹ (($\lambda t.$ *Re* (*g t*)) *has-vector-derivative* *Re* (*g*′ *t*)) (*at t*)
using *bounded-linear-Re*[*THEN bounded-linear.has-vector-derivative*] *gder* **by** *blast*
— Apply substitution: $\int_{Re(g\,v)}^{Re(g\,u)} f = \int_u^v Re(g') \cdot f(Re(g)) = \int_u^v Re(g') \cdot Im(g)$
have *subst*: (($\lambda t.$ *Re* (*g*′ *t*) ∗ *f* (*Re* (*g t*))) *has-integral* (*integral* {*Re* (*g u*)..*Re* (*g v*)} *f*)) {*u..v*}
using *has-integral-substitution-ac*[*OF uv Re-g-le acont-Reg negS deriv-Reg* -]
using *mono-Reg cont-f ax negS* **by** (*force simp*: *monotone-on-def*)
— Since $f(Re(g\,t)) = Im(g\,t)$, the left-hand side simplifies
have *integral* {*u..v*} ($\lambda t.$ *Re* (*g*′ *t*) ∗ *Im* (*g t*)) = *integral* {*Re* (*g u*)..*Re* (*g v*)} *f*
using *h has-integral-spike*[*OF negligible-empty - subst*] *integral-unique*

by (*fastforce simp*: *f-def*)
— Apply area-under-curve: measure of the subgraph = $\int f$
also have ... = *measure lebesgue* $\{z.\ \exists\, w \in g\ `\ \{u..v\}.\ Re\ w = Re\ z \wedge 0 \leq Im\ z \wedge Im\ z \leq Im\ w\}$
proof −
— First show the subgraph set equals the area under the graph of f
have *set-eq*: $\{z.\ \exists\, w \in g\ `\ \{u..v\}.\ Re\ w = Re\ z \wedge 0 \leq Im\ z \wedge Im\ z \leq Im\ w\}$ = $\{z.\ Re\ (g\ u) \leq Re\ z \wedge Re\ z \leq Re\ (g\ v) \wedge 0 \leq Im\ z \wedge Im\ z \leq f\ (Re\ z)\}$ (**is** *?L=?R*)
proof (*intro antisym subsetI*)
fix z **assume** $z \in ?L$
then obtain $w\ t$ **where** *wt*: $t \in \{u..v\}$ $w = g\ t$ $Re\ w = Re\ z$ $0 \leq Im\ z$ $Im\ z \leq Im\ w$
by *auto*
have $Re\ z \in ax$ **unfolding** *ax-def* **using** *wt* **by** *force*
then show $z \in ?R$
using *f-def h wt ax* **by** *fastforce*
next
fix z **assume** *z*: $z \in ?R$
then have *Rez*: $Re\ z \in ax$ **using** *ax* **by** *auto*
then show $z \in ?L$
unfolding *ax-def* **using** *h z* **by** (*fastforce simp*: *f-def*)
qed
— Then apply area-under-curve (Fubini/Cavalieri)
show *?thesis* **unfolding** *set-eq*
using *has-integral-area-under-curve*[*OF Re-g-le - -*] *cont-f f-nonneg ax*
by (*metis* (*no-types*, *lifting*))
qed
finally show *?thesis* **.**
qed
qed

end

3.2 Area above an arc.

lemma *area-above-arclet*:
fixes g :: *real* ⇒ *complex* **and** g' :: *real* ⇒ *complex*
assumes $u \leq v$
and *Re-g-ge*: $Re\ (g\ v) \leq Re\ (g\ u)$
and *ac-g*: *absolutely-continuous-on* $\{u..v\}$ g
and *gim*: $g\ `\ \{u..v\} \subseteq \{z.\ Im\ z \leq 0\}$
and *injg*: *inj-on* $g\ \{u..v\}$
and *injRe*: *inj-on* $Re\ (g\ `\ \{u..v\})$
and *negligible* S
and *vder-g*: $\bigwedge t.\ t \in \{u..v\} - S \Longrightarrow (g\ has\text{-}vector\text{-}derivative\ g'\ t)\ (at\ t)$
shows $(\lambda t.\ Re\ (g'\ t) * Im\ (g\ t))$ *absolutely-integrable-on* $\{u..v\}$ (**is** *?th1*)
and *integral* $\{u..v\}$ $(\lambda t.\ Re\ (g'\ t) * Im\ (g\ t)) =$

measure lebesgue {*z*. ∃ *w* ∈ *g* ‘ {*u*..*v*}. *Re w* = *Re z* ∧ *Im w* ≤ *Im z* ∧ *Im z* ≤ *0*} (**is** *?th2*)
proof −
— Symmetry: define $h(t) = \overline{g(u+v-t)}$ and apply *area-below-arclet*
define φ **where** $\varphi \equiv \lambda t.\ u + v - t$
define *h* **where** $h \equiv cnj \circ g \circ \varphi$
define h' **where** $h' \equiv \lambda t.\ -\ cnj\ (g'\ (\varphi\ t))$
interpret *Area h h′ u v* φ ‘ *S*
proof
show $u \le v$
by *fact*
show *Re* (*h u*) ≤ *Re* (*h v*)
by (*simp add*: φ*-def Re-g-ge h-def*)
show *absolutely-continuous-on* {*u*..*v*} *h*
by (*simp add*: φ*-def absolutely-continuous-on-compose-linear absolutely-continuous-on-reflect assms*(*3*) *h-def linear-cnj*)
show *h* ‘ {*u*..*v*} ⊆ {*z*. *0* ≤ *Im z*}
using *gim* **by** (*auto simp*: *h-def* φ*-def image-subset-iff*)
show *inj-on h* {*u*..*v*}
using *injg* **by** (*fastforce simp*: *inj-on-def h-def* φ*-def*)
show *inj-on Re* (*h* ‘ {*u*..*v*})
using *injRe* **by** (*fastforce simp*: *inj-on-def h-def* φ*-def*)
have φ ‘ *S* = (+) (*u* + *v*) ‘ (*uminus* ‘ *S*)
unfolding φ*-def image-image* **by** (*simp add*: *algebra-simps*)
then show *negligible* (φ ‘ *S*)
by (*simp add*: ‹*negligible S*› *linear-uminus negligible-linear-image negligible-translation*)
show ⋀*t*. *t* ∈ {*u*..*v*} − (φ‘*S*) ⟹ (*h has-vector-derivative h′ t*) (*at t*)
unfolding *has-vector-derivative-def h-def h′-def* φ*-def*
by (*rule vder-g* [*unfolded has-vector-derivative-def*] *derivative-eq-intros* | *force*)+
qed
have *integrand-eq*: ⋀*t*. *Re* (*h′ t*) ∗ *Im* (*h t*) = *Re* (*g′* (φ *t*)) ∗ *Im* (*g* (φ *t*))
unfolding *h-def h′-def* **by** (*simp add*: *o-def cnj.sel*)
have *integral-eq*: *integral* {*u*..*v*} (λ*t*. *Re* (*g′* (φ *t*)) ∗ *Im* (*g* (φ *t*))) =
integral {*u*..*v*} (λ*t*. *Re* (*g′ t*) ∗ *Im* (*g t*))
proof −
define *f* **where** $f \equiv \lambda t.\ Re\ (g'\ t) * Im\ (g\ t)$
have *comp-eq*: (*f* ∘ (+) (*u* + *v*)) ∘ *uminus* = *f* ∘ φ
unfolding φ*-def comp-def* **by** (*simp add*: *algebra-simps*)
have (*f* ∘ φ) *absolutely-integrable-on* {*u*..*v*}
using *below-arclet*(*1*) *f-def integrand-eq set-integrable-cong* **by** *fastforce*
then have *integral* {*u*..*v*} (*f* ∘ φ) = *integral* (*uminus* ‘ {*u*..*v*}) (*f* ∘ (+) (*u* + *v*))
by (*subst integral-change-of-variables-linear*[*OF linear-uminus*]) (*auto simp*: *comp-eq*)
also have ... = *integral* {*u*..*v*} *f*
using *integral-shift-Icc-real*[*of* −*v* −*u f u*+*v*] **by** (*simp add*: *algebra-simps*)

```
    finally show ?thesis
      by (simp add: f-def comp-def)
  qed
  show ?th1
  proof −
    have Re-gp-ai: (λt. Re (g′ t)) absolutely-integrable-on {u..v}
      using Re-absolutely-integrable-on has-vector-derivative-at-within assms
      by (metis vder-g absolutely-integrable-absolutely-continuous-derivative)
    have Im-g-cont: continuous-on {u..v} (λt. Im (g t))
       by (simp add: absolutely-continuous-on-imp-continuous assms(3) continuous-on-Im)
    have Im-g-bdd: bounded ((λt. Im (g t)) ‘ {u..v})
     by (intro compact-imp-bounded compact-continuous-image[OF Im-g-cont compact-Icc])
    have Im-g-meas: (λt. Im (g t)) ∈ borel-measurable (lebesgue-on {u..v})
     using Im-g-cont integrable-continuous-real integrable-imp-measurable by blast
    show ?thesis
     using absolutely-integrable-bounded-measurable-product-real [OF Im-g-meas - Im-g-bdd Re-gp-ai]
      by (simp add: mult.commute)
  qed
  have φ-image: φ ‘ {u..v} = {u..v}
    using assms(1) unfolding φ-def by (auto simp: image-iff)
  have h-image: h ‘ {u..v} = cnj ‘ (g ‘ {u..v})
    by (metis φ-image h-def image-comp)
  define A where A ≡ {z. ∃w ∈ h ‘ {u..v}. Re w = Re z ∧ 0 ≤ Im z ∧ Im z ≤ Im w}
  define B where B ≡ {z. ∃w ∈ g ‘ {u..v}. Re w = Re z ∧ Im w ≤ Im z ∧ Im z ≤ 0}
  have AB: A = cnj ‘ B
    unfolding A-def h-image B-def by (force simp: in-image-cnj-iff)
  have cont-g-uv: continuous-on {u..v} g
   using assms(3) absolutely-continuous-on-imp-continuous is-interval-cc by blast
  have compact B
  proof −
    — B is the continuous image of the compact set {u..v} × {0..1}
    define ψ where ψ ≡ λ(t,s). Complex (Re (g t)) ((1 − s) ∗ Im (g t))
    have cont-ψ: continuous-on ({u..v} × {0..1}) ψ
      unfolding ψ-def split-def
     by (intro continuous-intros continuous-on-compose2[OF continuous-on-Re[OF cont-g-uv]]
              continuous-on-compose2[OF continuous-on-Im[OF cont-g-uv]] continuous-on-fst) auto
    have img: ψ ‘ ({u..v} × {0..1}) = B
    proof (rule set-eqI)
      fix z :: complex
      show z ∈ ψ ‘ ({u..v} × {0..1}) ⟷ z ∈ B
      proof
        assume z ∈ ψ ‘ ({u..v} × {0..1})
```

```
      then obtain t s where ts: t ∈ {u..v} s ∈ {0..1} z = Complex (Re (g t))
((1 − s) ∗ Im (g t))
        unfolding ψ-def by auto
       have Im-le: Im (g t) ≤ 0
        using assms(4) ts(1) by (auto simp: image-subset-iff)
       have g t ∈ g ‘ {u..v} using ts(1) by auto
       moreover have Im (g t) ≤ Im z
        using ts Im-le mult-right-mono-neg by (simp add: mult-le-cancel-right1)
       moreover have Im z ≤ 0
        using ts Im-le mult-nonneg-nonpos[of 1−s Im (g t)] by auto
       ultimately show z ∈ B unfolding B-def using ts by auto
     next
       assume z ∈ B
       then obtain w t where wt: t ∈ {u..v} w = g t Re w = Re z Im w ≤ Im z
Im z ≤ 0
        unfolding B-def by auto
       show z ∈ ψ ‘ ({u..v} × {0..1})
       proof (cases Im (g t) = 0)
        case True
        then show ?thesis
         using wt by (force simp: image-iff ψ-def complex-eq-iff)
       next
        case False
        then have neg: Im (g t) < 0
         using wt by force
        define s where s ≡ 1 − Im z / Im (g t)
        have 0 ≤ Im z / Im (g t)
         using wt(5) neg by (simp add: field-simps)
        then have s ∈ {0..1}
         using neg s-def wt(2,4) by force
        moreover have z = ψ (t, s) unfolding ψ-def s-def
         using wt(2,3) False by (simp add: complex-eq-iff field-simps)
        ultimately show ?thesis using wt(1) by auto
       qed
     qed
   qed
   then show ?thesis
    using img compact-continuous-image[OF cont-ψ] by (simp add: compact-Times)
  qed
  then have B-meas: B ∈ lmeasurable using lmeasurable-compact by blast
  have measure-eq: measure lebesgue {z. ∃ w ∈ h ‘ {u..v}. Re w = Re z ∧ 0 ≤ Im
z ∧ Im z ≤ Im w} =
                   measure lebesgue {z. ∃ w ∈ g ‘ {u..v}. Re w = Re z ∧ Im w ≤ Im
z ∧ Im z ≤ 0}
    using AB measure-linear-image[OF linear-cnj B-meas] det-complex
    by (simp add: A-def B-def)
  show ?th2
    using below-arclet(2) integrand-eq integral-eq measure-eq by (simp add: o-def)
qed
```

3.3 Lemmas for Green's theorem

definition *Green-concl* :: (*real* ⇒ *complex*) ⇒ (*real* ⇒ *complex*) ⇒ *bool* **where**
Green-concl g g′ ≡ (λ*t*. *Re* (*g′ t*) ∗ *Im* (*g t*)) *absolutely-integrable-on* {*0..1*}
∧ |*integral* {*0..1*} (λ*t*. *Re* (*g′ t*) ∗ *Im* (*g t*))| = *measure lebesgue* (*inside* (*path-image g*))

At most two points on the frontier of a 2-dimensional convex body can share the same inner product with a non-zero vector. Consequence: if three distinct points on the frontier have the same *Re* (or *Im*), the body must lie on one side.

lemma *frontier-vertical-at-most-two*:
fixes *S* :: *complex set* **and** *c* :: *real*
assumes *convex S compact S interior S* ≠ {}
and *sides*: ∃ *p* ∈ *S*. *Re p* < *c* ∃ *q* ∈ *S*. *c* < *Re q*
and *xyz*: *x* ∈ *frontier S y* ∈ *frontier S z* ∈ *frontier S*
Re x = *c Re y* = *c Re z* = *c*
shows ¬ (*x* ≠ *y* ∧ *x* ≠ *z* ∧ *y* ≠ *z*)
proof −
define *T* **where** *T* ≡ {*w* :: *complex*. *1* ∙ *w* = *c*}
have *aff-T*: *affine T* **unfolding** *T-def* **by** (*rule affine-hyperplane*)
have *T-eq*: *T* = {*w*. *Re w* = *c*} **unfolding** *T-def* **by** (*simp add*: *complex-inner-1*)
have *xT*: *x* ∈ *T y* ∈ *T z* ∈ *T*
using *xyz* **unfolding** *T-eq* **by** *auto*
— The interior of *S* intersects *T*
have *int-T*: *interior S* ∩ *T* ≠ {}
proof −
have *cl-int*: *closure* (*interior S*) = *S*
by (*simp add*: *assms compact-imp-closed convex-closure-interior*)
— Find interior points on each side of *Re* = *c*
obtain *p* **where** *p*: *p* ∈ *interior S Re p* < *c*
proof −
obtain *p0* **where** *p0*: *p0* ∈ *S Re p0* < *c* **using** *sides*(*1*) **by** *auto*
then obtain *ps* **where** *ps*: ∀ *n*. *ps n* ∈ *interior S ps* ⟶ *p0*
using *cl-int closure-sequential* **by** *blast*
obtain *n* **where** *Re* (*ps n*) < *c* **using** *eventually-sequentially*
by (*metis LIMSEQ-le-const not-less p0*(*2*) *ps*(*2*) *tendsto-complex-iff*)
then show *thesis* **using** *that ps*(*1*) **by** *auto*
qed
obtain *q* **where** *q*: *q* ∈ *interior S c* < *Re q*
proof −
obtain *q0* **where** *q0*: *q0* ∈ *S c* < *Re q0* **using** *sides*(*2*) **by** *auto*
then obtain *qs* **where** *qs*: ∀ *n*. *qs n* ∈ *interior S qs* ⟶ *q0*
using *cl-int closure-sequential* **by** *blast*
then obtain *n* **where** *c* < *Re* (*qs n*) **using** *eventually-sequentially*
by (*metis LIMSEQ-le-const2 not-le q0*(*2*) *tendsto-complex-iff*)
then show *thesis* **using** *that qs*(*1*) **by** *auto*
qed
— IVT on the segment {*p..q*} ⊆ *interior S*

```
    have seg-sub: closed-segment p q ⊆ interior S
      using convex-interior ‹convex S› closed-segment-subset p(1) q(1) by metis
    obtain r where r ∈ closed-segment p q 1 · r = c
      using connected-ivt-hyperplane[OF connected-segment]
      by (smt (verit, del-insts) complex-inner-1 ends-in-segment p(2) q(2))
    then show ?thesis using seg-sub T-eq by auto
  qed
  — Apply convex-affine-rel-frontier-Int
  have rf-eq: rel-frontier (S ∩ T) = frontier S ∩ T
    using convex-affine-rel-frontier-Int[OF assms(1) aff-T int-T] .
  — S ∩ T is a compact convex collinear set, hence a closed segment
  have ST-ne: S ∩ T ≠ {}
    using int-T interior-subset by blast
  have ST-compact: compact (S ∩ T)
    using T-def assms(2) closed-hyperplane by blast
  have ST-convex: convex (S ∩ T)
    using assms(1) convex-Int aff-T affine-imp-convex by auto
  have aff-dim T = int (DIM(complex) − 1)
    unfolding T-def by (rule aff-dim-hyperplane) simp
  then have aff-dim T = 1 by simp
  then have ST-collinear: collinear (S ∩ T)
    by (metis collinear-aff-dim collinear-subset dual-order.refl inf-le2)
  obtain p q where pq: S ∩ T = closed-segment p q
   using compact-convex-collinear-segment[OF ST-ne ST-compact ST-convex ST-collinear]
by auto
  — The rel-frontier of a closed segment has at most two elements
  have rf-sub: rel-frontier (S ∩ T) ⊆ {p, q}
  proof (cases p = q)
    case True
    then show ?thesis unfolding pq by simp
  next
    case False then show ?thesis
       by (simp add: Diff-Diff-Int pq rel-frontier-def rel-interior-segment(1) seg-
ment(1))
  qed
  — x, y, z all lie in rel-frontier (S ∩ T) = frontier S ∩ T ⊆ {p,q}
  have x ∈ {p, q} y ∈ {p, q} z ∈ {p, q}
    using xyz xT rf-eq rf-sub by auto
  then show ?thesis by auto
qed


locale Green =
  fixes g :: real ⇒ complex and g′ :: real ⇒ complex
    and U :: real set
    and a b :: complex
  assumes g: simple-path g pathstart g = a pathfinish g = a
    and b: b ∈ path-image g Re a < Re b Im a = Im b
    and dab: dist a b = diameter (path-image g)
```

and *conv*: *convex* (*inside* (*path-image g*))
and *cont*: *absolutely-continuous-on* {*0..1*} *g*
and *U*: *negligible U*
and *vder*: ⋀*t*. *t* ∈ {*0..1*} − *U* ⟹ (*g has-vector-derivative g′ t*) (*at t*)

begin

definition *gop* ≡ *cnj* ∘ *reversepath g*
definition *gop′* ≡ *uminus* ∘ *cnj* ∘ *reversepath g′*

lemma *cnj-rev*: *Green gop gop′* ((λ*t*. *1*−*t*) ‘ *U*) (*cnj a*) (*cnj b*)
proof
show *simple-path gop*
using *g* **by** (*simp add*: *simple-path-def gop-def loop-free-reversepath loop-free-cnj*)
show *pathstart gop* = *cnj a*
using *g* **by** (*simp add*: *gop-def pathstart-compose*)
show *pathfinish gop* = *cnj a*
using *g* **by** (*simp add*: *gop-def pathfinish-compose*)
show *cnj b* ∈ *path-image gop*
using *b* **by** (*simp add*: *gop-def path-image-compose*)
show *Re* (*cnj a*) < *Re* (*cnj b*) *Im* (*cnj a*) = *Im* (*cnj b*)
using *b* **by** *auto*
show *dist* (*cnj a*) (*cnj b*) = *diameter* (*path-image gop*)
by (*simp add*: *gop-def dab diameter-image-cnj path-image-compose flip*: *dab*)
show *convex* (*inside* (*path-image gop*))
unfolding *gop-def*
by (*metis conv convex-linear-vimage image-cnj-conv-vimage-cnj*
inside-cnj-image linear-cnj path-image-compose path-image-reversepath)
show *absolutely-continuous-on* {*0..1*} *gop*
using *cont* **unfolding** *gop-def*
by (*simp add*: *absolutely-continuous-on-compose-linear absolutely-continuous-on-reflect linear-cnj*
reversepath-o)
have *negligible* (*uminus* ‘ *U*)
by (*simp add*: *U linear-uminus negligible-linear-image-eq*)
then have *negligible* (((+)*1*) ‘ *uminus* ‘ *U*)
using *negligible-translation* **by** *blast*
then show *negligible* ((−) *1* ‘ *U*)
by (*smt* (*verit*, *best*) *image-cong image-image*)
next
fix *t* :: *real*
note *vder* [*unfolded has-vector-derivative-def*, *derivative-intros*]
assume *t* ∈ {*0..1*} − (−) *1* ‘ *U*
then have *0* ≤ *t t* ≤ *1 1*−*t* ∉ *U*
by (*auto simp*: *image-iff*)
then show (*gop has-vector-derivative gop′ t*) (*at t*)
unfolding *gop-def gop′-def has-vector-derivative-def reversepath-o*
by − (*rule derivative-eq-intros* | *simp add*: *o-def* | *assumption*)+
qed

lemma *rev*: *Green* (*reversepath g*) (*uminus* ∘ *reversepath g′*) ((λ*t*. 1−*t*) ‘ *U*) *a b*
proof
show *simple-path* (*reversepath g*)
using *g* **by** (*simp add*: *simple-path-def loop-free-reversepath*)
show *pathstart* (*reversepath g*) = *a*
using *g* **by** (*simp add*: *pathstart-compose*)
show *pathfinish* (*reversepath g*) = *a*
using *g* **by** (*simp add*: *pathfinish-compose*)
show *b* ∈ *path-image* (*reversepath g*)
using *b* **by** (*simp add*: *path-image-compose*)
show *Re a* < *Re b Im a* = *Im b*
using *b* **by** *auto*
show *dist a b* = *diameter* (*path-image* (*reversepath g*))
by (*simp add*: *gop-def dab path-image-compose flip*: *dab*)
show *convex* (*inside* (*path-image* (*reversepath g*)))
by (*simp add*: *gop-def conv*)
show *absolutely-continuous-on* {*0..1*} (*reversepath g*)
using *cont* **unfolding** *gop-def*
by (*simp add*: *absolutely-continuous-on-compose-linear absolutely-continuous-on-reflect linear-cnj*
reversepath-o)
have *negligible* (*uminus* ‘ *U*)
by (*simp add*: *U linear-uminus negligible-linear-image-eq*)
then have *negligible* (((+)*1*) ‘ *uminus* ‘ *U*)
using *negligible-translation* **by** *blast*
then show *negligible* ((−) *1* ‘ *U*)
by (*smt* (*verit*, *best*) *image-cong image-image*)
next
fix *t* :: *real*
note *vder* [*unfolded has-vector-derivative-def*, *derivative-intros*]
assume *t* ∈ {*0..1*} − (−) *1* ‘ *U*
then have *0* ≤ *t t* ≤ *1 1*−*t* ∉ *U*
by (*auto simp*: *image-iff*)
then show ((*reversepath g*) *has-vector-derivative* (*uminus* ∘ *reversepath g′*) *t*) (*at t*)
unfolding *gop-def gop′-def has-vector-derivative-def reversepath-o*
by − (*rule derivative-eq-intros* | *simp add*: *o-def* | *assumption*)+
qed

lemma *f-abs-int*: (λ*s*. *Re* (*g′ s*) ∗ *Im* (*g s*)) *absolutely-integrable-on* {*0..1*}
proof −
have *cont-g*: *continuous-on* {*0..1*} *g*
using *simple-path-imp-path*[*OF g*(*1*)] **by** (*simp add*: *path-def*)
have *Im-g-cont*: *continuous-on* {*0..1*} (λ*t*. *Im* (*g t*))
by (*intro continuous-intros cont-g*)
have *Im-g-bdd*: *bounded* ((λ*t*. *Im* (*g t*)) ‘ {*0..1*})
by (*intro compact-imp-bounded compact-continuous-image*[*OF Im-g-cont com-*

pact-Icc])
have *Im-g-meas*: $(\lambda t.\ Im\ (g\ t)) \in$ *borel-measurable* (*lebesgue-on* $\{0..1\}$)
using *continuous-imp-measurable-on-sets-lebesgue*[*OF Im-g-cont*]
atLeastAtMost-borel lborelD
by (*metis sets-completionI-sets*)
have *gp-ai*: g' *absolutely-integrable-on* $\{0..1\}$
using *absolutely-integrable-absolutely-continuous-derivative*[*OF cont U*]
vder has-vector-derivative-at-within **by** *blast*
then show *?thesis*
using *absolutely-integrable-bounded-measurable-product-real*[*OF Im-g-meas - Im-g-bdd Re-absolutely-integrable-on*]
by (*simp add*: *mult.commute*)
qed

lemma *arc-inj-on*:
assumes *huv*: $0 \le u\ v \le 1\ u < v$ **and** *hne*: $u > 0 \vee v < 1$
shows *inj-on g* $\{u..v\}$
proof (*rule inj-onI*)
fix *s1 s2* **assume** *s1*: $s1 \in \{u..v\}$ **and** *s2*: $s2 \in \{u..v\}$ **and** *eq*: $g\ s1 = g\ s2$
then have *s1-01*: $s1 \in \{0..1\}$ **and** *s2-01*: $s2 \in \{0..1\}$ **using** *huv* **by** *auto*
show $s1 = s2$
proof (*rule ccontr*)
assume *neq*: $s1 \ne s2$
from *g*(*1*) **have** *lf*: *loop-free g* **by** (*simp add*: *simple-path-def*)
have $s1 = s2 \vee s1 = 0 \wedge s2 = 1 \vee s1 = 1 \wedge s2 = 0$
using *eq lf loop-free-def s1-01 s2-01* **by** *blast*
with *neq* **show** *False* **using** *s1 s2 huv hne* **by** *auto*
qed
qed

lemma *arc-Re-inj-on*:
assumes *inj-on g* $\{u..v\}$
and $\bigwedge s1\ s2.\ [\![s1 \in \{u..v\};\ s2 \in \{u..v\};\ Re\ (g\ s1) = Re\ (g\ s2);\ s1 \ne s2]\!] \Longrightarrow Re\ (g\ u) = Re\ (g\ v)$
and $Re\ (g\ u) \ne Re\ (g\ v)$
shows *inj-on Re* (*g* ‘ $\{u..v\}$)
using *assms* **by** (*blast intro*: *inj-onI*)

lemma *Re-inj-upper-gen*:
assumes *s1t*: $s1 \in \{0..t\}$ **and** *s2t*: $s2 \in \{0..t\}$
and *Re-eq*: $Re\ (g\ s1) = Re\ (g\ s2)$ **and** *neq*: $s1 \ne s2$
and *geq0*: $g\ 0 = 0\ g\ 1 = 0$
and *ht*: $0 < t\ t < 1\ g\ t = b$
shows $(s1 = 0 \wedge s2 = t) \vee (s1 = t \wedge s2 = 0)$
proof (*rule ccontr*)
assume *not-endpts*: $\neg\ ((s1 = 0 \wedge s2 = t) \vee (s1 = t \wedge s2 = 0))$
— Define $S \equiv$ *convex hull path-image g*. This is the right set for *frontier-vertical-at-most-two*.
define S **where** $S \equiv$ *convex hull* (*path-image g*)

have *S-convex*: *convex S* **unfolding** *S-def* **by** (*rule convex-convex-hull*)
have *S-compact*: *compact S* **unfolding** *S-def*
using *compact-simple-path-image*[*OF g*(*1*)] *compact-convex-hull* **by** *auto*
have *frontier-S*: *frontier S* = *path-image g*
unfolding *S-def* **using** *frontier-convex-hull-eq-path-image*[*OF g*(*1*) - *conv*] *g*(*2*,*3*) **by** *auto*
have *S-int-ne*: *interior S* ≠ {}
using *Jordan-inside-outside*[*OF g*(*1*)] *g*
by (*metis Diff-empty S-convex convex-closure frontier-S frontier-def inside-convex*)
— Step 2: $g\,s_1 \neq g\,s_2$ (from injectivity of g on $\{0..t\}$, which is a proper sub-arc).
have *inj-sub*: *inj-on g* {*0*..*t*}
using *arc-inj-on*[*of 0 t*] *ht* **by** *auto*
have *g-neq*: *g s1* ≠ *g s2*
by (*meson neq inj-on-def inj-sub s1t s2t*)
— Step 3: Both $g\,s_1$ and $g\,s_2$ are on *frontier S* = *path-image g*.
have *s1-01*: *s1* ∈ {*0*..*1*} **using** *s1t ht*(*2*) **by** *auto*
have *s2-01*: *s2* ∈ {*0*..*1*} **using** *s2t ht*(*2*) **by** *auto*
have *gs1-frontier*: *g s1* ∈ *frontier S*
using *frontier-S s1-01* **by** (*auto simp*: *path-image-def*)
have *gs2-frontier*: *g s2* ∈ *frontier S*
using *frontier-S s2-01* **by** (*auto simp*: *path-image-def*)
— Find a third distinct point on *frontier S* with the same real part. Key insight: one of $g(0) = 0$ or $g(t) = b$ has a *different* parameter from s_1 and s_2, and since g is injective on $\{0..t\}$, it gives a distinct point.
define *c* **where** *c* ≡ *Re* (*g s1*)
— Show $c \in \{0, Re\,b\}$ forces s_1 or s_2 to an endpoint, contradicting *not-endpts*.
have *c-strict*: *0* < *c* ∧ *c* < *Re b*
proof −
have *a0*: *a* = *0* **using** *geq0*(*1*) *g*(*2*) **by** (*simp add*: *pathstart-def*)
have *Imb*: *Im b* = *0* **using** *b*(*3*) *a0* **by** *simp*
have *Reb*: *Re b* > *0* **using** *b*(*2*) *a0* **by** *simp*
have *bdd*: *bounded* (*path-image g*)
using *compact-simple-path-image*[*OF g*(*1*)] *compact-imp-bounded* **by** *blast*
have *gs1-pi*: *g s1* ∈ *path-image g* **and** *gs2-pi*: *g s2* ∈ *path-image g*
using *s1-01 s2-01* **by** (*auto simp*: *path-image-def*)
have *g0-pi*: *0* ∈ *path-image g* **using** *geq0*(*1*) **by** (*metis pathstart-def path-start-in-path-image*)
have *diam-eq*: *dist 0 b* = *diameter* (*path-image g*) **using** *dab a0* **by** *simp*
have *dist-0b*: *dist 0 b* = *Re b*
using *Imb Reb cmod-eq-Re* **by** *auto*
— Every point on the curve is within distance $Re\,b$ of both 0 and b
have *d1*: *dist* (*g s1*) *b* ≤ *Re b*
using *diameter-bounded-bound*[*OF bdd gs1-pi b*(*1*)] *diam-eq dist-0b* **by** *simp*
have *d2*: *dist 0* (*g s1*) ≤ *Re b*
using *diameter-bounded-bound*[*OF bdd g0-pi gs1-pi*] *diam-eq dist-0b* **by** *simp*
— Helper: from *cmod z* ≤ *Re b*, derive $(Re\,z)^2 + (Im\,z)^2 \leq (Re\,b)^2$
have *cmod-sq*: (*Re z*)^2 + (*Im z*)^2 ≤ (*Re b*)^2 **if** *cmod z* ≤ *Re b* **for** *z*
by (*metis cmod-power2 norm-ge-zero power-mono that*)
— Helper: from *cmod* $(z - b) \leq Re\,b$, derive $(Re\,z - Re\,b)^2 + (Im\,z)^2 \leq (Re\,b)^2$

have *cmod-sq-b*: $(Re\ z - Re\ b)^2 + (Im\ z)^2 \leq (Re\ b)^2$ **if** $cmod\ (z - b) \leq Re\ b$ **for** z
using *Imb cmod-sq that* **by** *force*
— Helper: injectivity gives $s = 0$ from $g\,s = 0$, and $s = t$ from $g\,s = b$
have *eq-0*: $s = 0$ **if** $g\ s = 0\ s \in \{0..t\}$ **for** s
using *geq0(1) inj-onD inj-sub that* **by** *fastforce*
have *eq-t*: $s = t$ **if** $g\ s = b\ s \in \{0..t\}$ **for** s
using *ht(3) inj-on-contraD inj-sub that* **by** *fastforce*
— Case $c = 0$: $Re(g\,s_1) = 0$, and $dist\,(g\,s_1)\,b \leq Re\,b$ forces $Im(g\,s_1) = 0$, so $g\,s_1 = 0$
have $c \neq 0$
proof
assume $c = 0$
have $(Re\ (g\ s1) - Re\ b)^2 + (Im\ (g\ s1))^2 \leq (Re\ b)^2$
using *cmod-sq-b d1* **by** (*simp add*: *dist-norm*)
then have $g\ s1 = 0$ **using** ‹$c = 0$› **by** (*auto simp*: *c-def complex-eq-iff*)
then have $s1 = 0$ **using** *eq-0 s1t* **by** *simp*
moreover have $Re\ (g\ s2) = 0$ **using** *Re-eq* ‹$c = 0$› *c-def* **by** *simp*
then have $(Re\ (g\ s2) - Re\ b)^2 + (Im\ (g\ s2))^2 \leq (Re\ b)^2$
using *cmod-sq-b*[*of g s2*] *diameter-bounded-bound*[*OF bdd gs2-pi b*(*1*)] *diam-eq dist-0b* **by** (*simp add*: *dist-norm*)
then have $g\ s2 = 0$ **using** ‹$Re\ (g\ s2) = 0$› **by** (*auto simp*: *complex-eq-iff*)
then have $s2 = 0$ **using** *eq-0 s2t* **by** *simp*
ultimately show *False* **using** *neq* **by** *simp*
qed
— Case $c = Re\,b$: $dist\,0\,(g\,s_1) \leq Re\,b$ forces $Im(g\,s_1) = 0$, so $g\,s_1 = b$
moreover have $c \neq Re\ b$
proof
assume *c*: $c = Re\ b$
have $(Re\ (g\ s1))^2 + (Im\ (g\ s1))^2 \leq (Re\ b)^2$
using *cmod-sq d2* **by** (*simp add*: *dist-norm*)
then have $g\ s1 = b$ **using** *c Imb* **by** (*auto simp*: *c-def complex-eq-iff*)
then have $s1 = t$ **using** *eq-t s1t* **by** *simp*
moreover have $Re\ (g\ s2) = Re\ b$ **using** *Re-eq* ‹$c = Re\ b$› *c-def* **by** *simp*
then have $(Re\ (g\ s2))^2 + (Im\ (g\ s2))^2 \leq (Re\ b)^2$
using *cmod-sq*[*of g s2*] *diameter-bounded-bound*[*OF bdd g0-pi gs2-pi*] *diam-eq dist-0b* **by** (*simp add*: *dist-norm*)
then have $g\ s2 = b$ **using** ‹$Re\ (g\ s2) = Re\ b$› *Imb* **by** (*simp add*: *complex.expand*)
then have $s2 = t$ **using** *eq-t s2t* **by** *simp*
ultimately show *False* **using** *neq* **by** *simp*
qed
— c is bounded: $0 \leq c \leq Re\,b$ from the diameter bound
moreover have $0 \leq c$
using *abs-Re-le-cmod*[*of g s1 − b*] *d1 c-def* **by** (*simp add*: *dist-norm*)
moreover have $c \leq Re\ b$
by (*smt* (*verit*) *c-def complex-Re-le-cmod d2 dist-0-norm*)
ultimately show *?thesis* **by** *linarith*
qed

— By the IVT on $\{t..1\}$, find s_3 with $Re(g\,s_3) = c$.
have *cont-Re-g*: *continuous-on* $\{t..1\}$ $(Re \circ g)$
using *absolutely-continuous-on-imp-continuous assms(2) cont continuous-on-Re continuous-on-eq continuous-on-subset* **by** *fastforce*
obtain *s3* **where** *s3*: $s3 \in \{t..1\}$ *Re* $(g\ s3) = c$
proof −
have *Re* $(g\ t) \in (Re \circ g)$ ‘ $\{t..1\}$ **using** *ht* **by** (*auto simp*: *image-def*)
then have *Regt-in*: *Re* $b \in (Re \circ g)$ ‘ $\{t..1\}$ **using** *ht(3)* **by** *simp*
have *Re* $(g\ 1) \in (Re \circ g)$ ‘ $\{t..1\}$ **using** *ht(2)* **by** (*auto simp*: *image-def*)
then have *Re0-in*: $0 \in (Re \circ g)$ ‘ $\{t..1\}$ **using** *geq0* **by** *simp*
have *is-interval* $((Re \circ g)$ ‘ $\{t..1\})$
using *connected-Icc connected-continuous-image cont-Re-g is-interval-connected-1*
by *blast*
then have $c \in (Re \circ g)$ ‘ $\{t..1\}$
using *Regt-in Re0-in c-strict* **by** (*auto simp*: *is-interval-1*)
then show *?thesis* **using** *that* **by** (*auto simp*: *image-def*)
qed
— $g\,s_3$ is on *frontier* S and distinct from $g\,s_1$ and $g\,s_2$.
have *s3-01*: $s3 \in \{0..1\}$ **using** *s3(1) ht(1)* **by** *auto*
have *loopfr-g*: *loop-free g* **using** *g* **by** (*simp add*: *simple-path-def*)
have *gs3-frontier*: *g s3* $\in$ *frontier S*
using *frontier-S s3-01* **by** (*auto simp*: *path-image-def*)
have *gs3-ne-gs1*: *g s3* $\neq$ *g s1*
proof
assume *eq*: *g s3* = *g s1*
with *loopfr-g* **have** $s3 = s1 \vee s3 = 0 \wedge s1 = 1 \vee s3 = 1 \wedge s1 = 0$
using *eq loop-free-def s1-01 s3-01* **by** *blast*
then show *False*
using *assms(1) c-def c-strict eq geq0(2) ht(3) s3(1)* **by** *auto*
qed
have *gs3-ne-gs2*: *g s3* $\neq$ *g s2*
proof
assume *eq*: *g s3* = *g s2*
with *loopfr-g* **have** $s3 = s2 \vee s3 = 0 \wedge s2 = 1 \vee s3 = 1 \wedge s2 = 0$
using *eq loop-free-def s2-01 s3-01* **by** *blast*
then show *False*
using *assms(2) c-strict geq0(2) ht(3) s3(1,2)* **by** *fastforce*
qed
— The “sides” condition for *frontier-vertical-at-most-two*.
have *side-left*: $\exists\, p \in S.$ *Re* $p < c$
by (*metis S-def assms(5) c-strict hull-inc pathstart-def pathstart-in-path-image zero-complex.sel(1)*)
have *side-right*: $\exists\, q \in S.$ $c <$ *Re* q
by (*metis S-def b(1) c-strict hull-inc*)
— Apply *frontier-vertical-at-most-two* for the contradiction.
have *three-distinct*: *g s1* $\neq$ *g s2* $\wedge$ *g s1* $\neq$ *g s3* $\wedge$ *g s2* $\neq$ *g s3*
using *g-neq gs3-ne-gs1 gs3-ne-gs2* **by** *auto*
have *Re-all-c*: *Re* $(g\ s1) = c$ *Re* $(g\ s2) = c$ *Re* $(g\ s3) = c$
unfolding *c-def* **using** *Re-eq s3(2) c-def* **by** *auto*

have $\neg$ (*g s1* $\neq$ *g s2* $\wedge$ *g s1* $\neq$ *g s3* $\wedge$ *g s2* $\neq$ *g s3*)
using *frontier-vertical-at-most-two*[*OF S-convex S-compact S-int-ne side-left side-right*
gs1-frontier gs2-frontier gs3-frontier Re-all-c] **.**
then show *False* **using** *three-distinct* **by** *auto*
qed

lemma *not-all-above*:
assumes *Reb*: *Re b* > *0*
assumes *a* = *0*
assumes *real-on-curve*: $\bigwedge$*z*. *z* $\in$ *path-image g* $\Longrightarrow$ *Im z* = *0* $\Longrightarrow$ *z* = *0* $\vee$ *z* = *b*
shows $\neg$ (*path-image g* $\subseteq$ {*z*. *0* $\leq$ *Im z*})
proof
assume *path-image g* $\subseteq$ {*z*. *0* $\leq$ *Im z*}
then have *convex hull* (*path-image g*) $\subseteq$ {*z*. *0* $\leq$ *Im z*}
by (*intro hull-minimal convex-halfspace-Im-ge*)
then have *inside* (*path-image g*) $\subseteq$ {*z*. (*0*::*real*) $\leq$ i $\cdot$ *z*}
using *closure-subset conv convex-hull-eq-closure-inside g* **by** *auto*
then have *inside* (*path-image g*) $\subseteq$ *interior* {*z*. (*0*::*real*) $\leq$ i $\cdot$ *z*}
using *interior-maximal open-inside Jordan-inside-outside g* **by** *blast*
also have ... = {*z*. *0* < i $\cdot$ *z*}
by (*rule interior-halfspace-ge*) *simp*
finally have $*$: *inside* (*path-image g*) $\subseteq$ {*z*. *0* < *Im z*}
by *simp*
have *seg-infinite*: $\neg$ *finite* (*open-segment a b*)
using *Reb assms* **by** *force*
have *seg-Im0*: *open-segment a b* $\subseteq$ {*z*. *Im z* = *0*}
using *assms b* **by** (*auto simp*: *in-segment complex-eq-iff*)
then have *seg-in-closure*: *open-segment a b* $\subseteq$ *closure* (*inside* (*path-image g*))
by (*metis b*(*1*) *conv convex-contains-open-segment convex-convex-hull convex-hull-eq-closure-inside*
g hull-inc pathfinish-in-path-image)
have *open-segment a b* $\subseteq$ {*z* $\in$ *path-image g*. *Im z* = *0*}
using $*$ *Jordan-inside-outside*[*of g*] *frontier-def g interior-open seg-Im0 seg-in-closure*
by *fastforce*
also have ... $\subseteq$ {*0*, *b*}
using *real-on-curve* **by** *blast*
finally show *False* **using** *seg-infinite finite-subset* **by** *blast*
qed

interpretation *CR*: *Green gop gop'* (*λt*. *1*−*t*) ‘ *U cnj a cnj b*
by (*rule cnj-rev*)

lemma *Green-area-zero-A*:
assumes *a* = *0*
and *t*: *0* < *t t* < *1*
and *hgt*: *g t* = *b*
and *above*: *g* ‘ {*0*..*t*} $\subseteq$ {*z*. *0* $\leq$ *Im z*}
and *below*: *g* ‘ {*t*..*1*} $\subseteq$ {*z*. *Im z* $\leq$ *0*}

shows *Green-concl g g′*
proof −
— Common facts used by both *split-case* and *split-case′*
have *g0*: *g 0 = 0* **using** *assms g(2)* **by** (*simp add*: *pathstart-def*)
have *g1*: *g 1 = 0* **using** *assms g(3)* **by** (*simp add*: *pathfinish-def*)
have *Reb*: *Re b > 0* **and** *Imb*: *Im b = 0* **using** *b assms* **by** *auto*
— Re-injectivity: on each arc, $Re \circ g$ is injective (except at endpoints). Otherwise *frontier-vertical-at-most-two* gives a contradiction via three points on *frontier* (*closure* (*inside*)) with the same real part.
have *Re-inj-upper*: ⟦*s1 ∈ {0..t}*; *s2 ∈ {0..t}*; *Re (g s1) = Re (g s2)*; *s1 ≠ s2*⟧
⟹ *(s1 = 0 ∧ s2 = t) ∨ (s1 = t ∧ s2 = 0)* **for** *s1 s2*
using *Re-inj-upper-gen g0 g1* **using** *hgt t* **by** *presburger*
have *Re-inj-lower*: ⟦*s1 ∈ {t..1}*; *s2 ∈ {t..1}*; *Re (g s1) = Re (g s2)*; *s1 ≠ s2*⟧
⟹ *(s1 = t ∧ s2 = 1) ∨ (s1 = 1 ∧ s2 = t)* **for** *s1 s2*
using *CR.Re-inj-upper-gen*[*of 1−s1 1−t 1−s2*] *hgt t* **using** *g g0 g1 assms*
by (*auto simp*: *gop-def reversepath-def*)
— Step 0: absolute integrability (needed for integral splitting)
define *f* **where** *f ≡ λs. Re (g′ s) ∗ Im (g s)*
have *f-int*: *f integrable-on {0..1}*
using *set-lebesgue-integral-eq-integral(1)*[*OF f-abs-int*] *f-def* **by** *argo*
have *split-int*: *integral {0..1} f = integral {0..t} f + integral {t..1} f*
using *Henstock-Kurzweil-Integration.integral-combine*[*of 0 t 1 f*] *t f-int* **by** *auto*
— Upper arc integral ≥ 0. By the change of variables $x = Re(g(s))$ and Re-injectivity, the integral $\int_0^t Re(g') \cdot Im(g)\, ds = \int_0^{Re\, b} f_{upper}(x)\, dx \geq 0$ since $f_{upper} = Im \circ g \circ Re^{-1} \geq 0$ on the upper arc.
interpret *A0t*: *Area g g′ 0 t U*
proof
show *Re (g 0) ≤ Re (g t)*
using *g0 hgt Reb* **by** *simp*
show *absolutely-continuous-on {0..t} g*
using *absolutely-continuous-on-subset*[*OF cont*] *t* **by** *auto*
show *inj-on g {0..t}*
using *arc-inj-on t less-eq-real-def* **by** *presburger*
then show *inj-on Re (g ‘ {0..t})*
using *Reb Re-inj-upper g0 t*
by (*intro arc-Re-inj-on*; *fastforce simp*: *assms b(2)*)
qed (*use above U vder t* **in** *auto*)
have *upper-int*: *integral {0..t} f ≥ 0*
unfolding *f-def* **using** *A0t.below-arclet(2)* **by** *auto*
— Lower arc integral ≥ 0 as well.
have *Re-le′*: *Re (g 1) ≤ Re (g t)* **using** *g1 hgt Reb* **by** *simp*
have *ac-sub′*: *absolutely-continuous-on {t..1} g*
using *absolutely-continuous-on-subset*[*OF cont*] *t* **by** *auto*
have *inj-g-lower*: *inj-on g {t..1}*
using *arc-inj-on t(2) less-eq-real-def t* **by** *presburger*
then have *inj-Re-lower*: *inj-on Re (g ‘ {t..1})*
using *Reb Re-inj-lower g1 t*
by (*intro arc-Re-inj-on*; *fastforce simp*: *assms b(2)*)
have *lower-int*: *integral {t..1} f ≥ 0*

unfolding *f-def*
using *t vder area-above-arclet(2)*[*OF - Re-le′ ac-sub′ below inj-g-lower inj-Re-lower U*]
by *auto*
— Total integral = area of the inside. The inside decomposes as the region between the two arcs: *inside* (*path_image g*) = $\{z \mid Re\, z \in (0, Re\, b) \wedge f_{lower}(Re\, z) < Im\, z < f_{upper}(Re\, z)\}$. By Fubini, its area = $\int_0^{Re\, b} (f_{upper}(x) - f_{lower}(x))\, dx$, and by the change-of-variables computations above, this equals *integral* $\{0..t\}$ *f* + *integral* $\{t..1\}$ *f* = *integral* $\{0..1\}$ *f*.
have *area-decomp*: *measure lebesgue* (*inside* (*path-image g*)) = *integral* $\{0..t\}$ *f* + *integral* $\{t..1\}$ *f*
proof −
— Re-derive the integral-equals-measure identities (proved locally in *upper-int*/*lower-int*)
have *Re-le*: *Re* (*g 0*) ≤ *Re* (*g t*) **and** *Re-le′*: *Re* (*g 1*) ≤ *Re* (*g t*)
using *g0 g1 hgt Reb* **by** *auto*
— The integral-equals-measure identities
define *Au* **where** *Au* ≡ {*z*. ∃ *w* ∈ *g* ‘ $\{0..t\}$. *Re w* = *Re z* ∧ *0* ≤ *Im z* ∧ *Im z* ≤ *Im w*}
define *Al* **where** *Al* ≡ {*z*. ∃ *w* ∈ *g* ‘ $\{t..1\}$. *Re w* = *Re z* ∧ *Im w* ≤ *Im z* ∧ *Im z* ≤ *0*}
have *int-upper*: *integral* $\{0..t\}$ *f* = *measure lebesgue Au*
using *A0t.below-arclet(2)* **unfolding** *f-def Au-def* **by** *auto*
have *int-lower*: *integral* $\{t..1\}$ *f* = *measure lebesgue Al*
using *t vder area-above-arclet(2)*[*OF - Re-le′ ac-sub′ below inj-g-lower inj-Re-lower U*]
unfolding *f-def Al-def* **by** *auto*
— Step A: *Au* and *Al* are measurable (compact, hence *lmeasurable*)
have *cont-g-upper*: *continuous-on* $\{0..t\}$ *g*
by (*simp add*: *absolutely-continuous-on-imp-continuous A0t.acont-g*)
define φ **where** $\varphi \equiv \lambda(s,r).$ *Complex* (*Re* (*g s*)) (*r* ∗ *Im* (*g s*))
have *cont-*φ: *continuous-on* ($\{0..t\} \times \{0..1\}$) φ
unfolding φ*-def split-def*
by (*intro continuous-intros continuous-on-compose2*[*OF cont-g-upper*] *continuous-on-fst*) *auto*
have *img*: φ ‘ ($\{0..t\} \times \{0..1\}$) = *Au*
proof (*rule set-eqI*)
fix *z*
show *z* ∈ φ ‘ ($\{0..t\} \times \{0..1\}$) ⟷ *z* ∈ *Au*
proof
assume *z* ∈ φ ‘ ($\{0..t\} \times \{0..1\}$)
then obtain *s r* **where** *sr*: *s* ∈ $\{0..t\}$ *r* ∈ $\{0..1\}$ *z* = *Complex* (*Re* (*g s*)) (*r* ∗ *Im* (*g s*))
unfolding φ*-def* **by** *auto*
have *Im-ge*: *Im* (*g s*) ≥ *0*
using *subsetD*[*OF above imageI*[*OF sr(1)*]] **by** *simp*
have *g s* ∈ *g* ‘ $\{0..t\}$ **using** *sr(1)* **by** *auto*
moreover have *Im z* ≤ *Im* (*g s*)
using *sr Im-ge* **by** (*auto simp*: *mult-left-le-one-le*)
ultimately show *z* ∈ *Au* **using** *sr Im-ge* **unfolding** *Au-def* **by** *auto*

```
    next
      assume z ∈ Au
      then obtain w where w: w ∈ g ‘ {0..t} Re w = Re z 0 ≤ Im z Im z ≤ Im
w
        unfolding Au-def by auto
      then obtain s where s: s ∈ {0..t} w = g s by auto
      show z ∈ φ ‘ ({0..t} × {0..1})
      proof (cases Im w = 0)
        case True
         then have z = φ (s, 0) unfolding φ-def using w s(2) by (simp add:
complex-eq-iff)
        then show ?thesis using s(1) by auto
      next
        case False
        define r where r ≡ Im z / Im w
        have Im w > 0 using False w(3,4) by linarith
        moreover have z = φ (s, r)
           unfolding φ-def r-def using False w s(2) by (simp add: field-simps
complex-eq-iff)
        ultimately show ?thesis using w s(1) by (auto simp: r-def)
      qed
    qed
  qed
  have Au-meas: Au ∈ lmeasurable
     using img compact-continuous-image[OF cont-φ] lmeasurable-compact by
(metis compact-Icc compact-Times)
  have cont-g-lower: continuous-on {t..1} g
    using absolutely-continuous-on-imp-continuous[OF ac-sub'] is-interval-cc by
blast
  define ψ where ψ ≡ λ(s::real, r::real). Complex (Re (g s)) (r ∗ Im (g s))
  have cont-ψ: continuous-on ({t..1} × {0..1}) ψ
    unfolding ψ-def split-def
   by (intro continuous-intros continuous-on-compose2[OF cont-g-lower] contin-
uous-on-fst) auto
  have img: ψ ‘ ({t..1} × {0..1}) = Al
  proof (rule set-eqI)
    fix z :: complex
    show z ∈ ψ ‘ ({t..1} × {0..1}) ⟷ z ∈ Al
    proof
      assume z ∈ ψ ‘ ({t..1} × {0..1})
      then obtain s r where sr: s ∈ {t..1} r ∈ {0..1} z = Complex (Re (g s))
(r ∗ Im (g s))
        unfolding ψ-def by auto
      have Im-le: Im (g s) ≤ 0
        using subsetD[OF below imageI[OF sr(1)]] by simp
      then have Im (g s) ≤ Im z
        using less-eq-real-def sr by fastforce
      moreover have Im z ≤ 0
        using sr Im-le mult-nonneg-nonpos[of r Im (g s)] by simp
```

```
      ultimately show z ∈ Al
        unfolding Al-def using sr by auto
    next
      assume z ∈ Al
      then obtain w where w: w ∈ g ‘ {t..1} Re w = Re z Im w ≤ Im z Im z
≤ 0
        unfolding Al-def by auto
      then obtain s where s: s ∈ {t..1} w = g s by auto
      show z ∈ ψ ‘ ({t..1} × {0..1})
      proof (cases Im w = 0)
        case True
         then have z = ψ (s, 0) unfolding ψ-def using w s(2) by (simp add:
complex-eq-iff)
        then show ?thesis using s(1) by auto
      next
        case False
        define r where r ≡ Im z / Im w
        have r ∈ {0..1} unfolding r-def using w
          using False by (auto simp: field-simps)
        moreover have z = ψ (s, r)
       unfolding ψ-def r-def using False w(2) s(2) by (simp add: complex-eq-iff)
        ultimately show ?thesis using s(1) by auto
      qed
    qed
  qed
  have compact ({t..1} × {0..1::real}) by (intro compact-Times compact-Icc)
  then have Al-meas: Al ∈ lmeasurable — duality
   using img compact-continuous-image[OF cont-ψ] lmeasurable-compact by blast

  have ch-eq: convex hull (path-image g) = closure (inside (path-image g))
    using convex-hull-eq-closure-inside[OF g(1) - conv] g(2,3) by auto
  have zero-in-ch: 0 ∈ convex hull (path-image g)
    using hull-subset[of path-image g convex] g0
    by (auto simp: path-image-def intro!: imageI[of 0])
  have b-in-ch: b ∈ convex hull (path-image g)
    using hull-subset[of path-image g convex] b(1) by auto
  have bdd-pi: bounded (path-image g)
    using compact-simple-path-image[OF g(1)] compact-imp-bounded by blast
      — Key fact: every point on the path has Re ∈ [0, Re b]
  have zero-in-pi: (0::complex) ∈ path-image g
    using g0 by (auto simp: path-image-def intro!: imageI[of 0])
  have Re-bounds: 0 ≤ Re w ∧ Re w ≤ Re b if w ∈ path-image g for w
  proof −
    have diameter (path-image g) = dist 0 b using dab g0 g1 assms by simp
    then have diam-eq: diameter (path-image g) = Re b
      using Imb Re-le cmod-eq-Re g0 hgt by auto
    have cmod w ≤ Re b
      using diameter-bounded-bound[OF bdd-pi that zero-in-pi] diam-eq by (simp
add: dist-norm)
```

moreover have *cmod* $(w - b) \leq$ *Re b*
using *diameter-bounded-bound*[*OF bdd-pi that b*(*1*)] *diam-eq* **by** (*simp add*: *dist-norm*)
ultimately show *?thesis* **using** *abs-Re-le-cmod*[*of w*] *abs-Re-le-cmod*[*of w* − *b*] **by** *auto*
qed
— Sublemma: *Complex* (*Re w*) *0* ∈ *closed-segment 0 b* for any *w* on the path
have *real-point-in-seg*: *Complex* (*Re w*) *0* ∈ *closed-segment 0 b*
if *w* ∈ *path-image g* **for** *w*
proof −
have *bds*: *0* ≤ *Re w Re w* ≤ *Re b* **using** *Re-bounds*[*OF that*] **by** *auto*
define *u* **where** *u* ≡ *Re w* / *Re b*
have *0* ≤ *u u* ≤ *1* **unfolding** *u-def* **using** *bds Reb* **by** *auto*
have *Complex* (*Re w*) *0* = (*1* − *u*) ∗$_R$ *0* + *u* ∗$_R$ *b*
unfolding *u-def* **using** *Reb Imb*
by (*simp add*: *complex-eq-iff scaleR-complex.ctr*)
then show *?thesis* **using** ‹*0* ≤ *u*› ‹*u* ≤ *1*›
unfolding *closed-segment-def* **by** *auto*
qed
— Sublemma: any *z* between $p =$ *Complex* (*Re w*) 0 and *w* is in the convex hull
have *in-ch-via-seg*: *z* ∈ *convex hull* (*path-image g*)
if *w-pi*: *w* ∈ *path-image g*
and *Re-eq*: *Re w* = *Re z*
and *Im-between*: (*0* ≤ *Im z* ∧ *Im z* ≤ *Im w*) ∨ (*Im w* ≤ *Im z* ∧ *Im z* ≤ *0*)
for *z w*
proof −
define *p* **where** *p* ≡ *Complex* (*Re w*) *0*
have *p-in-ch*: *p* ∈ *convex hull* (*path-image g*)
using *b-in-ch closed-segment-subset p-def real-point-in-seg w-pi zero-in-ch* **by** *blast*
have *w-in-ch*: *w* ∈ *convex hull* (*path-image g*)
using *hull-subset*[*of path-image g convex*] *w-pi* **by** *auto*
show *z* ∈ *convex hull* (*path-image g*)
proof (*cases Im w* = *0*)
case *True*
with *that* **show** *?thesis* **using** *p-in-ch*
unfolding *p-def*
by (*metis complex.exhaust-sel*[*of z*] *nle-le*)
next
case *False*
define *u* **where** *u* ≡ *Im z* / *Im w*
have *0* ≤ *u u* ≤ *1* **unfolding** *u-def* **using** *Im-between False*
by (*auto simp*: *field-simps split*: *if-splits*)
have *z* = (*1* − *u*) ∗$_R$ *p* + *u* ∗$_R$ *w*
using *False* **by** (*simp add*: *Re-eq p-def u-def complex-eq-iff scaleR-complex.ctr field-simps*)
then show *?thesis*
by (*simp add*: ‹*0* ≤ *u*› ‹*u* ≤ *1*› *convexD-alt p-in-ch w-in-ch*)
qed

qed
have *Au-sub*: *Au* ⊆ *convex hull* (*path-image g*)
proof (*rule subsetI*)
fix *z* **assume** *z* ∈ *Au*
then obtain *w* **where** *w* ∈ *g* ‘ {*0..t*} *Re w* = *Re z 0* ≤ *Im z Im z* ≤ *Im w*
unfolding *Au-def* **by** *auto*
then show *z* ∈ *convex hull* (*path-image g*)
using *t in-ch-via-seg*[*of w z*] **by** (*auto simp*: *path-image-def*)
qed
have *Al-sub*: *Al* ⊆ *convex hull* (*path-image g*)
proof (*rule subsetI*)
fix *z* **assume** *z* ∈ *Al*
then obtain *w* **where** *w* ∈ *g* ‘ {*t..1*} *Re w* = *Re z Im w* ≤ *Im z Im z* ≤ *0*
unfolding *Al-def* **by** *auto*
then show *z* ∈ *convex hull* (*path-image g*)
using *t in-ch-via-seg*[*of w z*] **by** (*auto simp*: *path-image-def*)
qed
have *Au-Al-sub-closure*: *Au* ∪ *Al* ⊆ *closure* (*inside* (*path-image g*))
using *Au-sub Al-sub ch-eq* **by** *auto*

have *inside-sub-Au-Al*: *inside* (*path-image g*) ⊆ *Au* ∪ *Al*
proof (*rule subsetI*)
fix *z* **assume** *z-in*: *z* ∈ *inside* (*path-image g*)
— Set up the convex hull S and its key properties
define *S* **where** *S* ≡ *convex hull* (*path-image g*)
have *S-bounded*: *bounded S*
by (*simp add*: *S-def bounded-convex-hull bounded-simple-path-image g*)
have *frontier-S*: *frontier S* = *path-image g*
unfolding *S-def* **using** *frontier-convex-hull-eq-path-image*[*OF g(1) - conv*] *g*(*2*,*3*) **by** *auto*
have *inside-eq-int*: *inside* (*path-image g*) = *interior S*
using *S-bounded S-def frontier-S inside-frontier-eq-interior* **by** *force*
have *S-int-ne*: *interior S* ≠ {}
using *z-in inside-eq-int* **by** *auto*
have *rel-fr-eq*: *rel-frontier S* = *frontier S*
using *rel-frontier-nonempty-interior*[*OF S-int-ne*] **.**
have *z-int*: *z* ∈ *interior S* **and** *z-rel-int*: *z* ∈ *rel-interior S*
using *z-in inside-eq-int rel-interior-nonempty-interior* **by** *auto*
— Case split on the sign of *Im z*
show *z* ∈ *Au* ∪ *Al*
proof (*cases Im z* ≥ *0*)
case *True*
— Shoot a ray upward from *z* in direction i. By *ray-to-rel-frontier*, we hit a point on *frontier S* = *path-image g*.
obtain *d* **where** *d*: *d* > *0 z* + *d* ∗$_R$ i ∈ *rel-frontier S*
by (*metis S-bounded complex-i-not-zero ray-to-frontier rel-fr-eq z-int*)
define *w* **where** *w* ≡ *z* + *d* ∗$_R$ i
have *w-on-path*: *w* ∈ *path-image g*
using *d*(*2*) *rel-fr-eq frontier-S w-def* **by** *auto*

have *Re-w*: *Re w* = *Re z* **and** *Im-w*: *Im w* = *Im z* + *d* **unfolding** *w-def* **by** *auto*
have *Im-w-pos*: *Im w* > *0* **using** *True d(1) Im-w* **by** *linarith*
— Since $Im\, w > 0$ and the lower arc has $Im \leq 0$, w must be on the upper arc
have {*0..1*} = {*0..t*} ∪ {*t..1*} **using** *t* **by** (*auto simp*: *ivl-disj-un-two-touch*)
then have *w-upper*: *w* ∈ *g* ‘ {*0..t*}
using *w-on-path Im-w-pos below subsetD* **by** (*fastforce simp*: *path-image-def*)
show *z* ∈ *Au* ∪ *Al*
using *Au-def Im-w Re-w True d(1) w-upper* **by** *auto*
next
case *False*
then have *Im-z-neg*: *Im z* ≤ *0* **by** *simp*
— Shoot a ray downward from z in direction $-\mathrm{i}$
obtain *d* **where** *d*: *d* > *0 z* + *d* ∗$_R$ (−i) ∈ *frontier S*
by (*metis S-bounded complex-i-not-zero neg-equal-0-iff-equal ray-to-frontier z-int*)
have *d2*: *z* − *d* ∗$_R$ i ∈ *rel-frontier S*
using *d(2) rel-fr-eq* **by** (*simp add*: *real-vector.scale-minus-right*)
define *w* **where** *w* ≡ *z* − *d* ∗$_R$ i
have *w-on-path*: *w* ∈ *path-image g*
using *d2 rel-fr-eq frontier-S w-def* **by** *auto*
have *Re-w*: *Re w* = *Re z* **and** *Im-w*: *Im w* = *Im z* − *d* **unfolding** *w-def* **by** *auto*
have *Im-w-neg*: *Im w* < *0* **using** *Im-z-neg d(1) Im-w* **by** *linarith*
— Since $Im\, w < 0$, w must be on the lower arc
have *w-lower*: *w* ∈ *g* ‘ {*t..1*}
proof −
have {*0..1*} = {*0..t*} ∪ {*t..1*} **using** *t* **by** (*auto simp*: *ivl-disj-un-two-touch*)
then have *path-image g* = *g* ‘ {*0..t*} ∪ *g* ‘ {*t..1*}
unfolding *path-image-def* **by** (*simp add*: *image-Un*)
then have *w* ∈ *g* ‘ {*0..t*} ∪ *g* ‘ {*t..1*} **using** *w-on-path* **by** *simp*
moreover have *w* ∉ *g* ‘ {*0..t*}
using *above Im-w-neg* **by** (*auto simp*: *subset-iff*)
ultimately show *?thesis* **by** *blast*
qed
show *z* ∈ *Au* ∪ *Al*
using *Al-def Im-w Im-z-neg Re-w d(1) w-lower* **by** *auto*
qed
qed
have *inside-eq*: *measure lebesgue* (*inside* (*path-image g*)) = *measure lebesgue* (*Au* ∪ *Al*)
proof −
have *bdd-inside*: *bounded* (*inside* (*path-image g*))
and *frontier-inside*: *frontier* (*inside* (*path-image g*)) = *path-image g*
using *Jordan-inside-outside g* **by** *auto*
have *neg-frontier*: *negligible* (*frontier* (*inside* (*path-image g*)))
using *negligible-convex-frontier*[*OF conv*] **.**
have *inside-meas*: *inside* (*path-image g*) ∈ *lmeasurable*

using *measurable-Jordan*[*OF bdd-inside neg-frontier*] **.**
— The symmetric difference is contained in *path-image g*, which is negligible
have *inside* (*path-image g*) Δ (*Au* ∪ *Al*) ⊆ *path-image g*
by (*metis Au-Al-sub-closure Diff-mono Diff-subset-conv closure-Un-frontier frontier-inside*
inside-sub-Au-Al le-iff-sup)
then show *?thesis*
using *measure-negligible-symdiff*[*OF inside-meas*] *negligible-subset neg-frontier frontier-inside*
by *metis*
qed
— Step D: *Au* ∩ *Al* ⊆ {*z*. *Im z* = *0*}, which is negligible in $\mathbb{R}^2$. Therefore *measure* (*Au* ∪ *Al*) = *measure Au* + *measure Al*.
have *inter-null*: *Au* ∩ *Al* ⊆ {*z*. *Im z* = *0*}
unfolding *Au-def Al-def* **by** *auto*
have *measure lebesgue* (*Au* ∪ *Al*) = *measure lebesgue Au* + *measure lebesgue Al*
proof −
have *negligible* (*Au* ∩ *Al*)
using *negligible-hyperplane*[*of* i *0*] *negligible-subset inter-null* **by** *auto*
then have *measure lebesgue* (*Au* ∩ *Al*) = *0*
by (*rule negligible-imp-measure0*)
moreover have *measure lebesgue* (*Au* ∪ *Al*) = *measure lebesgue Au* + *measure lebesgue Al* − *measure lebesgue* (*Au* ∩ *Al*)
using *measure-Un3*[*of Au lebesgue Al*] *Au-meas Al-meas* **by** *auto*
ultimately show *?thesis* **by** *simp*
qed
show *?thesis*
using *inside-eq* ‹*measure lebesgue* (*Au* ∪ *Al*) = *measure lebesgue Au* + *measure lebesgue Al*›
int-upper int-lower **by** *simp*
qed
have *int-eq*: |*integral* {*0..1*} *f*| = *measure lebesgue* (*inside* (*path-image g*))
using *split-int area-decomp upper-int lower-int* **by** *linarith*
show *?thesis* **unfolding** *Green-concl-def f-def*
using *int-eq f-abs-int* **unfolding** *f-def* **by** *auto*
qed

3.4 Green's theorem special case at zero

interpretation *R*: *Green reversepath g uminus* ∘ *reversepath g′* (λ*t*. *1*−*t*) ‘ *U a b*
by (*rule rev*)

lemma *Green-area-zero*:
assumes *a* = *0*
shows *Green-concl g g′*
proof −
have *g0*: *g 0* = *0* **using** *assms g*(*2*) **by** (*simp add*: *pathstart-def*)
have *g1*: *g 1* = *0* **using** *assms g*(*3*) **by** (*simp add*: *pathfinish-def*)

define f **where** $f \equiv \lambda s.\ Re\ (g'\ s) * Im\ (g\ s)$
have *split-case*: *Green-concl g g'*
if *assms*: $a = 0$
and t: $0 < t\ t < 1$
and *hgt*: $g\ t = b$
and *above*: $g\ `\ \{0..t\} \subseteq \{z.\ 0 \le Im\ z\}$
and *below*: $g\ `\ \{t..1\} \subseteq \{z.\ Im\ z \le 0\}$
for $t :: real$
using *Green-area-zero-A*[*OF assms t hgt above below*] **.**
have *split-case'*: *Green-concl g g'*
if *assms*: $a = 0$
and t: $0 < t\ t < 1$
and *hgt*: $g\ t = b$
and *below*: $g\ `\ \{0..t\} \subseteq \{z.\ Im\ z \le 0\}$
and *above*: $g\ `\ \{t..1\} \subseteq \{z.\ 0 \le Im\ z\}$
for $t :: real$
proof −
have *Green-concl* (*reversepath g*) (*uminus* ∘ *reversepath g'*)
proof (*intro R.Green-area-zero-A*)
show *a=0 0 < 1−t 1−t < 1*
using *assms t* **by** *auto*
show *reversepath g* (*1−t*) = *b*
by (*simp add*: *hgt reversepath-def*)
show *reversepath g* ‘ {*0..1−t*} ⊆ {*z. 0 ≤ Im z*}
using *above* **by** (*force simp*: *reversepath-def image-subset-iff*)
show *reversepath g* ‘ {*1−t..1*} ⊆ {*z. Im z ≤ 0*}
using *below* **by** (*force simp*: *reversepath-def image-subset-iff*)
qed
moreover have *integral* {*0..1*} (λ*t. Re* (*reversepath g'* *t*) ∗ *Im* (*reversepath g t*))
= *integral* {*0..1*} (λ*t. Re* (*g'* *t*) ∗ *Im* (*g t*))
using *has-integral-affinity* [*of f - 0 1 −1 1*] *f-abs-int*
by (*fastforce simp add*: *reversepath-def f-def absolutely-integrable-on-def*)
ultimately show *?thesis*
using *f-abs-int* **by** (*auto simp*: *Green-concl-def*)
qed

have *Reb*: *Re b > 0* **using** *b*(*2*) *assms* **by** *simp*
have *Im-a*: *Im a = 0* **using** *assms* **by** *simp*
have *Im-b*: *Im b = 0* **using** *b*(*3*) *Im-a* **by** *simp*
have *path-g*: *path g* **using** *g*(*1*) *simple-path-imp-path* **by** *blast*
have *cont-g*: *continuous-on* {*0..1*} *g*
using *path-g* **by** (*simp add*: *path-def*)
obtain t **where** t: $g\ t = b\ 0 < t\ t < 1$
proof −
obtain t **where** *t0*: $g\ t = b\ t \in \{0..1\}$
using *b* **by** (*auto simp*: *path-image-def*)
have $0 < t$
by (*metis atLeastAtMost-iff b*(*2*) *g*(*2*) *not-le order-eq-iff pathstart-def t0*)

```
    moreover have t < 1
     by (smt (verit, best) b(2) box-real(2) g(3) mem-box-real(2) pathfinish-def t0)
    ultimately show thesis using t0 that by blast
  qed
  have g ‘ {0..t} ⊆ {z. 0 ≤ Im z} ∧ g ‘ {t..1} ⊆ {z. Im z ≤ 0} ∨
       g ‘ {0..t} ⊆ {z. Im z ≤ 0} ∧ g ‘ {t..1} ⊆ {z. 0 ≤ Im z}
  proof −
    have open-segment 0 b ⊆ frontier (inside (path-image g)) ∨
         open-segment 0 b ⊆ interior (inside (path-image g))
    proof (rule convex-open-segment-cases-alt)
      show convex (inside (path-image g))
        by (simp add: conv)
      show 0 ∈ closure (inside (path-image g))
        using hull-inc convex-hull-eq-closure-inside
        by (metis assms conv g pathfinish-in-path-image)
      show b ∈ closure (inside (path-image g))
        using hull-inc convex-hull-eq-closure-inside
        using R.g(2) b(1) conv g(1,2) by fastforce
    qed
    then consider open-segment 0 b ⊆ path-image g
      | open-segment 0 b ⊆ inside (path-image g)
      by (metis Jordan-inside-outside g interior-subset subset-trans)
    then show ?thesis
    proof cases
      case 1
      have ∗: connectedin euclidean (open-segment 0 b)
        by (simp add: convex-connected)
      with 1 have conn: connectedin (subtopology euclidean (path-image g))
                          (open-segment 0 b)
        by (simp add: connectedin-subtopology)
      have pi1: g ‘ {0<..<t} ⊆ path-image g and pi2: g ‘ {t<..<1} ⊆ path-image
g
        using t by (auto simp: path-image-def image-iff)
     have cl1: g ‘ {0<..<t} ∩ path-image g ∩ closure (path-image g ∩ g ‘ {t<..<1})
= {}
      proof −
        have lf: loop-free g using g(1) by (simp add: simple-path-def)
        have inj: inj-on g {0<..<1} using lf loop-free-inj-on by blast
        have disj: g ‘ {0<..<t} ∩ g ‘ {t..1} = {}
        proof (rule ccontr)
          assume ¬ ?thesis
          then obtain x y where xy: x ∈ {0<..<t} y ∈ {t..1} g x = g y
            by blast
          have x01: x ∈ {0..1} using xy(1) t(3) by auto
          have y01: y ∈ {0..1} using xy(2)
            using t(2) by auto
          have x ≠ y using xy(1,2) by auto
          then have x = 0 ∧ y = 1 ∨ x = 1 ∧ y = 0
            using lf xy(3) x01 y01 unfolding loop-free-def by blast
```

```
      then show False using xy(1,2) by auto
    qed
    have closed-img: closed (g ‘ {t..1})
      using compact-continuous-image[OF continuous-on-subset[OF cont-g]]
      by (simp add: compact-imp-closed less-eq-real-def t(2))
    have closure (g ‘ {t<..<1}) ⊆ g ‘ {t..1}
     using closed-img closure-minimal by (meson closure-minimal image-mono
greaterThanLessThan-subseteq-atLeastAtMost-iff order.refl less-imp-le)
    then have closure (path-image g ∩ g ‘ {t<..<1}) ⊆ g ‘ {t..1}
      by (meson Int-lower2 closure-mono dual-order.trans)
    then show ?thesis using disj by auto
  qed
 have cl2: g ‘ {t<..<1} ∩ path-image g ∩ closure (path-image g ∩ g ‘ {0<..<t})
= {}
  proof −
    have lf: loop-free g using g(1) by (simp add: simple-path-def)
    have disj: g ‘ {t<..<1} ∩ g ‘ {0..t} = {}
    proof (rule ccontr)
      assume ¬ ?thesis
      then obtain x y where xy: x ∈ {t<..<1} y ∈ {0..t} g x = g y
        by blast
      have x01: x ∈ {0..1} using xy(1) using t(2) by force
      have y01: y ∈ {0..1} using xy(2) t(3) by auto
      have x ≠ y using xy(1,2) by auto
      then have x = 0 ∧ y = 1 ∨ x = 1 ∧ y = 0
        using lf xy(3) x01 y01 unfolding loop-free-def by blast
      then show False using xy(1,2) by auto
    qed
    have closed-img: closed (g ‘ {0..t})
      using compact-continuous-image[OF continuous-on-subset[OF cont-g] -]
      by (meson atLeastatMost-subset-iff compact-imp-closed compact-interval
          eucl-less-le-not-le order.refl t(3))
    have closure (g ‘ {0<..<t}) ⊆ g ‘ {0..t}
     using closed-img closure-minimal by (meson closure-minimal image-mono
greaterThanLessThan-subseteq-atLeastAtMost-iff order.refl less-imp-le)
    then have closure (path-image g ∩ g ‘ {0<..<t}) ⊆ g ‘ {0..t}
      by (meson Int-lower2 closure-mono dual-order.trans)
    then show ?thesis using disj by auto
  qed
  have sub: open-segment 0 b ⊆ g ‘ {0<..<t} ∪ g ‘ {t<..<1}
  proof −
    have decomp: path-image g ⊆ g ‘ {0<..<t} ∪ g ‘ {t<..<1} ∪ {0, b}
    proof
      fix z assume z ∈ path-image g
     then obtain s where s: s ∈ {0..1} g s = z by (auto simp: path-image-def)
      then consider s = 0 | s ∈ {0<..<t} | s = t | s ∈ {t<..<1} | s = 1
        using t by force
      then show z ∈ g ‘ {0<..<t} ∪ g ‘ {t<..<1} ∪ {0, b}
        using ‹g s = z› t g0 g1 by blast
```

```
    qed
    have open-segment 0 b ∩ {0, b} = {}
      unfolding open-segment-def by auto
    then show ?thesis using 1 decomp by auto
  qed
  have sep: separatedin (subtopology euclidean (path-image g))
                        (g ` {0<..<t}) (g ` {t<..<1})
  proof −
    have d1: g ` {0<..<t} ∩ closure (g ` {t<..<1}) = {}
      by (metis cl1 inf.absorb-iff2 inf.commute pi1 pi2)
    have d2: g ` {t<..<1} ∩ closure (g ` {0<..<t}) = {}
      by (metis Int-absorb1 Int-commute cl2 pi1 pi2)
    have separatedin euclidean (g ` {0<..<t}) (g ` {t<..<1})
      unfolding separatedin-def using d1 d2
      by (simp add: euclidean-closure-of topspace-euclidean)
    then show ?thesis
      using pi1 pi2 by (simp add: separatedin-subtopology)
  qed
  have open-segment 0 b ⊆ g ` {0<..<t} ∨ open-segment 0 b ⊆ g ` {t<..<1}
    using connectedin-subset-separated-union[OF conn sep sub] .
  then have closed-segment 0 b ⊆ g ` {0..t} ∨ closed-segment 0 b ⊆ g ` {t..1}
    unfolding closed-segment-eq-open
    by (elim disj-forward) (use g0 g1 t in auto)
 then have seg-eq: g ` {0..t} = closed-segment 0 b ∨ g ` {t..1} = closed-segment
0 b
  proof (elim disjE)
    assume sub: closed-segment 0 b ⊆ g ` {0..t}
    have inj: inj-on g {0..t} using arc-inj-on[of 0 t] t by auto
    have continuous-on {0..t} g
      using continuous-on-subset[OF cont-g] t by auto
    then have cont-inv: continuous-on (g ` {0..t}) (the-inv-into {0..t} g)
      using continuous-on-inv-into[OF - compact-Icc inj] by metis
     have inv-mem: the-inv-into {0..t} g z ∈ {0..t} if z ∈ closed-segment 0 b
for z
      using the-inv-into-into[OF inj - order-refl] sub that by blast
    have eq0: the-inv-into {0..t} g 0 = 0
      using the-inv-into-f-f[OF inj, of 0] g0 t by auto
    have eqt: the-inv-into {0..t} g b = t
      using the-inv-into-f-f[OF inj, of t] t by auto
    have conn: connected (the-inv-into {0..t} g ` closed-segment 0 b)
       by (intro connected-continuous-image continuous-on-subset[OF cont-inv
sub]
          connected-segment)
    have 0 ∈ the-inv-into {0..t} g ` closed-segment 0 b
      using eq0 by (auto intro: rev-image-eqI ends-in-segment)
    moreover have t ∈ the-inv-into {0..t} g ` closed-segment 0 b
      using eqt by (auto intro: rev-image-eqI ends-in-segment)
    ultimately have {0..t} ⊆ the-inv-into {0..t} g ` closed-segment 0 b
      using connected-contains-Icc[OF conn] by auto
```

```
    then have g ‘ {0..t} ⊆ g ‘ (the-inv-into {0..t} g ‘ closed-segment 0 b)
      by (rule image-mono)
    also have ... ⊆ closed-segment 0 b
      using f-the-inv-into-f[OF inj] sub by (auto simp: image-image)
    finally show ?thesis using sub by auto
  next
    assume sub: closed-segment 0 b ⊆ g ‘ {t..1}
    have inj: inj-on g {t..1} using arc-inj-on[of t 1] t by auto
    have cont-sub: continuous-on {t..1} g
      using continuous-on-subset[OF cont-g] t by auto
    have cont-inv: continuous-on (g ‘ {t..1}) (the-inv-into {t..1} g)
      using continuous-on-inv-into[OF cont-sub compact-Icc inj] .
    have eqt: the-inv-into {t..1} g b = t
      using the-inv-into-f-f[OF inj, of t] t by auto
    have eq1: the-inv-into {t..1} g 0 = 1
      using the-inv-into-f-f[OF inj, of 1] g1 t by auto
    have conn: connected (the-inv-into {t..1} g ‘ closed-segment 0 b)
       by (intro connected-continuous-image continuous-on-subset[OF cont-inv
sub]
          connected-segment)
    have t ∈ the-inv-into {t..1} g ‘ closed-segment 0 b
      using eqt by (auto intro: rev-image-eqI ends-in-segment)
    moreover have 1 ∈ the-inv-into {t..1} g ‘ closed-segment 0 b
      using eq1 by (auto intro: rev-image-eqI ends-in-segment)
    ultimately have {t..1} ⊆ the-inv-into {t..1} g ‘ closed-segment 0 b
      using connected-contains-Icc[OF conn] by auto
    then have g ‘ {t..1} ⊆ g ‘ (the-inv-into {t..1} g ‘ closed-segment 0 b)
      by (rule image-mono)
    also have ... ⊆ closed-segment 0 b
      using f-the-inv-into-f[OF inj] sub by (auto simp: image-image)
    finally show ?thesis using sub by auto
  qed
    — Use convex-triple-rel-frontier to show the inside is on one side of Im = 0
  have inside-side: inside (path-image g) ⊆ {z. Im z ≤ 0} ∨
                    inside (path-image g) ⊆ {z. 0 ≤ Im z}
  proof −
    have J: inside (path-image g) ≠ {} open (inside (path-image g))
      frontier (inside (path-image g)) = path-image g
      using Jordan-inside-outside g by blast+
    have intne: interior (inside (path-image g)) ≠ {}
      using J(1,2) interior-eq by auto
   have rf-eq: rel-frontier (inside (path-image g)) = frontier (inside (path-image
g))
      using rel-frontier-nonempty-interior intne by blast
    have mid-on: midpoint 0 b ∈ path-image g
      using 1 Reb by force
    have rf0: (0::complex) ∈ rel-frontier (inside (path-image g))
      using rf-eq J(3) g0 by (auto simp: path-image-def intro!: image-eqI[of - -
0])
```

```
      have rfb: b ∈ rel-frontier (inside (path-image g))
        using rf-eq J(3) b(1) by auto
      have rfm: midpoint 0 b ∈ rel-frontier (inside (path-image g))
        using rf-eq J(3) mid-on by auto
     have ne1: (0::complex) ≠ b using b(2) assms by (auto simp: complex-eq-iff)
      have ne2: (0::complex) ≠ midpoint 0 b
        using ne1 by (simp add: midpoint-def complex-eq-iff)
      have ne3: b ≠ midpoint 0 b
        using ne1 by (simp add: midpoint-def complex-eq-iff)
      have §: i · midpoint 0 b = 0
        using Im-b by (simp add: midpoint-def complex-inner-i-left)
      show ?thesis
        using Im-b convex-triple-rel-frontier[OF conv rf0 rfb rfm ne1 ne2 ne3 - -
§]
        by (auto simp: midpoint-def complex-inner-i-left)
    qed
    have pi-sub: path-image g ⊆ closure (inside (path-image g))
      using hull-subset[of path-image g convex] convex-hull-eq-closure-inside[OF
g(1)] g(2,3) conv by force
      — The closed segment from 0 to b has Im = 0, so lies in both half-planes.
    have seg-both: closed-segment 0 b ⊆ {z. Im z ≤ 0} closed-segment 0 b ⊆ {z.
0 ≤ Im z}
      using Im-b by (auto simp: closed-segment-def)
        — If the inside is contained in a half-plane, then so is path-image g (by
closure).
    have inside (path-image g) ⊆ {z. Im z ≤ 0} ⟹ path-image g ⊆ {z. Im z ≤
0}
        using pi-sub closure-minimal[OF - closed-halfspace-le[of i 0, simplified
complex-inner-i-left]]
      by auto
    moreover
    have inside (path-image g) ⊆ {z. 0 ≤ Im z} ⟹ path-image g ⊆ {z. 0 ≤ Im
z}
        using pi-sub closure-minimal[OF - closed-halfspace-ge[of 0 i, simplified
complex-inner-i-left]]
      by auto
    ultimately show ?thesis
      using seg-eq inside-side seg-both
      by (auto simp: path-image-def image-subset-iff)
  next
    case seg-inside:2
    have real-on-curve: z = 0 ∨ z = b
      if z-on: z ∈ path-image g and z-real: Im z = 0 for z
    proof (rule ccontr)
      assume non: ¬ ?thesis
        — diameter bounds force z into closed-segment 0 b.
      have bdd: bounded (path-image g)
        using g(1) bounded-simple-path-image by blast
      have z0-on: 0 ∈ path-image g
```

using *pathstart-in-path-image*[*of g*] *g*(*2*) ‹*a=0*› **by** *simp*
have *diam-eq*: *diameter* (*path-image g*) = *Re b*
using *dab* ‹*a=0*› *Im-b Reb* **by** (*simp add*: *dist-complex-def cmod-eq-Re*)
have *d1*: *dist 0 z* ≤ *Re b*
using *diameter-bounded-bound*[*OF bdd z0-on z-on*] *diam-eq* **by** *simp*
have *d2*: *dist z b* ≤ *Re b*
using *diameter-bounded-bound*[*OF bdd z-on b*(*1*)] *diam-eq* **by** *simp*
have *Re-le*: *Re z* ≤ *Re b*
using *d1 z-real* **by** (*simp add*: *dist-complex-def cmod-eq-Re*)
have *Re-ge*: *Re z* ≥ *0*
using *d2 z-real Im-b* **by** (*simp add*: *dist-complex-def cmod-eq-Re minus-complex.sel*)
have *z-eq*: *z* = *of-real* (*Re z*)
using *z-real complex-is-Real-iff of-real-Re* **by** *metis*
have *b-eq*: *b* = *of-real* (*Re b*)
using *Im-b complex-is-Real-iff of-real-Re* **by** *metis*
have *z-in-seg*: *z* ∈ *closed-segment 0 b*
by (*metis Re-ge Re-le Reb atLeastAtMost-iff b-eq closed-segment-eq-real-ivl1 less-eq-real-def of-real-0 of-real-closed-segment z-eq*)
— Step 4: z is on the curve, so $z \notin$ *inside* (*path-image g*). Hence $z \notin$ *open-segment 0 b*. Combined with $z \in$ *closed-segment 0 b*, we get $z = 0 \vee z = b$.
have *z* ∉ *inside* (*path-image g*)
using *inside-no-overlap z-on* **by** *blast*
then have *z* ∉ *open-segment 0 b*
using *seg-inside* **by** *blast*
then show *False*
using *non z-in-seg* **by** (*auto simp*: *closed-segment-eq-open*)
qed

— *Im* ∘ *g* doesn't change sign on either arc: if it did, the IVT gives a real point in the interior of the arc, contradicting *real-on-curve* and injectivity.
have *no-cross*: (∀ *s* ∈ {*u..v*}. *Im* (*g s*) ≥ *0*) ∨ (∀ *s* ∈ {*u..v*}. *Im* (*g s*) ≤ *0*)
if *huv*: *u* < *v* {*u..v*} ⊆ {*0..1*} **and** *hinj*: *inj-on g* {*u..v*}
and *hend*: *Im* (*g u*) = *0 Im* (*g v*) = *0* **for** *u v*
proof (*rule ccontr*)
assume ¬ *?thesis*
then obtain s_1 s_2 **where** *s1*: s_1 ∈ {*u..v*} *Im* (*g* s_1) > *0*
and *s2*: s_2 ∈ {*u..v*} *Im* (*g* s_2) < *0*
by (*meson linorder-not-le*)
have *cont-uv*: *continuous-on* {*u..v*} *g*
using *cont-g continuous-on-subset huv*(*2*) **by** *blast*
— The IVT gives $s \in (u, v)$ with $Im(g\,s) = 0$
obtain *s* **where** *s*: *s* ∈ {*u..v*} *Im* (*g s*) = *0 s* ≠ *u s* ≠ *v*
proof (*cases* $s_1 \le s_2$)
case *True*
obtain *s* **where** *hs*: *s* ∈ {s_1..s_2} *Im* (*g s*) = *0*
using *ivt-decreasing-component-on-1*[*OF True*, *of g* i *0*] *continuous-on-subset*[*OF cont-uv*] *s1 s2*
by (*force simp*: *complex-inner-i-right*)

```
        then obtain s ≠ u s ≠ v s ∈ {u..v}
          using s1 s2 by force
        then show thesis using that hs by blast
      next
        case False
        then have le: s₂ ≤ s₁ by linarith
        obtain s where hs: s ∈ {s₂..s₁} Im (g s) = 0
       using ivt-increasing-component-on-1[OF le, of g i 0] continuous-on-subset[OF
cont-uv] s1 s2
          by (force simp: complex-inner-i-right)
        then obtain  s ≠ u s ≠ v s ∈ {u..v}
          using s1 s2 by force
        then show thesis using that hs by blast
      qed

      — g s is on the path, so g s ∈ {0, b} by real-on-curve
      have g s = 0 ∨ g s = b using real-on-curve s(2)
        by (metis subsetD imageI path-image-def s(1) that(2))
      — But g is injective on {u..v} and s ∈ (u, v), so g s ≠ g u and g s ≠ g v
      moreover have g s ≠ g u g s ≠ g v
        using inj-onD[OF hinj] s by auto
          — Since {g u, g v} ⊆ {0, b}, this gives the contradiction
      moreover have g u ∈ {0, b} g v ∈ {0, b}
        using real-on-curve hend huv by (auto simp: path-image-def subset-iff)
      ultimately show False
        using ‹u < v› inj-onD [OF hinj] by (auto simp: order-class.less-le)
    qed
    have no-cross-1: (∀ s ∈ {0..t}. Im (g s) ≥ 0) ∨ (∀ s ∈ {0..t}. Im (g s) ≤ 0)
      using no-cross[of 0 t] arc-inj-on[of 0 t] t g0 Im-b by auto
    have no-cross-2: (∀ s ∈ {t..1}. Im (g s) ≥ 0) ∨ (∀ s ∈ {t..1}. Im (g s) ≤ 0)
      using no-cross[of t 1] arc-inj-on[of t 1] t g1 Im-b by auto
    — Eliminate the case where both arcs are on the same side of the real axis.
    have not-all-above: ¬ (path-image g ⊆ {z. 0 ≤ Im z})
      using Reb assms not-all-above real-on-curve t Im-b by blast
    have not-all-below: ¬ (path-image g ⊆ {z. Im z ≤ 0})
      using CR.not-all-above using g g0 g1 assms Reb real-on-curve
      by (force simp add: gop-def  path-image-compose)
        — With the elimination, the 4-way case split from no-cross-1/no-cross-2
reduces to two
    have path-image g = g ‘ {0..t} ∪ g ‘ {t..1}
      unfolding path-image-def using t
      by (metis image-Un ivl-disj-un-two-touch(4) less-eq-real-def)
    with no-cross-1 no-cross-2 not-all-above not-all-below
    show ?thesis by (auto simp: image-subset-iff)
  qed
 qed
 then show ?thesis
  using assms split-case split-case′ t by blast
qed
```

3.5 Conclusion of Green's theorem and the signed area formula for a convex closed curve.

lemma *Green-invariant*:
assumes $\bigwedge g\ g'\ U\ b.$ *Green* $g\ g'\ U\ 0\ b \Longrightarrow$ *Green-concl* $g\ g'$
shows *Green-concl* $g\ g'$
proof −
have $*$: *Green-concl* $((\lambda x.\ -a+x) \circ g)\ g'$
proof (*intro assms*)
show *Green* $((+)\ (-\ a) \circ g)\ g'\ U\ 0\ (-a+b)$
proof
show *simple-path* $((+)\ (-\ a) \circ g)$
by (*simp add*: *g simple-path-translation-eq*)
show *pathstart* $((+)\ (-\ a) \circ g) = 0$
by (*simp add*: *g pathstart-compose*)
show *pathfinish* $((+)\ (-\ a) \circ g) = 0$
by (*simp add*: *g pathfinish-compose*)
show $-\ a + b \in$ *path-image* $((+)\ (-\ a) \circ g)$
by (*simp add*: *b path-image-translation*)
show *dist 0* $(-\ a + b) =$ *diameter* (*path-image* $((+)\ (-\ a) \circ g)$)
by (*metis add.commute dab diameter-translation dist-0-norm dist-commute dist-norm path-image-compose pth-2*)
show *convex* (*inside* (*path-image* $((+)\ (-\ a) \circ g)$))
by (*metis path-image-translation inside-translation convex-translation-eq conv*)
show *absolutely-continuous-on* $\{0..1\}$ $((+)\ (-\ a) \circ g)$
by (*metis* (*no-types*, *lifting*) *ext absolutely-continuous-on-add absolutely-continuous-on-const cont o-apply*)
next
fix *t*
assume $t \in \{0..1\} - U$
then show $((+)\ (-\ a) \circ g$ *has-vector-derivative* $g'\ t)$ (*at t*)
using *has-vector-derivative-shift vder* **by** *blast*
qed (*use b U* **in** *auto*)
qed
let *?G* $= \lambda t.$ *Re* $(g'\ t) *$ *Im* $(g\ t)$
show *?thesis*
unfolding *Green-concl-def*
proof (*intro conjI f-abs-int*)
have *meas-eq*: *measure lebesgue* (*inside* (*path-image* $((+)\ (-\ a) \circ g)$))
$=$ *measure lebesgue* (*inside* (*path-image g*))
by (*metis path-image-translation inside-translation measure-translation*)
have $(g'$ *has-integral* $(g\ 1\ -\ g\ 0))\ \{0..1\}$
using *fundamental-theorem-of-calculus-absolutely-continuous*[*OF U - cont*, *of g'*]
by (*metis zero-le-one vder has-vector-derivative-at-within*)
then have *int-g'*: $(g'$ *has-integral 0*$)\ \{0..1\}$
using *g* **by** (*simp add*: *pathstart-def pathfinish-def*)
have *Re-has-int*: $((\lambda t.$ *Re* $(g'\ t))$ *has-integral 0*$)\ \{0..1\}$
using *has-integral-Re*[*OF int-g'*] **by** *simp*

have *Ima-has-int*: $((\lambda t.\ Im\ a * Re\ (g'\ t))\ has\text{-}integral\ 0)\ \{0..1\}$
using *has-integral-mult-right*[*OF Re-has-int*] **by** *simp*
have *ai-translated*: $(\lambda t.\ Re\ (g'\ t) * Im\ (((+)\ (-\ a) \circ g)\ t))\ integrable\text{-}on\ \{0..1\}$
using $*$ **unfolding** *Green-concl-def absolutely-integrable-on-def* **by** *auto*
have $integral\ \{0..1\}\ ?G$
$= integral\ \{0..1\}\ (\lambda t.\ Re\ (g'\ t) * Im\ (((+)\ (-\ a) \circ g)\ t) + Im\ a * Re\ (g'\ t))$
by (*simp add*: *o-def plus-complex.sel uminus-complex.sel algebra-simps*)
also have $\ldots = integral\ \{0..1\}\ (\lambda t.\ Re\ (g'\ t) * Im\ (((+)\ (-\ a) \circ g)\ t))$
using *integral-add*[*OF ai-translated has-integral-integrable*]
Ima-has-int integral-unique[*OF Ima-has-int*] **by** *force*
finally show $|integral\ \{0..1\}\ ?G| = Sigma\text{-}Algebra.measure\ lebesgue\ (inside\ (path\text{-}image\ g))$
using $*$ *meas-eq* **unfolding** *Green-concl-def* **by** *auto*
qed
qed

theorem *area-theorem*:
obtains $(\lambda t.\ Re\ (g'\ t) * Im\ (g\ t))\ absolutely\text{-}integrable\text{-}on\ \{0..1\}$
and $|integral\ \{0..1\}\ (\lambda t.\ Re\ (g'\ t) * Im\ (g\ t))| =$
$measure\ lebesgue\ (inside\ (path\text{-}image\ g))$
using *Green.Green-area-zero Green-concl-def Green-invariant* **by** *blast*

end

end

4 Isoperimetric theorem for convex curves

theory *Isoperimetric*
imports *Green-Variant*

begin

Formalisation of the isoperimetric inequality, following John Harrison's HOL Light proof in *100/isoperimetric.ml*.

The proof has six parts:

1. Convex curve lemmas (switching between views of a convex simple closed curve)
2. The Wirtinger inequality
3. A special case of Green's theorem for convex area
4. The isoperimetric theorem for convex curves
5. Convexification of an arbitrary rectifiable simple closed curve

6. The full isoperimetric theorem

   Infrastructure is provided by the prerequisite theories:

- *Bounded-Variation*: bounded variation and vector variation
- *Absolute-Continuity*: absolute continuity and FTC
- *Rectifiable-Path*: rectifiable paths and path length
- *Green-Variant*: a form of the Green Area Theorem

  AFP dependencies:

- *Fourier*: trigonometric orthonormal system, Bessel inequality, L2 Fourier convergence (useful for Wirtinger inequality)
- *Lp* (via Fourier): Hölder inequality, Minkowski inequality

The kernel lemma: the isoperimetric inequality for a convex curve that has been normalized to arc-length parametrization with zero-mean imaginary part and diameter along the real axis starting at a point with $Re = 0$. This is where the Wirtinger inequality is applied.

lemma *isoperimetric-kernel*:
  fixes *g* :: *real* ⇒ *complex* **and** *L* :: *real* **and** *a b* :: *complex*
  assumes *0 < L*
    and *conv-in*: *convex* (*inside* (*path-image g*))
    and *ab*: *a* ∈ *path-image g b* ∈ *path-image g*
    and *dist-ab*: *dist a b* = *diameter* (*path-image g*)
    and *bma*: *b* − *a* = *of-real* (*dist a b*)
    and *ga*: *pathstart g* = *a pathfinish g* = *a*
    and *g*: *rectifiable-path g simple-path g*
    and *L*: *path-length g* = *L*
    and *arc-length*: ⋀*t*. *t* ∈ {*0..1*} ⟹ *path-length* (*subpath 0 t g*) = *L* ∗ *t*
    and *lipschitz*: ⋀*x y*. *x* ∈ {*0..1*} ⟹ *y* ∈ {*0..1*} ⟹ *dist* (*g x*) (*g y*) ≤ *L* ∗ *dist x y*
    and *Re a* = *0*
    and (*Im* ∘ *g has-integral 0*) {*0..1*}
  shows *measure lebesgue* (*inside* (*path-image g*)) ≤ *L*² / (*4* ∗ *pi*)
    and *measure lebesgue* (*inside* (*path-image g*)) = *L*² / (*4* ∗ *pi*) ⟹
      ∃ *c r*. *path-image g* = *sphere c r*
proof −
  have *a*≠*b*
   using *ab*(*1*) *dist-ab bounded-simple-path-image diameter-eq-0 g*(*2*) *nonempty-simple-path-endless*
    by *fastforce*
  have *acont-g*: *absolutely-continuous-on* {*0..1*} *g*
   by (*metis Lipschitz-imp-absolutely-continuous dist-norm lipschitz real-norm-def*)
  define *S* **where** *S* = {*x* ∈ {*0..1*}. ¬ *g differentiable* (*at x*)}
  have *negS*: *negligible S*

unfolding *S-def* **using** *Lebesgue-differentiation-thm-compact*
by (*metis* (*full-types*) *absolutely-continuous-on-imp-has-bounded-variation-on acont-g cbox-interval compact-Icc compact-imp-bounded*)
define g' **where** $g' \equiv (\lambda x.\ vector\text{-}derivative\ g\ (at\ x))$
have g'*-deriv*: $\bigwedge x.\ x \in \{0..1\} - S \Longrightarrow (g\ has\text{-}vector\text{-}derivative\ g'\ x)\ (at\ x)$
by (*simp add*: *S-def* g'*-def vector-derivative-works*)
have g'*-int*: $g'\ absolutely\text{-}integrable\text{-}on\ \{0..t\} \wedge integral\ \{0..t\}\ g' = g\ t - a$
if $t \in \{0..1::real\}$ **for** t
proof −
have *lhs*: $g'\ absolutely\text{-}integrable\text{-}on\ \{0..1\} \wedge (\forall x{\in}\{0..1\}.\ (g'\ has\text{-}integral\ g\ x - g\ 0)\ \{0..x\})$
unfolding *absolute-integral-absolutely-continuous-derivative-eq*
by (*metis has-vector-derivative-at-within acont-g negS* g'*-deriv*)
have *abs-int-t*: $g'\ absolutely\text{-}integrable\text{-}on\ \{0..t\}$
using *absolutely-integrable-on-subinterval*[*OF conjunct1*[*OF lhs*]] *that* **by** *auto*
moreover have $integral\ \{0..t\}\ g' = g\ t - a$
using *ga* **by** (*metis integral-unique lhs pathstart-def that*)
ultimately show *?thesis*
by *auto*
qed
have *norm-*g'*-int*: $(\lambda x.\ norm\ (g'\ x))\ absolutely\text{-}integrable\text{-}on\ \{0..t\} \wedge integral\ \{0..t\}\ (\lambda x.\ norm\ (g'\ x)) = L * t$
if $t \in \{0..1\}$ **for** t
proof −
have *acont-gt*: *absolutely-continuous-on* $\{0..t\}\ g$
using *absolutely-continuous-on-subset*[*OF acont-g*] *that* **by** *auto*
have g'*-deriv-t*: $\bigwedge x.\ x \in \{0..t\} - S \Longrightarrow (g\ has\text{-}vector\text{-}derivative\ g'\ x)\ (at\ x)$
using g'*-deriv that* **by** *auto*
have *vector-variation* $\{0..t\}\ g = integral\ \{0..t\}\ (\lambda u.\ norm\ (g'\ u))$
using *vector-variation-integral-norm-derivative*[*OF negS - acont-gt* g'*-deriv-t*] *that*
by *presburger*
moreover have *vector-variation* $\{0..t\}\ g = L * t$
using *that* ‹*rectifiable-path g*› *path-length-subpath-eq* [*of 0 t*, *symmetric*] *arc-length*
by (*fastforce simp*: *closed-segment-eq-real-ivl1*)
moreover have $(\lambda x.\ norm\ (g'\ x))\ absolutely\text{-}integrable\text{-}on\ \{0..t\}$
using g'*-int set-integrable-norm that* **by** *blast*
ultimately show *?thesis*
using *that* **by** *auto*
qed
have *norm-*g'*-le*: $norm\ (g'\ x) \le L$ **if** $x \in \{0..1\} - S$ **for** x
proof −
have *ev*: $\forall_F\ y\ in\ at\ x\ within\ \{0..1\}.\ norm\ (g\ y - g\ x) \le norm\ (L * y - L * x)$
unfolding *eventually-at-filter*
proof (*intro always-eventually allI impI*)
fix y **assume** $y \neq x\ y \in \{0..1\}$
have $norm\ (g\ y - g\ x) \le |L * (y - x)|$

using *lipschitz* ‹$y \in \{0..1\}$› *that* **by** (*fastforce simp*: *dist-norm*)
also have $\ldots = norm\ (L * y - L * x)$
by (*simp add*: *right-diff-distrib*)
finally show $norm\ (g\ y - g\ x) \leq norm\ (L * y - L * x)$ **.**
qed
have *gd*: (*g has-vector-derivative g' x*) (*at x*) **using** *g'-deriv that* **by** *auto*
have *xlimpt*: *x islimpt* $\{0..1::real\}$
using *limpt-of-convex*[*of* $\{0..1::real\}$ *x*] *that* **by** *auto*
have *Ld*: ($\lambda t.\ L * t$) *has-vector-derivative L*) (*at x within* $\{0..1\}$)
using *has-vector-derivative-mult-right*[*OF has-vector-derivative-id*] **by** *simp*
show $norm\ (g'\ x) \leq L$
using ‹$0 < L$› *norm-vector-derivatives-le-within* [*OF - Ld ev*] *xlimpt gd has-vector-derivative-at-within*
by (*force simp add*: *trivial-limit-within*)
qed

have *norm-g'-sq-int*: $(\lambda x.\ (norm\ (g'\ x))^2)$ *absolutely-integrable-on* $\{0..1\}$
proof (*rule measurable-bounded-by-integrable-imp-absolutely-integrable-ae*)
show $(\lambda x.\ (norm\ (g'\ x))^2) \in$ *borel-measurable* (*lebesgue-on* $\{0..1\}$)
by (*simp add*: *absolutely-integrable-imp-borel-measurable borel-measurable-power norm-g'-int*)
fix *x* **assume** $x \in \{0..1\} - S$
then show $norm\ ((norm\ (g'\ x))^2) \leq L^2$
using *norm-g'-le* **by** (*simp add*: *power-mono*)
qed (*use negS* **in** *auto*)

have *integral-norm-g'-sq*: $integral^L$ (*lebesgue-on* $\{0..1\}$) $(\lambda x.\ (norm\ (g'\ x))^2) = L^2$
proof −
let *?int01* = $\{0..1::real\}$
have *meas01*: *?int01* ∈ *sets lebesgue* **by** *simp*
have *norm-g'-leb*: *integrable* (*lebesgue-on* $\{0..1\}$) $(\lambda x.\ norm\ (g'\ x))$
using *norm-g'-int*[*of 1*] *absolutely-integrable-imp-integrable*
using *absolutely-integrable-imp-integrable*[*OF - meas01*] **by** *auto*
have *int-norm-g'*: $integral^L$ (*lebesgue-on* $\{0..1\}$) $(\lambda x.\ norm\ (g'\ x)) = L$
by (*simp add*: *lebesgue-integral-eq-integral norm-g'-int norm-g'-leb*)
have *const-leb*: *integrable* (*lebesgue-on* $\{0..1\}$) ($\lambda x::real.\ L$)
by (*simp add*: *integrable-const-ivl*)
have *int-const*: $integral^L$ (*lebesgue-on* $\{0..1\}$) ($\lambda x::real.\ L$) = *L*
using *lebesgue-integral-const*[*of lebesgue-on* $\{0..1\}$ *L*]
by (*simp add*: *measure-restrict-space*)
have *ae-le*: *AE x in lebesgue-on* $\{0..1\}$. $norm\ (g'\ x) \leq L$
proof −
have $S \cap \{0..1\} \in$ *null-sets* (*lebesgue-on* $\{0..1\}$)
using *negS negligible-iff-null-sets null-sets-restrict-space*
by (*metis inf-le2 meas01 null-set-Int2*)
then have *AE x in lebesgue-on* $\{0..1\}$. $x \notin S$
by (*metis AE-not-in Collect-subset S-def inf.orderE*)
then show *?thesis*

using *norm-g′-le* **by** (*auto elim*: *eventually-mono*)
qed
— Therefore $norm\,(g'\,x) = L$ a.e.
have *ae-eq*: *AE x in lebesgue-on* {*0..1*}. *norm* $(g'\,x) = L$
using *integral-ineq-eq-0-then-AE*[*OF ae-le norm-g′-leb const-leb*] *int-norm-g′ int-const*
by *simp*
— Therefore $(norm\,(g'\,x))^2 = L^2$ a.e.
have *ae-sq*: *AE x in lebesgue-on* {*0..1*}. $(norm\ (g'\ x))^2 = L^2$
using *ae-eq* **by** (*rule AE-mp*) *auto*
— Conclude by *integral-cong-AE*
have *meas-sq*: $(\lambda x.\ (norm\ (g'\ x))^2) \in$ *borel-measurable* (*lebesgue-on* {*0..1*})
using *absolutely-integrable-imp-borel-measurable meas01 norm-g′-sq-int* **by** *blast*
have $integral^L$ (*lebesgue-on ?int01*) $(\lambda x.\ (norm\ (g'\ x))^2) =$
$integral^L$ (*lebesgue-on ?int01*) $(\lambda x.\ L^2)$
by (*rule integral-cong-AE*[*OF meas-sq - ae-sq*]) *simp*
also have $\ldots = L^2$
using *lebesgue-integral-const* **by** (*simp add*: *measure-restrict-space*)
finally show *?thesis* **.**
qed
let *?G* = $\lambda t.$ *Re* $(g'\ t) *$ *Im* $(g\ t)$

Use the Green formula for the area inside the curve.

have *green-ai*: *?G absolutely-integrable-on* {*0..1*}
and *green-area*: $|integral\ \{0..1\}\ ?G| =$ *measure lebesgue* (*inside* (*path-image g*))
proof −
interpret *G*: *Green g g′ S a b*
proof (*intro Green.intro g ga negS g′-deriv ab dist-ab conv-in acont-g*)
show *Re a* < *Re b*
by (*metis* ‹*a*≠*b*› *Re-complex-of-real* ‹*Re a* = *0*› *bma diff-zero dist-nz minus-complex.sel*(*1*))
show *Im a* = *Im b*
using *bma* **by** (*simp add*: *complex-of-real-def complex-eq-iff*)
qed *auto*
from *G.area-theorem* **show** *?G absolutely-integrable-on* {*0..1*}
and $|integral\ \{0..1\}\ ?G| =$ *measure lebesgue* (*inside* (*path-image g*))
by (*metis* (*full-types*))+
qed

have *integrable*: *?G integrable-on* {*0..1*}
using *green-ai absolutely-integrable-on-def* **by** *blast*
obtain *sgn* :: *real* **where** *sgn2*: $sgn^2 = 1$
and *has-int-green*: (*?G has-integral* (*sgn* ∗ *measure lebesgue* (*inside* (*path-image g*)))) {*0..1*}
proof (*cases integral* {*0..1*} *?G* ≥ *0*)
case *True*
then show *thesis*

```
    using that[of 1] integrable green-area by (simp add: has-integral-integrable-integral)
  next
    case False
    then show thesis using that[of −1]
      using integrable green-area by (simp add: has-integral-iff)
  qed

  have has-int-norm-sq: ((λx. (norm (g′ x))²) has-integral L²) {0..1}
    using lebesgue-integral-eq-integral[of {0..1} λx. (norm (g′ x))²]
        absolutely-integrable-imp-integrable[OF norm-g′-sq-int]
    using integral-norm-g′-sq norm-g′-sq-int
    by (auto simp: absolutely-integrable-on-def)

  have has-int-key: ((λx. (Re (g′ x) − 2 ∗ pi ∗ sgn ∗ Im (g x))² +
    (Im (g′ x))² − (2 ∗ pi ∗ Im (g x))²) has-integral
    (L² − measure lebesgue (inside (path-image g)) ∗ 4 ∗ pi)) {0..1}
  proof −
    have integrand-eq: ⋀x. (Re (g′ x) − 2 ∗ pi ∗ sgn ∗ Im (g x))² + (Im (g′ x))²
− (2 ∗ pi ∗ Im (g x))² =
      (norm (g′ x))² − 4 ∗ pi ∗ sgn ∗ Re (g′ x) ∗ Im (g x)
      using sgn2 cmod-power2 by (simp add: power2-eq-square algebra-simps)
    have val: 4 ∗ pi ∗ sgn ∗ (sgn ∗ measure lebesgue (inside (path-image g))) =
      measure lebesgue (inside (path-image g)) ∗ 4 ∗ pi
      using sgn2 by (simp add: power2-eq-square algebra-simps)
    have ((λt. 4 ∗ pi ∗ sgn ∗ Re (g′ t) ∗ Im (g t)) has-integral
      (measure lebesgue (inside (path-image g)) ∗ 4 ∗ pi)) {0..1}
      using has-integral-mult-right[OF has-int-green, of 4 ∗ pi ∗ sgn]
      unfolding val by (simp add: algebra-simps)
    then show ?thesis
      using has-integral-diff[OF has-int-norm-sq] integrand-eq by presburger
  qed

  have key: 0 ≤ L² − measure lebesgue (inside (path-image g)) ∗ 4 ∗ pi ∧
            (L² − measure lebesgue (inside (path-image g)) ∗ 4 ∗ pi = 0 ⟶
            (∃ c r. path-image g = sphere c r))
  proof (cases inside(path-image g) = {})
    case False
    have Im-g′-has-int: ((λt. Im (g′ t)) has-integral (Im (g x) − Im (g 0))) {0..x}
      if x ∈ {0..1} for x
    using ga has-integral-Im g′-int has-integral-iff set-lebesgue-integral-eq-integral(1)
that
      by (metis pathstart-def minus-complex.simps(2))
    have Im-g-periodic: Im (g 1) = Im (g 0)
      using ga by (simp add: pathstart-def pathfinish-def)
    have Im-g-zero-mean: ((λx. Im (g x)) has-integral 0) {0..1}
      using assms by (simp add: o-def)

    have (λx. (Im (g′ x))²) absolutely-integrable-on {0..1}
    proof (rule measurable-bounded-by-integrable-imp-absolutely-integrable-ae)
```

show $(\lambda x.\ (Im\ (g'\ x))^2) \in$ *borel-measurable* (*lebesgue-on* $\{0..1\}$)
by (*simp add*: *Im-absolutely-integrable-on absolutely-integrable-imp-borel-measurable borel-measurable-power g'-int sets-completionI-sets*)
fix x **assume** $x \in \{0..1\} - S$
then have *norm* $(g'\ x) \leq L$ **using** *norm-g'-le* **by** *auto*
then show *norm* $((Im\ (g'\ x))^2) \leq L^2$
by (*metis abs-Im-le-cmod order.trans norm-ge-zero norm-power power-mono real-norm-def*)
qed (*use negS* **in** *auto*)
then have *Im-g'-sq-int*: $(\lambda x.\ (Im\ (g'\ x))^2)$ *integrable-on* $\{0..1\}$
using *absolutely-integrable-on-def* **by** *blast*
have *wirt1*: $(\lambda x.\ (Im\ (g\ x))^2)$ *integrable-on* $\{0..1\}$
and *wirt2*: *integral* $\{0..1\}$ $(\lambda x.\ (2{*}pi * Im\ (g\ x))^2) \leq$ *integral* $\{0..1\}$ $(\lambda x.\ (Im\ (g'\ x))^2)$
and *wirt3*: *integral* $\{0..1\}$ $(\lambda x.\ (2{*}pi * Im\ (g\ x))^2) =$ *integral* $\{0..1\}$ $(\lambda x.\ (Im\ (g'\ x))^2) \Longrightarrow$
$\exists\, c\ a.\ \forall\, x \in \{0..1\}.\ Im\ (g\ x) = c * sin\ (2{*}pi{*}x - a)$
using *scaled-Wirtinger-inequality*[*OF Im-g'-has-int Im-g-periodic Im-g-zero-mean Im-g'-sq-int*]
by *auto*
have *sq*: $(\lambda x.\ (2 * pi * Im\ (g\ x))^2)$ *integrable-on* $\{0..1\}$
using *integrable-cmul*[*OF wirt1*, *of* $(2{*}pi)^2$]
by (*simp add*: *power-mult-distrib mult.commute*)
then obtain w **where** w: $((\lambda x.\ (Im\ (g'\ x))^2 - (2 * pi * Im\ (g\ x))^2)$ *has-integral* $w)$ $\{0..1\}$
using *integrable-diff*[*OF Im-g'-sq-int sq*] **by** *force*
then have $w =$ *integral* $\{0..1\}$ $(\lambda x.\ (Im\ (g'\ x))^2 - (2 * pi * Im\ (g\ x))^2)$
by (*simp add*: *integral-unique*)
also have *w-eq*: $\ldots =$ *integral* $\{0..1\}$ $(\lambda x.\ (Im\ (g'\ x))^2) -$ *integral* $\{0..1\}$ $(\lambda x.\ (2 * pi * Im\ (g\ x))^2)$
using *sq integral-diff*[*OF Im-g'-sq-int*]
by (*simp add*: *power-mult-distrib mult.commute*)
finally have *w-nonneg*: $0 \leq w$
and *w-zero*: $w = 0 \Longrightarrow \exists\, c\ a.\ \forall\, x \in \{0..1\}.\ Im\ (g\ x) = c * sin\ (2{*}pi{*}x - a)$
using *wirt2 wirt3* **by** *argo+*
define d **where** $d = L^2 -$ *measure lebesgue* (*inside* (*path-image g*)) $*$ *4* $*$ *pi* $- w$
have *key-eq*: $L^2 -$ *measure lebesgue* (*inside* (*path-image g*)) $*$ *4* $*$ *pi* $= d + w$
unfolding *d-def* **by** *linarith*

have *sq-has-int*: $((\lambda x.\ (Re\ (g'\ x) - 2 * pi * sgn * Im\ (g\ x))^2)$ *has-integral* $d)$ $\{0..1\}$
using *has-integral-diff*[*OF has-int-key w*] *d-def* **by** *auto*
have *d-nonneg*: $0 \leq d$
using *integral-nonneg* [*OF has-integral-integrable*[*OF sq-has-int*]] *integral-unique*[*OF sq-has-int*]
by *simp*
have $\exists\, c\ r.$ *path-image* $g =$ *sphere* $c\ r$ **if** $d + w = 0$

proof −
have *d0*: $d = 0$ **and** *w0*: $w = 0$
using *that d-nonneg w-nonneg* **by** *linarith+*
obtain *C A* **where** *CA*: $\bigwedge x.\ x \in \{0..1\} \Longrightarrow Im\ (g\ x) = C * sin\ (2{*}pi{*}x - A)$
using *w-zero*[*OF w0*] **by** *blast*
have *sq-zero*: $((\lambda x.\ (Re\ (g'\ x) - 2 * pi * sgn * Im\ (g\ x))^2)\ has\text{-}integral\ 0)\ \{0..1\}$
using *sq-has-int d0* **by** *simp*
have *neg-Re*: $negligible\ \{x \in \{0..1\}.\ Re\ (g'\ x) - 2 * pi * sgn * Im\ (g\ x) \neq 0\}$
proof −
have *sq-abs*: $(\lambda x.\ (Re\ (g'\ x) - 2 * pi * sgn * Im\ (g\ x))^2)\ absolutely\text{-}integrable\text{-}on\ \{0..1\}$
using *nonnegative-absolutely-integrable-1*[*OF has-integral-integrable*[*OF sq-zero*]]
by (*simp add*: *zero-le-power2*)
have *sq-leb*: $integrable\ (lebesgue\text{-}on\ \{0..1\})\ (\lambda x.\ (Re\ (g'\ x) - 2 * pi * sgn * Im\ (g\ x))^2)$
by (*rule absolutely-integrable-imp-integrable*[*OF sq-abs*]) *simp*
have *leb-zero*: $integral^L\ (lebesgue\text{-}on\ \{0..1\})\ (\lambda x.\ (Re\ (g'\ x) - 2 * pi * sgn * Im\ (g\ x))^2) = 0$
using *lebesgue-integral-eq-integral*[*OF sq-leb*] *integral-unique*[*OF sq-zero*] **by** *simp*
have $AE\ x\ in\ lebesgue\text{-}on\ \{0..1\}.\ (Re\ (g'\ x) - 2 * pi * sgn * Im\ (g\ x))^2 = 0$
using *integral-nonneg-eq-0-iff-AE*[*OF sq-leb*] *leb-zero*
by (*simp add*: *zero-le-power2*)
then have *ae*: $AE\ x\ in\ lebesgue\text{-}on\ \{0..1\}.\ Re\ (g'\ x) - 2 * pi * sgn * Im\ (g\ x) = 0$
by (*rule AE-mp*) (*auto simp*: *power2-eq-square*)
then obtain *N0* **where** *N0*: $N0 \in null\text{-}sets\ (lebesgue\text{-}on\ \{0..1\})$
and *sub*: $\{x \in space\ (lebesgue\text{-}on\ \{0..1\}).\ Re\ (g'\ x) - 2 * pi * sgn * Im\ (g\ x) \neq 0\} \subseteq N0$
by (*auto simp*: *eventually-ae-filter*[*of - lebesgue-on* $\{0..1\}$])
have *negligible N0*
by (*meson N0 fmeasurableD lmeasurable-interval*(*1*) *negligible-iff-null-sets null-sets-restrict-space*)
moreover have $\{x \in \{0..1\}.\ Re\ (g'\ x) - 2 * pi * sgn * Im\ (g\ x) \neq 0\} \subseteq N0$
using *sub* **by** (*auto simp*: *space-lebesgue-on*)
ultimately show *?thesis*
by (*meson negligible-subset*)
qed
have *neg-Re'*: $negligible\ \{x \in \{0..1\}.\ Re\ (g'\ x) - 2 * pi * sgn * C * sin\ (2{*}pi{*}x - A) \neq 0\}$
using *neg-Re* **by** (*simp add*: *CA mult.assoc cong*: *conj-cong*)
have *Re-g*: $Re\ (g\ x) = - sgn * C * (cos\ (2{*}pi{*}x - A) - cos\ A)$
if *x*: $x \in \{0..1\}$ **for** *x*

proof −
have $((\lambda t.\ Re\ (g'\ t))\ has\text{-}integral\ Re\ (g\ x - a))\ \{0..x\}$
by (*metis has-integral-Re g'-int integrable-integral set-lebesgue-integral-eq-integral*(*1*) *that*)
then have *Re-g'-int*: $((\lambda t.\ Re\ (g'\ t))\ has\text{-}integral\ Re\ (g\ x))\ \{0..x\}$
using ‹*Re a* = *0*› **by** *simp*
have §: $((\lambda t. - sgn * C * cos\ (2 * pi * t - A))\ has\text{-}vector\text{-}derivative$
$2 * pi * sgn * C * sin\ (2 * pi * t - A))\ (at\ t\ within\ \{0..x\})$ **for** *t*
unfolding *has-real-derivative-iff-has-vector-derivative* [*symmetric*]
by (*intro derivative-eq-intros*) (*auto simp*: *algebra-simps*)
have *sin-int*: $((\lambda t.\ 2 * pi * sgn * C * sin\ (2 * pi * t - A))\ has\text{-}integral$
$(- sgn * C * (cos\ (2 * pi * x - A) - cos\ A)))\ \{0..x\}$
using *fundamental-theorem-of-calculus*[*OF* - §] *x* **by** (*simp add*: *algebra-simps*)
— the integrand is 0 a.e., so the integral is 0
have *neg-sub*: $negligible\ \{t \in \{0..x\}.\ Re\ (g'\ t) - 2 * pi * sgn * C * sin\ (2 * pi * t - A) \neq 0\}$
by (*rule negligible-subset*[*OF neg-Re'*]) (*use x* **in** *auto*)
have $((\lambda t.\ Re\ (g'\ t) - 2 * pi * sgn * C * sin\ (2 * pi * t - A))\ has\text{-}integral\ 0)\ \{0..x\}$
by (*rule has-integral-spike*[*OF neg-sub* - *has-integral-0*]) *auto*
then show *?thesis*
using *has-integral-unique has-integral-diff*[*OF Re-g'-int sin-int*]
using *right-minus-eq* **by** *blast*
qed
— Final step: *path-image g* = *sphere c* |*C*|
define *c* **where** *c* = *Complex* (*sgn* ∗ *C* ∗ *cos A*) *0*
have *subset*: *path-image g* ⊆ *sphere c* |*C*|
proof −
have *cmod* (*g t* − *c*) = |*C*| **if** *t* ∈ {*0..1*} **for** *t*
proof −
have *eq-gt*: *g t* − *c* = *Complex* (− *sgn* ∗ *C* ∗ *cos* (*2*∗*pi*∗*t* − *A*)) (*C* ∗ *sin* (*2*∗*pi*∗*t* − *A*))
unfolding *c-def* **using** *Re-g*[*OF that*] *CA*[*OF that*] **by** (*simp add*: *complex-eqI algebra-simps*)
have $(cmod\ (g\ t - c))^2 = (sgn * C * cos\ (2*pi*t - A))^2 + (C * sin\ (2*pi*t - A))^2$
using *eq-gt* **by** (*simp add*: *complex-norm power2-eq-square*)
also have ... = $C^2 * (sgn^2 * (cos\ (2*pi*t - A))^2 + (sin\ (2*pi*t - A))^2)$
using *eq-gt* **by** (*simp add*: *complex-norm power2-eq-square algebra-simps power2-eq-square*)
also have ... = $C^2 * ((cos\ (2*pi*t - A))^2 + (sin\ (2*pi*t - A))^2)$
using *sgn2* **by** (*simp add*: *algebra-simps power2-eq-square*)
also have ... = $|C|^2$
by (*simp add*: *power2-abs sin-cos-squared-add3*)
finally show *cmod* (*g t* − *c*) = |*C*|
by (*simp add*: *cmod-def*)
qed
then show *?thesis*

```
    by (auto simp: path-image-def sphere-def dist-norm norm-minus-commute)
  qed
  have supset: sphere c |C| ⊆ path-image g
  proof (cases C = 0)
    case True
    have g 0 = c
      using Re-g[of 0] CA[of 0] by (simp add: c-def True complex-eqI)
    moreover have g 0 ∈ path-image g
      by (simp add: path-image-def)
    ultimately show ?thesis by (simp add: True sphere-def dist-self)
  next
    case Cne: False
    show ?thesis
    proof (rule subsetI)
      fix z assume z: z ∈ sphere c |C|
      then have zc-norm: cmod (z − c) = |C|
        by (simp add: sphere-def dist-norm norm-minus-commute)
      — Find the angle for (z − c) scaled to the unit circle
      have unit: cmod (Complex (− Re (z − c) / (sgn ∗ C)) (Im (z − c) / C))
= 1
      proof −
        have (cmod (Complex (− Re (z − c) / (sgn ∗ C)) (Im (z − c) / C)))²
            = (Re (z − c))² / (sgn² ∗ C²) + (Im (z − c))² / C²
          by (metis (no-types, lifting) cmod-power2 complex.sel power2-minus
power-divide
            power-mult-distrib)
        also have ... = |C|² / C²
          by (metis zc-norm add-divide-distrib cmod-power2 mult-cancel-right2
sgn2)
        also have ... = 1
          using Cne by (simp add: power2-abs)
        finally show ?thesis
          using norm-ge-zero by (simp add: abs-square-eq-1)
      qed
      obtain ϑ where ϑ-bounds: 0 ≤ ϑ ϑ < 2∗pi
          and ϑ-eq: Complex (− Re (z − c) / (sgn ∗ C)) (Im (z − c) / C) =
Complex (cos ϑ) (sin ϑ)
        using complex-unimodular-polar[OF unit] by auto
      have ϑ-Re: − Re (z − c) / (sgn ∗ C) = cos ϑ
        and ϑ-Im: Im (z − c) / C = sin ϑ
        using ϑ-eq by (simp-all add: complex.expand)
        — Find t ∈ [0, 1] with 2πt − A ≡ θ (mod 2π)
      define t where t = frac ((ϑ + A) / (2 ∗ pi))
      have t01: t ∈ {0..1}
        using Multiseries-Expansion-Bounds.trivial-bounds-frac t-def by blast
      have ∗: 2∗pi∗t = (ϑ + A) − of-int ⌊(ϑ + A) / (2∗pi)⌋ ∗ (2∗pi)
        using pi-gt-zero by (simp add: t-def frac-def field-simps)
      have cos-eq: cos (2∗pi∗t − A) = cos ϑ
        unfolding ∗ by (simp add: cos-diff mult-of-int-commute)
```

```
        have sin-eq: sin (2*pi*t - A) = sin ϑ
          unfolding * by (simp add: mult-of-int-commute sin-diff)
        have g t = z
        proof (rule complex-eqI)
          have sgn ≠ 0 using sgn2 by fastforce
          have Re (g t) = - sgn * C * cos ϑ + sgn * C * cos A
            using cos-eq by (simp add: Re-g[OF t01] algebra-simps)
          also have ... = Re (z - c) + Re c
            using ‹sgn ≠ 0› sgn2 ϑ-Re Cne by (simp add: c-def field-simps)
          finally show Re (g t) = Re z by simp
        next
          show Im (g t) = Im z
            using CA Cne ϑ-Im sin-eq c-def nonzero-divide-eq-eq t01 by fastforce
        qed
        moreover have g t ∈ path-image g
          using t01 by (auto simp: path-image-def)
        ultimately show z ∈ path-image g by simp
      qed
    qed
    show ?thesis
      using subset supset by (auto intro!: exI[of - c] exI[of - |C|])
  qed
  then show ?thesis
    using d-nonneg w-nonneg key-eq by argo
 qed (use ‹L>0› in auto)
 show measure lebesgue (inside (path-image g)) ≤ L^2 / (4 * pi)
      measure lebesgue (inside (path-image g)) = L^2 / (4 * pi) ⟹
   ∃ c r. path-image g = sphere c r
   using key by (simp-all add: field-simps)
qed
```

Reduction lemmas for the reparametrization steps.

```
lemma isoperimetric-reduce-shift:
 fixes g :: real ⇒ complex
 assumes rectifiable-path g simple-path g
   pathfinish g = pathstart g
   convex (inside (path-image g))
   path-length g = L
   a ∈ path-image g
 obtains h where rectifiable-path h simple-path h
   pathfinish h = pathstart h pathstart h = a
   convex (inside (path-image h))
   path-length h = L
   path-image h = path-image g
proof -
 obtain t where t: t ∈ {0..1} g t = a
   using assms(6) by (auto simp: path-image-def)
 let ?h = shiftpath t g
 show thesis
```

proof
show *rectifiable-path ?h*
using *assms rectifiable-path-shiftpath t(1)* **by** *blast*
show *simple-path ?h*
using *assms(2,3) simple-path-shiftpath t(1)* **by** *auto*
show *pathfinish ?h = pathstart ?h*
using *assms(3) closed-shiftpath t(1)* **by** *blast*
show *pathstart ?h = a*
using *pathstart-shiftpath t* **by** *auto*
show *path-image ?h = path-image g*
using *assms(3) path-image-shiftpath t(1)* **by** *blast*
then show *convex (inside (path-image ?h))*
by (*simp add*: *assms(4)*)
show *path-length ?h = L*
using *assms path-length-shiftpath t(1)* **by** *blast*
qed
qed

lemma *isoperimetric-reduce-rotate-translate*:
fixes *g* :: *real* ⇒ *complex* **and** *a b* :: *complex*
assumes *rectifiable-path g simple-path g*
pathfinish g = pathstart g pathstart g = a
convex (inside (path-image g))
path-length g = L
b ∈ *path-image g dist a b = diameter (path-image g)*
a ≠ *b*
obtains *h a′ b′* **where** *rectifiable-path h simple-path h*
pathfinish h = pathstart h pathstart h = a′
convex (inside (path-image h))
path-length h = L
b′ ∈ *path-image h dist a′ b′ = diameter (path-image h)*
b′ − a′ = of-real (dist a′ b′)
Re a′ = 0
measure lebesgue (inside (path-image h)) = measure lebesgue (inside (path-image g))
⋀*c r. path-image h = sphere c r* ⟹ ∃ *c′ r′. path-image g = sphere c′ r′*
proof −
define *r* **where** *r = cis (− Arg (b − a))*
define *h* **where** *h* = (∗) *r* ∘ (+) (−*a*) ∘ *g*
define *b′* **where** *b′ = r ∗ (b − a)*
have *r-norm*: *norm r = 1* **unfolding** *r-def* **by** *simp*
have *r-ne*: *r* ≠ *0* **using** *r-norm* **by** *auto*
have *lin-r*: *linear* ((∗) *r*) **by** (*intro linearI*) (*auto simp*: *algebra-simps scaleR-conv-of-real*)
have *inj-r*: *inj* ((∗) *r*) **using** *r-ne* **by** (*simp add*: *inj-def*)
have *norm-r*: ⋀*x. norm (r ∗ x) = norm x* **using** *r-norm*
by (*simp add*: *norm-mult*)
have *dist-r*: ⋀*x y. dist (r ∗ x) (r ∗ y) = dist x y*
by (*simp add*: *dist-mult-left r-norm*)
— Translation step: *g1* = (+) (− *a*) ∘ *g*

define *g1* **where** *g1* = (+) (−*a*) ∘ *g*
have *rect-g1*: *rectifiable-path g1*
unfolding *g1-def* **using** *assms*(*1*) *rectifiable-path-translation-eq* **by** *blast*
have *sp-g1*: *simple-path g1*
unfolding *g1-def* **using** *assms*(*2*) *simple-path-translation-eq* **by** *blast*
have *pi-g1*: *path-image g1* = (+) (−*a*) ‘ *path-image g*
unfolding *g1-def* **by** (*simp add*: *path-image-compose image-comp*)
have *ps-g1*: *pathstart g1* = *0*
unfolding *g1-def* **using** *assms*(*4*) **by** (*simp add*: *pathstart-compose*)
have *pf-g1*: *pathfinish g1* = *0*
unfolding *g1-def* **using** *assms*(*3*,*4*) **by** (*simp add*: *pathstart-compose pathfinish-compose*)
have *pl-g1*: *path-length g1* = *L*
unfolding *g1-def* **using** *assms*(*6*) *path-length-translation* **by** *blast*
— Rotation step: *h* = (∗) *r* ∘ *g1*
have *h-eq*: *h* = (∗) *r* ∘ *g1* **unfolding** *h-def g1-def* **by** (*simp add*: *comp-assoc*)
have *pi-h*: *path-image h* = (∗) *r* ‘ *path-image g1*
unfolding *h-eq* **by** (*simp add*: *path-image-compose image-comp*)
— Key: *r* · (*b* − *a*) is a positive real
have *r* ∗ (*b* − *a*) = *cis* (− *Arg* (*b*−*a*)) ∗ (*b*−*a*)
unfolding *r-def* **by** *simp*
also have ... = *of-real* (*cmod* (*b*−*a*))
by (*subst* (*2*) *rcis-cmod-Arg*[*symmetric*, *of b* − *a*]) (*simp add*: *rcis-def cis-mult*)
finally have *rb-real*: *b′* = *of-real* (*cmod* (*b*−*a*)) **unfolding** *b′-def* **by** *simp*
show *?thesis*
proof
show *rectifiable-path h*
unfolding *h-eq* **using** *rect-g1 rectifiable-path-linear-image-eq*[*OF lin-r inj-r*] **by** *simp*
show *simple-path h*
unfolding *h-eq* **using** *sp-g1 simple-path-linear-image-eq*[*OF lin-r inj-r*] **by** *simp*
show *pathfinish h* = *pathstart h*
unfolding *h-eq* **using** *pf-g1 ps-g1* **by** (*simp add*: *pathstart-compose pathfinish-compose*)
show *pathstart h* = *0*
unfolding *h-eq* **using** *ps-g1* **by** (*simp add*: *pathstart-compose*)
show *path-length h* = *L*
unfolding *h-eq* **using** *pl-g1 path-length-linear-image*[*OF lin-r norm-r*] **by** *simp*
show *b′* ∈ *path-image h*
unfolding *pi-h b′-def g1-def* **using** *assms*(*7*)
by (*auto simp*: *path-image-compose image-comp image-iff*)
show *dist 0 b′* = *diameter* (*path-image h*)
proof −
have *diameter* (*path-image h*) = *diameter* ((∗) *r* ‘ *path-image g1*)
unfolding *pi-h* **by** *simp*
also have ... = *diameter* (*path-image g1*)
proof −

```
      have (λ(x,y). dist x y) ‘ ((∗) r ‘ path-image g1 × (∗) r ‘ path-image g1) =
            (λ(x,y). dist x y) ‘ (path-image g1 × path-image g1)
        by (force simp: image-iff dist-r)
      then show ?thesis by (simp add: diameter-def)
    qed
    also have ... = diameter (path-image g)
      unfolding pi-g1 by (metis diameter-translation)
    finally have diam-eq: diameter (path-image h) = diameter (path-image g) .
    have dist 0 b′ = dist a b
      unfolding b′-def by (simp add: dist-norm norm-r norm-minus-commute)
    then show ?thesis using diam-eq assms(8) by simp
  qed
  have inside ((∗) r ‘ path-image g1) = (∗) r ‘ inside (path-image g1)
  proof (rule set-eqI)
    fix x
    define y where y = inverse r ∗ x
    then have xy: x = r ∗ y using r-ne by simp
    have bij-r: bij ((∗) r)
      unfolding bij-def using lin-r inj-r eucl.linear-inj-imp-surj[OF lin-r inj-r]
by blast
    have compl-img: (∗) r ‘ (− path-image g1) = − ((∗) r ‘ path-image g1)
      using bij-image-Compl-eq[OF bij-r] .
    have homeo: homeomorphism (− path-image g1) ((∗) r ‘ (− path-image g1))
((∗) r) ((∗) (inverse r))
    proof (rule homeomorphismI)
      show continuous-on (− path-image g1) ((∗) r)
        continuous-on ((∗) r ‘ (− path-image g1)) ((∗) (inverse r))
        by (intro continuous-intros)+
      show (∗) (inverse r) ‘ ((∗) r ‘ (− path-image g1)) ⊆ − path-image g1
        using r-ne
     by (clarsimp simp: image-iff) (metis divide-inverse-commute nonzero-mult-div-cancel-left)
    qed (use r-ne in auto)
    have cc: connected-component-set (− ((∗) r ‘ path-image g1)) x =
                (∗) r ‘ connected-component-set (− path-image g1) y
    proof (cases y ∈ path-image g1)
      case True
      then show ?thesis
        using xy connected-component-eq-empty by blast
    next
      case False
      then show ?thesis
        using compl-img xy connected-component-set-homeomorphism[OF homeo]
by simp
    qed
    have bounded ((∗) r ‘ connected-component-set (− path-image g1) y) =
          bounded (connected-component-set (− path-image g1) y)
      by (simp add: bounded-iff norm-r image-iff)
  then show (x ∈ inside ((∗) r ‘ path-image g1)) = (x ∈ (∗) r ‘ inside (path-image
g1))
```

```
      using xy cc by (simp add: inj-image-mem-iff [OF inj-r] inside-def)
    qed
    then have inside (path-image h) = (∗) r ‘ inside (path-image g1)
      unfolding pi-h .
    also have inside (path-image g1) = (+) (−a) ‘ inside (path-image g)
      unfolding pi-g1 using inside-translation[of −a path-image g] by simp
      finally have inside-h: inside (path-image h) = (∗) r ‘ (+) (−a) ‘ inside
(path-image g)
      .
    show convex (inside (path-image h))
      using inside-h assms(5) by (metis convex-linear-image convex-translation-eq
lin-r)
     show measure lebesgue (inside (path-image h)) = measure lebesgue (inside
(path-image g))
    proof −
      have meas-g: inside (path-image g) ∈ lmeasurable
        by (simp add: Jordan-inside-outside assms lmeasurable-open)
      have measure lebesgue ((∗) r ‘ (+) (−a) ‘ inside (path-image g)) =
            measure lebesgue ((+) (−a) ‘ inside (path-image g))
      proof −
        have meas-t: (+) (−a) ‘ inside (path-image g) ∈ lmeasurable
          using meas-g measurable-translation by blast
        have |eucl.det ((∗) r)| = 1
          unfolding det-complex r-def by simp
        then show ?thesis
          using measure-linear-image[OF lin-r meas-t] by simp
      qed
      also have ... = measure lebesgue (inside (path-image g))
        using measure-translation[of −a inside (path-image g)] by simp
      finally show ?thesis using inside-h by simp
    qed
    show ∃ c′ r′. path-image g = sphere c′ r′
      if sph: path-image h = sphere c0 r0 for c0 r0
    proof −
      have (+) (−a) ‘ path-image g = (∗) (inverse r) ‘ sphere c0 r0
      proof −
        have ∗: ⋀z. inverse r ∗ (r ∗ z) = z
          using r-ne by (metis left-inverse mult.assoc mult-1)
          have (∗) (inverse r) ‘ ((∗) r ‘ (+) (−a) ‘ path-image g) = (+) (−a) ‘
path-image g
          by (auto simp: image-iff ∗)
        then show ?thesis
          using pi-g1 pi-h sph by argo
      qed
      then have path-image g = (+) a ‘ (∗) (inverse r) ‘ sphere c0 r0
        by (metis translation-galois)
      moreover have (∗) (inverse r) ‘ sphere c0 r0 = sphere (inverse r ∗ c0) r0
        by (auto simp: nonzero-norm-inverse r-ne r-norm sphere-cscale)
      moreover have (+) a ‘ sphere (inverse r ∗ c0) r0 = sphere (a + inverse r
```

$*$ *c0*) *r0*
using *sphere-translation*[*of a inverse r* $*$ *c0 r0*] **by** *simp*
ultimately show $\exists$ *c' r'*. *path-image g* = *sphere c' r'* **by** *auto*
qed
qed (*auto simp*: *rb-real dist-norm*)
qed

lemma *isoperimetric-reduce-arc-length*:
fixes *g* :: *real* $\Rightarrow$ *complex*
assumes *rectifiable-path g simple-path g*
pathfinish g = *pathstart g*
convex (*inside* (*path-image g*))
path-length g = *L* 0 < *L*
obtains *h* **where** *rectifiable-path h simple-path h*
pathfinish h = *pathstart h pathstart h* = *pathstart g*
convex (*inside* (*path-image h*))
path-length h = *L*
path-image h = *path-image g*
$\bigwedge$*t*. *t* $\in$ {*0..1*} $\Longrightarrow$ *path-length* (*subpath 0 t h*) = *L* $*$ *t*
$\bigwedge$*x y*. *x* $\in$ {*0..1*} $\Longrightarrow$ *y* $\in$ {*0..1*} $\Longrightarrow$ *dist* (*h x*) (*h y*) $\leq$ *L* $*$ *dist x y*
using *arc-length-reparametrization assms* **by** *metis*

lemma *isoperimetric-reduce-zero-mean*:
fixes *g* :: *real* $\Rightarrow$ *complex* **and** *b* :: *complex*
assumes *rectifiable-path g simple-path g*
pathfinish g = *pathstart g*
convex (*inside* (*path-image g*))
path-length g = *L*
b $\in$ *path-image g*
dist (*pathstart g*) *b* = *diameter* (*path-image g*)
b $-$ *pathstart g* = *of-real* (*dist* (*pathstart g*) *b*)
Re (*pathstart g*) = *0*
and *subp*: $\bigwedge$*t*. *t* $\in$ {*0..1*} $\Longrightarrow$ *path-length* (*subpath 0 t g*) = *L* $*$ *t*
and *dist*: $\bigwedge$*x y*. *x* $\in$ {*0..1*} $\Longrightarrow$ *y* $\in$ {*0..1*} $\Longrightarrow$ *dist* (*g x*) (*g y*) $\leq$ *L* $*$ *dist x y*
obtains *h a' b'* **where** *rectifiable-path h simple-path h*
pathfinish h = *pathstart h*
convex (*inside* (*path-image h*))
path-length h = *L*
a' $\in$ *path-image h b'* $\in$ *path-image h*
dist a' b' = *diameter* (*path-image h*)
b' $-$ *a'* = *of-real* (*dist a' b'*)
pathstart h = *a' pathfinish h* = *a'*
Re a' = *0*
$\bigwedge$*t*. *t* $\in$ {*0..1*} $\Longrightarrow$ *path-length* (*subpath 0 t h*) = *L* $*$ *t*
$\bigwedge$*x y*. *x* $\in$ {*0..1*} $\Longrightarrow$ *y* $\in$ {*0..1*} $\Longrightarrow$ *dist* (*h x*) (*h y*) $\leq$ *L* $*$ *dist x y*
(*Im* $\circ$ *h has-integral 0*) {*0..1*}
measure lebesgue (*inside* (*path-image h*)) = *measure lebesgue* (*inside* (*path-image g*))

$\bigwedge c\ r.$ *path-image* $h =$ *sphere* $c\ r \Longrightarrow \exists\, c'\ r'.$ *path-image* $g =$ *sphere* $c'\ r'$
proof −
define c **where** $c =$ *integral* $\{0..1\}$ $(Im \circ g)$
define d **where** $d = -(\mathrm{i} * (of\text{-}real\ c :: complex))$
define h **where** $h = (+)\ d \circ g$
define a' **where** $a' =$ *pathstart* $g + d$
define b' **where** $b' = b + d$
have *h-eq*: $\bigwedge t.\ h\ t = g\ t + d$ **unfolding** *h-def comp-def* **by** *simp*
have *pi-h*: *path-image* $h = (+)\ d$ ‘ *path-image* g
unfolding *h-def image-comp* [*symmetric*] *path-image-compose* **by** *simp*
show *?thesis*
proof
show *rectifiable-path h*
unfolding *h-def* **using** *assms*(*1*) *rectifiable-path-translation-eq*[*of d g*] **by** *simp*
show *simple-path h*
unfolding *h-def* **using** *assms*(*2*) *simple-path-translation-eq*[*of d g*] **by** *simp*
show *pathfinish h* = *pathstart h*
unfolding *h-def* **using** *assms*(*3*) **by** (*simp add*: *pathstart-compose pathfinish-compose*)
show *path-length h* = *L*
unfolding *h-def* **using** *assms*(*5*) *path-length-translation*[*of d g*] **by** *simp*
show *pathstart h* = *a'* **unfolding** *h-def a'-def* **by** (*simp add*: *pathstart-compose*)
show *pathfinish h* = *a'* **unfolding** *h-def a'-def*
using *assms*(*3*) **by** (*simp add*: *pathstart-compose pathfinish-compose*)
show *a'* ∈ *path-image h*
unfolding *a'-def* **using** *pi-h path-image-def pathstart-def* **by** *fastforce*
show *b'* ∈ *path-image h*
unfolding *b'-def* **using** *pi-h assms*(*6*) **by** *auto*
show *b'* − *a'* = *of-real* (*dist a' b'*)
unfolding *a'-def b'-def* **using** *assms*(*8*) **by** (*simp add*: *dist-norm*)
show *dist a' b'* = *diameter* (*path-image h*)
by (*simp add*: *a'-def assms*(*7*) *b'-def diameter-translation pi-h*)
show *Re a'* = *0* **unfolding** *a'-def d-def* **using** *assms*(*9*) **by** *simp*
show *convex* (*inside* (*path-image h*))
by (*simp add*: *assms*(*4*) *inside-translation pi-h*)
show $\bigwedge t.\ t \in \{0..1\} \Longrightarrow$ *path-length* (*subpath 0 t h*) = *L* ∗ *t*
by (*simp add*: *assms*(*10*) *h-def path-length-translation subpath-image*)
show $\bigwedge x\ y.\ x \in \{0..1\} \Longrightarrow y \in \{0..1\} \Longrightarrow$ *dist* (*h x*) (*h y*) ≤ *L* ∗ *dist x y*
by (*simp add*: *dist h-eq*)
show (*Im* ∘ *h has-integral 0*) $\{0..1\}$
proof −
have *cont-g*: *continuous-on* $\{0..1\}$ *g*
using *rectifiable-path-imp-path*[*OF assms*(*1*)] **unfolding** *path-def* **.**
have *int-Im-g*: (λ*t*. *Im* (*g t*)) *integrable-on* $\{0..1\}$
using *integrable-continuous-real*[*OF continuous-on-Im*[*OF cont-g*]] **.**
have *Im-h*: $\bigwedge t.$ *Im* (*h t*) = *Im* (*g t*) − *c*
unfolding *h-def comp-def d-def* **by** *simp*
have *eq*: $\bigwedge t.$ (*Im* ∘ *h*) *t* = (λ*t*. *Im* (*g t*) − *c*) *t*
using *Im-h* **unfolding** *comp-def* **by** *simp*

have *int-sub*: $(\lambda t.\ Im\ (g\ t) - c)$ *integrable-on* $\{0..1\}$
by (*rule integrable-diff*[*OF int-Im-g integrable-const-ivl*])
have *int-h*: $(Im \circ h)$ *integrable-on* $\{0..1\}$
using *integrable-spike-finite*[*OF finite.emptyI - int-sub*] *eq* **by** *simp*
have *integral* $\{0..1\}$ $(Im \circ h) =$ *integral* $\{0..1\}$ $(\lambda t.\ Im\ (g\ t) - c)$
using *eq* **by** *presburger*
also have $\ldots =$ *integral* $\{0..1\}$ $(\lambda t.\ Im\ (g\ t))$ $-$ *integral* $\{0..1\}$ $(\lambda$-::*real*. *c*::*real*)
using *Henstock-Kurzweil-Integration.integral-diff int-Im-g* **by** *blast*
also have $\ldots = 0$ **unfolding** *c-def comp-def* **by** *simp*
finally show *?thesis* **using** *int-h has-integral-iff* **by** *blast*
qed
show *measure lebesgue* (*inside* (*path-image h*)) = *measure lebesgue* (*inside* (*path-image g*))
by (*metis inside-translation measure-translation pi-h*)
show $\exists\, c'\ r'.$ *path-image g* = *sphere c' r'* **if** *path-image h* = *sphere c0 r* **for** *c0 r*
by (*metis pi-h sphere-translation that translation-galois*)
qed
qed

theorem *isoperimetric-theorem-convex*:
fixes *g* :: *real* ⇒ *complex*
assumes *rectifiable-path g simple-path g*
pathfinish g = *pathstart g*
convex (*inside* (*path-image g*))
path-length g = *L*
shows *measure lebesgue* (*inside* (*path-image g*)) $\leq L^2$ / (4 ∗ *pi*)
and *measure lebesgue* (*inside* (*path-image g*)) = L^2 / (4 ∗ *pi*) ⟹
$\exists\, a\ r.$ *path-image g* = *sphere a r*
proof −
have *Lpos*: 0 < *L*
using *simple-path-length-pos-lt*[*OF assms*(1,2)] *assms*(5) **by** *simp*

Step 1: obtain diameter endpoints

have *compact-pi*: *compact* (*path-image g*)
using *compact-simple-path-image*[*OF assms*(2)] **.**
have *nonempty-pi*: *path-image g* ≠ {}
using *path-image-nonempty* **.**
obtain *a b* **where** *ab*: *a* ∈ *path-image g b* ∈ *path-image g*
dist a b = *diameter* (*path-image g*)
using *diameter-compact-attained*[*OF compact-pi nonempty-pi*] **by** *auto*

Step 2: shift start to diameter endpoint a

obtain *g1* **where** *g1*: *rectifiable-path g1 simple-path g1*
pathfinish g1 = *pathstart g1 pathstart g1* = *a*
convex (*inside* (*path-image g1*))
path-length g1 = *L path-image g1* = *path-image g*
using *isoperimetric-reduce-shift*[*OF assms*(1,2,3,4,5) *ab*(1)] **by** *metis*

have *ab1*: $a \in$ *path-image g1* $b \in$ *path-image g1*
dist a b = *diameter* (*path-image g1*)
using *ab g1*(*7*) **by** *auto*
have *a-ne-b*: $a \neq b$
proof
assume $a = b$
then have *diameter* (*path-image g*) = *0* **using** *ab*(*3*) **by** *simp*
then have *path-image g* = $\{a\}$
by (*metis ab*(*1*) *compact-eq-bounded-closed compact-pi diameter-eq-0 equals0D insertE*)
then show *False* **using** *simple-path-image-uncountable*[*OF assms*(*2*)]
by (*simp add*: *countable-finite*)
qed

Step 3: rotate and translate to normalize diameter direction

obtain *g2 a2 b2* **where** *g2*: *rectifiable-path g2 simple-path g2*
pathfinish g2 = *pathstart g2 pathstart g2* = *a2*
convex (*inside* (*path-image g2*))
path-length g2 = *L*
b2 $\in$ *path-image g2 dist a2 b2* = *diameter* (*path-image g2*)
b2 − *a2* = *of-real* (*dist a2 b2*)
Re a2 = *0*
measure lebesgue (*inside* (*path-image g2*)) = *measure lebesgue* (*inside* (*path-image g*))
and *sphere-back2*: $\bigwedge c\ r.$ *path-image g2* = *sphere c r* $\Longrightarrow$
$\exists\, c'\ r'.$ *path-image g* = *sphere c' r'*
using *isoperimetric-reduce-rotate-translate g1 ab1 a-ne-b* **by** (*metis* (*no-types*, *lifting*))

Step 4: arc-length reparametrization

obtain *g3* **where** *g3*: *rectifiable-path g3 simple-path g3*
pathfinish g3 = *pathstart g3 pathstart g3* = *pathstart g2*
convex (*inside* (*path-image g3*))
path-length g3 = *L*
path-image g3 = *path-image g2*
$\bigwedge t.\ t \in \{0..1\} \Longrightarrow$ *path-length* (*subpath 0 t g3*) = *L* ∗ *t*
$\bigwedge x\ y.\ x \in \{0..1\} \Longrightarrow y \in \{0..1\} \Longrightarrow$ *dist* (*g3 x*) (*g3 y*) ≤ *L* ∗ *dist x y*
using *isoperimetric-reduce-arc-length*[*OF g2*(*1*,*2*,*3*,*5*,*6*) *Lpos*] **by** *metis*
have *g3-facts*: *b2* $\in$ *path-image g3 dist* (*pathstart g3*) *b2* = *diameter* (*path-image g3*)
b2 − *pathstart g3* = *of-real* (*dist* (*pathstart g3*) *b2*) *Re* (*pathstart g3*) = *0*
using *g2 g3* **by** *auto*

Step 5: vertical translation for zero-mean imaginary part

obtain *h a' b'* **where** *h*: *rectifiable-path h simple-path h*
pathfinish h = *pathstart h*
convex (*inside* (*path-image h*))
path-length h = *L*
a' $\in$ *path-image h b'* $\in$ *path-image h*

dist a′ b′ = diameter (path-image h)
b′ − a′ = of-real (dist a′ b′)
pathstart h = a′ pathfinish h = a′
Re a′ = 0
⋀t. t ∈ {0..1} ⟹ path-length (subpath 0 t h) = L ∗ t
⋀x y. x ∈ {0..1} ⟹ y ∈ {0..1} ⟹ dist (h x) (h y) ≤ L ∗ dist x y
(Im ∘ h has-integral 0) {0..1}
and *meas-eq5*: *measure lebesgue (inside (path-image h)) =*
measure lebesgue (inside (path-image g3))
and *sphere-back5*: *⋀c r. path-image h = sphere c r ⟹*
∃ c′ r′. path-image g3 = sphere c′ r′
using *isoperimetric-reduce-zero-mean g3 g3-facts* **by** *blast*
have *meas-eq*: *measure lebesgue (inside (path-image g)) =*
measure lebesgue (inside (path-image h))
using *meas-eq5 g3(5,7) g2(11)* **by** *simp*
have *sphere-back*: *⋀c r. path-image h = sphere c r ⟹ ∃ c′ r′. path-image g = sphere c′ r′*
by (*metis g3(7) sphere-back2 sphere-back5*)

Step 6: apply the kernel lemma

have *kernel-hyps*: *0 < L convex (inside (path-image h))*
a′ ∈ path-image h b′ ∈ path-image h
dist a′ b′ = diameter (path-image h)
b′ − a′ = of-real (dist a′ b′)
pathstart h = a′ pathfinish h = a′
rectifiable-path h simple-path h
path-length h = L
Re a′ = 0
(Im ∘ h has-integral 0) {0..1}
using *Lpos h* **by** *auto*
have *ineq-h*: *measure lebesgue (inside (path-image h)) ≤ L² / (4 ∗ pi)*
using *h isoperimetric-kernel(1) kernel-hyps* **by** *blast*
show *measure lebesgue (inside (path-image g)) ≤ L² / (4 ∗ pi)*
using *ineq-h meas-eq* **by** *simp*
show *∃ a r. path-image g = sphere a r*
if *measure lebesgue (inside (path-image g)) = L² / (4 ∗ pi)*
using *that h isoperimetric-kernel simple-path-length-pos-lt sphere-back*
by (*metis meas-eq*)
qed

4.1 Convexification

A connected subset of a segment that contains both endpoints is the whole segment. (HOL Light: CONNECTED_SUBSET_SEGMENT.)

lemma *connected-subset-segment*:
fixes *a b* :: *'a::euclidean-space*
assumes *connected S S ⊆ closed-segment a b a ∈ S b ∈ S*
shows *S = closed-segment a b*
proof (*cases a = b*)

```
  case True
  then show ?thesis using assms by auto
next
  case False
  define f where f = (λx::'a. (b − a) · x)
  define K where K = (b − a) · (b − a)
  have contf: continuous-on S f by (auto simp: f-def continuous-intros)
 have conn-fS: connected (f ‘ S) using assms(1) contf connected-continuous-image
by blast
  have nz: K > 0 using False by (simp add: K-def inner-gt-zero-iff)
  have fseg: ⋀u. f ((1−u) *R a + u *R b) = (b − a) · a + u * K
   by (simp add: f-def K-def inner-diff-right algebra-simps inner-diff-left inner-commute)
  show ?thesis
  proof (intro subset-antisym assms)
    show closed-segment a b ⊆ S
    proof
      fix x assume x ∈ closed-segment a b
      then obtain u where u: x = (1 − u) *R a + u *R b 0 ≤ u u ≤ 1
        by (auto simp: in-segment)
      have fx: f x = (b − a) · a + u * K using u(1) fseg by simp
      have mem: f x ∈ closed-segment (f a) (f b)
        unfolding closed-segment-eq-real-ivl using nz u(2,3) f-def fseg[of 1] fx
        by (auto simp: mult-left-le-one-le)
      have closed-segment (f a) (f b) ⊆ f ‘ S
        using conn-fS assms(3,4) connected-contains-Icc
        by (metis closed-segment-eq-real-ivl image-eqI)
      then obtain s where s: s ∈ S f s = f x s ∈ closed-segment a b
        using mem assms by (metis image-iff subsetD)
      then obtain v where v: s = (1 − v) *R a + v *R b 0 ≤ v v ≤ 1
        by (auto simp: in-segment)
      have (b − a) · a + v * K = (b − a) · a + u * K
        using s(2) v(1) fseg fx by simp
      then show x ∈ S using nz s(1) v(1) u(1) by simp
    qed
  qed
qed
```

A connected subset of two arcs joined only at their common endpoints, containing both endpoints, must contain one of the two arcs in full. (HOL Light: CONNECTED_SUBSET_ARC_PAIR.)

```
lemma connected-subset-arc-pair:
  fixes g h :: real ⇒ 'a::euclidean-space
  assumes arc g arc h
    pathstart g = pathstart h pathfinish g = pathfinish h
    and pai-eq: path-image g ∩ path-image h = {pathstart g, pathfinish g}
    connected S and Ssub: S ⊆ path-image g ∪ path-image h
    and pathstart g ∈ S pathfinish g ∈ S
  shows path-image g ⊆ S ∨ path-image h ⊆ S
proof (rule ccontr)
```

assume ¬ (*path-image g* ⊆ *S* ∨ *path-image h* ⊆ *S*)
then have *ng*: ¬ *path-image g* ⊆ *S* **and** *nh*: ¬ *path-image h* ⊆ *S* **by** *auto*
from *ng* **obtain** *pg* **where** *pg*: *pg* ∈ *path-image g pg* ∉ *S* **by** *blast*
then obtain *p* **where** *p*: *p* ∈ {*0..1*} *g p* = *pg* **by** (*auto simp*: *path-image-def*)
from *nh* **obtain** *ph* **where** *ph*: *ph* ∈ *path-image h ph* ∉ *S* **by** *blast*
then obtain *q* **where** *q*: *q* ∈ {*0..1*} *h q* = *ph* **by** (*auto simp*: *path-image-def*)
have *gp*: *g p* ∉ *S* **using** *p*(*2*) *pg*(*2*) **by** *simp*
have *hq*: *h q* ∉ *S* **using** *q*(*2*) *ph*(*2*) **by** *simp*
have *pathg*: *path g* **and** *pathh*: *path h* **using** *assms*(*1*,*2*) *arc-imp-path* **by** *auto*
have *img-g0p*: *path-image* (*subpath 0 p g*) = *g* ‘ {*0..p*} **using** *p*(*1*) **by** (*simp add*: *path-image-subpath*)
have *img-gp1*: *path-image* (*subpath p 1 g*) = *g* ‘ {*p..1*} **using** *p*(*1*) **by** (*simp add*: *path-image-subpath*)
have *img-h0q*: *path-image* (*subpath 0 q h*) = *h* ‘ {*0..q*} **using** *q*(*1*) **by** (*simp add*: *path-image-subpath*)
have *img-hq1*: *path-image* (*subpath q 1 h*) = *h* ‘ {*q..1*} **using** *q*(*1*) **by** (*simp add*: *path-image-subpath*)
have *comb-g*: *g* ‘ {*0..p*} ∪ *g* ‘ {*p..1*} = *path-image g*
using *p*(*1*) **by** (*simp add*: *path-image-def image-Un* [*symmetric*]) (*metis ivl-disj-un-two-touch*(*4*))
have *comb-h*: *h* ‘ {*0..q*} ∪ *h* ‘ {*q..1*} = *path-image h*
using *q*(*1*) **by** (*simp add*: *path-image-def image-Un* [*symmetric*]) (*metis ivl-disj-un-two-touch*(*4*))
define *E1* **where** *E1* = *S* − (*path-image* (*subpath p 1 g*) ∪ *path-image* (*subpath q 1 h*))
define *E2* **where** *E2* = *S* − (*path-image* (*subpath 0 p g*) ∪ *path-image* (*subpath 0 q h*))
have *openE1*: *openin* (*top-of-set S*) *E1*
proof −
have *E1* = *S* ∩ (− (*path-image* (*subpath p 1 g*) ∪ *path-image* (*subpath q 1 h*)))
unfolding *E1-def* **by** *blast*
moreover have *closed* (*path-image* (*subpath p 1 g*) ∪ *path-image* (*subpath q 1 h*))
using *p q pathg pathh* **by** (*force intro*: *closed-path-image*)
ultimately show *?thesis* **by** *blast*
qed
have *openE2*: *openin* (*top-of-set S*) *E2*
proof −
have *E2* = *S* ∩ (− (*path-image* (*subpath 0 p g*) ∪ *path-image* (*subpath 0 q h*)))
unfolding *E2-def* **by** *blast*
moreover have *closed* (*path-image* (*subpath 0 p g*) ∪ *path-image* (*subpath 0 q h*))
using *p q pathg pathh* **by** (*force intro*: *closed-path-image*)
ultimately show *?thesis* **by** *blast*
qed
have *injg*: *inj-on g* {*0..1*} **and** *injh*: *inj-on h* {*0..1*} **using** *assms* **by** (*simp-all add*: *arc-def*)
have *p* > *0*
using *assms pg p* **by** (*force simp*: *pathstart-def*)
have *q* > *0*
using *assms ph q* **by** (*force simp*: *pathstart-def*)

```
  have p < 1
    using assms pg p by (force simp: pathfinish-def)
  have q < 1
    using assms ph q by (force simp: pathfinish-def)
  have ne-E1: pathstart g ∈ E1
  proof −
    have g 0 ∉ g ‘ {p..1}
      using injg p(1) ‹p > 0› by (force simp: inj-on-def)
    moreover have g 0 ∉ h ‘ {q..1}
     using injh q(1) ‹q > 0› assms(3) by (fastforce simp: pathstart-def inj-on-def)
    ultimately show ?thesis
     using ‹pathstart g ∈ S› img-gp1 img-hq1 by (simp add: E1-def pathstart-def)
  qed
  have ne-E2: pathfinish g ∈ E2
  proof −
    have g 1 ∉ g ‘ {0..p}
      using injg ‹p < 1› p(1) by (force simp: inj-on-def)
    moreover have g 1 ∉ h ‘ {0..q}
      by (smt (verit) arcD assms(2,4) atLeastAtMost-iff imageE pathfinish-def ‹q
< 1›)
    ultimately show ?thesis
     using ‹pathfinish g ∈ S› img-g0p img-h0q by (simp add: E2-def pathfinish-def)
  qed
  have ps-g: pathstart g = g 0 by (simp add: pathstart-def)
  have pf-g: pathfinish g = g 1 by (simp add: pathfinish-def)
  have cross-gh: g s = g 0 ∨ g s = g 1
    if s ∈ {0..1} t ∈ {0..1} g s = h t for s t
    using that pf-g ps-g assms(5) by (force simp: path-image-def)
  have gh11: g 1 = h 1 using assms(4) by (simp add: pathfinish-def)
  have gh00: g 0 = h 0 using assms(3) by (simp add: pathstart-def)
  have cover-False: False
    if xS: x ∈ S and inR1: x ∈ g ‘ {p..1} ∪ h ‘ {q..1} and inR2: x ∈ g ‘ {0..p}
∪ h ‘ {0..q} for x
  proof −
    from inR1 consider (g1) x ∈ g ‘ {p..1} | (h1) x ∈ h ‘ {q..1} by auto
    then show False
    proof cases
      case g1
      then obtain s where s: s ∈ {p..1} x = g s by auto
      from inR2 consider (a) x ∈ g ‘ {0..p} | (b) x ∈ h ‘ {0..q} by auto
      then show False
      proof cases
        case a
        then obtain t where t: t ∈ {0..p} x = g t by auto
        have s = t using injg s t p s(2) t(2) by (force simp: inj-on-def)
        then have s = p using s(1) t(1) by simp
        then show False using gp xS s(2) by simp
      next
        case b
```

```
      then obtain t where t: t ∈ {0..q} x = h t by auto
      have eq: g s = h t using s(2) t(2) by simp
     have g s = g 0 ∨ g s = g 1 using cross-gh s(1) p q t(1) eq by (auto intro:
order.trans)
      then show False
      proof
        assume g s = g 0
        then have s = 0 using injg s(1) p by (force simp: inj-on-def)
        then show False using s(1) ‹p > 0› by simp
      next
        assume g s = g 1
        then have t = 1 using eq gh11 injh t(1) q by (force simp: inj-on-def)
        then show False using t(1) ‹q < 1› by simp
      qed
    qed
  next
    case h1
    then obtain s where s: s ∈ {q..1} x = h s by auto
    from inR2 consider (a) x ∈ g ‘ {0..p} | (b) x ∈ h ‘ {0..q} by auto
    then show False
    proof cases
      case a
      then obtain t where t: t ∈ {0..p} x = g t by auto
      have eq: g t = h s using s(2) t(2) by simp
     have g t = g 0 ∨ g t = g 1 using cross-gh t(1) p s(1) q eq by (auto intro:
order.trans)
      then show False
      proof
        assume g t = g 1
        then have t = 1 using injg t(1) p by (force simp: inj-on-def)
        then show False using t(1) ‹p < 1› by simp
      next
        assume g t = g 0
        then have s = 0 using injh eq gh00 s(1) q by (force simp: inj-on-def)
        then show False using s(1) ‹q > 0› by simp
      qed
    next
      case b then show False
        using arcD assms(2) ph(2) q(2) s xS by fastforce
    qed
  qed
  have cover: S ⊆ E1 ∪ E2
    using arcD assms(1) gp cover-False
    unfolding E1-def E2-def img-gp1 img-hq1 img-g0p img-h0q by blast
  have (g ‘ {p..1} ∪ h ‘ {q..1}) ∪ (g ‘ {0..p} ∪ h ‘ {0..q}) = path-image g ∪
path-image h
    using comb-g comb-h by blast
  with assms(7)  have disjoint: E1 ∩ E2 = {}
```

```
    unfolding E1-def E2-def img-gp1 img-hq1 img-g0p img-h0q by blast
  have openin (top-of-set S) E1 ∧ openin (top-of-set S) E2 ∧ S ⊆ E1 ∪ E2 ∧ E1
∩ E2 = {} ∧ E1 ≠ {} ∧ E2 ≠ {}
    using openE1 openE2 cover disjoint ne-E1 ne-E2 by auto
  with ‹connected S› show False
    using connected-openin by blast
qed
```

If two frontier points of a bounded convex 2-dimensional set are joined by a frontier arc whose interior (the arc minus the two points) stays connected and whose convex hull is the whole set, then the straight segment between the two points also lies on the frontier. The connectedness of the arc-minus-endpoints is what rules out the spurious "chord through the interior" case.

```
lemma seg-frontier-aux:
  fixes S :: complex set
  assumes cvx: convex S and cpt: compact S and intne: interior S ≠ {}
    and ga-fr: ga ∈ frontier S and gb-fr: gb ∈ frontier S and ne: ga ≠ gb
    and D1fr: path-image D1 ⊆ frontier S
    and D1con: connected (path-image D1 − {ga, gb})
    and gaD1: ga ∈ path-image D1 and gbD1: gb ∈ path-image D1
    and hullD1: convex hull (path-image D1) = S
  shows closed-segment ga gb ⊆ frontier S
proof −
  have relfr: rel-frontier S = frontier S using intne by (rule rel-frontier-nonempty-interior)
  have ga-cl: ga ∈ closure S and gb-cl: gb ∈ closure S
    using ga-fr gb-fr by (auto simp: frontier-def)
  have dich: open-segment ga gb ⊆ frontier S ∨ open-segment ga gb ⊆ interior S
    using convex-open-segment-cases-alt[OF cvx ga-cl gb-cl] .
  define A where A = i ∗ (gb − ga)
  have A-nz: A ≠ 0 using ne by (simp add: A-def)
  define e where e = inner A ga
  have eb: inner A gb = e
    by (auto simp: A-def inner-complex-def e-def algebra-simps)
  have opfr: open-segment ga gb ⊆ frontier S
  proof (rule ccontr)
    assume ¬ open-segment ga gb ⊆ frontier S
    then have segint: open-segment ga gb ⊆ interior S using dich by blast
    have mid-int: midpoint ga gb ∈ interior S
      using segint ne by (simp add: midpoint-in-open-segment subsetD)
    have mid-e: inner A (midpoint ga gb) = e
      using e-def eb by (simp add: midpoint-def inner-add-right inner-diff-right
field-simps)
    have not-le: False if path-image D1 ⊆ {x. inner A x ≤ e}
    proof −
      have S ⊆ {x. inner A x ≤ e}
        by (metis convex-halfspace-le hullD1 hull-minimal that)
      then have inner A (midpoint ga gb) < e using mid-int
        by (metis (mono-tags, lifting) A-nz interior-halfspace-le interior-mono
            mem-Collect-eq subsetD)
```

```
    then show False using mid-e by simp
  qed
  have not-ge: False if path-image D1 ⊆ {x. e ≤ inner A x}
  proof −
    have S ⊆ {x. e ≤ inner A x}
      by (metis convex-halfspace-ge hullD1 hull-minimal that)
    then have e < inner A (midpoint ga gb) using mid-int
      by (metis A-nz interior-halfspace-ge interior-mono mem-Collect-eq subsetD)
    then show False using mid-e by simp
  qed
   from not-le obtain x1 where x1: x1 ∈ path-image D1 inner A x1 > e by
force
   from not-ge obtain x2 where x2: x2 ∈ path-image D1 inner A x2 < e by
force
  have x1ne: x1 ∉ {ga, gb} using x1(2) e-def eb by auto
  have x2ne: x2 ∉ {ga, gb} using x2(2) e-def eb by auto
  have x1D: x1 ∈ path-image D1 − {ga,gb} using x1(1) x1ne by simp
  have x2D: x2 ∈ path-image D1 − {ga,gb} using x2(1) x2ne by simp
  obtain w where w: w ∈ path-image D1 − {ga,gb} inner A w = e
    using connected-ivt-hyperplane[OF D1con x2D x1D, where a=A and b=e]
x1(2) x2(2) by force
  have w-fr: w ∈ rel-frontier S using w(1) D1fr relfr by auto
  have ga-rf: ga ∈ rel-frontier S using ga-fr relfr by simp
  have gb-rf: gb ∈ rel-frontier S using gb-fr relfr by simp
  have wne: w ≠ ga w ≠ gb using w(1) by auto
  have triple: S ⊆ {x. inner A x ≤ e} ∨ S ⊆ {x. e ≤ inner A x}
     using assms(1,6) convex-triple-rel-frontier e-def eb ga-rf gb-rf w(2) w-fr
wne(1,2) by blast
  have D1S: path-image D1 ⊆ S
    by (metis hullD1 hull-subset)
  from triple show False
    using D1S local.not-le not-ge by blast
 qed
 have {ga, gb} ⊆ frontier S using ga-fr gb-fr by simp
 then show ?thesis using opfr by (simp add: closed-segment-eq-open)
qed
```

The step lemma: replacing an arc that deviates from the convex hull frontier with a straight segment shortens the path while preserving the convex hull.

```
lemma step-lemma:
 fixes g :: real ⇒ complex
 assumes simple-path g pathfinish g = pathstart g
  and cont: ⋀x y. x ∈ {0..1} ⟹ y ∈ {0..1} ⟹ dist (g x) (g y) ≤ L * dist x y
  and a < b
  and ab01: a ∈ {0..1} b ∈ {0..1}
  and ga: g a ∈ frontier (convex hull (path-image g))
  and gb: g b ∈ frontier (convex hull (path-image g))
  and disj: g ` {a<..<b} ∩ frontier (convex hull (path-image g)) = {}
```

obtains h **where** *simple-path h*
and *pathstart h = pathstart g* **and** *pathfinish h = pathstart g*
and $\bigwedge x\ y.\ x \in \{0..1\} \Longrightarrow y \in \{0..1\} \Longrightarrow dist\ (h\ x)\ (h\ y) \leq L * dist\ x\ y$
and *path-length h < path-length g*
and *convex hull (path-image h) = convex hull (path-image g)*
and $\bigwedge x.\ x \notin \{a<..<b\} \Longrightarrow h\ x = g\ x$
and *h ‘ {a..b} ⊆ frontier (convex hull (path-image g))*
proof (*cases box a b = {}*)
case *True*
with ‹*a<b*› **show** *?thesis* **by** *auto*
next
case *False*
have *sub*: *g ‘ {a<..<b} ⊆ convex hull (path-image g)*
using *ab01(1,2) hull-subset path-defs(4)* **by** *fastforce*
moreover have *closed*: *closed (convex hull (path-image g))*
by (*simp add*: ‹*simple-path g*› *compact-convex-hull compact-imp-closed compact-simple-path-image*)
ultimately
have *interior-subset*: *g ‘ {a<..<b} ⊆ interior (convex hull (path-image g))*
using *disj frontier-def* **by** *fastforce*
have *interior-ne*: *interior (convex hull (path-image g)) ≠ {}*
using *interior-subset* ‹*a<b*› **by** *fastforce*
show *?thesis*
proof (*cases g a = g b*)
case *True*
have *ab-eq*: *a = 0 b = 1*
using *True* ‹*simple-path g*› *ab01* ‹*a < b*›
by (*force simp*: *simple-path-def loop-free-def*)+
have *g01*: *g 0 = g 1*
using *assms(2)* **by** (*simp add*: *pathfinish-def pathstart-def*)
have *pi-eq*: *path-image g = {g 0} ∪ g ‘ {0<..<1}*
using *g01* **by** (*fastforce simp*: *path-image-def image-iff*)
have *int-sub*: *g ‘ {0<..<1} ⊆ interior (convex hull (path-image g))*
using *interior-subset ab-eq* **by** *simp*
— Every extreme point of the convex hull lies in *path-image g* but not in the interior
have *ext-sub*: *{x. x extreme-point-of (convex hull (path-image g))} ⊆ {g 0}*
using *extreme-point-of-convex-hull extreme-point-not-in-interior int-sub* **by** (*fastforce simp*: *pi-eq*)
— By Krein–Milman, the convex hull collapses to a single point
have *compact-hull*: *compact (convex hull (path-image g))*
by (*rule compact-convex-hull*[*OF compact-simple-path-image*[*OF* ‹*simple-path g*›]])
have *convex hull (path-image g) = convex hull {x. x extreme-point-of (convex hull (path-image g))}*
using *Krein-Milman-Minkowski*[*OF compact-hull convex-convex-hull*] **by** *simp*
also have *... ⊆ convex hull {g 0}*
using *ext-sub* **by** (*intro hull-mono*)
also have *... = {g 0}* **by** (*simp add*: *convex-hull-singleton*)

```
    finally have convex hull (path-image g) ⊆ {g 0} .
    with interior-ne show ?thesis
      by (simp add: subset-singleton-iff)
  next
    case False
    have hull-eq: convex hull (g ` ({0..1} - {a<..<b})) = convex hull (path-image
g)
    proof
      show convex hull (g ` ({0..1} - {a<..<b})) ⊆ convex hull (path-image g)
        by (intro hull-mono image-mono) (auto simp: path-image-def)
        — For ⊇, use extreme points: they lie in path-image g but not in the interior
      have compact-hull: compact (convex hull (path-image g))
      by (rule compact-convex-hull[OF compact-simple-path-image[OF ‹simple-path
g›]])
       have ext-in-rest: {x. x extreme-point-of (convex hull (path-image g))} ⊆ g `
({0..1} - {a<..<b})
      using extreme-point-of-convex-hull extreme-point-not-in-interior interior-subset
        unfolding path-image-def by blast
      show convex hull (path-image g) ⊆ convex hull (g ` ({0..1} - {a<..<b}))
        using Krein-Milman-Minkowski[OF compact-hull convex-convex-hull]
        using ext-in-rest hull-mono by blast
    qed
     have hull-seg-eq: convex hull (closed-segment (g a) (g b) ∪ g ` ({0..1} -
{a<..<b})) = convex hull (path-image g)
    proof
      have g a ∈ g ` ({0..1} - {a<..<b}) g b ∈ g ` ({0..1} - {a<..<b})
        using ab01 ‹a < b› by auto
       then have seg-sub: closed-segment (g a) (g b) ⊆ convex hull (g ` ({0..1} -
{a<..<b}))
        by (meson closed-segment-subset convex-convex-hull hull-inc)
      show convex hull (closed-segment (g a) (g b) ∪ g ` ({0..1} - {a<..<b})) ⊆
convex hull (path-image g)
        by (metis hull-eq convex-convex-hull hull-subset le-supI seg-sub subset-hull)
     show convex hull (path-image g) ⊆ convex hull (closed-segment (g a) (g b) ∪
g ` ({0..1} - {a<..<b}))
        by (metis Un-commute Un-upper1 hull-eq hull-mono)
    qed
  — Step 1: double arc decomposition of g
    obtain g0 g1 where arcs:
      arc g0 arc g1
      pathstart g0 = g a pathfinish g0 = g b
      pathstart g1 = g b pathfinish g1 = g a
      path-image g0 = g ` {a..b}
      path-image g1 = g ` ({0..1} - {a<..<b})
      (path-image g0) ∩ (path-image g1) = {g a, g b}
      (path-image g0) ∪ (path-image g1) = path-image g
      using exists-double-arc-explicit[OF ‹simple-path g› ‹pathfinish g = pathstart
g›
          ab01(1) ab01(2) less-imp-le[OF ‹a < b›] False] by blast
```

— Step 2: the frontier of the convex hull admits a rectifiable simple loop parametrization (this corresponds to HOL Light's *RECTIFIABLE-LOOP-RELATIVE-FRONTIER-CONVEX*

obtain *d* **where** *d-props*:
simple-path d pathfinish d = pathstart d rectifiable-path d
path-image d = frontier (convex hull (path-image g))
by (*metis assms(1) bounded-convex-hull bounded-simple-path-image convex-convex-hull interior-ne*
rectifiable-loop-frontier-convex)

— Step 4: double arc decomposition of the frontier loop *d*, with inside decomposition

have *ga-ne-gb*: *g a ≠ g b* **using** *False* **.**
obtain *d0 d1* **where** *d-split*:
arc d0 arc d1
pathstart d0 = g a pathfinish d0 = g b
pathstart d1 = g b pathfinish d1 = g a
path-image d0 ∩ path-image d1 = {g a, g b}
path-image d0 ∪ path-image d1 = path-image d
inside (path-image d0 ∪ path-image g0) ∩ inside (path-image d1 ∪ path-image g0) = {}
inside (path-image d0 ∪ path-image g0) ∪ inside (path-image d1 ∪ path-image g0) ∪ (path-image g0 − {g a, g b}) =
interior (convex hull (path-image g))
(path-image g1 − {g b, g a}) ∩ path-image d0 = {}
proof −
have *ga-d*: *g a ∈ path-image d* **and** *gb-d*: *g b ∈ path-image d*
using *d-props(4) assms(7,8)* **by** *simp-all*
obtain *d0 d1* **where** *da*:
arc d0 arc d1
pathstart d0 = g a pathfinish d0 = g b
pathstart d1 = g b pathfinish d1 = g a
path-image d0 ∩ path-image d1 = {g a, g b}
path-image d0 ∪ path-image d1 = path-image d
using *exists-double-arc*[*OF d-props(1) d-props(2) ga-d gb-d ga-ne-gb*] **by** *metis*
— Endpoints and basic simple-path facts for the frontier arcs *d0*, *d1*
have *sp-d0*: *simple-path d0* **and** *sp-d1*: *simple-path d1*
using *da(1,2) arc-imp-simple-path* **by** *blast+*
have *rev-ends*: *pathstart (reversepath d1) = g a pathfinish (reversepath d1) = g b*
using *da(5,6)* **by** (*simp-all add*: *pathstart-reversepath pathfinish-reversepath*)
have *sp-rev-d1*: *simple-path (reversepath d1)*
using *sp-d1* **by** (*simp add*: *simple-path-reversepath*)
have *sp-g0*: *simple-path g0* **using** *arcs(1) arc-imp-simple-path* **by** *blast*
note *g0-ends = arcs(3,4)*
have *gab-d1*: *g a ∈ path-image d1 g b ∈ path-image d1*
using *da(5,6)* **by** (*metis pathstart-in-path-image pathfinish-in-path-image*)+
have *d0-sub*: *path-image d0 ⊆ path-image d* **and** *d1-sub*: *path-image d1 ⊆ path-image d*

using *da*(*8*) **by** *blast*+

— The open part of *g0* lies in the interior, hence *g0* meets the frontier only at *g a*, *g b*

have *g0-decomp*: *path-image g0* = *g* ‘ {*a*<..<*b*} ∪ {*g a*, *g b*}

using ‹*a* < *b*› **by** (*force simp*: *arcs*(*7*) *image-iff*)

have *g0-d-int*: *path-image g0* ∩ *path-image d* = {*g a*, *g b*}

using *assms*(*9*) *d-props*(*4*) *ga-d gb-d g0-decomp* **by** *auto*

have *d0-g0-int*: *path-image d0* ∩ *path-image g0* = {*g a*, *g b*}

using *da*(*7*,*8*) *g0-d-int* **by** *auto*

have *d1-g0-int*: *path-image d1* ∩ *path-image g0* = {*g a*, *g b*}

using *d1-sub da*(*7*) *g0-d-int* **by** *auto*

— Split the inside via *SPLIT-INSIDE-SIMPLE-CLOSED-CURVE* on *d0*, *reversepath d1*, *g0*

have *inside-eq*: *inside* (*path-image d0* ∪ *path-image* (*reversepath d1*)) = *interior* (*convex hull path-image g*)

by (*simp add*: *assms*(*1*) *bounded-convex-hull bounded-simple-path-image d-props*(*4*) *da*(*8*)

inside-frontier-eq-interior)

have *g* ‘ {*a*<..<*b*} ≠ {}

using ‹*a* < *b*› **by** *auto*

then have *g0-inside-ne*: *path-image g0* ∩ *inside* (*path-image d0* ∪ *path-image* (*reversepath d1*)) ≠ {}

using ‹*a* < *b*› *g0-decomp interior-subset inside-eq* **by** *blast*

have *d0-rev-int*: *path-image d0* ∩ *path-image* (*reversepath d1*) = {*g a*, *g b*}

using *da*(*7*) *path-image-reversepath* **by** *simp*

have *split*:

inside (*path-image d0* ∪ *path-image g0*) ∩ *inside* (*path-image* (*reversepath d1*) ∪ *path-image g0*) = {}

inside (*path-image d0* ∪ *path-image g0*) ∪ *inside* (*path-image* (*reversepath d1*) ∪ *path-image g0*) ∪ (*path-image g0* − {*g a*, *g b*}) = *inside* (*path-image d0* ∪ *path-image* (*reversepath d1*))

using *split-inside-simple-closed-curve*[*OF sp-d0 da*(*3*,*4*) *sp-rev-d1 rev-ends sp-g0 g0-ends ga-ne-gb*

d0-rev-int d0-g0-int - *g0-inside-ne*]

by (*simp-all add*: *path-image-reversepath d1-g0-int*)

have *split1*: *inside* (*path-image d0* ∪ *path-image g0*) ∩ *inside* (*path-image d1* ∪ *path-image g0*) = {}

using *split*(*1*) *path-image-reversepath* **by** *simp*

have *split2*: *inside* (*path-image d0* ∪ *path-image g0*) ∪ *inside* (*path-image d1* ∪ *path-image g0*) ∪ (*path-image g0* − {*g a*, *g b*}) = *interior* (*convex hull path-image g*)

using *split*(*2*) *path-image-reversepath inside-eq* **by** *simp*

— Step 4 (cont.): orient the split so that *g1*'s interior avoids *d0*. This is a connectedness argument on *path-image g1* − {*g a*, *g b*}.

have *arc-rev-g0*: *arc* (*reversepath g0*) **using** *arcs*(*1*) **by** (*simp add*: *arc-reversepath*)

have *J0-loop*: *simple-path* (*d0* +++ *reversepath g0*)

proof (*intro simple-path-join-loop da arc-rev-g0*)

qed (*use da g0-ends d0-g0-int* **in** *auto*)

have *J0-close*: *pathfinish* (*d0* +++ *reversepath g0*) = *pathstart* (*d0* +++

reversepath g0)
using *da*(*3*) *g0-ends* **by** (*simp add*: *pathstart-reversepath pathfinish-reversepath*)
have *J0-pi*: *path-image* (*d0 +++ reversepath g0*) = *path-image d0* ∪ *path-image g0*
using *da*(*4*) *g0-ends* **by** (*simp add*: *path-image-join pathstart-reversepath path-image-reversepath*)
have *J1-loop*: *simple-path* (*reversepath d1 +++ reversepath g0*)
proof (*intro simple-path-join-loop arc-rev-g0*)
show *arc* (*reversepath d1*) **using** *da*(*2*) **by** (*simp add*: *arc-reversepath*)
qed (*use da g0-ends d1-g0-int* **in** *auto*)
have *J1-close*: *pathfinish* (*reversepath d1 +++ reversepath g0*) = *pathstart* (*reversepath d1 +++ reversepath g0*)
using *da*(*6*) *g0-ends* **by** (*simp add*: *pathstart-reversepath pathfinish-reversepath*)
have *J1-pi*: *path-image* (*reversepath d1 +++ reversepath g0*) = *path-image d1* ∪ *path-image g0*
using *da*(*5*) *g0-ends* **by** (*simp add*: *path-image-join pathstart-reversepath pathfinish-reversepath path-image-reversepath*)
have *J0-jio*: *frontier* (*inside* (*path-image d0* ∪ *path-image g0*)) = *path-image d0* ∪ *path-image g0*
using *Jordan-inside-outside*[*OF J0-loop J0-close*] *J0-pi* **by** *simp*
have *J1-jio*: *frontier* (*inside* (*path-image d1* ∪ *path-image g0*)) = *path-image d1* ∪ *path-image g0*
using *Jordan-inside-outside*[*OF J1-loop J1-close*] *J1-pi* **by** *simp*
have *cl-J0*: *closure* (*inside* (*path-image d0* ∪ *path-image g0*)) = *inside* (*path-image d0* ∪ *path-image g0*) ∪ *path-image d0* ∪ *path-image g0*
by (*simp add*: *J0-jio Un-ac*(*1*) *closure-Un-frontier*)
have *cl-J1*: *closure* (*inside* (*path-image d1* ∪ *path-image g0*)) = *inside* (*path-image d1* ∪ *path-image g0*) ∪ *path-image d1* ∪ *path-image g0*
by (*simp add*: *J1-jio Un-assoc closure-Un-frontier*)
have *sp-g1*: *simple-path g1* **using** *arcs*(*2*) *arc-imp-simple-path* **by** *blast*
define *S* **where** *S* = *path-image g1* − {*g b*, *g a*}
have *S-conn*: *connected S*
using *connected-simple-path-endless*[*OF sp-g1*] *arcs*(*5*,*6*) **unfolding** *S-def*
by (*simp add*: *insert-commute*)
have *g1-g0-int*: *path-image g1* ∩ *path-image g0* = {*g a*, *g b*}
using *arcs*(*9*) **by** (*simp add*: *Int-commute*)
have *S-g0*: *S* ∩ *path-image g0* = {}
using *g1-g0-int* **unfolding** *S-def* **by** *blast*
have *cldd0*: *closed* (*path-image d0* ∪ *path-image g0*)
using *da*(*1*) *arcs*(*1*) **by** (*simp add*: *closed-path-image arc-imp-path closed-Un*)
have *cldd1*: *closed* (*path-image d1* ∪ *path-image g0*)
using *da*(*2*) *arcs*(*1*) **by** (*simp add*: *closed-path-image arc-imp-path closed-Un*)
have *g1-in-hull*: *path-image g1* ⊆ *convex hull path-image g*
by (*metis arcs*(*8*) *hull-subset local.hull-eq*)
have *hull-closed*: *closed* (*convex hull path-image g*)
by (*simp add*: *compact-imp-closed compact-convex-hull compact-simple-path-image* ‹*simple-path g*›)
have *g1-cover*: *path-image g1* ⊆ *interior* (*convex hull path-image g*) ∪ *frontier* (*convex hull path-image g*)

using *g1-in-hull hull-closed* **by** (*simp add*: *frontier-def closure-closed*) *blast*

— S (which is *g1* minus its endpoints) is covered by the two *inside*-plus-arc regions ...

have *cover*: $\bigwedge z.\ z \in S \Longrightarrow z \in (\mathit{inside}\ (\mathit{path\text{-}image}\ d1 \cup \mathit{path\text{-}image}\ g0) \cup \mathit{path\text{-}image}\ d1) \cup (\mathit{path\text{-}image}\ d0 \cup \mathit{inside}\ (\mathit{path\text{-}image}\ d0 \cup \mathit{path\text{-}image}\ g0))$

using *IntE IntI S-def S-g0 Un-iff d-props*(*4*) *da*(*8*) *g1-cover split2* **by** *fastforce*

have *ic0*: $\mathit{inside}\ (\mathit{path\text{-}image}\ d1 \cup \mathit{path\text{-}image}\ g0) \cap \mathit{closure}\ (\mathit{inside}\ (\mathit{path\text{-}image}\ d0 \cup \mathit{path\text{-}image}\ g0)) = \{\}$

using *open-Int-closure-eq-empty cldd1 split1* **by** (*metis Int-commute open-inside*)

have *ic1*: $\mathit{inside}\ (\mathit{path\text{-}image}\ d0 \cup \mathit{path\text{-}image}\ g0) \cap \mathit{closure}\ (\mathit{inside}\ (\mathit{path\text{-}image}\ d1 \cup \mathit{path\text{-}image}\ g0)) = \{\}$

using *open-Int-closure-eq-empty cldd0 split1* **by** (*metis open-inside*)

have *d0-cl*: $\mathit{path\text{-}image}\ d0 \subseteq \mathit{closure}\ (\mathit{inside}\ (\mathit{path\text{-}image}\ d0 \cup \mathit{path\text{-}image}\ g0))$

using *cl-J0* **by** *blast*

have *d1-cl*: $\mathit{path\text{-}image}\ d1 \subseteq \mathit{closure}\ (\mathit{inside}\ (\mathit{path\text{-}image}\ d1 \cup \mathit{path\text{-}image}\ g0))$

using *cl-J1* **by** *blast*

have *in1-d0*: $\mathit{inside}\ (\mathit{path\text{-}image}\ d1 \cup \mathit{path\text{-}image}\ g0) \cap \mathit{path\text{-}image}\ d0 = \{\}$

using *ic0 d0-cl* **by** *blast*

have *d1-in0*: $\mathit{path\text{-}image}\ d1 \cap \mathit{inside}\ (\mathit{path\text{-}image}\ d0 \cup \mathit{path\text{-}image}\ g0) = \{\}$

using *ic1 d1-cl* **by** *blast*

have *in1-in0*: $\mathit{inside}\ (\mathit{path\text{-}image}\ d1 \cup \mathit{path\text{-}image}\ g0) \cap \mathit{inside}\ (\mathit{path\text{-}image}\ d0 \cup \mathit{path\text{-}image}\ g0) = \{\}$

using *split1* **by** (*simp add*: *Int-commute*)

have *d1-d0-S*: $\bigwedge z.\ z \in S \Longrightarrow \neg(z \in \mathit{path\text{-}image}\ d1 \wedge z \in \mathit{path\text{-}image}\ d0)$

using *da*(*7*) **unfolding** *S-def* **by** *blast*

— ... and the two regions are disjoint on S, so S splits into two relatively clopen pieces

have *disj*: $\bigwedge z.\ z \in S \Longrightarrow \neg(z \in (\mathit{inside}\ (\mathit{path\text{-}image}\ d1 \cup \mathit{path\text{-}image}\ g0) \cup \mathit{path\text{-}image}\ d1) \wedge z \in (\mathit{path\text{-}image}\ d0 \cup \mathit{inside}\ (\mathit{path\text{-}image}\ d0 \cup \mathit{path\text{-}image}\ g0)))$

using *in1-d0 in1-in0 d1-in0 d1-d0-S* **by** *blast*

have *eqA*: $S - \mathit{closure}\ (\mathit{inside}\ (\mathit{path\text{-}image}\ d1 \cup \mathit{path\text{-}image}\ g0)) = S \cap (\mathit{path\text{-}image}\ d0 \cup \mathit{inside}\ (\mathit{path\text{-}image}\ d0 \cup \mathit{path\text{-}image}\ g0))$

using *cl-J1 S-g0 cover disj* **by** *blast*

have *eqB*: $S - \mathit{closure}\ (\mathit{inside}\ (\mathit{path\text{-}image}\ d0 \cup \mathit{path\text{-}image}\ g0)) = S \cap (\mathit{path\text{-}image}\ d1 \cup \mathit{inside}\ (\mathit{path\text{-}image}\ d1 \cup \mathit{path\text{-}image}\ g0))$

using *cl-J0 S-g0 cover disj* **by** *blast*

have *opA*: $\mathit{openin}\ (\mathit{top\text{-}of\text{-}set}\ S)\ (S - \mathit{closure}\ (\mathit{inside}\ (\mathit{path\text{-}image}\ d1 \cup \mathit{path\text{-}image}\ g0)))$

by (*simp add*: *Diff-eq open-Compl openin-open-Int*)

have *opB*: $\mathit{openin}\ (\mathit{top\text{-}of\text{-}set}\ S)\ (S - \mathit{closure}\ (\mathit{inside}\ (\mathit{path\text{-}image}\ d0 \cup \mathit{path\text{-}image}\ g0)))$

by (*metis Diff-eq interior-complement open-interior openin-open-Int*)

have *AB-cover*: $(S - \mathit{closure}\ (\mathit{inside}\ (\mathit{path\text{-}image}\ d1 \cup \mathit{path\text{-}image}\ g0))) \cup (S - \mathit{closure}\ (\mathit{inside}\ (\mathit{path\text{-}image}\ d0 \cup \mathit{path\text{-}image}\ g0))) = S$

using *eqA eqB cover* **by** *blast*

have *AB-disj*: $(S - \mathit{closure}\ (\mathit{inside}\ (\mathit{path\text{-}image}\ d1 \cup \mathit{path\text{-}image}\ g0))) \cap (S$

− *closure* (*inside* (*path-image d0* ∪ *path-image g0*))) = {}
using *eqA eqB in1-d0 in1-in0 d1-in0 d1-d0-S split1* **by** *auto*
have *one-empty*: (*S* − *closure* (*inside* (*path-image d1* ∪ *path-image g0*))) = {} ∨ (*S* − *closure* (*inside* (*path-image d0* ∪ *path-image g0*))) = {}
using *S-conn opA opB AB-cover AB-disj connected-openin* **by** *blast*
have *disjunction*: *S* ∩ *path-image d0* = {} ∨ *S* ∩ *path-image d1* = {}
using *eqA eqB one-empty* **by** *auto*
— Whichever arc *S* avoids becomes *d0* (reversing the pair in the second case)
show *thesis*
proof (*cases S* ∩ *path-image d0* = {})
case *True* **then show** *thesis*
using *S-def da split1 split2 that* **by** *auto*
next
case *False*
then have *d1e*: (*path-image g1* − {*g b*, *g a*}) ∩ *path-image d1* = {}
using *disjunction* **unfolding** *S-def* **by** *blast*
then show *thesis*
using *da arc-reversepath pathstart-reversepath pathfinish-reversepath*
by (*metis* (*no-types*, *lifting*) *Int-commute path-image-reversepath split1 split2 sup-commute*
that)
qed
qed
— Step 3: *g*(*a*) and *g*(*b*) are on the *path-image d*
have *ga-in-d*: *g a* ∈ *path-image d* **and** *gb-in-d*: *g b* ∈ *path-image d*
using *d-props*(*4*) *assms*(*7*,*8*) **by** *auto*
— Step 4: build *h* as the straight segment from *g*(*a*) to *g*(*b*) on (*a*, *b*), unchanged elsewhere. (*front-arc* is no longer used; see *step-lemma-proof-body.thy* in memory for derivation notes.)
show *?thesis*
proof −
have *hull-closed*: *closed* (*convex hull path-image g*)
by (*simp add*: *compact-imp-closed compact-convex-hull compact-simple-path-image* ‹*simple-path g*›)
have *compact-hull*: *compact* (*convex hull path-image g*)
by (*simp add*: *compact-convex-hull compact-simple-path-image* ‹*simple-path g*›)
have *d1-sub-hull*: *convex hull* (*path-image d1*) ⊆ *convex hull* (*path-image g*)
using *d-props*(*4*) *d-split*(*8*) *frontier-subset-eq hull-closed*
by (*intro hull-minimal*) *auto*
have *km-ext*: *convex hull path-image g* = *convex hull* {*x*. *x extreme-point-of* (*convex hull path-image g*)}
using *Krein-Milman-Minkowski*[*OF compact-hull convex-convex-hull*] **by** *simp*
have *test-cl*: ⋀*x*. *x* ∈ *path-image g* ⟹ *x* ∈ *closure* (*convex hull path-image g*)
using *closure-subset hull-subset* **by** (*meson subset-iff*)
have *test-front*: ⋀*x*. *x* ∈ *closure* (*convex hull path-image g*) ⟹ *x* ∉ *interior*

(*convex hull path-image g*) ⟹ *x* ∈ *frontier* (*convex hull path-image g*)
by (*simp add*: *frontier-def*)
have *ext-in-g-front*: ⋀*x*. *x extreme-point-of* (*convex hull path-image g*) ⟹ *x* ∈ *path-image g* ∩ *path-image d*
by (*simp add*: *d-props*(*4*) *extreme-point-not-in-interior extreme-point-of-convex-hull test-cl*
test-front)
have *gab-d1*: *g a* ∈ *path-image d1 g b* ∈ *path-image d1*
using *d-split*(*5*,*6*) **by** (*metis pathstart-in-path-image pathfinish-in-path-image*)+
have *g0-int-d*: *path-image g0* ∩ *path-image d* = {*g a*, *g b*}
proof −
have {*a*..*b*} = {*a*<..<*b*} ∪ {*a*, *b*} **using** ‹*a* < *b*› **by** *auto*
then have *path-image g0* = *g* ‘ {*a*<..<*b*} ∪ {*g a*, *g b*}
using *arcs* **by** *simp*
moreover have {*g a*, *g b*} ⊆ *path-image d* **using** *ga-in-d gb-in-d* **by** *simp*
ultimately show *?thesis*
using *assms*(*9*) *d-props*(*4*) **by** *blast*
qed
have *g1-int-d*: *path-image g1* ∩ *path-image d* ⊆ *path-image d1*
using *d-split* **by** *auto*
have *hull-d1-eq*: *convex hull* (*path-image d1*) = *convex hull* (*path-image g*)
proof (*intro d1-sub-hull subset-antisym*)
have *path-image g* ∩ *path-image d* ⊆ *path-image d1*
proof −
have *path-image g* ∩ *path-image d* = (*path-image g0* ∩ *path-image d*) ∪ (*path-image g1* ∩ *path-image d*)
using *arcs*(*10*) **by** *blast*
also have ... ⊆ *path-image d1* **using** *g0-int-d g1-int-d gab-d1* **by** *simp*
finally show *?thesis* **.**
qed
then have {*x*. *x extreme-point-of* (*convex hull path-image g*)} ⊆ *path-image d1*
using *ext-in-g-front* **by** *blast*
then have *convex hull* {*x*. *x extreme-point-of* (*convex hull path-image g*)} ⊆ *convex hull* (*path-image d1*)
by (*rule hull-mono*)
then show *convex hull* (*path-image g*) ⊆ *convex hull* (*path-image d1*) **using** *km-ext* **by** *simp*
qed
— The straight segment between $g(a)$ and $g(b)$ lies on the frontier, via *seg-frontier-aux* applied to the arc *d1* (whose hull is the whole convex hull by *hull-d1-eq*).
have *seg-in-frontier*: *closed-segment* (*g a*) (*g b*) ⊆ *frontier* (*convex hull path-image g*)
proof (*rule seg-frontier-aux*[*of - - - d1*])
show *path-image d1* ⊆ *frontier* (*convex hull path-image g*)
using *d-split*(*8*) *d-props*(*4*) **by** *blast*
show *connected* (*path-image d1* − {*g a*, *g b*})
using *connected-simple-path-endless*[*OF arc-imp-simple-path*[*OF d-split*(*2*)]]

d-split(*5*,*6*)
by (*simp add*: *insert-commute*)
show *convex hull path-image d1* = *convex hull path-image g* **by** (*rule hull-d1-eq*)
qed (*use assms gab-d1 ga-ne-gb convex-convex-hull compact-hull interior-ne* **in** *auto*)
have *rev-d1-arc*: *arc* (*reversepath d1*) **using** *d-split*(*2*) **by** (*simp add*: *arc-reversepath*)
have *rev-d1-start*: *pathstart* (*reversepath d1*) = *g a* **using** *d-split*(*6*) **by** (*simp add*: *pathstart-reversepath*)
have *rev-d1-finish*: *pathfinish* (*reversepath d1*) = *g b* **using** *d-split*(*5*) **by** (*simp add*: *pathfinish-reversepath*)
have *rev-d1-pi*: *path-image* (*reversepath d1*) = *path-image d1* **by** (*simp add*: *path-image-reversepath*)
have *d0d1-pi*: *path-image d0* ∪ *path-image* (*reversepath d1*) = *frontier* (*convex hull path-image g*)
using *rev-d1-pi d-split*(*8*) *d-props*(*4*) **by** *simp*
have *int-ends*: *path-image d0* ∩ *path-image* (*reversepath d1*) = {*pathstart d0*, *pathfinish d0*}
using *d-split*(*7*) *rev-d1-pi d-split*(*3*,*4*) **by** *simp*
have *seg-sub-union*: *closed-segment* (*g a*) (*g b*) ⊆ *path-image d0* ∪ *path-image* (*reversepath d1*)
using *seg-in-frontier d0d1-pi* **by** *simp*
have *dich*: *path-image d0* ⊆ *closed-segment* (*g a*) (*g b*) ∨ *path-image* (*reversepath d1*) ⊆ *closed-segment* (*g a*) (*g b*)
proof (*rule connected-subset-arc-pair*)
show *arc d0* **by** (*rule d-split*(*1*))
show *arc* (*reversepath d1*) **by** (*rule rev-d1-arc*)
show *pathstart d0* = *pathstart* (*reversepath d1*) **using** *d-split*(*3*) *rev-d1-start* **by** *simp*
show *pathfinish d0* = *pathfinish* (*reversepath d1*) **using** *d-split*(*4*) *rev-d1-finish* **by** *simp*
show *path-image d0* ∩ *path-image* (*reversepath d1*) = {*pathstart d0*, *pathfinish d0*} **by** (*rule int-ends*)
show *connected* (*closed-segment* (*g a*) (*g b*)) **by** *simp*
show *closed-segment* (*g a*) (*g b*) ⊆ *path-image d0* ∪ *path-image* (*reversepath d1*) **by** (*rule seg-sub-union*)
show *pathstart d0* ∈ *closed-segment* (*g a*) (*g b*) **using** *d-split*(*3*) **by** *simp*
show *pathfinish d0* ∈ *closed-segment* (*g a*) (*g b*) **using** *d-split*(*4*) **by** *simp*
qed
have *False* **if** *path-image* (*reversepath d1*) ⊆ *closed-segment* (*g a*) (*g b*)
proof −
have *convex hull* (*path-image d1*) ⊆ *convex hull* (*closed-segment* (*g a*) (*g b*))
using *that* **by** (*simp add*: *hull-mono*)
also have ... = *closed-segment* (*g a*) (*g b*)
by (*simp add*: *hull-mono hull-same convex-closed-segment*)
finally have *interior* (*convex hull* (*path-image g*)) ⊆ *interior* (*closed-segment* (*g a*) (*g b*))
by (*simp add*: *hull-d1-eq interior-mono*)
also have *interior* (*closed-segment* (*g a*) (*g b*)) = {}

by (*rule interior-closed-segment-ge2*) *simp*
finally show *False* **using** *interior-ne* **by** *simp*
qed
with *dich* **have** *d0-sub-seg*: *path-image d0* ⊆ *closed-segment* (*g a*) (*g b*)
by *blast*
have *d0-eq-seg*: *path-image d0* = *closed-segment* (*g a*) (*g b*)
proof (*rule connected-subset-segment*)
show *connected* (*path-image d0*) **using** *d-split*(*1*) **by** (*simp add*: *connected-arc-image*)
show *path-image d0* ⊆ *closed-segment* (*g a*) (*g b*) **by** (*rule d0-sub-seg*)
show *g a* ∈ *path-image d0* **using** *gab-d1 d-split*(*3*) **by** (*metis path-start-in-path-image*)
show *g b* ∈ *path-image d0* **using** *d-split*(*4*) **by** (*metis pathfinish-in-path-image*)
qed
define *h* **where** *h* ≡ λ*t*. *if t* ∈ {*a*<..<*b*} *then g a* + ((*t* − *a*)/(*b* − *a*)) ∗*R* (*g b* − *g a*) *else g t*
have *interval-img*: (λ*t*. (*t* − *a*)/(*b* − *a*)) ‘ {*a*..*b*} = {*0*..*1*}
proof −
have (λ*t*. (*t* − *a*)/(*b* − *a*)) ‘ {*a*..*b*} = {*a*/(*b*−*a*) − *a*/(*b*−*a*) .. *b*/(*b*−*a*) − *a*/(*b*−*a*)}
using ‹*a* < *b*› **by** (*simp add*: *diff-divide-distrib image-affinity-atLeastAtMost-div-diff*)
also have ... = {*0*..*1*} **using** ‹*a* < *b*› **by** (*simp flip*: *diff-divide-distrib*)
finally show *?thesis* **.**
qed
have *lin*: (λ*t*. *g a* + ((*t* − *a*)/(*b* − *a*)) ∗*R* (*g b* − *g a*)) ‘ {*a*..*b*} = *closed-segment* (*g a*) (*g b*)
proof −
have (λ*t*. *g a* + ((*t* − *a*)/(*b* − *a*)) ∗*R* (*g b* − *g a*)) ‘ {*a*..*b*}
= (λ*u*. *g a* + *u* ∗*R* (*g b* − *g a*)) ‘ ((λ*t*. (*t*−*a*)/(*b*−*a*)) ‘ {*a*..*b*})
by (*simp add*: *image-image*)
also have ... = *closed-segment* (*g a*) (*g b*)
using *interval-img* **by** (*simp add*: *closed-segment-image-interval algebra-simps cong*: *image-cong*)
finally show *?thesis* **.**
qed
have *hh-eq-lin*: ⋀*t*. *t* ∈ {*a*..*b*} ⟹ *h t* = *g a* + ((*t* − *a*)/(*b* − *a*)) ∗*R* (*g b* − *g a*)
using *h-def* **by** *fastforce*
have *hh-seg*: *h* ‘ {*a*..*b*} = *closed-segment* (*g a*) (*g b*)
by (*simp add*: *hh-eq-lin lin*)
have *a-ge0*: *0* ≤ *a* **and** *b-le1*: *b* ≤ *1* **using** *ab01* **by** *auto*
have *hh-0*: *h 0* = *g 0* **using** *a-ge0* **by** (*simp add*: *h-def*)
have *hh-1*: *h 1* = *g 1* **using** *b-le1* **by** (*simp add*: *h-def*)
have *hh-start*: *pathstart h* = *pathstart g* **by** (*simp add*: *pathstart-def hh-0*)
have *hh-finish*: *pathfinish h* = *pathstart g*
using *assms*(*2*) **by** (*simp add*: *pathstart-def pathfinish-def hh-1*)
have *hh-frontier*: *h* ‘ {*a*..*b*} ⊆ *frontier* (*convex hull path-image g*)
using *hh-seg seg-in-frontier* **by** *simp*
have *hh-out-img*: *h* ‘ ({*0*..*1*} − {*a*<..<*b*}) = *g* ‘ ({*0*..*1*} − {*a*<..<*b*})

using *h-def* **by** *auto*
have $\{0..1\} = \{a..b\} \cup (\{0..1\} - \{a<..<b\})$
using *ab01* ‹$a < b$› **by** *auto*
then have *pi-h*: *path-image h* = *closed-segment* (*g a*) (*g b*) ∪ *g* ‘ ($\{0..1\} - \{a<..<b\}$)
by (*metis hh-out-img hh-seg image-Un path-image-def*)
have *front-in-hull*: *frontier* (*convex hull path-image g*) ⊆ *convex hull path-image g*
using *hull-closed* **by** (*simp add*: *frontier-def closure-closed*)
have *hull-hh*: *convex hull* (*path-image h*) = *convex hull* (*path-image g*)
using *hull-seg-eq pi-h* **by** *argo*
have *dist* (*g a*) (*g b*) ≤ *L* ∗ *dist a b* **using** *assms*(*3*) *ab01* **by** *blast*
then have *L-nonneg*: *0* ≤ *L*
by (*smt* (*verit*) *False dist-le-zero-iff zero-le-mult-iff*)
have *dab*: *dist* (*g a*) (*g b*) ≤ *L* ∗ (*b* − *a*)
by (*smt* (*verit*) *assms dist-real-def*)
have *lip-mid*: *L*−*lipschitz-on* $\{a..b\}$ *h*
proof (*rule lipschitz-onI*)
show *0* ≤ *L* **by** (*rule L-nonneg*)
fix *x y* **assume** *x*: $x \in \{a..b\}$ **and** *y*: $y \in \{a..b\}$
have *factor*: *h x* − *h y* = ((*x* − *a*)/(*b* − *a*) − (*y* − *a*)/(*b* − *a*)) ∗$_R$ (*g b* − *g a*)
by (*simp add*: *hh-eq-lin*[*OF x*] *hh-eq-lin*[*OF y*] *scaleR-left-diff-distrib*)
have *e1*: *dist* (*h x*) (*h y*) = |(*x* − *a*)/(*b* − *a*) − (*y* − *a*)/(*b* − *a*)| ∗ *norm* (*g b* − *g a*)
by (*simp add*: *dist-norm factor*)
have *e2*: |(*x* − *a*)/(*b* − *a*) − (*y* − *a*)/(*b* − *a*)| = |*x* − *y*| / (*b* − *a*)
using ‹$a < b$› **by** (*simp add*: *divide-simps*)
have *e3*: *norm* (*g b* − *g a*) / (*b* − *a*) ≤ *L*
by (*metis* ‹$a < b$› *dab diff-gt-0-iff-gt dist-commute dist-norm mult-imp-div-pos-le*)
have *dist* (*h x*) (*h y*) = (*norm* (*g b* − *g a*) / (*b* − *a*)) ∗ |*x* − *y*|
using *e1 e2* **by** *simp*
also have ... ≤ *L* ∗ |*x* − *y*|
using *mult-right-mono*[*OF e3 abs-ge-zero*] **by** *simp*
finally show *dist* (*h x*) (*h y*) ≤ *L* ∗ *dist x y* **by** (*simp add*: *dist-real-def*)
qed
have *lip-g*: *L*−*lipschitz-on* $\{0..1\}$ *g*
by (*rule lipschitz-onI*[*OF - L-nonneg*]) (*rule assms*(*3*))
have *hh-lo*: $\bigwedge x.\ x \in \{0..a\} \Longrightarrow h\ x = g\ x$
by (*auto simp*: *h-def*)
have *hh-hi*: $\bigwedge x.\ x \in \{b..1\} \Longrightarrow h\ x = g\ x$
using ‹$a < b$› **by** (*auto simp*: *h-def*)
have *lip-lo*: *L*−*lipschitz-on* $\{0..a\}$ *h*
by (*smt* (*verit*, *ccfv-threshold*) *ab01*(*1*) *atLeastAtMost-iff hh-lo lip-g lipschitz-on-def*)
have *lip-hi*: *L*−*lipschitz-on* $\{b..1\}$ *h*
by (*smt* (*verit*) *ab01*(*2*) *atLeastAtMost-iff hh-hi lip-g lipschitz-on-def*)
have *lip-lomid*: *L*−*lipschitz-on* $\{0..b\}$ *h*
using *lip-lo lip-mid lipschitz-on-concat-max* **by** *fastforce*

have *lip-hh*: *L−lipschitz-on* {*0..1*} *h*
by (*smt* (*verit*, *ccfv-SIG*) *lip-hi lip-lomid lipschitz-on-concat lipschitz-on-cong*)
have *path-hh*: *path h*
using *lip-hh* **by** (*simp add*: *path-def lipschitz-on-continuous-on*)
have *g1-img*: *g ‘* ({*0..1*} − {*a<..<b*}) = *path-image g1* **using** *arcs*(*8*) **by** *simp*
have *opse-g1-disj*: *open-segment* (*g a*) (*g b*) ∩ *path-image g1* = {}
by (*metis Int-Diff Int-commute d0-eq-seg d-split*(*11*) *insert-commute open-segment-def*)
have *hh-in-opse*: *h x* ∈ *open-segment* (*g a*) (*g b*) **if** *x* ∈ {*a<..<b*} **for** *x*
proof −
have (*x* − *a*)/(*b* − *a*) ∈ {*0<..<1*}
using *that* ‹*a* < *b*› **by** (*auto simp*: *divide-simps*)
then have *g a* + ((*x*−*a*)/(*b*−*a*)) ∗*R* (*g b* − *g a*) ∈ *open-segment* (*g a*) (*g b*)
using *ga-ne-gb* **by** (*auto simp*: *in-segment algebra-simps*)
then show *h x* ∈ *open-segment* (*g a*) (*g b*) **using** *that* **by** (*simp add*: *h-def*)
qed
have *hh-inj-mid*: *x* = *y*
if *x* ∈ {*a<..<b*} *y* ∈ {*a<..<b*} *h x* = *h y* **for** *x y*
using *that False* **by** (*simp add*: *h-def*)
have *g-loop-free*: *loop-free g* **using** *assms*(*1*) **by** (*simp add*: *simple-path-def*)
have *hh-out-g*: ⋀*x*. *x* ∉ {*a<..<b*} ⟹ *h x* = *g x* **unfolding** *h-def* **by** *presburger*
have *hh-out-in-g1*: ⋀*x*. *x* ∈ {*0..1*} ⟹ *x* ∉ {*a<..<b*} ⟹ *h x* ∈ *path-image g1*
using *hh-out-g g1-img* **by** *blast*
have *gloopD*: ⋀*x y*. *x* ∈ {*0..1*} ⟹ *y* ∈ {*0..1*} ⟹ *g x* = *g y* ⟹ *x* = *y* ∨ *x* = *0* ∧ *y* = *1* ∨ *x* = *1* ∧ *y* = *0*
using *g-loop-free* **unfolding** *loop-free-def* **by** *blast*
have *testM1*: ⋀*x y*. *x* ∈ {*a<..<b*} ⟹ *y* ∈ {*0..1*} ⟹ *y* ∉ {*a<..<b*} ⟹ *h x* = *h y* ⟹ *False*
by (*metis disjoint-iff hh-in-opse hh-out-in-g1 opse-g1-disj*)
have *loopfree-hh*: *loop-free h*
unfolding *loop-free-def*
by (*metis gloopD h-def hh-inj-mid testM1*)
have *simple-hh*: *simple-path h*
using *path-hh loopfree-hh* **by** (*simp add*: *simple-path-def*)
have *lip-imp-bv*: *has-bounded-variation-on f* {*c..d*}
if *lip*: *M−lipschitz-on* {*c..d*} *f* **for** *f* :: *real* ⇒ *complex* **and** *M c d*
proof −
show *has-bounded-variation-on f* {*c..d*}
unfolding *has-bounded-variation-on-interval*
proof (*intro exI*[**where** *x*=*max 0* (*M* ∗ (*d* − *c*))] *allI impI*)
fix *D* **assume** *D*: *D division-of* {*c..d*}
have *Mnn*: *0* ≤ *M* **using** *lip* **by** (*simp add*: *lipschitz-on-nonneg*)
have *elem*: *norm* (*f* (*Sup k*) − *f* (*Inf k*)) ≤ *M* ∗ *content k* **if** *k*: *k* ∈ *D* **for** *k*
proof −
obtain *a′ b′* **where** *ab′*: *k* = *cbox a′ b′* **and** *ksub*: *k* ⊆ {*c..d*} **and** *kne*: *k* ≠ {}
using *division-ofD D k* **by** *metis*

then have *leab*: $a' \leq b'$ **using** *ab'* **by** (*simp add*: *box-ne-empty*)
have *infk*: *Inf* $k = a'$ **and** *supk*: *Sup* $k = b'$ **using** *ab' leab* **by** *auto*
have *norm* $(f\ b' - f\ a') =$ *dist* $(f\ b')\ (f\ a')$ **by** (*simp add*: *dist-norm*)
also have $\ldots \leq M *$ *dist* $b'\ a'$ **using** *lipschitz-onD*[*OF lip*] *ksub ab' leab* **by** *simp*
also have *dist* $b'\ a' = b' - a'$ **using** *leab* **by** (*simp add*: *dist-real-def*)
also have $M * (b' - a') = M *$ *content k* **using** *ab' leab* **by** (*simp add*: *content-real*)
finally show *norm* $(f\ (Sup\ k) - f\ (Inf\ k)) \leq M *$ *content k* **using** *infk supk* **by** *simp*
qed
have $(\sum k{\in}D.$ *norm* $(f\ (Sup\ k) - f\ (Inf\ k))) \leq (\sum k{\in}D.\ M *$ *content k*$)$
by (*rule sum-mono*) (*rule elem*)
also have $\ldots = M *$ *content* $\{c..d\}$
by (*metis D box-real*(2) *sum-content.division sum-distrib-left*)
also have ... $\leq$ *max 0* $(M * (d - c))$
using *Mnn* **by** (*simp add*: *content-real-if*)
finally show $(\sum k{\in}D.$ *norm* $(f\ (Sup\ k) - f\ (Inf\ k))) \leq$ *max 0* $(M * (d - c))$ **.**
qed
qed
have *bv-hh*: *has-bounded-variation-on h* $\{0..1\}$ **using** *lip-imp-bv*[*OF lip-hh*] **.**
have *bv-g*: *has-bounded-variation-on g* $\{0..1\}$ **using** *lip-imp-bv*[*OF lip-g*] **.**
have *bv-hh-ab*: *has-bounded-variation-on h* $\{a..b\}$
using *bv-hh has-bounded-variation-on-subset a-ge0 b-le1* ‹*a*<*b*› **by** (*meson atLeastatMost-subset-iff dual-order.refl*)
have *bv-g-ab*: *has-bounded-variation-on g* $\{a..b\}$
using *bv-g has-bounded-variation-on-subset a-ge0 b-le1* ‹*a*<*b*› **by** (*meson atLeastatMost-subset-iff dual-order.refl*)
have *a01*: $a \in \{0..1\}$ **and** *b01*: $b \in \{0..1\}$ **using** *ab01* **by** *auto*
have *bv-hh-a1*: *has-bounded-variation-on h* $\{a..1\}$
using *bv-hh has-bounded-variation-on-subset a-ge0 b-le1* ‹*a*<*b*› **by** (*meson atLeastatMost-subset-iff order-refl*)
have *bv-g-a1*: *has-bounded-variation-on g* $\{a..1\}$
using *bv-g has-bounded-variation-on-subset a-ge0 b-le1* ‹*a*<*b*› **by** (*meson atLeastatMost-subset-iff order-refl*)
have *split-hh*: *vector-variation* $\{0..1\}$ *h* = *vector-variation* $\{0..a\}$ *h* + *vector-variation* $\{a..b\}$ *h* + *vector-variation* $\{b..1\}$ *h*
using *vector-variation-combine*[*OF bv-hh a01*] *vector-variation-combine*[*OF bv-hh-a1*] ‹*a*<*b*› *b-le1*
by *simp*
have *split-g*: *vector-variation* $\{0..1\}$ *g* = *vector-variation* $\{0..a\}$ *g* + *vector-variation* $\{a..b\}$ *g* + *vector-variation* $\{b..1\}$ *g*
using *vector-variation-combine*[*OF bv-g a01*] *vector-variation-combine*[*OF bv-g-a1*] ‹*a*<*b*› *b-le1*
by *simp*
have *lo-eq*: *vector-variation* $\{0..a\}$ *h* = *vector-variation* $\{0..a\}$ *g*
by (*rule vector-variation-cong*) (*simp add*: *hh-lo*)
have *hi-eq*: *vector-variation* $\{b..1\}$ *h* = *vector-variation* $\{b..1\}$ *g*

by (*rule vector-variation-cong*) (*simp add*: *hh-hi*)
have *hh-a*: $h\ a = g\ a$ **using** ‹*a*<*b*› **by** (*simp add*: *h-def*)
have *hh-b*: $h\ b = g\ b$ **using** ‹*a*<*b*› **by** (*simp add*: *h-def*)
have *mid-hh-ge*: *vector-variation* $\{a..b\}\ h \geq norm\ (g\ b - g\ a)$
using *vector-variation-ge-norm-function*[*OF bv-hh-ab*, *of b a*] ‹*a*<*b*› *hh-a hh-b*
by *fastforce*
have *mid-hh-le*: *vector-variation* $\{a..b\}\ h \leq norm\ (g\ b - g\ a)$
proof −
have *key*: $(\sum k{\in}D.\ norm\ (h\ (Sup\ k) - h\ (Inf\ k))) \leq norm\ (g\ b - g\ a)$
if *D*: *D division-of* $\{a..b\}$ **for** *D*
proof −
have *elem*: $norm\ (h\ (Sup\ k) - h\ (Inf\ k)) = ((Sup\ k - Inf\ k)/(b-a)) * norm\ (g\ b - g\ a)$
if *k*: $k \in D$ **for** *k*
proof −
obtain $a'\ b'$ **where** *ab'*: $k = cbox\ a'\ b'$ **and** *ksub*: $k \subseteq \{a..b\}$ **and** *kne*: $k \neq \{\}$
using *division-ofD D k* **by** *metis*
then have *leab*: $a' \leq b'$ **using** *ab'* **by** (*simp add*: *box-ne-empty*)
have *infk*: $Inf\ k = a'$ **and** *supk*: $Sup\ k = b'$ **using** *ab' leab* **by** *auto*
have *a'*: $a' \in \{a..b\}$ **and** *b'*: $b' \in \{a..b\}$ **using** *ksub ab' leab* **by** *auto*
have $h\ b' - h\ a' = ((b'-a')/(b-a)) *_R (g\ b - g\ a)$
by (*simp add*: *hh-eq-lin*[*OF a'*] *hh-eq-lin*[*OF b'*] *scaleR-left-diff-distrib diff-divide-distrib*)
then have $norm\ (h\ b' - h\ a') = |(b'-a')/(b-a)| * norm\ (g\ b - g\ a)$ **by** *simp*
also have $|(b'-a')/(b-a)| = (b'-a')/(b-a)$ **using** *leab* ‹*a*<*b*› **by** *simp*
finally show $norm\ (h\ (Sup\ k) - h\ (Inf\ k)) = ((Sup\ k - Inf\ k)/(b-a)) * norm\ (g\ b - g\ a)$
using *infk supk* **by** *simp*
qed
have *content-elem*: $(Sup\ k - Inf\ k) = content\ k$ **if** *k*: $k \in D$ **for** *k*
using *division-ofD*(*3*,*4*)[*OF D k*] **by** (*auto simp*: *content-real*)
then have $(\sum k{\in}D.\ norm\ (h\ (Sup\ k) - h\ (Inf\ k))) = (\sum k{\in}D.\ ((Sup\ k - Inf\ k)/(b-a)) * norm\ (g\ b - g\ a))$
using *local.elem* **by** *auto*
also have $\ldots = ((\sum k{\in}D.\ (Sup\ k - Inf\ k))/(b-a)) * norm\ (g\ b - g\ a)$
by (*simp add*: *sum-distrib-right sum-divide-distrib*)
also have $(\sum k{\in}D.\ (Sup\ k - Inf\ k)) = (\sum k{\in}D.\ content\ k)$
using *content-elem* **by** *force*
also have $(\sum k{\in}D.\ content\ k) = content\ \{a..b\}$
by (*metis D additive-content-division cbox-interval*)
also have $content\ \{a..b\} = b - a$ **using** ‹*a*<*b*› **by** (*simp add*: *content-real*)
also have $((b-a)/(b-a)) * norm\ (g\ b - g\ a) = norm\ (g\ b - g\ a)$ **using** ‹*a*<*b*› **by** *simp*
finally show $(\sum k{\in}D.\ norm\ (h\ (Sup\ k) - h\ (Inf\ k))) \leq norm\ (g\ b - g\ a)$ **by** *simp*
qed

show *?thesis*
using *bv-hh-ab key has-bounded-vector-variation-on-interval* **by** *blast*
qed
have *mid-hh-eq*: *vector-variation* $\{a..b\}$ h = *norm* $(g\ b - g\ a)$
using *mid-hh-ge mid-hh-le* **by** *simp*
define c **where** $c = (a + b)/2$
have *c-mem*: $c \in \{a<..<b\}$ **and** *ac*: $a < c$ **and** *cb*: $c < b$
using ‹$a<b$› **by** (*auto simp*: *c-def*)
have *gc-not-seg*: $g\ c \notin$ *closed-segment* $(g\ a)$ $(g\ b)$
using *c-mem frontier-def local.interior-subset seg-in-frontier* **by** *fastforce*
have ¬ *between* $(g\ a, g\ b)$ $(g\ c)$ **using** *gc-not-seg* **by** (*simp add*: *between-mem-segment*)
with *dist-triangle* **have** *strict-tri*: *dist* $(g\ a)$ $(g\ b)$ < *dist* $(g\ a)$ $(g\ c)$ + *dist* $(g\ c)$ $(g\ b)$
using *between order-less-le* **by** *blast*
have *c-in-ab*: $c \in \{a..b\}$ **using** *ac cb* **by** *simp*
have *bv-g-ac*: *has-bounded-variation-on* g $\{a..c\}$
using *bv-g-ab has-bounded-variation-on-subset ac cb* **by** (*meson atLeastatMost-subset-iff order-refl less-imp-le*)
have *bv-g-cb*: *has-bounded-variation-on* g $\{c..b\}$
using *bv-g-ab has-bounded-variation-on-subset ac cb* **by** (*meson atLeastatMost-subset-iff order-refl less-imp-le*)
have *mid-g-gt*: *vector-variation* $\{a..b\}$ g > *norm* $(g\ b - g\ a)$
proof −
have *split*: *vector-variation* $\{a..b\}$ g = *vector-variation* $\{a..c\}$ g + *vector-variation* $\{c..b\}$ g
using *vector-variation-combine*[*OF bv-g-ab c-in-ab*] **.**
have *ge1*: *norm* $(g\ c - g\ a) \leq$ *vector-variation* $\{a..c\}$ g
using *vector-variation-ge-norm-function*[*OF bv-g-ac*, *of c a*] *ac cb* **by** (*simp add*: *norm-minus-commute*)
have *ge2*: *norm* $(g\ b - g\ c) \leq$ *vector-variation* $\{c..b\}$ g
using *vector-variation-ge-norm-function*[*OF bv-g-cb*, *of b c*] *ac cb* **by** (*simp add*: *norm-minus-commute*)
have *norm* $(g\ b - g\ a)$ < *norm* $(g\ c - g\ a)$ + *norm* $(g\ b - g\ c)$
using *strict-tri* **by** (*simp add*: *dist-norm norm-minus-commute*)
also have ... ≤ *vector-variation* $\{a..b\}$ g
using *split ge1 ge2* **by** *simp*
finally show *?thesis* **.**
qed
have *path-length-lt*: *path-length* h < *path-length* g
by (*simp add*: *hi-eq lo-eq mid-g-gt mid-hh-eq path-length-def split-g split-hh*)
have *lip-show*: $\bigwedge x\ y.\ x \in \{0..1\} \Longrightarrow y \in \{0..1\} \Longrightarrow$ *dist* $(h\ x)$ $(h\ y) \leq L *$ *dist x y*
using *lipschitz-onD*[*OF lip-hh*] **.**
have *hh-agree*: $\bigwedge x.\ x \notin \{a<..<b\} \Longrightarrow h\ x = g\ x$ **using** *hh-out-g* **.**
show *thesis*
using *that*[*OF simple-hh hh-start hh-finish lip-show path-length-lt hull-hh hh-agree hh-frontier*] **.**
qed
qed

qed

A nonempty bounded open connected subset of the reals is an open interval.

lemma *open-bounded-connected-real-is-interval*:
fixes *c* :: *real set*
assumes *open c connected c c* ≠ {} *bounded c*
shows *c* = {*Inf c*<..<*Sup c*}
proof −
have *isiv*: *is-interval c* **using** *assms*(*2*) **by** (*simp add*: *is-interval-connected-1*)
have *bb*: *bdd-below c* **and** *ba*: *bdd-above c* **using** *assms*(*4*) *bounded-imp-bdd-below bounded-imp-bdd-above* **by** *auto*
have *InfnotIn*: *Inf c* ∉ *c*
proof
assume *Inf c* ∈ *c*
then obtain *b* **where** *b* < *Inf c* {*b*<..*Inf c*} ⊆ *c*
using *open-left*[*OF assms*(*1*) ‹*Inf c* ∈ *c*›, *of Inf c* − *1*] **by** *auto*
then have (*Inf c* + *b*)/*2* ∈ *c* **by** *auto*
moreover have (*Inf c* + *b*)/*2* < *Inf c* **using** ‹*b* < *Inf c*› **by** *simp*
ultimately show *False* **using** *bb cInf-lower leD* **by** *blast*
qed
have *SupnotIn*: *Sup c* ∉ *c*
proof
assume *Sup c* ∈ *c*
then obtain *b* **where** *Sup c* < *b* {*Sup c*..<*b*} ⊆ *c*
using *open-right*[*OF assms*(*1*) ‹*Sup c* ∈ *c*›, *of Sup c* + *1*] **by** *auto*
then have (*Sup c* + *b*)/*2* ∈ *c* **by** *auto*
moreover have *Sup c* < (*Sup c* + *b*)/*2* **using** ‹*Sup c* < *b*› **by** *simp*
ultimately show *False* **using** *ba cSup-upper leD* **by** *blast*
qed
show *?thesis*
proof (*intro set-eqI iffI*)
fix *x* **assume** *x* ∈ *c*
then have *Inf c* ≤ *x x* ≤ *Sup c* **using** *bb ba cInf-lower cSup-upper* **by** *auto*
moreover have *x* ≠ *Inf c x* ≠ *Sup c* **using** ‹*x* ∈ *c*› *InfnotIn SupnotIn* **by** *auto*
ultimately show *x* ∈ {*Inf c*<..<*Sup c*} **by** *auto*
next
fix *x* **assume** *x*: *x* ∈ {*Inf c*<..<*Sup c*}
obtain *u* **where** *u*: *u* ∈ *c u* < *x* **using** *x assms*(*3*) *bb*
by (*metis cInf-lessD greaterThanLessThan-iff*)
obtain *v* **where** *v*: *v* ∈ *c x* < *v* **using** *x assms*(*3*) *ba*
by (*metis less-cSupD greaterThanLessThan-iff*)
show *x* ∈ *c* **using** *isiv u v* **by** (*meson is-interval-1 less-imp-le*)
qed
qed

The reduced convexification subgoal: a unit-speed (Lipschitz) simple closed rectifiable loop starting on the frontier of its convex hull can be replaced by a no-longer loop with the same convex hull whose image IS that

frontier. The components of the part of the parameter interval whose image avoids the frontier are countably many open intervals.

proposition *convexification-unit-speed*:
fixes γ :: *real* $\Rightarrow$ *complex*
assumes *rect*: *rectifiable-path* γ **and** *simp*: *simple-path* γ **and** *loop*: *pathfinish* γ = *pathstart* γ
and *frstart*: *pathstart* $\gamma \in$ *frontier* (*convex hull* (*path-image* γ))
and *lip*: $\bigwedge x\ y.\ x \in \{0..1\} \Longrightarrow y \in \{0..1\} \Longrightarrow$ *dist* (γ x) (γ y) $\leq$ *path-length* γ $*$ *dist* x y
shows $\exists h.$ *rectifiable-path* $h \wedge$ *simple-path* $h \wedge$ *pathfinish* h = *pathstart* $h \wedge$ *path-length* $h \leq$ *path-length* $\gamma \wedge$ *convex hull* (*path-image* h) = *convex hull* (*path-image* γ) $\wedge$ *path-image* h = *frontier* (*convex hull* (*path-image* γ))
proof (*cases path-image* $\gamma \subseteq$ *frontier* (*convex hull* (*path-image* γ)))
case *True*
— Already on the frontier: γ itself works (its image equals the frontier).
have *path-image* γ = *frontier* (*convex hull* (*path-image* γ))
using *frontier-convex-hull-subset-path-image*[*OF simp loop True*] *True* **by** *blast*
then show *?thesis* **using** *rect simp loop* **by** (*intro exI*[*of - γ*]) *auto*
next
case *False*
have *pathγ*: *path* γ **using** *simp* **by** (*rule simple-path-imp-path*)
define F **where** F = *frontier* (*convex hull* (*path-image* γ))
define s **where** $s = \{t \in \{0..1\}.\ \gamma\ t \notin F\}$
have *contγ*: *continuous-on* $\{0..1\}$ γ **using** *pathγ* **by** (*simp add*: *path-def*)
have *g0F*: $\gamma\ 0 \in F$ **using** *frstart F-def* **by** (*simp add*: *pathstart-def*)
have *g1F*: $\gamma\ 1 \in F$ **using** *frstart loop F-def* **by** (*simp add*: *pathstart-def pathfinish-def*)
— Since both endpoints map into F, the deviating set s avoids 0 and 1, hence is open in $\mathbb{R}$.
have *s-sub*: $s \subseteq \{0<..<1\}$
proof
fix t **assume** $t \in s$
then have *bounds*: $0 \leq t\ t \leq 1$ **and** *outside*: $\gamma\ t \notin F$
unfolding *s-def* **by** *auto*
have $t \neq 0\ t \neq 1$ **using** *outside g0F g1F* **by** *auto*
then show $t \in \{0<..<1\}$ **using** *bounds* **by** *auto*
qed
have *openin* (*top-of-set* $\{0..1\}$) ($\{0..1\} \cap \gamma -$ ‘ ($-$ F))
using *contγ* **by** (*intro continuous-openin-preimage-gen*) (*auto simp*: *F-def*)
then have *s-openin*: *openin* (*top-of-set* $\{0..1\}$) s
by (*smt* (*verit*) *Collect-cong Compl-iff Int-def s-def vimage-eq*)
then have *openin* (*top-of-set* $\{0<..<1\}$) s
by (*metis interior-atLeastAtMost-real interior-subset openin-subset-trans s-sub*)
then have *opens*: *open* s
by (*simp add*: *openin-open-trans*)
have *s-ne*: $s \neq \{\}$
using *False* **by** (*auto simp*: *s-def F-def path-image-def*)
— Component decomposition: the components of s are countably many open intervals $\{a\ n<..<b\ n\}$, each with both endpoints mapped by γ onto the frontier

F.
have *decomp*: $\exists\, a\ b{::}nat{\Rightarrow}real.\ (\forall\, n.\ a\ n \in \{0..1\}) \wedge (\forall\, n.\ b\ n \in \{0..1\}) \wedge (\forall\, n.\ a\ n \leq b\ n) \wedge (\forall\, n.\ \gamma\ (a\ n) \in F) \wedge (\forall\, n.\ \gamma\ (b\ n) \in F) \wedge components\ s = \{\{a\ n{<}..{<}b\ n\} \mid n.\ n \in (UNIV{::}nat\ set)\}$
proof −
have *comp-open*: $\bigwedge c.\ c \in components\ s \Longrightarrow open\ c$ **using** *opens* **by** (*rule open-components*)
have *comp-disj*: *disjoint* (*components s*)
using *pairwise-disjoint-components* **by** (*simp add*: *disjoint-def pairwise-def disjnt-def*)
have *comp-count*: *countable* (*components s*)
using *comp-open comp-disj* **by** (*rule countable-disjoint-open-subsets*)
have *comp-ne*: *components s* $\neq \{\}$ **using** *s-ne* **by** (*simp add*: *components-eq-empty*)
define *q* **where** *q* = *from-nat-into* (*components s*)
have *q-range*: *range q* = *components s*
unfolding *q-def* **using** *comp-ne comp-count* **by** (*rule range-from-nat-into*)
have *q-comp*: $\bigwedge n.\ q\ n \in components\ s$ **using** *q-range* **by** *auto*
have *s-sub01*: *s* $\subseteq \{0..1\}$ **using** *s-sub* **by** *auto*
have *s-bdd*: *bounded s* **using** *s-sub01 bounded-closed-interval bounded-subset* **by** *blast*
have *qsub*: $\bigwedge n.\ q\ n \subseteq s$ **using** *q-comp in-components-subset* **by** *blast*
have *q-interval*: *q n* = $\{Inf\ (q\ n){<}..{<}Sup\ (q\ n)\}$ **for** *n*
proof −
have *bounded* (*q n*) **using** *qsub s-bdd bounded-subset* **by** *blast*
then show *q n* = $\{Inf\ (q\ n){<}..{<}Sup\ (q\ n)\}$
by (*metis comp-open in-components-maximal open-bounded-connected-real-is-interval q-comp*)
qed
define *a* **where** *a* = $(\lambda n.\ Inf\ (q\ n))$
define *b* **where** *b* = $(\lambda n.\ Sup\ (q\ n))$
have *qab*: $\bigwedge n.\ q\ n = \{a\ n{<}..{<}b\ n\}$ **using** *q-interval* **by** (*simp add*: *a-def b-def*)
have *ablt*: *a n* < *b n* **for** *n*
by (*metis greaterThanLessThan-empty-iff in-components-maximal linorder-not-le q-comp qab*)
have *clq*: $\bigwedge n.\ closure\ (q\ n) = \{a\ n..b\ n\}$ **using** *qab ablt* **by** (*simp add*: *closure-greaterThanLessThan*)
have *cls01*: *closure s* $\subseteq \{0..1\}$
using *s-sub closure-mono*[*of s* $\{0{<}..{<}1\}$] *closure-greaterThanLessThan*[*of* 0::*real 1*] **by** *simp*
have *ab01*: *a n* $\in \{0..1\} \wedge$ *b n* $\in \{0..1\}$ **for** *n*
proof −
have *sub*: $\{a\ n..b\ n\} \subseteq \{0..1\}$
by (*metis closure-mono clq cls01 qsub subset-trans*)
then show *a n* $\in \{0..1\} \wedge$ *b n* $\in \{0..1\}$
by (*meson ablt atLeastAtMost-iff less-eq-real-def subset-eq*)
qed
have *a-notin*: $\bigwedge n.\ a\ n \notin s$
proof (*rule ccontr*)
fix *n* **assume** ¬ *a n* $\notin$ *s*

then have *sub*: $\{a\ n..<b\ n\} \subseteq s$
using *less-eq-real-def qab qsub* **by** *fastforce*
then have $\{a\ n..<b\ n\} = q\ n$
using *q-comp*[*of n*] *in-components-maximal*
by (*metis connected-Ico interior-atLeastLessThan interior-subset qab subset-empty*)
then show *False* **using** *ablt*[*of n*] *qab*
by (*metis atLeastLessThan-iff greaterThanLessThan-iff less-irrefl order-refl*)
qed
have *b-notin*: $\bigwedge n.\ b\ n \notin s$
proof (*rule ccontr*)
fix *n* **assume** $\neg\ b\ n \notin s$
then have *sub*: $\{a\ n<..b\ n\} \subseteq s$
using *less-eq-real-def qab qsub* **by** *fastforce*
then have $\{a\ n<..b\ n\} = q\ n$
using *q-comp*[*of n*] *in-components-maximal*
by (*metis connected-Ioc interior-lessThanAtMost interior-subset qab subset-empty*)
then show *False* **using** *ablt*[*of n*] *qab*
by (*metis greaterThanAtMost-iff greaterThanLessThan-iff less-irrefl order-refl*)
qed
have *gaF*: $\bigwedge n.\ \gamma\ (a\ n) \in F$ **and** *gbF*: $\bigwedge n.\ \gamma\ (b\ n) \in F$
using *ab01 a-notin b-notin s-def* **by** *blast+*
have *comp-eq*: *components* $s = \{\{a\ n<..<b\ n\} \mid n.\ n \in (UNIV::nat\ set)\}$
using *full-SetCompr-eq q-range qab* **by** *force*
show $\exists\, a\ b::nat \Rightarrow real.\ (\forall\, n.\ a\ n \in \{0..1\}) \wedge (\forall\, n.\ b\ n \in \{0..1\}) \wedge (\forall\, n.\ a\ n \leq b\ n) \wedge (\forall\, n.\ \gamma\ (a\ n) \in F) \wedge (\forall\, n.\ \gamma\ (b\ n) \in F) \wedge components\ s = \{\{a\ n<..<b\ n\} \mid n.\ n \in (UNIV::nat\ set)\}$
using *ab01 ablt gaF gbF comp-eq* **by** (*intro exI*[*of - a*] *exI*[*of - b*]) (*auto simp*: *less-imp-le*)
qed
— Extract the deviating arcs from the decomposition.
from *decomp* **obtain** *a b* :: *nat* ⇒ *real* **where**
ab01: $\bigwedge n.\ a\ n \in \{0..1\}\ \bigwedge n.\ b\ n \in \{0..1\}$ **and**
able: $\bigwedge n.\ a\ n \leq b\ n$ **and**
gaF: $\bigwedge n.\ \gamma\ (a\ n) \in F$ **and** *gbF*: $\bigwedge n.\ \gamma\ (b\ n) \in F$ **and**
comps: $components\ s = \{\{a\ n<..<b\ n\} \mid n.\ n \in (UNIV::nat\ set)\}$ **by** *auto*
— $U\ n$ collects the first n deviating arcs; $P\ n\ h$ is the invariant for the n-th approximation: a Lipschitz simple closed loop with the same convex hull, mapping the first n arcs onto the frontier and equal to γ elsewhere.
define *U* **where** $U = (\lambda n::nat.\ \bigcup\ \{\{a\ m<..<b\ m\} \mid m.\ m < n\})$
define *P* **where** $P = (\lambda n\ h.\ simple\text{-}path\ h \wedge rectifiable\text{-}path\ h \wedge pathstart\ h = pathstart\ \gamma \wedge pathfinish\ h = pathfinish\ \gamma \wedge convex\ hull\ (path\text{-}image\ h) = convex\ hull\ (path\text{-}image\ \gamma) \wedge (\forall\, x{\in}\{0..1\}.\ \forall\, y{\in}\{0..1\}.\ dist\ (h\ x)\ (h\ y) \leq path\text{-}length\ \gamma * dist\ x\ y) \wedge (\forall\, x{\in}U\ n.\ h\ x \in F) \wedge (\forall\, x.\ x \notin U\ n \longrightarrow h\ x = \gamma\ x))$
have *arc-comp*: $\bigwedge n.\ \{a\ n<..<b\ n\} \in components\ s$ **using** *comps* **by** *auto*
have *arc-in-s*: $\bigwedge n.\ \{a\ n<..<b\ n\} \subseteq s$ **using** *arc-comp in-components-subset* **by** *blast*

have *USuc*: $\bigwedge n.\ U\ (Suc\ n) = U\ n \cup \{a\ n{<}..{<}b\ n\}$
by (*auto simp*: *U-def less-Suc-eq*)
have *U-sub-s*: $\bigwedge n.\ U\ n \subseteq s$
using *U-def arc-in-s* **by** *blast*
have *an-notin-U*: $\bigwedge n.\ a\ n \notin U\ n$ **and** *bn-notin-U*: $\bigwedge n.\ b\ n \notin U\ n$
using *U-sub-s gaF gbF s-def* **by** *blast+*
have *arc-sub-U*: $\bigwedge i\ j{::}nat.\ i < j \Longrightarrow \{a\ i{<}..{<}b\ i\} \subseteq U\ j$
by (*auto simp*: *U-def*)
have *U-mem*: $\bigwedge x\ n.\ x \in U\ n \Longrightarrow \exists i{<}n.\ x \in \{a\ i{<}..{<}b\ i\}$
unfolding *U-def* **by** *blast*
have *arc-disj*: $\bigwedge i\ j.\ \{a\ i{<}..{<}b\ i\} \neq \{a\ j{<}..{<}b\ j\} \Longrightarrow \{a\ i{<}..{<}b\ i\} \cap \{a\ j{<}..{<}b\ j\} = \{\}$
using *arc-comp components-nonoverlap* **by** *blast*
have *arc-disj-U*: $\bigwedge n.\ \neg\ \{a\ n{<}..{<}b\ n\} \subseteq U\ n \Longrightarrow \{a\ n{<}..{<}b\ n\} \cap U\ n = \{\}$
using *U-mem arc-disj arc-sub-U* **by** *blast*
have *F-eq*: $F =$ *frontier* (*convex hull* (*path-image* γ)) **by** (*simp add*: *F-def*)
— Inductive step: straighten the n-th arc with *step-lemma* (unless it is empty or already handled), preserving the invariant.
have *step*: $\bigwedge n\ h.\ P\ n\ h \Longrightarrow \exists h'.\ P\ (Suc\ n)\ h' \wedge (\forall x.\ \neg(x \in \{a\ n{<}..{<}b\ n\} \wedge x \notin U\ n) \longrightarrow h'\ x = h\ x)$
proof −
fix $n\ h$ **assume** *Ph*: $P\ n\ h$
have *hsimple*: *simple-path h* **and** *hrect*: *rectifiable-path h*
and *hps*: *pathstart h* = *pathstart* γ **and** *hpf*: *pathfinish h* = *pathfinish* γ
and *hhull*: *convex hull* (*path-image h*) = *convex hull* (*path-image* γ)
and *hlip*: $\bigwedge x\ y.\ x \in \{0..1\} \Longrightarrow y \in \{0..1\} \Longrightarrow dist\ (h\ x)\ (h\ y) \leq$ *path-length* $\gamma * dist\ x\ y$
and *hUF*: $\bigwedge x.\ x \in U\ n \Longrightarrow h\ x \in F$
and *hoff*: $\bigwedge x.\ x \notin U\ n \Longrightarrow h\ x = \gamma\ x$
using *Ph* **unfolding** *P-def* **by** *auto*
have *hloop*: *pathfinish h* = *pathstart h* **using** *hps hpf loop* **by** *simp*
have *hF*: *frontier* (*convex hull* (*path-image h*)) = *F* **using** *hhull F-eq* **by** *simp*
show $\exists h'.\ P\ (Suc\ n)\ h' \wedge (\forall x.\ \neg(x \in \{a\ n{<}..{<}b\ n\} \wedge x \notin U\ n) \longrightarrow h'\ x = h\ x)$
proof (*cases* $\{a\ n{<}..{<}b\ n\} = \{\} \vee \{a\ n{<}..{<}b\ n\} \subseteq U\ n$)
case *True* **with** *USuc Ph* **show** *?thesis*
by (*intro exI*[*of* - *h*]) (*auto simp*: *P-def*)
next
case *False*
then have *ablt-n*: $a\ n < b\ n$ **and** *arc-disj-Un*: $\{a\ n{<}..{<}b\ n\} \cap U\ n = \{\}$
using *arc-disj-U* **by** *auto*
have *ha*: $h\ (a\ n) = \gamma\ (a\ n)$ **and** *hb*: $h\ (b\ n) = \gamma\ (b\ n)$
using *hoff an-notin-U bn-notin-U* **by** *auto*
have *harc*: $\bigwedge x.\ x \in \{a\ n{<}..{<}b\ n\} \Longrightarrow h\ x = \gamma\ x$ **using** *hoff arc-disj-Un* **by** *blast*
have *haF*: $h\ (a\ n) \in$ *frontier* (*convex hull* (*path-image h*)) **using** *ha gaF hF* **by** *simp*
have *hbF*: $h\ (b\ n) \in$ *frontier* (*convex hull* (*path-image h*)) **using** *hb gbF hF* **by** *simp*

have *harc-offF*: h ‘ $\{a\ n{<}..{<}b\ n\} \cap$ *frontier* (*convex hull* (*path-image* h)) = {}
using *arc-in-s s-def hF harc* **by** *fastforce*
obtain h' **where** *h'simple*: *simple-path* h'
and *h'p*: *pathstart* h' = *pathstart h pathfinish* h' = *pathstart h*
and *h'lip*: $\bigwedge x\ y.\ x \in \{0..1\} \Longrightarrow y \in \{0..1\} \Longrightarrow$ *dist* (h' x) (h' y) $\leq$ *path-length* γ $*$ *dist x y*
and *h'hull*: *convex hull* (*path-image* h') = *convex hull* (*path-image h*)
and *h'off*: $\bigwedge x.\ x \notin \{a\ n{<}..{<}b\ n\} \Longrightarrow h'\ x = h\ x$
and *h'arcF*: h' ‘ $\{a\ n..b\ n\} \subseteq$ *frontier* (*convex hull* (*path-image h*))
using *step-lemma*[*OF hsimple hloop hlip ablt-n ab01*(*1*)[*of n*] *ab01*(*2*)[*of n*] *haF hbF harc-offF*]
by *metis*
have *h'rect*: *rectifiable-path* h'
by (*metis lipschitz-imp-rectifiable-path dist-norm h'lip*)
have *h'agree*: $\bigwedge x.\ \neg(x \in \{a\ n{<}..{<}b\ n\} \wedge x \notin U\ n) \Longrightarrow h'\ x = h\ x$
by (*meson arc-disj-Un disjoint-iff h'off*)
have *h'UF*: $\bigwedge x.\ x \in U\ (Suc\ n) \Longrightarrow h'\ x \in F$
using *F-eq USuc h'agree h'arcF hUF hhull* **by** *fastforce*
have *h'offSuc*: $h'\ x = \gamma\ x$ **if** $x \notin U\ (Suc\ n)$ **for** x
using *USuc that h'off hoff* **by** *force*
have *PSuc*: $P\ (Suc\ n)\ h'$
using *P-def Ph h'UF h'hull h'lip h'offSuc h'p h'rect h'simple hloop* **by** *presburger*
show *?thesis* **using** *PSuc h'agree* **by** *blast*
qed
qed
— Dependent choice yields the sequence of approximations *f*.
have *base*: $P\ 0\ \gamma$
using *simp rect lip* **by** (*simp add*: *P-def U-def*)
obtain f **where** f: $\bigwedge n.\ P\ n\ (f\ n)$
and *fstep*: $\bigwedge n\ x.\ \neg(x \in \{a\ n{<}..{<}b\ n\} \wedge x \notin U\ n) \Longrightarrow f\ (Suc\ n)\ x = f\ n\ x$
using *dependent-nat-choice*[**where** $P{=}P$ **and** $Q{=}\lambda n\ h\ h'.\ \forall x.\ \neg(x \in \{a\ n{<}..{<}b\ n\} \wedge x \notin U\ n) \longrightarrow h'\ x = h\ x$]
base step **by** *blast*
have *evconst*: $\exists y.\ \forall_F\ n$ *in sequentially*. $f\ n\ x = y$ **if** *x01*: $x \in \{0..1\}$ **for** x
proof (*cases* $\exists n.\ x \in \{a\ n{<}..{<}b\ n\}$)
case *False*
have *stab*: $f\ n\ x = f\ 0\ x$ **for** n
by (*induct n*; *use fstep False* **in** *presburger*)
then show *?thesis*
by (*meson eventually-at-top-dense*)
next
case *True*
define N **where** N = (*LEAST n*. $x \in \{a\ n{<}..{<}b\ n\}$)
have *xN*: $x \in \{a\ N{<}..{<}b\ N\}$ **using** *True LeastI-ex N-def* **by** (*metis* (*mono-tags*, *lifting*))
have *stab2*: $\bigwedge m.\ N < m \Longrightarrow f\ (Suc\ m)\ x = f\ m\ x$
using *arc-sub-U fstep xN* **by** *blast*

have $f\ (Suc\ N + d)\ x = f\ (Suc\ N)\ x$ **for** d
by (*induct d*; *use stab2* **in** *fastforce*)
then have *eventually* $(\lambda n.\ f\ n\ x = f\ (Suc\ N)\ x)$ *sequentially*
by (*metis* (*mono-tags*, *lifting*) *Suc-leI eventually-at-top-dense nat-le-iff-add*)
then show *?thesis* **by** *blast*
qed
— Skolemize to obtain the limit path h: $f\ n\ x = h\ x$ eventually, for each x.
obtain h **where** h: $\bigwedge x.\ x \in \{0..1\} \Longrightarrow$ *eventually* $(\lambda n.\ f\ n\ x = h\ x)$ *sequentially*
using *evconst* **by** (*metis* (*mono-tags*))
— Properties of the approximants $f\ n$, extracted from the invariant P.
have *fsimple*: $\bigwedge n.$ *simple-path* $(f\ n)$ **using** f **unfolding** *P-def* **by** *blast*
have *fps*: $\bigwedge n.$ *pathstart* $(f\ n) =$ *pathstart* γ **using** f **unfolding** *P-def* **by** *blast*
have *fpf*: $\bigwedge n.$ *pathfinish* $(f\ n) =$ *pathfinish* γ **using** f **unfolding** *P-def* **by** *blast*
have *fhull*: $\bigwedge n.$ *convex hull* (*path-image* $(f\ n)$) = *convex hull* (*path-image* γ) **using** f **unfolding** *P-def* **by** *blast*
have *flip*: $\bigwedge n\ x\ y.\ x \in \{0..1\} \Longrightarrow y \in \{0..1\} \Longrightarrow$ *dist* $(f\ n\ x)\ (f\ n\ y) \leq$ *path-length* $\gamma *$ *dist* $x\ y$
using f **unfolding** *P-def* **by** *blast*
have *fUF*: $\bigwedge n\ x.\ x \in U\ n \Longrightarrow f\ n\ x \in F$ **using** f **unfolding** *P-def* **by** *blast*
have *foff*: $\bigwedge n\ x.\ x \notin U\ n \Longrightarrow f\ n\ x = \gamma\ x$ **using** f **unfolding** *P-def* **by** *blast*
— The limit path h inherits the L-Lipschitz bound (pointwise limit of L-Lipschitz maps).
have *hlip*: *dist* $(h\ x)\ (h\ y) \leq$ *path-length* $\gamma *$ *dist* $x\ y$
if *xy*: $x \in \{0..1\}\ y \in \{0..1\}$ **for** $x\ y$
proof −
have *ev*: *eventually* $(\lambda n.\ f\ n\ x = h\ x \wedge f\ n\ y = h\ y)$ *sequentially*
using *h*[*OF xy*(*1*)] *h*[*OF xy*(*2*)] **by** *eventually-elim simp*
then obtain n **where** n: $f\ n\ x = h\ x\ f\ n\ y = h\ y$
unfolding *eventually-sequentially* **by** *auto*
then show *dist* $(h\ x)\ (h\ y) \leq$ *path-length* $\gamma *$ *dist* $x\ y$
using *n flip*[*OF xy*] **by** *metis*
qed
have *hrect*: *rectifiable-path h*
by (*rule lipschitz-imp-rectifiable-path*[**where** *B*=*path-length* γ])
(*use hlip* **in** ‹*simp add*: *dist-norm*›)
have *hpath*: *path h* **using** *hrect* **by** (*rule rectifiable-path-imp-path*)
have *hsimple*: *simple-path h*
unfolding *simple-path-def loop-free-def*
proof (*intro conjI strip*)
show *path h* **by** (*rule hpath*)
fix $x\ y$ :: *real* **assume** *xy*: $x \in \{0..1\}\ y \in \{0..1\}$ **and** *eq*: $h\ x = h\ y$
then have *ev*: *eventually* $(\lambda n.\ f\ n\ x = h\ x \wedge f\ n\ y = h\ y)$ *sequentially*
by (*metis* (*full-types*) *eventually-conj-iff h*)
then obtain n **where** n: $f\ n\ x = h\ x\ f\ n\ y = h\ y$
unfolding *eventually-sequentially* **by** *auto*
then show $x = y \vee x = 0 \wedge y = 1 \vee x = 1 \wedge y = 0$
using *n eq fsimple*[*of n*] *xy* **unfolding** *simple-path-def loop-free-def* **by** *presburger*
qed

have *hloop*: *pathfinish h* = *pathstart h*
proof −
have *z01*: (*0*::*real*) ∈ {*0..1*} **and** *o01*: (*1*::*real*) ∈ {*0..1*} **by** *auto*
then obtain *n* **where** *n*: *f n 0* = *h 0 f n 1* = *h 1*
using *h*[*OF z01*] *h*[*OF o01*] *fpf* **by** (*auto simp*: *path-defs eventually-sequentially*)
have *f n 1* = *f n 0* **using** *fps*[*of n*] *fpf*[*of n*] *loop* **by** (*simp add*: *pathstart-def pathfinish-def*)
with *n* **show** *?thesis* **by** (*simp add*: *pathstart-def pathfinish-def*)
qed
have *hlen*: *path-length h* ≤ *path-length* γ
by (*metis Rectifiable-Path.path-length-lipschitz dist-norm hlip*)
— The image of *h* lies on the frontier: each parameter either lands in an arc (so eventually in *F*) or maps via γ to a point already on the frontier.
have *notarc-notin-s*: *False* **if** *noarc*: ∄*n*. *x* ∈ {*a n*<..<*b n*} *x* ∈ *s* **for** *x*
proof −
from *that Union-components*
obtain *c* **where** *c* ∈ *components s x* ∈ *c* **by** *blast*
then show *False* **using** *comps noarc* ‹*x* ∈ *c*› **by** *blast*
qed
have *hx-in-F*: ⋀*x*. *x* ∈ {*0..1*} ⟹ *h x* ∈ *F*
proof −
fix *x* :: *real* **assume** *x01*: *x* ∈ {*0..1*}
show *h x* ∈ *F*
proof (*cases* ∃*n*. *x* ∈ {*a n*<..<*b n*})
case *True*
then obtain *n* **where** *xn*: *x* ∈ {*a n*<..<*b n*} **by** *blast*
obtain *N* **where** *N*: ⋀*m*. *N* ≤ *m* ⟹ *f m x* = *h x* **using** *h*[*OF x01*] **unfolding** *eventually-sequentially* **by** *blast*
define *m* **where** *m* = *max N* (*Suc n*)
have *N* ≤ *m n* < *m* **by** (*auto simp*: *m-def*)
then show *h x* ∈ *F* **using** *N fUF* ‹*N* ≤ *m*›
using *arc-sub-U xn* **by** *fastforce*
next
case *False*
obtain *n* **where** *n*: *f n x* = *h x* **using** *h*[*OF x01*] **unfolding** *eventually-sequentially* **by** *blast*
have *x* ∉ *U n* **using** *False U-mem* **by** *blast*
moreover have γ *x* ∈ *F* **using** *x01 s-def*
using *False notarc-notin-s* **by** *blast*
ultimately show *h x* ∈ *F* **using** *n*
by (*simp add*: *foff*)
qed
qed
have *hsub-F*: *path-image h* ⊆ *F* **using** *hx-in-F* **by** (*auto simp*: *path-image-def*)
— Off the arcs, *h* still agrees with γ; so γ's image outside the deviating set *s* is part of the image of *h*.
have *hoff-s*: *h x* = γ *x* **if** *x01*: *x* ∈ {*0..1*} **and** *xs*: *x* ∉ *s* **for** *x*
by (*metis* (*mono-tags*, *lifting*) *U-sub-s eventually-sequentially foff h le-refl subset-eq*

that)
have *gout-sub-h*: γ ‘ ($\{0..1\} - s$) $\subseteq$ *path-image h*
using *hoff-s* **by** (*force simp*: *path-image-def*)
— The convex hulls agree.
have *hhull*: *convex hull* (*path-image h*) = *convex hull* (*path-image* γ)
proof (*rule subset-antisym*)
— $\subseteq$: every point $h\,x = f\,n\,x$ for some n lies in *convex hull path-image* (f n) = *convex hull path-image* γ.
have *ph-sub*: *path-image h* $\subseteq$ *convex hull* (*path-image* γ)
proof
fix z **assume** $z \in$ *path-image h*
then obtain x **where** x: $x \in \{0..1\}$ $z = h\ x$ **by** (*auto simp*: *path-image-def*)
obtain n **where** n: $f\ n\ x = h\ x$ **using** h[*OF x*(*1*)] **unfolding** *eventually-sequentially* **by** *blast*
have $h\ x = f\ n\ x$ **using** n **by** *simp*
then have $h\ x \in$ *convex hull* (*path-image* (f n)) **using** *hull-subset*
by (*metis hull-inc imageI path-image-def x*(*1*))
then show $z \in$ *convex hull* (*path-image* γ)
by (*simp add*: *fhull x*(*2*))
qed
show *convex hull* (*path-image h*) $\subseteq$ *convex hull* (*path-image* γ)
using *ph-sub convex-convex-hull* **by** (*rule hull-minimal*)
next
— $\supseteq$: the extreme points of the hull are frontier points, hence images of parameters outside s, which lie in *path-image h*. (Krein–Milman + redundancy of interior points.)
have *cpt-g*: *compact* (*convex hull path-image* γ)
by (*simp add*: *compact-convex-hull compact-path-image path*γ)
have *km-g*: *convex hull path-image* γ = *convex hull* $\{x.\ x$ *extreme-point-of* (*convex hull path-image* γ)$\}$
using *Krein-Milman-Minkowski*[*OF cpt-g convex-convex-hull*] **by** *simp*
have *ext-in-out*: $z \in \gamma$ ‘ ($\{0..1\} - s$)
if z *extreme-point-of* (*convex hull path-image* γ) **for** z
proof −
have *zpig*: $z \in$ *path-image* γ **using** *extreme-point-of-convex-hull*[*OF that*] **.**
have *znotint*: $z \notin$ *interior* (*convex hull path-image* γ)
using *extreme-point-not-in-interior*[*OF that*] **.**
have $z \in F$
using *F-def closure-subset frontier-def hull-inc znotint zpig* **by** *fastforce*
then show *?thesis* **using** *zpig*
by (*auto simp*: *path-image-def s-def*)
qed
have $\{x.\ x$ *extreme-point-of* (*convex hull path-image* γ)$\} \subseteq \gamma$ ‘ ($\{0..1\} - s$)
using *ext-in-out* **by** *blast*
then have *convex hull* (*path-image* γ) $\subseteq$ *convex hull* (γ ‘ ($\{0..1\} - s$))
using *hull-mono km-g* **by** *blast*
also have *convex hull* (γ ‘ ($\{0..1\} - s$)) $\subseteq$ *convex hull* (*path-image h*)
using *gout-sub-h* **by** (*rule hull-mono*)
finally show *convex hull* (*path-image* γ) $\subseteq$ *convex hull* (*path-image h*) **.**

qed
— The image of h is exactly the frontier: $\subseteq$ is *hsub-F*; $\supseteq$ because h is a simple closed curve whose image lies on the frontier of its (now equal) convex hull.
have *hF*: *frontier* (*convex hull* (*path-image h*)) = *F* **using** *hhull F-eq* **by** *simp*
have *h-image-F*: *path-image h* = *F*
using *frontier-convex-hull-subset-path-image hF hloop hsimple hsub-F* **by** *blast*
show *?thesis*
using *F-def h-image-F hhull hlen hloop hrect hsimple* **by** *blast*
qed

theorem *isoperimetric-convexification*:
fixes *g* :: *real* ⇒ *complex*
assumes *rectifiable-path g simple-path g*
pathfinish g = *pathstart g*
obtains *h* **where** *rectifiable-path h* **and** *simple-path h*
and *pathfinish h* = *pathstart h*
and *path-length h* ≤ *path-length g*
and *convex hull* (*path-image h*) = *convex hull* (*path-image g*)
and *path-image h* = *frontier* (*convex hull* (*path-image g*))
proof −
— Strengthened version, assuming the loop starts on the frontier of its convex hull. (Here used to derive the general statement by shifting the basepoint.)
have ∗: ∃ *h*. *rectifiable-path h* ∧ *simple-path h* ∧ *pathfinish h* = *pathstart h* ∧ *path-length h* ≤ *path-length G* ∧ *convex hull* (*path-image h*) = *convex hull* (*path-image G*) ∧ *path-image h* = *frontier* (*convex hull* (*path-image G*))
if *Grect*: *rectifiable-path G* **and** *Gsimple*: *simple-path G* **and** *Gloop*: *pathfinish G* = *pathstart G*
and *Gfr*: *pathstart G* ∈ *frontier* (*convex hull* (*path-image G*)) **for** *G* :: *real* ⇒ *complex*
using *arc-length-reparametrization*[*OF Grect*] *convexification-unit-speed Gsimple Gloop Gfr*
by (*metis* (*no-types*, *lifting*) *ext*)
— Some point of the loop lies on the frontier (an extreme point of the convex hull).
have *cpt*: *compact* (*convex hull path-image g*)
by (*simp add*: *assms*(*2*) *compact-convex-hull compact-simple-path-image*)
obtain *x* **where** *x-ext*: *x extreme-point-of* (*convex hull path-image g*)
using *Krein-Milman-Minkowski cpt* **by** *fastforce*
have *x-fr*: *x* ∈ *frontier* (*convex hull path-image g*)
by (*metis Krein-Milman-frontier convex-convex-hull cpt extreme-point-of-convex-hull x-ext*)
obtain *t* **where** *t*: *t* ∈ {*0..1*} *g t* = *x*
using *extreme-point-of-convex-hull*[*OF x-ext*] **by** (*auto simp*: *path-image-def*)
— Shift *g* so that it starts at the frontier point *g t*; properties are preserved.
have *sp-rect*: *rectifiable-path* (*shiftpath t g*) **using** *assms*(*1*,*3*) *t*(*1*) **by** (*rule rectifiable-path-shiftpath*)
have *sp-simple*: *simple-path* (*shiftpath t g*) **using** *assms*(*2*,*3*) *t*(*1*) **by** (*simp add*: *simple-path-shiftpath*)
have *sp-loop*: *pathfinish* (*shiftpath t g*) = *pathstart* (*shiftpath t g*) **using** *assms*(*3*)

t(*1*) **by** (*rule closed-shiftpath*)
have *sp-pi*: *path-image* (*shiftpath t g*) = *path-image g* **using** *t*(*1*) *assms*(*3*) **by** (*rule path-image-shiftpath*)
have *sp-start*: *pathstart* (*shiftpath t g*) = *g t* **using** *t*(*1*) **by** (*simp add*: *path-start-shiftpath*)
have *sp-len*: *path-length* (*shiftpath t g*) = *path-length g* **using** *assms*(*1*,*3*) *t*(*1*) **by** (*rule path-length-shiftpath*)
have *sp-startfr*: *pathstart* (*shiftpath t g*) ∈ *frontier* (*convex hull* (*path-image* (*shiftpath t g*)))
using *sp-start sp-pi x-fr t*(*2*) **by** *simp*
show *?thesis*
using ∗[*OF sp-rect sp-simple sp-loop sp-startfr*] *sp-len sp-pi that* **by** *force*
qed

theorem *isoperimetric-convexification-strict*:
fixes *g* :: *real* ⇒ *complex*
assumes *rectifiable-path g simple-path g*
pathfinish g = *pathstart g*
¬ *convex* (*inside* (*path-image g*))
obtains *h* **where** *rectifiable-path h* **and** *simple-path h*
and *pathfinish h* = *pathstart h*
and *path-length h* ≤ *path-length g*
and *convex hull* (*path-image h*) = *convex hull* (*path-image g*)
and *path-image h* = *frontier* (*convex hull* (*path-image g*))
and *measure lebesgue* (*inside* (*path-image g*)) < *measure lebesgue* (*inside* (*path-image h*))
proof −
obtain *h* **where** *h*: *rectifiable-path h simple-path h pathfinish h* = *pathstart h*
path-length h ≤ *path-length g convex hull* (*path-image h*) = *convex hull* (*path-image g*)
path-image h = *frontier* (*convex hull* (*path-image g*))
using *isoperimetric-convexification*[*OF assms*(*1*,*2*,*3*)] **by** *metis*
have *bdd-hull*: *bounded* (*convex hull* (*path-image g*))
by (*simp add*: *bounded-convex-hull bounded-simple-path-image assms*(*2*))
have *ins-g-sub*: *inside* (*path-image g*) ⊆ *interior* (*convex hull* (*path-image g*))
proof −
let *?C* = *convex hull* (*path-image g*)
have *conn*: *connected* (− *?C*) **using** *bdd-hull* **by** (*simp add*: *connected-complement-bounded-convex*)
have *unbdd*: ¬ *bounded* (− *?C*)
using *bdd-hull cobounded-imp-unbounded* **by** *blast*
have *un*: (*?C* − *path-image g*) ∪ (− *?C*) = − *path-image g*
using *hull-subset* **by** *fastforce*
then have *inside* (*path-image g*) ⊆ *?C* **using** *inside-subset*[*OF conn unbdd un*] **by** *blast*
moreover have *open* (*inside* (*path-image g*)) **using** *assms*(*2*) **by** (*simp add*: *open-inside closed-simple-path-image*)
ultimately show *?thesis* **using** *interior-maximal* **by** *blast*
qed
— Measurability: *inside* is bounded and open, hence Lebesgue measurable (cf.

HOL Light's *MEASURABLE-INSIDE*, which is just *MEASURABLE-OPEN*; *BOUNDED-INSIDE*; *OPEN-INSIDE*).
have *clg*: *closed* (*path-image g*) **using** *assms*(*2*) **by** (*simp add*: *closed-simple-path-image*)
have *bounded* (*inside* (*path-image g*))
using *Jordan-inside-outside*[*OF assms*(*2*,*3*)] **by** *simp*
then have *meas-ins-g*: *inside* (*path-image g*) ∈ *lmeasurable*
using *clg lmeasurable-open open-inside* **by** *blast*
have *ins-h*: *inside* (*path-image h*) = *interior* (*convex hull* (*path-image g*))
using *h*(*6*) *inside-frontier-eq-interior*[*OF bdd-hull convex-convex-hull*] **by** *simp*
have *meas-ins-h*: *inside* (*path-image h*) ∈ *lmeasurable*
by (*simp add*: *bdd-hull ins-h lmeasurable-interior*)
— The path image is not contained in the frontier of its convex hull (else *inside g* would be convex), so it meets the interior of the hull.
have *not-sub*: ¬ *path-image g* ⊆ *frontier* (*convex hull* (*path-image g*))
using *assms frontier-convex-hull-subset-path-image h*(*6*) *ins-h* **by** *fastforce*
have *pig-hull*: *path-image g* ⊆ *convex hull* (*path-image g*) **by** (*rule hull-subset*)
have *ex-int*: ∃ *x*. *x* ∈ *path-image g* ∧ *x* ∈ *interior* (*convex hull* (*path-image g*))
proof −
obtain *x* **where** *x*: *x* ∈ *path-image g x* ∉ *frontier* (*convex hull* (*path-image g*))
using *not-sub* **by** *blast*
then have *x* ∈ *closure* (*convex hull* (*path-image g*)) **using** *pig-hull closure-subset* **by** *blast*
then show *?thesis* **using** *x*
using *frontier-def* **by** *auto*
qed
have *open-outside*: *open* (*outside* (*path-image g*))
and *frontier-outside*: *frontier* (*outside* (*path-image g*)) = *path-image g*
using *Jordan-inside-outside*[*OF assms*(*2*,*3*)] **by** *blast*+
— Find a frontier point *x* of *g* in the hull interior; *frontier-straddle* gives a nearby outside point *y*, and a ball about *y* lies in *outside g* ∩ *interior*(*hull*).
obtain *x* **where** *x-pig*: *x* ∈ *path-image g* **and** *x-int*: *x* ∈ *interior* (*convex hull* (*path-image g*))
using *ex-int* **by** *blast*
obtain *r1* **where** *r1*: *r1* > *0 ball x r1* ⊆ *interior* (*convex hull* (*path-image g*))
using *x-int open-contains-ball*[*of interior* (*convex hull* (*path-image g*))] **by** *auto*
have *x-fro-out*: *x* ∈ *frontier* (*outside* (*path-image g*)) **using** *x-pig frontier-outside* **by** *simp*
obtain *y* **where** *y* ∈ *outside* (*path-image g*) *dist x y* < *r1*
using *frontier-straddle r1*(*1*) *x-fro-out* **by** *blast*
then have*y* ∈ *outside* (*path-image g*) ∩ *interior* (*convex hull* (*path-image g*))
using *r1*(*2*) **by** *fastforce*
moreover have *open* (*outside* (*path-image g*) ∩ *interior* (*convex hull* (*path-image g*)))
using *open-outside open-Int open-interior* **by** *blast*
ultimately obtain *r2* **where** *r2*: *r2* > *0 ball y r2* ⊆ *outside* (*path-image g*) ∩ *interior* (*convex hull* (*path-image g*))
by (*meson open-contains-ball*)
— This ball lies in *inside h* − *inside g*: it is in *interior*(*hull*) = *inside h* and in *outside g*, which is disjoint from *inside g*.

have *ball-sub*: *ball y r2* ⊆ *inside* (*path-image h*) − *inside* (*path-image g*)
proof
fix *z* **assume** *z*: *z* ∈ *ball y r2*
have *z* ∈ *outside* (*path-image g*) *z* ∈ *interior* (*convex hull* (*path-image g*))
using *z r2*(*2*) **by** *blast*+
then show *z* ∈ *inside* (*path-image h*) − *inside* (*path-image g*)
by (*metis Diff-iff Diff-triv ins-h inside-Int-outside*)
qed
— A nonempty ball has positive measure, so the difference does too.
have *ball-pos*: *measure lebesgue* (*ball y r2*) > *0*
by (*simp add*: *r2*(*1*))
have *diff-meas*: *inside* (*path-image h*) − *inside* (*path-image g*) ∈ *lmeasurable*
using *meas-ins-h meas-ins-g* **by** (*simp add*: *fmeasurable.Diff*)
have *diff-pos*: *measure lebesgue* (*inside* (*path-image h*) − *inside* (*path-image g*))
> *0*
by (*meson ball-pos ball-sub diff-meas negligible-iff-measure negligible-subset*
zero-less-measure-iff)
have *ins-g-sub-h*: *inside* (*path-image g*) ⊆ *inside* (*path-image h*)
using *ins-g-sub ins-h* **by** *simp*
have *meas-eq*: *measure lebesgue* (*inside* (*path-image h*) − *inside* (*path-image g*))
=
measure lebesgue (*inside* (*path-image h*)) − *measure lebesgue* (*inside* (*path-image*
g))
using *measurable-measure-Diff*[*OF meas-ins-h - ins-g-sub-h*] *meas-ins-g* **by**
(*simp add*: *fmeasurableD*)
have *measure lebesgue* (*inside* (*path-image g*)) < *measure lebesgue* (*inside* (*path-image*
h))
using *diff-pos meas-eq* **by** *linarith*
with *h that* **show** *?thesis* **by** *blast*
qed

4.2 The isoperimetric theorem

theorem *isoperimetric-theorem*:
fixes *g* :: *real* ⇒ *complex*
assumes *rectifiable-path g simple-path g*
pathfinish g = *pathstart g*
path-length g = *L*
shows *measure lebesgue* (*inside* (*path-image g*)) ≤ L^2 / (*4* ∗ *pi*) ∧
(*measure lebesgue* (*inside* (*path-image g*)) = L^2 / (*4* ∗ *pi*) ⟶ (∃ *a r*.
path-image g = *sphere a r*))
proof (*cases convex* (*inside* (*path-image g*)))
case *True*
show *?thesis*
using *isoperimetric-theorem-convex*[*OF assms*(*1*−*3*) *True assms*(*4*)]
by *blast*
next
case *False*
obtain *h* **where** *h*: *rectifiable-path h simple-path h*

pathfinish h = pathstart h
path-length h ≤ path-length g
convex hull (path-image h) = convex hull (path-image g)
path-image h = frontier (convex hull (path-image g))
measure lebesgue (inside (path-image g)) < measure lebesgue (inside (path-image h))
by (*rule isoperimetric-convexification-strict*[*OF assms*(*1*−*3*) *False*])
have *bounded-hull*: *bounded* (*convex hull* (*path-image g*))
by (*intro bounded-convex-hull compact-imp-bounded compact-simple-path-image assms*(*2*))
have *eq-int*: *inside* (*path-image h*) = *interior* (*convex hull* (*path-image g*))
using *inside-frontier-eq-interior*[*OF bounded-hull convex-convex-hull*] *h*(*6*) **by** *simp*
have *convex-h*: *convex* (*inside* (*path-image h*))
using *eq-int convex-interior*[*OF convex-convex-hull*] **by** *simp*
have *ineq-h*: *measure lebesgue* (*inside* (*path-image h*)) ≤ (*path-length h*)^2 / (*4* ∗ *pi*)
using *isoperimetric-theorem-convex*(*1*)[*OF h*(*1*−*3*) *convex-h refl*] **.**
have *mono*: (*path-length h*)^2 / (*4* ∗ *pi*) ≤ L^2 / (*4* ∗ *pi*)
by (*intro divide-right-mono power-mono*)
(*use h*(*4*) *assms*(*4*) *path-length-pos-le*[*OF h*(*1*)] *pi-gt-zero* **in** *simp-all*)
show *?thesis* **using** *h*(*7*) *ineq-h mono* **by** *linarith*
qed

end