Pick's Theorem

Sage Binder and Katherine Kosaian

February 6, 2026

Abstract

We formalize Pick's theorem for finding the area of a simple polygon whose vertices are integral lattice points [1]. We are inspired by John Harrison's formalization of Pick's theorem in HOL Light [2], but tailor our proof approach to avoid a primary challenge point in his formalization, which is proving that any polygon with more than three vertices can be split (in its interior) by a line between some two vertices. Our formalization involves augmenting the existing geometry libraries in various foundational ways (e.g., by adding the definition of a polygon and formalizing some key properties thereof).

Contents

1 Misc. Linear Algebra Setup **3**

2 Integral Bijective Matrix Determinant **5**

3 Polygon Definitions **8**

4 Jordan Curve Theorem for Polygons **9**

5 Properties of make polygonal path, pathstart and pathfinish of a polygon **22**

6 Loop Free Properties **30**

7 Explicit Linepath Characterization of Polygonal Paths **36**

8 A Triangle is a Polygon **46**

9 Polygon Vertex Rotation **55**

10 Translating a Polygon **84**

11 Misc. properties **86**

12 Properties of Sublists of Polygonal Path Vertex Lists **90**

13 Reversing Polygonal Path Vertex List **116**

14 Collinearity Properties **121**

15 Linepath Properties **125**

16 Measure of linepaths **133**

17 Misc. Convex Polygon Properties **136**

18 Vertices on Convex Frontier Implies Polygon is Convex **142**

19 Polygon Splitting **156**

20 Triangles **179**

21 Measure Setup **188**

22 Unit Triangle **188**

23 Unit Square **193**

24 Unit Triangle Area is 1/2 **198**

25 Area of Elementary Triangle is 1/2 **201**

26 Setup **206**
26.1 Integral Points Cardinality Properties 206

27 Pick splitting **209**

28 Convex Hull Has Good Linepath **222**

29 Pick's Theorem **228**
29.1 Pick's Theorem Triangle Case 229
29.2 Pocket properties 255
29.3 Arbitrary Polygon Case 332

theory *Integral-Matrix*
imports
Complex-Main
HOL−Analysis.Finite-Cartesian-Product
HOL−Analysis.Linear-Algebra
HOL−Analysis.Determinants
begin

1 Misc. Linear Algebra Setup

lemma *vec-scaleR-2*: (*c*::*real*) $*_R$ ((*vector* [*a*, *b*])::*real^2*) = *vector* [*a* * *c*, *b* * *c*]
proof−
  have (*c* $*_R$ (*vector* [*a*, *b*])::*real^2*)$*1* = *a* * *c* **by** *simp*
  moreover have (*c* $*_R$ (*vector* [*a*, *b*])::*real^2*)$*2* = ((*vector* [*a*, *b*])::*real^2*)$*2* * *c* **by** *simp*
  ultimately show *?thesis* **by** (*smt* (*verit*, *best*) *exhaust-2 vec-eq-iff vector-2*(*1*) *vector-2*(*2*))
qed

definition *is-int* :: *real* ⇒ *bool* **where**
  is-int x ⟷ (∃ *n*::*int*. *x* = *n*)

lemma *is-int-sum*: *is-int x* ∧ *is-int y* ⟶ *is-int* (*x* + *y*)
  by (*metis is-int-def of-int-add*)

lemma *is-int-minus*: *is-int x* ∧ *is-int y* ⟶ *is-int* (*x* − *y*)
  by (*metis is-int-def of-int-diff*)

lemma *is-int-mult*: *is-int x* ∧ *is-int y* ⟶ *is-int* (*x* * *y*)
  by (*metis is-int-def of-int-mult*)

definition *integral-vec* :: *real^2* ⇒ *bool* **where**
  integral-vec v ⟷ (*is-int* (*v*$*1*) ∧ *is-int* (*v*$*2*))

lemma *integral-vec-sum*: *integral-vec v* ∧ *integral-vec w* ⟶ *integral-vec* (*v* + *w*)
proof(*rule impI*)
  fix *v w* :: *real^2*
  let *?x* = *v* + *w*
  assume *integral-vec v* ∧ *integral-vec w*
  then obtain *v1 v2 w1 w2* :: *int* **where** *v*$*1* = *v1* ∧ *v*$*2* = *v2* ∧ *w*$*1* = *w1* ∧ *w*$*2* = *w2*
    using *integral-vec-def is-int-def* **by** *auto*
  then have *?x*$*1* = *v1* + *w1* **and** *?x*$*2* = *v2* + *w2* **by** *auto*
  thus *integral-vec ?x* **using** *integral-vec-def is-int-def* **by** *blast*
qed

lemma *integral-vec-minus*: *integral-vec v* ⟶ *integral-vec* (−*v*)
proof(*rule impI*)
  assume *integral-vec v*
  then obtain *x y* :: *int* **where** *v*$*1* = *x* ∧ *v*$*2* = *y*
    using *integral-vec-def is-int-def* **by** *auto*
  then have (−*v*)$*1* = −*x* **and** (−*v*)$*2* = −*y*
    using *integral-vec-def is-int-def* **by** *auto*
  thus *integral-vec* (−*v*)
    using *integral-vec-def is-int-def* **by** *blast*
qed

lemma *real-2-inner*:
shows ((*vector* [*a*, *b*])::(*real*^2)) · ((*vector* [*c*, *d*])::(*real*^2)) = *a***c* + *b***d*
(**is** *?v* · *?w* = *a***c* + *b***d*)
proof−
have *?v* · *?w* = (∑ *i* ∈ *UNIV*. *?v*$*i* · *?w*$*i*) **using** *inner-vec-def*[*of ?v ?w*] **by** *blast*
moreover have ∀ *i*. *?v*$*i* · *?w*$*i* = *?v*$*i* * *?w*$*i* **using** *inner-real-def* **by** *simp*
ultimately have *?v* · *?w* = (∑ *i* ∈ *UNIV*. *?v*$*i* * *?w*$*i*) **by** *presburger*
thus *?thesis* **by** (*simp add*: *sum-2*)
qed

lemma *integral-vec-2*:
fixes *a b* :: *int*
assumes *v* = *vector* [*a*, *b*]
shows *integral-vec v*
by (*simp add*: *assms is-int-def integral-vec-def*)

definition *matrix-inv* :: *real*^2^2 ⇒ *real*^2^2 ⇒ *bool* **where**
matrix-inv A A' ⟷ (*A* ** *A'* = *mat 1* ∧ *A'* ** *A* = *mat 1*)

lemma *mat-vec-mult-2*:
fixes *v* :: *real*^2 **and**
T :: *real*^2^2
defines *x*: *x* ≡ *v*$*1* **and** *y*: *y* ≡ *v*$*2* **and**
a: *a* ≡ *T*$*1*$*1* **and** *b*: *b* ≡ *T*$*1*$*2* **and**
c: *c* ≡ *T*$*2*$*1* **and** *d*: *d* ≡ *T*$*2*$*2*
shows (*T* **v v*) = *vector* [*x***a* + *y***b*, *x***c* + *y***d*]
proof−
have (*T* **v v*)$*1* = *x***a* + *y***b* **by** (*simp add*: *a b matrix-vector-mult-def sum-2 x y*)
moreover have (*T* **v v*)$*2* = *x***c* + *y***d* **by** (*simp add*: *c d matrix-vector-mult-def sum-2 x y*)
ultimately show *T* **v v* = *vector* [*x***a* + *y***b*, *x***c* + *y***d*]
by (*smt* (*verit*) *exhaust-2 vec-eq-iff vector-2*(*1*) *vector-2*(*2*))
qed

definition *integral-mat* :: *real*^2^2 ⇒ *bool* **where**
integral-mat T ⟷ (∀ *v*. *integral-vec v* ⟶ *integral-vec* (*T* **v v*))

definition *integral-mat-surj* :: *real*^2^2 ⇒ *bool* **where**
integral-mat-surj T ⟷ (∀ *v*. *integral-vec v* ⟶ (∃ *w*. *integral-vec w* ∧ *T* **v w* = *v*))

definition *integral-mat-bij* :: *real*^2^2 ⇒ *bool* **where**
integral-mat-bij T ⟷ *integral-mat T* ∧ *integral-mat-surj T*

lemma *integral-mat-integral-vec*: *integral-mat A* ⟶ *integral-vec v* ⟶ *integral-vec* (*A* **v v*)
using *integral-mat-def* **by** *blast*

lemma *integral-mat-int-entries*:
fixes T :: *real^2^2*
assumes *integral-mat* T
defines a: $a \equiv T\$1\$1$ **and** b: $b \equiv T\$1\$2$ **and**
c: $c \equiv T\$2\$1$ **and** d: $d \equiv T\$2\$2$
shows *is-int* $a \wedge$ *is-int* $b \wedge$ *is-int* $c \wedge$ *is-int* d
proof−
let *?v* = *vector* [*1*, *0*]
have *integral-vec* (*?v*) **using** *integral-vec-2*[*of ?v 1 0*] **by** *auto*
then have *integral-vec* (T ∗*v ?v*) **using** *assms integral-mat-def* **by** *blast*
moreover have T ∗*v ?v* = *vector* [a, c]
using *mat-vec-mult-2*[*of T ?v*] *a b c d* **by** *auto*
ultimately have *integral-vec* (*vector* [a, c]) **by** *auto*
then have *1*: *is-int a* ∧ *is-int c* **using** *integral-vec-def* **by** *auto*

let *?w* = *vector* [*0*, *1*]
have *integral-vec* (*?w*) **using** *integral-vec-2*[*of ?w 0 1*] **by** *auto*
then have *integral-vec* (T ∗*v ?w*) **using** *assms integral-mat-def* **by** *blast*
moreover have T ∗*v ?w* = *vector* [b, d]
using *mat-vec-mult-2*[*of T ?w*] *a b c d* **by** *auto*
ultimately have *integral-vec* (*vector* [b, d]) **by** *auto*
then have *2*: *is-int b* ∧ *is-int d* **using** *integral-vec-def* **by** *auto*

thus *?thesis* **using** *1 2* **by** *auto*
qed

2 Integral Bijective Matrix Determinant

lemma *integral-mat-int-det*:
fixes T :: *real^2^2*
assumes *integral-mat* T
shows *is-int* (*det* T)
proof−
obtain *a b c d* **where** *abcd*: $T\$1\$1 = a \wedge T\$1\$2 = b \wedge T\$2\$1 = c \wedge T\$2\$2 = d$ **by** *auto*
have *abcd-int*: *is-int a* ∧ *is-int b* ∧ *is-int c* ∧ *is-int d*
using *integral-mat-int-entries*[*of T*] *abcd assms* **by** *auto*
obtain *ai bi ci di* :: *int* **where** *abcdi*: $ai = a \wedge bi = b \wedge ci = c \wedge di = d$
using *abcd-int is-int-def* **by** *auto*
have *det T* = *a*∗*d* − *b*∗*c* **using** *det-2*[*of T*] *abcd* **by** *auto*
also have ... = *ai*∗*di* − *bi*∗*ci* **using** *abcdi* **by** *auto*
finally show *?thesis* **using** *is-int-def* **by** *blast*
qed

lemma *integral-mat-bij-inv*:
fixes T :: *real^2^2*
assumes *integral-mat-bij T*

obtains *Tinv* **where** *invertible T* ∧ *integral-mat-bij Tinv* ∧ *matrix-inv T Tinv*
proof−
let *?e1* = *vector* [*1*, *0*]
let *?e2* = *vector* [*0*, *1*]
let *?I* = (*vector* [*?e1*, *?e2*])::(*real^2^2*)
have *id*: *?I* = ((*mat 1*)::(*real^2^2*))
unfolding *vec-eq-iff*
by (*smt* (*verit*, *ccfv-threshold*) *exhaust-2 mat-def vec-lambda-beta vector-2*)
have *integral-vec ?e1*
by (*simp add*: *integral-vec-def is-int-def*)
moreover have *integral-vec ?e2*
by (*simp add*: *integral-vec-def is-int-def*)
ultimately obtain *x y* **where** *xy*: *T* ∗*v x* = *?e1* ∧ *integral-vec x* ∧ *T* ∗*v y* = *?e2* ∧ *integral-vec y*
by (*meson assms integral-mat-bij-def integral-mat-surj-def*)

let *?Tinv* = *transpose* (*vector* [*x*, *y*])::(*real^2^2*)
have *T* ∗∗ *?Tinv* = *mat 1* (**is** *?TxTinv* = *mat 1*)
proof−
have *column 1 ?TxTinv* = *T* ∗*v* (*column 1 ?Tinv*)
by (*metis matrix-vector-mul-assoc matrix-vector-mult-basis*)
also have ... = *T* ∗*v x*
by (*simp add*: *row-def*)
finally have [*simp*]: *column 1 ?TxTinv* = *?e1*
using *xy* **by** *presburger*

have *column 2 ?TxTinv* = *T* ∗*v* (*column 2 ?Tinv*)
by (*metis matrix-vector-mul-assoc matrix-vector-mult-basis*)
also have ... = *T* ∗*v y*
by (*simp add*: *row-def*)
finally have [*simp*]: *column 2 ?TxTinv* = *?e2*
using *xy* **by** *presburger*

have ∀ *v*. *?TxTinv* ∗*v v* = *v*
proof(*rule allI*)
fix *v* :: *real^2*

have (*?TxTinv* ∗*v v*)$*1* = (*column 1 ?TxTinv*)$*1* ∗ *v*$*1* + (*column 2 ?TxTinv*)$*1* ∗ *v*$*2*
by (*metis* (*no-types*, *lifting*) *cart-eq-inner-axis mat-vec-mult-2 matrix-vector-mul-component matrix-vector-mult-basis mult.commute vector-2*(*1*))
also have ... = *v*$*1* **by** *simp*
finally have *v1*: (*?TxTinv* ∗*v v*)$*1* = *v*$*1* **.**

have (*?TxTinv* ∗*v v*)$*2* = (*column 1 ?TxTinv*)$*2* ∗ *v*$*1* + (*column 2 ?TxTinv*)$*2* ∗ *v*$*2*
by (*metis* (*no-types*, *lifting*) *cart-eq-inner-axis mat-vec-mult-2 matrix-vector-mul-component matrix-vector-mult-basis mult.commute vector-2*(*2*))
also have ... = *v*$*2* **by** *simp*

```
    finally have v2: (?TxTinv *v v)$2 = v$2 .

   show ?TxTinv *v v = v using v1 v2 by (metis mat-vec-mult-2 matrix-vector-mul-lid)
   qed
   thus ?thesis by (simp add: matrix-eq)
  qed
  then have matrix-inv T ?Tinv
   by (simp add: Integral-Matrix.matrix-inv-def matrix-left-right-inverse)
  moreover have invertible T using calculation invertible-def matrix-inv-def by
blast
  moreover have integral-mat-bij ?Tinv
    by (smt (verit, del-insts) ‹T ** Finite-Cartesian-Product.transpose (vector
[x, y]) = mat 1› assms integral-mat-bij-def integral-mat-def integral-mat-surj-def
matrix-left-right-inverse matrix-mul-lid matrix-vector-mul-assoc)
  ultimately show ?thesis
   using ‹T ** Finite-Cartesian-Product.transpose (vector [x, y]) = mat 1› in-
vertible-right-inverse that by blast
qed


lemma integral-mat-bij-det-pm1:
  fixes T :: real^2^2
  assumes integral-mat-bij T
  shows det T = 1 ∨ det T = −1
proof−
  obtain Tinv where Tinv: invertible T ∧ integral-mat-bij Tinv ∧ matrix-inv T
Tinv
   using integral-mat-bij-inv[of T] assms by auto
  moreover have is-int (det Tinv)
   using integral-mat-bij-def integral-mat-int-det[of Tinv] calculation by auto
  moreover have is-int (det T)
   using integral-mat-bij-def integral-mat-int-det[of T] assms by auto
  moreover have det Tinv = 1 / det T
  proof−
   have id: Tinv ** T = mat 1 using Tinv unfolding matrix-inv-def invertible-def
     by (simp add: verit-sko-ex′)
   have det Tinv * det T = det (Tinv ** T) by (simp add: det-mul)
   also have ... = det ((mat 1)::real^2^2) using id by auto
   also have ... = (1::real) by auto
   finally have det Tinv * det T = 1 .
   thus ?thesis using invertible-det-nz nonzero-eq-divide-eq by fastforce
  qed
  ultimately have T-Tinv-int: is-int (det T) ∧ is-int (1 / det T) by auto
  thus det T = 1 ∨ det T = −1
  proof−
   have abs (det T) ≤ 1 (is ?D ≤ 1)
   proof(rule ccontr)
    assume ¬ ?D ≤ 1
    then have ?D > 1 by auto
```

moreover from *this* **have** *1 / ?D < 1* **by** *auto*
moreover from *calculation* **have** *1 / ?D > 0* **by** *auto*
ultimately have ¬ *is-int (1 / ?D)* **unfolding** *is-int-def* **by** *force*
moreover from *T-Tinv-int* **have** *is-int (1 / ?D)*
by (*smt* (*verit*) ‹*1 / |det T| < 1*› *abs-div-pos abs-divide abs-ge-self abs-minus-cancel divide-cancel-left divide-pos-neg int-less-real-le is-int-def of-int-code*(*2*))
ultimately show *False* **by** *auto*
qed
then have *det T ≥ −1 ∧ det T ≤ 1*
using *assms* **by** *auto*
moreover have *det T ≠ 0* **using** *integral-mat-bij-inv invertible-det-nz assms* **by** *auto*
ultimately show *det T = 1 ∨ det T = −1* **using** *is-int-def T-Tinv-int* **by** *auto*
qed
qed

end
theory *Polygon-Jordan-Curve*
imports
HOL−Analysis.Cartesian-Space
HOL−Analysis.Path-Connected
Poincare-Bendixson.Poincare-Bendixson
Integral-Matrix

begin

3 Polygon Definitions

type-synonym *R-to-R2 = (real ⇒ real^2)*

definition *closed-path :: R-to-R2 ⇒ bool* **where**
closed-path g ⟷ path g ∧ pathstart g = pathfinish g

definition *path-inside :: R-to-R2 ⇒ (real^2) set* **where**
path-inside g = inside (path-image g)

definition *path-outside :: R-to-R2 ⇒ (real^2) set* **where**
path-outside g = outside (path-image g)

fun *make-polygonal-path :: (real^2) list ⇒ R-to-R2* **where**
make-polygonal-path [] = linepath 0 0
| *make-polygonal-path [a] = linepath a a*
| *make-polygonal-path [a,b] = linepath a b*
| *make-polygonal-path (a # b # xs) = (linepath a b) +++ make-polygonal-path (b # xs)*

definition *polygonal-path :: R-to-R2 ⇒ bool* **where**
polygonal-path g ⟷ g ∈ make-polygonal-path‘{xs :: (real^2) list. True}

definition *all-integral* :: $(real\hat{}2)\ list \Rightarrow bool$ **where**
all-integral $l = (\forall x \in set\ l.\ integral\text{-}vec\ x)$

definition *polygon* :: *R-to-R2* $\Rightarrow$ *bool* **where**
polygon $g \longleftrightarrow polygonal\text{-}path\ g \wedge simple\text{-}path\ g \wedge closed\text{-}path\ g$

definition *integral-polygon* :: *R-to-R2* $\Rightarrow$ *bool* **where**
integral-polygon $g \longleftrightarrow$
$(polygon\ g \wedge (\exists\, vts.\ g = make\text{-}polygonal\text{-}path\ vts \wedge all\text{-}integral\ vts))$

definition *make-triangle* :: $real\hat{}2 \Rightarrow real\hat{}2 \Rightarrow real\hat{}2 \Rightarrow$ *R-to-R2* **where**
make-triangle $a\ b\ c = make\text{-}polygonal\text{-}path\ [a, b, c, a]$

definition *polygon-of* :: *R-to-R2* $\Rightarrow (real\hat{}2)\ list \Rightarrow bool$ **where**
polygon-of $p\ vts \longleftrightarrow polygon\ p \wedge p = make\text{-}polygonal\text{-}path\ vts$

definition *good-linepath* :: $real\hat{}2 \Rightarrow real\hat{}2 \Rightarrow (real\hat{}2)\ list \Rightarrow bool$ **where**
good-linepath $a\ b\ vts \longleftrightarrow (let\ p = make\text{-}polygonal\text{-}path\ vts\ in$
$a \neq b \wedge \{a, b\} \subseteq set\ vts \wedge path\text{-}image\ (linepath\ a\ b) \subseteq path\text{-}inside\ p \cup \{a, b\})$

definition *good-polygonal-path* :: $real\hat{}2 \Rightarrow (real\hat{}2)\ list \Rightarrow real\hat{}2 \Rightarrow (real\hat{}2)\ list$
$\Rightarrow bool$ **where**
good-polygonal-path $a\ cutvts\ b\ vts \longleftrightarrow$ (
let $p = make\text{-}polygonal\text{-}path\ vts\ in$
let $p\text{-}cut = make\text{-}polygonal\text{-}path\ ([a] @ cutvts @ [b])\ in$
$(a \neq b \wedge \{a, b\} \subseteq set\ vts \wedge path\text{-}image\ (p\text{-}cut) \subseteq path\text{-}inside\ p \cup \{a, b\} \wedge$
$loop\text{-}free\ p\text{-}cut))$

4 Jordan Curve Theorem for Polygons

definition *inside-outside* :: *R-to-R2* $\Rightarrow (real\hat{}2)\ set \Rightarrow (real\hat{}2)\ set \Rightarrow bool$ **where**
inside-outside $p\ ins\ outs \longleftrightarrow$
$(ins \neq \{\} \wedge open\ ins \wedge connected\ ins \wedge$
$outs \neq \{\} \wedge open\ outs \wedge connected\ outs \wedge$
$bounded\ ins \wedge \neg\ bounded\ outs \wedge$
$ins \cap outs = \{\} \wedge ins \cup outs = -\ path\text{-}image\ p \wedge$
$frontier\ ins = path\text{-}image\ p \wedge frontier\ outs = path\text{-}image\ p)$

lemma *Jordan-inside-outside-real2*:
fixes $p :: real \Rightarrow real\hat{}2$
assumes $simple\text{-}path\ p\ pathfinish\ p = pathstart\ p$
shows $inside(path\text{-}image\ p) \neq \{\} \wedge$
$open(inside(path\text{-}image\ p)) \wedge$
$connected(inside(path\text{-}image\ p)) \wedge$
$outside(path\text{-}image\ p) \neq \{\} \wedge$
$open(outside(path\text{-}image\ p)) \wedge$
$connected(outside(path\text{-}image\ p)) \wedge$

bounded(*inside*(*path-image p*)) ∧
¬ *bounded*(*outside*(*path-image p*)) ∧
inside(*path-image p*) ∩ *outside*(*path-image p*) = {} ∧
inside(*path-image p*) ∪ *outside*(*path-image p*) =
− *path-image p* ∧
frontier(*inside*(*path-image p*)) = *path-image p* ∧
frontier(*outside*(*path-image p*)) = *path-image p*
proof −
have *good-type*: *c1-on-open-R2-axioms TYPE*((*real*, *2*) *vec*)
unfolding *c1-on-open-R2-axioms-def* **by** *auto*
have *inside*(*path-image p*) ≠ {} ∧
open(*inside*(*path-image p*)) ∧
connected(*inside*(*path-image p*)) ∧
outside(*path-image p*) ≠ {} ∧
open(*outside*(*path-image p*)) ∧
connected(*outside*(*path-image p*)) ∧
bounded(*inside*(*path-image p*)) ∧
¬ *bounded*(*outside*(*path-image p*)) ∧
inside(*path-image p*) ∩ *outside*(*path-image p*) = {} ∧
inside(*path-image p*) ∪ *outside*(*path-image p*) =
− *path-image p* ∧
frontier(*inside*(*path-image p*)) = *path-image p* ∧
frontier(*outside*(*path-image p*)) = *path-image p*
using *assms c1-on-open-R2.Jordan-inside-outside-R2*[*of* - - - *p*]
unfolding *c1-on-open-R2-def c1-on-open-euclidean-def c1-on-open-def* **using** *good-type*
by (*metis continuous-on-empty equals0D open-empty*)
then show *?thesis* **unfolding** *inside-outside-def*
using *path-inside-def path-outside-def* **by** *auto*
qed

lemma *inside-outside-polygon*:
fixes *p* :: *R-to-R2*
assumes *polygon*: *polygon p*
shows *inside-outside p* (*path-inside p*) (*path-outside p*)
proof−
have *good-type*: *c1-on-open-R2-axioms TYPE*((*real*, *2*) *vec*)
unfolding *c1-on-open-R2-axioms-def* **by** *auto*
have *simple-path p pathfinish p* = *pathstart p* **using** *assms polygon-def closed-path-def* **by** *auto*
then show *?thesis* **using** *Jordan-inside-outside-real2* **unfolding** *inside-outside-def*
using *path-inside-def path-outside-def* **by** *auto*
qed

lemma *inside-outside-unique*:
fixes *p* :: *R-to-R2*
assumes *polygon p*
assumes *io1*: *inside-outside p inside1 outside1*

assumes *io2*: *inside-outside p inside2 outside2*
shows *inside1 = inside2 ∧ outside1 = outside2*
proof −
have *inner1*: *inside*(*path-image p*) = *inside1*
using *dual-order.antisym inside-subset interior-eq interior-inside-frontier*
using *io1* **unfolding** *inside-outside-def*
by *metis*
have *inner2*: *inside*(*path-image p*) = *inside2*
using *dual-order.antisym inside-subset interior-eq interior-inside-frontier*
using *io2* **unfolding** *inside-outside-def*
by *metis*
have *eq1*: *inside1 = inside2*
using *inner1 inner2*
by *auto*
have *h1*: *inside1 ∪ outside1 = − path-image p*
using *io1* **unfolding** *inside-outside-def* **by** *auto*
have *h2*: *inside1 ∩ outside1 = {}*
using *io1* **unfolding** *inside-outside-def* **by** *auto*
have *outer1*: *outside*(*path-image p*) = *outside1*
using *io1 inner1* **unfolding** *inside-outside-def*
using *h1 h2 outside-inside* **by** *auto*
have *h3*: *inside2 ∪ outside2 = − path-image p*
using *io2* **unfolding** *inside-outside-def* **by** *auto*
have *h4*: *inside2 ∩ outside2 = {}*
using *io2* **unfolding** *inside-outside-def* **by** *auto*
have *outer2*: *outside*(*path-image p*) = *outside2*
using *io2 inner2* **unfolding** *inside-outside-def*
using *h3 h4 outside-inside* **by** *auto*
then have *eq2*: *outside1 = outside2*
using *outer1 outer2* **by** *auto*
then show *?thesis* **using** *eq1 eq2* **by** *auto*
qed

lemma *polygon-jordan-curve*:
fixes *p* :: *R-to-R2*
assumes *polygon p*
obtains *inside outside* **where**
inside-outside p inside outside
proof−
have *good-type*: *c1-on-open-R2-axioms TYPE*((*real*, *2*) *vec*)
unfolding *c1-on-open-R2-axioms-def* **by** *auto*
have *simple-path p pathfinish p = pathstart p* **using** *assms polygon-def closed-path-def*
by *auto*
then obtain *inside outside* **where**
inside ≠ {} open inside connected inside
outside ≠ {} open outside connected outside
bounded inside ¬ bounded outside inside ∩ outside = {}
inside ∪ outside = − path-image p
frontier inside = path-image p

frontier outside = path-image p
using *c1-on-open-R2.Jordan-curve-R2[of - - - p]*
unfolding *c1-on-open-R2-def c1-on-open-euclidean-def c1-on-open-def* **using** *good-type*
by (*metis continuous-on-empty equals0D open-empty*)
then show *?thesis*
using *inside-outside-def that* **by** *auto*
qed

lemma *connected-component-image*:
fixes *f* :: *'a*::*euclidean-space* ⇒ *'b*::*euclidean-space*
assumes *linear f bij f*
shows *f* ‘ (*connected-component-set S x*) = *connected-component-set* (*f* ‘ *S*) (*f x*)
proof −
have *conn*: ⋀*S*. *connected S* ⟹ *connected* (*f* ‘ *S*)
by (*simp add*: *assms*(*1*) *connected-linear-image*)
then have *h1*: ⋀*T*. *T* ∈ {*T*. *connected T* ∧ *x* ∈ *T* ∧ *T* ⊆ *S*} ⟹ *f* ‘ *T* ∈ {*T*. *connected T* ∧ (*f x*) ∈ *T* ∧ *T* ⊆ (*f* ‘ *S*)}
by *auto*
then have *subset1*: *f* ‘ *connected-component-set S x* ⊆ *connected-component-set* (*f* ‘ *S*) (*f x*)
using *connected-component-Union*
by (*smt* (*verit*, *ccfv-threshold*) *assms*(*2*) *bij-is-inj connected-component-eq-empty connected-component-maximal connected-component-refl-eq connected-component-subset connected-connected-component image-is-empty inj-image-mem-iff mem-Collect-eq*)
have ⋀*S*. *connected* (*f* ‘ *S*) ⟹ *connected S*
using *assms connected-continuous-image assms linear-continuous-on linear-conv-bounded-linear bij-is-inj homeomorphism-def linear-homeomorphism-image*
by (*smt* (*verit*, *del-insts*))
then have *h2*: ⋀*T*. *f* ‘ *T* ∈ {*T*. *connected T* ∧ (*f x*) ∈ *T* ∧ *T* ⊆ (*f* ‘ *S*)} ⟹ *T* ∈ {*T*. *connected T* ∧ *x* ∈ *T* ∧ *T* ⊆ *S*}
by (*simp add*: *assms*(*2*) *bij-is-inj image-subset-iff inj-image-mem-iff subsetI*)
then have *subset2*: *connected-component-set* (*f* ‘ *S*) (*f x*) ⊆ *f* ‘ *connected-component-set S x*
using *connected-component-Union*[*of S x*] *connected-component-Union*[*of f*‘*S f x*]
by (*smt* (*verit*, *del-insts*) *assms*(*2*) *bij-is-inj connected-component-eq-empty connected-component-maximal connected-component-refl-eq connected-component-subset connected-connected-component image-mono inj-image-mem-iff mem-Collect-eq subset-imageE*)
show *f* ‘ (*connected-component-set S x*) = *connected-component-set* (*f* ‘ *S*) (*f x*)
using *subset1 subset2* **by** *auto*
qed

lemma *bounded-map*:
fixes *f* :: *'a*::*euclidean-space* ⇒ *'b*::*euclidean-space*
assumes *linear f bij f*
shows *bounded* (*f* ‘ *S*) = *bounded S*

proof −
have *h1*: *bounded S* ⟹ *bounded* (*f* ‘ *S*)
using *assms*
using *bounded-linear-image linear-conv-bounded-linear* **by** *blast*
have *bounded-linear f*
using *linear-conv-bounded-linear assms* **by** *auto*
then have *bounded-linear* (*inv f*)
using *assms* **unfolding** *bij-def*
by (*smt* (*verit*, *ccfv-threshold*) *bij-betw-def bij-betw-subset dim-image-eq inv-equality linear-conv-bounded-linear linear-surjective-isomorphism subset-UNIV*)
then have *h2*: *bounded* (*f* ‘ *S*) ⟹ *bounded S*
using *assms*
by (*metis bij-is-inj bounded-linear-image image-inv-f-f*)
then show *?thesis*
using *assms h1 h2* **by** *auto*
qed

lemma *inside-bijective-linear-image*:
fixes *f* :: *'a*::*euclidean-space* ⇒ *'b*::*euclidean-space*
fixes *c* :: *real* ⇒ *'a*
assumes *c-simple*:*path c*
assumes *linear f bij f*
shows *inside* (*f* ‘ (*path-image c*)) = *f* ‘ (*inside*(*path-image c*))
proof −
have *set1*: {*x*. *x* ∉ *f* ‘ *path-image c*} = *f* ‘ {*x*. *x* ∉ *path-image c*}
using *assms path-image-compose* **unfolding** *bij-def*
by (*smt* (*verit*, *best*) *UNIV-I imageE inj-image-mem-iff mem-Collect-eq subsetI subset-antisym*)
have *linear-inv*: *linear* (*inv f*)
using *assms*
by (*metis bij-imp-bij-inv bij-is-inj inv-o-cancel linear-injective-left-inverse o-inv-o-cancel*)
have *bij-inv*: *bij* (*inv f*)
using *assms*
using *bij-imp-bij-inv* **by** *blast*
have *inset1*: ⋀*x*. *x* ∈ {*x*. *bounded* (*connected-component-set* (− *f* ‘ *path-image c*) *x*)} ⟹ *x* ∈ *f* ‘ {*x*. *bounded* (*connected-component-set* (− *path-image c*) *x*)}
proof −
fix *x*
assume ∗: *x* ∈ {*x*. *bounded* (*connected-component-set* (− *f* ‘ *path-image c*) *x*)}
have *inj f*
using *assms*(*3*) *bij-betw-imp-inj-on* **by** *blast*
then show *x* ∈ *f* ‘ {*x*. *bounded* (*connected-component-set* (− *path-image c*) *x*)}
using ∗ *connected-component-image*[*OF linear-inv bij-inv*]
by (*smt* (*z3*) ‹⋀*x S*. *inv f* ‘ *connected-component-set S x* = *connected-component-set* (*inv f* ‘ *S*) (*inv f x*)› ‹*bij* (*inv f*)› ‹*linear* (*inv f*)› ‹*x* ∈ {*x*. *bounded* (*connected-component-set* (− *f* ‘ *path-image c*) *x*)}› *bij-image-Compl-eq bounded-map connected-component-eq-empty image-empty image-inv-f-f mem-Collect-eq*)
qed
have *inset2*: ⋀*x*. *x* ∈ *f* ‘ {*x*. *bounded* (*connected-component-set* (− *path-image*

$c)\ x)\} \Longrightarrow x \in \{x.\ bounded\ (connected\text{-}component\text{-}set\ (-\ f\ `\ path\text{-}image\ c)\ x)\}$
proof −
fix x
assume $x \in f\ `\ \{x.\ bounded\ (connected\text{-}component\text{-}set\ (-\ path\text{-}image\ c)\ x)\}$
then obtain $x1$ **where** $x = f\ x1\ x1 \in \{x.\ bounded\ (connected\text{-}component\text{-}set\ (-\ path\text{-}image\ c)\ x)\}$
by *auto*
then show $x \in \{x.\ bounded\ (connected\text{-}component\text{-}set\ (-\ f\ `\ path\text{-}image\ c)\ x)\}$

using *bounded-map*[*OF assms*(2) *assms*(3)] *connected-component-image*[*OF assms*(2) *assms*(3)]
by (*metis assms*(3) *bij-image-Compl-eq mem-Collect-eq*)
qed
have *set2*: $f\ `\ \{x.\ bounded\ (connected\text{-}component\text{-}set\ (-\ path\text{-}image\ c)\ x)\} = \{x.\ bounded\ (connected\text{-}component\text{-}set\ (-\ f\ `\ path\text{-}image\ c)\ x)\}$
using *inset1 inset2* **by** *auto*
have *inset1*: $\bigwedge x.\ x \in f\ `\ \{x.\ x \notin path\text{-}image\ c \wedge bounded\ (connected\text{-}component\text{-}set\ (-\ path\text{-}image\ c)\ x)\} \Longrightarrow$
$x \in \{x.\ x \notin f\ `\ path\text{-}image\ c \wedge bounded\ (connected\text{-}component\text{-}set\ (-\ f\ `\ path\text{-}image\ c)\ x)\}$
proof −
fix x
assume $x \in f\ `\ \{x.\ x \notin path\text{-}image\ c \wedge bounded\ (connected\text{-}component\text{-}set\ (-\ path\text{-}image\ c)\ x)\}$
then show $x \in \{x.\ x \notin f\ `\ path\text{-}image\ c \wedge bounded\ (connected\text{-}component\text{-}set\ (-\ f\ `\ path\text{-}image\ c)\ x)\}$
by (*metis* (*no-types*, *lifting*) *image-iff mem-Collect-eq set1 set2*)
qed
have *inset2*: $\bigwedge x.\ x \in \{x.\ x \notin f\ `\ path\text{-}image\ c \wedge bounded\ (connected\text{-}component\text{-}set\ (-\ f\ `\ path\text{-}image\ c)\ x)\} \Longrightarrow$
$x \in f\ `\ \{x.\ x \notin path\text{-}image\ c \wedge bounded\ (connected\text{-}component\text{-}set\ (-\ path\text{-}image\ c)\ x)\}$
proof −
fix x
assume $x \in \{x.\ x \notin f\ `\ path\text{-}image\ c \wedge bounded\ (connected\text{-}component\text{-}set\ (-\ f\ `\ path\text{-}image\ c)\ x)\}$
then show $x \in f\ `\ \{x.\ x \notin path\text{-}image\ c \wedge bounded\ (connected\text{-}component\text{-}set\ (-\ path\text{-}image\ c)\ x)\}$
by (*smt* (*verit*, *best*) *image-iff mem-Collect-eq set2*)
qed
have *same-set*: $\{x.\ x \notin f\ `\ path\text{-}image\ c \wedge bounded\ (connected\text{-}component\text{-}set\ (-\ f\ `\ path\text{-}image\ c)\ x)\} =$
$f\ `\ \{x.\ x \notin path\text{-}image\ c \wedge bounded\ (connected\text{-}component\text{-}set\ (-\ path\text{-}image\ c)\ x)\}$
using *inset1 inset2*
by *blast*
have *ins1*: $\bigwedge x.\ x \in inside\ (f\ `\ path\text{-}image\ c) \Longrightarrow x \in f\ `\ inside\ (path\text{-}image\ c)$
proof −
fix x

assume ∗: $x \in inside\ (f\ `\ path\text{-}image\ c)$
show $x \in f\ `\ inside\ (path\text{-}image\ c)$
by (*metis* (*no-types*) ∗ *same-set inside-def*)
qed
then have $inside\ (f\ `\ (path\text{-}image\ c)) \subseteq f\ `\ (inside(path\text{-}image\ c))$
by *auto*
have *ins2*: $\bigwedge xa.\ xa \in inside\ (path\text{-}image\ c) \Longrightarrow f\ xa \in inside\ (f\ `\ path\text{-}image\ c)$
proof −
fix *xa*
assume ∗: $xa \in inside\ (path\text{-}image\ c)$
show $f\ xa \in inside\ (f\ `\ path\text{-}image\ c)$
by (*metis* (*no-types*, *lifting*) ∗ *same-set assms*(*3*) *bij-def inj-image-mem-iff inside-def mem-Collect-eq*)
qed
then have $f\ `\ (inside(path\text{-}image\ c)) \subseteq inside\ (f\ `\ (path\text{-}image\ c))$
by *auto*
show *?thesis*
using *ins1 ins2* **by** *auto*
qed

lemma *bij-image-intersection*:
assumes $path\text{-}image\ c1 \cap path\text{-}image\ c2 = S$
assumes *bij f*
assumes $c \in path\text{-}image\ (f \circ c1) \cap path\text{-}image\ (f \circ c2)$
shows $c \in f\ `\ S$
proof −
have $c \in f\ `\ path\text{-}image\ c1 \cap f\ `\ path\text{-}image\ c2$
using *assms path-image-compose*[*of f c1*] *path-image-compose*[*of f c2*]
by *auto*
then obtain *w* **where** *c-is*: $w \in path\text{-}image\ c1 \wedge w \in path\text{-}image\ c2 \wedge c = f\ w$
using *assms* **unfolding** *bij-def inj-def surj-def*
by *auto*
then have $w \in S$
using *assms* **by** *auto*
then show $c \in f\ `\ S$
using *c-is* **by** *auto*
qed

theorem (**in** *c1-on-open-R2*) *split-inside-simple-closed-curve-locale*:
fixes $c :: real \Rightarrow 'a$
assumes *c1-simple*:*simple-path c1* **and** *c1-start*: *pathstart c1* = *a* **and** *c1-end*: *pathfinish c1* = *b*
assumes *c2-simple*: *simple-path c2* **and** *c2-start*: *pathstart c2* = *a* **and** *c2-end*: *pathfinish c2* = *b*
assumes *c-simple*: *simple-path c* **and** *c-start*: *pathstart c* = *a* **and** *c-end*: *pathfinish c* = *b*
assumes *a-neq-b*: $a \neq b$

and *c1c2*: *path-image c1* ∩ *path-image c2* = {*a*,*b*}
and *c1c*: *path-image c1* ∩ *path-image c* = {*a*,*b*}
and *c2c*: *path-image c2* ∩ *path-image c* = {*a*,*b*}
and *ne-12*: *path-image c* ∩ *inside*(*path-image c1* ∪ *path-image c2*) ≠ {}
obtains *inside*(*path-image c1* ∪ *path-image c*) ∩ *inside*(*path-image c2* ∪ *path-image c*) = {}
inside(*path-image c1* ∪ *path-image c*) ∪ *inside*(*path-image c2* ∪ *path-image c*) ∪
(*path-image c* − {*a*,*b*}) = *inside*(*path-image c1* ∪ *path-image c2*)
proof −
let *?cc1* = (*complex-of* ∘ *c1*)
let *?cc2* = (*complex-of* ∘ *c2*)
let *?cc* = (*complex-of* ∘ *c*)
have *cc1-simple*:*simple-path ?cc1*
using *bij-betw-imp-inj-on c1-simple complex-of-bij*
using *simple-path-linear-image-eq*[*OF complex-of-linear*]
by *blast*
have *cc1-start*:*pathstart ?cc1* = (*complex-of a*)
using *c1-start* **by** (*simp add*:*pathstart-compose*)
have *cc1-end*:*pathfinish ?cc1* = (*complex-of b*)
using *c1-end* **by** (*simp add*: *pathfinish-compose*)
have *cc2-simple*:*simple-path ?cc2*
using *c2-simple complex-of-bij bij-betw-imp-inj-on*
using *simple-path-linear-image-eq*[*OF complex-of-linear*]
by *blast*
have *cc2-start*:*pathstart ?cc2* = (*complex-of a*)
using *c2-start* **by** (*simp add*:*pathstart-compose*)
have *cc2-end*:*pathfinish ?cc2* = (*complex-of b*)
using *c2-end* **by** (*simp add*: *pathfinish-compose*)
have *cc-simple*:*simple-path ?cc* **using** *c-simple complex-of-bij*
using *bij-betw-imp-inj-on*
using *simple-path-linear-image-eq*[*OF complex-of-linear*]
by *blast*
have *cc-start*:*pathstart ?cc* = (*complex-of a*)
using *c-start* **by** (*simp add*:*pathstart-compose*)
have *cc-end*:*pathfinish ?cc* = (*complex-of b*)
using *c-end* **by** (*simp add*: *pathfinish-compose*)
have *ca-neq-cb*: *complex-of a* ≠ *complex-of b*
using *a-neq-b*
by (*meson bij-betw-imp-inj-on complex-of-bij inj-eq*)
have *image-set-eq1*: {*complex-of a*, *complex-of b*} ⊆ *path-image ?cc1* ∩ *path-image ?cc2*
using *c1c2 path-image-compose*[*of complex-of c1*] *path-image-compose*[*of complex-of c2*]
by *auto*
have *image-set-eq2*: ⋀*c*. *c* ∈ *path-image ?cc1* ∩ *path-image ?cc2* ⟹ *c* ∈{*complex-of a*, *complex-of b*}
using *bij-image-intersection*[*of c1 c2* {*a*, *b*} *complex-of*]
using *c1c2 complex-of-bij* **by** *auto*

have *cc1c2*: *path-image ?cc1* ∩ *path-image ?cc2* = {(*complex-of a*),(*complex-of b*)}
using *image-set-eq1 image-set-eq2* **by** *auto*
have *image-set-eq1*: {*complex-of a*, *complex-of b*} ⊆ *path-image ?cc1* ∩ *path-image ?cc*
using *c1c path-image-compose*[*of complex-of c1*] *path-image-compose*[*of complex-of c*]
by *auto*
have *image-set-eq2*: ⋀*c*. *c* ∈ *path-image ?cc1* ∩ *path-image ?cc* ⟹ *c* ∈{*complex-of a*, *complex-of b*}
using *bij-image-intersection*[*of c1 c* {*a*, *b*} *complex-of*]
using *c1c complex-of-bij* **by** *auto*
have *cc1c*: *path-image ?cc1* ∩ *path-image ?cc* = {(*complex-of a*),(*complex-of b*)}

using *image-set-eq1 image-set-eq2* **by** *auto*
have *image-set-eq1*: {*complex-of a*, *complex-of b*} ⊆ *path-image ?cc2* ∩ *path-image ?cc*
using *c2c path-image-compose*[*of complex-of c2*] *path-image-compose*[*of complex-of c*]
by *auto*
have *image-set-eq2*: ⋀*c*. *c* ∈ *path-image ?cc2* ∩ *path-image ?cc* ⟹ *c* ∈{*complex-of a*, *complex-of b*}
using *bij-image-intersection*[*of c2 c* {*a*, *b*} *complex-of*]
using *c2c complex-of-bij* **by** *auto*
have *cc2c*: *path-image ?cc2* ∩ *path-image ?cc* = {(*complex-of a*),(*complex-of b*)}
using *image-set-eq1 image-set-eq2* **by** *auto*

let *?j* = *c1* +++ (*reversepath c*)
let *?cj* = *?cc1* +++ (*reversepath ?cc*)
have *cj-and-j*: *path-image ?cj* = *complex-of* ‘ (*path-image ?j*)
by (*metis path-compose-join path-compose-reversepath path-image-compose*)
have *pathstart* (*reversepath c*) = *b*
using *c-end*
by *auto*
then have *j-path*: *path* (*c1* +++ (*reversepath c*))
using *c1-end c1-simple c-simple* **unfolding** *simple-path-def path-def*
by (*metis continuous-on-joinpaths path-def path-reversepath*)
then have *path ?j* ∧ *path-image ?j* = *path-image c1* ∪ *path-image c*
using ‹*pathstart* (*reversepath c*) = *b*› *c1-end path-image-join path-image-reversepath*
by *blast*
then have *inside*(*path-image c1* ∪ *path-image c*) = *inside*(*path-image ?j*)
by *auto*
have *pathstart* (*reversepath ?cc*) = *complex-of b*
using *cc-end*
by *auto*
then have *cj-path*: *path ?cj*
using *cc1-end cc1-simple cc-simple* **unfolding** *simple-path-def path-def*
by (*metis continuous-on-joinpaths path-def path-reversepath*)

then have *path ?cj ∧ path-image ?cj = path-image ?cc1 ∪ path-image ?cc*
by (*metis ‹pathstart (reversepath (complex-of ∘ c)) = complex-of b› cc1-end path-image-join path-image-reversepath*)
then have *ins-cj*: *inside*(*path-image ?cc1 ∪ path-image ?cc*) = *inside* (*path-image ?cj*)
by *auto*
have *inside*(*path-image ?cj*) = *complex-of ‘* (*inside*(*path-image ?j*))
using *inside-bijective-linear-image*[*of ?j complex-of*] *j-path*
using *cj-and-j complex-of-bij complex-of-linear* **by** *presburger*
then have *i1*: *inside*(*path-image ?cc1 ∪ path-image ?cc*) = *complex-of ‘* (*inside*(*path-image c1 ∪ path-image c*)) **using** *complex-of-real-of* **unfolding** *image-comp*
using *cj-and-j*
by (*simp add*: *ins-cj ‹inside (path-image c1 ∪ path-image c) = inside (path-image (c1 +++ reversepath c))›*)

let *?j2 = c2 +++* (*reversepath c*)
let *?cj2 = ?cc2 +++* (*reversepath ?cc*)
have *cj2-and-j2*: *path-image ?cj2 = complex-of ‘* (*path-image ?j2*)
by (*metis path-compose-join path-compose-reversepath path-image-compose*)
have *pathstart* (*reversepath c*) = *b*
using *c-end* **by** *auto*
then have *j2-path*: *path* (*c2 +++* (*reversepath c*))
using *c2-end c2-simple c-simple* **unfolding** *simple-path-def path-def*
by (*metis continuous-on-joinpaths path-def path-reversepath*)
then have *path ?j2 ∧ path-image ?j2 = path-image c2 ∪ path-image c*
using *‹pathstart (reversepath c) = b› c2-end path-image-join path-image-reversepath* **by** *blast*
then have *inside*(*path-image c2 ∪ path-image c*) = *inside*(*path-image ?j2*)
by *auto*
have *pathstart* (*reversepath ?cc*) = *complex-of b*
using *cc-end* **by** *auto*
then have *cj2-path*: *path ?cj2*
using *cc2-end cc2-simple cc-simple* **unfolding** *simple-path-def path-def*
by (*metis continuous-on-joinpaths path-def path-reversepath*)
then have *path ?cj2 ∧ path-image ?cj2 = path-image ?cc2 ∪ path-image ?cc*
by (*metis ‹pathstart (reversepath (complex-of ∘ c)) = complex-of b› cc2-end path-image-join path-image-reversepath*)
then have *ins-cj2*: *inside*(*path-image ?cc2 ∪ path-image ?cc*) = *inside* (*path-image ?cj2*)
by *auto*
have *inside*(*path-image ?cj2*) = *complex-of ‘* (*inside*(*path-image ?j2*))
using *inside-bijective-linear-image*[*of ?j2 complex-of*] *j2-path*
using *cj2-and-j2 complex-of-bij complex-of-linear*
by *presburger*
then have *i2*: *inside* (*path-image* (*complex-of ∘ c2*) ∪ *path-image* (*complex-of ∘ c*))
= *complex-of ‘ inside* (*path-image c2 ∪ path-image c*)
using *cj2-and-j2*

by (*simp add*: *ins-cj2* ‹*inside* (*path-image c2* ∪ *path-image c*) = *inside* (*path-image* (*c2* +++ *reversepath c*))›)

let *?j3* = *c2* +++ (*reversepath c1*)
let *?cj3* = *?cc2* +++ (*reversepath ?cc1*)
have *cj3-and-j3*: *path-image ?cj3* = *complex-of* ‘ (*path-image ?j3*)
by (*metis path-compose-join path-compose-reversepath path-image-compose*)
have *pathstart* (*reversepath c1*) = *b*
using *c1-end* **by** *auto*
then have *j3-path*: *path* (*c2* +++ (*reversepath c1*))
using *c2-end c2-simple c1-simple* **unfolding** *simple-path-def path-def*
by (*metis continuous-on-joinpaths path-def path-reversepath*)
then have *path-j3*: *path ?j3* ∧ *path-image ?j3* = *path-image c2* ∪ *path-image c1*
using ‹*pathstart* (*reversepath c1*) = *b*› *c2-end path-image-join path-image-reversepath* **by** *blast*
then have *inside*(*path-image c2* ∪ *path-image c1*) = *inside*(*path-image ?j3*)
by *auto*
have *pathstart* (*reversepath ?cc1*) = *complex-of b*
using *cc1-end* **by** *auto*
then have *cj3-path*: *path ?cj3*
using *cc2-end cc2-simple cc1-simple* **unfolding** *simple-path-def path-def*
by (*metis continuous-on-joinpaths path-def path-reversepath*)
then have *path-cj3*: *path ?cj3* ∧ *path-image ?cj3* = *path-image ?cc2* ∪ *path-image ?cc1*
by (*metis* ‹*pathstart* (*reversepath* (*complex-of* ∘ *c1*)) = *complex-of b*› *cc2-end path-image-join path-image-reversepath*)
then have *ins-cj3*: *inside*(*path-image ?cc2* ∪ *path-image ?cc1*) = *inside* (*path-image ?cj3*)
by *auto*
have *inside*(*path-image ?cj3*) = *complex-of* ‘ (*inside*(*path-image ?j3*))
using *inside-bijective-linear-image*[*of ?j3 complex-of*] *j3-path*
using *cj3-and-j3 complex-of-bij complex-of-linear*
by *presburger*
then have *i3*: *inside* (*path-image* (*complex-of* ∘ *c1*) ∪ *path-image* (*complex-of* ∘ *c2*))
= *complex-of* ‘ *inside* (*path-image c1* ∪ *path-image c2*)
by (*simp add*: *path-cj3 path-j3 sup-commute*)
obtain *y* **where** *y-prop*: *y* ∈ *path-image c* ∩ *inside* (*path-image c1* ∪ *path-image c2*)
using *ne-12* **by** *auto*
then have *y-in1*: *complex-of y* ∈ *path-image ?cc*
by (*metis IntD1 image-eqI path-image-compose*)
have *y-in2*: *complex-of y* ∈ *complex-of* ‘ (*inside* (*path-image c1* ∪ *path-image c2*))
using *y-prop* **by** *auto*
then have *cne-12*: *path-image ?cc* ∩ *inside*(*path-image ?cc1* ∪ *path-image ?cc2*) ≠ {}
using *ne-12 y-in1 y-in2 i3* **by** *force*

obtain *for-reals*: *inside*(*path-image ?cc1* ∪ *path-image ?cc*) ∩ *inside*(*path-image ?cc2* ∪ *path-image ?cc*) = {}
inside(*path-image ?cc1* ∪ *path-image ?cc*) ∪ *inside*(*path-image ?cc2* ∪ *path-image ?cc*) ∪
(*path-image ?cc* − {*complex-of a*, *complex-of b*}) = *inside*(*path-image ?cc1* ∪ *path-image ?cc2*)
using *split-inside-simple-closed-curve*[*OF cc1-simple cc1-start cc1-end cc2-simple cc2-start*
cc2-end cc-simple cc-start cc-end ca-neq-cb cc1c2 cc1c cc2c cne-12]
by *auto*
let *?rin1* = *real-of* ‘ *inside*(*path-image ?cc1* ∪ *path-image ?cc*)
let *?rin2* = *real-of* ‘ *inside*(*path-image ?cc2* ∪ *path-image ?cc*)

have *h1*: *inside*(*path-image c1* ∪ *path-image c*) ∩ *inside*(*path-image c2* ∪ *path-image c*) ≠ {} ⟹ *False*
proof−
assume *inside*(*path-image c1* ∪ *path-image c*) ∩ *inside*(*path-image c2* ∪ *path-image c*) ≠ {}
then obtain *a* **where** *a-prop*: *a* ∈ *inside*(*path-image c1* ∪ *path-image c*) ∧ *a* ∈ *inside*(*path-image c2* ∪ *path-image c*)
by *auto*
have *in1*: *complex-of a* ∈ *inside* (*path-image* (*complex-of* ∘ *c1*) ∪ *path-image* (*complex-of* ∘ *c*))
using *a-prop i1* **by** *auto*
have *in2*: *complex-of a* ∈ *inside* (*path-image* (*complex-of* ∘ *c2*) ∪ *path-image* (*complex-of* ∘ *c*))
using *a-prop i2* **by** *auto*
show *False* **using** *in1 in2 for-reals*(*1*) **by** *auto*
qed
have *h*: *path-image* (*complex-of* ∘ *c*) − {*complex-of a*, *complex-of b*} = *complex-of* ‘ (*path-image c*) − *complex-of* ‘{*a*,*b*}
using *path-image-compose* **by** *auto*
have *complex-of* ‘ *path-image c* − *complex-of* ‘ {*a*, *b*} = *complex-of* ‘ (*path-image c* − {*a*, *b*})
proof −
have ⋀*x*. *x* ∈ (*complex-of* ‘ *path-image c* − *complex-of* ‘ {*a*, *b*}) ⟷ *x* ∈ *complex-of* ‘ (*path-image c* − {*a*, *b*})
using *Diff-iff bij-betw-imp-inj-on complex-of-bij image-iff inj-eq* **by** (*smt* (*z3*))
then show *?thesis* **by** *blast*
qed
then have *path-image* (*complex-of* ∘ *c*) − {*complex-of a*, *complex-of b*} = *complex-of* ‘ (*path-image c* − {*a*,*b*})
using *h* **by** *simp*
then have *h2*: *inside*(*path-image c1* ∪ *path-image c*) ∪ *inside*(*path-image c2* ∪ *path-image c*) ∪
(*path-image c* − {*a*,*b*}) = *inside*(*path-image c1* ∪ *path-image c2*)
proof−
have ⋀*x* . *x* ∈ *inside*(*path-image c1* ∪ *path-image c2*) ⟷ *complex-of x* ∈ *complex-of* ‘ *inside* (*path-image c1* ∪ *path-image c2*)

using *i3* **by** (*metis bij-betw-imp-inj-on complex-of-bij image-iff inj-eq*)
then have *in-iff*: $\bigwedge x.\ x \in inside(path\text{-}image\ c1 \cup path\text{-}image\ c2) \longleftrightarrow complex\text{-}of\ x \in inside\ (path\text{-}image\ (complex\text{-}of \circ c1) \cup path\text{-}image\ (complex\text{-}of \circ c)) \cup$
inside (*path-image* (*complex-of* ∘ *c2*) ∪ *path-image* (*complex-of* ∘ *c*)) ∪
(*path-image* (*complex-of* ∘ *c*) − {*complex-of a*, *complex-of b*})
using *for-reals*(*2*)
using *i3* **by** *presburger*
have $\bigwedge x.$ *complex-of x* ∈ *inside* (*path-image* (*complex-of* ∘ *c1*) ∪ *path-image* (*complex-of* ∘ *c*)) ∪
inside (*path-image* (*complex-of* ∘ *c2*) ∪ *path-image* (*complex-of* ∘ *c*)) ∪
(*path-image* (*complex-of* ∘ *c*) − {*complex-of a*, *complex-of b*})
⟷ *complex-of x* ∈ *inside* (*path-image* (*complex-of* ∘ *c1*) ∪ *path-image* (*complex-of* ∘ *c*))
∨ *complex-of x* ∈ *inside* (*path-image* (*complex-of* ∘ *c2*) ∪ *path-image* (*complex-of* ∘ *c*))
∨ *complex-of x* ∈ (*path-image* (*complex-of* ∘ *c*) − {*complex-of a*, *complex-of b*})
by *blast*
then have $\bigwedge x.$ *complex-of x* ∈ *inside* (*path-image* (*complex-of* ∘ *c1*) ∪ *path-image* (*complex-of* ∘ *c*)) ∪
inside (*path-image* (*complex-of* ∘ *c2*) ∪ *path-image* (*complex-of* ∘ *c*)) ∪
(*path-image* (*complex-of* ∘ *c*) − {*complex-of a*, *complex-of b*})
⟷ *x* ∈ *inside*(*path-image c1* ∪ *path-image c*) ∪ *inside*(*path-image c2* ∪ *path-image c*) ∪
(*path-image c* − {*a*,*b*})
using *i1 i2 i3 Un-iff* ‹*path-image* (*complex-of* ∘ *c*) − {*complex-of a*, *complex-of b*} = *complex-of* ‘ (*path-image c* − {*a*, *b*})› *bij-betw-imp-inj-on complex-of-bij image-iff inj-def*
by (*smt* (*verit*, *best*))
then have $\bigwedge x.\ x \in inside(path\text{-}image\ c1 \cup path\text{-}image\ c2) \longleftrightarrow x \in (inside(path\text{-}image\ c1 \cup path\text{-}image\ c) \cup inside(path\text{-}image\ c2 \cup path\text{-}image\ c) \cup$
(*path-image c* − {*a*,*b*}))
using *in-iff* **by** *meson*
then show *?thesis* **by** *auto*
qed
show *?thesis* **using** *that h1 h2* **by** *auto*
qed

lemma *split-inside-simple-closed-curve-real2*:
fixes *c* :: *real* ⇒ *real*^2
assumes *c1-simple*:*simple-path c1* **and** *c1-start*: *pathstart c1* = *a* **and** *c1-end*: *pathfinish c1* = *b*
assumes *c2-simple*: *simple-path c2* **and** *c2-start*: *pathstart c2* = *a* **and** *c2-end*: *pathfinish c2* = *b*
assumes *c-simple*: *simple-path c* **and** *c-start*: *pathstart c* = *a* **and** *c-end*: *pathfinish c* = *b*
assumes *a-neq-b*: *a* ≠ *b*
and *c1c2*: *path-image c1* ∩ *path-image c2* = {*a*,*b*}

and *c1c*: *path-image c1* ∩ *path-image c* = {*a*,*b*}
and *c2c*: *path-image c2* ∩ *path-image c* = {*a*,*b*}
and *ne-12*: *path-image c* ∩ *inside*(*path-image c1* ∪ *path-image c2*) ≠ {}
obtains *inside*(*path-image c1* ∪ *path-image c*) ∩ *inside*(*path-image c2* ∪ *path-image c*) = {}
inside(*path-image c1* ∪ *path-image c*) ∪ *inside*(*path-image c2* ∪ *path-image c*) ∪
(*path-image c* − {*a*,*b*}) = *inside*(*path-image c1* ∪ *path-image c2*)
proof −
have *good-type*: *c1-on-open-R2-axioms TYPE*((*real*, *2*) *vec*)
unfolding *c1-on-open-R2-axioms-def* **by** *auto*
then show *?thesis*
using *c1-on-open-R2.split-inside-simple-closed-curve-locale*[*of - - - c1 a b c2 c*] *assms*
unfolding *c1-on-open-R2-def c1-on-open-euclidean-def c1-on-open-def*
using *good-type that* **by** *blast*
qed

end
theory *Polygon-Lemmas*
imports
Polygon-Jordan-Curve
HOL−Library.Sublist
HOL.Set-Interval
HOL.Fun

begin

5 Properties of make polygonal path, pathstart and pathfinish of a polygon

lemma *make-polygonal-path-induct*[*case-names Empty Single Two Multiple*]:
fixes *ell* :: (*real*^*2*) *list*
assumes *empty*: ⋀*ell*. *ell* = [] ⟹ *P ell*
and *single*: ⋀*ell*. ⟦*length ell* = *1*⟧ ⟹ *P ell*
and *two*: ⋀*ell*. ⟦*length ell* = *2*⟧ ⟹ *P ell*
and *multiple*: ⋀*ell*.
⟦*length ell* > *2*;
P ([(*ell*!*0*), (*ell*!*1*)]);
P ((*ell*!*1*)#(*drop 2 ell*))⟧ ⟹ *P ell*
shows *P ell*
apply(*induct ell rule*: *make-polygonal-path.induct*)
using *empty single two multiple* **by** *auto*

lemma *make-polygonal-path-gives-path*:
fixes *v* :: (*real*^*2*) *list*
shows *path* (*make-polygonal-path v*)
proof(*induction length v arbitrary*: *v*)

case *0*
thus *path* (*make-polygonal-path v*)
by *auto*
next
case (*Suc x*)
show *?case*
by (*smt* (*verit*, *best*) *Suc.hyps*(*1*) *Suc.hyps*(*2*) *Suc-length-conv list.distinct*(*1*) *list.inject make-polygonal-path.elims path-join-imp path-linepath pathfinish-linepath pathstart-join pathstart-linepath*)
qed

corollary *polygonal-path-is-path*:
fixes *g* :: *R-to-R2*
assumes *polygonal-path g*
shows *path g*
using *assms polygonal-path-def make-polygonal-path-gives-path* **by** *auto*

lemma *polygon-to-polygonal-path*:
fixes *p* :: *R-to-R2*
assumes *polygon p*
obtains *ell* **where** *p* = *make-polygonal-path ell*
using *assms* **unfolding** *polygon-def polygonal-path-def*
by *auto*

lemma *polygon-pathstart*:
fixes *g* :: *R-to-R2*
assumes *l* ≠ []
assumes *g* = *make-polygonal-path l*
shows *pathstart g* = *l*!*0*
using *assms make-polygonal-path.simps*
by (*smt* (*verit*) *list.discI list.expand make-polygonal-path.elims nth-Cons-0 pathstart-join pathstart-linepath*)

lemma *polygon-pathfinish*:
fixes *g* :: *R-to-R2*
assumes *l* ≠ []
assumes *g* = *make-polygonal-path l*
shows *pathfinish g* = *l*!(*length l* − *1*)
using *assms*
proof (*induct length l arbitrary*: *g l*)
case *0*
then show *?case* **by** *auto*
next
case (*Suc x*)
{assume ∗: *length l* = *1*
then obtain *a* **where** *l-is*: *l* = [*a*]
by (*metis Suc.prems*(*1*) *Suc-neq-Zero diff-Suc-1 diff-self-eq-0 length-Cons remdups-adj.cases*)

```
    then have pathfinish g = a
      using Suc make-polygonal-path.simps
      by (simp add: pathfinish-def)
    then have pathfinish g = l!(length l − 1)
      using Suc l-is
      by auto
   } moreover {assume ∗: length l = 2
    then obtain a b where l-is: l = [a, b]
        by (metis (no-types, opaque-lifting) One-nat-def Suc-eq-plus1 list.size(3)
list.size(4) min-list.cases nat.simps(1) nat.simps(3) numeral-2-eq-2)
    then have g-is: g = linepath a b
      using Suc by auto
    have pf: pathfinish g = b using g-is by auto
    then have pathfinish g = l!(length l − 1)
      using Suc ∗ l-is
      by auto
   }
   moreover {assume ∗: length l > 2
    then obtain a b c where l-is: l = a # b # c
        by (metis Suc.prems(1) Zero-neq-Suc length-Cons less-Suc0 list.size(3)
numeral-2-eq-2 remdups-adj.cases)
    then have g-is: g = (linepath a b) +++ make-polygonal-path (b # c)
      using Suc l-is
    proof −
      have c ≠ []
        using ∗ l-is by auto
      then show ?thesis
      by (metis (full-types) Suc(4) l-is list.exhaust make-polygonal-path.simps(4))
    qed
    then have pf: pathfinish g = pathfinish (make-polygonal-path (b # c))
      by auto
    have len-x: length (b # c) = x
      using l-is Suc by auto
    then have pathfinish (make-polygonal-path (b # c)) = (b # c)!(length l − 2)
      using Suc.hyps l-is
      by simp
    then have pathfinish g = l!(length l − 1)
      using l-is pf
      by auto
  }
  ultimately show ?case
    using Suc
    by (metis One-nat-def less-Suc-eq-0-disj less-antisym numeral-2-eq-2)
qed

lemma make-polygonal-path-image-property:
  assumes length vts ≥ 2
  assumes p-is-path: x ∈ path-image (make-polygonal-path vts)
  shows ∃ k < length vts − 1. x ∈ path-image (linepath (vts ! k) (vts ! (k + 1)))
```

using *assms*
proof (*induct vts*)
case *Nil*
then show *?case* **by** *auto*
next
case (*Cons a vts*)
then have *len-gteq*: *length vts* ≥ *1*
by *simp*
{assume ∗: *length vts* = *1*
then obtain *b* **where** *vts-is*: *vts* = [*b*]
by (*metis One-nat-def* ‹*1* ≤ *length vts*› *drop-eq-Nil id-take-nth-drop less-numeral-extra*(*1*) *self-append-conv2 take-eq-Nil2*)
then have *x* ∈ *path-image* (*make-polygonal-path* [*a*, *b*])
using *Cons* **by** *auto*
then have *x* ∈ *path-image* (*linepath a b*)
by *auto*
then have *x* ∈ *path-image* (*linepath* ((*a*#*vts*) ! *0*) ((*a*#*vts*) ! *1*))
using *Cons vts-is*
by *force*
then have ∃ *k*<*length* (*a* # *vts*) − *1*. *x* ∈ *path-image* (*linepath* ((*a* # *vts*) ! *k*) ((*a* # *vts*) ! (*k* + *1*)))
using ∗
by *simp*
} moreover {assume ∗: *length vts* > *1*
then obtain *b vts*′ **where** *vts-is*: *vts* = *b* # *vts*′
by (*metis One-nat-def le-zero-eq len-gteq list.exhaust list.size*(*3*) *n-not-Suc-n*)
then have *x* ∈ *path-image* ((*linepath a b*) +++ *make-polygonal-path* (*b* # *vts*′))
using *Cons*
by (*metis* (*no-types*, *lifting*) ∗ *One-nat-def length-Cons list.exhaust list.size*(*3*) *make-polygonal-path.simps*(*4*) *nat-less-le*)
then have *eo*: *x* ∈*path-image* ((*linepath a b*)) ∨ *x* ∈ *path-image* (*make-polygonal-path* (*b* # *vts*′))
using *not-in-path-image-join* **by** *blast*
{assume ∗∗ : *x* ∈*path-image* ((*linepath a b*))
then have ∃ *k*<*length* (*a* # *vts*) − *1*. *x* ∈ *path-image* (*linepath* ((*a* # *vts*) ! *k*) ((*a* # *vts*) ! (*k* + *1*)))
using *vts-is*
by *auto*
} moreover {assume ∗∗ : *x* ∈ *path-image* (*make-polygonal-path* (*b* # *vts*′))
then have ∃ *k*<*length vts* − *1*. *x* ∈ *path-image* (*linepath* (*vts* ! *k*) (*vts* ! (*k* + *1*)))
using *Cons.hyps*(*1*) ∗
by (*simp add*: *Suc-leI vts-is*)
then have ∃ *k*<*length* (*a* # *vts*) − *1*. *x* ∈ *path-image* (*linepath* ((*a* # *vts*) ! *k*) ((*a* # *vts*) ! (*k* + *1*)))

using *add.commute add-diff-cancel-left*′ *length-Cons less-diff-conv nth-Cons-Suc plus-1-eq-Suc* **by** *auto*
}

```
    ultimately have ∃k<length (a # vts) − 1. x ∈ path-image (linepath ((a #
vts) ! k) ((a # vts) ! (k + 1)))
     using eo by auto
  }
  ultimately show ?case
    using len-gteq
    by fastforce
qed

lemma linepaths-subset-make-polygonal-path-image:
  assumes length vts ≥ 2
  assumes k < length vts − 1
 shows path-image (linepath (vts ! k) (vts ! (k + 1))) ⊆ path-image (make-polygonal-path
vts)
  using assms
proof (induct vts arbitrary: k)
  case Nil
  then show ?case by auto
next
  case (Cons a vts)
  { assume ∗: length vts = 1
    then have k-is: k = 0
      using Cons.prems(2) by auto
    obtain b where vts-is: vts = [b]
      using ∗
    by (metis One-nat-def drop-eq-Nil id-take-nth-drop le-numeral-extra(4) self-append-conv2
take-eq-Nil2 zero-less-one)
    then have path-image (make-polygonal-path (a # vts)) = path-image (linepath
a b)
      by auto
    then have path-image (linepath ((a # vts) ! k) ((a # vts) ! (k + 1)))
     ⊆ path-image (make-polygonal-path (a # vts))
      using k-is vts-is
      by simp
  } moreover
  { assume ∗: length vts > 1
    then obtain b c vts′ where vts-is: vts = b#c#vts′
    by (metis diff-0-eq-0 diff-Suc-1 diff-is-0-eq leD length-Cons list.exhaust list.size(3))
    { assume ∗∗: k = 0
      then have same-path-image: path-image (linepath ((a # vts) ! k) ((a # vts)
! (k + 1))) = path-image (linepath a b)
        using vts-is
        by auto
       have path-image (linepath a b) ⊆ path-image (make-polygonal-path (a # b
#c#vts′))
        using vts-is make-polygonal-path.simps path-image-join
        by (metis (no-types, lifting) Un-iff list.discI nth-Cons-0 pathfinish-linepath
polygon-pathstart subsetI)
       then have path-image (linepath ((a # vts) ! k) ((a # vts) ! (k + 1))) ⊆
```

path-image (*make-polygonal-path* (*a* # *vts*))
using *vts-is same-path-image*
by *presburger*
} **moreover** {**assume** ∗∗: *k* > *0*
then have *k-minus-lt*: *k*−*1* < *length vts* − *1*
using *Cons*
by *auto*
then have *path-image-is*: *path-image* (*linepath* ((*a* # *vts*) ! *k*) ((*a* # *vts*) ! (*k* + *1*))) = *path-image* (*linepath* (*vts* ! (*k* −*1*)) (*vts* ! *k*))
using ∗∗
by *auto*
then have *path-im-subset1*: *path-image* (*linepath* (*vts* ! (*k*−*1*)) (*vts* ! *k*)) ⊆ *path-image* (*make-polygonal-path vts*)
using *k-minus-lt Cons.hyps*(*1*)[*of k*−*1*] ∗ ∗∗ *Suc-leI Suc-pred add.right-neutral add-Suc-right nat-1-add-1 plus-1-eq-Suc*
by *auto*
have *path-im-subset2*: *path-image* (*make-polygonal-path vts*) ⊆ *path-image* (*make-polygonal-path* (*a* # *vts*))
using *vts-is make-polygonal-path.simps*(*4*)
by (*metis dual-order.refl list.distinct*(*1*) *nth-Cons-0 path-image-join pathfinish-linepath polygon-pathstart sup.coboundedI2*)
then have *path-image* (*linepath* ((*a* # *vts*) ! *k*) ((*a* # *vts*) ! (*k* + *1*))) ⊆ *path-image* (*make-polygonal-path* (*a* # *vts*))
using *path-image-is path-im-subset1 path-im-subset2*
by *blast*
}
ultimately have *path-image* (*linepath* ((*a* # *vts*) ! *k*) ((*a* # *vts*) ! (*k* + *1*))) ⊆ *path-image* (*make-polygonal-path* (*a* # *vts*))
by *blast*
}
ultimately show *?case*
by (*metis Cons.prems*(*1*) *Suc-1 leD length-Cons linorder-neqE-nat nat-add-left-cancel-less plus-1-eq-Suc*)
qed

lemma *vertices-on-path-image*: **shows** *set vts* ⊆ *path-image* (*make-polygonal-path vts*)
proof (*induct vts rule*:*make-polygonal-path.induct*)
case *1*
then show *?case* **by** *auto*
next
case (*2 a*)
then show *?case* **by** *auto*
next
case (*3 a b*)
then show *?case* **by** *auto*
next
case (*4 a b v va*)
then have *a-in-image*: *a* ∈ *path-image* (*make-polygonal-path* (*a* # *b* # *v* # *va*))

using *make-polygonal-path.simps*
by (*metis list.distinct*(*1*) *nth-Cons-0 pathstart-in-path-image polygon-pathstart*)

have *path-image-union*:
path-image (*make-polygonal-path* (*a* # *b* # *v* # *va*))
= *path-image* (*linepath a b*) ∪ *path-image* (*make-polygonal-path* (*b* # *v* # *va*))
by (*metis make-polygonal-path.simps*(*4*) *linepath-1' list.discI nth-Cons-0 path-image-join pathfinish-def polygon-pathstart*)
have *set* (*a* # *b* # *v* # *va*) = {*a*} ∪ *set*(*b* # *v* # *va*)
by *auto*
then show *?case* **using** *a-in-image 4 make-polygonal-path.simps*
path-image-union **by** *auto*
qed

lemma *path-image-cons-union*:
assumes *p* = *make-polygonal-path vts*
assumes *p'* = *make-polygonal-path vts'*
assumes *vts'* ≠ []
assumes *vts* = *a* # *vts'* ∧ *b* = *vts'*!*0*
shows *path-image p* = *path-image* (*linepath a b*) ∪ *path-image p'*
proof−
have *pathfinish* (*linepath a b*) = *pathstart p'* **using** *assms polygon-pathstart* **by** *auto*
moreover have *length vts* = *2* ⟹ *?thesis*
by (*smt* (*verit*) *Cons-nth-drop-Suc One-nat-def Suc-1 assms*(*1*) *assms*(*2*) *assms*(*3*) *assms*(*4*) *closed-segment-idem diff-Suc-1 drop0 drop-eq-Nil insert-subset le-iff-sup le-numeral-extra*(*4*) *length-Cons length-greater-0-conv list.discI list.inject list.set*(*1*) *list.set*(*2*) *make-polygonal-path.elims path-image-linepath sup-commute vertices-on-path-image*)
moreover have *length vts* > *2* ⟹ *?thesis*
by (*metis* (*no-types*, *lifting*) *Cons-nth-drop-Suc One-nat-def Suc-1 assms*(*1*) *assms*(*2*) *assms*(*3*) *assms*(*4*) *calculation*(*1*) *drop0 drop-Suc-Cons length-greater-0-conv make-polygonal-path.simps*(*4*) *path-image-join*)
moreover have *length vts* ≥ *2* **using** *assms* **by** (*simp add*: *Suc-le-eq*)
ultimately show *?thesis* **by** *linarith*
qed

lemma *polygonal-path-image-linepath-union*:
assumes *p* = *make-polygonal-path vts*
assumes *n* = *length vts*
assumes *n* ≥ *2*
shows *path-image p* = (⋃ {*path-image* (*linepath* (*vts*!*i*) (*vts*!(*i*+*1*))) | *i*. *i* ≤ *n* − *2*})
using *assms*
proof(*induct n arbitrary*: *vts p*)
case *0*
then show *?case* **by** *linarith*
next
case (*Suc n*)
{ assume ∗: *Suc n* = *2*

then obtain *a b* **where** *ab*: *vts* = [*a*, *b*]
by (*metis Suc.prems*(*2*−*3*) *Cons-nth-drop-Suc One-nat-def Suc-1 drop0 drop-eq-Nil lessI pos2*)
then have *path-image p* = *path-image* (*linepath a b*)
using *make-polygonal-path.simps Suc.prems* **by** *presburger*
moreover have ... = (⋃ {*path-image* (*linepath* (*vts*!*i*) (*vts*!(*i*+*1*))) | *i*. *i* ≤ *Suc n* − *2*})
using *ab Suc.prems*
by (*smt* (*verit*, *ccfv-threshold*) *Suc-eq-plus1 Sup-least Sup-upper* ∗ *diff-is-0-eq diff-zero dual-order.refl mem-Collect-eq nth-Cons-0 nth-Cons-Suc subset-antisym*)
ultimately have *?case* **by** *presburger*
} moreover
{ assume ∗: *Suc n* > *2*
then obtain *a b vts*′ **where** *vts*′: *vts* = *a* # *vts*′ ∧ *b* = *vts*′!*0* ∧ *vts*′ = *tl vts*
by (*metis Suc.prems*(*2*) *list.collapse list.size*(*3*) *nat.distinct*(*1*))

let *?p*′ = *make-polygonal-path vts*′
let *?P*′ = *path-image ?p*′
let *?P* = *path-image p*
let *?P-union* = (⋃ {*path-image* (*linepath* (*vts*!*i*) (*vts*!(*i*+*1*))) | *i*. *i* ≤ *n* − *1*})

have *vts*′*-len*: *length vts*′ = *n* **using** *vts*′ *Suc.prems* **by** *fastforce*
then have *?P*′ = (⋃ {*path-image* (*linepath* (*vts*′!*i*) (*vts*′!(*i*+*1*))) | *i*. *i* ≤ *n* − *2*})
using *Suc.prems Suc.hyps* ∗ **by** *force*
moreover have ∀ *i* ≤ *n*−*2*. *vts*′!*i* = *vts*!(*i*+*1*) ∧ *vts*′!(*i*+*1*) = *vts*!(*i*+*2*) **using** *vts*′ **by** *force*
ultimately have *?P*′ = (⋃ {*path-image* (*linepath* (*vts*!(*i*+*1*)) (*vts*!(*i*+*2*))) | *i*. *i* ≤ *n* − *2*})
by *fastforce*
moreover have ... = (⋃ {*path-image* (*linepath* (*vts*!*i*) (*vts*!(*i*+*1*))) | *i*. *1* ≤ *i* ∧ *i* ≤ *n* − *1*})
(**is** ... = *?P*′*-union*)
proof−
have ⋀*x i*. *x* ∈ {*vts* ! *Suc i*−−*vts* ! *Suc* (*Suc i*)}
⟹ *i* ≤ *n* − *2*
⟹ ∃ *xa*. (∃ *i*. *xa* = {*vts* ! *i*−−*vts* ! *Suc i*} ∧ *Suc 0* ≤ *i* ∧ *i* ≤ *n* − *Suc 0*) ∧ *x* ∈ *xa*
by (*metis* ∗ *One-nat-def Suc-diff-Suc Suc-le-mono add-2-eq-Suc*′ *bot-nat-0*.*extremum diff-Suc-Suc le-add-diff-inverse plus-1-eq-Suc*)
moreover have ⋀*x i*. *x* ∈ {*vts* ! *i*−−*vts* ! *Suc i*}
⟹ *Suc 0* ≤ *i*
⟹ *i* ≤ *n* − *Suc 0*
⟹ ∃ *xa*. (∃ *i*. *xa* = {*vts* ! *Suc i*−−*vts* ! *Suc* (*Suc i*)} ∧ *i* ≤ *n* − *2*) ∧ *x* ∈ *xa*
by (*metis* ∗ *Suc-diff-Suc gr0-implies-Suc linorder-not-le not-less-eq-eq numeral-2-eq-2*)
ultimately show *?thesis* **by** *auto*
qed

moreover have *path-image* (*linepath a b*) ∪ *?P′-union* = *?P-union*
proof −
have ⋀*x*. *x* ∈ {*a*−−*b*} ⟹ ∃ *xa*. (∃ *i*. *xa* = {*vts* ! *i*−−*vts* ! *Suc i*} ∧ *i* ≤ *n* − *Suc 0*) ∧ *x* ∈ *xa*
using *vts′* **by** *fastforce*
moreover have ⋀*x i*. *x* ∈ {*vts* ! *i*−−*vts* ! *Suc i*}
⟹ ∀ *xa*. (∀ *i*≥*Suc 0*. *xa* = {*vts* ! *i*−−*vts* ! *Suc i*} ⟶ ¬ *i* ≤ *n* − *Suc 0*) ∨ *x* ∉ *xa*
⟹ *i* ≤ *n* − *Suc 0*
⟹ *x* ∈ {*a*−−*b*}
by (*metis Suc-le-eq bot-nat-0.not-eq-extremum nth-Cons-0 nth-Cons-Suc vts′*)
ultimately show *?thesis* **by** *auto*
qed
moreover have *?P* = (*path-image* (*linepath a b*)) ∪ *?P′*
using *Suc.prems vts′ path-image-cons-union*
by (*metis One-nat-def Suc-1 vts′-len bot-nat-0.extremum list.size*(*3*) *not-less-eq-eq*)
ultimately have *?case* **by** *force*
}
ultimately show *?case* **using** *Suc.prems* **by** *linarith*
qed

6 Loop Free Properties

lemma *constant-linepath-is-not-loop-free*:
shows ¬(*loop-free* ((*linepath a a*)::*real* ⇒ *real^2*))
proof −
have *all-zero1*: ⋀*x y*::*real*. (*1* − *x*) ∗$_R$ (*a*::*real^2*) + *x* ∗$_R$ *a* = *a*
by *auto*
have *all-zero2*: ⋀*x y*::*real*. (*1* − *y*) ∗$_R$ (*a*::*real^2*) + *y* ∗$_R$ *a* = *a*
by *auto*
then have ∃ *x*::*real*∈{*0..1*}. ∃ *y*::*real*∈{*0..1*}. *x* ≠ *y* ∧ (*x* = *0* ⟶ *y* ≠ *1*) ∧ (*x* = *1* ⟶ *y* ≠ *0*)
by (*metis atLeastAtMost-iff field-lbound-gt-zero less-eq-real-def linorder-not-less zero-less-one*)
then show *?thesis*
unfolding *loop-free-def linepath-def*
using *all-zero1 all-zero2* **by** *auto*
qed

lemma *doubling-back-is-not-loop-free*:
assumes *a* ≠ *b*
shows ¬(*loop-free* ((*make-polygonal-path* [*a*, *b*, *a*])::*real* ⇒ *real^2*))
proof −
let *?p1* = (*1/4*::*real*)
let *?p2* = (*3/4*::*real*)
have *same-point*: ((*linepath a b*) +++ (*linepath b a*)) (*1/4*::*real*) = ((*linepath a b*) +++ (*linepath b a*)) (*3/4*::*real*)
unfolding *linepath-def joinpaths-def* **by** *auto*

have *?p1* ∈ {*0..1*} ∧ *?p2* ∈ {*0..1*} ∧ *?p1* ≠ *?p2* ∧ (*?p1* = *0* ⟶ *?p2* ≠ *1*) ∧ (*?p1* = *1* ⟶ *?p2* ≠ *0*)
by *auto*
then have ∃ *x*∈{*0..1*}. ∃ *y*∈{*0..1*}. (*linepath a b* +++ *linepath b a*) *x* = (*linepath a b* +++ *linepath b a*) *y* ∧ *x* ≠ *y* ∧ (*x* = *0* ⟶ *y* ≠ *1*) ∧ (*x* = *1* ⟶ *y* ≠ *0*)
using *same-point* **by** *blast*
then have ¬(*loop-free* ((*linepath a b*) +++ (*linepath b a*)))
unfolding *loop-free-def* **by** *auto*
then show *?thesis* **using** *make-polygonal-path.simps*
by *auto*
qed

lemma *not-loop-free-first-component*:
assumes ¬(*loop-free p1*)
shows ¬(*loop-free* (*p1*+++*p2*))
proof −
obtain *x y* **where** *xy-prop*: *0* ≤ *x x*≤ *1 0* ≤ *y y*≤ *1 x* ≠ *y* (*x* = *0* ⟶ *y* ≠ *1*) (*x* = *1* ⟶ *y* ≠ *0*) *p1 x* = *p1 y*
using *assms* **unfolding** *loop-free-def*
by *auto*
then have *xy-prop2*: *0* ≤ *x*/*2 x*/*2*≤ *1*/*2 0* ≤ *y*/*2 y*/*2*≤ *1*/*2 x*/*2* ≠ *y*/*2*
by *auto*
then have (*p1*+++*p2*) (*x*/*2*) = (*p1*+++*p2*) (*y*/*2*)
unfolding *joinpaths-def* **using** *xy-prop*(*8*)
by *auto*
then have *props*: (*p1* +++ *p2*) (*x*/*2*) = (*p1* +++ *p2*) (*y*/*2*) ∧ (*x*/*2*) ≠ (*y*/*2*) ∧ ((*x*/*2*) = *0* ⟶ (*y*/*2*) ≠ *1*) ∧ ((*x*/*2*) = *1* ⟶ (*y*/*2*) ≠ *0*)
using *xy-prop2* **by** *auto*
have *x*/*2* ∈ {*0..1*} ∧ *y*/*2* ∈ {*0..1*}
using *xy-prop2* **by** *auto*
then have ∃ *x*∈{*0..1*}. ∃ *y*∈{*0..1*}. (*p1* +++ *p2*) *x* = (*p1* +++ *p2*) *y* ∧ *x* ≠ *y* ∧ (*x* = *0* ⟶ *y* ≠ *1*) ∧ (*x* = *1* ⟶ *y* ≠ *0*)
using *props*
by *blast*
then show *?thesis*
unfolding *loop-free-def* **by** *auto*
qed

lemma *not-loop-free-second-component*:
assumes *pathfinish-pathstart*: *pathfinish p1* = *pathstart p2*
assumes ¬(*loop-free p2*)
shows ¬(*loop-free* (*p1*+++*p2*))
proof −
obtain *x y* **where** *xy-prop*: *0* ≤ *x x*≤ *1 0* ≤ *y y*≤ *1 x* ≠ *y*

```
    (x = 0 ⟶ y ≠ 1) (x = 1 ⟶ y ≠ 0)
    p2 x = p2 y
    using assms unfolding loop-free-def
    by auto
  then have xy-prop2: (x + 1)/2 ≥ 1/2 (x + 1)/2 ≤ 1 (y + 1)/2 ≥ 1/2 (y +
1)/2 ≤ 1
  (x + 1)/2 ≠ (y + 1)/2
    by auto
  have x-same: 2*((x + 1)/2) − 1 = x
   by (metis add.right-neutral add-diff-eq cancel-comm-monoid-add-class.diff-cancel
class-dense-linordered-field.between-same mult-1 mult-2 times-divide-eq-left times-divide-eq-right)
  have y-same: 2*((y + 1)/2) − 1 = y
   by (metis add.right-neutral add-diff-eq cancel-comm-monoid-add-class.diff-cancel
class-dense-linordered-field.between-same mult-1 mult-2 times-divide-eq-left times-divide-eq-right)
  have p2 (2*((x + 1)/2) − 1) = p2 (2*((y + 1)/2) −1 )
    using xy-prop(8) x-same y-same
    by auto
  have relate-start-finish: p1 1 = p2 0
    using pathfinish-pathstart
    unfolding pathfinish-def pathstart-def
    by auto
  then have xh1: (x + 1)/2 = 1/2 ⟹ (p1 +++ p2) ((x + 1)/2) = p2 x
    unfolding joinpaths-def
    by auto
  have xh2: (x + 1)/2 > 1/2 ⟹ (p1 +++ p2) ((x + 1)/2) = p2 x
    using xy-prop2 unfolding joinpaths-def
    using x-same by force
  then have xh: (p1 +++ p2) ((x + 1)/2) = p2 x
    using xh1 xh2 xy-prop2
    by linarith
  have yh1: (y + 1)/2 = 1/2 ⟹ (p1 +++ p2) ((y + 1)/2) = p2 y
    using relate-start-finish unfolding joinpaths-def
    by auto
  have yh2: (y + 1)/2 > 1/2 ⟹ (p1 +++ p2) ((y + 1)/2) = p2 y
    using xy-prop2 unfolding joinpaths-def
    using y-same by force
  then have yh: (p1 +++ p2) ((y + 1)/2) = p2 y
    using yh1 yh2 xy-prop2
    by linarith
  then have same-eval: (p1+++p2) ((x + 1)/2)  = (p1+++p2) ((y + 1)/2)
    using xh yh xy-prop(8)
    by presburger
  have inset1: (x + 1)/2 ∈ {0..1}
    using xy-prop2
    by simp
  have inset2: (y + 1)/2 ∈ {0..1}
    using xy-prop2
    by simp
   have ∃x∈{0..1}.
```

$\exists\, y{\in}\{0..1\}.$
$(p1 \mathbin{+\!\!+\!\!+} p2)\ x = (p1 \mathbin{+\!\!+\!\!+} p2)\ y \wedge$
$x \neq y \wedge (x = 0 \longrightarrow y \neq 1) \wedge (x = 1 \longrightarrow y \neq 0)$
using *xy-prop2 same-eval inset1 inset2*
by *fastforce*
then show *?thesis*
unfolding *loop-free-def* **by** *auto*
qed

lemma *loop-free-subpath*:
assumes *path p*
assumes *u-and-v*: $u \in \{0..1\}$ $v \in \{0..1\}$ $u < v$
assumes $\neg$ (*loop-free* (*subpath u v p*))
shows $\neg$ (*loop-free p*)
proof −
have *path* (*subpath u v p*)
using *path-subpath assms* **by** *auto*
then show *?thesis* **using** *simple-path-subpath assms*
unfolding *simple-path-def*
by *blast*
qed

lemma *loop-free-associative*:
assumes *path p*
assumes *path q*
assumes *path r*
assumes *pathfinish p* = *pathstart q*
assumes *pathfinish q* = *pathstart r*
shows $\neg$ (*loop-free* ((*p* +++ *q*) +++ *r*)) $\longleftrightarrow$ $\neg$ (*loop-free* (*p* +++ (*q* +++ *r*)))
by (*metis* (*mono-tags*, *lifting*) *assms*(*1*) *assms*(*2*) *assms*(*3*) *assms*(*4*) *assms*(*5*)
path-join-imp pathfinish-join pathstart-join simple-path-assoc simple-path-def)

lemma *polygon-at-least-3-vertices*:
assumes *polygon p* **and**
p = *make-polygonal-path vts*
shows *card* (*set vts*) ≥ *3*
using *assms*
proof (*induct vts rule*: *make-polygonal-path.induct*)
case *1*
then show *?case* **unfolding** *polygon-def*
using *constant-linepath-is-not-loop-free make-polygonal-path.simps*(*1*)
by (*metis simple-path-def*)
next
case (*2 a*)
then show *?case* **unfolding** *polygon-def*
using *constant-linepath-is-not-loop-free make-polygonal-path.simps*(*2*)
by (*metis simple-path-def*)
next
case (*3 a b*)

```
  { assume ∗: a = b
    then have False using 3  unfolding polygon-def
      using constant-linepath-is-not-loop-free make-polygonal-path.simps(3)
      by (metis simple-path-def)
  } moreover {assume ∗: a ≠ b
    then have False using 3  unfolding polygon-def closed-path-def
      pathstart-def pathfinish-def using make-polygonal-path.simps(3)
      by (simp add: linepath-0′ linepath-1′)
  }
  ultimately show ?case
    by auto
next
  case (4 a b v va)
  have finset: finite (set (a # b # v # va))
    by blast
  have subset: {a, b, v} ⊆ set (a # b # v # va)
    by auto
  have neq1: a ≠ b
    using constant-linepath-is-not-loop-free not-loop-free-first-component
   by (metis 4.prems(2) make-polygonal-path.simps(4) polygon-def assms(1) sim-
ple-path-def)
  have loop-free-2: loop-free (make-polygonal-path (b # v # va))
    using 4 not-loop-free-second-component
   by (metis make-polygonal-path.simps(4) polygon-def list.distinct(1) nth-Cons-0
pathfinish-linepath polygon-pathstart simple-path-def)
  have contra: b = v ⟹ ¬(loop-free (make-polygonal-path (b # v # va)))
    using constant-linepath-is-not-loop-free[of b] make-polygonal-path.simps
    not-loop-free-first-component
    by (metis neq-Nil-conv)
  then have neq2: b ≠ v
    using loop-free-2 contra
    by auto

  have  ¬ loop-free ((linepath a b) +++ (linepath  b a))
    using doubling-back-is-not-loop-free[of a b]  neq1
    by auto
 have make-path-is: make-polygonal-path (a # b # a # va) = (linepath a b) +++
((linepath  b a) +++ (make-polygonal-path (a#va)))
    using make-polygonal-path.simps
   by (metis (no-types, opaque-lifting) 4.prems(1) 4.prems(2) closed-path-def poly-
gon-def ‹¬ loop-free (linepath a b +++ linepath b a)› linepath-1′ min-list.cases
nth-Cons-0 pathfinish-def pathfinish-join polygon-pathstart simple-path-def)
  have ¬ loop-free (((linepath a b) +++ (linepath  b a)) +++ (make-polygonal-path
(a#va)))
    using make-polygonal-path.simps not-loop-free-first-component
    using ‹¬ loop-free (linepath a b +++ linepath b a)›
    by auto
  then have ¬ loop-free (make-polygonal-path (a # b # a # va))
    using loop-free-associative
```

by (*metis make-polygonal-path-gives-path list.discI make-path-is nth-Cons-0 path-linepath pathfinish-linepath pathstart-linepath polygon-pathstart*)
then have *neq3*: $v \neq a$
using *4*
using *polygon-def simple-path-def* **by** *blast*
have *card-3*: *card* $\{a, b, v\} = 3$
using *neq1 neq2 neq3*
by *auto*
then show *?case*
using *subset finset*
by (*metis card-mono*)
qed

lemma *polygon-vertices-length-at-least-4*:
assumes *polygon p* **and**
p = *make-polygonal-path vts*
shows *length vts* $\geq$ *4*
proof −
have *card-set*: *card* (*set vts*) $\geq$ *3*
using *polygon-at-least-3-vertices assms*
by *blast*
have *len-gt3*: *length vts* $\geq$ *3*
using *card-length local.card-set order-trans* **by** *blast*
then have *non-empty*: *vts* $\neq$ []
using *card-set*
by *auto*
have *eq*: *p 0* = *p 1*
using *assms* **unfolding** *polygon-def closed-path-def pathstart-def pathfinish-def* **by** *auto*
have *p0*: *p 0* = *vts* ! *0*
using *polygon-pathstart*[*OF non-empty*] **using** *assms* **unfolding** *pathstart-def*
by *auto*
have *p1*: *p 1* = *vts* ! (*length vts* − *1*)
using *polygon-pathfinish*[*OF non-empty*] **using** *assms* **unfolding** *pathfinish-def*
by *auto*
have *vts* ! *0* = *vts* ! (*length vts* −*1*)
using *assms* **unfolding** *polygon-def*
using *p0 p1 eq* **by** *auto*
then have *set vts* = *set* (*drop 1 vts*)
using *len-gt3*
by (*smt* (*verit*, *best*) *Cons-nth-drop-Suc Suc-eq-plus1 Suc-le-eq add.commute add-0 add-leD2 drop0 dual-order.refl insert-subset last.simps last-conv-nth last-in-set list.distinct*(*1*) *list.set*(*2*) *numeral-3-eq-3 order-antisym-conv*)
then have *length* (*drop 1 vts*) $\geq$ *3*
using *card-set*
by (*metis dual-order.trans length-remdups-card-conv length-remdups-leq*)
then show *?thesis*
using *card-set*
by (*metis One-nat-def Suc-1 Suc-eq-plus1 Suc-pred add-Suc-right length-drop*

length-greater-0-conv non-empty not-less-eq-eq numeral-3-eq-3 numeral-Bit0)
qed

lemma *linepath-loop-free*:
assumes $a \neq b$
shows *loop-free* (*linepath a b*)
unfolding *loop-free-def linepath-def*
by (*smt* (*z3*) *add.assoc add.commute add-scaleR-degen assms diff-add-cancel scaleR-left-diff-distrib*)

7 Explicit Linepath Characterization of Polygonal Paths

lemma *triangle-linepath-images*:
fixes x :: *real*
assumes $vts = [a, b, c]$
assumes p = *make-polygonal-path vts*
shows $x \in \{0..1/2\} \Longrightarrow p\ x = ((linepath\ a\ b))\ (2{*}x)$
$x \in \{1/2..1\} \Longrightarrow p\ x = ((linepath\ b\ c))\ (2{*}x - 1)$
proof−
fix x :: *real*
assume $x \in \{0..1/2\}$
thus $p\ x = ((linepath\ a\ b))\ (2{*}x)$
unfolding *assms*
using *make-polygonal-path.simps*(*4*)[*of a b c Nil*] **unfolding** *joinpaths-def* **by** *presburger*
next
fix x :: *real*
assume $*$: $x \in \{1/2..1\}$
{ assume $x > 1/2$
then have $p\ x = ((linepath\ b\ c))\ (2{*}x - 1)$
unfolding *assms*
using *make-polygonal-path.simps*(*4*)[*of a b c Nil*] **unfolding** *joinpaths-def* **by** *force*
} moreover
{ assume $x = 1/2$
then have $p\ x = b \wedge ((linepath\ b\ c))\ (2{*}x - 1) = b$
unfolding *assms*
using *make-polygonal-path.simps*(*4*)[*of a b c Nil*] **unfolding** *joinpaths-def*
by (*simp add*: *linepath-def mult.commute*)
}
ultimately show $p\ x = ((linepath\ b\ c))\ (2{*}x - 1)$ **using** $*$ **by** *fastforce*
qed

lemma *polygon-linepath-images1*:
fixes n:: *nat*
assumes $n \geq 3$
assumes *length ell* = n

assumes $x \in \{0..1/2\}$
shows *make-polygonal-path ell x* = ((*linepath* (*ell* ! *0*) (*ell* ! *1*))) (*2*∗*x*)
proof −
have *make-polygonal-path ell* = *linepath* (*ell* ! *0*) (*ell* ! *1*) +++ *make-polygonal-path* (*drop 1 ell*)
using *make-polygonal-path.simps*
by (*smt* (*verit*, *del-insts*) *numeral-3-eq-3 Cons-nth-drop-Suc One-nat-def Suc-1 Suc-eq-plus1 add-Suc-right assms*(*1*) *assms*(*2*) *drop0 length-greater-0-conv less-add-Suc2 list.size*(*3*) *not-numeral-le-zero nth-Cons-0 numeral-Bit0 order-less-le-trans plus-1-eq-Suc*)
then show *?thesis*
using *assms make-polygonal-path.simps*
by (*simp add*: *joinpaths-def*)
qed

lemma *sum-insert* [*simp*]:
assumes $x \notin F$ **and** *finite F*
shows ($\sum y \in$*insert x F*. *P y*) = ($\sum y \in F$. *P y*) + *P x*
using *assms insert-def* **by**(*simp add*: *add.commute*)

lemma *sum-of-index-diff* [*simp*]:
fixes *f*:: *nat* ⇒ *'a*::*comm-monoid-add*
shows ($\sum i \in \{a..<a+b\}$. *f*(*i*−*a*)) = ($\sum i \in \{..<b\}$. *f*(*i*))
proof (*induction b*)
case *0*
then show *?case* **by** *simp*
next
case (*Suc b*)
then show *?case* **by** *simp*
qed

lemma *sum-of-index-diff2* [*simp*]:
fixes *f* :: *nat* ⇒ *'a*::*comm-monoid-add*
shows ($\sum i \in \{a+c..b+c\}$. *f*(*i*)) = ($\sum i \in \{a..b\}$. *f*(*i*+*c*))
using *Set-Interval.comm-monoid-add-class.sum.shift-bounds-cl-nat-ivl* **by** *blast*

lemma *sum-split* [*simp*]:
fixes *f* :: *nat* ⇒ *'a*::*comm-monoid-add*
assumes $c \in \{a..b\}$
shows ($\sum i \in \{a..b\}$. *f i*) = ($\sum i \in \{a..c\}$. *f i*) + ($\sum i \in \{c+1..b\}$. *f i*)
by (*metis Suc-eq-plus1 Suc-le-mono assms atLeastAtMost-iff atLeastLessThanSuc-atLeastAtMost le-SucI sum.atLeastLessThan-concat*)

lemma *summation-helper*:
fixes *x* :: *real*
fixes *k* :: *nat*
assumes *1* ≤ *k*
shows (*2*::*real*) ∗ ($\sum i = 1..k$. *1* / *2* ^ *i*) − *1* = ($\sum i = 1..(k-1)$. (*1* / (*2*^*i*)))

proof−
have *frac-cancel*: $\forall i{::}nat \geq 1.\ 2\ /\ (2\hat{}i) = 2\ /\ (2 * (2{::}real)\hat{}(i-1))$
using *power.simps*(2)[*of* 2::*real*] **by** (*metis Suc-diff-le diff-Suc-1*)
have $(2{::}real) * (\sum i = 1..k.\ 1\ /\ 2\hat{}i) = (\sum i = 1..k.\ (2\ /\ 2\hat{}i))$
by (*simp add*: *sum-distrib-left*)
also have ... $= (\sum i = 1..k.\ (2\ /\ (2 * 2\hat{}(i-1))))$ **using** *frac-cancel* **by** *simp*
also have ... $= (\sum i = 1..k.\ (1\ /\ (2\hat{}(i-1))))$ **by** *force*
also have ... $= (\sum i = 1..<(k+1).\ (1\ /\ (2\hat{}(i-1))))$
using *Suc-eq-plus1 atLeastLessThanSuc-atLeastAtMost* **by** *presburger*
also have ... $= (\sum i \in \{..<k\}.\ (1\ /\ (2\hat{}i)))$
using *sum-of-index-diff*[*of* $\lambda i.\ (1\ /\ 2\hat{}i)$ 1 *k*] **by** *simp*
finally have $(2{::}real) * (\sum i = 1..k.\ 1\ /\ 2\ \hat{}\ i) = (\sum i = 0..(k-1).\ (1\ /\ (2\hat{}i)))$
by (*metis assms atLeast0AtMost diff-Suc-1 lessThan-Suc-atMost nat-le-iff-add plus-1-eq-Suc*)
then have $(2{::}real) * (\sum i = 1..k.\ 1\ /\ 2\ \hat{}\ i) - 1 = (\sum i = 0..(k-1).\ (1\ /\ (2\hat{}i))) - 1$
by *auto*
also have ... $= (\sum i = 1..(k-1).\ (1\ /\ (2\hat{}i))) + (1/2\hat{}0) - 1$
using *sum-insert*[*of* 0 $\{1..k-1\}$ *power* (1/2)]
by (*simp add*: *Icc-eq-insert-lb-nat add.commute*)
also have ... $= (\sum i = 1..(k-1).\ (1\ /\ (2\hat{}i)))$ **by** *force*
finally show $(2{::}real) * (\sum i = 1..k.\ 1\ /\ 2\ \hat{}\ i) - 1 = (\sum i = 1..(k-1).\ (1\ /\ (2\hat{}i)))$ **.**
qed

lemma *polygon-linepath-images2*:
fixes *n k*:: *nat*
fixes *ell*:: (*real*^2) *list*
fixes *f* :: *nat* ⇒ *real* ⇒ *real*
assumes $n \geq 3$
assumes $0 \leq k \wedge k \leq n - 3$
assumes *length ell* = *n*
assumes *p*: *p* = *make-polygonal-path ell*
assumes $f = (\lambda k\ x.\ (x - (\sum i \in \{1..k\}.\ 1/(2\hat{}i))) * (2\hat{}(k+1)))$
assumes $x \in \{(\sum i \in \{1..k\}.\ 1/(2\hat{}i))..(\sum i \in \{1..(k+1)\}.\ 1/(2\hat{}i))\}$
shows *p x* = ((*linepath* (*ell* ! *k*) (*ell* ! (*k*+1)) (*f k x*)))
using *assms*
proof (*induct n arbitrary*: *ell k x p*)
case *0*
then show *?case* **by** *auto*
next
case (*Suc n*)
{ assume ∗: *k* = *0*
have *x*: $x \in \{0..1/2\}$ **using** ∗ *Suc.prems*(6) **by** *simp*
moreover have *f k x* = 2∗*x* **using** ∗ *Suc.prems*(5) **by** *simp*
ultimately have *?case*
using *polygon-linepath-images1*[*of Suc n ell x*, *OF Suc.prems*(1) *Suc.prems*(3) *x*] ∗
by (*simp add*: *Suc.prems*(4))

```
  } moreover
  { assume ∗: k ≥ 1
    then have suc-n: Suc n > 3 using Suc.prems(2) by linarith
    then have ell-is: ell = (ell!0) # (ell!1) # (ell!2) # (drop 3 ell)
      using Suc.prems(3)
     by (metis Cons-nth-drop-Suc One-nat-def Suc-1 Suc-le-lessD drop0 nat-less-le
numeral-3-eq-3)
    then have ell'-is: drop 1 ell = (ell!1) # (ell!2) # (drop 3 ell)
      by (metis One-nat-def diff-Suc-1 drop0 drop-Cons-numeral numerals(1))
    let ?ell' = drop 1 ell
    have len-ell': length ?ell' > 2 using suc-n Suc.prems(3) by simp
    let ?p' = make-polygonal-path ?ell'
    have p-tl: p = (linepath (ell ! 0) (ell ! 1)) +++ make-polygonal-path (drop 1
ell)
      using Suc.prems(4) Suc.prems(3) ∗  make-polygonal-path.simps ell-is ell'-is
      by metis

    have (∑ i = 1..k. 1 / (2 ^ i::real)) ≥ (∑ i = 1..1. 1 / (2 ^ i::real))
      using Suc.prems(2) ∗
    proof (induct k)
      case 0
      then show ?case by auto
    next
      case (Suc k)
      { assume ∗:  1 = Suc k
        then have ?case by auto
      } moreover {assume ∗:  1 < Suc k
        then have 1 ≤ k ∧ k ≤ Suc n − 3
          using Suc.prems by auto
        then have ind-h: (∑ i = 1..1. 1 / (2 ^ i::real)) ≤ (∑ i = 1..k. 1 / 2 ^ i)
          using Suc.hyps Suc.prems(2) by blast
        have (∑ i = 1..Suc k. 1 /( 2 ^ i::real)) = 1/(2^(Suc k)) + (∑ i = 1..k. 1
/ (2 ^ i::real))
          using ∗ by simp
         then have (∑ i = 1..Suc k. 1 /( 2 ^ i::real)) > (∑ i = 1..k. 1 / (2 ^
i::real))
          by simp
        then have ?case using ind-h by linarith
      }
      ultimately show ?case by linarith
    qed
    then have (∑ i = 1..k. 1 / (2 ^ i::real)) ≥ 1/2
      by auto
    then have x-gteq: x ≥ 1/2 using Suc.prems(2,6)
      by (meson atLeastAtMost-iff order-trans)
    have xonehalf: p x = ?p' (2∗x − 1) if x-is: x = 1/2 using p-tl joinpaths-def
    proof −
      have p x = (linepath (ell ! 0) (ell ! 1)) 1
        using p-tl joinpaths-def x-is
```

```
    by (metis mult.commute nle-le nonzero-divide-eq-eq zero-neq-numeral)
   then have p x = ell ! 1
    using polygon-pathfinish[of [(ell ! 0), (ell ! 1)]] unfolding pathfinish-def
    using make-polygonal-path.simps by simp
   then have p x = make-polygonal-path (drop 1 ell) 0
    using polygon-pathstart[of drop 1 ell] * len-ell' unfolding pathstart-def
    by simp
   then show ?thesis using x-is by force
  qed
  have x-gtonehalf: x > 1/2 ⟹ p x = ?p' (2*x - 1) using p-tl joinpaths-def
   by (smt (verit, ccfv-threshold))
  then have px: p x = ?p' (2*x - 1) using xonehalf x-gtonehalf x-gteq
   by linarith
  { assume k-eq: k = 1
   then have f k x = (x - (∑ i = 1..1. 1 / 2 ^ i)) * 2 ^ 2
    using Suc.prems(5) by auto
   then have fkx: f k x =  4*x - 2
    by auto
   have x ∈ {1/2..3/4}
    using k-eq Suc.prems(6) by auto
   then have 2*x - 1 ∈ {0..1/2} by simp
   then have ?p' (2*x - 1) = (linepath (?ell'!0) (?ell'!1)) (4*x - 2)
    using Suc.hyps[of k ?ell' ?p' 2*x - 1] Suc.prems
     by (smt (verit, ccfv-SIG) suc-n diff-Suc-1 leD le-Suc-eq length-drop poly-
gon-linepath-images1)
   also have ... = (linepath (ell!1) (ell!2)) (4*x - 2)
    using * Suc.prems(3)
    using ell'-is by fastforce
   also have ... = ((linepath (ell ! k) (ell ! (k+1)) (f k x))) using k-eq
     Suc.prems(5) fkx
    by (smt (verit, del-insts) nat-1-add-1)
   finally have ?case using px by simp
  } moreover
  { assume k-gt: k > 1
   then have fkminus: f (k-1) (2 * x - 1) = ((2 * x - 1) - (∑ i = 1..(k-1).
1 / 2 ^ i)) * 2 ^ k
    using Suc.prems(5) by force
   have fk: f k x = (x - (∑ i = 1..k. 1 / 2 ^ i)) * 2 ^ (k + 1)
    using Suc.prems(5) by blast
   have f-is: f (k - 1) (2 * x - 1) = f k x
   proof−
    have i: ∀ i::nat ∈ {2..k}. i - 2 + 2 = i
     by auto
    have f (k - 1) (2 * x - 1) = (2 * x - 1 - (∑ i = 1..k - 1. 1 / 2 ^ i))
* 2 ^ (k - 1 + 1)
     unfolding Suc.prems(5) by auto
    also have ... = (x - 1/2 - (∑ i = 1..k - 1. 1 / 2^i) / 2) * 2 ^ (k + 1)
     using k-gt by fastforce
    also have ... = (x - 1/2 - (∑ i = 1..k - 1. (1 / 2^i) / 2)) * 2 ^ (k + 1)
```

by (*simp add*: *sum-divide-distrib*)
also have ... = $(x - 1/2 - (\sum i = 1..k - 1.\ (1\ /\ 2)\hat{}i * 1/2)) * 2\ \hat{}\ (k + 1)$
by (*simp add*: *power-divide*)
also have ... = $(x - 1/2 - (\sum i = 1..k - 1.\ (1\ /\ 2)\hat{}(i+1))) * 2\ \hat{}\ (k + 1)$ **by** *force*
also have ... = $(x - 1/2 - (\sum i = 1..<1 + (k - 1).\ (1\ /\ 2)\hat{}(i+1))) * 2\ \hat{}\ (k + 1)$
using *Suc-eq-plus1-left atLeastLessThanSuc-atLeastAtMost* **by** *presburger*
also have ... = $(x - 1/2 - (\sum i = 1..<1 + (k - 1).\ (1\ /\ 2)\hat{}(i - 1 + 2))) * 2\ \hat{}\ (k + 1)$
by *auto*
also have ... = $(x - 1/2 - (\sum i \in \{..<k - 1\}.\ ((1\ /\ 2)\hat{}(i+2)))) * 2\ \hat{}\ (k + 1)$
using *sum-of-index-diff*[*of* $(\lambda x.\ (1/2)\hat{}(x+2))$ *1 k−1*] **by** *metis*
also have ... = $(x - 1/2 - (\sum i \in \{2..<k - 1 + 2\}.\ ((1\ /\ 2)\hat{}(i - 2 + 2)))) * 2\ \hat{}\ (k + 1)$
using *sum-of-index-diff*[*of* $(\lambda x.\ (1/2)\hat{}(x+2))$ *2 k−1*] **by** (*smt* (*verit*) *add.commute*)
also have ... = $(x - 1/2 - (\sum i \in \{2..k\}.\ ((1\ /\ 2)\hat{}(i - 2 + 2)))) * 2\ \hat{}\ (k + 1)$
using *k-gt atLeastLessThanSuc-atLeastAtMost* **by** *force*
also have ... = $(x - 1/2 - (\sum i \in \{2..k\}.\ ((1\ /\ 2)\hat{}(i)))) * 2\ \hat{}\ (k + 1)$ **using** *i* **by** *force*
also have ... = $(x - (1/2 + (\sum i \in \{2..k\}.\ ((1\ /\ 2)\hat{}(i))))) * 2\ \hat{}\ (k + 1)$ **by** *argo*
also have ... = $(x - (\sum i = 1..k.\ (1\ /\ 2)\hat{}(i))) * 2\ \hat{}\ (k + 1)$
using *sum-insert*[*of 1* $\{2..k\}$ $\lambda x.\ (1/2)\hat{}x$]
by (*smt* (*verit*, *ccfv-SIG*) *Suc-1 Suc-n-not-le-n atLeastAtMost-iff atLeastAtMost-insertL finite-atLeastAtMost k-gt less-imp-le-nat power-one-right*)
also have ... = $(x - (\sum i = 1..k.\ 1\ /\ (2\hat{}i))) * 2\ \hat{}\ (k + 1)$ **by** (*meson power-one-over*)
also have ... = *f k x* **using** *fk* **by** *argo*
finally show *?thesis* **.**
qed

have *ih1*: $3 \leq n$ **using** *suc-n* **by** *force*
have *ih2*: $0 \leq k - 1 \wedge k - 1 \leq n - 3$ **using** *k-gt Suc.prems*(*2*) *Suc.prems*(*3*) **by** *auto*
have *ih3*: *length ?ell′* = *n* **using** *Suc.prems*(*3*) **by** *auto*
have *ih4*: *?p′* = *make-polygonal-path ?ell′* **by** *blast*

have $2*x - 1 \geq (\sum i \in \{1..k-1\}.\ 1/(2\hat{}i))$
proof−
have (*2*::*real*) $* (\sum i = 1..k.\ 1\ /\ 2\ \hat{}\ i) - 1 = (\sum i = 1..(k-1).\ (1\ /\ (2\hat{}i)))$
using *summation-helper k-gt* **by** *auto*
moreover have $x \geq (\sum i = 1..k.\ 1\ /\ 2\ \hat{}\ i)$ **using** *Suc.prems*(*6*) **by** *presburger*

```
    ultimately show 2*x − 1 ≥ (∑ i ∈ {1..k−1}. 1/(2^i)) by linarith
   qed
   moreover have 2*x − 1 ≤ (∑ i ∈ {1..k}. 1/(2^i))
   proof−
     have (2::real) * (∑ i ∈ {1..(k + 1)}. 1/(2^i)) − 1 = (∑ i ∈ {1..k}.
1/(2^i))
      using summation-helper[of k + 1] k-gt by auto
     moreover have x ≤ (∑ i ∈ {1..(k + 1)}. 1/(2^i)) using Suc.prems(6)
by presburger
     ultimately show ?thesis by linarith
   qed
   ultimately have 2*x − 1 ∈ {(∑ i ∈ {1..k−1}. 1/(2^i))..(∑ i ∈ {1..k}.
1/(2^i))} by presburger
  then have ih5: 2*x − 1 ∈ {(∑ i ∈ {1..k−1}. 1/(2^i))..(∑ i ∈ {1..k−1+1}.
1/(2^i))}
     using k-gt by auto

   have p = make-polygonal-path (ell!0 # ell!1 # ell!2 # (drop 3 ell))
     using ell-is Suc.prems(4) by argo
   then have p = (linepath (ell!0) (ell!1)) +++ make-polygonal-path (ell!1 #
ell!2 # (drop 3 ell))
     using make-polygonal-path.simps by auto
  then have p x = ?p' (2*x − 1) unfolding joinpaths-def using x-gteq px by
fastforce
   also have ... = (linepath (?ell'!(k−1)) (?ell'!k)) (f (k−1) (2*x − 1))
    using Suc.hyps[OF ih1 ih2 ih3 ih4 Suc.prems(5), of 2*x − 1, OF ih5] using
k-gt by auto
   also have ... = (linepath (ell!k) (ell!(k+1))) (f (k−1) (2*x − 1))
     using Suc.prems(2) Suc.prems(3)
      by (smt (verit, del-insts) add-implies-diff ell'-is ell-is k-gt nth-Cons-pos
order-le-less-trans trans-less-add1 zero-less-one-class.zero-le-one)
   also have ... = (linepath (ell!k) (ell!(k+1))) (f k x)
     using f-is by auto
   finally have ?case .
  }
  ultimately have ?case using Suc.prems(2) * by linarith
 }
 ultimately show ?case
  using Suc.prems by linarith
qed

lemma polygon-linepath-images3:
 fixes n k:: nat
 fixes ell:: (real^2) list
 assumes n ≥ 3
 assumes length ell = n
 assumes p = make-polygonal-path ell
 assumes x ∈ {(∑ i ∈ {1..n−2}. 1/(2^i))..1}
 assumes f = (λx. (x − (∑ i ∈ {1..n−2}. 1/(2^i))) * (2^(n−2)))
```

```
  shows p x = (linepath (ell ! (n−2)) (ell ! (n−1))) (f x)
  using assms
proof (induct n arbitrary: ell k x p f)
  case 0
  then show ?case by auto
next
  case (Suc n)
  { assume ∗: Suc n = 3
    then have ell-is: ell = [ell ! 0, ell ! 1, ell ! 2]
      using Suc.prems(2)
     by (metis Cons-nth-drop-Suc One-nat-def Suc-1 cancel-comm-monoid-add-class.diff-cancel
drop0 length-0-conv length-drop lessI less-add-Suc2 numeral-3-eq-3 plus-1-eq-Suc
zero-less-Suc)
    have (∑ i = 1..(Suc n)−2. 1 / ((2 ^ i)::real)) = (∑ i∈{1}. 1 / ((2 ^ i)::real))
      by (simp add: ∗)
    then have eq1: (∑ i = 1..(Suc n)−2. 1 / ((2 ^ i)::real)) = 1/2
      by auto
    then have f-is: f = (λx. (x − (1/2)) ∗ 2) using ∗ Suc.prems(5) by auto
    have x ∈ {(1/2)::real..1} using eq1 Suc.prems(4) by metis
    moreover then have p x = linepath (ell ! 1) (ell ! 2) (2 ∗ x − 1)
      using triangle-linepath-images(2) using ell-is Suc.prems(3) by blast
    moreover have f x = 2∗x − 1 using f-is by simp
    ultimately have p x = (linepath (ell ! ((Suc n)−2)) (ell ! ((Suc n)−1))) (f x)
      using ∗ Suc.prems ell-is
      by (metis One-nat-def Suc-1 diff-Suc-1 diff-Suc-Suc numeral-3-eq-3)
  } moreover
  { assume ∗: Suc n > 3
    let ?ell′ = drop 1 ell
    let ?p′ = make-polygonal-path ?ell′
    let ?x′ = 2∗x − 1
    let ?f′ = (λx. (x − (∑ i ∈ {1..n−2}. 1/(2^i))) ∗ (2^(n−2)))
    have ell-is: ell = ell!0 # ell!1 # ell!2 # (drop 3 ell)
     by (metis ∗ Cons-nth-drop-Suc One-nat-def Suc.prems(2) Suc-1 drop0 le-Suc-eq
linorder-not-less numeral-3-eq-3 zero-less-Suc)
     then have p-tl: p = (linepath (ell ! 0) (ell ! 1)) +++ make-polygonal-path
(drop 1 ell)
      using make-polygonal-path.simps(4)[of ell!0 ell!1 ell!2 drop 3 ell]
      by (metis One-nat-def Suc.prems(3) drop-0 drop-Suc-Cons)
    have sum-split: (∑ i = 1..Suc n − 2. 1 / (2 ^ i::real)) = 1/(2^1::real) + (∑ i
= 2..Suc n − 2. 1 / (2 ^ i::real))
      using ∗
     by (metis Suc-1 Suc-eq-plus1 Suc-lessD add-le-imp-le-diff diff-Suc-Suc eval-nat-numeral(3)
less-Suc-eq-le sum.atLeast-Suc-atMost)
    let ?k = Suc n
    have helper-arith: ⋀i. i > 0 ⟹ 1 / (2 ^ i::real) > 0 by simp
    have k ≥ 2 ⟹ (∑ i = 2..k. 1 / (2 ^ i::real)) > 0 for k
    proof (induct k)
      case 0
      then show ?case by auto
```

```
  next
    case (Suc k)
    {assume *: Suc k = 2
      then have (∑ i = 2..Suc k. 1 / (2 ^ i::real)) = (∑ i = 2..2. 1 / (2 ^
i::real))
        by presburger
      then have ?case
        using helper-arith
        by (simp add: *)
    } moreover {assume *: Suc k > 2
      then have ind-h: 0 < (∑ i = 2..k. 1 / (2 ^ i::real))
        using Suc.hyps less-Suc-eq-le by blast
      have (∑ i = 2..Suc k. 1 / (2 ^ i::real)) = (∑ i = 2..k. 1 / (2 ^ i::real))
+ 1 / (2 ^ (Suc k)::real)
        using Suc.prems add.commute by auto
      then have ?case using ind-h helper-arith
        by (smt (verit) divide-less-0-1-iff zero-le-power)
    }
    ultimately show ?case
      using Suc.prems by linarith
  qed
  then  have (∑ i = 2..Suc n − 2. 1 / (2 ^ i::real)) > 0
    using * by auto
  then have (∑ i = 1..Suc n − 2. 1 / (2 ^ i::real)) > 1/2
    using sum-split by auto
  then have x > 1/2 using Suc.prems(4)
   by (smt (verit, del-insts) atLeastAtMost-iff linorder-not-le order-le-less-trans)
  then have p'x'-eq-px: ?p' ?x' = p x unfolding joinpaths-def by (simp add:
joinpaths-def p-tl)

  have 1: n ≥ 3 using * by auto
  have 2: length ?ell' = n using Suc.prems(2) by simp
  have 3: ?p' = make-polygonal-path ?ell' by auto
  have x ≤ 1 using Suc.prems(4) by auto
  then have x'-lteq: 2*x − 1 ≤ 1 by auto
  have x ≥ (∑ i = 1..Suc n − 2. 1 / 2 ^ i)
    using Suc.prems(4) by auto
  then have x'-gteq: ?x' ≥ (∑ i = 1..n − 2. 1 / 2 ^ i)
    using summation-helper[of Suc n − 2] *
   by (smt (verit) Suc.prems(1) Suc-1 Suc-diff-le Suc-leD Suc-le-mono diff-Suc-1
diff-Suc-eq-diff-pred eval-nat-numeral(3))
  have 4: ?x' ∈ {(∑ i = 1..n − 2. 1 / 2 ^ i)..1} using Suc.prems(4)
    using summation-helper[of Suc n − 2] * x'-lteq x'-gteq atLeastAtMost-iff by
blast
  have 5: ?f' = (λx. (x − (∑ i = 1..n − 2. 1 / 2 ^ i)) * 2 ^ (n − 2)) by auto
  have f x = (x − (∑ i = 1..Suc n − 2. 1 / 2 ^ i)) * 2 ^ (n − 2)*2
  proof −
    have (λr. (r − (∑ n = 1..n − 1. 1 / 2 ^ n)) * 2 ^ (n − 1)) = f
      by (simp add: Suc.prems(5))
```

then have $2 \hat{} (n - 1) * (x - (\sum n = 1..n - 1.\ 1 / 2 \hat{} n)) = f\ x$
using *Groups.mult-ac*(2) **by** *blast*
then have $(x - (\sum n = 1..n - 1.\ 1 / 2 \hat{} n)) * (2 \hat{} (n - Suc\ 1) * 2) = f\ x$
by (*metis* (*no-types*) *Groups.mult-ac*(2) *Suc.prems*(2) *diff-Suc-1 diff-Suc-Suc ell-is length-Cons power.simps*(2))
then show *?thesis*
by (*metis* (*no-types*) *Groups.mult-ac*(1) *Suc-1 diff-Suc-Suc*)
qed
then have *fx-is*: $f\ x = (2*x - 2*(\sum i = 1..Suc\ n - 2.\ 1 / 2 \hat{} i))* 2 \hat{} (n - 2)$
by *argo*
have *sum-is*: $1 + (\sum i = 1..n - 2.\ 1 /(2 \hat{} i::real)) = 2*(\sum i = 1..Suc\ n - 2.\ 1 / (2 \hat{} i::real))$
proof −
have *sum-ish1*: $(\sum i = 1..Suc\ n - 2.\ 1 / (2 \hat{} i::real)) = 1/2 + (\sum i = 2..Suc\ n - 2.\ 1 / (2 \hat{} i::real))$
by (*metis power-one-right sum-split*)
have $n \geq 2 \Longrightarrow 2*(\sum i = 2..n - 1.\ 1 / (2 \hat{} i::real)) = (\sum i = 1..n - 2.\ 1 /(2 \hat{} i::real))$
proof (*induct n*)
case *0*
then show *?case* **by** *auto*
next
case (*Suc n*)
{assume ∗: *Suc n = 2*
then have *?case* **by** *auto*
} moreover {assume ∗: *Suc n > 2*
then have *ind-h*: $2 * (\sum i = 2..n - 1.\ 1 / (2 \hat{} i::real)) = (\sum i = 1..n - 2.\ 1 / (2 \hat{} i::real))$
using *Suc* **by** *fastforce*
have *mult*: $2*1/(2\hat{}(Suc\ n - 1)::real) = 1/(2\hat{}(n - 1)::real)$
using ∗
by (*smt* (*z3*) *One-nat-def add-diff-inverse-nat bot-nat-0.not-eq-extremum diff-Suc-1 div-by-1 le-zero-eq less-Suc-eq-le mult.commute nonzero-mult-div-cancel-left nonzero-mult-divide-mult-cancel-left plus-1-eq-Suc power-Suc zero-less-numeral*)
have *sum-prop*: $\bigwedge a::nat.\ \bigwedge f::nat \Rightarrow real.(\sum i = 1..a.\ (f\ i)) + (f\ (a+1)) = (\sum i = 1..a+1.\ (f\ i))$
by *auto*
have $n - 2 + 1 = n - 1$
using ∗ **by** *auto*
then have *sum-same*: $(\sum i = 1..n - 2.\ 1 / (2 \hat{} i::real)) + 1 / 2 \hat{} (n - 1) = (\sum i = 1..n - 1.\ 1 / (2 \hat{} i::real))$
using ∗ *sum-prop*[*of* $\lambda i.\ 1 / (2 \hat{} i::real)$ $n-2$] **by** *metis*
have $2 * (\sum i = 2..Suc\ n - 1.\ 1 / (2 \hat{} i::real)) = 2 * ((\sum i = 2..n - 1.\ 1 / (2 \hat{} i::real)) + 1/(2\hat{}(Suc\ n - 1)::real))$
using ∗
by (*smt* (*z3*) *add-2-eq-Suc add-diff-inverse-nat diff-Suc-1 distrib-left-numeral ind-h not-less-eq sum.cl-ivl-Suc*)
then have $2 * (\sum i = 2..Suc\ n - 1.\ 1 / (2 \hat{} i::real)) = (\sum i = 1..n -$

*2. 1 / (2 ^ i::real)) + 2*1/(2^(Suc n − 1)::real)*
using *ind-h* **by** *argo*
then have *2 * (∑ i = 2..Suc n − 1. 1 / (2 ^ i::real)) = (∑ i = 1..n − 2. 1 / (2 ^ i::real)) + 1/(2^(n − 1)::real)*
using ** mult* **by** *auto*
then have *?case* **using** *sum-same* **by** *auto*
}
ultimately show *?case* **by** *fastforce*
qed
then have *sum-ish2:2*(∑ i = 2..Suc n − 2. 1 / (2 ^ i::real)) = (∑ i = 1..n − 2. 1 /(2 ^ i::real))*
using ** by auto*
show *?thesis* **using** *sum-ish1 sum-ish2* **by** *simp*
qed
have *?p' ?x' = (linepath (?ell' ! (n−2)) (?ell' ! (n−1))) (?f' ?x')*
using *Suc.hyps[OF 1 2 3 4 5]* **by** *blast*
moreover have *?f' ?x' = f x*
using *Suc.prems(5) fx-is sum-is*
by (*smt* (*verit*, *best*))
moreover have *?ell' ! (n−2) = ell ! ((Suc n)−2)*
by (*metis Nat.diff-add-assoc One-nat-def Suc.prems(1) Suc.prems(2) Suc-1 add-diff-cancel-left le-add1 nth-drop numeral-3-eq-3 plus-1-eq-Suc*)
moreover have *?ell' ! (n−1) = ell ! ((Suc n)−1)*
using *Suc.prems(1) Suc.prems(2)* **by** *auto*
ultimately have *?case* **using** *p'x'-eq-px* **by** *presburger*
}
ultimately show *?case* **using** *Suc.prems(1)* **by** *linarith*
qed

8 A Triangle is a Polygon

lemma *not-collinear-linepaths-intersect-helper*:
assumes *not-collinear*: *¬collinear {a,b,c}*
assumes *0 ≤ k1*
assumes *k1 ≤ 1*
assumes *0 ≤ k2*
assumes *k2 ≤ 1*
assumes *eo*: *k2 = 0 ⟹ k1 ≠ 1*
shows *¬ ((linepath a b) k1 = (linepath b c) k2)*
proof −
have *a-neq-b:a ≠ b*
using *not-collinear*
by *auto*
then have *nonz-1*: *a − b ≠ 0*
by *auto*
have *b-neq-c*: *b ≠ c*
using *not-collinear*
by *auto*
then have *nonz-2*: *b − c ≠ 0*

by *auto*
have ¬ *collinear* {*a*−*b*, *0*, *c*−*b*}
using *not-collinear*
by (*metis NO-MATCH-def collinear-3 insert-commute*)
then have *notcollinear*: ¬ *collinear* { *0*, *a*−*b*, *c*−*b*}
by (*simp add*: *insert-commute*)
have (*1* − *k1*) ∗$_R$ *a* + *k1* ∗$_R$ *b* = (*1* − *k2*) ∗$_R$ *b* + *k2* ∗$_R$ *c* ⟹ (*a* − *k1*∗$_R$ *a*) + *k1* ∗$_R$ *b* = (*b* − *k2* ∗$_R$ *b*) + *k2* ∗$_R$ *c*
by (*metis add-diff-cancel scaleR-collapse*)
then have (*1* − *k1*) ∗$_R$ *a* + *k1* ∗$_R$ *b* = (*1* − *k2*) ∗$_R$ *b* + *k2* ∗$_R$ *c* ⟹ (*1* − *k1*) ∗$_R$ *a* + *k1* ∗$_R$ *b* − *b* = − *k2* ∗$_R$ *b* + *k2* ∗$_R$ *c*
by (*metis* (*no-types*, *lifting*) *add-diff-cancel-left scaleR-collapse scaleR-minus-left uminus-add-conv-diff*)
then have (*1* − *k1*) ∗$_R$ *a* + *k1* ∗$_R$ *b* = (*1* − *k2*) ∗$_R$ *b* + *k2* ∗$_R$ *c* ⟹ (*1* − *k1*) ∗$_R$ *a* + *k1* ∗$_R$ *b* − *b* = *k2* ∗$_R$ (*c*−*b*)
by (*simp add*: *scaleR-right-diff-distrib*)
then have *rewrite*: (*1* − *k1*) ∗$_R$ *a* + *k1* ∗$_R$ *b* = (*1* − *k2*) ∗$_R$ *b* + *k2* ∗$_R$ *c* ⟹ (*1*−*k1*)∗$_R$(*a* − *b*) = *k2* ∗$_R$ (*c*−*b*)
by (*metis add-diff-cancel-right scaleR-collapse scaleR-right-diff-distrib*)
{assume ∗: *k2* ≠ *0*
then have (*1* − *k1*) ∗$_R$ *a* + *k1* ∗$_R$ *b* = (*1* − *k2*) ∗$_R$ *b* + *k2* ∗$_R$ *c* ⟹ *c* − *b* = ((*1*−*k1*)/*k2*)∗$_R$(*a* − *b*)
using *rewrite assms*(*2*−*3*)
by (*smt* (*verit*, *ccfv-SIG*) *vector-fraction-eq-iff*)
then have (*1* − *k1*) ∗$_R$ *a* + *k1* ∗$_R$ *b* = (*1* − *k2*) ∗$_R$ *b* + *k2* ∗$_R$ *c* ⟹ *collinear* {*0*, *a*−*b*, *c*−*b*}
using *collinear-lemma*[*of a* −*b c*−*b*] **by** *auto*
then have (*1* − *k1*) ∗$_R$ *a* + *k1* ∗$_R$ *b* = (*1* − *k2*) ∗$_R$ *b* + *k2* ∗$_R$ *c* ⟹ *False*
using *notcollinear* **by** *auto*
} moreover {assume ∗: *k2* = *0*
then have *k1* ≠*1*
using *assms* **by** *auto*
then have (*1* − *k1*) ∗$_R$ *a* + *k1* ∗$_R$ *b* = (*1* − *k2*) ∗$_R$ *b* + *k2* ∗$_R$ *c* ⟹ *a* − *b* = (*k2*/(*1*−*k1*)) ∗$_R$ (*c*−*b*)
using *rewrite*
by (*smt* (*verit*, *ccfv-SIG*) *vector-fraction-eq-iff*)
then have (*1* − *k1*) ∗$_R$ *a* + *k1* ∗$_R$ *b* = (*1* − *k2*) ∗$_R$ *b* + *k2* ∗$_R$ *c* ⟹ *collinear* {*0*, *a*−*b*, *c*−*b*}
using *collinear-lemma*[*of c*−*b a*−*b*]
by (*simp add*: *insert-commute*)
then have (*1* − *k1*) ∗$_R$ *a* + *k1* ∗$_R$ *b* = (*1* − *k2*) ∗$_R$ *b* + *k2* ∗$_R$ *c* ⟹ *False*
using *notcollinear* **by** *auto*
}
ultimately show *?thesis*
unfolding *linepath-def*
by *blast*
qed

lemma *not-collinear-linepaths-intersect-helper-2*:
assumes *not-collinear*: $\neg$*collinear* {*a*,*b*,*c*}
assumes *0* $\leq$ *k1*
assumes *k1* $\leq$ *1*
assumes *0* $\leq$ *k2*
assumes *k2* $\leq$ *1*
assumes *eo*: *k1* = *0* $\Longrightarrow$ *k2* $\neq$ *1*
shows $\neg$ ((*linepath a b*) *k1* = (*linepath c a*) *k2*)
using *not-collinear-linepaths-intersect-helper*[*of c a b k2 k1*] *assms*
by (*simp add*: *insert-commute*)

lemma *not-collinear-loopfree-path*: $\bigwedge$*a b c*::*real*^*2*. $\neg$*collinear* {*a*,*b*,*c*} $\Longrightarrow$ *loop-free* ((*linepath a b*) +++ (*linepath b c*))
proof −
fix *a b c*::*real*^*2*
assume *not-collinear*: $\neg$*collinear* {*a*,*b*,*c*}
then have *a-neq-b*:*a* $\neq$ *b*
by *auto*
have *b-neq-c*: *b* $\neq$ *c*
using *not-collinear*
by *auto*
have $\bigwedge$*x y*::*real*. (*linepath a b* +++ *linepath b c*) *x* = (*linepath a b* +++ *linepath b c*) *y* $\Longrightarrow$
x < *y* $\Longrightarrow$
x = *0* $\longrightarrow$ *y* $\neq$ *1* $\Longrightarrow$ *0* $\leq$ *x* $\Longrightarrow$ *x* $\leq$ *1* $\Longrightarrow$ *0* $\leq$ *y* $\Longrightarrow$ *y* $\leq$ *1* $\Longrightarrow$ *False*
proof −
fix *x y*:: *real*
assume *same-eval*: (*linepath a b* +++ *linepath b c*) *x* = (*linepath a b* +++ *linepath b c*) *y*
assume *x-neq-y*: *x* < *y*
assume *x-zero-imp*: *x* = *0* $\longrightarrow$ *y* $\neq$ *1*
assume *x-gt*: *0* $\leq$ *x*
assume *x-lt*: *x* $\leq$ *1*
assume *y-gt*: *0* $\leq$ *y*
assume *y-lt*: *y* $\leq$ *1*
{**assume** $*$: *x* $\leq$ *1/2* $\wedge$ *y* $\leq$ *1/2*
then have (*1* − *2* $*$ *x*) $*_R$ *a* + (*2* $*$ *x*) $*_R$ *b* = (*1* − *2* $*$ *y*) $*_R$ *a* + (*2* $*$ *y*) $*_R$ *b* $\Longrightarrow$ *False*
using *x-gt y-gt x-neq-y a-neq-b linepath-loop-free*[*of a b*]
by (*smt* (*z3*) *add-diff-cancel-left add-diff-cancel-right*′ *add-diff-eq scaleR-cancel-left scaleR-left-diff-distrib*)
then have *False*
using $*$ *same-eval* **unfolding** *joinpaths-def linepath-def*
by *auto*
} **moreover** {**assume** $*$: *x* > *1/2* $\wedge$ *y* > *1/2*
have *False*
using *x-lt y-lt x-neq-y b-neq-c linepath-loop-free*[*of b c*]
using $*$ *same-eval* **unfolding** *joinpaths-def linepath-def*
by (*smt* (*z3*) *add-diff-cancel-left add-diff-cancel-right*′ *add-diff-eq scaleR-cancel-left*

scaleR-collapse scaleR-left-diff-distrib)
  } **moreover** {**assume** ∗: $x \leq 1/2 \wedge y > 1/2$

    then have *lp-eq*: (*linepath a b*) (*2 ∗ x*) = (*linepath b c*) (*2 ∗ y − 1*)
      using ∗ *same-eval* **unfolding** *joinpaths-def*
      by *auto*
    have (*2 ∗ y − 1*) = *0* ⟶ (*2∗x*) ≠ *1* ∧ *0* ≤ (*2∗x*) ∧ (*2∗x*) ≤ *1* ∧ *0* ≤ (*2 ∗ y − 1*) ∧ (*2 ∗ y − 1*) ≤ *1*
      using *x-lt x-gt x-neq-y* ∗ **by** *auto*
    then have *False*
      using *lp-eq not-collinear-linepaths-intersect-helper*[*of a b c 2∗x 2 ∗ y − 1*]
      not-collinear
      using ∗ *x-gt y-lt* **by** *auto*
  }
  ultimately show *False*
    using *x-lt y-lt x-neq-y*
    by *linarith*
 qed
 then have ⋀*x y*::*real*. (*linepath a b +++ linepath b c*) *x* = (*linepath a b +++ linepath b c*) *y* ⟹
        x ≠ *y* ⟹
        x = *0* ⟶ *y* ≠ *1* ⟹ *x* = *1* ⟶ *y* ≠ *0* ⟹ *0* ≤ *x* ⟹ *x* ≤ *1* ⟹ *0* ≤ *y* ⟹ *y* ≤ *1* ⟹ *False*
  by (*metis linorder-less-linear*)
 then show *loop-free* (*linepath a b +++ linepath b c*)
  unfolding *loop-free-def*
  by (*metis atLeastAtMost-iff*)
qed

lemma *triangle-is-polygon*: ⋀*a b c*. ¬*collinear* {*a*,*b*,*c*} ⟹ *polygon* (*make-triangle a b c*)
proof −
 fix *a b c*::*real^2*
 assume *not-coll*:¬*collinear* {*a*,*b*,*c*}
 then have *a-neq-b*:*a* ≠ *b*
  by *auto*
 have *b-neq-c*: *b* ≠ *c*
  using *not-coll*
  by *auto*
 have *a-neq-c*: *c* ≠ *a*
  using *not-coll*
  using *collinear-3-eq-affine-dependent* **by** *blast*
 let *?vts* = [*a*, *b*, *c*, *a*]
 have *polygonal-path*: *polygonal-path* (*make-polygonal-path* [*a*, *b*, *c*, *a*])
  by (*metis Collect-const UNIV-I image-eqI polygonal-path-def*)
 then have *path*: *path* (*make-polygonal-path* [*a*, *b*, *c*, *a*])
  by *auto*
 then have *closed-path*: *closed-path* (*make-polygonal-path* [*a*, *b*, *c*, *a*])
  unfolding *closed-path-def* **using** *polygon-pathstart polygon-pathfinish*

by *auto*
let *?seg1* = (*linepath a b*) +++ (*linepath b c*)
have *lf1*: *loop-free* ((*linepath a b*) +++ (*linepath b c*))
using *not-collinear-loopfree-path not-coll*
by *auto*
then have ∀ *x*∈{*0..1*}. ∀ *y*∈{*0..1*}. *?seg1 x* = *?seg1 y* ⟶ *x* = *y*
using *a-neq-c* **unfolding** *loop-free-def*
by (*metis* (*no-types*, *lifting*) *path-defs*(*2*) *pathfinish-def pathfinish-join pathfinish-linepath pathstart-join pathstart-linepath*)
let *?seg2* = (*linepath b c*) +++ (*linepath c a*)
have *lf2*: *loop-free* ((*linepath b c*) +++ (*linepath c a*))
using *not-collinear-loopfree-path not-coll*
by (*simp add*: *insert-commute*)
then have ∀ *x*∈{*0..1*}. ∀ *y*∈{*0..1*}. *?seg2 x* = *?seg2 y* ⟶ *x* = *y*
using *a-neq-b* **unfolding** *loop-free-def*
by (*metis* (*no-types*, *lifting*) *path-defs*(*2*) *pathfinish-def pathfinish-join pathfinish-linepath pathstart-join pathstart-linepath*)
let *?seg3* = (*linepath c a*) +++ (*linepath a b*)
have *lf3*: *loop-free* ((*linepath c a*) +++ (*linepath a b*))
using *not-collinear-loopfree-path not-coll*
by (*simp add*: *insert-commute*)
then have ∀ *x*∈{*0..1*}. ∀ *y*∈{*0..1*}. *?seg3 x* = *?seg3 y* ⟶ *x* = *y*
using *b-neq-c* **unfolding** *loop-free-def*
by (*metis* (*no-types*, *lifting*) *path-defs*(*2*) *pathfinish-def pathfinish-join pathfinish-linepath pathstart-join pathstart-linepath*)
have *mpp-is*: ∀ *x*∈{*0..1*}. *make-polygonal-path* [*a*, *b*, *c*, *a*] *x* = ((*linepath a b*) +++ (*linepath b c*) +++ (*linepath c a*)) *x*
by *auto*
have *x-in-int1*: ∀ *x*∈{*0..*(*1*/*2*)}. *make-polygonal-path* [*a*, *b*, *c*, *a*] *x* = ((*linepath a b*)) (*2*∗*x*)
using *mpp-is*
unfolding *joinpaths-def* **by** *auto*
have *x-in-int2*: ∀ *x*∈{*1*/*2*<*..*(*3*/*4*)}. *make-polygonal-path* [*a*, *b*, *c*, *a*] *x* = ((*linepath b c*)) (*2*∗(*2*∗*x* − *1*))
using *mpp-is* **unfolding** *joinpaths-def*
by *auto*
have *x-in-int3*: ∀ *x*∈{*3*/*4*<*..1*}. *make-polygonal-path* [*a*, *b*, *c*, *a*] *x* = ((*linepath c a*)) (*2* ∗ (*2* ∗ *x* − *1*) − *1*)
using *mpp-is* **unfolding** *joinpaths-def*
by *auto*
have ⋀*x y*. *0* ≤ *x* ∧ *x* ≤ *1* ∧ *0* ≤ *y* ∧ *y* ≤ *1* ∧ *x* ≠ *y* ∧ (*x* = *0* ⟶ *y* ≠ *1*) ∧ (*x* = *1* ⟶ *y* ≠ *0*) ⟹ *make-polygonal-path* [*a*, *b*, *c*, *a*] *x* = *make-polygonal-path* [*a*, *b*, *c*, *a*] *y* ⟹ *False*
proof −
fix *x y*:: *real*
assume *big*: *0* ≤ *x* ∧ *x* ≤ *1* ∧ *0* ≤ *y* ∧ *y* ≤ *1* ∧ *x* ≠ *y* ∧ (*x* = *0* ⟶ *y* ≠ *1*) ∧ (*x* = *1* ⟶ *y* ≠ *0*)
assume *false-hyp*: *make-polygonal-path* [*a*, *b*, *c*, *a*] *x* = *make-polygonal-path* [*a*, *b*, *c*, *a*] *y*

```
    {assume ∗: x ∈ {0..(1/2)}
     then have x-eval: make-polygonal-path [a, b, c, a] x = ((linepath a b)) (2∗x)
       using x-in-int1 by auto
     {assume ∗∗: y ∈ {0..(1/2)}
          then have y-eval: make-polygonal-path [a, b, c, a] y = ((linepath a b))
(2∗y)
            using x-in-int1 by auto
           then have ((linepath a b)) (2∗x) = ((linepath a b)) (2∗y)
            using false-hyp x-eval y-eval by auto
           then have False
            using linepath-loop-free big ∗ ∗∗
            unfolding loop-free-def
                using a-neq-b add-diff-cancel-left add-diff-cancel-right′ add-diff-eq
linepath-def scaleR-cancel-left scaleR-collapse scaleR-left-diff-distrib
            by (smt (verit))
         } moreover {assume ∗∗: y ∈ {(1/2)<..(3/4)}
           then have y-eval: make-polygonal-path [a, b, c, a] y = ((linepath b c))
(2∗(2∗y − 1))
            using x-in-int2 by auto
           then have ((linepath a b)) (2∗x) = ((linepath b c)) (2∗(2∗y − 1))
            using false-hyp x-eval y-eval by auto
          then have False
              using big ∗ ∗∗ not-collinear-linepaths-intersect-helper[of a b c 2∗x
(2∗(2∗y − 1))] not-coll
           by auto
         } moreover {assume ∗∗: y ∈ {(3/4)<..1}
           then have y-eval: make-polygonal-path [a, b, c, a] y = ((linepath c a))
((2 ∗ (2 ∗ y − 1) − 1))
            using x-in-int3 by auto
           then have ((linepath a b)) (2∗x) = ((linepath c a)) ((2 ∗ (2 ∗ y − 1)
− 1))
            using false-hyp x-eval y-eval by auto
          then have False
            using big ∗ ∗∗ not-collinear-linepaths-intersect-helper-2[of a b c (2∗x)
((2 ∗ (2 ∗ y − 1) − 1))] not-coll
            by auto
         }
         ultimately have False
          using big
          by (metis atLeastAtMost-iff greaterThanAtMost-iff linorder-not-le)
       } moreover {assume ∗: x ∈ {(1/2)<..(3/4)}
          then have x-eval: make-polygonal-path [a, b, c, a] x = ((linepath b c))
(2∗(2∗x − 1))
            using x-in-int2 by auto
         {assume ∗∗: y ∈ {0..(1/2)}
          then have y-eval: make-polygonal-path [a, b, c, a] y = ((linepath a b))
(2∗y)
            using x-in-int1 by auto
         then have lp-eq: ((linepath a b)) (2∗y) = ((linepath b c)) (2∗(2∗x − 1))
```

```
      using false-hyp x-eval y-eval by auto
    have 2 * (2 * x - 1) ≠ 0
      using * by auto
    then have False
     using lp-eq big * ** not-collinear-linepaths-intersect-helper[of a b c 2*y
(2*(2*x - 1))] not-coll
      by auto
  } moreover {assume **: y ∈ {(1/2)<..(3/4)}
     then have y-eval: make-polygonal-path [a, b, c, a] y = ((linepath b c))
(2*(2*y - 1))
      using x-in-int2 by auto
       then have lp-eq: ((linepath b c)) (2*(2*y - 1)) = ((linepath b c))
(2*(2*x - 1))
      using false-hyp x-eval y-eval by auto
    then have False
      using linepath-loop-free[OF b-neq-c] big * **
       unfolding loop-free-def
       using add-diff-cancel-left add-diff-cancel-right' add-diff-eq linepath-def
scaleR-cancel-left scaleR-collapse scaleR-left-diff-distrib
       by (smt (verit) b-neq-c)
  } moreover {assume **: y ∈ {(3/4)<..1}
     then have y-eval: make-polygonal-path [a, b, c, a] y = ((linepath c a))
((2 * (2 * y - 1) - 1))
      using x-in-int3 by auto
     then have lp-eq: ((linepath b c)) (2*(2*x - 1)) = ((linepath c a)) ((2
* (2 * y - 1) - 1))
      using false-hyp x-eval y-eval
      by auto
    have not-coll2: ¬ collinear {b, c, a}
      using not-coll
      by (simp add: insert-commute)
    have 2 * (2 * x - 1) ≠ 0
      using * by auto
    then have False using lp-eq
      using big * ** not-collinear-linepaths-intersect-helper[of b c a 2*(2*x
- 1) (2 * (2 * y - 1) - 1)] not-coll2
      by auto
  }
  ultimately have False
    using big
    by (metis atLeastAtMost-iff greaterThanAtMost-iff linorder-not-le)
 } moreover {assume *: x ∈ {(3/4)<..1}
  then have x-eval: make-polygonal-path [a, b, c, a] x = ((linepath c a)) ((2
* (2 * x - 1) - 1))
       using x-in-int3 by auto
  {assume **: y ∈ {0..(1/2)}
     then have y-eval: make-polygonal-path [a, b, c, a] y = ((linepath a b))
(2*y)
      using x-in-int1 by auto
```

```
        then have lp-eq: ((linepath c a)) ((2 * (2 * x − 1) − 1)) = ((linepath
a b)) (2∗y)
          using x-eval y-eval
          using false-hyp by presburger
        have not-coll2: ¬ collinear {c, a, b}
          using not-coll
          by (simp add: insert-commute)
        have ((2 * (2 * x − 1) − 1)) ≠ 0
          using ∗ by auto
        then have False
          using lp-eq big ∗ ∗∗ not-coll2
           not-collinear-linepaths-intersect-helper[of c a b (2 * (2 * x − 1) − 1)
2∗y]
          by auto
      } moreover {assume ∗∗: y ∈ {(1/2)<..(3/4)}
        then have y-eval: make-polygonal-path [a, b, c, a] y = ((linepath b c))
(2∗(2∗y − 1))
          using x-in-int2 by auto
        then have lp-eq: ((linepath b c)) (2∗(2∗y − 1)) = ((linepath c a)) ((2
* (2 * x − 1) − 1))
          using x-eval y-eval false-hyp
          using false-hyp by presburger
        have not-coll2: ¬ collinear {b, c, a}
          using not-coll
          by (simp add: insert-commute)
        have ((2 * (2 * x − 1) − 1)) ≠ 0
          using ∗ by auto
        then have False
          using lp-eq big ∗ ∗∗ not-coll2
           not-collinear-linepaths-intersect-helper[of b c a (2∗(2∗y − 1)) (2 * (2
* x − 1) − 1)]
          by auto
      } moreover {assume ∗∗: y ∈ {(3/4)<..1}
        then have y-eval: make-polygonal-path [a, b, c, a] y = ((linepath c a))
((2 * (2 * y − 1) − 1))
          using x-in-int3 by auto
        then have ((linepath c a)) ((2 * (2 * y − 1) − 1)) = ((linepath c a))
((2 * (2 * x − 1) − 1))
          using x-eval y-eval false-hyp
          using false-hyp by presburger
        then have False
          using linepath-loop-free[OF a-neq-c] big ∗ ∗∗
           unfolding loop-free-def
           using  add-diff-cancel-left add-diff-cancel-right′ add-diff-eq linepath-def
scaleR-cancel-left scaleR-collapse scaleR-left-diff-distrib
           by (smt (verit) a-neq-c add-diff-cancel-left′)
      }
      ultimately have False
        using big
```

```
      by (metis atLeastAtMost-iff greaterThanAtMost-iff linorder-not-le)
    }
    ultimately show False using big
      by (metis atLeastAtMost-iff greaterThanAtMost-iff linorder-not-le)
  qed
  then have loop-free: loop-free (make-polygonal-path [a, b, c, a])
    unfolding loop-free-def
    by (meson atLeastAtMost-iff)
  show polygon (make-triangle a b c)
    unfolding make-triangle-def polygon-def simple-path-def
    using polygonal-path closed-path loop-free by auto
qed


lemma have-wraparound-vertex:
  assumes polygon p
  assumes p = make-polygonal-path vts
  shows vts = (take (length vts −1) vts)@[vts ! 0]
proof −
  have card (set vts) ≥ 3
    using polygon-at-least-3-vertices assms by auto
  then have nonempty: vts ≠ []
    by auto
  then have vts = (take (length vts −1) vts)@[vts ! (length vts − 1)]
    by (metis append-butlast-last-id butlast-conv-take last-conv-nth)
  then show ?thesis
    using assms(1) unfolding polygon-def closed-path-def
   using polygon-pathstart[OF nonempty assms(2)] polygon-pathfinish[OF nonempty
assms(2)]
    by presburger
qed


lemma polygon-at-least-3-vertices-wraparound:
  assumes polygon p
  assumes p = make-polygonal-path vts
  shows card (set (take (length vts −1) vts)) ≥ 3
proof −
  let ?distinct-vts = take (length vts −1) vts
  have card-vts: card (set vts) ≥ 3
    using polygon-at-least-3-vertices assms by auto
  then have vts-is: vts = ?distinct-vts@[vts ! 0]
    using have-wraparound-vertex assms by auto
  then have ?distinct-vts ≠ []
    using card-vts
   by (metis One-nat-def append-Nil distinct-card distinct-singleton eval-nat-numeral(3)
length-append-singleton list.size(3) not-less-eq-eq one-le-numeral)
  then have vts ! 0 ∈ set ?distinct-vts
    by (metis ‹vts = take (length vts − 1) vts @ [vts ! 0]› length-greater-0-conv
```

nth-append nth-mem)
then have *card* (*set ?distinct-vts*) = *card* (*set vts*)
using *vts-is*
by (*metis Un-insert-right append.right-neutral insert-absorb list.set*(*2*) *set-append*)
then show *?thesis* **using** *card-vts* **by** *auto*
qed

9 Polygon Vertex Rotation

definition *rotate-polygon-vertices*:: *'a list* ⇒ *nat* ⇒ *'a list*
where *rotate-polygon-vertices ell i* =
(*let ell1* = *rotate i* (*butlast ell*) *in ell1* @ [*ell1* ! *0*])

lemma *rotate-polygon-vertices-same-set*:
assumes *polygon* (*make-polygonal-path vts*)
shows *set* (*rotate-polygon-vertices vts i*) = *set vts*
proof −
have *card-gteq*: *card* (*set vts*) ≥ *3*
using *polygon-at-least-3-vertices assms*
by *auto*
then have *len-gteq*: *length vts* ≥ *3*
using *card-length order-trans* **by** *blast*
let *?ell1* = *rotate i* (*take* (*length vts* − *1*) *vts*)
have *inset*: *vts* ! *0* = *vts* ! (*length vts* − *1*)
using *assms polygon-pathstart polygon-pathfinish* **unfolding** *polygon-def closed-path-def*
by (*metis len-gteq list.size*(*3*) *not-numeral-le-zero*)
have *set vts* = *set* (*take* (*length vts* − *1*) *vts*) ∪ {*vts* ! (*length vts* − *1*)}
by (*metis Cons-nth-drop-Suc One-nat-def Un-insert-right assms card.empty diff-zero drop-rev length-greater-0-conv list.set*(*1*) *list.set*(*2*) *not-numeral-le-zero order.refl polygon-at-least-3-vertices rev-nth set-rev sup-bot.right-neutral take-all*)
then have *set vts* = *set* (*take* (*length vts* − *1*) *vts*)
using *inset*
by (*metis* (*no-types*, *lifting*) *One-nat-def Suc-neq-Zero Suc-pred Un-insert-right add-diff-cancel-left' butlast-conv-take diff-is-0-eq' insert-absorb len-gteq length-butlast length-greater-0-conv list.size*(*3*) *nth-mem nth-take numeral-3-eq-3 plus-1-eq-Suc sup-bot.right-neutral*)
then have *same-set*: *set vts* = *set ?ell1*
by *auto*
then have *rotate i* (*take* (*length vts* − *1*) *vts*) ! *0* ∈ *set vts*
using *len-gteq*
by (*metis card-gteq card-length le-zero-eq length-greater-0-conv list.size*(*3*) *nth-mem numeral-3-eq-3 zero-less-Suc*)
then have *set vts* = *set* (*?ell1* @ [*?ell1* ! *0*])
using *same-set* **by** *auto*
then show *?thesis*
unfolding *rotate-polygon-vertices-def*
using *card-gteq*
by (*metis butlast-conv-take*)
qed

lemma *arb-rotation-as-single-rotation*:
fixes *i*:: *nat*
shows *rotate-polygon-vertices vts* (*Suc i*) = *rotate-polygon-vertices* (*rotate-polygon-vertices vts i*) *1*
unfolding *rotate-polygon-vertices-def*
by (*metis butlast-snoc plus-1-eq-Suc rotate-rotate*)

lemma *rotation-sum*:
fixes *i j* :: *nat*
shows *rotate-polygon-vertices vts* (*i* + *j*) = *rotate-polygon-vertices* (*rotate-polygon-vertices vts i*) *j*
proof(*induct j*)
case *0*
thus *?case* **by** (*metis Nat.add-0-right butlast-snoc id-apply rotate0 rotate-polygon-vertices-def*)
next
case (*Suc j*)
have *rotate-polygon-vertices vts* (*i* + (*Suc j*)) = *rotate-polygon-vertices vts* (*Suc* (*i* + *j*)) **by** *simp*
also have ... = *rotate-polygon-vertices* (*rotate-polygon-vertices vts* (*i* + *j*)) *1*
using *arb-rotation-as-single-rotation* **by** *blast*
also have ... = *rotate-polygon-vertices* (*rotate-polygon-vertices* (*rotate-polygon-vertices vts i*) *j*) *1*
using *Suc.hyps* **by** *simp*
also have ... = *rotate-polygon-vertices* (*rotate-polygon-vertices vts i*) (*Suc j*)
using *arb-rotation-as-single-rotation* **by** *metis*
finally show *?case* **.**
qed

lemma *rotated-polygon-vertices-helper*:
fixes *p* :: *R-to-R2*
assumes *poly-p*: *polygon p*
assumes *p-is-path*: *p* = *make-polygonal-path vts*
assumes *p'-is*: *p'* = *make-polygonal-path* (*rotate-polygon-vertices vts 1*)
shows (*vts* ! *0*) = (*rotate-polygon-vertices vts 1*) ! (*length* (*rotate-polygon-vertices vts 1*) − *2*)
(*rotate-polygon-vertices vts 1*) ! (*length* (*rotate-polygon-vertices vts 1*) − *1*) = (*vts* ! *1*)
proof −
have *len-gteq*: *length vts* ≥ *3*
using *polygon-at-least-3-vertices assms*
using *card-length order-trans* **by** *blast*
let *?rotated-vts* = *rotate-polygon-vertices vts 1*
have *same-len*: *length ?rotated-vts* = *length vts*
unfolding *rotate-polygon-vertices-def* **using** *length-rotate*
by (*metis One-nat-def Suc-pred card.empty length-append-singleton length-butlast length-greater-0-conv list.set*(*1*) *not-numeral-le-zero p-is-path poly-p polygon-at-least-3-vertices*)
then have *len-rotated-gt-eq3*: *length ?rotated-vts* ≥ *3*
using *len-gteq* **by** *auto*

show *vts1*: *vts* ! *0* = *?rotated-vts* ! (*length ?rotated-vts* − *2*)
unfolding *rotate-polygon-vertices-def*
using *nth-rotate*[*of length ?rotated-vts* − *2 butlast vts 1*]
Suc-diff-Suc butlast-snoc length-butlast length-greater-0-conv lessI less-nat-zero-code list.size(*3*) *mod-self nat-1-add-1 nth-butlast plus-1-eq-Suc rotate-polygon-vertices-def same-len zero-less-diff*
by (*smt* (*z3*) *One-nat-def len-gteq length-append-singleton numeral-le-one-iff semiring-norm*(*70*))
have (*rotate 1* (*butlast vts*)) ! *0* = *vts* ! *1*
unfolding *rotate-polygon-vertices-def*
using *nth-rotate*[*of 0 butlast vts 1*] *len-gteq len-rotated-gt-eq3*
by (*metis* (*no-types*, *lifting*) *One-nat-def Suc-le-eq length-butlast less-diff-conv less-nat-zero-code mod-less not-gr-zero nth-butlast numeral-3-eq-3 plus-1-eq-Suc*)
then show *vts2*: *?rotated-vts* ! (*length ?rotated-vts* − *1*) = *vts* ! *1*
unfolding *rotate-polygon-vertices-def*
by (*smt* (*verit*, *best*) *Suc-diff-Suc Suc-eq-plus1 butlast-snoc length-butlast length-greater-0-conv less-nat-zero-code list.size*(*3*) *nth-append-length one-add-one rotate-polygon-vertices-def zero-less-diff*)
qed

lemma *rotate-polygon-vertices-same-length*:
fixes *vts* :: (*real*^*2*) *list*
assumes *length vts* ≥ *1*
shows *length vts* = *length* (*rotate-polygon-vertices vts i*)
using *assms*
proof(*induction length vts arbitrary*: *i*)
case *0*
then show *?case* **by** *auto*
next
case (*Suc x*)
then show *?case* **using** *arb-rotation-as-single-rotation*[*of vts x*]
by (*metis diff-Suc-1 length-append-singleton length-butlast length-rotate rotate-polygon-vertices-def*)
qed

lemma *rotated-polygon-vertices-helper2*:
assumes *len-gteq*: *length vts* ≥ *2*
assumes *i* < *length vts* − *1*
assumes *hd vts* = *last vts*
shows (*rotate-polygon-vertices vts 1*) ! *i* = *vts* ! (*i*+*1*)
proof −
let *?rotated-vts* = *rotate-polygon-vertices vts 1*
have *length* (*butlast vts*) = *length vts* − *1*
by *auto*
then have *same-len*: *length ?rotated-vts* = *length vts*
unfolding *rotate-polygon-vertices-def* **using** *length-rotate len-gteq*
by (*metis dual-order.trans le-add-diff-inverse length-append-singleton one-le-numeral plus-1-eq-Suc*)
then have *len-rotated-gt-eq3*: *length ?rotated-vts* ≥ *2*

using *len-gteq* **by** *auto*
let *?n = length vts*
{assume ∗: *i < length vts − 2*
then have *same-mod*: *(1 + i) mod length (butlast vts) = 1+i*
using *assms* **by** *simp*
have *i < length (butlast vts)*
using *assms* **by** *simp*
then have *rotate 1 (butlast vts) ! i = butlast vts ! (i + 1)*
using *nth-rotate*[*of i butlast vts 1*] *same-mod*
by (*metis add.commute*)
then have (*rotate-polygon-vertices vts 1*) *! i = vts ! (i+1)*
by (*metis* (*no-types*, *lifting*) *Suc-eq-plus1* ‹*i < length* (*butlast vts*)› *butlast-snoc length-butlast length-greater-0-conv less-nat-zero-code list.size*(*3*) *mod-less-divisor nth-butlast plus-1-eq-Suc rotate-polygon-vertices-def same-len same-mod*)
} moreover {assume ∗: *i = length vts − 2*
then have *same-mod*: *(1 + i) mod length (butlast vts) = 0*
using *assms*
by (*metis Suc-diff-Suc* ‹*length* (*butlast vts*) = *length vts − 1*› *length-greater-0-conv less-nat-zero-code list.size*(*3*) *mod-Suc mod-if one-add-one plus-1-eq-Suc zero-less-diff*)
have *i < length (butlast vts)*
using *assms* **by** *simp*
then have *rotate-prop*: *rotate 1 (butlast vts) ! i = butlast vts ! 0*
using *nth-rotate*[*of i butlast vts 1*] *same-mod*
by *metis*
have *butlast vts ! 0 = vts ! 0*
using *assms*(*1*)
by (*simp add*: *nth-butlast*)
then have *butlast vts ! 0 = vts ! (length vts − 1)*
by (*metis assms*(*3*) *hd-conv-nth last-conv-nth length-0-conv zero-diff*)
then have (*rotate-polygon-vertices vts 1*) *! i = vts ! (i+1)*
by (*metis ∗ rotate-prop Suc-diff-Suc Suc-eq-plus1* ‹*butlast vts ! 0 = vts ! 0*› *add-2-eq-Suc′ le-add-diff-inverse2 len-gteq less-add-Suc2 one-add-one same-len butlast-snoc length-butlast lessI nth-butlast rotate-polygon-vertices-def*)
}
ultimately show *?thesis*
using *assms*(*2*) **by** *linarith*
qed

lemma *polygon-rotation-t-translation1*:
assumes *polygon-of p vts*
assumes *p′ = make-polygonal-path* (*rotate-polygon-vertices vts 1*)
(**is** *p′ = make-polygonal-path ?vts′*)
assumes *x′ ∈ {(∑ i ∈ {1..k}. 1/(2^i))..(∑ i ∈ {1..k+1}. 1/(2^i))}*
assumes *n = length vts*
assumes *0 ≤ k ∧ k ≤ n − 4*
assumes *l = x′ − (∑ i ∈ {1..k}. 1/(2^i))*
assumes *x = l/2 + (∑ i ∈ {1..(k + 1)}. 1/(2^i))*
shows *x ∈ {(∑ i ∈ {1..k+1}. 1/(2^i))..(∑ i ∈ {1..k+2}. 1/(2^i))}*
p′ x′ = p x

proof−
let *?f* = λ(*k*::*nat*) (*x*::*real*). (*x* − (∑ *i* ∈ {*1*..*k*}. *1*/(*2*^*i*))) ∗ (*2*^(*k*+*1*))
have *x* ≥ (∑ *i* ∈ {*1*..*k*+*1*}. *1*/(*2*^*i*))
proof−
have *l* ≥ *0* **using** *assms*(*3*,*6*) **by** *auto*
then show *?thesis* **using** *assms*(*7*) **by** *linarith*
qed
moreover have *x* ≤ (∑ *i* ∈ {*1*..*k*+*2*}. *1*/(*2*^*i*))
proof−
have *x'* ≤ (∑ *i* ∈ {*1*..*k*+*1*}. *1*/(*2*^*i*)) **using** *assms*(*3*) **by** *presburger*
then have *l* ≤ (∑ *i* ∈ {*1*..*k*+*1*}. *1*/(*2*^*i*)) − (∑ *i* ∈ {*1*..*k*}. *1*/(*2*^*i*)) **using** *assms*(*6*) **by** *argo*
also have ... = (*1*/*2*^(*k*+*1*)) + (∑ *i* ∈ {*1*..*k*}. *1*/(*2*^*i*)) − (∑ *i* ∈ {*1*..*k*}. *1*/(*2*^*i*))
using *sum-insert*[*of k*+*1* {*1*..*k*} λ*i*. *1*/(*2*^*i*)]
by (*smt* (*verit*) *Suc-eq-plus1 Suc-n-not-le-n add.commute atLeastAtMost-Suc-conv atLeastAtMost-iff finite-atLeastAtMost le-add2 one-add-one*)
also have ... = (*1*/*2*^(*k*+*1*)) **by** *argo*
finally have *l* ≤ (*1*/*2*^(*k*+*1*)) **.**
then have *x* ≤ (*1*/*2*^(*k*+*1*))/*2* + (∑ *i* ∈ {*1*..*k*+*1*}. *1*/(*2*^*i*)) **using** *assms*(*7*) **by** *simp*
also have ... = *1*/*2*^(*k*+*2*) + (∑ *i* ∈ {*1*..*k*+*1*}. *1*/(*2*^*i*)) **by** *simp*
also have ... = (∑ *i* ∈ {*1*..*k*+*2*}. *1*/(*2*^*i*))
using *sum-insert*[*of k*+*2* {*1*..*k*+*2*} λ*i*. *1*/(*2*^*i*)] **by** *simp*
finally show *?thesis* **.**
qed
ultimately show *x*: *x* ∈ {(∑ *i* ∈ {*1*..*k*+*1*}. *1*/(*2*^*i*))..(∑ *i* ∈ {*1*..*k*+*2*}. *1*/(*2*^*i*))} **by** *presburger*
have *1*: *n* ≥ *4* **using** *polygon-vertices-length-at-least-4 assms*
using *polygon-of-def* **by** *blast*
then have *2*: *length vts* = *length ?vts'*
using *assms rotate-polygon-vertices-same-length* **by** *auto*
then have *3*: *length ?vts'* = *n* **using** *assms* **by** *auto*

have *p' x'* = ((*linepath* (*?vts'* ! *k*) (*?vts'* ! (*k*+*1*)) (*?f k x'*)))
using *polygon-linepath-images2*[*of n k ?vts' p' ?f x'*] *assms*(*2*,*3*,*5*) *1 3* **by** *fastforce*
moreover have *p x* = ((*linepath* (*vts* ! (*k*+*1*)) (*vts* ! (*k*+*2*)) (*?f* (*k*+*1*) *x*)))
using *polygon-linepath-images2*[*of n k*+*1 vts p ?f x*] *assms*(*2*,*3*,*5*) *1 2 3 x*
by (*smt* (*verit*, *ccfv-threshold*) *Nat.diff-add-assoc add.commute add-diff-cancel-left add-le-imp-le-left add-left-mono assms*(*1*) *nat-add-1-add-1 one-plus-numeral polygon-of-def semiring-norm*(*2*) *semiring-norm*(*4*) *trans-le-add1*)
moreover have *?vts'* ! *k* = *vts* ! (*k*+*1*)
using *rotated-polygon-vertices-helper2*
by (*smt* (*verit*, *best*) *1 Nat.le-diff-conv2 Suc-pred' add-leD1 assms*(*1*) *assms*(*4*) *assms*(*5*) *diff-diff-cancel diff-less have-wraparound-vertex hd-conv-nth leD length-greater-0-conv less-Suc-eq nat-less-le numeral-Bit0 numeral-eq-one-iff polygon-of-def semiring-norm*(*83*) *snoc-eq-iff-butlast zero-less-numeral*)
moreover have *?vts'* ! (*k*+*1*) = *vts* ! (*k*+*2*)

using *rotated-polygon-vertices-helper2*[*of vts k+1*]
by (*metis* (*no-types*, *lifting*) *assms*(*1*,*4*,*5*) *1 One-nat-def Suc-diff-Suc add-Suc-right diff-zero have-wraparound-vertex hd-conv-nth le-add-diff-inverse2 less-add-Suc2 nat-less-le not-less-eq-eq numeral-Bit0 one-add-one plus-1-eq-Suc polygon-of-def snoc-eq-iff-butlast*)
moreover have *?f k x′ = ?f* (*k+1*) *x* **using** *assms*(*6*) *assms*(*7*) **by** *force*
ultimately show *p′ x′ = p x* **by** *presburger*
qed

lemma *polygon-rotation-t-translation1-strict*:
assumes *polygon-of p vts*
assumes *p′ = make-polygonal-path* (*rotate-polygon-vertices vts 1*)
(**is** *p′ = make-polygonal-path ?vts′*)
assumes *x′* ∈ {(∑ *i* ∈ {*1..k*}. *1*/(*2*^*i*))..<(∑ *i* ∈ {*1..k+1*}. *1*/(*2*^*i*))}
assumes *n = length vts*
assumes *0* ≤ *k* ∧ *k* ≤ *n* − *4*
assumes *l = x′* − (∑ *i* ∈ {*1..k*}. *1*/(*2*^*i*))
assumes *x = l*/*2* + (∑ *i* ∈ {*1..*(*k* + *1*)}. *1*/(*2*^*i*))
shows *x* ∈ {(∑ *i* ∈ {*1..k+1*}. *1*/(*2*^*i*))..<(∑ *i* ∈ {*1..k+2*}. *1*/(*2*^*i*))}
p′ x′ = p x
proof −
let *?f* = λ(*k*::*nat*) (*x*::*real*). (*x* − (∑ *i* ∈ {*1..k*}. *1*/(*2*^*i*))) ∗ (*2*^(*k+1*))
have *x* ≥ (∑ *i* ∈ {*1..k+1*}. *1*/(*2*^*i*))
proof−
have *l* ≥ *0* **using** *assms*(*3*,*6*) **by** *auto*
then show *?thesis* **using** *assms*(*7*) **by** *linarith*
qed
moreover have *x* < (∑ *i* ∈ {*1..k+2*}. *1*/(*2*^*i*))
proof−
have *x′* < (∑ *i* ∈ {*1..k+1*}. *1*/(*2*^*i*)) **using** *assms*(*3*) **by** *auto*
then have *l* < (∑ *i* ∈ {*1..k+1*}. *1*/(*2*^*i*)) − (∑ *i* ∈ {*1..k*}. *1*/(*2*^*i*)) **using** *assms*(*6*) **by** *argo*
also have ... = (*1*/*2*^(*k+1*)) + (∑ *i* ∈ {*1..k*}. *1*/(*2*^*i*)) − (∑ *i* ∈ {*1..k*}. *1*/(*2*^*i*))
using *sum-insert*[*of k+1* {*1..k*} λ*i*. *1*/(*2*^*i*)]
by (*smt* (*verit*) *Suc-eq-plus1 Suc-n-not-le-n add.commute atLeastAtMost-Suc-conv atLeastAtMost-iff finite-atLeastAtMost le-add2 one-add-one*)
also have ... = (*1*/*2*^(*k+1*)) **by** *argo*
finally have *l* < (*1*/*2*^(*k+1*)) **.**
then have *x* < (*1*/*2*^(*k+1*))/*2* + (∑ *i* ∈ {*1..k+1*}. *1*/(*2*^*i*)) **using** *assms*(*7*) **by** *simp*
also have ... = *1*/*2*^(*k+2*) + (∑ *i* ∈ {*1..k+1*}. *1*/(*2*^*i*)) **by** *simp*
also have ... = (∑ *i* ∈ {*1..k+2*}. *1*/(*2*^*i*))
using *sum-insert*[*of k+2* {*1..k+2*} λ*i*. *1*/(*2*^*i*)] **by** *simp*
finally show *?thesis* **.**
qed
ultimately show *x* ∈ {(∑ *i* ∈ {*1..k+1*}. *1*/(*2*^*i*))..<(∑ *i* ∈ {*1..k+2*}. *1*/(*2*^*i*))}
by *auto*
show *p′ x′ = p x*
using *assms*(*3*) *polygon-rotation-t-translation1*[*OF assms*(*1*) *assms*(*2*) - *assms*(*4*)

assms(5) *assms*(6) *assms*(7)]
by (*meson atLeastAtMost-iff atLeastLessThan-iff less-eq-real-def*)
qed

lemma *polygon-rotation-t-translation2*:
assumes *polygon-of p vts*
assumes $p' = $ *make-polygonal-path* (*rotate-polygon-vertices vts 1*)
(**is** $p' = $ *make-polygonal-path ?vts'*)
assumes $n = $ *length vts*
assumes $x' \in \{(\sum i \in \{1..(n-3)\}.\ 1/(2\hat{}i))..(\sum i \in \{1..(n-2)\}.\ 1/(2\hat{}i))\}$
assumes $x = x' + 1/(2\hat{}(n-2))$
shows $x \in \{(\sum i \in \{1..n-2\}.\ 1/(2\hat{}i))..1\}$
$p'\ x' = p\ x$
proof−
let *?k* $= n-3$
let *?f'* $= (\lambda(k{::}nat)\ x{::}real.\ (x - (\sum i \in \{1..k\}.\ 1/(2\hat{}i))) * (2\hat{}(k+1)))$
have *n-geq-4*: $n \geq 4$ **using** *polygon-vertices-length-at-least-4 assms*
using *polygon-of-def* **by** *blast*
moreover then have *same-len*: *length vts* = *length ?vts'*
using *assms rotate-polygon-vertices-same-length*[*of vts*] **by** *auto*
moreover then have *length ?vts'* $= n$ **using** *assms*(3) **by** *auto*
ultimately have *p'x'*: $p'\ x' = ((linepath\ (?vts' !\ ?k)\ (?vts' !\ (?k+1))\ (?f'\ ?k\ x')))$
using *polygon-linepath-images2*[*of n ?k ?vts' p' ?f' x'*] *assms*
by (*smt* (*verit*, *ccfv-threshold*) *One-nat-def Suc-diff-Suc diff-diff-left diff-is-0-eq' le-add2 le-add-diff-inverse2 linorder-not-le nat-le-linear numeral-3-eq-3 numeral-Bit0 numeral-le-iff numeral-le-one-iff numerals*(1) *one-plus-numeral plus-1-eq-Suc trans-le-add2*)
let *?f* $= (\lambda x{::}real.\ (x - (\sum i \in \{1..n-2\}.\ 1/(2\hat{}i))) * (2\hat{}(n-2)))$
have *sum-prop*: $\bigwedge i{::}nat.\ \bigwedge f{::}nat \Rightarrow real.\ (\sum i = 1..i.\ f\ i) + f\ (i+1) = (\sum i = 1..i+1.\ f\ i)$
by *auto*
have *sum-upto*: $(\sum i = 1..n-3.\ 1\ /\ (2\ \hat{}\ i{::}real)) + 1\ /\ 2\ \hat{}\ (n-2) = (\sum i = 1..n-2.\ 1\ /\ (2\ \hat{}\ i{::}real))$
using *sum-prop*[*of* $\lambda i.\ 1\ /\ (2\ \hat{}\ i{::}real)\ n-3$] *n-geq-4*
by (*smt* (*verit*, *del-insts*) *Nat.add-diff-assoc2 add-numeral-left diff-cancel2 le-add-diff-inverse le-numeral-extra*(4) *nat-1-add-1 nat-add-left-cancel-le numeral-Bit1 numerals*(1) *semiring-norm*(2) *semiring-norm*(8) *trans-le-add1*)
have $x' \geq (\sum i = 1..?k.\ 1\ /\ 2\ \hat{}\ i)$
using *assms* **by** *presburger*
then have *x-geq*: $x \geq (\sum i \in \{1..n-2\}.\ 1/(2\hat{}i))$
using *assms*(5) *sum-upto*
by *linarith*
have $x' \leq (\sum i = 1..n-2.\ 1\ /\ 2\ \hat{}\ i)$
using *assms*(4) **by** *auto*
then have *x-leq*: $x \leq 1$
using *assms*(5)
by (*smt* (*verit*, *del-insts*) *add.left-commute add-diff-cancel-left' diff-diff-eq le-add-diff-inverse2 le-numeral-extra*(4) *n-geq-4 nat-add-1-add-1 numeral-Bit0 numeral-Bit1 sum-upto summation-helper trans-le-add2*)

show $x \in \{(\sum i \in \{1..n-2\}.\ 1/(2\hat{}i))..1\}$
using *x-geq x-leq*
by *auto*
then have *px*: $p\ x = (linepath\ (vts\ !\ (n-2))\ (vts\ !\ (n-1)))\ (?f\ x)$
using *polygon-linepath-images3*[*of n vts p x ?f*] *n-geq-4 assms polygon-of-def*
by *fastforce*
moreover have $?vts'\ !\ (n-3) = vts\ !\ (n-2)$
using *n-geq-4 assms*(3) *rotated-polygon-vertices-helper2 assms*(1−3)
unfolding *polygon-of-def*
by (*smt* (*verit*) *One-nat-def Suc-diff-Suc add.commute diff-is-0-eq diff-less dual-order.trans have-wraparound-vertex hd-conv-nth le-add-diff-inverse length-greater-0-conv linorder-not-le nat-1-add-1 not-add-less2 numeral-3-eq-3 plus-1-eq-Suc pos2 rotated-polygon-vertices-helper*(1) *same-len snoc-eq-iff-butlast*)
moreover have $?vts'\ !\ (n-2) = vts\ !\ (n-1)$
using *n-geq-4 assms*(3) *assms*
unfolding *polygon-of-def*
by (*metis closed-path-def list.size*(3) *not-numeral-le-zero polygon-def polygon-pathfinish polygon-pathstart rotated-polygon-vertices-helper*(1) *same-len*)
moreover have $?f'\ ?k\ x' = ?f\ x$ **using** *assms*(4−5) *n-geq-4*
by (*smt* (*verit*, *del-insts*) *One-nat-def Suc-diff-Suc Suc-eq-plus1 add-diff-cancel-right' add-numeral-left le-antisym linorder-not-le numeral-3-eq-3 numeral-code*(2) *numerals*(1) *semiring-norm*(2) *sum-upto trans-le-add2*)
ultimately show $p'\ x' = p\ x$ **using** *px p'x'*
by (*smt* (*verit*, *ccfv-SIG*) *Nat.add-diff-assoc2 assms*(5) *diff-cancel2 le-add-diff-inverse le-add-diff-inverse2 le-numeral-extra*(4) *n-geq-4 nat-1-add-1 numeral-Bit0 numeral-Bit1 trans-le-add1*)
qed

lemma *polygon-rotation-t-translation2-strict*:
assumes *polygon-of p vts*
assumes $p' = make\text{-}polygonal\text{-}path\ (rotate\text{-}polygon\text{-}vertices\ vts\ 1)$
(**is** $p' = make\text{-}polygonal\text{-}path\ ?vts'$)
assumes $n = length\ vts$
assumes $x' \in \{(\sum i \in \{1..(n-3)\}.\ 1/(2\hat{}i))..<(\sum i \in \{1..(n-2)\}.\ 1/(2\hat{}i))\}$
assumes $x = x' + 1/(2\hat{}(n-2))$
shows $x \in \{(\sum i \in \{1..n-2\}.\ 1/(2\hat{}i))..<1\}$
$p'\ x' = p\ x$
proof −
have *n-geq-4*: $n \geq 4$ **using** *polygon-vertices-length-at-least-4 assms*
using *polygon-of-def* **by** *blast*
have *sum-prop*: $\bigwedge i{::}nat.\ \bigwedge f{::}nat \Rightarrow real.\ (\sum i = 1..i.\ f\ i) + f\ (i+1) = (\sum i = 1..i+1.\ f\ i)$
by *auto*
have *sum-upto*: $(\sum i = 1..n-3.\ 1\ /\ (2\ \hat{}\ i{::}real)) + 1\ /\ 2\ \hat{}\ (n-2) = (\sum i = 1..n-2.\ 1\ /\ (2\ \hat{}\ i{::}real))$
using *sum-prop*[*of* $\lambda i.\ 1\ /\ (2\ \hat{}\ i{::}real)$ *n−3*] *n-geq-4*
by (*smt* (*verit*, *del-insts*) *Nat.add-diff-assoc2 add-numeral-left diff-cancel2 le-add-diff-inverse le-numeral-extra*(4) *nat-1-add-1 nat-add-left-cancel-le numeral-Bit1 numerals*(1) *semir-*

ing-norm(2) semiring-norm(8) trans-le-add1)
have *x-geq*: $x \geq (\sum i \in \{1..n-2\}.\ 1/(2\hat{}i))$
using *assms(4) polygon-rotation-t-translation2*[*OF assms(1) assms(2) assms(3)*
- *assms(5)*]
by *simp*
have $x' < (\sum i = 1..n - 2.\ 1\ /\ 2\ \hat{}\ i)$
using *assms(4)* **by** *auto*
then have *x-leq*: $x < 1$
using *assms(5)*
by (*smt* (*verit*, *del-insts*) *add.left-commute add-diff-cancel-left' diff-diff-eq le-add-diff-inverse2*
le-numeral-extra(4) n-geq-4 nat-add-1-add-1 numeral-Bit0 numeral-Bit1 sum-upto
summation-helper trans-le-add2)
show $x \in \{(\sum i \in \{1..n-2\}.\ 1/(2\hat{}i))..<1\}$
using *x-geq x-leq* **by** *auto*
show $p'\ x' = p\ x$
using *assms(4) polygon-rotation-t-translation2*[*OF assms(1) assms(2) assms(3)*
- *assms(5)*]
by (*meson atLeastAtMost-iff atLeastLessThan-iff less-eq-real-def*)
qed

lemma *polygon-rotation-t-translation3*:
assumes *polygon-of p vts*
assumes $p' =$ *make-polygonal-path* (*rotate-polygon-vertices vts 1*)
(**is** $p' =$ *make-polygonal-path ?vts'*)
assumes $x' \in \{(\sum i \in \{1..n-2\}.\ 1/(2\hat{}i))..1\}$
assumes $n =$ *length vts*
assumes $l = x' - (\sum i \in \{1..n-2\}.\ 1/(2\hat{}i))$
assumes $x = l * (2\hat{}(n-3))$
shows $x \in \{0..1/2\}$
$p'\ x' = p\ x$
proof−
let *?f* $= (\lambda x{::}real.\ (x - (\sum i \in \{1..n-2\}.\ 1/(2\hat{}i))) * (2\hat{}(n-2)))$
have *n-geq-4*: $n \geq 4$ **using** *polygon-vertices-length-at-least-4 assms*
using *polygon-of-def* **by** *blast*
moreover then have *same-len*: *length vts* = *length ?vts'*
using *assms rotate-polygon-vertices-same-length* **by** *auto*
moreover have *length-vts'*: *length ?vts'* $= n$
using *assms(4) same-len* **by** *auto*
ultimately have *p'x'*: $p'\ x' =$ (*linepath* (*?vts'* ! $(n-2)$) (*?vts'* ! $(n-1)$)) (*?f x'*)
using *polygon-linepath-images3*[*of n ?vts' p' x' ?f*] *assms*
unfolding *polygon-of-def* **by** *fastforce*

have *x-is*: $x = (x' - (\sum i = 1..n - 2.\ 1\ /\ 2\ \hat{}\ i)) * 2\ \hat{}\ (n - 3)$
using *assms(5−6)* **by** *auto*
then have *x-gt*: $x \geq 0$
using *assms(3)* **by** *simp*
have *sum-prop*: $k \geq 1 \Longrightarrow 1 - (\sum i = 1..k.\ 1\ /\ (2\ \hat{}\ i{::}real)) = 1/(2\hat{}k)$ **for** k
proof (*induct k*)
case *0*

```
  then show ?case by auto
 next
  case (Suc k)
  { assume *: Suc k = 1
   then have ?case by auto
  } moreover
  { assume *: Suc k > 1
   then have  1 - (∑ i = 1..k. 1 / (2 ^ i::real)) = 1 / 2 ^ k
    using Suc by linarith
   then have ?case by simp
  }
  ultimately show ?case
   by linarith
 qed
 have x' ≤ 1
  using assms(3) by auto
 then have x ≤ (1 - (∑ i = 1..n - 2. 1 / (2 ^ i::real))) * 2 ^ (n - 3)
  using x-is
  using mult-right-mono zero-le-power by fastforce
 then have x ≤ 1/(2^(n-2))*2^(n-3)
  using sum-prop n-geq-4
  by auto
 then have x-lt: x ≤ 1/2
  using n-geq-4
  by (smt (verit, ccfv-SIG) One-nat-def Suc-1 Suc-diff-Suc add-diff-cancel-right'
diff-is-0-eq dual-order.trans linorder-not-le nonzero-mult-divide-mult-cancel-right2
numeral-3-eq-3 numeral-code(2) power.simps(2) power-commutes power-not-zero
times-divide-eq-left zero-neq-numeral)
 then show x ∈ {0..1/2}
  using x-gt x-lt by auto
 moreover have n ≥ 3 using n-geq-4 by auto
 ultimately have px: p x = (linepath (vts ! 0) (vts ! 1)) (2 * x)
   using polygon-linepath-images1[of n vts] assms unfolding polygon-of-def by
blast

 have ?vts' ! (n-2) = vts ! 0 ∧ ?vts' ! (n-1) = vts ! 1
  unfolding rotate-polygon-vertices-def
   by (metis length-vts' assms(1) polygon-of-def rotate-polygon-vertices-def ro-
tated-polygon-vertices-helper(1) rotated-polygon-vertices-helper(2))
 moreover have ?f x' = 2 * x
 proof-
  have 2 * x = 2 * (x' - (∑ i ∈ {1..n-2}. 1/(2^i))) * (2^(n-3)) using assms
by auto
  moreover have ... = (x' - (∑ i ∈ {1..n-2}. 1/(2^i))) * (2^(n-2))
   using n-geq-4 Suc-1 Suc-diff-Suc Suc-le-eq bot-nat-0.not-eq-extremum diff-Suc-1
le-antisym mult.left-commute mult.right-neutral mult-cancel-left not-less-eq-eq num-double
numeral-3-eq-3 numeral-eq-Suc numeral-times-numeral power.simps(2) pred-numeral-simps(2)
zero-less-diff zero-neq-numeral
  proof -
```

```
    have f1: ∀ r ra. (ra::real) * r = r * ra
      by simp
    have f2: ∀ r n ra. (r::real) * (r ^ n * ra) = r ^ Suc n * ra
      by simp
    have f3: pred-numeral (num.Bit1 num.One) = Suc (Suc 0)
      by simp
    have f4: Suc 0 = 1
      by linarith
    have Suc 1 < n
      using n-geq-4 by linarith
    then have 2 * ((x' - (∑ n = 1..n - Suc 1. 1 / 2 ^ n)) * 2 ^ (n - 3)) =
(x' - (∑ n = 1..n - Suc 1. 1 / 2 ^ n)) * 2 ^ (n - Suc 1)
      using f4 f3 f2 f1 Suc-diff-Suc numeral-eq-Suc by presburger
    then show ?thesis
      by (metis (no-types) Suc-1 mult.assoc)
  qed
  moreover have ... = ?f x' by auto
  ultimately show ?thesis by presburger
 qed
 ultimately show p' x' = p x using p'x' px by auto
qed

lemma polygon-rotation-t-translation3-strict:
 assumes polygon-of p vts
 assumes p' = make-polygonal-path (rotate-polygon-vertices vts 1)
   (is p' = make-polygonal-path ?vts')
 assumes x' ∈ {(∑ i ∈ {1..n-2}. 1/(2^i))..<1}
 assumes n = length vts
 assumes l = x' - (∑ i ∈ {1..n-2}. 1/(2^i))
 assumes x = l * (2^(n-3))
 shows x ∈ {0..<1/2}
       p' x' = p x
proof -
 have n-geq-4: n ≥ 4 using polygon-vertices-length-at-least-4 assms
   using polygon-of-def by blast
 have x-is: x = (x' - (∑ i = 1..n - 2. 1 / 2 ^ i)) * 2 ^ (n - 3)
   using assms(5-6) by auto
 then  have x-gt: x ≥ 0
   using assms(3) by simp
 have sum-prop: k ≥ 1 ⟹ 1 - (∑ i = 1..k. 1 / (2 ^ i::real)) = 1/(2^k) for k
 proof (induct k)
   case 0
   then show ?case by auto
 next
   case (Suc k)
   { assume * :Suc k = 1
     then have ?case by auto
   } moreover
   { assume *: Suc k > 1
```

```
    then have  1 - (∑ i = 1..k. 1 / (2 ^ i::real)) = 1 / 2 ^ k
      using Suc by linarith
    then have ?case by simp
   }
   ultimately show ?case
    by linarith
 qed
 have x' < 1
   using assms(3) by auto
 then have x < (1 - (∑ i = 1..n - 2. 1 / (2 ^ i::real))) * 2 ^ (n - 3)
   using x-is
   using mult-right-mono zero-le-power by fastforce
 then have x < 1/(2^(n-2))*2^(n-3)
   using sum-prop n-geq-4
   by auto
 then have x-lt: x < 1/2
   using n-geq-4
   by (smt (verit, ccfv-SIG) One-nat-def Suc-1 Suc-diff-Suc add-diff-cancel-right'
diff-is-0-eq dual-order.trans linorder-not-le nonzero-mult-divide-mult-cancel-right2
numeral-3-eq-3 numeral-code(2) power.simps(2) power-commutes power-not-zero
times-divide-eq-left zero-neq-numeral)
 show x ∈ {0..<1/2}
   using x-lt x-gt by auto
 show p' x' = p x
  using assms(3) polygon-rotation-t-translation3[OF assms(1) assms(2) - assms(4)
assms(5) assms(6)]
   by simp
qed

lemma f-gteq-0-sum-gt: ⋀f::nat ⇒ real. (⋀i::nat. (f i) > 0) ⟹ a > b ⟹ (∑ i
= 1..a. (f i)) > (∑ i = 1..b. (f i)) for a b :: nat
 proof (induct a arbitrary: b)
   case 0
   then show ?case by auto
 next
   case (Suc a)
   {assume *: b = a
     then have sum f {1..(Suc a)} = sum f {1.. b} + f (Suc a)
       by force
     then have ?case
       using Suc(2)[of Suc a] * by linarith
   } moreover {assume *: b < a
     then have ?case using Suc
    by (smt (verit, ccfv-threshold) Suc-eq-plus1 dual-order.trans le-add2 sum.nat-ivl-Suc')
   }
   ultimately show ?case
     using Suc.prems(2) less-antisym by blast
 qed
```

lemma *rotation-intervals-disjoint*:
assumes *k1* $\neq$ *k2*
shows $\{\sum i = 1..k1.\ 1\ /\ (2\ \hat{}\ i::real)..<\sum i = 1..k1+1.\ 1\ /\ 2\ \hat{}\ i\} \cap \{\sum i = 1..k2.\ 1\ /\ (2\ \hat{}\ i::real)..<\sum i = 1..k2+1.\ 1\ /\ 2\ \hat{}\ i\} = \{\}$
proof −
have *lambda-gt*: $(\bigwedge i.\ 0 < 1\ /\ (2\ \hat{}\ i::real))$
by *simp*
have *h1*: *?thesis* **if** ∗:*k1* < *k2*
proof −
have *eo*: *k1+1* ≤ *k2*
using ∗ **by** *auto*
have *k1+1* = *k2* $\Longrightarrow$ $(\sum i = 1..k1+1.\ 1\ /\ 2\ \hat{}\ i) \leq (\sum i = 1..k2.\ 1\ /\ (2\ \hat{}\ i::real))$
by *auto*
have $(\sum i = 1..k1+1.\ 1\ /\ 2\ \hat{}\ i) \leq (\sum i = 1..k2.\ 1\ /\ (2\ \hat{}\ i::real))$ **if** ∗∗: *k1+1* < *k2*
using *f-gteq-0-sum-gt*[*OF lambda-gt* ∗∗]
using *less-eq-real-def* **by** *presburger*
then have $(\sum i = 1..k1+1.\ 1\ /\ 2\ \hat{}\ i) \leq (\sum i = 1..k2.\ 1\ /\ (2\ \hat{}\ i::real))$
using ∗ *eo* **by** *fastforce*
then show *?thesis* **by** *auto*
qed
have *h2*: *?thesis* **if** ∗: *k2* < *k1*
proof −
have *eo*: *k2+1* ≤ *k1*
using ∗ **by** *auto*
have *k2+1* = *k1* $\Longrightarrow$ $(\sum i = 1..k2+1.\ 1\ /\ 2\ \hat{}\ i) \leq (\sum i = 1..k1.\ 1\ /\ (2\ \hat{}\ i::real))$
by *auto*
have $(\sum i = 1..k2+1.\ 1\ /\ 2\ \hat{}\ i) \leq (\sum i = 1..k1.\ 1\ /\ (2\ \hat{}\ i::real))$ **if** ∗∗: *k2+1* < *k1*
using *f-gteq-0-sum-gt*[*OF lambda-gt* ∗∗]
using *less-eq-real-def* **by** *presburger*
then have $(\sum i = 1..k2+1.\ 1\ /\ 2\ \hat{}\ i) \leq (\sum i = 1..k1.\ 1\ /\ (2\ \hat{}\ i::real))$
using ∗ *eo* **by** *fastforce*
then show *?thesis* **by** *auto*
qed
show *?thesis*
using *h1 h2 assms* **by** *linarith*
qed

lemma *bounding-interval-helper1*:
shows $(\sum i = 1..k.\ 1\ /\ (2\ \hat{}\ i::real)) = (2\hat{}k - 1)/(2\hat{}k)$
proof(*induct k*)
case *0*
then show *?case* **by** *simp*
next
case (*Suc k*)
have $(\sum i = 1..(Suc\ k).\ 1\ /\ (2\ \hat{}\ i::real)) = (\sum i = 1..k.\ 1\ /\ (2\ \hat{}\ i::real)) +$

1/2^(Suc k)
by *force*
also have ... = *(2^k − 1)/(2^k) + 1/2^(Suc k)* **using** *Suc.hyps* **by** *presburger*
also have ... = *(2^k − 1)/(2^k) + 1/2^(k+1)* **by** *simp*
also have ... = *(2^(k+1) − 1)/(2^(k+1))*
by (*smt* (*verit*, *del-insts*) *Suc add.commute add-diff-cancel-right′ add-divide-distrib calculation field-sum-of-halves le-add2 plus-1-eq-Suc power-divide power-one summation-helper*)
finally show *?case* **by** *force*
qed

lemma *bounding-interval-helper2*:
fixes *x* :: *real*
assumes *x* ∈ {*0*..<*1*}
shows ∃ *k*. *x* < (∑ *i* = *1*..*k*. *1* / (*2* ^ *i*::*real*))
proof−
let *?f* = λ*k*::*nat*. *(2^k − 1)/(2^k)*
have *lim*: ∀ ε::*real*>*0*. ∃ *k*. *(1 − (?f k))* < ε
proof *clarify*
fix ε::*real*
assume ε > *0*
then obtain *m* **where** *m* > *0* ∧ *1* / *m* < ε
by (*metis Groups.mult-ac*(*2*) *divide-less-eq linordered-field-no-ub order-less-trans zero-less-divide-1-iff*)
moreover obtain *k* **where** *2^k* > *m* **using** *real-arch-pow* **by** *fastforce*
ultimately have *1* / *(2^k)* < ε **by** (*smt* (*verit*) *frac-less2*)
moreover have (*1*::*real*) − ((*2^k − 1*) / (*2^k*)) = (*1*/(*2^k*)) **by** (*simp add*: *diff-divide-distrib*)
ultimately show ∃ *k*. *1 − (2^k − 1) / (2^k)* < ε **by** (*smt* (*verit*))
qed
have ∃ *k*. *?f k* > *x*
proof−
let *?ε* = *1 − x*
obtain *k* **where** *1 − (?f k)* < *?ε* **by** (*metis assms lim atLeastLessThan-iff diff-gt-0-iff-gt*)
thus *?thesis* **by** *auto*
qed
thus *?thesis* **using** *bounding-interval-helper1* **by** *presburger*
qed

lemma *bounding-interval-for-reals-btw01*:
fixes *x*::*real*
assumes *x* ∈ {*0*..<*1*}
shows ∃ *k*. *x* ∈ {(∑ *i* ∈ {*1*..*k*}. *1*/(*2^i*::*real*))..<(∑ *i* ∈ {*1*..(*k* + *1*)}. *1*/(*2^i*))}
proof −
let *?S* = λ*k*. (∑ *i* = *1*..*k*. *1* / (*2* ^ *i*::*real*))
let *?A* = {*k*::*nat*. *x* < (∑ *i* = *1*..*k*. *1* / (*2* ^ *i*::*real*))}
let *?m* = *LEAST k*. *k* ∈ *?A*
have ∃ *k*. *x* < (∑ *i* = *1*..*k*. *1* / (*2* ^ *i*::*real*)) **using** *assms bounding-interval-helper2*

by *blast*
then have *?m ∈ ?A* **by** (*metis* (*mono-tags*, *lifting*) *LeastI2-wellorder mem-Collect-eq*)
moreover then have *?m − 1 ∉ ?A*
by (*smt* (*verit*, *ccfv-SIG*) *One-nat-def Suc-n-not-le-n Suc-pred′ assms atLeastLessThan-iff atLeastatMost-empty′ bot-nat-0.not-eq-extremum linorder-not-less mem-Collect-eq not-less-Least sum.empty*)
ultimately have *x* < (∑ *i* = *1..?m*. *1* / (*2* ^ *i*::*real*)) ∧ *x* ≥ (∑ *i* = *1..?m−1*. *1* / (*2* ^ *i*::*real*))
by *simp*
thus *?thesis*
by (*smt* (*verit*, *best*) *add.commute assms atLeastLessThan-iff le-add-diff-inverse linorder-not-less sum.head-if*)
qed

lemma *all-rotation-intervals-between-0and1*:
shows {(∑ *i* ∈ {*1..k*}. *1*/(*2*^*i*::*real*))..(∑ *i* ∈ {*1..*(*k*+*1*)}. *1*/(*2*^*i*))} ⊆ {*0..*<*1*}
proof −
have *gt*: ⋀*k*. (∑ *i* ∈ {*1..k*}. *1*/(*2*^*i*::*real*)) ≥ *0*
by (*simp add*: *sum-nonneg*)
have *lt*: ⋀*k*. (∑ *i* ∈ {*1..k*}. *1*/(*2*^*i*::*real*)) < *1*
by (*smt* (*verit*, *ccfv-SIG*) *diff-Suc-1 f-gteq-0-sum-gt less-Suc-eq-le linorder-not-le summation-helper zero-less-divide-1-iff zero-less-power*)
show *?thesis*
using *gt lt*
by (*meson atLeastAtMost-subseteq-atLeastLessThan-iff*)
qed

lemma *all-rotation-intervals-between-0and1-strict*:
shows {(∑ *i* ∈ {*1..k*}. *1*/(*2*^*i*::*real*))..<(∑ *i* ∈ {*1..*(*k*+*1*)}. *1*/(*2*^*i*))} ⊆ {*0..*<*1*}
using *all-rotation-intervals-between-0and1*
by (*smt* (*verit*, *ccfv-SIG*) *atLeastAtMost-subseteq-atLeastLessThan-iff ivl-subset nle-le order-trans*)

lemma *one-polygon-rotation-is-loop-free*:
assumes *polygon-of p vts*
assumes *p′* = *make-polygonal-path* (*rotate-polygon-vertices vts 1*)
(**is** *p′* = *make-polygonal-path ?vts′*)
shows *loop-free p′*
proof(*rule ccontr*)
assume ¬ *loop-free p′*
moreover have *p′ 0* = *p′ 1*
using *assms*
by (*smt* (*verit*, *ccfv-SIG*) *assms*(*2*) *butlast-snoc length-butlast linepath-0′ linepath-1′ make-polygonal-path.simps*(*1*) *not-gr-zero nth-append-length nth-butlast path-defs*(*2*) *path-defs*(*3*) *polygon-pathfinish polygon-pathstart rotate-polygon-vertices-def*)
ultimately obtain *x′ y′* **where** *x′y′*: *x′* < *y′* ∧ {*x′*, *y′*} ⊆ {*0..*<*1*} ∧ *p′ x′* = *p′ y′*
unfolding *loop-free-def*
by (*smt* (*verit*, *del-insts*) *atLeastAtMost-iff atLeastLessThan-iff bot-least in-*

sert-subset linorder-not-le order.refl order-antisym zero-less-one)

let *?n = length vts*
have *n-geq-4*: *?n ≥ 4* **using** *polygon-vertices-length-at-least-4 assms*
using *polygon-of-def* **by** *blast*
obtain *xk* **where** *x'-in*: *x' ∈ {(∑ i ∈ {1..xk}. 1/(2^i))..<(∑ i ∈ {1..(xk + 1)}. 1/(2^i))}* **using** *x'y'*
using *bounding-interval-for-reals-btw01 x'y'*
by (*metis insert-subset*)
then have *xk-gteq*: *xk ≥ 0*
by *blast*
obtain *yk* **where** *y'-in*: *y' ∈ {(∑ i ∈ {1..yk}. 1/(2^i))..<(∑ i ∈ {1..(yk + 1)}. 1/(2^i))}*
using *bounding-interval-for-reals-btw01 x'y'*
by (*metis insert-subset*)
then have *yk-gteq*: *yk ≥ 0*
by *blast*

have *all-pows-of-2-pos*: (⋀*i. 0 < 1 / (2 ^ i::real)*)
by *simp*

let *?x1 = (x' − (∑ i ∈ {1..xk}. 1/(2^i)))/2 + (∑ i ∈ {1..(xk + 1)}. 1/(2^i))*
have *xk-lt-nminus3*: *xk ≤ ?n − 4 ⟹ ?x1 ∈ {(∑ i ∈ {1..xk+1}. 1/(2^i))..<(∑ i ∈ {1..xk+2}. 1/(2^i))} ∧ p ?x1 = p' x'*
using *polygon-rotation-t-translation1-strict*[*OF assms*(*1*) *assms*(*2*) *x'-in*] *xk-gteq*
by *metis*
let *?y1 = (y' − (∑ i ∈ {1..yk}. 1/(2^i)))/2 + (∑ i ∈ {1..(yk + 1)}. 1/(2^i))*
have *yk-lt-nminus3*: *yk ≤ ?n − 4 ⟹ ?y1 ∈ {(∑ i ∈ {1..yk+1}. 1/(2^i))..<(∑ i ∈ {1..yk+2}. 1/(2^i))} ∧ p ?y1 = p' y'*
using *polygon-rotation-t-translation1-strict*[*OF assms*(*1*) *assms*(*2*) *y'-in*] *yk-gteq*

by *metis*

let *?x2 = x' + 1/(2^(?n−2))*
have *xk = ?n−3 ⟹ x' ∈ {∑ i = 1..length vts − 3. 1 / (2 ^ i::real)..<∑ i = 1..length vts − 2. 1 / 2 ^ i}*
using *x'-in*
by (*smt* (*verit*, *best*) *Nat.add-diff-assoc2 ‹4 ≤ length vts› diff-cancel2 le-add-diff-inverse nat-add-left-cancel-le nat-le-linear numeral-Bit0 numeral-Bit1 numerals*(*1*) *trans-le-add1*)
then have *xk-eq-nminus3*: *xk = ?n − 3 ⟹ p ?x2 = p' x' ∧ ?x2 ∈ {(∑ i ∈ {1..?n−2}. 1/(2^i))..<1}*
using *polygon-rotation-t-translation2-strict*[*OF assms*(*1*) *assms*(*2*), *of ?n x' ?x2*] *x'-in xk-gteq*
by *presburger*
let *?y2 = y' + 1/(2^(?n−2))*
have *yk = ?n−3 ⟹ y' ∈ {∑ i = 1..length vts − 3. 1 / (2 ^ i::real)..<∑ i = 1..length vts − 2. 1 / 2 ^ i}*
using *y'-in*

by (*smt* (*verit*, *best*) *Nat.add-diff-assoc2* ‹*4* ≤ *length vts*› *diff-cancel2 le-add-diff-inverse nat-add-left-cancel-le nat-le-linear numeral-Bit0 numeral-Bit1 numerals*(*1*) *trans-le-add1*)
then have *yk-eq-nminus3*: *yk* = *?n* − *3* ⟹ *p ?y2* = *p′ y′* ∧ *?y2* ∈ {(∑ *i* ∈ {*1*..*?n*−*2*}. *1*/(*2*^*i*))..<*1*}
using *polygon-rotation-t-translation2-strict*[*OF assms*(*1*) *assms*(*2*), *of ?n y′ ?y2*] *x′-in xk-gteq*
by *presburger*

let *?x3* = (*x′* − (∑ *i* ∈ {*1*..*?n*−*2*}. *1*/(*2*^*i*)))∗(*2*^(*?n*−*3*))
have *x′-leq*: *x′* < *1*
using *x′y′* **by** *simp*
have *x′-geq*: *xk* ≥ *?n* − *2* ⟹ (∑ *i* = *1*..*xk*. *1* / (*2* ^ *i*::*real*)) ≥ (∑ *i* = *1*..*length vts* − *2*. *1* / (*2* ^ *i*::*real*))
using *x′-in f-gteq-0-sum-gt*[*of* λ*i*. *1* / (*2* ^ *i*::*real*)]
by (*metis le-antisym less-eq-real-def linorder-not-le zero-less-divide-1-iff zero-less-numeral zero-less-power*)
have *xk* ≥ *?n*−*2* ⟹ *x′* ∈ {∑ *i* = *1*..*length vts* − *2*. *1* / (*2* ^ *i*::*real*)..<*1*}
using *x′-leq x′-geq x′-in*
by *fastforce*
then have *xk-gt-nminus3*: *xk* ≥ *?n* − *2* ⟹ *p ?x3* = *p′ x′* ∧ *?x3* ∈ {*0*..<*1*/*2*}
using *polygon-rotation-t-translation3-strict*[*OF assms*(*1*) *assms*(*2*), *of x′ ?n*] *xk-gteq*
by *presburger*
let *?y3* = (*y′* − (∑ *i* ∈ {*1*..*?n*−*2*}. *1*/(*2*^*i*)))∗(*2*^(*?n*−*3*))
have *y′-leq*: *y′* < *1*
using *x′y′* **by** *simp*
have *y′-geq*: *yk* ≥ *?n* − *2* ⟹ (∑ *i* = *1*..*yk*. *1* / (*2* ^ *i*::*real*)) ≥ (∑ *i* = *1*..*length vts* − *2*. *1* / (*2* ^ *i*::*real*))
using *y′-in f-gteq-0-sum-gt*[*of* λ*i*. *1* / (*2* ^ *i*::*real*)]
by (*metis le-antisym less-eq-real-def linorder-not-le zero-less-divide-1-iff zero-less-numeral zero-less-power*)
have *yk* ≥ *?n*−*2* ⟹ *y′* ∈ {∑ *i* = *1*..*length vts* − *2*. *1* / (*2* ^ *i*::*real*)..<*1*}
using *y′-leq y′-geq y′-in*
by *fastforce*
then have *yk-gt-nminus3*: *yk* ≥ *?n* − *2* ⟹ *p ?y3* = *p′ y′* ∧ *?y3* ∈ {*0*..<*1*/*2*}
using *polygon-rotation-t-translation3-strict*[*OF assms*(*1*) *assms*(*2*), *of y′ ?n*] *yk-gteq*
by *presburger*

have *interval-helper*: *a1* ≥ *b2* ∧*x* ∈ {*a1*..<*a2*} ∧ *y* ∈ {*b1*..<*b2*} ⟹ *y* < *x* **for** *a1 a2 b1 b2 x y*::*real*
by *simp*

{ assume *xk-lt*: *xk* < *?n* − *3*
then have *p-x′*: *p ?x1* = *p′ x′*
using *xk-lt-nminus3* **by** *auto*
have *x1-in*: *?x1* ∈ {(∑ *i* ∈ {*1*..(*xk* + *1*)}. *1*/(*2*^*i*))..<(∑ *i* ∈ {*1*..(*xk* + *2*)}. *1*/(*2*^*i*))}
using *xk-lt xk-lt-nminus3*

by *auto*
then have *x1-in-01*: *?x1* ∈ {*0*..<*1*}
using *all-rotation-intervals-between-0and1-strict*[*of xk+1*]
by *fastforce*
{ assume *yk-lt*: *yk* < *?n* − *3*
then have *p-y'*: *p ?y1* = *p' y'*
using *yk-lt-nminus3* **by** *auto*
have *y1-in*: *?y1* ∈ {(∑ *i* ∈ {*1*..(*yk* + *1*)}. *1*/(*2*^*i*))..<(∑ *i* ∈ {*1*..(*yk* + *2*)}. *1*/(*2*^*i*))}
using *yk-lt yk-lt-nminus3* **by** *auto*
then have *y1-in-01*: *?y1* ∈ {*0*..<*1*}
using *all-rotation-intervals-between-0and1-strict*[*of yk+1*]
by *fastforce*
have {∑ *i* = *1*..*xk* + *1*. *1* / *2* ^ *i*..<∑ *i* = *1*..*xk* + *2*. *1* / (*2* ^ *i*::*real*)} ∩ {∑ *i* = *1*..*yk* + *1*. *1* / (*2* ^ *i*::*real*)..<∑ *i* = *1*..*yk* + *2*. *1* / *2* ^ *i*} = {} **if** *xk-neq*:*xk* ≠ *yk*
using *rotation-intervals-disjoint*[*of xk+1 yk+1*] *xk-neq*
by *fastforce*
then have *eq-then-eq*: *?x1* = *?y1* ⟹ *xk* = *yk*
using *x1-in y1-in*
by (*smt* (*verit*) *Int-iff empty-iff*)
have *xk* = *yk* ⟹ *?x1* ≠ *?y1*
using *x'y' x1-in y1-in* **by** *simp*
then have *?x1* ≠ *?y1*
using *eq-then-eq* **by** *blast*
moreover have {*?x1*, *?y1*} ⊆ {*0*..<*1*}
using *x1-in-01 y1-in-01* **by** *fast*
ultimately have *?x1* ≠ *?y1* ∧ {*?x1*, *?y1*} ⊆ {*0*..<*1*} ∧ *p ?x1* = *p ?y1*
using *p-x' p-y' x'y'* **by** *presburger*
then have ∃ *x y* . *x* ≠ *y* ∧ {*x*, *y*} ⊆ {*0*..<*1*} ∧ *p x* = *p y*
by *auto*
then have *False*
using *assms*(*1*) **unfolding** *polygon-of-def polygon-def simple-path-def loop-free-def*
by *fastforce*
} moreover { assume *yk* = *?n* − *3*
then have *y2*: *p ?y2* = *p' y'* ∧ *?y2* ∈ {(∑ *i* ∈ {*1*..*?n*−*2*}. *1*/(*2*^*i*))..<*1*}
using *yk-eq-nminus3*
by *auto*
then have *y2-in-01*: *?y2* ∈ {*0*..<*1*}
using *all-rotation-intervals-between-0and1-strict*[*of ?n*−*2*]
by *fastforce*
have *xkplus-eq*: *xk* + *2* = *?n* − *2* ⟹ (∑ *i* ∈ {*1*..(*xk* + *2*)}. *1*/(*2*^*i*::*real*)) ≤ (∑ *i* ∈ {*1*..*?n*−*2*}. *1*/(*2*^*i*))
by *simp*
have *xkplus-lt*: *xk* + *2* < *?n* − *2* ⟹ (∑ *i* ∈ {*1*..(*xk* + *2*)}. *1*/(*2*^*i*::*real*)) ≤ (∑ *i* ∈ {*1*..*?n*−*2*}. *1*/(*2*^*i*))
using *xk-lt f-gteq-0-sum-gt*[*OF all-pows-of-2-pos*, *of xk* + *2 ?n* − *2*]
by (*smt* (*verit*, *best*) *f-gteq-0-sum-gt zero-less-divide-1-iff zero-less-power*)
then have (∑ *i* ∈ {*1*..(*xk* + *2*)}. *1*/(*2*^*i*::*real*)) ≤ (∑ *i* ∈ {*1*..*?n*−*2*}. *1*/(*2*^*i*))

using *xkplus-eq xkplus-lt xk-lt*
using *One-nat-def Suc-diff-Suc Suc-eq-plus1 Suc-le-eq add-Suc-right le-neq-implies-less linorder-not-le nat-1-add-1 nat-diff-split numeral-3-eq-3 xk-gteq* **by** *linarith*
then have *?x1* $\neq$ *?y2*
using *x1-in y2*
by (*smt* (*verit*, *ccfv-SIG*) *interval-helper*)
moreover have {*?x1*, *?y2*} $\subseteq$ {*0..<1*}
using *x1-in-01 y2-in-01* **by** *fast*
ultimately have *?x1* $\neq$ *?y2* $\wedge$ {*?x1*, *?y2*} $\subseteq$ {*0..<1*} $\wedge$ *p ?x1* = *p ?y2*
using *p-x' y2 x'y'* **by** *presburger*
then have $\exists$ *x y . x* $\neq$ *y* $\wedge$ {*x*, *y*} $\subseteq$ {*0..<1*} $\wedge$ *p x* = *p y*
by *auto*
then have *False*
using *assms*(*1*) **unfolding** *polygon-of-def polygon-def simple-path-def loop-free-def*
by *fastforce*
}
moreover { assume *yk* > *?n* − *3*
then have *y3*: *p ?y3* = *p' y'* $\wedge$ *?y3* $\in$ {*0..<*(*1/2*::*real*)}
using *yk-gt-nminus3*
by *auto*
then have *y3-in-01*: *?y3* $\in$ {*0..<1*}
by *simp*

have *simplify-interval*: ($\sum$ *i* = *1..1*. *1* / (*2* ^ *i*::*real*)) = *1/2*
by *simp*
then have *xk-eq-0*: *xk* = *0* $\Longrightarrow$ ($\sum$ *i* $\in$ {*1..*(*xk* + *1*)}. *1*/(*2*^*i*::*real*)) $\geq$ *1/2*
by *simp*
have *xk* > *0* $\Longrightarrow$ ($\sum$ *i* $\in$ {*1..*(*xk* + *1*)}. *1*/(*2*^*i*::*real*)) $\geq$ *1/2*
using *f-gteq-0-sum-gt*[*OF all-pows-of-2-pos*, *of 1 xk +1*]
simplify-interval
by (*smt* (*verit*, *ccfv-SIG*) *Suc-le-eq add.commute add.right-neutral all-pows-of-2-pos f-gteq-0-sum-gt linorder-not-le plus-1-eq-Suc*)
then have ($\sum$ *i* $\in$ {*1..*(*xk* + *1*)}. *1*/(*2*^*i*::*real*)) $\geq$ *1/2*
using *xk-eq-0 xk-gteq* **by** *blast*
then have *?x1* $\neq$ *?y3*
using *x1-in y3*
by (*smt* (*verit*, *best*) *interval-helper*)
moreover have {*?x1*, *?y3*} $\subseteq$ {*0..<1*}
using *x1-in-01 y3-in-01* **by** *fast*
ultimately have *?x1* $\neq$ *?y3* $\wedge$ {*?x1*, *?y3*} $\subseteq$ {*0..<1*} $\wedge$ *p ?x1* = *p ?y3*
using *p-x' y3 x'y'*
by *presburger*
then have $\exists$ *x y . x* $\neq$ *y* $\wedge$ {*x*, *y*} $\subseteq$ {*0..<1*} $\wedge$ *p x* = *p y*
by *auto*
then have *False*
using *assms*(*1*) **unfolding** *polygon-of-def polygon-def simple-path-def loop-free-def*
by *fastforce*

```
    }
    ultimately have False by linarith
  } moreover {assume xk-eq : xk = ?n−3
    then have p-x': p ?x2 = p' x'
      using xk-eq-nminus3 by auto
    have x2-in: ?x2 ∈ {(∑ i ∈ {1..?n−2}. 1/(2^i))..<1}
      using xk-eq xk-eq-nminus3
      by auto
    then have ?x2 ≥ 0
      using n-geq-4
        by (metis add-sign-intros(4) atLeastLessThan-iff insert-subset leD nle-le
power-one-over x'y' zero-le-power zero-less-divide-1-iff zero-less-numeral)
    then have x2-in-01: ?x2 ∈ {0..<1}
      using x2-in by auto
    { assume yk < ?n − 3
     then have interval-helper-helper: (∑ i = 1..yk + 1. 1 / (2 ^ i::real)) ≤ (∑ i
= 1..xk. 1 / (2 ^ i::real))
        using xk-eq f-gteq-0-sum-gt
      by (metis Suc-eq-plus1 less-eq-real-def linorder-neqE-nat not-less-eq zero-less-divide-1-iff
zero-less-numeral zero-less-power)
      then have x' > y'
        using x'-in y'-in  interval-helper[of (∑ i = 1..yk + 1. 1 / (2 ^ i::real))
(∑ i = 1..xk. 1 / (2 ^ i::real))]
        by blast
      then have False using x'y'
        by auto
    } moreover { assume yk = ?n − 3
      then have y2: p ?y2 = p' y' ∧ ?y2 ∈ {(∑ i ∈ {1..?n−2}. 1/(2^i))..<1}
      using yk-eq-nminus3
      by auto
    then have y2-in-01: ?y2 ∈ {0..<1}
        using all-rotation-intervals-between-0and1-strict[of ?n−2]
        by fastforce
      then have ?x2 ≠ ?y2
        using x'y' by auto
    moreover have {?x2, ?y2} ⊆ {0..<1}
        using x2-in-01 y2-in-01 by fast
     ultimately have  ?x2 ≠ ?y2 ∧ {?x2, ?y2} ⊆ {0..<1} ∧ p ?x2 = p ?y2
        using  p-x' y2  x'y' by presburger
      then have ∃ x y . x ≠ y ∧ {x, y} ⊆ {0..<1} ∧ p x = p y
        by meson
      then have False
            using assms(1) unfolding polygon-of-def polygon-def simple-path-def
loop-free-def
        by fastforce
    } moreover { assume yk-gt: yk > ?n − 3
    then have y3: p ?y3 = p' y'
      using yk-gt-nminus3 by auto
    have y3-in: ?y3 ∈ {0..<1/2}
```

```
    using yk-gt yk-gt-nminus3
    by auto
  then have y3-in-01: ?y3 ∈ {0..<1}
    by auto
  have (∑ i = 1..length vts − 2. 1 / (2 ^ i::real)) > (∑ i = 1..1. 1 / (2 ^ i::real))
    using n-geq-4 f-gteq-0-sum-gt[OF all-pows-of-2-pos,of 1 length vts − 2]
    by fastforce
  then have (∑ i = 1..length vts − 2. 1 / (2 ^ i::real)) > 1/2
    by simp
  then have ?x2 ≠ ?y3
    using y3-in x2-in by auto
  moreover have {?x2, ?y3} ⊆ {0..<1}
     using x2-in-01 y3-in-01 by fast
   ultimately have ?x2 ≠ ?y3 ∧ {?x2, ?y3} ⊆ {0..<1} ∧ p ?x2 = p ?y3
    using p-x' y3 x'y' by presburger
    then have ∃ x y . x ≠ y ∧ {x, y} ⊆ {0..<1} ∧ p x = p y
      by meson
    then have False
        using assms(1) unfolding polygon-of-def polygon-def simple-path-def
loop-free-def
      by fastforce
  }
  ultimately have False
    using not-less-iff-gr-or-eq by auto
 } moreover { assume xk-gt: xk > ?n − 3
  then have p-x': p ?x3 = p' x'
    using xk-gt-nminus3 by auto
  have x3-in: ?x3 ∈ {0..<1/2}
    using xk-gt xk-gt-nminus3
    by auto
  then have x3-in-01: ?x3 ∈ {0..<1}
    by auto
  { assume yk ≤ ?n − 3
    then have (∑ i = 1..xk. 1 / (2 ^ i::real)) ≥ (∑ i = 1..yk + 1. 1 / (2 ^ i::real))
      using xk-gt f-gteq-0-sum-gt[of λi. 1 / (2 ^ i::real) xk yk]
    proof −
      obtain rr :: nat ⇒ real where
        f1: ∀ B-x. rr B-x = 1 / 2 ^ B-x
        by force
      then have f2: ∀ n. 0 < rr n
        by simp
      have yk < xk
        using ‹length vts − 3 < xk› ‹yk ≤ length vts − 3› order-le-less-trans by
blast
      then show ?thesis
      using f2 f1 by (metis (no-types) Suc-eq-plus1 f-gteq-0-sum-gt less-eq-real-def
nat-neq-iff not-less-eq order.refl)
```

```
    qed
    then have x' > y'
     using x'-in y'-in interval-helper[of (∑ i = 1..yk + 1. 1 / (2 ^ i::real)) (∑ i
= 1..xk. 1 / (2 ^ i::real))]
      by blast
    then have False using x'y'
      by auto
   } moreover
   { assume yk-gt: yk > ?n − 3
    then have p-y': p ?y3 = p' y'
      using yk-gt-nminus3 by auto
    have y3-in: ?y3 ∈ {0..<1/2}
      using yk-gt yk-gt-nminus3
      by auto
    then have y3-in-01: ?y3 ∈ {0..<1}
      by auto
    have (x' − (∑ i = 1..length vts − 2. 1 / 2 ^ i)) ≠
       (y' − (∑ i = 1..length vts − 2. 1 / 2 ^ i))
      using x'y' by auto
    then have ?x3 ≠ ?y3 by auto
    moreover have {?x3, ?y3} ⊆ {0..<1}
      using x3-in-01 y3-in-01 by fast
    ultimately have  ?x3 ≠ ?y3 ∧ {?x3, ?y3} ⊆ {0..<1} ∧ p ?x3 = p ?y3
      using  p-x' p-y'  x'y'
      by presburger
    then have ∃ x y . x ≠ y ∧ {x, y} ⊆ {0..<1} ∧ p x = p y
      by meson
    then have False
        using assms(1) unfolding polygon-of-def polygon-def simple-path-def
loop-free-def
      by fastforce
   }
   ultimately have False by linarith
  }
  ultimately show False by linarith
qed

lemma one-rotation-is-polygon:
  fixes p :: R-to-R2
  fixes i :: nat
  assumes poly-p: polygon p and
        p-is-path: p = make-polygonal-path vts and
        p'-is: p' = make-polygonal-path (rotate-polygon-vertices vts 1)
          (is p' = make-polygonal-path ?vts')
  shows polygon p'
proof−
  have polygonal-path p' using p'-is by (simp add: polygonal-path-def)
  moreover have closed-path p'
    using p'-is unfolding rotate-polygon-vertices-def closed-path-def
```

by (*metis* (*no-types*, *opaque-lifting*) *Nil-is-append-conv append-self-conv2 diff-Suc-1 hd-append2 hd-conv-nth length-append-singleton make-polygonal-path-gives-path not-Cons-self nth-Cons-0 nth-append-length pathfinish-def pathstart-def polygon-pathfinish polygon-pathstart*)
moreover have *simple-path p′*
using *one-polygon-rotation-is-loop-free*
by (*metis make-polygonal-path-gives-path p′-is p-is-path poly-p polygon-of-def simple-path-def*)
ultimately show *?thesis* **unfolding** *polygon-def* **by** *simp*
qed

lemma *rotation-is-polygon*:
fixes *p* :: *R-to-R2*
fixes *i*:: *nat*
assumes *polygon p* **and**
p = *make-polygonal-path vts*
shows *polygon* (*make-polygonal-path* (*rotate-polygon-vertices vts i*))
using *assms*
proof (*induct i*)
case *0*
then show *?case* **using** *rotate0* **unfolding** *rotate-polygon-vertices-def*
by (*smt* (*z3*) *assms*(*2*) *butlast.simps*(*1*) *butlast-conv-take eq-id-iff have-wraparound-vertex hd-append2 hd-conv-nth rotate-polygon-vertices-def rotate-polygon-vertices-same-set self-append-conv2 the-elem-set*)
next
case (*Suc i*)
then show *?case* **using** *one-rotation-is-polygon arb-rotation-as-single-rotation*
by *metis*
qed

lemma *polygon-rotate-mod*:
fixes *vts* :: (*real^2*) *list*
assumes *n* = *length vts*
assumes *n* ≥ *2*
assumes *hd vts* = *last vts*
shows *rotate-polygon-vertices vts* (*n* − *1*) = *vts*
proof−
let *?vts′* = *rotate* (*n* − *1*) (*butlast vts*)
have *rotate-polygon-vertices vts* (*n* − *1*) = *?vts′* @ [*?vts′*!*0*]
unfolding *rotate-polygon-vertices-def* **by** *metis*
moreover have *?vts′* = *butlast vts* **using** *assms* **by** *simp*
moreover have ... = *rotate 0* (*butlast vts*) **by** *simp*
moreover then have ... @ [...!*0*] = *rotate-polygon-vertices vts 0*
unfolding *rotate-polygon-vertices-def* **by** *metis*
moreover have ... = *vts*
unfolding *rotate-polygon-vertices-def* **using** *assms*
by (*metis* (*no-types*, *lifting*) *Suc-le-eq calculation*(*3*) *hd-conv-nth length-butlast length-greater-0-conv nat-1-add-1 nth-butlast order-less-le-trans plus-1-eq-Suc pos2 snoc-eq-iff-butlast zero-less-diff*)

ultimately show *?thesis* **by** *argo*
qed

lemma *polygon-rotate-mod-arb*:
fixes *vts* :: (*real^2*) *list*
assumes *n* = *length vts*
assumes *n* ≥ *2*
assumes *hd vts* = *last vts*
shows *rotate-polygon-vertices vts* ((*n* − *1*) ∗ *i*) = *vts*
proof(*induct i*)
case *0*
then show *?case* **using** *polygon-rotate-mod*
by (*metis append.right-neutral append-Nil assms*(*1*) *assms*(*2*) *assms*(*3*) *id-apply length-butlast mult-zero-right rotate0 rotate-append rotate-polygon-vertices-def*)
next
case (*Suc i*)
then have *vts* = *rotate-polygon-vertices vts* ((*n* − *1*) ∗ *i*) **using** *Suc.prems* **by** *argo*
also have ... = *rotate-polygon-vertices vts* ((*n* − *1*) ∗ *Suc i*)
using *polygon-rotate-mod assms*(*1*) *assms*(*2*) *assms*(*3*) *calculation rotation-sum*
by (*metis mult-Suc-right*)
finally show *?case* **by** *argo*
qed

lemma *unrotation-is-polygon*:
fixes *p* :: *R-to-R2*
fixes *i*:: *nat*
assumes *polygon* (*make-polygonal-path* (*rotate-polygon-vertices vts i*))
(**is** *polygon* (*make-polygonal-path ?vts'*))
p = *make-polygonal-path vts*
hd vts = *last vts*
shows *polygon p*
proof−
have *len-vts*: *length vts* ≥ *2*
using *assms polygon-vertices-length-at-least-4 rotate-polygon-vertices-same-length*
by (*metis* (*no-types*, *opaque-lifting*) *Suc-1 Suc-eq-numeral Suc-le-lessD diff-is-0-eq' eval-nat-numeral*(*2*) *gr-implies-not0 length-append-singleton length-butlast length-rotate not-less-eq-eq rotate-polygon-vertices-def*)

let *?n* = *length vts* − *1*
obtain *k* **where** *k*: *k*∗*?n* > *i*
using *len-vts*
by (*metis Suc-1 Suc-le-eq add-0 div-less-iff-less-mult le-add2 less-diff-conv*)
let *?j* = *k*∗*?n* − *i*
have *j-i-n*: *?j* + *i* = *k*∗*?n* **using** *k* **by** *simp*

have *rotate-polygon-vertices ?vts' ?j* = *rotate-polygon-vertices vts* (*?j* + *i*)
using *rotation-sum*[*of vts i ?n*] **by** (*simp add*: *add.commute rotation-sum*)
also have ... = *rotate-polygon-vertices vts* (*k*∗*?n*) **using** *assms j-i-n* **by** *presburger*

also have ... = *vts* **using** *polygon-rotate-mod-arb len-vts assms* **by** (*metis mult.commute*)
finally show *?thesis* **using** *rotation-is-polygon assms* **by** *metis*
qed

lemma *rotated-polygon-vertices*:
assumes $vts' = rotate\text{-}polygon\text{-}vertices\ vts\ j$
assumes $hd\ vts = last\ vts$
assumes $length\ vts \geq 2$
assumes $j \leq i \wedge i < length\ vts$
shows $vts\ !\ i = vts'\ !\ (i - j)$
using *assms*
proof(*induct j arbitrary*: *vts vts'*)
case *0*
then show *?case*
by (*metis Suc-1 Suc-le-eq diff-is-0-eq diff-zero hd-conv-nth id-apply length-butlast linorder-not-le list.size*(*3*) *nth-butlast rotate0 rotate-polygon-vertices-def snoc-eq-iff-butlast*)
next
case (*Suc j*)
then have $vts' = rotate\text{-}polygon\text{-}vertices\ (rotate\text{-}polygon\text{-}vertices\ vts\ 1)\ j$
by (*metis plus-1-eq-Suc rotation-sum*)
moreover have ...$!(i - Suc\ j) = (rotate\text{-}polygon\text{-}vertices\ vts\ 1)!(i - 1)$
using *Suc.hyps Suc.prems*(*3*) *Suc.prems*(*4*) *Suc-1 Suc-diff-le Suc-leD diff-Suc-Suc hd-conv-nth length-append-singleton length-butlast length-rotate nth-butlast rotate-polygon-vertices-def snoc-eq-iff-butlast zero-less-Suc*
by (*smt* (*z3*) *One-nat-def Suc.prems*(*1*) *Suc.prems*(*2*) *Suc-eq-plus1 Suc-le-eq arb-rotation-as-single-rotation calculation diff-diff-cancel diff-is-0-eq diff-less-mono diff-zero not-less-eq-eq plus-1-eq-Suc rotated-polygon-vertices-helper2*)
moreover have ... = *vts*!*i* **using** *rotated-polygon-vertices-helper2*
by (*metis Suc.prems*(*2*) *Suc.prems*(*3*) *Suc.prems*(*4*) *add-leD1 le-add-diff-inverse2 less-diff-conv plus-1-eq-Suc*)
ultimately show *?case*
by *presburger*
qed

lemma *polygon-path-image*:
assumes *poly-p*: *polygon p*
assumes *p-is-path*: $p = make\text{-}polygonal\text{-}path\ vts$
shows $path\text{-}image\ p = p\ `\ \{0\ ..<\ 1\}$
proof −
have *vts-nonempty*: $vts \neq []$
using *polygon-at-least-3-vertices*[*OF poly-p p-is-path*]
by *auto*
have *at-0*: $p\ `\ \{0\}\ = \{pathstart\ p\}$
using *p-is-path*
by (*metis image-empty image-insert pathstart-def*)
have *at-1*: $p\ `\ \{1\} = \{pathfinish\ p\}$
using *p-is-path*
by (*simp add*: *pathfinish-def*)
have *same-point*: $p\ 0 = p\ 1$

using *assms* **unfolding** *polygon-def closed-path-def* **using** *polygon-pathstart*[*OF vts-nonempty p-is-path*]
using *polygon-pathfinish*[*OF vts-nonempty p-is-path*]
at-0 at-1 **by** *auto*
have $\bigwedge x.\ x \in p\ `\ \{0..1\} \Longrightarrow x \in p\ `\ \{0..<1\}$
proof −
fix *x*
assume $x \in p\ `\ \{0..1\}$
then have $\exists k \in \{0..1\}.\ p\ k = x$
by *auto*
then obtain *k* **where** *k-prop*: $k \in \{0..1\} \wedge p\ k = x$
by *auto*
{assume ∗: $k < 1$
then have $\exists k \in \{0..<1\}.\ p\ k = x$
using *k-prop* **by** *auto*
} moreover {assume ∗: $k = 1$
then have $p\ 0 = x$
using *same-point k-prop* **by** *auto*
then have $\exists k \in \{0..<1\}.\ p\ k = x$
by *auto*
}
ultimately have $\exists k \in \{0..<1\}.\ p\ k = x$
using *k-prop*
by (*metis atLeastAtMost-iff order-less-le*)
then show *?thesis*
by *auto*
qed
then show *?thesis*
unfolding *path-image-def* **by** *auto*
qed

lemma *polygon-vts-one-rotation*:
fixes *p* :: *R-to-R2*
assumes *poly-p*: *polygon p* **and**
p-is-path: $p = $ *make-polygonal-path vts* **and**
p'*-is*: $p' = $ *make-polygonal-path* (*rotate-polygon-vertices vts 1*)
shows *path-image p* = *path-image* p'
proof −
let *?rotated-vts* = (*rotate-polygon-vertices vts 1*)
have *card* (*set vts*) ≥ *3*
using *polygon-at-least-3-vertices*[*OF poly-p p-is-path*]
by *auto*
then have *len-gt-eq3*: *length vts* ≥ *3*
using *card-length order-trans* **by** *blast*
have *same-len*: *length ?rotated-vts* = *length vts*
unfolding *rotate-polygon-vertices-def* **using** *length-rotate*
by (*metis One-nat-def Suc-pred card.empty length-append-singleton length-butlast length-greater-0-conv list.set*(*1*) *not-numeral-le-zero p-is-path poly-p polygon-at-least-3-vertices*)
then have *len-rotated-gt-eq2*: *length ?rotated-vts* ≥ *2*

```
    using len-gt-eq3 by auto
  have h1: ⋀x. x ∈ (path-image p) ⟹ x ∈ path-image p'
  proof −
    fix x
    assume x ∈ (path-image p)
    then have ∃k<length vts − 1. x ∈ path-image (linepath (vts ! k) (vts ! (k +
1)))
      using p-is-path len-gt-eq3 make-polygonal-path-image-property[of vts x]
      by auto
    then obtain k where k-prop: k < length vts − 1 ∧ x ∈ path-image (linepath
(vts ! k) (vts ! (k + 1)))
      by auto
    {assume ∗: k = 0
      have vts1: vts ! 0 = ?rotated-vts ! (length ?rotated-vts − 2)
        unfolding rotate-polygon-vertices-def
        using nth-rotate[of length ?rotated-vts − 2 butlast vts 1]
       by (metis (no-types, lifting) ∗ One-nat-def Suc-pred butlast-snoc diff-diff-left
k-prop length-butlast lessI mod-self nat-1-add-1 nth-butlast plus-1-eq-Suc rotate-polygon-vertices-def
same-len)
      have (rotate 1 (butlast vts)) ! 0 = vts ! 1
        using nth-rotate[of 0 butlast vts 1] len-gt-eq3
        by (simp add: less-diff-conv mod-if nth-butlast)
      then have vts2: vts ! 1 = ?rotated-vts ! (length ?rotated-vts − 1)
        unfolding rotate-polygon-vertices-def
        by (metis butlast-snoc length-butlast nth-append-length)
      then have path-image (linepath (vts ! k) (vts ! (k + 1))) ⊆ path-image p'
        using linepaths-subset-make-polygonal-path-image[of vts 0]
        len-rotated-gt-eq2 ∗
        by (metis (no-types, lifting) One-nat-def Suc-eq-plus1 Suc-pred diff-diff-left
diff-less k-prop less-numeral-extra(1) linepaths-subset-make-polygonal-path-image nat-1-add-1
p'-is same-len vts1)
      then have x ∈ path-image p'
        using k-prop vts1 vts2
        by auto
    }
    moreover {assume ∗: k > 0
      then have k-minus-prop: k − 1 < length (rotate-polygon-vertices vts 1) − 1
        using same-len  k-prop less-imp-diff-less
        by presburger
      then have vts1: vts ! k = ?rotated-vts ! (k−1)
        using nth-rotate[of k−1 butlast vts 1] len-gt-eq3
        same-len
       by (metis ∗ One-nat-def Suc-pred butlast-snoc k-prop length-butlast mod-less
nth-butlast plus-1-eq-Suc rotate-polygon-vertices-def)
      have vts2: vts ! (k+1) = ?rotated-vts ! k
        using nth-rotate[of k butlast vts 1] len-gt-eq3 k-minus-prop
     by (metis (no-types, lifting) ∗ Suc-eq-plus1 Suc-leI butlast-snoc have-wraparound-vertex
k-prop le-imp-less-Suc length-butlast mod-less mod-self nat-less-le nth-append-length
nth-butlast p-is-path plus-1-eq-Suc poly-p rotate-polygon-vertices-def same-len)
```

```
    have path-image (linepath (?rotated-vts ! (k−1)) (?rotated-vts ! k)) ⊆ path-image
p'
          using linepaths-subset-make-polygonal-path-image[OF len-rotated-gt-eq2
k-minus-prop] p'-is
       by (simp add: ∗)
     then have x ∈ path-image p'
       using k-prop vts1 vts2
       by auto
   }
   ultimately show x ∈ path-image p'
     by auto
 qed
 have h2: ⋀x. x ∈ (path-image p') ⟹ x ∈ path-image p
 proof −
   fix x
   assume x ∈ (path-image p')
   then have ∃k<length ?rotated-vts − 1. x ∈ path-image (linepath (?rotated-vts
! k) (?rotated-vts ! (k + 1)))
    using p'-is len-rotated-gt-eq2 make-polygonal-path-image-property[of ?rotated-vts
x]
     by auto
    then obtain k where k-prop: k < length ?rotated-vts − 1 ∧ x ∈ path-image
(linepath (?rotated-vts ! k) (?rotated-vts ! (k + 1)))
     by auto
    {assume ∗: k = length ?rotated-vts − 2
     have vts1: vts ! 0 = ?rotated-vts ! (length ?rotated-vts − 2)
       unfolding rotate-polygon-vertices-def
       using nth-rotate[of length ?rotated-vts − 2 butlast vts 1]
         by (metis ∗ Suc-diff-Suc Suc-le-eq butlast-snoc k-prop len-rotated-gt-eq2
length-butlast mod-self nat-1-add-1 nth-butlast plus-1-eq-Suc rotate-polygon-vertices-def
same-len zero-less-Suc)
     have (rotate 1 (butlast vts)) ! 0 = vts ! 1
       unfolding rotate-polygon-vertices-def
       using nth-rotate[of 0 butlast vts 1] len-gt-eq3 len-rotated-gt-eq2
      by (metis (no-types, lifting) One-nat-def Suc-le-eq diff-diff-left length-butlast
less-nat-zero-code mod-less not-gr-zero nth-butlast numeral-3-eq-3 plus-1-eq-Suc zero-less-diff)
     then have vts2: ?rotated-vts ! (k+1) = vts ! 1
       unfolding rotate-polygon-vertices-def
    by (metis ∗ Suc-diff-Suc Suc-eq-plus1 Suc-le-eq len-rotated-gt-eq2 length-butlast
length-rotate nat-1-add-1 nth-append-length same-len)
     have path-image (linepath (vts ! 0) (vts ! 1)) ⊆ path-image p
       using linepaths-subset-make-polygonal-path-image[of vts 0]
       len-gt-eq3 ∗ less-diff-conv p-is-path same-len
       by auto
     then have x ∈ path-image p
       using ∗ vts1 vts2 k-prop
       by auto
   } moreover {assume ∗: k < length ?rotated-vts − 2
     then have vts1: ?rotated-vts ! k = vts ! (k+1)
```

```
      using nth-rotate[of k butlast vts 1] len-gt-eq3 *
        same-len
        by (smt (z3) Suc-eq-plus1 butlast-snoc diff-diff-left k-prop length-butlast
less-diff-conv mod-less nat-1-add-1 nth-butlast plus-1-eq-Suc rotate-polygon-vertices-def)
    have vts2: ?rotated-vts ! (k+1) = vts ! (k+2)
      using nth-rotate[of k+1 butlast vts 1] len-gt-eq3 *
        by (smt (verit, ccfv-threshold) One-nat-def Suc-le-eq add-Suc-right but-
last-snoc diff-diff-left have-wraparound-vertex len-rotated-gt-eq2 length-butlast less-diff-conv
mod-less mod-self nat-1-add-1 nat-less-le nth-append-length nth-butlast p-is-path
plus-1-eq-Suc poly-p rotate-polygon-vertices-def same-len)
    have path-image (linepath (vts ! (k+1)) (vts ! (k + 2))) ⊆ path-image p
      using linepaths-subset-make-polygonal-path-image[of vts k+1]
      len-gt-eq3 * less-diff-conv p-is-path same-len
      by auto
    then have x ∈ path-image p
      using vts1 vts2 k-prop
      by auto
  }
  ultimately show x ∈ path-image p
      using k-prop Suc-eq-plus1 add-le-imp-le-diff diff-diff-left len-rotated-gt-eq2
less-diff-conv2 linorder-neqE-nat not-less-eq one-add-one
    by linarith
 qed
 then show ?thesis
  using h1 h2 by auto
qed

lemma polygon-vts-arb-rotation:
 fixes p :: R-to-R2
 assumes polygon p and
       p = make-polygonal-path vts
 shows path-image p = path-image (make-polygonal-path (rotate-polygon-vertices
vts i))
 using assms
proof (induct i)
 case 0
 then show ?case unfolding rotate-polygon-vertices-def
   by (metis One-nat-def arb-rotation-as-single-rotation polygon-vts-one-rotation
rotate-polygon-vertices-def rotation-is-polygon)
next
 case (Suc i)
 let ?p' = make-polygonal-path (rotate-polygon-vertices vts (Suc i))
 {assume *: i = 0
  have path-image p = path-image ?p'
    using Suc polygon-vts-one-rotation[of p vts]
    by (simp add: *)
 }
 moreover {assume *: i > 0
  have path-image p = path-image ?p'
```

using *polygon-vts-one-rotation arb-rotation-as-single-rotation rotation-is-polygon*

by (*metis Suc.hyps Suc.prems*(*1*) *assms*(*2*))
}
ultimately show *?case* **by** *auto*
qed

10 Translating a Polygon

lemma *linepath-translation*:
linepath ((λ*x*. *x* + *u*) *a*) ((λ*x*. *x* + *u*) *b*) = (λ*x*. *x* + *u*) ∘ (*linepath a b*)
proof−
let *?l* = *linepath* ((λ*x*. *x* + *u*) *a*) ((λ*x*. *x* + *u*) *b*)
let *?l*′ = (λ*x*. *x* + *u*) ∘ (*linepath a b*)
have *?l x* = *?l*′ *x* **for** *x*
proof−
have *?l x* = (*1* − *x*) ∗*R* (*a* + *u*) + *x* ∗*R* (*b* + *u*) **unfolding** *linepath-def* **by** *simp*
also have ... = ((*1* − *x*) ∗*R* *a* + *x* ∗*R* *b*) + *u* **by** (*simp add*: *scaleR-right-distrib*)
also have ... = *?l*′ *x* **unfolding** *linepath-def* **by** *simp*
finally show *?thesis* **.**
qed
thus *?thesis* **by** *fast*
qed

lemma *make-polygonal-path-translate*:
assumes *length vts* ≥ *2*
shows *make-polygonal-path* (*map* (λ*x*. *x* + *u*) *vts*) = (λ*x*. *x* + *u*) ∘ (*make-polygonal-path vts*)
using *assms*
proof(*induct length vts arbitrary*: *u vts*)
case *0*
then show *?case* **by** *presburger*
next
case (*Suc n*)
let *?vts*′ = *map* (λ*x*. *x* + *u*) *vts*
let *?p*′ = *make-polygonal-path ?vts*′
{ assume *Suc n* = *2*
then obtain *a b* **where** *ab*: *vts* = [*a*, *b*]
by (*metis* (*no-types*, *lifting*) *One-nat-def Suc.hyps*(*2*) *Suc-1 Suc-length-conv length-0-conv*)
then have *?vts*′ = [(λ*x*. *x* + *u*) *a*, (λ*x*. *x* + *u*) *b*] **by** *simp*
then have *?p*′ = *linepath* ((λ*x*. *x* + *u*) *a*) ((λ*x*. *x* + *u*) *b*)
using *make-polygonal-path.simps*(*3*) **by** *presburger*
also have ... = (λ*x*. *x* + *u*) ∘ (*linepath a b*) **using** *linepath-translation* **by** *auto*
also have ... = (λ*x*. *x* + *u*) ∘ (*make-polygonal-path vts*) **using** *ab* **by** *auto*
finally have *?case* **.**
} moreover
{ assume ∗: *Suc n* > *2*

```
    then obtain a b c rest where abc: vts = a # b # c # rest
      by (metis One-nat-def Suc.hyps(2) Suc-1 Suc-leI Suc-le-length-iff)

    let ?vts-tl = tl vts
    let ?p-tl = make-polygonal-path ?vts-tl
    let ?vts'-tl = map (λx. x + u) ?vts-tl
    let ?p'-tl = make-polygonal-path ?vts'-tl

    have ?vts'-tl = tl ?vts' by (simp add: map-tl)
    then have ?p' = (linepath (?vts'!0) (?vts'!1)) +++ ?p'-tl
      using make-polygonal-path.simps(4) abc by force
    moreover have ?p'-tl = (λx. x + u) ∘ (?p-tl) using Suc.hyps(1) Suc.hyps(2)
∗ by force
    moreover have (linepath (?vts'!0) (?vts'!1)) = (λx. x + u) ∘ (linepath a b)
      using abc linepath-translation by auto
    ultimately have ?case by (simp add: abc path-compose-join)
  }
  ultimately show ?case using Suc by linarith
qed

lemma translation-is-polygon:
  assumes polygon-of p vts
  shows polygon-of ((λx. x + u) ∘ p) (map (λx. x + u) vts) (is polygon-of ?p'
?vts')
proof−
  have length vts ≥ 3
    by (metis One-nat-def Suc-eq-plus1 Suc-le-eq add-Suc-right assms nat-less-le nu-
meral-3-eq-3 numeral-Bit0 one-add-one polygon-of-def polygon-vertices-length-at-least-4)
  then have ∗: ?p' = make-polygonal-path ?vts'
    using make-polygonal-path-translate assms unfolding polygon-of-def by force
  moreover have polygon ?p'
  proof−
    have polygonal-path ?p' unfolding polygonal-path-def using ∗ by simp
    moreover have simple-path ?p'
      using assms unfolding polygon-of-def polygon-def
      using simple-path-translation-eq[of u p]
      by (metis add.commute fun.map-cong)
    moreover have closed-path ?p'
    proof−
      have ?p' 0 = p 0 + u by simp
      moreover have ?p' 1 = p 1 + u by simp
      moreover have p 0 = p 1
        using assms
        unfolding polygon-of-def polygon-def closed-path-def pathstart-def pathfin-
ish-def
        by blast
      moreover have path ?p' using make-polygonal-path-gives-path ∗ by simp
      ultimately show ?thesis
        unfolding closed-path-def pathstart-def pathfinish-def
```

by *argo*
qed
ultimately show *?thesis* **unfolding** *polygon-def* **by** *blast*
qed
ultimately show *?thesis* **unfolding** *polygon-of-def* **by** *blast*
qed

11 Misc. properties

lemma *tail-of-loop-free-polygonal-path-is-loop-free*:
assumes *loop-free* (*make-polygonal-path* (*x*#*tail*)) (**is** *loop-free ?p*) **and**
length tail ≥ *2*
shows *loop-free* (*make-polygonal-path tail*) (**is** *loop-free ?p'*)
proof−
obtain *y z tail'* **where** *tail'*: *tail* = *y* # *z* # *tail'*
by (*metis One-nat-def Suc-1 assms*(*2*) *length-Cons list.exhaust-sel list.size*(*3*) *not-less-eq-eq zero-le*)
have *path ?p* ∧ *path ?p'* **using** *make-polygonal-path-gives-path* **by** *auto*
have *loop-free ?p* **using** *assms* **unfolding** *simple-path-def* **by** *auto*
moreover have *?p* = (*linepath x y*) +++ *?p'*
using *tail' make-polygonal-path.simps*(*4*) **by** (*simp add*: *tail'*)
moreover from *calculation* **have** *loop-free ?p'*
by (*metis make-polygonal-path-gives-path not-loop-free-second-component path-join-path-ends*)
ultimately show *?thesis*
using *make-polygonal-path-gives-path simple-path-def* **by** *blast*
qed

lemma *tail-of-simple-polygonal-path-is-simple*:
assumes *simple-path* (*make-polygonal-path* (*x*#*tail*)) (**is** *simple-path ?p*) **and**
length tail ≥ *2*
shows *simple-path* (*make-polygonal-path tail*) (**is** *simple-path ?p'*)
using *tail-of-loop-free-polygonal-path-is-loop-free* **unfolding** *simple-path-def*
using *assms*(*1*) *assms*(*2*) *make-polygonal-path-gives-path simple-path-def* **by** *blast*

lemma *interior-vtx-in-path-image-interior*:
fixes *vts* :: (*real*^*2*) *list*
assumes *x* ∈ *set* (*butlast* (*drop 1 vts*))
shows ∃ *t*. *t* ∈ {*0*<..<*1*} ∧ (*make-polygonal-path vts*) *t* = *x*
using *assms*
proof(*induct vts rule*: *make-polygonal-path.induct*)
case *1*
then show *?case* **by** *simp*
next
case (*2 a*)
then show *?case* **by** *simp*
next
case (*3 a b*)
then show *?case* **by** *simp*
next

```
  case ih: (4 a b c tail')
  let ?vts = a # b # c # tail'
  let ?tl = b # c # tail'
  let ?p = make-polygonal-path ?vts
  let ?p-tl = make-polygonal-path ?tl
  { assume x ∈ set (butlast (drop 1 ?tl))
    then obtain t' where t': t' ∈ {0<..<1} ∧ ?p-tl t' = x using ih by blast
    then have ?p ((t' + 1) / 2) = x
      unfolding make-polygonal-path.simps joinpaths-def
    by (smt (verit, del-insts) field-sum-of-halves greaterThanLessThan-iff mult-2-right
not-numeral-le-zero zero-le-divide-iff)
    moreover have (t' + 1) / 2 ∈ {0<..<1} using t' by force
    ultimately have ?case
      by blast
  } moreover
  { assume x ∉ set (butlast (drop 1 ?tl))
    then have x = b
    by (metis One-nat-def butlast.simps(2) drop0 drop-Suc-Cons ih.prems list.distinct(1)
set-ConsD)
    then have ?p (1/2) = x unfolding make-polygonal-path.simps joinpaths-def
      by (simp add: linepath-1')
    moreover have ((1/2)::(real)) ∈ ({0<..<1}::(real set)) by simp
    ultimately have ?case by blast
  }
  ultimately show ?case by auto
qed

lemma loop-free-polygonal-path-vts-distinct:
  assumes loop-free (make-polygonal-path vts)
  shows distinct (butlast vts)
  using assms
proof(induct vts rule: make-polygonal-path.induct)
  case 1
  then show ?case by simp
next
  case (2 a)
  then show ?case by simp
next
  case (3 a b)
  then show ?case by simp
next
  case ih: (4 a b c tail')
  let ?vts = a # b # c # tail'
  let ?tl = b # c # tail'
  let ?p = make-polygonal-path ?vts
  let ?p-tl = make-polygonal-path ?tl

  have distinct (butlast ?tl)
    using ih tail-of-loop-free-polygonal-path-is-loop-free by simp
```

moreover have $a \notin set\ (butlast\ ?tl)$
proof(*rule ccontr*)
assume *a-in*: $\neg\ a \notin set\ (butlast\ ?tl)$
then have $a \in set\ (butlast\ (drop\ 1\ ?vts))$ **by** *simp*
then obtain t **where** t: $t \in \{0<..<1\} \wedge\ ?p\ t = a$
using *vertices-on-path-image interior-vtx-in-path-image-interior* **by** *metis*
then show *False*
using *ih.prems* **unfolding** *simple-path-def loop-free-def*
by (*metis atLeastAtMost-iff greaterThanLessThan-iff less-eq-real-def less-numeral-extra*(*3*)
less-numeral-extra(*4*) *list.distinct*(*1*) *nth-Cons-0 path-defs*(*2*) *polygon-pathstart zero-less-one-class.zero-le-one*)
qed
ultimately show *?case* **by** *simp*
qed

lemma *loop-free-polygonal-path-vts-drop1-distinct*:
assumes *loop-free* (*make-polygonal-path vts*)
shows *distinct* (*drop 1 vts*)
proof −
let *?p* = *make-polygonal-path vts*
let *?last-vts* = *vts* ! ((*length vts*) − *1*)
have *distinct* (*butlast vts*)
using *assms loop-free-polygonal-path-vts-distinct*
by *auto*
then have *distinct-butlast*: *distinct* (*butlast* (*drop 1 vts*))
by (*metis distinct-drop drop-butlast*)
{assume ∗: *length vts* > *1*
have *len-drop1*: *length* (*drop 1 vts*) = (*length vts*) − *1*
using ∗ **by** *simp*
have *simp-len*: *1* + ((*length vts*) − *2*) = (*length vts*) − *1*
using ∗ **by** *simp*
then have *vts-access*: *vts* ! (*1* + (*length vts* − *2*)) = *vts* ! ((*length vts*) − *1*)
by *argo*
have *drop 1 vts* ! ((*length vts*) − *2*) = *vts* ! (*1* + (*length vts* − *2*))
using ∗ **using** *nth-drop*[*of 1 vts* (*length vts*) − *2*] **by** *auto*
then have *?last-vts* = (*drop 1 vts*) ! ((*length vts*) − *2*)
using ∗ *simp-len vts-access* **by** *argo*
then have *?last-vts* = (*drop 1 vts*) ! (*length* (*drop 1 vts*) − *1*)
using ∗ *len-drop1*
using *diff-diff-left nat-1-add-1* **by** *presburger*
then have *drop1-is*: *drop 1 vts* = (*butlast* (*drop 1 vts*))@[*?last-vts*]
using ∗
by (*metis append-butlast-last-id drop-eq-Nil leD length-butlast nth-append-length*)
have *last-vts-not-in*: *?last-vts* ∉ *set* (*butlast* (*drop 1 vts*))
proof(*rule ccontr*)
assume *a-in*: ¬ *?last-vts* ∉ *set* (*butlast* (*drop 1 vts*))
then have *?last-vts* ∈ *set* (*butlast* (*drop 1 vts*)) **by** *simp*
then obtain t **where** t: $t \in \{0<..<1\} \wedge\ ?p\ t = ?last\text{-}vts$
using *vertices-on-path-image interior-vtx-in-path-image-interior* **by** *metis*

```
    have vts ! (length vts − 1) = ?p 1
      using polygon-pathfinish[of vts ?p] ∗
      by (metis list.size(3) not-one-less-zero pathfinish-def)
    then show False
      using t assms unfolding loop-free-def
    by (metis atLeastAtMost-iff greaterThanLessThan-iff leD less-eq-real-def zero-less-one-class.zero-le-one)
  qed
  have ⋀b::(real^2) list. distinct b ∧ a ∉ set b ⟹ distinct (b @[a]) for a::real^2
    by simp
  then have ?thesis using last-vts-not-in drop1-is distinct-butlast by metis
  }
   then show ?thesis by force
qed


lemma simple-polygonal-path-vts-distinct:
  assumes simple-path (make-polygonal-path vts)
  shows distinct (butlast vts)
  using assms loop-free-polygonal-path-vts-distinct
  unfolding simple-path-def
  by blast

lemma edge-subset-path-image:
  assumes p = make-polygonal-path vts and
          (i::int) ∈ {0..<((length vts) − 1)} and
          x = vts!i and
          y = vts!(i+1)
  shows path-image (linepath x y) ⊆ path-image p (is ?xy-img ⊆ ?p-img)
  using assms
proof(induct vts arbitrary: p i rule: make-polygonal-path.induct)
  case 1
  then show ?case by simp
next
  case (2 a)
  then show ?case by simp
next
  case (3 a b)
  then show ?case by (simp add: nth-Cons')
next
  case ih: (4 a b c tl)
  let ?tl = b # c # tl
  let ?p-tl = make-polygonal-path (?tl)
  { assume i = 0
    then have ?case
    by (metis (mono-tags, lifting) ih(2) ih(4) ih(5) Suc-eq-plus1 UnCI list.distinct(1)
make-polygonal-path.simps(4) nth-Cons-0 nth-Cons-Suc path-image-join pathfin-
ish-linepath polygon-pathstart subsetI)
  } moreover
  { assume i > 0
```

then have $x = ?tl!(i-1)$ **by** (*simp add*: *ih.prems*(*3*))
moreover have $y = ?tl!i$ **by** (*simp add*: *ih.prems*(*4*))
moreover have $i - 1 \in \{0..<(length\ (?tl) - 1)\}$ **using** *ih.prems*(*2*) **by** *force*
ultimately have *?xy-img* ⊆ *path-image ?p-tl* **using** *ih*(*1*) **by** (*simp add*: ‹*0* < *i*›)
then have *?case*
unfolding *ih*(*2*) *make-polygonal-path.simps*
by (*smt* (*verit*, *ccfv-SIG*) *UnCI make-polygonal-path.simps*(*4*) *make-polygonal-path-gives-path path-image-join path-join-path-ends subsetI subset-iff*)
}
ultimately show *?case* **by** *linarith*
qed

12 Properties of Sublists of Polygonal Path Vertex Lists

lemma *make-polygonal-path-image-append-var*:
assumes *length vts1* ≥ *2*
shows *path-image* (*make-polygonal-path* (*vts1* @ [*v*])) = *path-image* (*make-polygonal-path vts1* +++ (*linepath* (*vts1* ! (*length vts1* − *1*)) *v*))
using *assms*
proof (*induct vts1*)
case *Nil*
then show *?case* **by** *auto*
next
case (*Cons a vts1*)
{assume ∗: *length vts1* = *1*
then obtain *b* **where** *vts1* = [*b*]
by (*metis Cons-nth-drop-Suc One-nat-def drop0 drop-eq-Nil le-numeral-extra*(*4*) *less-numeral-extra*(*1*))
then have *path-image* (*make-polygonal-path* ((*a* # *vts1*) @ [*v*])) =
path-image (*make-polygonal-path* (*a* # *vts1*) +++ *linepath* ((*a* # *vts1*) ! (*length* (*a* # *vts1*) − *1*)) *v*)
using *make-polygonal-path.simps*
by *simp*
} **moreover {assume** ∗ : *length vts1* > *1*
then obtain *b c vts1′* **where** *vts1* = *b* # *c* # *vts1′*
by (*metis One-nat-def length-0-conv length-Cons less-numeral-extra*(*4*) *not-one-less-zero remdups-adj.cases*)
then have *h1*: *make-polygonal-path* ((*a* # *vts1*) @ [*v*]) = (*linepath a b*) +++ (*make-polygonal-path* (*vts1* @ [*v*]))
using *make-polygonal-path.simps*(*4*)
by *auto*
have *path-image* (*make-polygonal-path* (*vts1* @ [*v*])) =
path-image (*make-polygonal-path vts1* +++ *linepath* (*vts1* ! (*length vts1* − *1*)) *v*)
using ∗ *Cons* **by** *auto*
then have *path-image* (*make-polygonal-path* ((*a* # *vts1*) @ [*v*])) =

path-image (*make-polygonal-path* (*a* # *vts1*) +++ *linepath* ((*a* # *vts1*) ! (*length* (*a* # *vts1*) − *1*)) *v*)
using *h1*
by (*metis* (*no-types*, *lifting*) *Cons.prems Suc-1 Suc-le-eq Un-assoc* ‹*vts1* = *b* # *c* # *vts1'*› *add-diff-cancel-left' append-Cons length-Cons list.discI make-polygonal-path.simps*(*4*) *nth-Cons-0 nth-Cons-pos path-image-join pathfinish-linepath pathstart-linepath plus-1-eq-Suc polygon-pathfinish polygon-pathstart zero-less-diff*)
}
ultimately show *?case*
by (*metis Cons.prems Suc-1 add-diff-cancel-left' le-neq-implies-less length-Cons not-less-eq plus-1-eq-Suc*)
qed

lemma *make-polygonal-path-image-append-helper*:
assumes *length vts1* ≥ *1* ∧ *length vts2* ≥ *1*
shows *path-image* (*make-polygonal-path* (*vts1* @ [*v*] @ [*v*] @ *vts2*)) = *path-image* (*make-polygonal-path* (*vts1* @ [*v*] @ *vts2*))
using *assms*
proof (*induct vts1*)
case *Nil*
then show *?case* **by** *auto*
next
case (*Cons a vts1*)
{ assume ∗: *length vts1* = *0*
have *path-image* (*make-polygonal-path* ([*a*] @ [*v*] @ *vts2*)) =
path-image ((*linepath a v*) +++ *make-polygonal-path* (*v* # *vts2*))
using *make-polygonal-path.simps*
by (*metis Cons.prems One-nat-def append-Cons append-Nil append-take-drop-id linorder-not-le list.distinct*(*1*) *list.exhaust not-less-eq-eq take-hd-drop*)
then have *path-image* (*make-polygonal-path* ([*a*] @ [*v*] @ *vts2*)) =
path-image (*linepath a v*) ∪ *path-image* (*make-polygonal-path* (*v* # *vts2*))
by (*metis list.discI nth-Cons-0 path-image-join pathfinish-linepath polygon-pathstart*)
have *image-helper1*: *path-image* (*make-polygonal-path* ([*a*] @ [*v*] @ [*v*] @ *vts2*)) = *path-image* (*linepath a v* +++ *make-polygonal-path* (*v* # *v* # *vts2*))
by *simp*
have *path-image* (*make-polygonal-path* (*v* # *v* # *vts2*)) = *path-image* ((*linepath v v*) +++ *make-polygonal-path* (*v* # *vts2*))
using *make-polygonal-path.simps*
by (*metis Cons.prems One-nat-def append-Cons append-Nil append-take-drop-id linorder-not-le list.distinct*(*1*) *list.exhaust not-less-eq-eq take-hd-drop*)
moreover have ... = *path-image* (*linepath v v*) ∪ *path-image* (*make-polygonal-path* (*v* # *vts2*))
by (*metis list.discI nth-Cons-0 path-image-join pathfinish-linepath polygon-pathstart*)
ultimately have *image-helper2*: *path-image* (*make-polygonal-path* (*v* # *v* # *vts2*)) = {*v*} ∪ *path-image* (*make-polygonal-path* (*v* # *vts2*))
by *auto*
have *v* ∈ *path-image* (*make-polygonal-path* (*v* # *vts2*))
using *vertices-on-path-image* **by** *fastforce*

then have *path-image* (*make-polygonal-path* ([*a*] @ [*v*] @ [*v*] @ *vts2*)) =
path-image (*make-polygonal-path* ([*a*] @ [*v*] @ *vts2*))
using *image-helper1 image-helper2*
by (*metis* ‹*path-image* (*make-polygonal-path* ([*a*] @ [*v*] @ *vts2*)) = *path-image* (*linepath a v*) ∪ *path-image* (*make-polygonal-path* (*v* # *vts2*))› *insert-absorb insert-is-Un list.simps*(*3*) *nth-Cons-0 path-image-join pathfinish-linepath polygon-pathstart*)
}
moreover {assume ∗: *length vts1* > *0*
then have *ind-hyp*: *path-image* (*make-polygonal-path* (*vts1* @ [*v*] @ [*v*] @ *vts2*)) =
path-image (*make-polygonal-path* (*vts1* @ [*v*] @ *vts2*))
using *Cons.hyps Cons.prems* **by** *linarith*
obtain *b vts3* **where** *vts1-is*: *vts1* = *b*#*vts3*
using ∗
by (*metis* ∗ *Cons-nth-drop-Suc drop0*)
then have *path-image1*: *path-image* (*make-polygonal-path* ((*a* # *vts1*) @ [*v*] @ [*v*] @ *vts2*)) =
path-image ((*linepath a b*) +++ *make-polygonal-path* (*vts1* @ [*v*] @ [*v*] @ *vts2*))
by (*smt* (*verit*, *best*) *Cons.prems Nil-is-append-conv append-Cons length-greater-0-conv less-numeral-extra*(*1*) *list.inject make-polygonal-path.elims order-less-le-trans*)
obtain *c d* **where** *bcd*: *vts1* @ [*v*] @ *vts2* = *b* # *c* # *d*
using *vts1-is*
by (*metis append-Cons append-Nil neq-Nil-conv*)
have *path-image2*: *path-image* (*make-polygonal-path* ((*a* # *vts1*) @ [*v*] @ *vts2*)) = *path-image* ((*linepath a b*) +++ *make-polygonal-path* (*vts1* @ [*v*] @ *vts2*))
using *make-polygonal-path.simps bcd*
by *auto*
have *path-image* (*make-polygonal-path* ((*a* # *vts1*) @ [*v*] @ [*v*] @ *vts2*)) =
path-image (*make-polygonal-path* ((*a* # *vts1*) @ [*v*] @ *vts2*))
using *ind-hyp path-image1 path-image2*
by (*smt* (*verit*, *del-insts*) *Nil-is-append-conv append-Cons nth-Cons-0 path-image-join pathfinish-linepath polygon-pathstart vts1-is*)
}
ultimately show *?case*
using *Cons.prems*
by *blast*
qed

lemma *make-polygonal-path-image-append*:
assumes *length vts1* ≥ *2* ∧ *length vts2* ≥ *2*
shows *path-image* (*make-polygonal-path* (*vts1* @ *vts2*)) = *path-image* (*make-polygonal-path vts1* +++ (*linepath* (*vts1* ! (*length vts1* − *1*)) (*vts2* ! *0*)) +++ *make-polygonal-path vts2*)
using *assms*
proof (*induct vts1*)
case *Nil*
then show *?case*
by *simp*

next
case (*Cons a vts1*)
{assume ∗: *length vts1 = 1*
then obtain *b* **where** *vts1-is*: *vts1* = [*b*]
by (*metis Cons-nth-drop-Suc One-nat-def drop0 drop-eq-Nil le-numeral-extra*(*4*) *less-numeral-extra*(*1*))
then have *make-polygonal-path* ((*a* # *vts1*) @ *vts2*) = *make-polygonal-path* (*a* # *b* # *vts2*)
by *simp*
then have *make-polygonal-path* ((*a* # *vts1*) @ *vts2*) = (*linepath a b*) +++ (*make-polygonal-path* (*b* # *vts2*))
by (*metis Cons.prems length-0-conv make-polygonal-path.simps*(*4*) *neq-Nil-conv not-numeral-le-zero*)
then have *make-polygonal-path* ((*a* # *vts1*) @ *vts2*) = *make-polygonal-path* (*a* # *vts1*) +++ (*make-polygonal-path* (*b* # *vts2*))
using *vts1-is make-polygonal-path.simps*(*3*)
by *simp*
then have *make-polygonal-path* ((*a* # *vts1*) @ *vts2*) = *make-polygonal-path* (*a* # *vts1*) +++ *linepath b* (*vts2* ! *0*) +++ *make-polygonal-path vts2*
using *Cons.prems*
by (*smt* (*verit*, *ccfv-SIG*) ∗ *Suc-1 add-diff-cancel-left*′ *diff-is-0-eq*′ *length-greater-0-conv list.size*(*4*) *make-polygonal-path.elims make-polygonal-path.simps*(*4*) *nth-Cons-0 order-less-le-trans plus-1-eq-Suc pos2 vts1-is zero-neq-one*)
then have *make-polygonal-path* ((*a* # *vts1*) @ *vts2*) = *make-polygonal-path* (*a* # *vts1*) +++ *linepath* ((*a* # *vts1*) ! (*length* (*a* # *vts1*) − *1*)) (*vts2* ! *0*) +++ *make-polygonal-path vts2*
using *vts1-is*
by *simp*
} moreover {assume ∗: *length vts1 > 1*
then obtain *b c vts1*′ **where** *vts1*′: *vts1* = *b* # *c* # *vts1*′
by (*metis One-nat-def length-0-conv length-Cons less-numeral-extra*(*4*) *not-one-less-zero remdups-adj.cases*)
then have *h1*: *make-polygonal-path* ((*a* # *vts1*) @ *vts2*) = (*linepath a b*) +++ (*make-polygonal-path* (*vts1* @ *vts2*))
using *make-polygonal-path.simps*(*4*)
by *auto*
have *ind-h*: *path-image* (*make-polygonal-path* (*vts1* @ *vts2*)) = *path-image* (*make-polygonal-path vts1* +++ *linepath* (*vts1* ! (*length vts1* − *1*)) (*vts2* ! *0*) +++ *make-polygonal-path vts2*)
using *Cons* ∗ **by** *linarith*
then have *path-image* (*make-polygonal-path* ((*a* # *vts1*) @ *vts2*)) = *path-image* ((*linepath a b*)) ∪ *path-image*((*make-polygonal-path vts1* +++ *linepath* (*vts1* ! (*length vts1* − *1*)) (*vts2* ! *0*) +++ *make-polygonal-path vts2*))
by (*metis h1 make-polygonal-path-gives-path path-image-join path-join-path-ends*)
then have *path-image* (*make-polygonal-path* ((*a* # *vts1*) @ *vts2*)) = (*path-image* (*linepath a b*) ∪ *path-image* (*make-polygonal-path vts1*)) ∪ *path-image*((*linepath* (*vts1* ! (*length vts1* − *1*)) (*vts2* ! *0*) +++ *make-polygonal-path vts2*))

by (*metis* (*no-types*, *opaque-lifting*) ∗ *Un-assoc not-one-less-zero linepath-0′ list.size*(*3*)
path-image-join pathstart-def pathstart-join polygon-pathfinish)
then have *image-helper*: *path-image* (*make-polygonal-path* ((*a* # *vts1*) @ *vts2*))
= (*path-image* (*make-polygonal-path* (*a* # *vts1*))) ∪
path-image((*linepath* (*vts1* ! (*length vts1* − *1*)) (*vts2* ! *0*) +++ *make-polygonal-path vts2*))
by (*metis neq-Nil-conv nth-Cons′ path-image-cons-union vts1′*)
have *vts1* ! (*length vts1* − *1*) = (*a* # *vts1*) ! (*length* (*a* # *vts1*) − *1*)
using *Cons.prems*
by (*simp add*: *Suc-le-eq*)
then have *path-image* (*make-polygonal-path* ((*a* # *vts1*) @ *vts2*)) =
path-image
(*make-polygonal-path* (*a* # *vts1*) +++
linepath ((*a* # *vts1*) ! (*length* (*a* # *vts1*) − *1*)) (*vts2* ! *0*) +++ *make-polygonal-path vts2*)
using *image-helper*
by (*metis* (*no-types*, *lifting*) *Cons.prems length-greater-0-conv order-less-le-trans path-image-join pathstart-join pathstart-linepath polygon-pathfinish pos2*)
}
ultimately show *?case* **using** *Cons.prems*
by *fastforce*
qed

lemma *make-polygonal-path-image-append-alt*:
assumes *p* = *make-polygonal-path vts*
assumes *p1* = *make-polygonal-path vts1*
assumes *p2* = *make-polygonal-path vts2*
assumes *last vts1* = *hd vts2*
assumes *length vts1* ≥ *2* ∧ *length vts2* ≥ *2*
assumes *vts* = *vts1* @ (*tl vts2*)
shows *path-image p* = *path-image* (*p1* +++ *p2*)
proof−
have *path-image p* = *path-image p1* ∪ *path-image p2*
by (*smt* (*z3*) *Nitpick.size-list-simp*(*2*) *One-nat-def Suc-1 assms diff-Suc-1 last-conv-nth length-greater-0-conv list.collapse list.sel*(*3*) *make-polygonal-path.elims make-polygonal-path.simps*(*3*) *make-polygonal-path-image-append make-polygonal-path-image-append-var nat-less-le not-less-eq-eq nth-Cons-0 order-less-le-trans path-image-join polygon-pathfinish polygon-pathstart pos2 length-Cons length-tl path-image-cons-union pathfinish-linepath pathstart-join sup.absorb-iff1 sup.absorb-iff2*)
thus *?thesis*
by (*metis assms*(*2*) *assms*(*3*) *assms*(*4*) *assms*(*5*) *hd-conv-nth last-conv-nth length-greater-0-conv order-less-le-trans path-image-join polygon-pathfinish polygon-pathstart pos2*)
qed

lemma *cont-incr-interval-image*:
fixes *f* :: *real* ⇒ *real*
assumes *a* ≤ *b*

assumes *continuous-on* $\{a..b\}$ f
assumes $\forall x \in \{a..b\}.\ \forall y \in \{a..b\}.\ x \leq y \longrightarrow f\ x \leq f\ y$
shows $f`\{a..b\} = \{f\ a..f\ b\}$
proof−
have $f`\{a..b\} \subseteq \{f\ a..f\ b\}$
proof(*rule subsetI*)
fix x
assume $x \in f`\{a..b\}$
then obtain t **where** $t \in \{a..b\} \wedge f\ t = x$ **by** *blast*
moreover then have $a \leq t \wedge t \leq b$ **by** *presburger*
ultimately show $x \in \{f\ a..f\ b\}$ **using** *assms*(*3*) **by** *auto*
qed
moreover have $\{f\ a..f\ b\} \subseteq f`\{a..b\}$
proof−
obtain $c\ d$ **where** $f`\{a..b\} = \{c..d\}$ **using** *continuous-image-closed-interval assms* **by** *meson*
moreover then have $f\ a \in \{c..d\}$ **using** *assms*(*1*) **by** *auto*
moreover have $f\ b \in \{c..d\}$ **using** *assms*(*1*) *calculation* **by** *auto*
moreover have $\{f\ a..f\ b\} \subseteq \{c..d\}$ **using** *calculation* **by** *simp*
ultimately show *?thesis* **by** *presburger*
qed
ultimately show *?thesis* **by** *blast*
qed

lemma *two-x-minus-one-image*:
assumes $f = (\lambda x{::}real.\ 2{*}x - 1)$
assumes $a \leq b$
shows $f`\{a..b\} = \{f\ a..f\ b\}$
proof−
have *continuous-on* $\{a..b\}$ f
proof−
have *continuous-on* $\{a..b\}$ $(\lambda x{::}real.\ x)$ **by** *simp*
then have *continuous-on* $\{a..b\}$ $(\lambda x{::}real.\ 2{*}x)$ **using** *continuous-on-mult-const* **by** *blast*
thus *continuous-on* $\{a..b\}$ f
unfolding *assms* **using** *continuous-on-translation-eq*[*of* $\{a..b\}$ -1 $(\lambda x{::}real.\ 2{*}x)$] **by** *auto*
qed
thus *?thesis* **using** *cont-incr-interval-image assms* **by** *force*
qed

lemma *vts-split-path-image*:
assumes *p* = *make-polygonal-path vts*
assumes *p1* = *make-polygonal-path vts1*
assumes *p2* = *make-polygonal-path vts2*
assumes *vts1* = *take i vts*
assumes *vts2* = *drop* $(i-1)$ *vts*
assumes *n* = *length vts*
assumes $1 \leq i \wedge i < n$

assumes $x = (2\hat{}(i-1) - 1)/(2\hat{}(i-1))$
shows *path-image p1* $= p`\{0..x\} \wedge$ *path-image p2* $= p`\{x..1\}$
using *assms*
proof(*induct i arbitrary*: *p p1 p2 vts vts1 vts2 n x*)
case *0*
then show *?case* **by** *linarith*
next
case (*Suc i*)
{ assume $*$: *Suc i* $= 1$
then obtain *a* **where** *a*: *vts1* $= [a]$
using *Suc.prems*
by (*metis One-nat-def gr-implies-not0 list.collapse list.size*(*3*) *take-eq-Nil take-tl zero-neq-one*)
moreover have *vts2* $=$ *vts* **using** $*$ *Suc.prems* **by** *force*
ultimately have *p1* $=$ *linepath a a* $\wedge$ *p2* $= p$
using *Suc.prems make-polygonal-path.simps* **by** *meson*
moreover have $x = 0$ **using** *Suc.prems* $*$ **by** *simp*
moreover have *path-image p1* $= \{a\}$ **using** *calculation* **by** *simp*
moreover have $p`\{0..0\} = \{p\ 0\}$ **by** *auto*
moreover then have $p`\{0..0\} = \{a\}$ **using** *Suc.prems*
by (*metis a gr0-conv-Suc list.discI nth-Cons-0 nth-take pathstart-def polygon-pathstart take-eq-Nil*)
moreover have *path-image p1* $= p`\{0..x\}$ **using** *calculation* **by** *presburger*
moreover have *path-image p2* $= p`\{x..1\}$ **using** *calculation* **unfolding** *path-image-def* **by** *fast*
ultimately have *?case* **by** *blast*
} moreover
{ assume $*$: *Suc i* > 1

let *?a* $=$ *vts*$!0$
let *?b* $=$ *vts*$!1$
let *?l* $=$ *linepath ?a ?b*
let *?L* $=$ *path-image ?l*
let *?tl* $=$ *tl vts*
let *?vts1'* $=$ *take i ?tl*
let *?vts2'* $=$ *drop* $(i-1)$ *?tl*
let *?p'* $=$ *make-polygonal-path ?tl*
let *?p1'* $=$ *make-polygonal-path ?vts1'*
let *?p2'* $=$ *make-polygonal-path ?vts2'*
let *?x'* $= ((2::real)\hat{}(i-1)-1)/(2\hat{}(i-1))$
let *?P1'* $=$ *path-image ?p1'*
let *?P2'* $=$ *path-image ?p2'*

have *i*: $1 \leq i \wedge i <$ *length ?tl*
using *Suc.prems* $*$ **by** (*metis Suc-eq-plus1 length-tl less-Suc-eq-le less-diff-conv*)
then have *ih*: *?P1'* $=$ *?p'*$`\{0..?x'\} \wedge$ *?P2'* $=$ *?p'*$`\{?x'..1\}$
using *Suc.hyps*[*of ?p' ?tl ?p1' ?vts1' ?p2' ?vts2' length ?tl ?x'*] **by** *presburger*

let *?f* $= \lambda x::real.\ 2*x - 1$

have *fx*: *?f x = ?x′*
by (*metis i Suc.prems*(*8*) *bounding-interval-helper1 diff-Suc-1 summation-helper*)

moreover have *fhalf*: *?f* (*1/2*) = *0* **by** *simp*
moreover have *f1*: *?f 1 = 1* **by** *simp*
ultimately have *f*: *?f*‘{*x..1*} = {*?x′..1*} ∧ *?f*‘{*1/2..x*} = {*0..?x′*}
using *two-x-minus-one-image* **by** *auto*
have *x*: *1/2* ≤ *x* ∧ *x* ≤ *1*
by (*smt* (*verit*) *divide-le-eq-1-pos divide-nonneg-nonneg fhalf fx two-realpow-ge-one*)

have *n* ≥ *3* **using** *Suc.prems* ∗ **by** *linarith*
then have *p*: *p = ?l +++ ?p′*
proof −
have *f1*: ∀ *vs*. (*vs*::(*real*, *2*) *vec list*) ≠ [] ∨ ¬ *1* < *Suc* (*length vs*)
by *simp*
have *1* < *Suc n*
using *Suc.prems*(*7*) **by** *linarith*
then show *?thesis*
by (*smt* (*verit*) *f1 Suc-le-lessD i One-nat-def Suc.prems*(*6*) *Suc.prems*(*7*)
Suc-less-eq ‹*p = make-polygonal-path vts*› *hd-conv-nth length-Cons length-tl less-Suc-eq list.collapse list.exhaust make-polygonal-path.simps*(*4*) *nth-Cons-Suc zero-order*(*3*))

qed
have *p-to-p′*: ∀ *y* ≥ *1/2*. *p y* = (*?p′* ∘ *?f*) *y*
proof *clarify*
fix *y* :: *real*
assume ∗: *y* ≥ *1/2*
{ assume ∗∗: *y = 1/2*
then have *p y = ?b*
by (*smt* (*verit*) *fhalf joinpaths-def linepath-1′ p*)
moreover have *?f y = 0* **using** ∗∗ **by** *simp*
moreover have *?p′ 0 = ?b*
by (*metis i One-nat-def Suc.prems*(*6*) *length-greater-0-conv length-tl*
list.size(*3*) *nth-tl pathstart-def polygon-pathstart zero-order*(*3*))
ultimately have *p y* = (*?p′* ∘ *?f*) *y* **by** *simp*
} moreover
{ assume ∗∗: *y > 1/2*
then have *p y = ?p′* (*?f y*) **unfolding** *p joinpaths-def* **by** *simp*
then have *p y* = (*?p′* ∘ *?f*) *y* **by** *force*
}
ultimately show *p y* = (*?p′* ∘ *?f*) *y* **using** ∗ **by** *fastforce*
qed

have {*0..x*} = {*0..1/2*} ∪ {*1/2..x*} **using** *x* **by** (*simp add*: *ivl-disj-un-two-touch*(*4*))
then have *p*‘{*0..x*} = *p*‘{*0..1/2*} ∪ *p*‘{*1/2..x*} **by** *blast*
also have ... = *?L* ∪ *p*‘{*1/2..x*}
proof−

```
      have ?L ⊆ p‘{0..1/2}
      proof(rule subsetI)
        fix a
        assume ∗: a ∈ ?L
        then obtain t where t: t ∈ {0..1} ∧ ?l t = a unfolding path-image-def
by blast
        then have p (t/2) = a unfolding p joinpaths-def by auto
        moreover have t/2 ∈ {0..1/2} using t by simp
        ultimately show a ∈ p‘{0..1/2} by blast
      qed
      moreover have p‘{0..1/2} ⊆ ?L
      proof(rule subsetI)
        fix a
        assume ∗: a ∈ p‘{0..1/2}
        then obtain t where t ∈ {0..1/2} ∧ p t = a by blast
      moreover then have ?l (2∗t) = p t unfolding p joinpaths-def by presburger
        moreover have 2∗t ∈ {0..1} using calculation by simp
        ultimately show a ∈ ?L unfolding path-image-def by auto
      qed
      ultimately have ?L = p‘{0..1/2} by blast
      thus ?thesis by presburger
    qed
    also have ... = ?L ∪ (?p′ ∘ ?f)‘{1/2..x} using p-to-p′ by simp
    also have ... = ?L ∪ ?p′‘{0..?x′} using f by (metis image-comp)
    also have ... = ?L ∪ ?P1′ using ih by blast
    also have ... = path-image p1
    proof−
       have take i (tl vts) ≠ [] by (metis i less-zeroE list.size(3) not-one-le-zero
take-eq-Nil2)
      thus ?thesis using path-image-cons-union[of p1 vts1 ?p1′ ?vts1′ ?a ?b]
       by (metis ∗ Nitpick.size-list-simp(2) One-nat-def Suc.prems(2) Suc.prems(4)
Suc.prems(6) Suc.prems(7) bot-nat-0.extremum-strict hd-conv-nth length-greater-0-conv
nth-take nth-tl take-Suc take-tl)
    qed
    finally have 1: path-image p1 = p‘{0..x} by argo

    have p‘{x..1} = (?p′ ∘ ?f)‘{x..1} using p-to-p′ x by simp
    also have ... = ?p′‘{?x′..1} using f by (metis image-comp)
    also have ... = ?P2′ using ih by presburger
    also have ... = path-image p2
      using path-image-cons-union
       by (metis Suc.prems(3) Suc.prems(5) diff-Suc-1 drop-Suc gr0-implies-Suc i
linorder-neqE-nat not-less-zero not-one-le-zero)
    finally have 2: path-image p2 = p‘{x..1} by argo

    have ?case using 1 2 by fast
  }
  ultimately show ?case using Suc.prems by linarith
qed
```

lemma *drop-i-is-loop-free*:
fixes *vts* :: (*real^2*) *list*
assumes *m* = *length vts*
assumes *i* ≤ *m* − *2*
assumes *vts'* = *drop i vts*
assumes *p* = *make-polygonal-path vts*
assumes *p'* = *make-polygonal-path vts'*
assumes *loop-free p*
shows *loop-free p'*
using *assms*
proof(*induct i arbitrary*: *vts' p'*)
case *0*
then show *?case* **by** *simp*
next
case (*Suc i*)

let *?vts''* = *drop i vts*
let *?p''* = *make-polygonal-path ?vts''*
have *ih*: *loop-free ?p''*
using *Suc.hyps Suc.prems*(*2*) *Suc.prems*(*6*) *Suc-leD assms*(*1*) *assms*(*4*) **by** *blast*

obtain *a b* **where** *ab*: *?vts''* = *a* # *vts'* ∧ *b* = *vts'* ! *0*
by (*metis Cons-nth-drop-Suc Suc.prems*(*3*) *constant-linepath-is-not-loop-free drop-eq-Nil ih linorder-not-less make-polygonal-path.simps*(*1*))
then have *?vts''* = *a* # *b* # (*vts'* ! *1*) # (*drop 2 vts'*)
by (*smt* (*verit*, *ccfv-threshold*) *Cons-nth-drop-Suc Suc.prems*(*2*) *Suc.prems*(*3*) *Suc-1 Suc-diff-Suc Suc-le-eq assms*(*1*) *diff-Suc-1 diff-is-0-eq drop-drop le-add-diff-inverse length-drop nat-le-linear not-less-eq-eq zero-less-Suc*)
then have *?p''* = (*linepath a b*) +++ *p'*
using *make-polygonal-path.simps*(*4*)[*of a b vts'* ! *1 drop 2 vts'*] *Suc.prems* **by** (*simp add*: *ab*)
moreover have *pathfinish* (*linepath a b*) = *pathstart p'*
using *Suc.prems ab*
by (*metis constant-linepath-is-not-loop-free ih make-polygonal-path.simps*(*2*) *pathfinish-linepath polygon-pathstart*)
ultimately have *arc p'* **using** *simple-path-joinE*
by (*metis ih make-polygonal-path-gives-path simple-path-def*)
then show *?case* **using** *arc-imp-simple-path simple-path-def* **by** *blast*
qed

lemma *joinpaths-tl-transform*:
assumes *f* = (λ*x*::*real*. *2*∗*x* − *1*)
assumes *pathfinish g1* = *pathstart g2*
assumes *p* = *g1* +++ *g2*
assumes *x* ≥ *1*/*2*
shows *p x* = *g2* (*f x*)
proof−

```
  { assume x = 1/2
    moreover then have f x = 0 using assms by fastforce
    ultimately have p x = pathfinish g1 ∧ g2 (f x) = pathfinish g1
      using assms unfolding pathfinish-def pathstart-def joinpaths-def by force
    then have p x = g2 (f x) using assms unfolding joinpaths-def by simp
  } moreover
  { assume x > 1/2
    then have p x = g2 (f x) using assms unfolding joinpaths-def by simp
  }
  ultimately show p x = g2 (f x) using assms by fastforce
qed

lemma joinpaths-tl-image-transform:
  assumes f = (λx::real. 2*x - 1)
  assumes pathfinish g1 = pathstart g2
  assumes p = g1 +++ g2
  assumes 1/2 ≤ a ∧ a ≤ b
  shows p`{a..b} = g2`{f a..f b}
proof−
  have ∀ x ∈ {a..b}. p x = g2 (f x) using assms joinpaths-tl-transform[of f g1 g2
p] by force
  then have p`{a..b} = (g2 ∘ f)`{a..b} by simp
  also have ... = g2`{f a..f b} using two-x-minus-one-image by (metis assms(1,4)
image-comp)
  finally show ?thesis .
qed

lemma vts-sublist-path-image:
  assumes p = make-polygonal-path vts
  assumes p' = make-polygonal-path vts'
  assumes vts' = take j (drop i vts)
  assumes m = length vts
  assumes n = length vts'
  assumes k = i + j
  assumes k ≤ m − 1 ∧ 2 ≤ j
  assumes x1 = (2^i − 1)/(2^i)
  assumes x2 = (2^(k−1) − 1)/(2^(k−1))
  shows path-image p' = p`{x1..x2}
  using assms
proof(induct i arbitrary: vts p p' vts' m k x1 x2)
  case 0
  then show ?case using vts-split-path-image[of p drop 0 vts p' vts' - - j m x2]
   by (metis (no-types, opaque-lifting) Suc-diff-le add-0 cancel-comm-monoid-add-class.diff-cancel
diff-is-0-eq div-by-1 drop.simps(1) drop-0 le-add-diff-inverse length-drop less-one
linorder-not-le plus-1-eq-Suc pos2 power.simps(1))
next
  case (Suc i)

  let ?vts-tl = tl vts
```

let *?vts-tl′ = take j (drop i ?vts-tl)*
let *?p-tl = make-polygonal-path ?vts-tl*
let *?m′ = m−1*
let *?k′ = i+j*
let *?x1′ = (2^i − 1)/(2^i)*
let *?x2′ = (2^(?k′−1) − 1)/(2^(?k′−1))*
let *?f = λx. 2∗x − 1*

have *vts′ = ?vts-tl′* **using** *Suc.prems* **by** (*metis drop-Suc*)
then have *p′ = make-polygonal-path ?vts-tl′* **using** *Suc.prems* **by** *argo*
then have *ih*: *path-image p′ = ?p-tl‘{?x1′..?x2′}*
using *Suc.hyps*[*of ?p-tl ?vts-tl p′ ?vts-tl′ ?m′ ?k′ ?x1′ ?x2′*] *Suc.prems*
by (*smt* (*verit*, *ccfv-SIG*) *Suc-eq-plus1 add-diff-cancel-right′ add-leD1 diff-diff-left diff-is-0-eq drop-Suc le-add-diff-inverse length-tl linorder-not-le not-add-less2*)

let *?a = vts!0*
let *?b = vts!1*
let *?l = linepath ?a ?b*
have *p*: *p = ?l +++ ?p-tl*
proof−
have *length vts ≥ 3* **using** *Suc.prems* **by** *linarith*
then obtain *c w* **where** *vts = ?a # ?b # c # w*
by (*metis Cons-nth-drop-Suc One-nat-def Suc-le-eq drop0 numeral-3-eq-3 order-less-le*)
thus *?thesis*
using *Suc.prems make-polygonal-path.simps*(*4*)[*of ?a ?b c w*] **by** (*metis list.sel*(*3*))
qed
moreover have *x1 ≥ 1/2* **using** *Suc.prems* **by** (*simp add*: *plus-1-eq-Suc*)
moreover have *x2 ≥ x1*
using *Suc.prems*
by (*smt* (*verit*, *best*) *Nat.diff-add-assoc2 One-nat-def add-Suc-shift add-diff-cancel-left′ add-mono-thms-linordered-semiring*(*2*) *diff-add-cancel dual-order.trans group-cancel.rule0 numeral-One one-le-numeral one-le-power plus-1-eq-Suc power-increasing real-shrink-le trans-le-add2*)
moreover have *pathfinish ?l = pathstart ?p-tl*
by (*metis One-nat-def Suc.prems*(*4*) *Suc.prems*(*6*) *Suc.prems*(*7*) *Suc-neq-Zero add-is-0 diff-is-0-eq′ diff-zero length-tl linorder-not-less list.size*(*3*) *nth-tl pathfinish-linepath polygon-pathstart*)
ultimately have *p‘{x1..x2} = ?p-tl‘{?f x1..?f x2}*
using *joinpaths-tl-image-transform*[*of ?f ?l ?p-tl p x1 x2*] **by** *presburger*
also have *... = ?p-tl‘{?x1′..?x2′}*
by (*metis* (*no-types*, *lifting*) *Nat.add-diff-assoc Suc.prems*(*6−9*) *add.commute add-leD1 bounding-interval-helper1 diff-Suc-1 le-add2 nat-1-add-1 plus-1-eq-Suc summation-helper*)
also have *... = path-image p′* **using** *ih* **by** *blast*
finally show *?case* **by** *argo*
qed

lemma *one-append-simple-path*:
fixes *vts* :: (*real^2*) *list*
assumes *vts* = *vts'* @ [*z*]
assumes *n* = *length vts*
assumes *n* ≥ *3*
assumes *p* = *make-polygonal-path vts*
assumes *p'* = *make-polygonal-path vts'*
assumes *simple-path p*
shows *simple-path p'*
using *assms*
proof(*induct n arbitrary*: *vts vts' p p'*)
case *0*
then show *?case* **by** *linarith*
next
case (*Suc n*)
{ assume ∗: *Suc n* = *3*
then obtain *a b c* **where** *abc*: *vts* = [*a*, *b*, *c*] ∧ *vts'* = [*a*, *b*]
using *Suc.prems*
by (*smt* (*z3*) *Suc-le-length-iff Suc-length-conv append-Cons diff-Suc-1 drop0 length-0-conv length-append-singleton numeral-3-eq-3*)
then have *p'* = *linepath a b*
by (*simp add*: *Suc.prems*(*5*))
moreover have *a* ≠ *b* **using** *loop-free-polygonal-path-vts-distinct Suc.prems*
by (*metis abc butlast-snoc distinct-length-2-or-more simple-path-def*)
ultimately have *?case* **by** *blast*
} moreover
{ assume ∗: *Suc n* > *3*
then obtain *a b tl'* **where** *ab*: *vts'* = *a* # *tl'* ∧ *b* = *tl'*!*0* **using** *Suc.prems*
by (*metis Suc-le-length-iff Suc-le-mono length-append-singleton numeral-3-eq-3*)
moreover then have *p* = *make-polygonal-path* (*a* # (*tl'* @ [*z*])) **using** *Suc.prems*
by *auto*
moreover then have *p*: *p* = *linepath a b* +++ *make-polygonal-path* (*tl'* @ [*z*])
using *make-polygonal-path.simps ab*
by (*smt* (*verit*, *ccfv-threshold*) ∗ *Cons-nth-drop-Suc One-nat-def Suc.prems*(*1*) *Suc.prems*(*2*) *Suc-1 Suc-less-eq append-Cons drop0 length-Cons length-append-singleton length-greater-0-conv list.size*(*3*) *not-numeral-less-one numeral-3-eq-3*)
moreover then have *simple-path* ... **using** *Suc.prems* **by** *meson*
ultimately have *pre-ih*: *simple-path* (*make-polygonal-path* (*tl'* @ [*z*]))
using *Suc.prems*(*1*) *Suc.prems*(*2*) *Suc.prems*(*3*) *ab tail-of-simple-polygonal-path-is-simple*
by *simp*
then have *ih*: *simple-path* (*make-polygonal-path tl'*)
using *Suc.hyps* ∗ *Suc.prems*(*1*) *Suc.prems*(*2*) *ab* **by** *force*
have *simple-path* ((*linepath a b*) +++ *make-polygonal-path tl'*)
proof−
let *?g1* = *linepath a b*
let *?g2* = *make-polygonal-path tl'*
let *?G1* = *path-image ?g1*
let *?G2* = *path-image ?g2*
have *pathfinish ?g2* = *last tl'*

by (*metis constant-linepath-is-not-loop-free ih last-conv-nth make-polygonal-path.simps*(*1*) *polygon-pathfinish simple-path-def*)
also have ... = *vts* ! (*length vts* − *2*)
by (*metis ab Suc.prems*(*1*) *Suc-1 constant-linepath-is-not-loop-free diff-Suc-1 diff-Suc-Suc ih impossible-Cons last.simps last-conv-nth length-Cons length-append-singleton list.discI make-polygonal-path.simps*(*1*) *nle-le nth-append order-less-le simple-path-def*)
finally have *pathfinish-g2*: *pathfinish ?g2* = *vts* ! (*length vts* − *2*) **.**

have *pathfinish ?g1* = *pathstart ?g2*
by (*metis ab constant-linepath-is-not-loop-free ih linepath-1′ make-polygonal-path.simps*(*1*) *pathfinish-def polygon-pathstart simple-path-def*)
moreover have *arc ?g1*
by (*metis Suc.prems*(*6*) *p arc-linepath constant-linepath-is-not-loop-free not-loop-free-first-component simple-path-def*)
moreover have *arc ?g2*
proof−
have *pathstart ?g2* = *b*
using *calculation*(*1*) **by** *auto*
moreover have *b* = *vts*!*1*
by (*metis ab One-nat-def Suc.prems*(*1*) *Suc.prems*(*2*) *Suc.prems*(*3*) *Suc-le-eq length-append-singleton not-less-eq-eq nth-Cons-Suc nth-append numeral-3-eq-3*)
moreover have *last tl′* ≠ *vts*!*1*
using *loop-free-polygonal-path-vts-distinct Suc.prems*
by (*metis pre-ih ab append-Nil append-butlast-last-id butlast-conv-take butlast-snoc calculation*(*2*) *constant-linepath-is-not-loop-free hd-conv-nth ih index-Cons index-last list.collapse make-polygonal-path.simps*(*2*) *simple-path-def take0*)
ultimately have *pathfinish ?g2* ≠ *b*
using *pathfinish-g2* ‹*pathfinish* (*make-polygonal-path tl′*) = *last tl′*› **by** *presburger*
thus *?thesis*
using ‹*pathstart* (*make-polygonal-path tl′*) = *b*› *arc-simple-path ih* **by** *blast*
qed
moreover have *?G1* ∩ *?G2* ⊆ {*pathstart ?g2*}
proof(*rule subsetI*)
let *?z* = ((*2*::*real*)^(*n*−*1*) − *1*)/(*2*^(*n*−*1*))
have *g1*: *?G1* = *p*‘{*0*..*1*/*2*}
proof−
have *take 2 vts* = [*a*, *b*]
by (*smt* (*verit*) ∗ *One-nat-def Suc.prems*(*1*) *Suc.prems*(*2*) *Suc-1 ab append-Cons butlast-snoc drop0 drop-Suc-Cons length-append-singleton less-Suc-eq-le not-less-eq-eq nth-butlast numeral-3-eq-3 plus-1-eq-Suc same-append-eq take-Suc-Cons take-Suc-eq take-add take-all-iff*)
then have *?g1* = *make-polygonal-path* (*take 2 vts*)
using *make-polygonal-path.simps* **by** *presburger*
moreover have *1* < *n* **using** ∗ **by** *linarith*
ultimately have *?G1* = *p*‘{*0*..(*2*^(*2*−*1*) − *1*)/(*2*^(*2*−*1*))}
using *vts-split-path-image*
by (*metis* ∗ *Suc.prems*(*2*) *Suc.prems*(*4*) *Suc-1 Suc-leD Suc-lessD eval-nat-numeral*(*3*) *order.refl*)

```
      thus ?thesis by force
    qed
    have g2: ?G2 = p‘{1/2..?z}
    proof−
      have tl′ = take (n − 1) (drop 1 vts)
        using ab Suc.prems(1) Suc.prems(2) by simp
       moreover then have ?g2 = make-polygonal-path (take (n − 1) (drop 1
vts)) by blast
      ultimately have ?G2 = p‘{(2^1 − 1)/(2^1)..?z}
        using vts-sublist-path-image[of p vts ?g2 tl′ n−1 1 - - n ((2::real)^1 −
1)/(2^1) ?z]
          by (metis ∗ Suc.prems(1) Suc.prems(2) Suc.prems(4) Suc-eq-plus1
ab add-0 add-Suc-shift add-le-imp-le-diff diff-Suc-Suc diff-zero eval-nat-numeral(3)
length-Cons length-append less-Suc-eq-le list.size(3) order.refl)
      thus ?thesis by simp
    qed
    have 1/2 ≤ ?z
      using ∗ bounding-interval-helper1[of n−1] Suc.prems
        by (smt (verit) One-nat-def diff-Suc-Suc less-diff-conv numeral-3-eq-3
one-le-power plus-1-eq-Suc power-one-right power-strict-increasing-iff real-shrink-le
add-2-eq-Suc diff-add-inverse less-trans-Suc numeral-eq-Suc pos2 self-le-power zero-less-diff)
    moreover have ?z < 1 by auto
    ultimately have z: 1/2 ≤ ?z ∧ ?z < 1 by blast

    fix x
    assume x ∈ ?G1 ∩ ?G2
    then obtain t1 t2 where t1t2: t1 ∈ {0..1/2} ∧ t2 ∈ {1/2..?z} ∧ p t1 =
x ∧ p t2 = x
      by (smt (verit, del-insts) g1 g2 Int-iff imageE path-image-def)
    moreover have (t1 = t2) ∨ (t1 = 0 ∧ t2 = 1) ∨ (t1 = 1 ∧ t2 = 0)
    proof−
      have t1 ∈ {0..1} ∧ t2 ∈ {0..1}
        by (meson t1t2 z atLeastAtMost-iff dual-order.trans less-eq-real-def)
      thus ?thesis
        using Suc.prems(6) unfolding simple-path-def loop-free-def using t1t2
by presburger
    qed
    moreover have t1 = 1/2 using calculation by force
    ultimately have x = pathstart ?g2
   by (metis ab constant-linepath-is-not-loop-free dual-order.refl eq-divide-eq-numeral1(1)
ih joinpaths-def make-polygonal-path.simps(1) mult.commute p pathfinish-def pathfin-
ish-linepath polygon-pathstart simple-path-def zero-neq-numeral)
    thus x ∈ {pathstart ?g2} by simp
  qed
  ultimately show ?thesis using arc-join-eq ih by (metis arc-imp-simple-path)
 qed
 moreover have vts′ = a # tl′ using Suc.prems ab by argo
 moreover have p′ = (linepath a b) +++ make-polygonal-path tl′
 proof −
```

have *Suc* (*length tl'*) = *length vts'* **by** (*simp add*: *ab*)
then show *?thesis*
by (*metis* (*no-types*) ∗ *Cons-nth-drop-Suc Suc.prems*(*1*) *Suc.prems*(*2*) *Suc.prems*(*5*) *Suc-lessD ab drop-0 length-append-singleton make-polygonal-path.simps*(*4*) *not-less-eq numeral-3-eq-3*)
qed
ultimately have *?case* **by** *blast*
}
ultimately show *?case* **using** *Suc.prems* **by** *linarith*
qed

lemma *take-i-is-loop-free*:
fixes *vts* :: (*real^2*) *list*
assumes *n* = *length vts*
assumes *2* ≤ *i* ∧ *i* ≤ *n*
assumes *vts'* = *take i vts*
assumes *p* = *make-polygonal-path vts*
assumes *p'* = *make-polygonal-path vts'*
assumes *loop-free p*
shows *loop-free p'*
using *assms*
proof(*induct n*−*i arbitrary*: *vts' i p p'*)
case *0*
moreover then have *p* = *p'* **by** *auto*
ultimately show *?case* **by** *argo*
next
case (*Suc x*)

let *?i'* = *i*+*1*
let *?q-vts* = *take* (*i*+*1*) *vts*
let *?q* = *make-polygonal-path ?q-vts*

have *n*−*?i'* = *x* **using** *Suc.hyps*(*2*) **by** *linarith*
then have *loop-free ?q* **using** *Suc.hyps Suc.prems*(*2*) *Suc.prems*(*4*) *Suc.prems*(*6*) *assms*(*1*) **by** *auto*
moreover obtain *z* **where** *?q* = *make-polygonal-path* (*vts'* @ [*z*])
unfolding *Suc.prems*(*3*)
by (*metis Suc.hyps*(*2*) *Suc-eq-plus1 assms*(*1*) *take-Suc-conv-app-nth zero-less-Suc zero-less-diff*)
ultimately show *loop-free p'*
unfolding *Suc.prems* **using** *one-append-simple-path* **unfolding** *simple-path-def*
by (*metis One-nat-def Suc.prems*(*2*) *Suc-1 add-diff-cancel-right' append-take-drop-id assms*(*1*) *diff-diff-cancel length-append length-append-singleton length-drop make-polygonal-path-gives-path not-less-eq-eq numeral-3-eq-3*)
qed

lemma *sublist-is-loop-free*:
fixes *vts* :: (*real^2*) *list*
assumes *p* = *make-polygonal-path vts*

assumes $p' = make\text{-}polygonal\text{-}path\ vts'$
assumes *loop-free p*
assumes $m = length\ vts$
assumes $n = length\ vts'$
assumes *sublist vts' vts*
assumes $n \geq 2 \wedge m \geq 2$
shows *loop-free p'*
proof−
obtain *pre post* **where** *vts*: $vts = pre\ @\ vts'\ @\ post$ **using** *assms*(6) **unfolding** *sublist-def* **by** *blast*
then have $vts'\ @\ post = drop\ (length\ pre)\ vts$ **using** *vts* **by** *simp*
moreover have $vts' = take\ (length\ vts')\ (vts'\ @\ post)$ **using** *vts* **by** *simp*
moreover have *loop-free* (*make-polygonal-path* (*vts'* @ *post*))
using *drop-i-is-loop-free assms calculation*
by (*smt* (*verit*, *del-insts*) *One-nat-def Suc-1 Suc-leD diff-diff-cancel drop-all le-diff-iff' length-append length-drop list.size*(3) *nat-le-linear not-numeral-le-zero numeral-3-eq-3 trans-le-add1*)
ultimately show *?thesis*
using *take-i-is-loop-free assms*
by (*metis sublist-append-rightI sublist-length-le*)
qed

lemma *diff-points-path-image-set-property*:
fixes *a b*:: *real^2*
assumes $a \neq b$
shows *path-image* (*linepath a b*) $\neq \{a, b\}$
proof −
have *not-a*: (*linepath a b*) (*1/2*) $\neq a$
by (*smt* (*verit*) *add-diff-cancel-left' assms divide-eq-0-iff linepath-def scaleR-cancel-left scaleR-collapse*)
have *not-b*: (*linepath a b*) (*1/2*) $\neq b$
by (*smt* (*verit*, *ccfv-SIG*) *add-diff-cancel-right' assms divide-eq-1-iff linepath-def scaleR-cancel-left scaleR-collapse*)
have (*linepath a b*) (*1/2*) $\in$ *path-image* (*linepath a b*)
unfolding *path-image-def* **by** *simp*
then show *?thesis* **using** *not-a not-b* **by** *blast*
qed

lemma *polygonal-path-vertex-t*:
assumes $p = make\text{-}polygonal\text{-}path\ vts$
assumes $n = length\ vts$
assumes $n \geq 1$
assumes $0 \leq i \wedge i < n - 1$
assumes $x = (2\hat{}i - 1)/(2\hat{}i)$
shows $vts!i = p\ x$
using *assms*
proof(*induct i arbitrary*: *p vts n x*)
case *0*
then show *?case*

by (*metis bot-nat-0.extremum cancel-comm-monoid-add-class.diff-cancel diff-is-0-eq div-0 less-nat-zero-code list.size*(3) *pathstart-def polygon-pathstart power-0*)
next
case (*Suc i*)

let *?vts'* = *tl vts*
let *?p'* = *make-polygonal-path ?vts'*
let *?x'* = (2^*i* − 1)/(2^*i*)

have *p x* = *?p' ?x'*
proof−
let *?a* = *vts*!*0*
let *?b* = *vts*!*1*
let *?l* = *linepath ?a ?b*
have *n* ≥ *3* **using** *Suc.prems* **by** *linarith*
then have *length ?vts'* ≥ *2* **by** (*simp add*: *Suc.prems*(*2*))
then have *p* = *?l* +++ *?p'*
using *Suc.prems make-polygonal-path.simps*(*4*)[*of ?a ?b ?vts'*!*1 drop 2 ?vts'*]
by (*metis* (*no-types*, *opaque-lifting*) *Cons-nth-drop-Suc Suc-1 bot-nat-0.not-eq-extremum diff-Suc-1 diff-is-0-eq drop-0 drop-Suc less-Suc-eq zero-less-diff*)
moreover have *pathfinish ?l* = *pathstart ?p'*
by (*metis One-nat-def* ‹*2* ≤ *length* (*tl vts*)› *length-greater-0-conv nth-tl order-less-le-trans pathfinish-linepath polygon-pathstart pos2*)
moreover have (λ*x*::*real*. *2* ∗ *x* − *1*) *x* = *?x'*
using *Suc.prems*(*5*) *Suc-eq-plus1 bounding-interval-helper1 diff-Suc-1 le-add2 summation-helper*
by *presburger*
ultimately show *?thesis* **using** *joinpaths-tl-transform*[*of* λ*x*. *2*∗*x* − *1 ?l ?p' p x*]
by (*smt* (*verit*, *del-insts*) *divide-nonneg-nonneg half-bounded-equal two-realpow-ge-one*)
qed
moreover have *vts*!(*i*+*1*) = *?vts'*!*i* **using** *Suc.prems* **by** (*simp add*: *nth-tl*)
moreover have *?vts'*!*i* = *?p' ?x'* **using** *Suc.hyps Suc.prems* **by** *force*
ultimately show *?case* **by** *simp*
qed

lemma *loop-free-split-int*:
assumes *p* = *make-polygonal-path vts* ∧ *loop-free p*
assumes *vts1* = *take i vts*
assumes *vts2* = *drop* (*i*−*1*) *vts*
assumes *c1* = *make-polygonal-path vts1*
assumes *c2* = *make-polygonal-path vts2*
assumes *n* = *length vts*
assumes *1* ≤ *i* ∧ *i* < *n*
shows (*path-image c1*) ∩ (*path-image c2*) ⊆ {*pathstart c1*, *pathstart c2*}
(**is** *?C1* ∩ *?C2* ⊆ {*pathstart c1*, *pathstart c2*})
proof(*rule subsetI*)
let *?t* = ((*2*::*real*)^(*i*−*1*) − *1*)/(*2*^(*i*−*1*))

fix x
assume $x \in ?C1 \cap ?C2$
moreover have $c1c2$: $?C1 = p`\{0..?t\} \wedge ?C2 = p`\{?t..1\}$
using *vts-split-path-image assms polygon-of-def* **by** *metis*
ultimately obtain $t1\ t2$ **where** $t1t2$: $t1 \in \{0..?t\} \wedge t2 \in \{?t..1\} \wedge p\ t1 = x \wedge p\ t2 = x$ **by** *auto*
moreover have $t1 \in \{0..1\} \wedge t2 \in \{0..1\}$ **using** *calculation* **by** *force*
moreover have $(t1 = t2) \vee (t1 = 0 \wedge t2 = 1)$
using *assms*(1) *calculation* **unfolding** *polygon-of-def polygon-def simple-path-def loop-free-def*
by *fastforce*
ultimately have $x \in \{p\ ?t, p\ 0\}$ **by** *fastforce*
moreover have $p\ ?t = pathstart\ c2$
using *assms polygonal-path-vertex-t*
by (*smt* (*verit*, *ccfv-SIG*) *Cons-nth-drop-Suc diff-less-mono le-eq-less-or-eq length-drop less-imp-diff-less less-trans-Suc less-zeroE linorder-neqE-nat list.size*(3) *nth-Cons-0 numeral-1-eq-Suc-0 numerals*(1) *polygon-of-def polygon-pathstart*)
moreover have $p\ 0 = pathstart\ c1$ **using** *assms*
by (*metis One-nat-def diff-is-0-eq diff-zero linorder-not-less nth-take path-start-def polygon-pathstart take-eq-Nil zero-less-Suc*)
ultimately show $x \in \{pathstart\ c1, pathstart\ c2\}$ **by** *blast*
qed

lemma *loop-free-arc-split-int*:
assumes $p = make\text{-}polygonal\text{-}path\ vts \wedge loop\text{-}free\ p \wedge arc\ p$
assumes $vts1 = take\ i\ vts$
assumes $vts2 = drop\ (i-1)\ vts$
assumes $c1 = make\text{-}polygonal\text{-}path\ vts1$
assumes $c2 = make\text{-}polygonal\text{-}path\ vts2$
assumes $n = length\ vts$
assumes $1 \leq i \wedge i < n$
shows $(path\text{-}image\ c1) \cap (path\text{-}image\ c2) \subseteq \{pathstart\ c2\}$
(**is** $?C1 \cap ?C2 \subseteq \{pathstart\ c2\}$)
proof(*rule subsetI*)
let $?t = ((2::real)\hat{}(i-1) - 1)/(2\hat{}(i-1))$

fix x
assume $x \in ?C1 \cap ?C2$
moreover have $c1c2$: $?C1 = p`\{0..?t\} \wedge ?C2 = p`\{?t..1\}$
using *vts-split-path-image assms polygon-of-def* **by** *metis*
ultimately obtain $t1\ t2$ **where** $t1t2$: $t1 \in \{0..?t\} \wedge t2 \in \{?t..1\} \wedge p\ t1 = x \wedge p\ t2 = x$ **by** *auto*
moreover have $t1 \in \{0..1\} \wedge t2 \in \{0..1\}$ **using** *calculation* **by** *force*
moreover have $(t1 = t2) \vee (t1 = 0 \wedge t2 = 1)$
using *assms*(1) *calculation* **unfolding** *polygon-of-def polygon-def simple-path-def loop-free-def*
by *fastforce*
moreover then have $t1 = t2$
using *assms*(1) **unfolding** *arc-def* **using** *calculation*(1) *inj-on-contraD* **by**

fastforce
ultimately have $x \in \{p\ ?t\}$ **by** *fastforce*
moreover have $p\ ?t = pathstart\ c2$
using *assms polygonal-path-vertex-t*
by (*smt* (*verit*, *ccfv-SIG*) *Cons-nth-drop-Suc diff-less-mono le-eq-less-or-eq length-drop less-imp-diff-less less-trans-Suc less-zeroE linorder-neqE-nat list.size*(*3*) *nth-Cons-0 numeral-1-eq-Suc-0 numerals*(*1*) *polygon-of-def polygon-pathstart*)
ultimately show $x \in \{pathstart\ c2\}$ **by** *fast*
qed

lemma *loop-free-append*:
assumes $p = make\text{-}polygonal\text{-}path\ vts$
assumes $p1 = make\text{-}polygonal\text{-}path\ vts1$
assumes $p2 = make\text{-}polygonal\text{-}path\ vts2$
assumes $vts = vts1\ @\ (tl\ vts2)$
assumes $loop\text{-}free\ p1 \wedge loop\text{-}free\ p2$
assumes $path\text{-}image\ p1 \cap path\text{-}image\ p2 \subseteq \{pathstart\ p1, pathstart\ p2\}$
assumes $last\ vts2 \neq hd\ vts1 \longrightarrow path\text{-}image\ p1 \cap path\text{-}image\ p2 \subseteq \{pathstart\ p2\}$
assumes $last\ vts1 = hd\ vts2$
assumes $arc\ p1 \wedge arc\ p2$
shows *loop-free p*
using *assms*
proof(*induct length vts1 arbitrary*: *p p1 p2 vts vts1 vts2 rule*: *less-induct*)
case *less*
have *1*: $length\ vts1 \geq 2$
using *less*
by (*metis Suc-1 arc-distinct-ends constant-linepath-is-not-loop-free diff-is-0-eq' make-polygonal-path.simps*(*1*) *not-less-eq-eq polygon-pathfinish polygon-pathstart*)
moreover have $length\ vts2 \geq 2$
using *less.prems*
by (*metis One-nat-def Suc-1 Suc-leI arc-distinct-ends diff-Suc-1 length-greater-0-conv make-polygonal-path.simps*(*1*) *nat-less-le pathfinish-linepath pathstart-linepath polygon-pathfinish polygon-pathstart*)
ultimately have $length\ vts \geq 3$ **using** *less assms*(*4*) **by** *auto*
{ assume $*$: $length\ vts1 = 2$
then obtain *a b* **where** $vts1 = [a, b]$
by (*metis 1 Cons-nth-drop-Suc One-nat-def Suc-1 drop0 drop-eq-Nil lessI pos2*)
then have *p1*: $p1 = linepath\ a\ b$
using *less make-polygonal-path.simps*(*3*) **by** *metis*
have *p*: $p = p1\ +\!+\!+\ p2$
using *p1 less*
by (*smt* (*verit*) ‹$vts1 = [a, b]$› *append-Cons assms*(*4*) *constant-linepath-is-not-loop-free last-ConsL last-ConsR list.exhaust-sel list.inject list.simps*(*3*) *make-polygonal-path.elims self-append-conv2*)
have *b*: $pathstart\ p2 \in path\text{-}image\ p1 \cap path\text{-}image\ p2$
by (*metis IntI less*(*3*,*4*,*6*,*9*) *constant-linepath-is-not-loop-free hd-conv-nth last-conv-nth make-polygonal-path.simps*(*1*) *pathfinish-in-path-image pathstart-in-path-image polygon-pathfinish polygon-pathstart*)

{ assume *pathstart p1 = pathfinish p2*
then have *?case* **using** *simple-path-join-loop-eq*[*of p2 p1*] *less.prems*
by (*metis make-polygonal-path-gives-path p path-join-eq simple-path-def*)
} moreover
{ assume ∗∗: *pathstart p1 ≠ pathfinish p2*
then have *path-image p1 ∩ path-image p2 = {pathstart p2}*
using *less.prems b*
by (*metis constant-linepath-is-not-loop-free empty-subsetI hd-conv-nth insert-subset last-conv-nth make-polygonal-path.simps*(*1*) *polygon-pathfinish polygon-pathstart subset-antisym*)
then have *?case*
using *arc-join-eq*[*of p1 p2*]
by (*metis less*(*2*,*4*,*10*) *arc-imp-simple-path arc-join-eq-alt make-polygonal-path-gives-path p path-join-path-ends simple-path-def*)
}
ultimately have *?case* **by** *blast*
} moreover
{ assume ∗: *length vts1 > 2*
then have *len-p1*: *length vts1 ≥ 3* **by** *linarith*
then obtain *a b vts-tl* **where** *ab*: *vts = a # vts-tl ∧ b = hd vts-tl*
by (*metis ‹3 ≤ length vts› length-0-conv list.collapse not-numeral-le-zero*)
have *vts1-char*: *vts1 = (vts1 ! 0) # (vts1 ! 1) # (vts1 ! 2) # (drop 3 vts1)*
using *len-p1*
by (*metis 1 Cons-nth-drop-Suc One-nat-def Suc-1 drop0 length-greater-0-conv linorder-not-less list.size*(*3*) *not-less-eq-eq not-numeral-le-zero numeral-3-eq-3*)
then have *tail-vts1-char*: *tl vts1 = (vts1 ! 1) # (vts1 ! 2) # (drop 3 vts1)*
by (*metis list.sel*(*3*))

let *?l = linepath a b*
let *?vts1-tl = tl vts1*
let *?p1-tl = make-polygonal-path ?vts1-tl*
let *?vts2-tl = tl vts2*
let *?p2-tl = make-polygonal-path ?vts2-tl*
let *?p-tl = make-polygonal-path vts-tl*

have *p*: *p = ?l +++ ?p-tl*
unfolding *less.prems*(*1*)
by (*smt* (*verit*, *ccfv-SIG*) *Suc-le-length-iff ‹3 ≤ length vts› ab list.discI list.sel*(*1*) *list.sel*(*3*) *make-polygonal-path.elims numeral-3-eq-3*)
have *p1*: *p1 = ?l +++ ?p1-tl*
using *ab* **unfolding** *less.prems*(*2*)
by (*smt* (*verit*, *ccfv-SIG*) ∗ *Nitpick.size-list-simp*(*2*) *One-nat-def Suc-1 Suc-le-eq hd-append2 less.prems*(*4*) *list.sel*(*1*) *list.sel*(*3*) *make-polygonal-path.elims nat-less-le tl-append2*)

have *p1-img*: *path-image ?l ∩ path-image ?p1-tl = {pathstart ?p1-tl}*
by (*metis arc-join-eq-alt less.prems*(*2*) *less.prems*(*9*) *make-polygonal-path-gives-path p1 path-join-path-ends*)

have *vts-tl = ?vts1-tl @ (tl vts2)*
using *less.prems(4) ab*
by (*metis ∗ length-greater-0-conv list.sel(3) order.strict-trans pos2 tl-append2*)
moreover have *loop-free ?p1-tl ∧ loop-free p2*
using ‹*3 ≤ length vts1*› *less.prems(2) less.prems(5) sublist-is-loop-free* **by** *fastforce*
moreover have *path-image ?p1-tl ∩ path-image p2 ⊆ {pathstart p2}*
proof −
have *path-image ?p1-tl ⊆ path-image p1*
by (*metis* (*no-types*, *opaque-lifting*) *∗ Suc-1 Suc-lessD length-tl less.prems(2) list.collapse list.size(3) order.refl path-image-cons-union sup.bounded-iff zero-less-diff zero-order(3)*)
then have *path-image ?p1-tl ∩ path-image p2 ⊆ {pathstart p1*, *pathstart p2}*
using *less* **by** *blast*
moreover have *pathstart p1 ∉ path-image ?p1-tl*
proof(*rule ccontr*)
assume ¬ *pathstart p1 ∉ path-image ?p1-tl*
then have *pathstart p1 ∈ path-image ?p1-tl* **by** *blast*
thus *False*
by (*metis* (*no-types*, *lifting*) *IntI arc-def arc-simple-path less(10) make-polygonal-path-gives-path p1 p1-img path-join-path-ends pathstart-in-path-image pathstart-join simple-path-joinE singletonD*)
qed
ultimately have *path-image ?p1-tl ∩ path-image p2 ⊆ {pathstart p2}* **by** *blast*
thus *?thesis* **by** *blast*
qed
moreover then have *last vts2 ≠ hd ?vts1-tl*
⟶ path-image ?p1-tl ∩ path-image p2 ⊆ {pathstart p2} **by** *blast*
moreover have *last ?vts1-tl = hd vts2*
by (*metis ∗ Suc-1 drop-Nil drop-Suc-Cons last-drop last-tl less.prems(8) list.collapse*)
moreover have *arc ?p1-tl ∧ arc p2*
by (*smt* (*verit*, *best*) *∗ Nitpick.size-list-simp(2) Suc-1 arc-imp-simple-path constant-linepath-is-not-loop-free diff-Suc-Suc diff-is-0-eq leD length-greater-0-conv length-tl less.prems(2) less.prems(5) less.prems(9) list.sel(3) make-polygonal-path.elims make-polygonal-path-gives-path order.strict-trans path-join-path-ends pos2 simple-path-joinE*)
ultimately have *ih1*: *loop-free ?p-tl*
using *less.hyps*[*of ?vts1-tl ?p-tl vts-tl ?p1-tl p2 vts2*] *∗ less.prems(3)* **by** *fastforce*

have *p-tl-img*: *path-image ?p-tl = path-image ?p1-tl ∪ path-image p2*
by (*metis* (*no-types*, *lifting*) *∗ Suc-1 Suc-le-eq* ‹*2 ≤ length vts2*› ‹*last* (*tl vts1*) *= hd vts2*› ‹*vts-tl = tl vts1 @ tl vts2*› *hd-conv-nth last-conv-nth length-greater-0-conv length-tl less.prems(3) less-diff-conv make-polygonal-path-image-append-alt order-less-le-trans path-image-join plus-1-eq-Suc polygon-pathfinish polygon-pathstart pos2*)

have *1*: *length* [*a*, *b*] *< length vts1* **using** ‹*3 ≤ length vts1*› **by** *fastforce*
moreover have *2*: *p = make-polygonal-path vts* **using** *less.prems(1)* **by** *auto*

moreover have *3*: *?l = make-polygonal-path [a, b]* **by** *simp*
moreover have *4*: *?p-tl = make-polygonal-path vts-tl* **using** *less* **by** *simp*
moreover have *5*: *vts = [a, b] @ tl vts-tl*
using *ab ‹3 ≤ length vts› append-eq-Cons-conv* **by** *fastforce*
moreover have *6*: *loop-free ?l ∧ loop-free ?p-tl*
proof −
have *sublist [a, b] vts1*
by (*metis* (*no-types*, *opaque-lifting*) *1 Cons-nth-drop-Suc Suc-lessD ab append-Cons drop0 length-Cons less.prems*(*4*) *list.sel*(*1*) *list.sel*(*3*) *list.size*(*3*) *sublist-take take0 take-Suc-Cons*)
then have *loop-free* (*make-polygonal-path* [*a*, *b*])
using *sublist-is-loop-free ∗ less.prems*(*2*) *less.prems*(*5*) **by** *fastforce*
then have *loop-free ?l* **using** *make-polygonal-path.simps*(*3*) **by** *simp*
thus *?thesis* **using** *ih1* **by** *simp*
qed
moreover have *9*: *last [a, b] = hd vts-tl* **by** (*simp add*: *ab*)
moreover have *10*: *arc ?l ∧ arc ?p-tl*
proof −
have *pathstart ?p-tl = b*
by (*metis 6 ab constant-linepath-is-not-loop-free hd-conv-nth make-polygonal-path.simps*(*1*) *polygon-pathstart*)
moreover have *pathfinish ?p-tl ≠ b*
proof(*rule ccontr*)
assume ¬ *pathfinish ?p-tl ≠ b*
have *pathfinish ?p-tl = pathfinish p2*
by (*smt* (*verit*) *5 9 Nil-tl ‹2 ≤ length vts2› ‹¬ pathfinish* (*make-polygonal-path vts-tl*) *≠ b› ab arc-distinct-ends last-append last-conv-nth last-tl length-tl less.prems*(*3*) *less.prems*(*4*) *less.prems*(*9*) *list.size*(*3*) *not-numeral-le-zero polygon-pathfinish polygon-pathstart*)
moreover have *b ∈ path-image p1*
by (*metis list.size*(*3*)*1 Cons-nth-drop-Suc Suc-lessD UnCI ab append-eq-conv-conj drop0 hd-append2 hd-conv-nth length-Cons less.prems*(*2*) *less.prems*(*4*) *list.distinct*(*1*) *list.sel*(*3*) *path-image-cons-union pathstart-in-path-image polygon-pathstart tl-append2*)
moreover have *b ≠ pathstart p1*
by (*metis* (*no-types*, *lifting*) *1 6 ab constant-linepath-is-not-loop-free dual-order.strict-trans hd-append2 hd-conv-nth length-greater-0-conv less.prems*(*2*) *less.prems*(*4*) *list.sel*(*1*) *list.size*(*3*) *polygon-pathstart*)
moreover have *b ≠ pathfinish p2*
by (*metis* (*no-types*, *lifting*) *Int-insert-right-if1 arc-distinct-ends calculation*(*2*) *calculation*(*3*) *insert-absorb insert-iff insert-not-empty less.prems*(*6*) *less.prems*(*9*) *pathfinish-in-path-image subset-iff*)
ultimately show *False*
using *‹¬ pathfinish* (*make-polygonal-path vts-tl*) *≠ b›* **by** *fastforce*
qed
ultimately have *pathstart ?p-tl ≠ pathfinish ?p-tl* **by** *simp*
then have *arc ?p-tl*
using *ih1 arc-def loop-free-cases make-polygonal-path-gives-path* **by** *metis*
moreover have *arc ?l* **by** (*metis 6 arc-linepath constant-linepath-is-not-loop-free*)
ultimately show *?thesis* **by** *blast*

qed
moreover have *7*: *path-image ?l ∩ path-image ?p-tl ⊆ {pathstart ?l, pathstart ?p-tl}*
proof −
have *path-image ?l ⊆ path-image p1*
by (*metis Un-iff ‹loop-free (make-polygonal-path (tl vts1)) ∧ loop-free p2› ‹vts-tl = tl vts1 @ tl vts2› ab constant-linepath-is-not-loop-free hd-append2 hd-conv-nth make-polygonal-path.simps(1) p1 path-image-join pathfinish-linepath polygon-pathstart subsetI*)
then have *path-image ?l ∩ path-image p2 ⊆ {pathstart p1, pathstart p2}*
using *less.prems(6)* **by** *auto*
moreover have *pathstart p2 ∉ path-image ?l*
by (*smt (verit, ccfv-threshold) 10 Int-insert-left-if1 ‹arc (make-polygonal-path (tl vts1)) ∧ arc p2› ‹last (tl vts1) = hd vts2› ‹loop-free (make-polygonal-path (tl vts1)) ∧ loop-free p2› arc-def arc-distinct-ends arc-join-eq-alt constant-linepath-is-not-loop-free hd-conv-nth insert-absorb last-conv-nth less.prems(3) less.prems(9) make-polygonal-path.simps(1) p1 path-join-eq pathfinish-in-path-image polygon-pathfinish polygon-pathstart singleton-insert-inj-eq′*)
ultimately have *path-image ?l ∩ path-image ?p-tl ⊆ {pathstart p1, pathstart ?p1-tl}*
using *p1-img p-tl-img* **by** *blast*
moreover have *pathstart ?p1-tl = pathstart ?p-tl*
by (*metis 2 less.prems(2) make-polygonal-path-gives-path p p1 path-join-path-ends*)
moreover have *pathstart p1 = pathstart ?l* **by** (*simp add*: *p1*)
ultimately show *?thesis* **by** *argo*
qed
moreover have *8*: *last vts-tl ≠ hd [a, b]*
⟶ path-image ?l ∩ path-image ?p-tl ⊆ {pathstart ?p-tl}
proof *clarify*
fix *x*
assume *a1*: *last vts-tl ≠ hd [a, b]*
assume *a2*: *x ∈ path-image ?l*
assume *a3*: *x ∈ path-image ?p-tl*

have *hd vts1 ≠ last vts2*
using *less.prems*
by (*metis a1 ‹vts-tl = tl vts1 @ tl vts2› ab arc-distinct-ends constant-linepath-is-not-loop-free hd-append2 last-appendR last-tl length-tl list.sel(1) list.size(3) make-polygonal-path.simps(1) polygon-pathfinish polygon-pathstart*)
then have *p1-p2-int*: *path-image p1 ∩ path-image p2 ⊆ {pathstart p2}*
using *less.prems* **by** *argo*

have *x ≠ pathstart ?l*
proof(*rule ccontr*)
assume ∗∗: *¬ x ≠ pathstart ?l*
have *pathstart ?l ∉ path-image ?p1-tl*
by (*metis Int-iff arc-distinct-ends arc-join-eq-alt empty-iff insertE less.prems(2) less.prems(9) make-polygonal-path-gives-path p1 path-join-path-ends pathstart-in-path-image*)
then have *pathstart ?l ∈ path-image p2* **using** *p1-img p-tl-img ∗∗ a3* **by**

blast
then have *pathstart ?l* ∈ *path-image p1* ∩ *path-image p2*
by (*metis IntI p1 pathstart-in-path-image pathstart-join*)
moreover have *pathstart ?l* ≠ *pathstart p2*
by (*metis arc-distinct-ends constant-linepath-is-not-loop-free hd-conv-nth last-conv-nth less.prems*(2) *less.prems*(3) *less.prems*(5) *less.prems*(8) *less.prems*(9) *make-polygonal-path.simps*(1) *p1 pathstart-join polygon-pathfinish polygon-pathstart*)
ultimately show *False* **using** *p1-p2-int* **by** *blast*
qed
moreover have *x* = *pathstart ?l* ∨ *x* = *pathstart ?p-tl* **using** *7 a2 a3* **by** *blast*
ultimately show *x* = *pathstart ?p-tl* **by** *fast*
qed
ultimately have *?case* **using** *less.hyps*[*of* [*a*, *b*] *p vts ?l ?p-tl vts-tl*] **by** *blast*
}
ultimately show *?case* **using** *less 1* **by** *linarith*
qed

lemma *sublist-path-image-subset*:
assumes *sublist vts1 vts2*
assumes *length vts1* ≥ *1*
shows *path-image* (*make-polygonal-path vts1*) ⊆ *path-image* (*make-polygonal-path vts2*)
proof−
let *?p1* = *make-polygonal-path vts1*
let *?p2* = *make-polygonal-path vts2*
let *?m* = *length vts1*
let *?n* = *length vts2*
have *n-geq-m*: *?n* ≥ *?m* **by** (*simp add*: *assms*(1) *sublist-length-le*)

have *?thesis* **if** ∗: *length vts1* = *1*
proof−
have *path-image ?p1* = {*vts1*!*0*}
by (*metis Cons-nth-drop-Suc One-nat-def closed-segment-idem drop0 drop-eq-Nil le-numeral-extra*(4) *make-polygonal-path.simps*(2) *path-image-linepath that zero-less-one*)
moreover have *vts1*!*0* ∈ *set vts2*
by (*metis assms*(1) *less-numeral-extra*(1) *nth-mem set-mono-sublist subsetD that*)
ultimately show *?thesis*
using *vertices-on-path-image* **by** *force*
qed
moreover have *?thesis* **if** ∗: *length vts1* ≥ *2*
proof−
obtain *pre post* **where** *sublist*: *vts2* = *pre* @ *vts1* @ *post*
using *assms*(1) **unfolding** *sublist-def* **by** *blast*
let *?i* = *length pre*
let *?j* = *length vts1*
let *?k* = *?i* + *?j*
let *?x1* = (*2*^*?i* − *1*)/*2*^(*?i*)::*real*

```
    let ?x2 = (2^(?k−1) − 1)/(2^(?k−1))::real
    let ?x = (2 ^ (?i − 1) − 1) / 2 ^ (?i − 1)::real
    have path-image ?p1 = ?p2 ‘ {?x1..?x2} if **: length post ≥ 1
      using sublist * ** vts-sublist-path-image[of ?p2 vts2 ?p1 vts1 ?j ?i ?n ?m ?k
?x1 ?x2]
      by fastforce
    moreover have path-image ?p1 = ?p2 ‘ {?x1..1} if **: length post = 0
    proof−
      have sublist: vts2 = pre @ vts1 using ** sublist by blast
      moreover have vts1 = drop ?i vts2 using sublist * by simp
      moreover have 1 ≤ ?i + 1 ∧ ?i + 1 < length vts2 using sublist * ** by
simp
      ultimately show ?thesis
      using vts-split-path-image[of ?p2 vts2 - - ?p1 vts1 ?i + 1 ?n ?x1] add-diff-cancel-right′
        by metis
    qed
     moreover have ?p2 ‘ {?x1..?x2} ⊆ path-image ?p2 ∧ ?p2 ‘ {?x1..1} ⊆
path-image ?p2
    proof−
      have {?x1..?x2} ⊆ {0..1} ∧ {?x1..1} ⊆ {0..1} by simp
      thus ?thesis unfolding path-image-def by blast
    qed
    ultimately show ?thesis by (metis less-one linorder-not-le)
  qed
  ultimately show ?thesis using assms by linarith
qed

lemma integral-on-edge-subset-integral-on-path:
  assumes p = make-polygonal-path vts and
          (i::int) ∈ {0..<((length vts) − 1)} and
          x = vts!i and
          y = vts!(i+1)
  shows {v. integral-vec v ∧ v ∈ path-image (linepath x y)}
          ⊆ {v. integral-vec v ∧ v ∈ path-image p}
  using assms edge-subset-path-image by blast

lemma sublist-pair-integral-subset-integral-on-path:
  assumes p = make-polygonal-path vts and
          sublist [x, y] vts
  shows {v. integral-vec v ∧ v ∈ path-image (linepath x y)}
          ⊆ {v. integral-vec v ∧ v ∈ path-image p}
  using assms integral-on-edge-subset-integral-on-path
proof−
  obtain pre post where vts: vts = pre @ [x, y] @ post using assms(2) sublist-def
by blast
  let ?i = length pre
  have x = vts!?i using vts by simp
  moreover have y = vts!(?i + 1)
    by (metis vts add.right-neutral append-Cons nth-Cons-Suc nth-append-length
```

nth-append-length-plus plus-1-eq-Suc)
moreover have *?i* ∈ {*0*..<((*length vts*) − *1*)} **using** *vts* **by** *force*
ultimately show *?thesis* **using** *assms*(*1*) *integral-on-edge-subset-integral-on-path*
by *auto*
qed

lemma *sublist-integral-subset-integral-on-path*:
assumes *length ell* ≥ *2*
assumes *p* = *make-polygonal-path vts* **and**
sublist ell vts
shows {*v*. *integral-vec v* ∧ *v* ∈ *path-image* (*make-polygonal-path ell*)}
⊆ {*v*. *integral-vec v* ∧ *v* ∈ *path-image p*}
proof−
obtain *pre post* **where** *vts*: *vts* = *pre* @ *ell* @ *post* **using** *assms*(*3*) *sublist-def*
by *blast*
then have *len-vts*: *length vts* ≥ *2*
using *assms*(*1*)
by *auto*
let *?i* = *length pre*
have *v* ∈ *path-image p* **if** ∗: *v* ∈ *path-image* (*make-polygonal-path ell*) **for** *v*
proof −
have ∃*j*::*nat*. *v* ∈ *path-image* (*linepath* (*ell* ! *j*) (*ell* ! (*j*+*1*))) ∧ *j*+*1* < *length*
ell
using ∗ *polygonal-path-image-linepath-union assms*(*1*)
by (*meson less-diff-conv make-polygonal-path-image-property*)
then obtain *j* **where** *v-in*: *v* ∈ *path-image* (*linepath* (*ell* ! *j*) (*ell* ! (*j*+*1*)))
j+*1* < *length ell*
by *auto*
then have *ell-at*: *ell* ! *j* = *vts* ! (*j* + *length pre*) ∧ *ell* ! (*j*+*1*) = *vts* ! (*j* + *1*
+ *length pre*)
using *vts*
by (*simp add*: *nth-append*)
then have *v-in2*: *v* ∈ *path-image* (*linepath* (*vts* ! (*j* + *length pre*)) (*vts* ! (*j* +
length pre + *1*)))
using *v-in*(*1*) **by** *simp*
have *j* + *1* + *length pre* < *length vts*
using *ell-at v-in*(*2*) *vts* **by** *auto*
then have *j-plus*: *j* + *length pre* < *length vts* − *1*
by *auto*
then show *?thesis* **using** *v-in2 linepaths-subset-make-polygonal-path-image*[*OF*
len-vts j-plus] *assms*(*1*)
assms(*2*) **by** *auto*
qed
then show *?thesis* **by** *blast*
qed

13 Reversing Polygonal Path Vertex List

lemma *rev-vts-path-image*:

shows *path-image* (*make-polygonal-path* (*rev vts*)) = *path-image* (*make-polygonal-path vts*)
proof −
{ assume *length vts* ≤ *1*
then have *?thesis*
by (*smt* (*verit*, *best*) *One-nat-def Suc-length-conv le-SucE le-zero-eq length-0-conv rev.simps*(*1*) *rev-singleton-conv*)
} moreover
{ fix *x*
assume ∗: *x* ∈ *path-image* (*make-polygonal-path* (*rev vts*)) ∧ *length vts* ≥ *2*
then obtain *k* **where** *k-prop*: *k*<*length* (*rev vts*) − *1* ∧ *x* ∈ *path-image* (*linepath* (*rev vts* ! *k*) (*rev vts* ! (*k* + *1*)))
using *make-polygonal-path-image-property*[*of rev vts*] **by** *auto*
have *p1*: *rev vts* ! *k* = *vts* ! (*length vts* − *k* − *1*)
using *rev-nth*
by (*metis Suc-lessD* ‹*k* < *length* (*rev vts*) − *1* ∧ *x* ∈ *path-image* (*linepath* (*rev vts* ! *k*) (*rev vts* ! (*k* + *1*)))› *add.commute diff-diff-left length-rev less-diff-conv plus-1-eq-Suc*)
have *p2*: *rev vts* ! (*k* + *1*) = *vts* ! (*length vts* − *k* − *2*)
using *rev-nth*[*of k+1 vts*] *k-prop*
by *force*
then have *x* ∈ *path-image* (*linepath* (*vts* ! (*length vts* − *k* − *1*)) (*vts* ! (*length vts* − *k* − *2*)))
using *k-prop p1 p2* **by** *auto*
then have *x* ∈ *path-image* (*linepath* (*vts* ! (*length vts* − *k* − *2*)) (*vts* ! (*length vts* − *k* − *1*)))
using *reversepath-linepath path-image-reversepath*
by *metis*
then have *x* ∈ *path-image* (*make-polygonal-path vts*)
using *linepaths-subset-make-polygonal-path-image* ∗ *k-prop*
by (*smt* (*verit*, *best*) *Nat.diff-add-assoc add.commute add-diff-cancel-left′ diff-le-self length-rev less-Suc-eq less-diff-conv linorder-not-less nat-1-add-1 nat-neq-iff plus-1-eq-Suc subsetD*)
} moreover
{ fix *x*
assume ∗: *x* ∈ *path-image* (*make-polygonal-path vts*) ∧ *length vts* ≥ *2*
then obtain *k* **where** *k-prop*: *k*<*length vts* − *1* ∧ *x* ∈ *path-image* (*linepath* (*vts* ! *k*) (*vts* ! (*k* + *1*)))
using *make-polygonal-path-image-property*[*of vts*] **by** *auto*
have *p1*: *vts* ! *k* = (*rev vts*) ! (*length vts* − *k* − *1*)
using *rev-nth k-prop*
by (*metis Suc-eq-plus1 Suc-lessD diff-diff-left length-rev less-diff-conv rev-rev-ident*)
have *p2*: *vts* ! (*k* + *1*) = (*rev vts*) ! (*length vts* − *k* − *2*)
using *rev-nth*[*of k+1*]
by (*smt* (*verit*) *Suc-eq-plus1 add-2-eq-Suc′ diff-diff-left k-prop length-rev less-diff-conv rev-rev-ident*)
then have *x* ∈ *path-image* (*linepath* (*rev vts* ! (*length vts* − *k* − *2*)) (*rev vts* ! (*length vts* − *k* − *1*)))
using *reversepath-linepath path-image-reversepath*

by (*metis k-prop p1*)
then have $x \in$ *path-image* (*make-polygonal-path* (*rev vts*))
using *linepaths-subset-make-polygonal-path-image k-prop* $*$
by (*smt* (*verit*, *best*) *Suc-1 Suc-diff-Suc Suc-eq-plus1 Suc-le-eq Suc-lessD bot-nat-0.not-eq-extremum diff-commute diff-diff-left diff-less length-rev less-numeral-extra*(*1*) *subsetD zero-less-diff*)
}
ultimately show *?thesis* **by** *force*
qed

lemma *rev-vts-is-loop-free*:
assumes *p* = *make-polygonal-path vts*
assumes *loop-free p*
shows *loop-free* (*make-polygonal-path* (*rev vts*))
using *assms*
proof(*induct length vts arbitrary*: *p vts*)
case *0*
then show *?case* **by** *simp*
next
case (*Suc n*)
then have *Suc n* $\geq$ *2*
by (*metis One-nat-def Suc-length-conv constant-linepath-is-not-loop-free le-SucE le-add1 le-numeral-Suc length-greater-0-conv list.size*(*3*) *make-polygonal-path.simps*(*2*) *numeral-One plus-1-eq-Suc pred-numeral-simps*(*2*) *semiring-norm*(*26*))
moreover
{ assume $*$: *Suc n* = *2*
then obtain *a b* **where** *ab*: *p* = *linepath a b*
using *Suc.prems make-polygonal-path.simps*(*3*)
by (*metis* (*no-types*, *opaque-lifting*) *Cons-nth-drop-Suc One-nat-def Suc.hyps*(*2*) *Suc-1 diff-Suc-1 drop-0 drop-Suc length-0-conv length-tl zero-less-Suc*)
moreover then have *a* $\neq$ *b* **using** *Suc.prems*(*2*) *constant-linepath-is-not-loop-free* **by** *blast*
ultimately have *loop-free* (*linepath b a*) **by** (*simp add*: *linepath-loop-free*)
moreover have *make-polygonal-path* (*rev vts*) = *linepath b a*
by (*smt* (*z3*) $*$ *Cons-nth-drop-Suc One-nat-def Suc.hyps*(*2*) *Suc.prems*(*1*) *Suc-1 Suc-diff-Suc ab butlast-snoc diff-Suc-1 drop0 hd-conv-nth hd-rev last-conv-nth length-butlast length-rev lessI linepath-1' make-polygonal-path.simps*(*3*) *nth-append-length pathstart-def pathstart-linepath pos2 rev.simps*(*2*) *rev-is-Nil-conv rev-take take-eq-Nil*)
ultimately have *?case* **by** *simp*
} moreover
{ assume $*$: *Suc n* > *2*
let *?vts'* = *butlast vts*
let *?p'* = *make-polygonal-path ?vts'*
let *?vts'-rev* = *rev ?vts'*
let *?p'-rev* = *make-polygonal-path ?vts'-rev*

let *?vts-rev* = *rev vts*
let *?p-rev* = *make-polygonal-path ?vts-rev*

obtain *y z* **where** *yz*: *y* = *last ?vts′* ∧ *z* = *last vts* **by** *blast*
let *?l* = *linepath y z*
let *?l-rev* = *linepath z y*
have *loop-free ?p′*
by (*metis* ∗ *Suc.hyps*(2) *Suc.prems*(1) *Suc.prems*(2) *butlast-conv-take diff-Suc-1 le-add2 less-Suc-eq-le plus-1-eq-Suc take-i-is-loop-free*)
then have *loop-free-p′-rev*: *loop-free ?p′-rev* **using** *Suc.hyps* **by** *force*
moreover have *rev vts* = *z* # *?vts′-rev*
by (*metis Suc.hyps*(2) *yz append-butlast-last-id length-0-conv nat.distinct*(1) *rev-eq-Cons-iff rev-rev-ident*)
moreover have *y* = *hd ?vts′-rev* **using** *yz* **by** (*simp add*: *hd-rev*)
ultimately have *p-rev*: *?p-rev* = *?l-rev* +++ *?p′-rev*
by (*smt* (*verit*, *best*) *constant-linepath-is-not-loop-free list.sel*(1) *make-polygonal-path.elims make-polygonal-path.simps*(4))

have [*y*, *z*] = *drop* (*n*−1) *vts*
using *yz Suc.hyps*(2)
by (*metis* (*no-types*, *opaque-lifting*) ∗ *Cons-nth-drop-Suc Suc-1 Suc-diff-Suc Suc-lessD Suc-n-not-le-n append-butlast-last-id append-eq-conv-conj diff-Suc-1 last-conv-nth length-0-conv length-butlast less-nat-zero-code linorder-not-le nth-take*)
then have *?l* = *make-polygonal-path* (*drop* (*n*−1) *vts*)
using *make-polygonal-path.simps* **by** *metis*
moreover have *?p′* = *make-polygonal-path* (*take n vts*)
using *Suc.hyps*(2) **by** (*metis butlast-conv-take diff-Suc-1*)
ultimately have *path-image ?l* ∩ *path-image ?p′* ⊆ {*pathstart ?l*, *pathstart ?p′*}
using *loop-free-split-int*
by (*smt* (*verit*, *ccfv-SIG*) *Int-commute Suc.hyps*(2) *Suc.prems*(1) *Suc.prems*(2) *Suc-1 Suc-le-mono* ‹2 ≤ *Suc n*› *insert-commute lessI*)
moreover have *path-image ?l* = *path-image ?l-rev* **by** *auto*
moreover have *path-image ?p′* = *path-image ?p′-rev*
using ∗ *Suc.hyps*(2) *rev-vts-path-image* **by** *force*
moreover have *pathstart ?l* = *pathfinish ?l-rev* **by** *simp*
moreover have *pathstart ?p′* = *pathfinish ?p′-rev*
by (*metis Nil-is-rev-conv last.simps last-conv-nth last-rev list.distinct*(1) *list.exhaust-sel make-polygonal-path.simps*(1) *make-polygonal-path.simps*(2) *nth-Cons-0 polygon-pathfinish polygon-pathstart*)
ultimately have *path-image-int*:
path-image ?l-rev ∩ *path-image ?p′-rev* ⊆ {*pathfinish ?l-rev*, *pathfinish ?p′-rev*}
by *argo*

have *1*: *pathfinish ?l-rev* = *pathstart ?p′-rev*
by (*metis make-polygonal-path-gives-path p-rev path-join-path-ends*)
{ assume *pathfinish ?p′-rev* = *pathstart ?l-rev*
then have *?case* **using** *simple-path-join-loop 1 p-rev path-image-int*
by (*smt* (*verit*, *del-insts*) *Suc.hyps*(2) *Suc.prems*(1) *Suc.prems*(2) *Suc-1* ‹*linepath y z* = *make-polygonal-path* (*drop* (*n* − 1) *vts*)› ‹*loop-free* (*make-polygonal-path* (*rev* (*butlast vts*)))› *constant-linepath-is-not-loop-free diff-Suc-Suc drop-i-is-loop-free*

dual-order.eq-iff insert-commute linepath-loop-free make-polygonal-path-gives-path path-linepath pathfinish-linepath pathstart-linepath simple-path-cases simple-path-def)
} moreover
{ assume *pathfinish ?p'-rev ≠ pathstart ?l-rev*
then have *pathstart p ≠ pathfinish p*
by (*metis Suc.prems(1)* ‹*loop-free* (*make-polygonal-path* (*butlast vts*))› ‹*pathstart* (*make-polygonal-path* (*butlast vts*)) = *pathfinish* (*make-polygonal-path* (*rev* (*butlast vts*)))› *butlast-conv-take constant-linepath-is-not-loop-free last-conv-nth less-nat-zero-code make-polygonal-path.simps(1) nat-neq-iff nth-take pathstart-linepath polygon-pathfinish polygon-pathstart take-eq-Nil yz*)
then have *arc p*
by (*metis Suc.prems(1) Suc.prems(2) arc-def loop-free-cases make-polygonal-path-gives-path*)
then have *path-image ?l-rev ∩ path-image ?p'-rev ⊆* {*pathstart ?p'-rev*}
using *loop-free-arc-split-int*
by (*metis 1 Int-commute Suc.hyps(2) Suc.prems(1) Suc.prems(2)* ‹*2 ≤ Suc n*› ‹*linepath y z = make-polygonal-path* (*drop* (*n − 1*) *vts*)› ‹*make-polygonal-path* (*butlast vts*) = *make-polygonal-path* (*take n vts*)› ‹*path-image* (*linepath y z*) = *path-image* (*linepath z y*)› ‹*path-image* (*make-polygonal-path* (*butlast vts*)) = *path-image* (*make-polygonal-path* (*rev* (*butlast vts*)))› ‹*pathstart* (*linepath y z*) = *pathfinish* (*linepath z y*)› *le-numeral-Suc lessI numerals(1) pred-numeral-simps(2) semiring-norm(26)*)
moreover have *arc ?l-rev*
by (*metis Suc.hyps(2) Suc.prems(1) Suc.prems(2) Suc-1* ‹[*y, z*] = *drop* (*n − 1*) *vts*› *arc-linepath constant-linepath-is-not-loop-free diff-Suc-Suc drop-i-is-loop-free dual-order.refl make-polygonal-path.simps(3)*)
moreover have *arc ?p'-rev*
proof−
have *?p'-rev 0 = last* (*butlast vts*) **by** (*metis 1 pathfinish-linepath pathstart-def yz*)
moreover have *?p'-rev 1 = hd* (*butlast vts*)
by (*metis* ‹*loop-free* (*make-polygonal-path* (*butlast vts*))› ‹*pathstart* (*make-polygonal-path* (*butlast vts*)) = *pathfinish* (*make-polygonal-path* (*rev* (*butlast vts*)))› *constant-linepath-is-not-loop-free hd-conv-nth make-polygonal-path.simps(1) pathfinish-def polygon-pathstart*)
moreover have *last* (*butlast vts*) ≠ *hd* (*butlast vts*) **using** *Suc.prems*
by (*metis* (*no-types, lifting*) ∗ *Suc.hyps(2) Suc-1 diff-is-0-eq index-Cons index-last leD length-butlast less-diff-conv less-imp-le-nat list.collapse list.size(3) loop-free-polygonal-path-vts-distinct not-one-le-zero plus-1-eq-Suc*)
ultimately have *?p'-rev 0 ≠ ?p'-rev 1* **by** *simp*
thus *?thesis* **using** *loop-free-p'-rev*
by (*metis arc-def loop-free-cases make-polygonal-path-gives-path pathfinish-def pathstart-def*)
qed
ultimately have *?case*
using *arc-join-eq*[*OF 1*] *arc-imp-simple-path p-rev simple-path-def* **by** *auto*
}
ultimately have *?case* **by** *blast*
}
ultimately show *?case* **by** *linarith*
qed

lemma *rev-vts-is-polygon*:
assumes *polygon-of p vts*
shows *polygon* (*make-polygonal-path* (*rev vts*))
using *rev-vts-is-loop-free assms*
unfolding *polygon-of-def polygon-def simple-path-def*
using *make-polygonal-path-gives-path*
by (*metis One-nat-def closed-path-def UNIV-def length-greater-0-conv polygon-pathfinish polygon-pathstart polygonal-path-def rangeI rev.simps*(*1*) *rev-nth rev-rev-ident*)

end
theory *Linepath-Collinearity*
imports *Polygon-Lemmas*

begin

14 Collinearity Properties

lemma *points-on-linepath-collinear*:
assumes *exists-c*: (∃ *c*. *a* − *b* = *c* ∗$_R$ *u*)
assumes *x-in-linepath*: *x* ∈ *path-image* (*linepath a b*)
shows (∃ *c*. *x* − *a* = *c* ∗$_R$ *u*) (∃ *c*. *b* − *x* = *c* ∗$_R$ *u*)
proof −
obtain *k* :: *real* **where** *k-prop*: *0* ≤ *k* ∧ *k* ≤ *1* ∧ *x* = (*1* − *k*) ∗$_R$ *a* + *k* ∗$_R$ *b*
using *x-in-linepath* **unfolding** *linepath-def path-image-def* **by** *fastforce*
then have *x* = *a* − *k* ∗$_R$ *a* + *k* ∗$_R$ *b*
by (*simp add*: *eq-diff-eq*)
then have *x* − *a*= − *k* ∗$_R$ *a* + *k* ∗$_R$ *b*
by *auto*
then have *xminusa*: *x* − *a* = −*k*∗$_R$(*a* − *b*)
by (*simp add*: *scaleR-right-diff-distrib*)
obtain *c* **where** *c-prop*: *a* − *b* = *c* ∗$_R$ *u* **using** *exists-c* **by** *blast*
show (∃ *c*. *x* − *a* = *c* ∗$_R$ *u*) **using** *xminusa c-prop*
by (*metis scaleR-scaleR*)
then show (∃ *c*. *b* − *x* = *c* ∗$_R$ *u*)
using *exists-c*
by (*metis* (*no-types*, *opaque-lifting*) *add-diff-eq diff-add-cancel minus-diff-eq scaleR-left-distrib*)
qed

lemma *three-points-collinear-property*:
fixes *a b*:: *real*^*2*
assumes *exists-c1*: (∃ *c*. *a* − *x1* = *c* ∗$_R$ *u*)
assumes *exists-c2*: (∃ *c*. *a* − *x2* = *c* ∗$_R$ *u*)
shows ∃ *c*. *x1* − *x2* = *c*∗$_R$ *u*
proof −
obtain *c1* **where** *c1-prop*: *a* − *x1* = *c1* ∗$_R$ *u*
using *exists-c1* **by** *auto*
obtain *c2* **where** *c2-prop*: *a* − *x2* = *c2* ∗$_R$ *u*
using *exists-c2* **by** *auto*

then have $a - x2 - (a - x1) = c2 *_R u - c1 *_R u$
using *c1-prop c2-prop* **by** *simp*
then have $a - x2 - (a - x1) = (c2 - c1) *_R u$
by (*simp add*: *scaleR-left-diff-distrib*)
then show *?thesis*
by *auto*
qed

lemma *in-path-image-imp-collinear*:
fixes *a b*:: *real^2*
assumes $k \in$ *path-image* (*linepath a b*)
shows *collinear* $\{a, b, k\}$
proof −
obtain *w* **where** *w-prop*: $w \in \{0..1\} \wedge k = (1 - w) *_R a + w *_R b$
using *assms* **unfolding** *path-image-def linepath-def* **by** *fast*
have *collinear* $\{0, a-b, (1 - w) *_R a + (w-1) *_R b\}$
using *collinear*
by (*smt* (*verit*) *collinear-lemma diff-minus-eq-add scaleR-minus-left scaleR-right-diff-distrib*)
then have *collinear* $\{0, a - b, k - b\}$
using *w-prop*
by (*metis* (*no-types*, *lifting*) *add.commute add-diff-cancel-left collinear-lemma scaleR-collapse scaleR-right-diff-distrib*)
then show *?thesis* **using** *assms collinear-alt collinear-3*[*of a b k*]
by *auto*
qed

lemma *two-linepath-colinearity-property*:
fixes *a b c d*:: *real^2*
assumes $y \neq z \wedge \{y, z\} \subseteq$ (*path-image* (*linepath a b*)) ∩ (*path-image* (*linepath c d*))
shows *collinear* $\{a, b, c, d\}$
proof −
have *collinear* $\{a, b, y, z\}$
using *in-path-image-imp-collinear assms*
by (*metis* (*no-types*, *lifting*) *Int-closed-segment collinear-4-3 inf.boundedE inf-idem insert-absorb2 insert-subset path-image-linepath pathstart-in-path-image pathstart-linepath*)
moreover have *collinear* $\{c, d, y, z\}$
using *in-path-image-imp-collinear assms*
by (*metis* (*no-types*, *lifting*) *Int-closed-segment collinear-4-3 inf.boundedE inf-idem insert-absorb2 insert-subset path-image-linepath pathstart-in-path-image pathstart-linepath*)
ultimately show *?thesis*
using *assms collinear-3-eq-affine-dependent collinear-4-3 insert-absorb2 insert-commute*
by (*smt* (*z3*) *collinear-3-trans*)
qed

lemma *polygon-vts-not-collinear*:
assumes *polygon-of p vts*
shows ¬ *collinear* (*set vts*)

proof −
have *len-vts*: *length vts* ≥ *3*
using *polygon-at-least-3-vertices assms* **unfolding** *polygon-of-def*
using *card-length dual-order.trans* **by** *blast*
have *compact-and-connected*: *compact* (*path-image p*) ∧ *connected* (*path-image p*)
using *inside-outside-polygon assms* **unfolding** *polygon-of-def*
using *compact-simple-path-image connected-simple-path-image polygon-def*
by *auto*
have *nonempty-path-image*: *path-image p* ≠ {}
using *assms* **unfolding** *polygon-of-def*
using *vertices-on-path-image* **by** *simp*
have *collinear-imp*: *collinear* (*set vts*) ⟹ (*collinear* (*path-image p*))
proof −
assume *collinear* (*set vts*)
then obtain *u* **where** *u-prop*: ∀ *x*∈*set vts*. ∀ *y*∈*set vts*. ∃ *c*. *x* − *y* = *c* ∗*R u*
unfolding *collinear-def* **by** *blast*
then have ∃ *c*. *x* − *y* = *c* ∗*R u* **if** *xy-in-pathimage*: *y*∈*path-image p* ∧ *x*∈*path-image p* **for** *x y*
proof −
obtain *k1* **where** *k1-prop*: *k1*<*length vts* − *1* ∧ *x* ∈ *path-image* (*linepath* (*vts* ! *k1*) (*vts* ! (*k1* + *1*)))
using *make-polygonal-path-image-property xy-in-pathimage len-vts*
by (*metis One-nat-def Suc-1 Suc-leD assms numeral-3-eq-3 polygon-of-def*)
then have ∃ *c*. (*vts* ! *k1*) − (*vts* ! (*k1* + *1*)) = *c* ∗*R u*
by (*meson add-lessD1 in-set-conv-nth less-diff-conv u-prop*)
obtain *k2* **where** *k2-prop*: *k2*<*length vts* − *1* ∧ *y* ∈ *path-image* (*linepath* (*vts* ! *k2*) (*vts* ! (*k2* + *1*)))
using *make-polygonal-path-image-property xy-in-pathimage len-vts*
by (*metis One-nat-def Suc-1 Suc-leD assms numeral-3-eq-3 polygon-of-def*)
have ∃ *c*. *vts* ! (*k2* + *1*) − (*vts* ! *k1*) = *c* ∗*R u*
using *u-prop k1-prop k2-prop*
by (*meson add-lessD1 less-diff-conv nth-mem*)
have *k2-vts-prop*: ∃ *c*. *vts* ! (*k2* + *1*) − (*vts* ! *k2*) = *c* ∗*R u*
using *u-prop k2-prop* **by** *fastforce*
have *ex-c-k2*: ∃ *c*. *vts* ! (*k2* + *1*) − *y* = *c* ∗*R u*
using *points-on-linepath-collinear*[*of vts* ! (*k2* + *1*) *vts* ! *k2 u y*] *k2-prop k2-vts-prop*
by (*meson add-lessD1 points-on-linepath-collinear*(*2*) *less-diff-conv nth-mem u-prop*)
have *k1-vts-prop*: ∃ *c*. *vts* ! (*k1* + *1*) − (*vts* ! *k1*) = *c* ∗*R u*
using *u-prop k1-prop* **by** *fastforce*
have *ex-c-k1-y*: ∃ *c*. *vts* ! (*k1* + *1*) − *y* = *c* ∗*R u*
using *points-on-linepath-collinear*[*of vts* ! (*k1* + *1*) *vts* ! *k1 u y*] *k1-prop k1-vts-prop*
by (*meson* ‹∃ *c*. *vts* ! (*k2* + *1*) − *vts* ! *k1* = *c* ∗*R u*› ‹∃ *c*. *vts* ! *k1* − *vts* ! (*k1* + *1*) = *c* ∗*R u*› *three-points-collinear-property ex-c-k2*)
have *ex-c-k1-x*: ∃ *c*. *vts* ! (*k1* + *1*) − *x* = *c* ∗*R u*
using *points-on-linepath-collinear*[*of vts* ! (*k1* + *1*) *vts* ! *k1 u x*] *k1-prop*

k1-vts-prop
by (*meson add-lessD1 points-on-linepath-collinear*(*2*) *less-diff-conv nth-mem u-prop*)
show *?thesis*
using *ex-c-k1-y ex-c-k1-y three-points-collinear-property ex-c-k1-x* **by** *blast*
qed
then show (*collinear* (*path-image p*)) **unfolding** *collinear-def* **by** *auto*
qed
{ assume ∗: *collinear* (*set vts*)
then obtain *a b*::*real*^*2* **where** *im-closed*: *path-image p* = *closed-segment a b*
using *collinear-imp compact-convex-collinear-segment-alt*[*of path-image p*] *compact-and-connected nonempty-path-image*
by *blast*
have *inside* (*closed-segment a b*) = {}
by (*simp add*: *inside-convex*)
then have *path-inside p* = {}
unfolding *path-inside-def* **using** *im-closed* **by** *auto*
then have *False*
using *inside-outside-polygon assms* **unfolding** *polygon-of-def inside-outside-def*
by *blast*
}
then show *?thesis* **by** *blast*
qed

lemma *not-collinear-with-subset*:
assumes *collinear A*
assumes ¬ *collinear* (*A* ∪ {*x*})
assumes *card A* > *2*
assumes *a* ∈ *A*
shows ¬ *collinear* ((*A* − {*a*}) ∪ {*x*})
proof−
obtain *u v* **where** *uv*: *u* ∈ *A* ∧ *v* ∈ *A* ∧ *u* ≠ *v* ∧ *u* ≠ *a* ∧ *v* ≠ *a*
proof−
have *card* (*A* − {*a*}) ≥ *2* **using** *assms* **by** *auto*
then obtain *u B* **where** *u* ∈ (*A* − {*a*}) ∧ *B* = (*A* − {*a*} − {*u*})
by (*metis bot-nat-0.extremum-unique card.empty ex-in-conv zero-neq-numeral*)
moreover then obtain *v* **where** *v* ∈ *B*
by (*metis Diff-iff One-nat-def Suc-1 assms*(*3*) *assms*(*4*) *card.empty card.insert equals0I finite.intros*(*1*) *finite-insert insert-Diff insert-commute less-irrefl*)
ultimately show *?thesis* **using** *that* **by** *blast*
qed
then have *x* ∉ *affine hull* {*u*, *v*}
using *assms*
by (*smt* (*verit*, *ccfv-threshold*) *Un-commute Un-upper1 collinear-affine-hull-collinear hull-insert hull-mono insert-absorb insert-is-Un insert-subset*)
moreover have *u* ∈ *A* − {*a*} ∧ *v* ∈ *A* − {*a*} **using** *uv* **by** *blast*
ultimately show *?thesis*
by (*metis UnCI collinear-3-imp-in-affine-hull collinear-triples insert-absorb singletonD uv*)

qed

lemma *vec-diff-scale-collinear*:
fixes *a b c* :: *real^2*
assumes $b - a = m *_R (c - a)$
shows *collinear* $\{a, b, c\}$
proof−
{ assume $m = 0$
then have $b = a$ **using** *assms* **by** *simp*
then have *collinear* $\{a, b, c\}$ **by** *auto*
} moreover
{ assume *m-nz*: $m \neq 0$
then have *c-eq*: $c = (1/m) *_R (b - a) + a$ **using** *assms* **by** *simp*
then have $c - b = (1/m - 1) *_R (b - a)$ **using** *m-nz* **by** (*simp add*: *scaleR-left.diff*)
then obtain m' **where** $c - b = m' *_R (b - a)$ **by** *fast*

then have $c - b \in span(\{b - a\})$ **by** (*simp add*: *span-breakdown-eq*)
moreover from *this* **have** $b - c \in span(\{b - a\})$ **using** *span-0 span-add-eq2* **by** *fastforce*
moreover have $c - a \in span(\{b - a\})$ **using** *assms* **by** (*simp add*: *span-breakdown-eq c-eq*)
moreover from *this* **have** $a - c \in span(\{b - a\})$ **using** *span-0 span-add-eq2* **by** *fastforce*
moreover have $b - a \in span(\{b - a\})$ **by** (*simp add*: *span-base*)
moreover from *this* **have** $a - b \in span(\{b - a\})$ **using** *span-0 span-add-eq2* **by** *fastforce*
moreover have $\forall v \in \{a, b, c\}.\ v - v \in span(\{b - a\})$ **by** (*simp add*: *span-0*)
ultimately have $\forall v \in \{a, b, c\}.\ \forall w \in \{a, b, c\}.\ v - w \in span(\{b - a\})$ **by** *blast*
then have $\forall v \in \{a, b, c\}.\ \forall w \in \{a, b, c\}.\ \exists k.\ v - w = k *_R (b - a)$
by (*simp add*: *span-breakdown-eq*)
then have *collinear* $\{a, b, c\}$ **using** *collinear-def* **by** *blast*
}
ultimately show *?thesis* **using** *assms* **by** *auto*
qed

15 Linepath Properties

lemma *good-linepath-comm*: *good-linepath a b vts* $\Longrightarrow$ *good-linepath b a vts*
unfolding *good-linepath-def*
by (*metis* (*no-types*, *opaque-lifting*) *insert-commute path-image-linepath segment-convex-hull*)

lemma *finite-set-linepaths*:
assumes *polygon*: *polygon p*
assumes *polygonal-path*: $p =$ *make-polygonal-path vts*
shows *finite* $\{(a, b).\ (a, b) \in set\ vts \times set\ vts\}$
proof −

have *finite* (*set vts*)
using *polygonal-path* **by** *auto*
then have *finite* (*set vts* × *set vts*)
by *blast*
then show *?thesis*
by *auto*
qed

lemma *linepaths-intersect-once-or-collinear*:
fixes *a b c d* :: *real^2*
assumes *path-image* (*linepath a b*) ∩ *path-image* (*linepath c d*) ≠ {}
shows *collinear* {*a*, *b*, *c*, *d*} ∨ (∃ *x*. *path-image* (*linepath a b*) ∩ *path-image* (*linepath c d*) = {*x*})
proof *safe*
assume ¬ (∃ *x*. *path-image* (*linepath a b*) ∩ *path-image* (*linepath c d*) = {*x*})
then obtain *x y* **where** *x* ≠ *y* ∧ {*x*, *y*} ⊆ *path-image* (*linepath a b*) ∩ *path-image* (*linepath c d*)
using *assms* **by** *blast*
then show *collinear* {*a*, *b*, *c*, *d*} **using** *two-linepath-colinearity-property* **by** *meson*
qed

lemma *linepaths-intersect-once-or-collinear-alt*:
fixes *a b c d* :: *real^2*
assumes *path-image* (*linepath a b*) ∩ *path-image* (*linepath c d*) ≠ {}
shows *collinear* {*a*, *b*, *c*, *d*} ∨ *card* (*path-image* (*linepath a b*) ∩ *path-image* (*linepath c d*)) = *1*
proof−
have *card* (*path-image* (*linepath a b*) ∩ *path-image* (*linepath c d*)) = *1*
⟷ (∃ *x*. *path-image* (*linepath a b*) ∩ *path-image* (*linepath c d*) = {*x*})
using *is-singleton-altdef is-singleton-def* **by** *blast*
thus *?thesis* **using** *linepaths-intersect-once-or-collinear assms* **by** *presburger*
qed

lemma *path-image-linepath-union*:
fixes *a b* :: *'a*::*euclidean-space*
assumes *d* ∈ *path-image* (*linepath a b*)
shows *path-image* (*linepath a b*) = *path-image* (*linepath a d*) ∪ *path-image* (*linepath d b*)
proof−
have *path-image* (*linepath a b*) = *closed-segment a b* **using** *path-image-linepath* **by** *simp*
also then have ... = *closed-segment a d* ∪ *closed-segment d b*
using *Un-closed-segment assms* **by** *blast*
also have ... = *path-image* (*linepath a d*) ∪ *path-image* (*linepath d b*)
using *path-image-linepath* **by** *simp*
ultimately show *?thesis* **by** *order*
qed

lemma *path-image-linepath-split*:
assumes $i < (length\ vts) - 1$
assumes $x \in path\text{-}image\ (linepath\ (vts!i)\ (vts!(i+1)))$
assumes *x-notin*: $x \notin set\ vts$
shows $path\text{-}image\ (make\text{-}polygonal\text{-}path\ vts) = path\text{-}image\ (make\text{-}polygonal\text{-}path\ ((take\ (i+1)\ vts) @ [x] @ (drop\ (i+1)\ vts)))$
using *assms*
proof(*induct length vts arbitrary*: *vts i x*)
case *0*
then show *?case* **by** *linarith*
next
case (*Suc n*)
let $?vts' = (take\ (i+1)\ vts) @ [x] @ (drop\ (i+1)\ vts)$
let $?p = make\text{-}polygonal\text{-}path\ vts$
let $?p' = make\text{-}polygonal\text{-}path\ ?vts'$
have $Suc\ n \geq 2$ **using** *Suc* **by** *linarith*
then obtain *v1 v2 vts-tail* **where** *vts-is*: $vts = v1\#v2\#vts\text{-}tail$
by (*metis Suc*(*2*) *Cons-nth-drop-Suc One-nat-def Suc-1 Suc-le-eq drop0 zero-less-Suc*)

{ assume $*$: $i = 0$
then have $vts'\text{-}is$: $?vts' = [v1, x, v2] @ vts\text{-}tail$
using *vts-is* **by** *simp*
then have *x-in*: $x \in path\text{-}image\ (linepath\ v1\ v2)$
using $*$ *Suc.prems vts-is* **by** *simp*
{ assume $*$: $vts\text{-}tail = []$
then have *p-is*: $path\text{-}image\ ?p = path\text{-}image\ (linepath\ v1\ v2)$
using *vts-is* *make-polygonal-path.simps*(*3*)[*of v1 v2*]
by *simp*
have $path\text{-}image\ ?p' = path\text{-}image\ (linepath\ v1\ x) \cup path\text{-}image\ (linepath\ x\ v2)$
using $vts'\text{-}is$ $*$ *make-polygonal-path.simps*(*4*)[*of v1 x v2* []]
using *make-polygonal-path.simps*(*3*)[*of x v2*]
by (*metis append.right-neutral list.discI nth-Cons-0 path-image-cons-union*)
then have *?case*
using *p-is path-image-linepath-union*[*of x v1 v2*] *assms*(*3*) *vts-is x-in* **by** *blast*
} moreover
{ assume $*$: $vts\text{-}tail \neq []$
then have $path\text{-}image\ ?p = path\text{-}image\ (linepath\ v1\ v2) \cup path\text{-}image\ (make\text{-}polygonal\text{-}path\ (v2\#vts\text{-}tail))$
using *path-image-cons-union vts-is* **by** (*metis list.discI nth-Cons-0*)
moreover have $path\text{-}image\ (linepath\ v1\ x) \cup path\text{-}image\ (linepath\ x\ v2) = path\text{-}image\ (linepath\ v1\ v2)$
using *path-image-linepath-union x-in* **by** *blast*
ultimately have *?case*
by (*metis* (*no-types*, *lifting*) *append-Cons append-Nil inf-sup-aci*(*6*) *list.discI nth-Cons-0 path-image-cons-union* $vts'\text{-}is$)
}
ultimately have *?case* **by** *blast*

} moreover
{ assume $* : i > 0$
then have *Suc* $n > 2$ **using** *Suc* **by** *linarith*

let *?vts-tl* = *tl vts*
let *?vts-tl′* = (*take i ?vts-tl*) @ [*x*] @ (*drop i ?vts-tl*)
let *?p-tl* = *make-polygonal-path ?vts-tl*
let *?p-tl′* = *make-polygonal-path ?vts-tl′*

have *?vts-tl*!(*i*−*1*) = *vts*!*i* ∧ *?vts-tl*!*i* = *vts*!(*i*+*1*) **using** *Suc* * **by** (*simp add*: *vts-is*)
moreover then have *x* ∈ *path-image* (*linepath* (*?vts-tl*!(*i*−*1*)) (*?vts-tl*!*i*))
using *Suc* **by** *presburger*
ultimately have *path-image ?p-tl* = *path-image ?p-tl′*
using *Suc*
by (*smt* (*verit*) * *One-nat-def Suc-leI diff-Suc-1 le-add-diff-inverse2 length-tl less-diff-conv list.sel*(*3*) *list.set-intros*(*2*) *vts-is*)
moreover have *path-image ?p* = *path-image* (*linepath v1 v2*) ∪ *path-image ?p-tl*
using *path-image-cons-union vts-is* **by** *auto*
ultimately have *?case*
by (*smt* (*verit*, *ccfv-threshold*) *Nil-is-append-conv Suc-eq-plus1* ‹*i* = *0* ⟹ *path-image* (*make-polygonal-path vts*) = *path-image* (*make-polygonal-path* (*take* (*i* + *1*) *vts* @ [*x*] @ *drop* (*i* + *1*) *vts*))› *append-Cons append-same-eq append-take-drop-id drop-Suc hd-append2 hd-conv-nth list.sel*(*1*) *list.sel*(*3*) *path-image-cons-union take-eq-Nil vts-is*)
}
ultimately show *?case* **by** *linarith*
qed

lemma *linepath-split-is-loop-free*:
assumes *d* ∈ *path-image* (*linepath a b*)
assumes *d* ∉ {*a*, *b*}
shows *loop-free* (*make-polygonal-path* [*a*, *d*, *b*]) (**is** *loop-free ?p*)
proof−
let *?l1* = *linepath a d*
let *?l2* = *linepath d b*
have *path-image ?l1* ∩ *path-image ?l2* = {*d*} **using** *Int-closed-segment assms*(*1*) **by** *auto*
moreover have *arc ?l1* ∧ *arc ?l2* **using** *assms*(*2*) **by** *fastforce*
ultimately show *?thesis*
by (*metis arc-imp-simple-path arc-join-eq-alt make-polygonal-path.simps*(*3*) *make-polygonal-path.simps*(*4*) *pathfinish-linepath pathstart-linepath simple-path-def*)
qed

lemma *loop-free-linepath-split-is-loop-free*:
assumes *p* = *make-polygonal-path vts*
assumes *loop-free p*
assumes *n* = *length vts*

assumes $i < n - 1$
assumes $x \in$ *path-image* (*linepath* (*vts*!i) (*vts*!(i+1))) $\wedge$ $x \notin$ *set vts*
assumes $vts' =$ (*take* (i+1) *vts*) @ [x] @ (*drop* (i+1) *vts*)
assumes $p' =$ *make-polygonal-path vts'*
shows *loop-free p'* $\wedge$ *path-image p'* = *path-image p*
using *assms*
proof(*induct i arbitrary*: *p vts p' vts' n*)
case *0*
let *?vts-tl* = *tl vts*
let *?p-tl* = *make-polygonal-path ?vts-tl*
let *?vts'-tl* = *tl vts'*
let *?p'-tl* = *make-polygonal-path ?vts'-tl*
let *?a* = *vts*!*0*
let *?b* = *vts*!*1*
let *?l* = *linepath ?a ?b*
let *?l'* = *make-polygonal-path* [*?a*, *x*, *?b*]

have *vts'*: *vts'* = [*?a*, *x*] @ *?vts-tl*
using *0*
by (*metis* (*no-types*, *lifting*) *Suc-eq-plus1 append-Cons append-eq-append-conv2 append-self-conv bot-nat-0.not-eq-extremum diff-is-0-eq drop0 drop-Suc list.collapse nth-Cons-0 take-Suc take-all-iff take-eq-Nil*)

have $x \notin$ {*?a*, *?b*}
by (*metis 0*(*3*−*5*) *One-nat-def Suc-eq-plus1 bot-nat-0.not-eq-extremum diff-is-0-eq insert-iff less-diff-conv nth-mem singletonD take-Suc-eq take-all-iff*)
then have *lf-l'*: *loop-free ?l'* **using** *linepath-split-is-loop-free*[*of x ?a ?b*] *0* **by** *simp*

{ **assume** *length ?vts-tl* = *1*
then have *vts'* = [*?a*, *x*, *?b*]
by (*metis Cons-nth-drop-Suc One-nat-def append-eq-Cons-conv drop0 drop-eq-Nil le-numeral-extra*(*4*) *nth-tl vts' zero-less-one*)
then have *?case* **using** *linepath-split-is-loop-free path-image-linepath-split*
by (*metis 0.prems*(*1*) *0.prems*(*3*) *0.prems*(*4*) *0.prems*(*5*) *0.prems*(*6*) *0.prems*(*7*) *lf-l'*)
} **moreover**
{ **assume** ∗: *length ?vts-tl* ≥ *2*
then have *p*: *p* = *?l* +++ *?p-tl*
using *make-polygonal-path.simps*(*4*)[*of ?a ?b*]
by (*metis* (*no-types*, *opaque-lifting*) *0*(*1*) *0*(*3*) *0*(*4*) *Cons-nth-drop-Suc One-nat-def Suc-1 Suc-le-eq diff-is-0-eq drop-0 drop-Suc length-tl less-nat-zero-code nat-le-linear nth-tl*)

have *loop-free ?p-tl*
using *tail-of-loop-free-polygonal-path-is-loop-free 0* ∗
by (*metis list.exhaust-sel list.sel*(*2*))
moreover have *l-l'*: *path-image ?l* = *path-image ?l'*
using *path-image-linepath-split 0*

by (*metis One-nat-def Suc-eq-plus1 list.discI make-polygonal-path.simps*(3) *nth-Cons-0 path-image-cons-union path-image-linepath-union*)
moreover have *path-image ?l′* ∩ *path-image ?p-tl* ⊆ {*?a*, *?b*}
by (*metis* (*mono-tags*, *opaque-lifting*) *p l-l′ 0.prems*(1) *0.prems*(2) *make-polygonal-path-gives-path path-join-path-ends pathfinish-linepath pathstart-linepath simple-path-def simple-path-joinE*)
moreover have *arc p* ⟶ *path-image ?l′* ∩ *path-image ?p-tl* ⊆ {*?b*}
using *p l-l′*
by (*metis arc-def arc-join-eq make-polygonal-path-gives-path path-join-eq path-linepath pathfinish-linepath*)
moreover have *arc p* ⟷ *hd* [*?a*, *x*, *?b*] ≠ *last* (*tl vts*)
by (*metis* ∗ *0.prems*(1) *0.prems*(2) *arc-def arc-simple-path last-conv-nth last-tl list.sel*(1) *list.sel*(2) *list.size*(3) *loop-free-cases make-polygonal-path-gives-path not-numeral-le-zero polygon-pathfinish polygon-pathstart*)
moreover have *vts′* = [*?a*, *x*, *?b*] @ *tl ?vts-tl*
by (*metis drop-Suc 0.prems*(3) *0.prems*(4) *One-nat-def append-Cons append-Nil append-take-drop-id length-tl nth-tl take-Suc-conv-app-nth take-eq-Nil vts′*)
moreover have *last* [*?a*, *x*, *?b*] = *hd ?vts-tl*
by (*metis 0.prems*(3) *0.prems*(4) *One-nat-def hd-conv-nth last.simps length-greater-0-conv length-tl list.discI nth-tl*)
moreover have *pathfinish ?l* = *pathstart ?p-tl*
by (*metis* (*no-types*) *0.prems*(1) *make-polygonal-path.simps*(3) *make-polygonal-path-gives-path p path-join-eq*)
moreover have ⋀*v va vb vs. pathfinish* (*linepath v va*) = *pathstart* (*make-polygonal-path* (*va* # *vb* # *vs*))

by (*metis* (*no-types*) *make-polygonal-path.simps*(3) *make-polygonal-path.simps*(4) *make-polygonal-path-gives-path path-join-eq*)
ultimately have *loop-free p′*
using *loop-free-append*[*of p′ vts′ ?l′* [*?a*, *x*, *?b*] *?p-tl ?vts-tl*]
by (*metis* (*no-types*) *0.prems*(1) *0.prems*(2) *0.prems*(7) *arc-simple-path lf-l′ make-polygonal-path.simps*(3) *make-polygonal-path.simps*(4) *make-polygonal-path-gives-path p pathfinish-join pathstart-linepath simple-path-def simple-path-joinE*)
then have *?case*
using *0*(1) *0*(3) *0*(4) *0*(5) *0*(6) *0*(7) *path-image-linepath-split* **by** *blast*
}
ultimately show *?case*
by (*metis 0*(3,4) *One-nat-def Suc-lessI length-tl less-eq-Suc-le nat-1-add-1 plus-1-eq-Suc*)
next
case (*Suc i*)
let *?vts-tl* = *tl vts*
let *?p-tl* = *make-polygonal-path ?vts-tl*
let *?vts′-tl* = *tl vts′*
let *?p′-tl* = *make-polygonal-path ?vts′-tl*
let *?a* = *vts*!*0*
let *?b* = *vts*!*1*
let *?l* = *linepath ?a ?b*

have *?vts-tl*!*i* = *vts*!(*Suc i*) ∧ *?vts-tl*!(*i*+*1*) = *vts*!((*Suc i*) + *1*)

by (*metis Suc.prems*(3) *Suc.prems*(4) *add-Suc-right add-Suc-shift diff-is-0-eq linorder-not-le list.exhaust-sel list.size*(3) *not-less-zero nth-Cons-Suc*)
moreover have *set ?vts-tl* ⊆ *set vts*
by (*metis list.sel*(2) *list.set-sel*(2) *subsetI*)
ultimately have *x* ∈ *path-image* (*linepath* (*?vts-tl*!*i*) (*?vts-tl*!(*i*+1))) ∧ *x* ∉ *set ?vts-tl*
using *Suc.prems*(5) **by** *auto*
moreover have *vts'-tl*: *?vts'-tl* = (*take* (*i*+1) *?vts-tl*) @ [*x*] @ (*drop* (*i*+1) *?vts-tl*)
by (*metis Suc.prems*(3) *Suc.prems*(4) *Suc.prems*(6) *Suc-eq-plus1 drop-Suc leD length-tl take-all-iff take-eq-Nil take-tl tl-append2 zero-eq-add-iff-both-eq-0 zero-neq-one*)
moreover have *loop-free ?p-tl*
using *tail-of-loop-free-polygonal-path-is-loop-free Suc.prems*
by (*metis Nitpick.size-list-simp*(2) *Suc-1 Suc-leI Suc-neq-Zero diff-0-eq-0 diff-Suc-1 less-one linorder-neqE-nat list.collapse not-less-zero*)
ultimately have *ih*: *loop-free ?p'-tl* ∧ *path-image ?p'-tl* = *path-image ?p-tl*
using *Suc.prems Suc.hyps*[*of ?p-tl ?vts-tl - ?vts'-tl ?p'-tl*] **by** *simp*

have *p*: *p* = *?l* +++ *?p-tl*
proof −
have *f1*: ∀ *vs*. (*hd* (*tl vs*)::(*real*, 2) *vec*) = *vs* ! 1 ∨ [] = *vs* ∨ [] = *tl vs*
by (*metis* (*no-types*) *One-nat-def hd-conv-nth list.collapse nth-Cons-Suc*)
have [] ≠ *tl vts* ∧ *vts* ≠ [] ∧ *tl vts* ≠ [*hd* (*tl vts*)]
by (*metis Suc.prems*(1) *Suc.prems*(2) ‹*loop-free* (*make-polygonal-path* (*tl vts*))› *constant-linepath-is-not-loop-free make-polygonal-path.simps*(1) *make-polygonal-path.simps*(2))
then have *p* = *make-polygonal-path* [*hd vts*, *vts* ! 1] +++ *make-polygonal-path* (*tl vts*) ∧ *vts* ≠ []
using *f1* **by** (*metis* (*full-types*) *Suc.prems*(1) *list.collapse make-polygonal-path.simps*(3) *make-polygonal-path.simps*(4))
then show *?thesis*
by (*simp add*: *hd-conv-nth*)
qed

have *length vts'* ≥ 3 **using** *Suc.prems* **by** *force*
moreover have *ab*: *?a* = *vts'*!0 ∧ *?b* = *vts'*!1
using *Suc.prems*
by (*smt* (*verit*, *ccfv-SIG*) *One-nat-def Suc-eq-plus1 add-Suc-right append-Cons drop0 drop-Suc length-tl less-nat-zero-code list.exhaust-sel list.size*(3) *nat-diff-split nth-Cons-0 nth-Cons-Suc take-Suc zero-less-Suc*)
ultimately have *p'*: *p'* = *?l* +++ *?p'-tl*
using *Suc.prems*(7) *make-polygonal-path.simps*(4)[*of ?a ?b*]
by (*metis* (*no-types*, *opaque-lifting*) *Cons-nth-drop-Suc One-nat-def Suc-leD Suc-le-eq drop0 drop-Suc numeral-3-eq-3*)

have *nonarc*: *path-image ?l* ∩ *path-image ?p-tl* ⊆ {*?a*, *?b*}
using *simple-path-join-loop-eq Suc.prems*
by (*smt* (*verit*, *ccfv-threshold*) *p One-nat-def length-tl less-zeroE make-polygonal-path-gives-path nth-tl order.strict-iff-not order-le-less-trans path-join-eq path-linepath pathfinish-linepath pathstart-linepath polygon-pathstart simple-path-def simple-path-joinE take-Nil take-all-iff*)

have *arc*: *arc p* $\longrightarrow$ *path-image ?l* $\cap$ *path-image ?p-tl* $\subseteq$ $\{?b\}$
using *arc-join-eq*
by (*metis Suc.prems*(*1*) *p make-polygonal-path-gives-path path-join-eq path-linepath pathfinish-linepath*)

{ assume *arc p*
moreover then have *path-image ?l* $\cap$ *path-image ?p'-tl* $\subseteq$ $\{?b\}$ **using** *arc ih*
by *presburger*
moreover have *pathfinish ?l* = *pathstart ?p'-tl*
by (*metis Suc.prems*(*7*) *make-polygonal-path-gives-path p' path-join-path-ends*)
ultimately have *?case* **using** *p' arc-join-eq*[*of ?l ?p'-tl*]
by (*smt* (*verit*, *ccfv-SIG*) *Nil-is-append-conv Suc.prems*(*3*) *Suc.prems*(*4*) *Suc-eq-plus1 vts'-tl arc-simple-path drop-eq-Nil ih last-appendR last-conv-nth last-drop leD length-tl make-polygonal-path-gives-path p path-image-join path-join-eq path-linepath pathfinish-linepath polygon-pathfinish simple-path-def simple-path-joinE take-all-iff take-eq-Nil*)
} moreover
{ assume $\neg$ *arc p*
then have *pathstart ?l* = *pathfinish ?p'-tl* $\wedge$ *pathfinish ?l* = *pathstart ?p'-tl*
by (*smt* (*verit*, *del-insts*) *Nil-is-append-conv Nil-tl One-nat-def Suc.prems*(*2*) *Suc.prems*(*3*) *Suc.prems*(*4*) *Suc-eq-plus1 vts'-tl ab arc-def drop-eq-Nil last-appendR last-conv-nth last-drop leD length-tl list.collapse loop-free-cases make-polygonal-path-gives-path nth-Cons-Suc p path-join-eq path-linepath pathfinish-join pathfinish-linepath pathstart-join polygon-pathfinish polygon-pathstart take-all-iff take-eq-Nil*)
then have *?case* **using** *simple-path-join-loop-eq*[*of ?l ?p'-tl*] *p' nonarc*
by (*smt* (*verit*, *ccfv-threshold*) *One-nat-def Suc.prems*(*2*) *Suc.prems*(*3*) *Suc.prems*(*4*) *arc-def constant-linepath-is-not-loop-free dual-order.strict-trans ih leD length-tl loop-free-cases make-polygonal-path-gives-path not-loop-free-first-component nth-tl p path-image-join path-linepath pathfinish-linepath pathstart-linepath polygon-pathstart simple-path-def simple-path-join-loop-eq take-all-iff take-eq-Nil zero-less-Suc*)
}
ultimately show *?case* **by** *argo*
qed

lemma *polygon-linepath-split-is-polygon*:
assumes *polygon-of p vts*
assumes *i* < (*length vts*) − *1*
assumes *a* = *vts*!*i* $\wedge$ *b* = *vts*!(*i*+*1*)
assumes *x* $\in$ *path-image* (*linepath a b*) $\wedge$ *x* $\notin$ *set vts*
assumes *vts'* = (*take* (*i*+*1*) *vts*) @ [*x*] @ (*drop* (*i*+*1*) *vts*)
shows *polygon* (*make-polygonal-path vts'*)
proof−
let *?p'* = *make-polygonal-path vts'*
have *path ?p'* **using** *assms make-polygonal-path-gives-path* **by** *presburger*
moreover have *loop-free ?p'* **using** *assms loop-free-linepath-split-is-loop-free*
by (*metis polygon-def polygon-of-def simple-path-def*)
moreover have *closed-path ?p'*
proof−

have *hd vts′ = hd vts*
using *assms*
by (*metis hd-append2 hd-take le-diff-conv linorder-not-less take-all-iff take-eq-Nil2 trans-less-add2 zero-less-one*)
moreover have *last vts′ = last vts*
using *assms linordered-semidom-class.add-diff-inverse* **by** *auto*
ultimately show *?thesis*
by (*metis closed-path-def ‹path ?p′› append-butlast-last-id append-eq-conv-conj append-is-Nil-conv assms(1) assms(5) have-wraparound-vertex hd-conv-nth length-butlast not-Cons-self nth-append-length polygon-of-def polygon-pathfinish polygon-pathstart*)
qed
ultimately show *?thesis* **unfolding** *polygon-def polygonal-path-def simple-path-def assms(5)* **by** *blast*
qed

16 Measure of linepaths

lemma *linepath-is-negligible-vertical*:
fixes *a b* :: *real^2*
assumes *a$1 = b$1*
defines *p ≡ linepath a b*
shows *negligible* (*path-image p*)
proof−
have *p-t*: ∀ *t* ∈ {*0..1*}. (*p t*)$*1 = a$1*
using *linepath-in-path p-def segment-vertical assms* **by** *blast*

let *?x = a$1*
let *?e1 = (vector* [*1*, *0*])::*real^2*

have (*1*::*real*) ∈ *Basis* **by** *simp*
then have *axis 1* (*1*::*real*) ∈ (⋃ *i*. ⋃ *u*∈(*Basis*::(*real set*)). {*axis i u*}) **by** *blast*
moreover have *?e1 = axis 1* (*1*::*real*)
unfolding *axis-def vector-def* **by** *auto*
ultimately have *e1-basis*: *?e1* ∈ (*Basis*::((*real^2*) *set*)) **by** *simp*
then have *negligible* {*v*. *v* · *?e1 = ?x*} (**is** *negligible ?S*)
using *negligible-standard-hyperplane* **by** *auto*
moreover have ∀ *t* ∈ {*0..1*}. (*p t*) · *?e1 = ?x*
proof *clarify*
fix *t* :: *real*
assume *t*: *t* ∈ {*0..1*}
have (*p t*) · *?e1* = (*p t*)$*1*
by (*smt* (*verit*, *best*) *e1-basis cart-eq-inner-axis vec-nth-Basis vector-2*(*1*))
also have ... = *?x* **using** *p-t t* **by** *blast*
finally show (*p t*) · *?e1 = ?x* .
qed
moreover from *this* **have** *path-image p* ⊆ *?S* **unfolding** *path-image-def* **by** *blast*
ultimately show *?thesis* **using** *negligible-subset* **by** *blast*
qed

lemma *linepath-is-negligible-non-vertical*:
fixes *a b* :: *real^2*
assumes *a$1* < *b$1*
defines *p* ≡ *linepath a b*
shows *negligible* (*path-image p*)
proof−
let *?A* = (*vector* [*vector* [*1*, *b$1* − *a$1*], *vector* [*0*, *b$2* − *a$2*]])::(*real^2^2*)
let *?f1* = λ*v*::*real^2*. (*?A* ∗*v v*)
let *?id* = λ*v*::*real^2*. *v*
let *?f-a* = λ*v*::*real^2*. *a*
let *?f2* = λ*v*. *?id v* + *?f-a v*
let *?f* = *?f2* ∘ *?f1*

let *?O* = (*vector* [*0*, *0*])::*real^2*
let *?e2* = (*vector* [*0*, *1*])::*real^2*
let *?y-unit-seg-path* = *linepath ?O ?e2*
let *?y-unit-seg* = *path-image ?y-unit-seg-path*

have ∀ *t* ∈ {*0..1*}. *?f* (*?y-unit-seg-path t*) = *p t*
proof *clarify*
fix *t* :: *real*
assume *t*: *t* ∈ {*0..1*}
then obtain *v* **where** *v*: *?y-unit-seg-path t* = *v* **by** *auto*
then have *v* = (*1* − *t*) ∗*R ?O* + *t* ∗*R ?e2* **unfolding** *linepath-def* **by** *auto*
then have *v* = *t* ∗*R ?e2*
by (*smt* (*verit*, *best*) *t v exhaust-2 linepath-0 scaleR-zero-left vec-eq-iff vector-2*(*1*) *vector-2*(*2*) *vector-scaleR-component*)
then have *?f v* = *p t*
proof−
assume *v* = *t* ∗*R vector* [*0*, *1*]
then have *v* = *vector* [*t* ∗ *0*, *t* ∗ *1*]
by (*smt* (*verit*, *del-insts*) *exhaust-2 mult-cancel-left1 real-scaleR-def scaleR-zero-right vec-eq-iff vector-2*(*1*) *vector-2*(*2*) *vector-scaleR-component*)
then have *v*: *v* = *vector* [*0*, *t*] **by** *auto*

have *f1*: *?f1 v* = *vector* [*t* ∗ (*b$1* − *a$1*), *t* ∗ (*b$2* − *a$2*)] (**is** *?f1 v* = *?f1-v*)
by (*simp add*: *mat-vec-mult-2 v*)

have *?f2 ?f1-v* = *vector* [*t* ∗ (*b$1* − *a$1*), *t* ∗ (*b$2* − *a$2*)] + *vector* [*a$1*, *a$2*]
by (*smt* (*verit*) *exhaust-2 vec-eq-iff vector-2*(*1*) *vector-2*(*2*))
also have ... = *vector* [*t* ∗ (*b$1* − *a$1*) + *a$1*, *t* ∗ (*b$2* − *a$2*) + *a$2*]
by (*smt* (*verit*, *del-insts*) *vector-add-component exhaust-2 vec-eq-iff vector-2*(*1*) *vector-2*(*2*))
also have ... = *vector* [*t* ∗ *b$1* + (*1* − *t*) ∗ *a$1*, *t* ∗ *b$2* + (*1* − *t*) ∗ *a$2*]
by *argo*
also have ... = *t* ∗*R b* + (*1* − *t*) ∗*R a*
by (*smt* (*verit*, *del-insts*) *exhaust-2 real-scaleR-def vec-eq-iff vector-2*(*1*)

vector-2(2) vector-add-component vector-scaleR-component)
    finally have *?f2 ?f1-v = t* $*_R$ *b + (1 − t)* $*_R$ *a* **.**
    thus *?thesis* **using** *p-def f1* **unfolding** *linepath-def* **by** *simp*
   qed
   thus *?f* (*?y-unit-seg-path t*) = *p t* **using** *v* **by** *simp*
  qed

  then have *?f ‘ ?y-unit-seg = path-image p* **unfolding** *path-image-def* **by** *force*
  moreover have *?f differentiable-on ?y-unit-seg*
  proof−
   have *linear ?f1* **by** *auto*
   then have *?f1 differentiable-on ?y-unit-seg*
    using *linear-imp-differentiable* **by** (*simp add*: *linear-imp-differentiable-on*)
   moreover have *?f2 differentiable-on* (*?f1 ‘ ?y-unit-seg*)
   proof−
    have *?id differentiable-on ?f1 ‘ ?y-unit-seg*
     using *differentiable-const* **by** *simp*
    moreover have *?f-a differentiable-on ?f1 ‘ ?y-unit-seg*
     using *differentiable-ident* **by** *simp*
    ultimately show *?f2 differentiable-on ?f1 ‘ ?y-unit-seg*
     using *differentiable-compose* **by** *simp*
   qed
   ultimately show *?thesis* **using** *differentiable-compose*
    by (*simp add*: *differentiable-chain-within differentiable-on-def*)
  qed
  moreover have *negligible ?y-unit-seg*
   using *linepath-is-negligible-vertical*[*of ?O ?e2*] **by** *simp*
  ultimately show *?thesis*
   using *negligible-differentiable-image-negligible* **by** *fastforce*
qed

lemma *linepath-is-negligible*:
  fixes *a b* :: *real^2*
  defines *p* ≡ *linepath a b*
  shows *negligible* (*path-image p*)
proof−
  { **assume** *a$1 = b$1*
   then have *?thesis* **using** *linepath-is-negligible-vertical p-def* **by** *blast*
  } **moreover**
  { **assume** *a$1 < b$1*
   then have *?thesis* **using** *linepath-is-negligible-non-vertical p-def* **by** *blast*
  } **moreover**
  { **assume** *a*: *a$1 > b$1*
   let *?p-rev = reversepath p*
   have *path-image p = path-image ?p-rev* **by** *simp*
   moreover have *?p-rev = linepath b a* **using** *p-def* **by** *simp*
   ultimately have *?thesis* **using** *a linepath-is-negligible-non-vertical*[*of b a*] **by**
simp
  }

ultimately show *?thesis* **by** *linarith*
qed

lemma *linepath-has-emeasure-0*:
emeasure lebesgue (*path-image* (*linepath* (*a*::(*real*^*2*)) (*b*::(*real*^*2*)))) = *0*
using *linepath-is-negligible emeasure-notin-sets negligible-iff-emeasure0* **by** *blast*

lemma *linepath-has-measure-0*:
measure lebesgue (*path-image* (*linepath* (*a*::(*real*^*2*)) (*b*::(*real*^*2*)))) = *0*
using *linepath-has-emeasure-0 linepath-is-negligible negligible-imp-measure0* **by** *blast*

end
theory *Polygon-Convex-Lemmas*
imports
Polygon-Lemmas
Linepath-Collinearity

begin

17 Misc. Convex Polygon Properties

lemma *polygon-path-image-subset-convex*:
assumes *length vts* > *0*
shows *path-image* (*make-polygonal-path vts*) ⊆ *convex hull* (*set vts*) (**is** *path-image ?p* ⊆ *?S*)
using *assms*
proof(*induct vts rule*: *make-polygonal-path.induct*)
case *1*
then show *?case* **by** *simp*
next
case (*2 a*)
then show *?case* **by** *auto*
next
case (*3 a b*)
show *?case* (**is** *path-image ?p* ⊆ *?S*)
proof(*rule subsetI*)
fix *x*
assume *x-in-path-image*: *x* ∈ *path-image ?p*
then have *x* ∈ *path-image* (*linepath a b*) **by** *auto*
thus *x* ∈ *?S*
unfolding *path-image-def linepath-def*
by (*smt* (*verit*, *ccfv-SIG*) ‹*x* ∈ *path-image* (*linepath a b*)› *convex-alt convex-convex-hull hull-subset in-mono in-segment*(*1*) *linepath-image-01 list.set-intros*(*1*) *path-image-def set-subset-Cons*)
qed
next
case (*4 a b c tl*)
let *?vts* = *a* # *b* # *c* # *tl*

show *?case* (**is** *path-image ?p* $\subseteq$ *?S*)
proof(*rule subsetI*)
fix *x*
assume *x-in-path-image*: *x* $\in$ *path-image ?p*
show *x* $\in$ *?S*
proof *cases*
assume *x* $\in$ *set ?vts*
thus *?thesis* **by** (*simp add*: *hull-inc*)
next
assume *x-notin*: *x* $\notin$ *set ?vts*
obtain *u* **where** *p-u*: *u* $\in$ {*0..1*} $\wedge$ *?p u* = *x*
using *x-in-path-image* **unfolding** *path-image-def* **by** *auto*
then have *p-head-tail*: *?p* = (*linepath a b*) *+++ make-polygonal-path* (*b # c # tl*)
by *auto*
have *abc-in-S*: *set ?vts* $\subseteq$ *convex hull* (*set ?vts*) **by** (*simp add*: *hull-subset*)
{ assume *u-assm*: *u* $\leq$ *1/2*
then have *?p u* = (*1 − 2 ∗ u*) $*_R$ *a* + (*2 ∗ u*) $*_R$ *b*
using *p-head-tail* **unfolding** *linepath-def joinpaths-def*
by *presburger*
hence *x* $\in$ *?S*
using *abc-in-S convexD-alt*[*of ?S a b 2 ∗ u*] *u-assm p-u* **by** *simp*
} moreover
{ assume *u-assm*: *u* > *1/2*
then have *x* = (*make-polygonal-path* (*b # c # tl*) (*2 ∗ u − 1*)) (**is** *x* = (*?p′* (*2 ∗ u − 1*)))
using *p-head-tail p-u* **unfolding** *linepath-def joinpaths-def* **by** *auto*
moreover have *0* < (*2 ∗ u − 1*) **using** *u-assm* **by** *linarith*
ultimately have *x* $\in$ *path-image ?p′*
using *p-u* **by** (*simp add*: *path-image-def*)
moreover have *path-image ?p′* $\subseteq$ *convex hull* (*set* (*b # c # tl*)) **using** *4*(*1*) **by** *auto*
moreover have *...* $\subseteq$ *convex hull* (*set* (*a # b # c # tl*))
by (*meson hull-mono set-subset-Cons*)
ultimately have *x* $\in$ *?S* **by** *auto*
}
ultimately show *?thesis* **by** *linarith*
qed
qed
qed

lemma *convex-contains-simple-closed-path-imp-contains-path-inside*:
assumes *convex S*
assumes *simple-path p* $\wedge$ *closed-path p*
assumes *path-image p* $\subseteq$ *S*
shows *path-inside p* $\subseteq$ *S*
by (*metis* (*no-types*, *opaque-lifting*) *Compl-subset-Compl-iff Un-subset-iff assms*(*1*) *assms*(*3*) *boolean-algebra-class.boolean-algebra.double-compl outside-subset-convex path-inside-def union-with-inside*)

lemma *convex-polygon-is-convex-hull*:
assumes *polygon p*
assumes *convex* (*path-inside p* ∪ *path-image p*)
assumes *p* = *make-polygonal-path vts*
shows *convex hull* (*set vts*) = *path-inside p* ∪ *path-image p* (**is** *?hull* = *?poly*)
proof−
have *?hull* ⊆ *?poly*
proof(*rule subsetI*)
fix *x*
assume *x* ∈ *?hull*
moreover have ∀ *H*. (*convex H* ∧ (*set vts*) ⊆ *H*) ⟶ *?hull* ⊆ *H* **by** (*simp add*: *hull-minimal*)
moreover have *convex* (*?poly*) ∧ (*set vts*) ⊆ *?poly*
using *assms*(*2*) *assms*(*3*) *vertices-on-path-image* **by** *auto*
ultimately show *x* ∈ *?poly* **by** *auto*
qed
moreover have *?hull* ⊇ *?poly*
proof(*rule subsetI*)
fix *x*
assume *x* ∈ *?poly*
moreover have *path-image p* ⊆ *?hull*
using *polygon-path-image-subset-convex*[*of vts*] *polygon-at-least-3-vertices assms*
by *force*
moreover from *calculation* **have** *path-inside p* ⊆ *?hull*
using *convex-contains-simple-closed-path-imp-contains-path-inside polygon-def assms*(*1*)
by *auto*
ultimately show *x* ∈ *?hull* **by** *auto*
qed
ultimately show *?thesis* **by** *auto*
qed

lemma *convex-polygon-inside-is-convex-hull-interior*:
assumes *polygon p*
assumes *convex* (*path-inside p*)
assumes *p* = *make-polygonal-path vts*
shows *interior* (*convex hull* (*set vts*)) = *path-inside p*
by (*metis* (*no-types*, *lifting*) *assms closure-Un-frontier convex-closure convex-interior-closure convex-polygon-is-convex-hull inside-outside-def inside-outside-polygon interior-eq*)

lemma *convex-polygon-inside-is-convex-hull-interior2*:
assumes *polygon p*
assumes *convex* (*path-inside p* ∪ *path-image p*)
assumes *p* = *make-polygonal-path vts*
shows *interior* (*convex hull* (*set vts*)) = *path-inside p*
using *assms closure-Un-frontier convex-closure convex-interior-closure convex-polygon-is-convex-hull inside-outside-def inside-outside-polygon interior-eq*

by (*smt* (*verit*, *best*) *List.finite-set compact-eq-bounded-closed finite-imp-compact-convex-hull frontier-complement inside-frontier-eq-interior outside-inside path-inside-def path-outside-def sup-commute*)

lemma *polygon-convex-iff*:
assumes *polygon p*
shows *convex* (*path-inside p*) ⟷ *convex* (*path-inside p* ∪ *path-image p*)
using *convex-polygon-inside-is-convex-hull-interior*
using *convex-polygon-inside-is-convex-hull-interior2*
by (*metis Jordan-inside-outside-real2 closed-path-def assms closure-Un-frontier convex-closure convex-interior convex-polygon-is-convex-hull path-inside-def polygon-def polygon-to-polygonal-path*)

lemma *convex-polygon-frontier-is-path-image*:
assumes *polygon-of p vts*
assumes *convex* (*path-inside p*)
shows *frontier* (*convex hull* (*set vts*)) = *path-image p*
using *assms*
unfolding *frontier-def polygon-of-def*
by (*metis* (*no-types*, *lifting*) *Jordan-inside-outside-real2 closed-path-def convex-closure-interior convex-convex-hull convex-polygon-inside-is-convex-hull-interior frontier-def interior-interior path-inside-def polygon-def*)

lemma *convex-polygon-frontier-is-path-image2*:
assumes *polygon p*
assumes *convex* (*path-inside p*)
shows *frontier* (*path-image p* ∪ *path-inside p*) = *path-image p*
using *assms*
by (*simp add*: *Jordan-inside-outside-real2 closed-path-def path-inside-def polygon-def union-with-inside*)

lemma *convex-polygon-frontier-is-path-image3*:
assumes *polygon p*
assumes *convex* (*path-image p* ∪ *path-inside p*)
shows *frontier* (*path-image p* ∪ *path-inside p*) = *path-image p*
using *assms polygon-convex-iff*
by (*simp add*: *convex-polygon-frontier-is-path-image2 sup-commute*)

lemma *polygon-frontier-is-path-image*:
assumes *polygon p*
shows *frontier* (*path-inside p*) = *path-image p*
using *inside-outside-polygon* **unfolding** *inside-outside-def*
using *assms* **by** *presburger*

lemma *convex-path-inside-means-convex-polygon*:
assumes *polygon p*
assumes *frontier* (*convex hull* (*set vts*)) = *path-image p*
shows *convex* (*path-inside p*)
by (*metis List.finite-set assms*(*2*) *convex-convex-hull convex-interior finite-imp-bounded-convex-hull*

inside-frontier-eq-interior path-inside-def)

lemma *convex-hull-of-polygon-is-convex-hull-of-vts*:
assumes *polygon-of p vts*
shows *convex hull* (*path-image p* ∪ *path-inside p*) = *convex hull* (*set vts*)
proof −
have *len-vts*: *length vts* > *0*
by (*metis assms card.empty empty-set length-greater-0-conv not-numeral-le-zero polygon-at-least-3-vertices polygon-of-def*)
have *path-image p* ∪ *path-inside p* ⊆ *convex hull* (*set vts*)
using *polygon-path-image-subset-convex*[*OF len-vts*]
using *assms convex-contains-simple-closed-path-imp-contains-path-inside polygon-def polygon-of-def* **by** *auto*
then have *subset1*: *convex hull* (*path-image p* ∪ *path-inside p*) ⊆ *convex hull* (*set vts*)
by (*simp add*: *convex-hull-subset*)
have *set vts* ⊆ *path-image p* ∪ *path-inside p* **using** *assms vertices-on-path-image*

by (*simp add*: *polygon-of-def sup.coboundedI1*)
then have *subset2*: *convex hull* (*set vts*) ⊆ *convex hull* (*path-image p* ∪ *path-inside p*)
by (*simp add*: *hull-mono*)
show *?thesis* **using** *subset1 subset2*
by *auto*
qed

lemma *convex-hull-frontier-polygon*:
assumes *polygon-of p vts*
assumes ¬ *set vts* ⊆ *frontier* (*convex hull* (*set vts*))
shows ¬ *convex* (*path-inside p*)
by (*metis assms*(*1*) *assms*(*2*) *convex-polygon-frontier-is-path-image polygon-of-def vertices-on-path-image*)

lemma *frontier-int-subset*:
assumes *A* ⊆ *B*
shows (*frontier B*) ∩ *A* ⊆ *frontier A*
by (*metis assms closure-Un-frontier frontier-Int inf.absorb-iff2 inf-sup-aci*(*1*) *subset-Un-eq sup-inf-distrib2*)

lemma *in-frontier-in-subset*:
assumes *A* ⊆ *B*
assumes *x* ∈ *frontier B*
assumes *x* ∈ *A*
shows *x* ∈ *frontier A*
by (*metis assms frontier-int-subset IntI in-mono*)

lemma *in-frontier-in-subset-convex-hull*:
assumes *A* ⊆ *B*
assumes *x* ∈ *frontier* (*convex hull B*)

assumes $x \in$ *convex hull A*
shows $x \in$ *frontier* (*convex hull A*)
by (*metis in-frontier-in-subset assms hull-mono*)

lemma *convex-hull-two-extreme-points*:
fixes $S :: 'a::$*euclidean-space set*
assumes *finite S*
assumes *convex hull* $S \neq \{\}$
assumes $\forall x.$ *convex hull* $S \neq \{x\}$
shows *card* $\{x.$ *x extreme-point-of* (*convex hull S*)$\} \geq 2$ (**is** *card ?ep* ≥ 2)
proof−
have *compact* (*convex hull S*) **by** (*simp add*: *assms*(*1*) *finite-imp-compact-convex-hull*)
then have *convex hull S* = *convex hull ?ep*
using *Krein-Milman-Minkowski*[*OF - convex-convex-hull*] **by** *blast*
moreover then obtain *x* **where** $x \in$ *?ep* **using** *assms*(*2*) **by** *fastforce*
moreover have *?ep* $\neq \{x\}$ **using** *assms*(*3*) *calculation*(*1*) **by** *force*
ultimately obtain *y* **where** $x \in$ *?ep* $\wedge$ $y \in$ *?ep* $\wedge$ $x \neq y$ **by** *blast*
moreover have *finite ?ep* **using** *assms*(*1*) *extreme-points-of-convex-hull finite-subset*
by *blast*
ultimately show *?thesis*
by (*metis* (*no-types*, *lifting*) *One-nat-def Orderings.order-eq-iff Suc-1 Suc-leI*
card-1-singletonE card-gt-0-iff empty-iff insert-Diff not-less-eq-eq singleton-insert-inj-eq)
qed

lemma *convex-hull-two-vts-on-frontier*:
fixes $S :: 'a::$*euclidean-space set*
assumes *card S* ≥ 2
shows *card* (*S* $\cap$ *frontier* (*convex hull S*)) ≥ 2
proof−
have *S* $\subseteq$ *convex hull S* **by** (*simp add*: *hull-subset*)
then have *convex hull S* $\neq \{\} \wedge$ *card* (*convex hull S*) $\neq 1$
by (*metis Suc-1 add-leD2 assms card.empty card-1-singletonE convex-hull-eq-empty*
not-one-le-zero numeral-le-one-iff plus-1-eq-Suc semiring-norm(*69*) *subset-singletonD*)
moreover have *finite S* **using** *assms* **by** (*metis Suc-1 Suc-leD card-eq-0-iff*
not-one-le-zero)
ultimately have *card* $\{x.$ *x extreme-point-of* (*convex hull S*)$\} \geq 2$
using *convex-hull-two-extreme-points* **by** *fastforce*
moreover have $\{x.$ *x extreme-point-of* (*convex hull S*)$\} \subseteq$ *S* $\cap$ *frontier* (*convex*
hull S)
proof−
have $\{x.$ *x extreme-point-of* (*convex hull S*)$\} \subseteq$ *S* **by** (*simp add*: *extreme-points-of-convex-hull*)
moreover have $\{x.$ *x extreme-point-of* (*convex hull S*)$\}$ $\cap$ *interior* (*convex hull*
S) = $\{\}$
using *extreme-point-not-in-interior* **by** *blast*
moreover have $\{x.$ *x extreme-point-of* (*convex hull S*)$\} \subseteq$ *convex hull S*
using ‹*S* $\subseteq$ *convex hull S*› *calculation*(*1*) **by** *blast*
moreover have *convex hull S* = *interior* (*convex hull S*) $\cup$ *frontier* (*convex*
hull S)
by (*metis* (*no-types*, *lifting*) *Diff-empty Suc-1 assms card.infinite closure-Un-frontier*

closure-convex-hull convex-closure-interior convex-convex-hull empty-subsetI finite-imp-compact frontier-def interior-interior not-less-eq-eq sup-absorb2 zero-less-one-class.zero-le-one)
ultimately show *?thesis* **by** *blast*
qed
ultimately show *?thesis*
by (*smt* (*verit*, *del-insts*) *assms extreme-points-of-convex-hull card-gt-0-iff finite-Int linorder-not-less not-numeral-le-zero order-less-le order-less-le-trans psubset-card-mono*)
qed

18 Vertices on Convex Frontier Implies Polygon is Convex

lemma *convex-cut-aux*:
assumes $\forall\, v \in S.\ z \cdot v \leq 0$
shows *convex hull S* $\subseteq \{x.\ z \cdot x \leq 0\}$
by (*simp add*: *assms convex-halfspace-le hull-minimal subsetI*)

lemma *convex-cut-aux'*:
assumes $\forall\, v \in S.\ z \cdot v \geq 0$
shows *convex hull S* $\subseteq \{x.\ z \cdot x \geq 0\}$
using *convex-cut-aux*[*of S* $-z$] *assms* **by** *auto*

lemma *convex-cut*:
assumes $z \neq 0$
assumes $\{x.\ z \cdot x = 0\} \cap$ *interior* (*convex hull S*) $\neq \{\}$
obtains *v1 v2* **where** $v1 \neq v2 \wedge \{v1, v2\} \subseteq S \wedge v1 \in \{x.\ z \cdot x < 0\} \wedge v2 \in \{x.\ z \cdot x > 0\}$
proof−
let *?P1* $= \{x.\ z \cdot x \leq 0\}$
let *?P2* $= \{x.\ z \cdot x \geq 0\}$
have *frontier ?P1* $= \{x.\ z \cdot x = 0\}$
by (*simp add*: *assms*(*1*) *frontier-halfspace-le*)
moreover have *frontier ?P2* $= \{x.\ z \cdot x = 0\}$
by (*simp add*: *assms*(*1*) *frontier-halfspace-ge*)
ultimately have $\neg$ *convex hull S* $\subseteq$ *?P1* $\wedge \neg$ *convex hull S* $\subseteq$ *?P2*
by (*smt* (*verit*, *ccfv-SIG*) *DiffE IntE assms*(*2*) *disjoint-iff frontier-def inf.absorb-iff2 interior-Int*)
moreover have $(\forall\, v \in S.\ z \cdot v \leq 0) \Longrightarrow$ *convex hull S* $\subseteq$ *?P1* **using** *convex-cut-aux* **by** *blast*
moreover have $(\forall\, v \in S.\ z \cdot v \geq 0) \Longrightarrow$ *convex hull S* $\subseteq$ *?P2* **using** *convex-cut-aux'* **by** *blast*
ultimately obtain *v1 v2* **where** $\{v1, v2\} \subseteq S \wedge z \cdot v1 < 0 \wedge z \cdot v2 > 0$
using *linorder-not-le* **by** *auto*
thus *?thesis* **using** *that* **by** *fastforce*
qed

lemma *affine-2-int-convex*:

fixes $S :: 'a::euclidean\text{-}space\ set$
assumes $\{a, b\} \subseteq S$
assumes $\{a, b\} \subseteq frontier\ (convex\ hull\ S)$
assumes $affine\ hull\ \{a, b\} \cap interior\ (convex\ hull\ S) \neq \{\}$
shows $affine\ hull\ \{a, b\} \cap convex\ hull\ S = convex\ hull\ \{a, b\}$
proof−
let $?H = convex\ hull\ S$
let $?L = affine\ hull\ \{a, b\} \cap ?H$
have 1: $?L \supseteq convex\ hull\ \{a, b\}$
by (*meson Int-greatest assms*(*1*) *convex-hull-subset-affine-hull hull-mono*)
moreover have $?L \subseteq convex\ hull\ \{a, b\}$
proof(*rule subsetI*)
fix x
assume $*$: $x \in ?L$
then obtain $u\ v$ **where** uv: $x = u *_R a + v *_R b \wedge u + v = 1$ **using** *affine-hull-2* **by** *blast*

have *rel-interior ?L* $\subseteq$ *rel-interior ?H*
using *subset-rel-interior-convex*[*of ?L ?H*]
by (*metis assms*(*3*) *convex-affine-hull convex-convex-hull convex-rel-interior-inter-two inf-bot-right inf-le2 rel-interior-affine-hull rel-interior-nonempty-interior*)
moreover have *ab-frontier*: $a \in frontier\ ?H \wedge b \in frontier\ ?H$ **using** *assms* **by** *blast*
ultimately have *ab-rel-frontier*: $a \in rel\text{-}frontier\ ?L \wedge b \in rel\text{-}frontier\ ?L$
by (*metis IntI affine-affine-hull assms*(*3*) *convex-affine-rel-frontier-Int convex-convex-hull hull-subset inf-commute insert-subset*)

{ **assume** $**$: $u < 0$
then have $b \in open\text{-}segment\ a\ x$
proof−
from uv **have** $b = (1/v) *_R x - (u/v) *_R a$
by (*smt* (*verit*, *ccfv-threshold*) $**$ *divide-inverse-commute inverse-eq-divide real-vector-affinity-eq vector-space-assms*(*3*) *Groups.add-ac*(*2*))
moreover from uv **have** $1/v - u/v = 1$
by (*metis* $**$ *add.commute add-cancel-right-left diff-divide-distrib divide-self-if eq-diff-eq*′ *not-one-less-zero*)
ultimately have $b = (1 - 1/v) *_R a + (1/v) *_R x$ **by** (*simp add*: *diff-eq-eq*)
moreover from uv $**$ **have** $0 < 1/v \wedge 1/v < 1$ **by** *simp*
ultimately show *?thesis*
by (*metis 1 ab-rel-frontier affine-hull-sing convex-hull-singleton empty-iff equalityI in-segment*(*2*) *inf-le1 insert-absorb rel-frontier-sing scaleR-collapse singletonI*)
qed
then have $b \in rel\text{-}interior\ (convex\ hull\ \{a, x\})$
by (*metis empty-iff open-segment-idem rel-interior-closed-segment segment-convex-hull*)
moreover have $x \in ?H$ **using** $*$ **by** *blast*
ultimately have $b \in interior\ ?H$
by (*smt* (*verit*, *ccfv-threshold*) $*$ *IntD2 Int-empty-right 1 affine-affine-hull*

affine-hull-affine-Int-nonempty-interior affine-hull-convex-hull assms(3) *convex-Int convex-affine-hull convex-convex-hull convex-rel-interior-inter-two hull-hull hull-redundant-eq insert-commute insert-subsetI rel-interior-affine-hull rel-interior-mono rel-interior-nonempty-interior rel-interior-subset subset-hull subset-iff*)
then have *False* **by** (*metis DiffD2 ab-frontier frontier-def*)
} moreover
{ assume $**$: $v < 0$
then have $a \in$ *open-segment b x*
proof−
from *uv* **have** $a = (1/u) *_R x - (v/u) *_R b$
by (*smt* (*verit*, *ccfv-threshold*) $**$ *divide-inverse-commute inverse-eq-divide real-vector-affinity-eq vector-space-assms*(3) *Groups.add-ac*(2))
moreover from *uv* **have** $1/u - v/u = 1$
by (*metis* $**$ *add-cancel-right-left diff-divide-distrib divide-self-if eq-diff-eq*′ *not-one-less-zero*)
ultimately have $a = (1 - 1/u) *_R b + (1/u) *_R x$ **by** (*simp add*: *diff-eq-eq*)
moreover from *uv* $**$ **have** $0 < 1/u \wedge 1/u < 1$ **by** *simp*
ultimately show *?thesis*
by (*metis 1 ab-rel-frontier affine-hull-sing convex-hull-singleton empty-iff equalityI in-segment*(2) *inf-le1 insert-absorb rel-frontier-sing scaleR-collapse singletonI*)
qed
then have $a \in$ *rel-interior* (*convex hull* {*b*, *x*})
by (*metis empty-iff open-segment-idem rel-interior-closed-segment segment-convex-hull*)
moreover have $x \in$ *?H* **using** $*$ **by** *blast*
ultimately have $a \in$ *interior ?H*
by (*smt* (*verit*, *ccfv-threshold*) $*$ *IntD2 Int-empty-right 1 affine-affine-hull affine-hull-affine-Int-nonempty-interior affine-hull-convex-hull assms*(3) *convex-Int convex-affine-hull convex-convex-hull convex-rel-interior-inter-two hull-hull hull-redundant-eq insert-commute insert-subsetI rel-interior-affine-hull rel-interior-mono rel-interior-nonempty-interior rel-interior-subset subset-hull subset-iff*)
then have *False* **by** (*metis DiffD2 ab-frontier frontier-def*)
}
ultimately have $0 \leq u \wedge u \leq 1 \wedge 0 \leq v \wedge v \leq 1$ **using** *uv* **by** *argo*
thus $x \in$ *convex hull* {*a*, *b*} **by** (*simp add*: *convexD hull-inc uv*)
qed
ultimately show *?thesis* **by** *blast*
qed

lemma *halfplane-frontier-affine-hull*:
fixes *b v* :: *real*^2
assumes $b \neq 0$
assumes $v \neq 0$
assumes $b \in \{x.\ v \cdot x = 0\}$
shows $\{x.\ v \cdot x = 0\} =$ *affine hull* {0, *b*}
proof−
let *?F* = $\{x.\ v \cdot x = 0\}$
let *?A* = *affine hull* {0, *b*}

have *?F* ⊆ *?A*
proof(*rule subsetI*)
fix *y*
assume ∗: *y* ∈ *?F*
have *y* ∈ *?A* **if** *y* = *0* **by** (*simp add*: *assms*(*2*) *hull-inc that*)
moreover have *y* ∈ *?A* **if** *b$1* ≠ *0*
proof−
have *v* · *y* = *0* **using** ∗ **by** *fast*
moreover have *v* · *b* = *0* **using** *assms* **by** *force*
moreover have *v* · *y* = *v$1* ∗ *y$1* + *v$2* ∗ *y$2* **by** (*simp add*: *inner-vec-def sum-2 real-2-inner*)
moreover have *v* · *b* = *v$1* ∗ *b$1* + *v$2* ∗ *b$2* **by** (*simp add*: *inner-vec-def sum-2 real-2-inner*)
ultimately have *0*: *v$1* ∗ *y$1* + *v$2* ∗ *y$2* = *0* ∧ *0* = *v$1* ∗ *b$1* + *v$2* ∗ *b$2* **by** *auto*
moreover obtain *c* **where** *c*: *y$1* = *c* ∗ *b$1* **using** ‹*b$1* ≠ *0*›
by (*metis hyperplane-eq-Ex inner-real-def mult.commute*)
ultimately have *v$1* ∗ *y$1* + *v$2* ∗ *y$2* = *0* ∧ *0* = *c* ∗ *v$1* ∗ *b$1* + *c* ∗ *v$2* ∗ *b$2* **by** *algebra*
then have *v$1* ∗ *y$1* + *v$2* ∗ *y$2* = *v$1* ∗ *y$1* + *c* ∗ *v$2* ∗ *b$2* **using** *c* **by** *algebra*
then have *v$2* ∗ *y$2* = *c* ∗ *v$2* ∗ *b$2* **by** *argo*
then have *y$2* = *c* ∗ *b$2*
by (*smt* (*verit*, *ccfv-threshold*) *0 exhaust-2 mult.commute mult.left-commute mult-cancel-left that assms vec-eq-iff zero-index*)
then have *y* = *c* ∗$_R$ *b* **using** *c*
by (*smt* (*verit*) *exhaust-2 real-scaleR-def vec-eq-iff vector-scaleR-component*)
then have *y* ∈ *span* {*0*, *b*} **by** (*meson insert-subset span-mul span-superset*)
thus *y* ∈ *?A*
by (*simp add*: *affine-hull-span-0 assms*(*2*) *hull-inc*)
qed
moreover have *y* ∈ *?A* **if** *b$2* ≠ *0*
proof−
have *v* · *y* = *0* **using** ∗ **by** *fast*
moreover have *v* · *b* = *0* **using** *assms* **by** *force*
moreover have *v* · *y* = *v$1* ∗ *y$1* + *v$2* ∗ *y$2* **by** (*simp add*: *inner-vec-def sum-2 real-2-inner*)
moreover have *v* · *b* = *v$1* ∗ *b$1* + *v$2* ∗ *b$2* **by** (*simp add*: *inner-vec-def sum-2 real-2-inner*)
ultimately have *0*: *v$1* ∗ *y$1* + *v$2* ∗ *y$2* = *0* ∧ *0* = *v$1* ∗ *b$1* + *v$2* ∗ *b$2* **by** *auto*
moreover obtain *c* **where** *c*: *y$2* = *c* ∗ *b$2* **using** ‹*b$2* ≠ *0*›
by (*metis hyperplane-eq-Ex inner-real-def mult.commute*)
ultimately have *v$1* ∗ *y$1* + *v$2* ∗ *y$2* = *0* ∧ *0* = *c* ∗ *v$1* ∗ *b$1* + *c* ∗ *v$2* ∗ *b$2* **by** *algebra*
then have *v$1* ∗ *y$1* + *v$2* ∗ *y$2* = *0* ∧ *0* = *c* ∗ *v$1* ∗ *b$1* + *v$2* ∗ *y$2* **using** *c* **by** *algebra*
then have *v$1* ∗ *y$1* = *c* ∗ *v$1* ∗ *b$1* **by** *argo*
then have *y$1* = *c* ∗ *b$1*

by (*smt* (*verit*, *ccfv-threshold*) *0 exhaust-2 mult.commute mult.left-commute mult-cancel-left that assms vec-eq-iff zero-index*)
then have $y = c *_R b$ **using** *c*
by (*smt* (*verit*) *exhaust-2 real-scaleR-def vec-eq-iff vector-scaleR-component*)
then have $y \in span \{0, b\}$ **by** (*meson insert-subset span-mul span-superset*)
thus $y \in ?A$
by (*simp add*: *affine-hull-span-0 assms*(*2*) *hull-inc*)
qed
ultimately show $y \in ?A$
by (*metis* (*mono-tags*, *opaque-lifting*) *assms*(*1*) *exhaust-2 vec-eq-iff zero-index*)
qed
moreover have $?A \subseteq ?F$
proof(*rule subsetI*)
fix *x*
assume $x \in ?A$
then obtain $\alpha\ \beta$ **where** $x = \alpha *_R 0 + \beta *_R b \wedge \alpha + \beta = 1$ **using** *affine-hull-2*
by *blast*
then have $v \cdot x = \alpha * (v \cdot 0) + \beta * (v \cdot b)$ **by** (*simp add*: *assms*(*1*))
then have $v \cdot x = 0$ **using** *assms*(*3*) **by** *auto*
thus $x \in ?F$ **by** *fast*
qed
ultimately show *?thesis* **by** *blast*
qed

lemma *vts-on-convex-frontier-aux*:
assumes *polygon-of p vts*
assumes $vts!0 = 0$
assumes *set vts* $\subseteq$ *frontier* (*convex hull* (*set vts*))
shows *path-image* (*linepath* (*vts*!*0*) (*vts*!*1*)) $\subseteq$ *frontier* (*convex hull* (*set vts*))
proof−
let $?H = convex\ hull\ (set\ vts)$
let $?a = vts!0$
let $?b = vts!1$
let $?l = linepath\ ?a\ ?b$
let $?L = path\text{-}image\ ?l$
let $?A = affine\ hull\ \{?a, ?b\}$
let $?x = ?b - ?a$

obtain *v* **where** *v*: $v \cdot ?x = 0 \wedge v \neq 0$
proof−
let $?v = (vector\ [?x\$2, -?x\$1])::(real\hat{}2)$
have $?a \neq ?b$
by (*smt* (*verit*, *best*) *Cons-nth-drop-Suc One-nat-def Suc-le-eq arc-distinct-ends assms*(*1*) *assms*(*2*) *card.empty drop0 empty-set length-greater-0-conv list.sel*(*1*) *list.sel*(*3*) *make-polygonal-path.elims make-polygonal-path.simps*(*1*) *make-polygonal-path.simps*(*2*) *nth-drop pathfinish-linepath pathstart-linepath plus-1-eq-Suc polygon-at-least-3-vertices polygon-def polygon-of-def polygon-pathstart rel-simps*(*28*) *simple-path-joinE*)
then have $?x \neq 0$ **by** *simp*
then have $?v \cdot ?x = 0 \wedge ?v \neq 0$

```
    proof −
      have ?v · ?x = (?x$2 * ?x$1) + (− ?x$1 * ?x$2)
        by (simp add: inner-vec-def sum-2 real-2-inner)
      then have ?v · ?x = 0 by argo
      moreover have ?v ≠ 0
        by (smt (verit, best) ‹?x ≠ 0› exhaust-2 vec-eq-iff vector-2(1) vector-2(2)
zero-index)
      ultimately show ?thesis by blast
    qed
    thus ?thesis using that by blast
  qed

  let ?P1 = {x. v · x ≤ 0}
  let ?P2 = {x. v · x ≥ 0}
  let ?P1-int = {x. v · x < 0}
  let ?P2-int = {x. v · x > 0}
  let ?F = {x. v · x = 0}

  have ?b ≠ 0
   by (smt (verit) Cons-nth-drop-Suc One-nat-def Suc-le-eq Suc-le-length-iff arc-distinct-ends
assms(1) assms(2) card.empty drop0 drop-eq-Nil empty-set le-numeral-extra(4)
length-greater-0-conv list.inject make-polygonal-path.elims make-polygonal-path.simps(2)
nat-less-le pathfinish-linepath pathstart-linepath polygon-at-least-3-vertices polygon-def
polygon-of-def polygon-pathstart rel-simps(28) simple-path-joinE)
  moreover have ?b ∈ ?F using assms(2) v by auto
  ultimately have F: ?F = ?A
    using halfplane-frontier-affine-hull[of ?b v] v assms(2) by presburger
 moreover have ?L ⊆ ?A by (simp add: convex-hull-subset-affine-hull segment-convex-hull)
  ultimately have L-subset-F: ?L ⊆ ?F by blast
  have L-subset-H: ?L ⊆ ?H
   by (metis (no-types, lifting) add-gr-0 assms(1) card.empty convex-contains-segment
convex-convex-hull diff-less empty-set hull-subset leD length-greater-0-conv less-numeral-extra(1)
nth-mem numeral-3-eq-3 path-image-linepath plus-1-eq-Suc polygon-at-least-3-vertices
polygon-of-def rotate-polygon-vertices-same-set rotated-polygon-vertices-helper(2) sub-
set-code(1))

  have frontier-P1: frontier ?P1 = ?F by (simp add: v frontier-halfspace-le)
  have frontier-P2: frontier ?P2 = ?F by (simp add: v frontier-halfspace-ge)
  have interior-P1: interior ?P1 = ?P1-int by (simp add: v)
  have interior-P2: interior ?P2 = ?P2-int by (simp add: v)
  have convex-P1: convex ?P1 by (simp add: convex-halfspace-le)
  have convex-P2: convex ?P2 by (simp add: convex-halfspace-ge)
  have P1-int-P2: ?P1 ∩ ?P2 = ?F by (simp add: halfspace-Int-eq(1))

  let ?H1 = ?H ∩ ?P1
  let ?H2 = ?H ∩ ?P2

  have ¬ collinear (set vts) using polygon-vts-not-collinear assms(1) by simp
  then have nonempty-interior-H: interior ?H ≠ {}
```

by (*smt* (*verit*, *ccfv-SIG*) *Jordan-inside-outside-real2 closed-path-def Un-Int-eq*(*4*) *assms*(*1*) *convex-hull-of-polygon-is-convex-hull-of-vts disjoint-iff hull-subset inf.orderE interior-Int interior-eq interior-subset path-inside-def polygon-def polygon-of-def*)

have *convex-H1*: *convex ?H1* **by** (*simp add*: *convex-Int convex-P1*)
have *convex-H2*: *convex ?H2* **by** (*simp add*: *convex-Int convex-P2*)

have *?H* $\subseteq$ *?P1* $\vee$ *?H* $\subseteq$ *?P2*
proof(*rule ccontr*)
assume $*$: $\neg$ (*?H* $\subseteq$ *?P1* $\vee$ *?H* $\subseteq$ *?P2*)
moreover have *interior ?H* $\subseteq$ *?P1* $\Longrightarrow$ *?H* $\subseteq$ *?P1*
by (*metis* (*no-types*, *lifting*) *Int-Un-eq*(*3*) *Krein-Milman-frontier List.finite-set P1-int-P2 closure-Un-frontier closure-convex-hull closure-mono compact-frontier convex-closure-interior convex-convex-hull finite-imp-compact-convex-hull frontier-P1 nonempty-interior-H*)
moreover have *interior ?H* $\subseteq$ *?P2* $\Longrightarrow$ *?H* $\subseteq$ *?P2*
by (*metis* (*no-types*, *lifting*) *Int-Un-eq*(*3*) *Krein-Milman-frontier List.finite-set P1-int-P2 calculation*(*1*) *calculation*(*2*) *closure-Un-frontier closure-convex-hull closure-mono compact-frontier convex-closure-interior convex-convex-hull emptyE finite-imp-compact-convex-hull frontier-P2 inf-commute subsetI*)
ultimately have *interior ?H* $\cap$ *?P1* $\neq$ {} $\wedge$ *interior ?H* $\cap$ $-$*?P1* $\neq$ {} **by** *force*
moreover have *path-connected* (*interior ?H*) **by** (*simp add*: *convex-imp-path-connected*)
ultimately have *F-int-interior-H*: *?F* $\cap$ *interior ?H* $\neq$ {}
by (*metis* (*no-types*, *lifting*) *path-connected-frontier ComplD disjoint-eq-subset-Compl frontier-P1 subset-eq*)
then obtain *v1 v2* **where** *v1v2*: *v1* $\neq$ *v2* $\wedge$ {*v1*, *v2*} $\subseteq$ *set vts*
$\wedge$ *v1* $\in$ *interior ?P1* $\wedge$ *v2* $\in$ *interior ?P2*
using *convex-cut frontier-P1 interior-P1 interior-P2 v* **by** *metis*
then obtain *i j* **where** *ij*: *vts*!*i* = *v1* $\wedge$ *vts*!*j* = *v2*
$\wedge$ *2* $\leq$ *i* $\wedge$ *2* $\leq$ *j* $\wedge$ *i* $\neq$ *j* $\wedge$ *i* $<$ *length vts* $-$ *1* $\wedge$ *j* $<$ *length vts* $-$ *1*
proof$-$
obtain *i j* **where** *vts*!*i* = *v1* $\wedge$ *vts*!*j* = *v2* $\wedge$ *i* $\neq$ *j* $\wedge$ *i* $<$ *length vts* $\wedge$ *j* $<$ *length vts*
by (*metis in-set-conv-nth insert-subset v1v2*)
moreover have *2* $\leq$ *i*
proof$-$
{ assume *i* = *0* $\vee$ *i* = *1*
then have *vts*!*i* = *?a* $\vee$ *vts*!*i* = *?b* **by** *blast*
then have *vts*!*i* $\in$ *?F* **by** (*simp add*: *F hull-inc*)
then have *False* **using** *calculation*(*1*) *interior-P1 v1v2* **by** *auto*
}
thus *?thesis* **by** *presburger*
qed
moreover have *2* $\leq$ *j*
proof$-$
{ assume *j* = *0* $\vee$ *j* = *1*
then have *vts*!*j* = *?a* $\vee$ *vts*!*j* = *?b* **by** *blast*
then have *vts*!*j* $\in$ *?F* **by** (*simp add*: *F hull-inc*)

then have *False* **using** *calculation*(*1*) *interior-P2 v1v2* **by** *auto*
}
thus *?thesis* **by** *presburger*
qed
moreover have *False* **if** $i = length\ vts - 1$
by (*metis* (*no-types*, *lifting*) *F assms*(*1*) *calculation*(*1*) *frontier-P1 frontier-def have-wraparound-vertex hull-subset insertCI insert-Diff last-conv-nth last-snoc less-nat-zero-code list.size*(*3*) *polygon-of-def subset-Diff-insert that v1v2*)
moreover have *False* **if** $j = length\ vts - 1$
by (*metis* (*no-types*, *lifting*) *F assms*(*1*) *calculation*(*1*) *frontier-P2 frontier-def have-wraparound-vertex hull-subset insertCI insert-Diff last-conv-nth last-snoc less-nat-zero-code list.size*(*3*) *polygon-of-def subset-Diff-insert that v1v2*)
ultimately show *?thesis* **using** *that* **by** *fastforce*
qed

let $?i' = min\ i\ j$
let $?j' = max\ i\ j$
let $?vts' = take\ (?j' - ?i' + 1)\ (drop\ ?i'\ vts)$
let $?p' = make\text{-}polygonal\text{-}path\ ?vts'$
have *vts′-sublist*: *sublist ?vts′ vts* **using** *sublist-order.order.trans* **by** *blast*
then have *vts′-sublist-tl*: *sublist ?vts′* (*tl vts*)
by (*metis Suc-1 Suc-eq-plus1 drop-Suc ij max-def min-def nat-minus-add-max not-less-eq-eq sublist-drop sublist-order.dual-order.trans sublist-take*)

have *p′-start-finish*: $\{pathstart\ ?p', pathfinish\ ?p'\} = \{v1, v2\}$
proof−
have $?vts'!0 = vts!?i'$ **using** *ij* **by** *force*
moreover have $?vts'!(?j' - ?i') = vts!?j'$
using *diff-is-0-eq diff-zero ij less-numeral-extra*(*1*) *max.cobounded1 min-absorb2 min-def nth-drop nth-take order-less-imp-le*
by *fastforce*
moreover have $(vts!?i' = v1 \wedge vts!?j' = v2) \vee (vts!?i' = v2 \wedge vts!?j' = v1)$
using *ij* **by** *linarith*
moreover have $pathstart\ ?p' = ?vts'!0 \wedge pathfinish\ ?p' = ?vts'!(?j' - ?i')$
using *ij min-diff polygon-pathfinish polygon-pathstart*
by (*smt* (*verit*, *ccfv-SIG*) *add-diff-cancel-right′ add-diff-inverse-nat length-drop length-take less-diff-conv max.commute max-min-same*(*1*) *min.absorb4 nat-minus-add-max not-add-less2 plus-1-eq-Suc plus-nat.simps*(*2*) *take-eq-Nil zero-less-one*)
ultimately show *?thesis* **by** *auto*
qed
then have $path\text{-}image\ ?p' \cap interior\ ?P2 \neq \{\} \wedge path\text{-}image\ ?p' \cap interior\ ?P1 \neq \{\}$
by (*metis v1v2 IntI doubleton-eq-iff empty-iff pathfinish-in-path-image path-start-in-path-image*)
then have $path\text{-}image\ ?p' \cap -?P1 \neq \{\} \wedge path\text{-}image\ ?p' \cap ?P1 \neq \{\}$
using *interior-P2*
by (*smt* (*verit*, *best*) *disjoint-iff-not-equal in-mono inf-shunt interior-P1 mem-Collect-eq*)
moreover have *path-connected* (*path-image ?p′*)

using *make-polygonal-path-gives-path path-connected-path-image* **by** *blast*
ultimately obtain z **where** z: $z \in$ *path-image ?p'* $\cap$ *?F*
by (*smt* (*verit*, *del-insts*) *path-connected-frontier DiffE Diff-triv all-not-in-conv frontier-P1*)
moreover have *path-image ?p'* $\subseteq$ *?H*
proof −
have *path-image p* $\subseteq$ *?H*
by (*metis assms*(*1*) *insert-subset length-pos-if-in-set polygon-of-def polygon-path-image-subset-convex v1v2*)
moreover have *path-image ?p'* $\subseteq$ *path-image p*
by (*metis* (*no-types*, *lifting*) *vts'-sublist sublist-path-image-subset One-nat-def Suc-leI p'-start-finish assms*(*1*) *doubleton-eq-iff length-greater-0-conv make-polygonal-path.simps*(*1*) *pathfinish-linepath pathstart-linepath polygon-of-def v1v2*)
ultimately show *?thesis* **by** *blast*
qed
ultimately have $z \in$ *path-image ?p'* $\cap$ (*?H* $\cap$ *?F*) **by** *blast*
moreover have *?H* $\cap$ *?F* = *?L*
using *affine-2-int-convex*[*of ?a ?b set vts*]
by (*smt* (*verit*, *best*) *assms*(*3*) *F F-int-interior-H inf-commute segment-convex-hull path-image-linepath Suc-1 add-leD2 assms*(*1*) *empty-subsetI insert-subset length-greater-0-conv lessI nat-neq-iff nth-mem numeral-Bit0 order.strict-iff-not plus-1-eq-Suc polygon-of-def polygon-vertices-length-at-least-4 take-all-iff take-eq-Nil IntE inf.orderE*)
ultimately have $z \in$ *?L* $\cap$ *path-image ?p'* **by** *blast*
moreover have *?L* $\cap$ *path-image ?p'* $\subseteq$ {*?a*, *?b*}
proof −
let *?p-tl* = *make-polygonal-path* (*tl vts*)
have *p* = *make-polygonal-path vts* $\wedge$ *loop-free p*
using *assms* **unfolding** *polygon-of-def polygon-def simple-path-def* **by** *blast*
moreover have [*?a*, *?b*] = *take 2 vts*
by (*metis Cons-nth-drop-Suc One-nat-def Suc-1 append-Cons append-Nil calculation constant-linepath-is-not-loop-free drop0 drop-eq-Nil insert-subset length-pos-if-in-set linorder-not-le make-polygonal-path.simps*(*2*) *take0 take-Suc-conv-app-nth v1v2*)
moreover have *tl vts* = *drop* (*2* − *1*) *vts* **by** (*simp add*: *drop-Suc*)
moreover have *?l* = *make-polygonal-path* [*?a*, *?b*] **using** *make-polygonal-path.simps* **by** *simp*
moreover have *length vts* > *2* **using** *ij* **by** *linarith*
moreover have *pathstart ?l* = *?a* $\wedge$ *pathstart ?p-tl* = *?b*
using *calculation*(*3*) *calculation*(*5*) *polygon-pathstart* **by** *auto*
ultimately have *?L* $\cap$ *path-image ?p-tl* $\subseteq$ {*?a*, *?b*}
using *loop-free-split-int*[*of p vts* [*?a*, *?b*] *2 tl vts ?l ?p-tl length vts*] **by** *auto*
moreover have *path-image ?p'* $\subseteq$ *path-image ?p-tl*
using *sublist-path-image-subset*
by (*metis add.commute ij le-add2 length-drop length-take less-diff-conv min.absorb4 min.cobounded1 min-def vts'-sublist-tl*)
ultimately show *?thesis* **by** *blast*
qed
ultimately have ∗: z = *?a* $\vee$ z = *?b* **by** *blast*

let *?*i = *?i'*

let *?𝔧 = ?j′ − ?i′ + 1*
let *?𝔨 = ?𝔦 + ?𝔧*
let *?x1 = (2^?𝔦 − 1)/(2^?𝔦)::real*
let *?x2 = (2^(?𝔨−1) − 1)/(2^(?𝔨−1))::real*

have *?vts′ = take ?𝔧 (drop ?𝔦 vts)* **by** *blast*
moreover have *?𝔨 ≤ length vts − 1 ∧ 2 ≤ ?𝔧* **using** *ij* **by** *linarith*
ultimately have *path-image ?p′ = p‘{?x1..?x2}*
using *vts-sublist-path-image assms(1)* **unfolding** *polygon-of-def* **by** *metis*
moreover have *x1x2*: *?x1 > 1/2 ∧ ?x2 < 1*
proof−
have *?i′ ≥ 2* **using** *ij* **by** *linarith*
then have *(1::real) < 2^?i′ − 1*
by (*smt* (*z3*) *dual-order.strict-trans1 linorder-le-less-linear numeral-le-one-iff power-one-right power-strict-increasing semiring-norm*(*69*))
thus *?thesis* **by** *simp*
qed
moreover have *p 0 ∉ p‘{?x1..?x2} ∧ p (1/2) ∉ p‘{?x1..?x2}*
proof−
have *False* **if** ∗: *p 0 ∈ p‘{?x1..?x2}*
proof−
obtain *t* **where** *t*: *t ∈ {?x1..?x2} ∧ p t = p 0* **using** ∗ **by** *auto*
then have *t ≥ ?x1 ∧ t ≤ ?x2* **by** *presburger*
then have *1/2 < t ∧ t < 1* **using** *x1x2* **by** *argo*
thus *False*
using *t assms*(*1*) **unfolding** *polygon-of-def polygon-def simple-path-def loop-free-def*
by *force*
qed
moreover have *False* **if** ∗: *p (1/2) ∈ p‘{?x1..?x2}*
proof−
obtain *t* **where** *t*: *t ∈ {?x1..?x2} ∧ p t = p (1/2)* **using** ∗ **by** *auto*
then have *t ≥ ?x1 ∧ t ≤ ?x2* **by** *presburger*
then have *1/2 < t ∧ t < 1* **using** *x1x2* **by** *argo*
thus *False*
using *t assms*(*1*) **unfolding** *polygon-of-def polygon-def simple-path-def loop-free-def*
by *fastforce*
qed
ultimately show *?thesis* **by** *fast*
qed
moreover have *?a = p 0*
by (*metis assms*(*1*) *card.empty empty-set not-numeral-le-zero pathstart-def polygon-at-least-3-vertices polygon-of-def polygon-pathstart*)
moreover have *?b = p (1/2)*
proof−
have *p = ?l +++ (make-polygonal-path (tl vts))*
by (*smt* (*verit*, *best*) *One-nat-def Suc-1 assms*(*1*) *ij length-Cons length-greater-0-conv length-tl less-imp-le-nat list.sel*(*3*) *list.size*(*3*) *make-polygonal-path.elims nth-Cons-0*

nth-tl order-less-le-trans polygon-of-def pos2 zero-less-diff)
    then have *p* (*1*/*2*) = *?l 1*
      unfolding *joinpaths-def* **by** *simp*
    thus *?thesis* **by** (*simp add*: *linepath-1'*)
  qed
  ultimately have *?a* ∉ *path-image ?p'* ∧ *?b* ∉ *path-image ?p'* **by** *presburger*
  thus *False* **using** *z* ∗ **by** *blast*
 qed
 then have *frontier ?P1* ∩ *?H* ⊆ *frontier ?H* ∨ *frontier ?P2* ∩ *?H* ⊆ *frontier ?H*
  using *frontier-int-subset* **by** *auto*
 moreover have *?L* ⊆ *frontier ?P1* ∧ *?L* ⊆ *frontier ?P2*
  using *frontier-P1 frontier-P2 L-subset-F* **by** *presburger*
 ultimately show *?thesis* **using** *L-subset-H* **by** *fast*
qed

lemma *vts-on-convex-frontier-aux'*:
 assumes *polygon-of p vts*
 assumes *set vts* ⊆ *frontier* (*convex hull* (*set vts*))
 shows *path-image* (*linepath* (*vts*!*0*) (*vts*!*1*)) ⊆ *frontier* (*convex hull* (*set vts*))
proof−
 let *?a* = *vts*!*0*
 let *?f* = λ*v*. *v* + (−*?a*)
 let *?vts'* = *map ?f vts*
 let *?p'* = *make-polygonal-path ?vts'*

 have *len-vts*: *length vts* ≥ *2*
  using *assms*(*1*) *polygon-of-def polygon-vertices-length-at-least-4* **by** *fastforce*
 then have *p'*: *?p'* = *?f* ∘ *p*
  using *make-polygonal-path-translate*[*of vts* −*?a*] *assms* **unfolding** *polygon-of-def*
by *presburger*
 then have *0*: *?vts'*!*0* = *0*
  by (*metis len-vts neg-eq-iff-add-eq-0 nth-map order-less-le-trans pos2*)
 moreover have *vts'*: *set ?vts'* = *?f* ‘ (*set vts*) **by** *simp*
 ultimately have *convex hull* (*set ?vts'*) = *?f* ‘ (*convex hull* (*set vts*))
  using *convex-hull-translation*[*of* −*?a set vts*] **by** *force*
 then have *frontier* (*convex hull* (*set ?vts'*)) = *frontier* (*?f* ‘ (*convex hull* (*set*
vts)))
  by *auto*
 then have *frontier-translation*:
   frontier (*convex hull* (*set ?vts'*)) = *?f* ‘ (*frontier* ((*convex hull* (*set vts*))))
  using *frontier-translation*[*of* −*?a convex hull* (*set vts*)] **by** *simp*

 have *?f* (*vts*!*0*) = *?vts'*!*0* ∧ *?f* (*vts*!*1*) = *?vts'*!*1* **using** *0 len-vts* **by** *auto*
 then have *linepath-translation*:
     ?f ‘ *path-image* (*linepath* (*vts*!*0*) (*vts*!*1*)) = *path-image* (*linepath* (*?vts'*!*0*)
(*?vts'*!*1*))
  using *linepath-translation*[*of ?a* −*?a vts*!*1*] **by** (*simp add*: *path-image-compose*)

 have *polygon-of ?p' ?vts'* **using** *translation-is-polygon assms*(*1*) *p'* **by** *presburger*

moreover have *set ?vts' ⊆ frontier (convex hull (set ?vts'))*
proof−
have *frontier (convex hull (set ?vts')) = frontier (convex hull (?f ‘ (set vts)))*
using *vts'* **by** *presburger*
then have *frontier (convex hull (set ?vts')) = ?f ‘ (frontier (convex hull (set vts)))*
using *frontier-translation* **by** *presburger*
thus *?thesis* **using** *vts' assms(2)* **by** *auto*
qed
ultimately have *path-image (linepath (?vts'!0) (?vts'!1)) ⊆ frontier (convex hull (set ?vts'))*
using *vts-on-convex-frontier-aux assms 0* **by** *blast*
then have *?f ‘ path-image (linepath (vts!0) (vts!1)) ⊆ ?f ‘ (frontier ((convex hull (set vts))))*
using *linepath-translation frontier-translation* **by** *argo*
thus *?thesis* **by** *force*
qed

lemma *vts-on-convex-frontier*:
assumes *polygon-of p vts*
assumes *set vts ⊆ frontier (convex hull (set vts))*
assumes *i < length vts − 1*
shows *path-image (linepath (vts!i) (vts!(i+1))) ⊆ frontier (convex hull (set vts))*
proof−
let *?vts' = rotate-polygon-vertices vts i*
let *?p' = make-polygonal-path ?vts'*
have *polygon-of ?p' ?vts'*
using *assms(1) polygon-of-def rotation-is-polygon* **by** *blast*
moreover have *set ?vts' ⊆ frontier (convex hull (set ?vts'))*
using *assms(1) assms(2) polygon-of-def rotate-polygon-vertices-same-set* **by** *auto*
ultimately have *path-image (linepath (?vts'!0) (?vts'!1)) ⊆ frontier (convex hull (set ?vts'))*
using *vts-on-convex-frontier-aux'* **by** *presburger*
moreover have *?vts'!0 = vts!i ∧ ?vts'!1 = vts!(i+1)*
using *assms(3)*
using *rotated-polygon-vertices[of ?vts' vts i i+1]*
using *rotated-polygon-vertices[of ?vts' vts i i]*
by (*smt* (*verit*, *best*) *Suc-leI add.commute add.right-neutral add-2-eq-Suc' add-diff-cancel-left' add-lessD1 assms(1) have-wraparound-vertex hd-Nil-eq-last hd-conv-nth last-snoc le-add1 less-diff-conv plus-1-eq-Suc polygon-of-def*)
moreover have *frontier (convex hull (set ?vts')) = frontier (convex hull (set vts))*
by (*metis assms(1) polygon-of-def rotate-polygon-vertices-same-set*)
ultimately show *?thesis* **by** *argo*
qed

lemma *vts-on-frontier-means-path-image-on-frontier*:

assumes *polygon-of p vts*
assumes *set vts* ⊆ *frontier* (*convex hull* (*set vts*))
shows *path-image p* ⊆ *frontier* (*convex hull* (*set vts*))
proof(*rule subsetI*)
let *?H* = *convex hull* (*set vts*)
fix *x* **assume** *x* ∈ *path-image p*
moreover have *path-image p* = (⋃ {*path-image* (*linepath* (*vts*!*i*) (*vts*!(*i*+*1*))) |
i. *i* ≤ (*length vts*) − *2*})
using *polygonal-path-image-linepath-union assms* **unfolding** *polygon-of-def*
by (*metis* (*no-types*, *lifting*) *add-leD2 numeral-Bit0 polygon-vertices-length-at-least-4*)
ultimately obtain *i* **where** *i* ≤ (*length vts*) − *2* ∧ *x* ∈ *path-image* (*linepath*
(*vts*!*i*) (*vts*!(*i*+*1*)))
by *blast*
thus *x* ∈ *frontier ?H*
by (*smt* (*verit*, *ccfv-SIG*) *One-nat-def Suc-diff-Suc add.commute add-2-eq-Suc′*
assms(*1*) *assms*(*2*) *in-mono le-add1 le-zero-eq less-Suc-eq-le less-diff-conv linorder-not-less*
plus-1-eq-Suc vts-on-convex-frontier vts-on-convex-frontier-aux′)
qed

lemma *vts-on-convex-frontier-interior*:
assumes *polygon-of p vts*
assumes *set vts* ⊆ *frontier* (*convex hull* (*set vts*))
shows *path-inside p* = *interior* (*convex hull* (*set vts*))
proof−
let *?H* = *convex hull* (*set vts*)

have *path-inside p* ⊆ *interior* (*convex hull* (*set vts*))
by (*metis* (*no-types*, *lifting*) *Un-empty assms*(*1*) *convex-contains-simple-closed-path-imp-contains-path-inside*
convex-convex-hull convex-hull-eq-empty convex-hull-of-polygon-is-convex-hull-of-vts
empty-set inside-outside-def inside-outside-polygon interior-maximal length-greater-0-conv
polygon-def polygon-of-def polygon-path-image-subset-convex)
moreover have *interior* (*convex hull* (*set vts*)) ⊆ *path-inside p*
proof(*rule ccontr*)
assume ∗: ¬ *interior* (*convex hull* (*set vts*)) ⊆ *path-inside p*
then obtain *x* **where** *x*: *x* ∈ *interior* (*convex hull* (*set vts*)) − *path-inside p*
by *blast*
obtain *y* **where** *y*: *y* ∈ *path-inside p*
using *inside-outside-polygon assms* **unfolding** *inside-outside-def polygon-of-def*
by *fastforce*

let *?l* = *linepath x y*
have *1*: *path-image ?l* ⊆ *interior ?H*
by (*metis* (*no-types*, *lifting*) *DiffE calculation convex-contains-segment con-*
vex-convex-hull convex-interior in-mono linepath-image-01 path-defs(*4*) *x y*)
have *path-image ?l* ∩ *frontier* (*path-inside p*) ≠ {}
using *inside-outside-polygon assms* **unfolding** *inside-outside-def polygon-of-def*
by (*smt* (*verit*) ∗ *Diff-disjoint Diff-eq-empty-iff Int-Un-eq*(*2*) *Int-assoc Un-Int-eq*(*3*)
assms(*1*) *calculation connected-Int-frontier convex-connected convex-convex-hull con-*
vex-interior frontier-def inf.absorb-iff2 vts-on-frontier-means-path-image-on-frontier)

then have *2*: *path-image ?l* ∩ *path-image p* ≠ {}
using *inside-outside-polygon assms* **unfolding** *inside-outside-def polygon-of-def*
by *blast*

show *False*
using *1 2 vts-on-frontier-means-path-image-on-frontier*
using *Diff-disjoint Int-lower2 Int-subset-iff assms*(*1*) *assms*(*2*) *frontier-def inf-le1*
by *fastforce*
qed
ultimately show *?thesis* **by** *blast*
qed

lemma *vts-subset-frontier*:
assumes *polygon-of p vts*
assumes *set vts* ⊆ *frontier* (*convex hull* (*set vts*))
shows *convex* (*path-image p* ∪ *path-inside p*)
by (*metis assms*(*1*) *assms*(*2*) *vts-on-convex-frontier-interior convex-convex-hull convex-interior polygon-convex-iff polygon-of-def sup-commute*)

lemma *convex-hull-of-nonconvex-polygon-strict-subset-ep*:
assumes *polygon-of p vts*
assumes ¬ (*convex* (*path-image p* ∪ *path-inside p*))
shows {*v*. *v extreme-point-of* (*convex hull* (*set vts*))} ⊂ *set vts*
proof−
let *?ep* = {*v*. *v extreme-point-of* (*convex hull* (*set vts*))}
let *?H* = *convex hull* (*set vts*)
have *?ep* ⊆ *frontier ?H*
by (*metis Krein-Milman-frontier List.finite-set convex-convex-hull extreme-point-of-convex-hull finite-imp-compact-convex-hull mem-Collect-eq subsetI*)
thus *?thesis* **using** *assms vts-subset-frontier extreme-points-of-convex-hull* **by** *force*
qed

lemma *convex-hull-of-nonconvex-polygon-strict-subset*:
assumes *polygon-of p vts*
assumes ¬ (*convex* (*path-image p* ∪ *path-inside p*))
shows ∃ *v* ∈ *set vts*. *v* ∈ *interior* (*convex hull* (*set vts*))
using *assms vts-subset-frontier*
by (*smt* (*verit*) *Diff-iff UnCI closure-Un-frontier frontier-def hull-inc subsetI*)

lemma *convex-polygon-means-linepaths-inside*:
fixes *p* :: *R-to-R2*
assumes *polygon-of p vts*
assumes *convex-is*: *convex hull* (*set vts*) = (*path-inside p* ∪ *path-image p*)
assumes *a-in*: *a* ∈ (*path-inside p* ∪ *path-image p*)
assumes *b-in*: *b* ∈ (*path-inside p* ∪ *path-image p*)
shows *path-image* (*linepath a b*) ⊆ (*path-inside p* ∪ *path-image p*)
proof −

let *?conv* = *path-inside p* ∪ *path-image p*
have ∀ *u*≥*0*. ∀ *v*≥*0*. *u* + *v* = *1* ⟶ *u* *$_R$ *a* + *v* *$_R$ *b* ∈ *?conv*
using *convex-is a-in b-in* **unfolding** *convex-def*
by (*metis* (*no-types*, *lifting*) *convexD convex-convex-hull convex-is*)
then have (*1* − *x*) *$_R$ *a* + *x* *$_R$ *b* ∈ *?conv* **if** *x-in*: *x* ∈ {*0..1*} **for** *x*
using *x-in* **by** *auto*
then show *?thesis* **unfolding** *linepath-def path-image-def*
by *fast*
qed

end
theory *Polygon-Splitting*
imports
HOL−Analysis.Complete-Measure
Polygon-Jordan-Curve
Polygon-Convex-Lemmas
begin

19 Polygon Splitting

lemma *split-up-a-list-into-3-parts*:
fixes *i j*:: *nat*
assumes *i* < *length vts* ∧ *j* < *length vts* ∧ *i* < *j*
shows
vts = (*take i vts*) @ ((*vts* ! *i*) # ((*take* (*j* − *i* − *1*) (*drop* (*Suc i*) *vts*)) @ (*vts* ! *j*) # *drop* (*j* − *i*) (*drop* (*Suc i*) *vts*)))
proof −
let *?x* = *vts* ! *i*
let *?y* = *vts* ! *j*
let *?vts1* = (*take i vts*)
let *?drop-list* = *drop* (*Suc i*) *vts*
have *vts-is*: *vts* = *?vts1* @ *vts*!*i* # *drop* (*Suc i*) *vts*
using *split-list assms*
by (*meson id-take-nth-drop*)
then have *len-vts1*: *length ?vts1* = *i*
using *length-take*[*of i vts*] *assms*
by *auto*
have *gt-eq*: *j* − *i* − *1* ≥ *0*
using *assms* **by** *auto*
let *?ind* = *j* − *i* − *1*
have *drop-is*: *drop* (*Suc i*) *vts* ! (*j* − *i* − *1*) = *?y*
using *assms* **by** *auto*
then have *drop-list-is*: *?drop-list* = *take ?ind ?drop-list* @ *?y* # (*drop* (*j* − *i*) *?drop-list*)
by (*metis Suc-diff-Suc Suc-leI assms diff-Suc-1 diff-less-mono id-take-nth-drop length-drop*)
have *length* (*drop* (*Suc ?ind*) *?drop-list*) = *length vts* − *j* − *1*
using *length-drop*[*of Suc* (*j* − *i* − *1*) (*drop* (*Suc i*) *vts*)] *length-take assms*
by *auto*

then show *?thesis*
using *vts-is drop-list-is len-vts1*
by *presburger*
qed

definition *is-polygon-cut* :: (*real^2*) *list* ⇒ *real^2* ⇒ *real^2* ⇒ *bool* **where**
is-polygon-cut vts x y =
(*x* ≠ *y* ∧
polygon (*make-polygonal-path vts*) ∧
{*x*, *y*} ⊆ *set vts* ∧
path-image (*linepath x y*) ∩ *path-image* (*make-polygonal-path vts*) = {*x*, *y*} ∧
path-image (*linepath x y*) ∩ *path-inside* (*make-polygonal-path vts*) ≠ {})

definition *is-polygon-cut-path* :: (*real^2*) *list* ⇒ *R-to-R2* ⇒ *bool* **where**
is-polygon-cut-path vts cutpath =
(*let x* = *pathstart cutpath* ; *y* = *pathfinish cutpath in*
(*x* ≠ *y* ∧
polygon (*make-polygonal-path vts*) ∧
{*x*, *y*} ⊆ *set vts* ∧
simple-path cutpath ∧
path-image cutpath ∩ *path-image* (*make-polygonal-path vts*) = {*x*, *y*} ∧
path-image cutpath ∩ *path-inside* (*make-polygonal-path vts*) ≠ {}))

definition *is-polygon-split* ::
(*real^2*) *list* ⇒ *nat* ⇒ *nat* ⇒ *bool* **where**
is-polygon-split vts i j =
(*i* < *length vts* ∧ *j* < *length vts* ∧ *i* < *j* ∧
(*let vts1* = (*take i vts*) *in*
let vts2 = (*take* (*j* − *i* − *1*) (*drop* (*Suc i*) *vts*)) *in*
let vts3 = *drop* (*j* − *i*) (*drop* (*Suc i*) *vts*) *in*
let x = *vts* ! *i in*
let y = *vts* ! *j in*
let p = *make-polygonal-path* (*vts*@[*vts*!*0*]) *in*
let p1 = *make-polygonal-path* (*x*#(*vts2*@[*y*, *x*])) *in*
let p2 = *make-polygonal-path* (*vts1* @ [*x*, *y*] @ *vts3* @ [*vts* ! *0*]) *in*
let c1 = *make-polygonal-path* (*x*#(*vts2*@[*y*])) *in*
let c2 = *make-polygonal-path* (*vts1* @ [*x*, *y*] @ *vts3*) *in*
(*is-polygon-cut* (*vts*@[*vts*!*0*]) *x y* ∧
polygon p ∧ *polygon p1* ∧ *polygon p2* ∧
path-inside p1 ∩ *path-inside p2* = {} ∧
path-inside p1 ∪ *path-inside p2* ∪ (*path-image* (*linepath x y*) − {*x*, *y*}) = *path-inside p*
∧ ((*path-image p1*) − (*path-image* (*linepath x y*))) ∩ ((*path-image p2*) − (*path-image* (*linepath x y*)))
= {}
∧ *path-image p*
= ((*path-image p1*) − (*path-image* (*linepath x y*))) ∪ ((*path-image p2*) −

(*path-image* (*linepath x y*))) ∪ {*x*, *y*}
))

definition *is-polygon-split-path* :: (*real^2*) *list* ⇒ *nat* ⇒ *nat* ⇒ (*real^2*) *list* ⇒ *bool* **where**
is-polygon-split-path vts i j cutvts =
(*i* < *length vts* ∧ *j* < *length vts* ∧ *i* < *j* ∧
(*let vts1* = (*take i vts*) *in*
let vts2 = (*take* (*j* − *i* − *1*) (*drop* (*Suc i*) *vts*)) *in*
let vts3 = *drop* (*j* − *i*) (*drop* (*Suc i*) *vts*) *in*
let x = *vts*!*i in*
let y = *vts*!*j in*
let cutpath = *make-polygonal-path* (*x* # *cutvts* @ [*y*]) *in*
let p = *make-polygonal-path* (*vts*@[*vts*!*0*]) *in*
let p1 = *make-polygonal-path* (*x*#(*vts2* @ [*y*] @ (*rev cutvts*) @ [*x*])) *in*
let p2 = *make-polygonal-path* (*vts1* @ ([*x*] @ *cutvts* @ [*y*]) @ *vts3* @ [*vts* ! *0*]) *in*
let c1 = *make-polygonal-path* (*x*#(*vts2*@[*y*])) *in*
let c2 = *make-polygonal-path* (*vts1* @ ([*x*] @ *cutvts* @ [*y*]) @ *vts3*) *in*
(*is-polygon-cut-path* (*vts*@[*vts*!*0*]) *cutpath* ∧
polygon p ∧ *polygon p1* ∧ *polygon p2* ∧
path-inside p1 ∩ *path-inside p2* = {} ∧
path-inside p1 ∪ *path-inside p2* ∪ (*path-image cutpath* − {*x*, *y*}) = *path-inside p*
∧ ((*path-image p1*) − (*path-image cutpath*)) ∩ ((*path-image p2*) − (*path-image cutpath*)) = {}
∧ *path-image p*
= ((*path-image p1*) − (*path-image cutpath*)) ∪ ((*path-image p2*) − (*path-image cutpath*)) ∪ {*x*, *y*}
))

lemma *polygon-split-add-measure*:
fixes *p p1 p2* :: *R-to-R2*
assumes *is-polygon-split vts i j*
assumes *vts1* = (*take i vts*)
vts2 = (*take* (*j* − *i* − *1*) (*drop* (*Suc i*) *vts*))
vts3 = *drop* (*j* − *i*) (*drop* (*Suc i*) *vts*)
x = *vts* ! *i*
y = *vts* ! *j*
p = *make-polygonal-path* (*vts*@[*vts*!*0*])
p1 = *make-polygonal-path* (*x*#(*vts2*@[*y*, *x*]))
p2 = *make-polygonal-path* (*vts1* @ [*x*, *y*] @ *vts3* @ [*vts* ! *0*])
defines *M1* ≡ *measure lebesgue* (*path-inside p1*) **and**
M2 ≡ *measure lebesgue* (*path-inside p2*) **and**
M ≡ *measure lebesgue* (*path-inside p*)
shows *M1* + *M2* = *M*
proof−
let *?cut* = *linepath x y*
let *?cut-open-image* = (*path-image ?cut*) − {*x*, *y*}
let *?P* = *path-inside p*

let *?P1 = path-inside p1*
let *?P2 = path-inside p2*
let *?M = space lebesgue*
let *?A = sets lebesgue*
let *?μ = emeasure lebesgue*

have *open ?P1*
by (*metis assms(1) assms(3) assms(5) assms(6) assms(8) closed-path-image is-polygon-split-def open-inside path-inside-def polygon-def simple-path-def*)
then have *P1-measurable*: *?P1 ∈ ?A* **by** *simp*

have *open ?P2*
by (*metis assms(1) assms(2) assms(4) assms(5) assms(6) assms(9) closed-path-image is-polygon-split-def open-inside path-inside-def polygon-def simple-path-def*)
then have *P2-measurable*: *?P2 ∈ ?A* **by** *simp*

have *?P1 ∩ ?P2 = {}*
by (*metis assms(1) assms(2) assms(3) assms(4) assms(5) assms(6) assms(8) assms(9) is-polygon-split-def*)
then have *sum-union-finite*: *?μ ?P1 + ?μ ?P2 = ?μ (?P1 ∪ ?P2)*
using *plus-emeasure P1-measurable P2-measurable* **by** *blast*

have *measure lebesgue ?P1 = ?μ ?P1*
by (*metis assms(1) assms(3) assms(5) assms(6) assms(8) bounded-inside bounded-set-imp-lmeasurable bounded-simple-path-image emeasure-eq-ennreal-measure emeasure-notin-sets ennreal-0 fmeasurableD2 is-polygon-split-def measure-zero-top path-inside-def polygon-def*)
moreover have *measure lebesgue ?P2 = ?μ ?P2*
by (*metis Sigma-Algebra.measure-def assms(1) assms(2) assms(4) assms(5) assms(6) assms(9) bounded-inside bounded-path-image bounded-set-imp-lmeasurable emeasure-eq-ennreal-measure emeasure-notin-sets enn2real-top ennreal-0 fmeasurableD2 is-polygon-split-def path-inside-def polygon-def simple-path-def*)
ultimately have *?μ (?P1 ∪ ?P2) = M1 + M2*
using *assms(10) assms(11) sum-union-finite* **by** *auto*
moreover have *?μ (?P1 ∪ ?P2) = ?μ ?P*
proof−
have *?μ (path-image ?cut) = 0* **using** *linepath-has-emeasure-0* **by** *blast*
then have (*path-image ?cut*) ∈ *null-sets lebesgue* **by** *auto*
moreover have {*x*, *y*} ∈ *null-sets lebesgue* **by** *simp*
ultimately have *?cut-open-image ∈ null-sets lebesgue* **using** *measure-Diff-null-set* **by** *auto*
moreover have *?P = ?P1 ∪ ?P2 ∪ ?cut-open-image*
by (*metis assms(1) assms(2) assms(3) assms(4) assms(5) assms(6) assms(7) assms(8) assms(9) is-polygon-split-def*)
ultimately show *?thesis*
by (*simp add*: *P1-measurable P2-measurable emeasure-Un-null-set sets.Un*)
qed
ultimately show *?thesis*
by (*smt* (*verit*, *best*) *M1-def M2-def M-def emeasure-eq-ennreal-measure enn2real-ennreal*

ennreal-neq-top measure-nonneg)
qed

lemma *polygonal-paths-measurable*:
shows *path-image* (*make-polygonal-path vts*) ∈ *sets lebesgue*
proof (*induct vts rule*: *make-polygonal-path-induct*)
case (*Empty ell*)
then show *?case* **by** *auto*
next
case (*Single ell*)
then obtain *a* **where** *ell* = [*a*]
by (*metis Cons-nth-drop-Suc One-nat-def drop0 drop-eq-Nil le-numeral-extra*(*4*) *zero-less-one*)
then show *?case* **using** *make-polygonal-path.simps*(*2*)[*of a*] **by** *simp*
next
case (*Two ell*)
then obtain *a b* **where** *ell* = [*a*, *b*]
by (*metis Cons-nth-drop-Suc One-nat-def Suc-1 append-Nil drop-eq-Nil2 dual-order.refl id-take-nth-drop lessI pos2 take0*)
then show *?case* **using** *make-polygonal-path.simps*(*3*)[*of a b*] **by** *simp*
next
case (*Multiple ell*)
then have *ell* = (*ell* ! *0*) # (*ell* ! *1*) # (*ell* ! *2*) # (*drop 3 ell*)
by (*metis Cons-nth-drop-Suc One-nat-def Suc-1 drop0 le-Suc-eq linorder-not-less numeral-3-eq-3*)
then have *make-polygonal-path ell* =
linepath (*ell* ! *0*) (*ell* ! *1*) +++ *make-polygonal-path* (*ell* ! *1* # *ell* ! *2* # (*drop 3 ell*))
by (*metis make-polygonal-path.simps*(*4*))

then have *path-image* (*make-polygonal-path ell*) = *path-image* (*linepath* (*ell* ! *0*) (*ell* ! *1*)) ∪ *path-image* (*make-polygonal-path* (*ell* ! *1* # *ell* ! *2* # (*drop 2 ell*)))
using *Cons-nth-drop-Suc Multiple.hyps*(*1*) *One-nat-def Suc-1 Un-assoc* ‹*ell* = *ell* ! *0* # *ell* ! *1* # *ell* ! *2* # *drop 3 ell*› *list.discI make-polygonal-path.simps*(*2*) *make-polygonal-path.simps*(*3*) *nth-Cons-0 numeral-3-eq-3 path-image-cons-union*
proof−
have *f1*: *ell* = *ell* ! *0* # *ell* ! *1* # *ell* ! *Suc 1* # *drop 3 ell*
using *Suc-1* ‹*ell* = *ell* ! *0* # *ell* ! *1* # *ell* ! *2* # *drop 3 ell*› **by** *presburger*
have *Suc 1* < *length ell*
by (*smt* (*z3*) *Suc-1* ‹*2* < *length ell*›)
then have *f2*: *drop* (*Suc 1*) *ell* = *ell* ! *Suc 1* # *drop* (*Suc* (*Suc 1*)) *ell*
by (*smt* (*z3*) *Cons-nth-drop-Suc*)
have *f3*: ∀ *v va vs*. *path-image* (*make-polygonal-path* (*v* # *va* # *vs*)) = *path-image* (*linepath v va*) ∪ *path-image* (*make-polygonal-path* (*va* # *vs*))
by (*metis* (*no-types*) *list.discI nth-Cons-0 path-image-cons-union*)
have *f4*: ∀ *V v va*. *path-image* (*linepath* (*v*::(*real*, *2*) *vec*) *va*) ∪ (*path-image* (*linepath va va*) ∪ *V*) = *path-image* (*linepath v va*) ∪ *V*
by *auto*
have *path-image* (*make-polygonal-path ell*) = *path-image* (*make-polygonal-path*

(*ell* ! *0* # *ell* ! *1* # *drop* (*Suc 1*) *ell*))
using *f2 f1* **by** (*simp add*: *numeral-3-eq-3*)
then have *path-image* (*make-polygonal-path ell*) = *path-image* (*linepath* (*ell* ! *0*) (*ell* ! *1*)) ∪ *path-image* (*make-polygonal-path* (*ell* ! *1* # *ell* ! *Suc 1* # *drop* (*Suc 1*) *ell*))
using *f4 f3 f2* **by** *presburger*
then show *?thesis*
using *Suc-1* **by** *presburger*
qed
then show *?case* **using** *Multiple*(*3*)
by (*metis* (*no-types*, *lifting*) *Cons-nth-drop-Suc Multiple.hyps*(*1*) *Multiple.hyps*(*2*) *One-nat-def Suc-1* ‹*ell* = *ell* ! *0* # *ell* ! *1* # *ell* ! *2* # *drop 3 ell*› *list.discI make-polygonal-path.simps*(*3*) *nth-Cons-0 numeral-3-eq-3 path-image-cons-union sets.Un*)

qed

lemma *polygonal-path-has-emeasure-0*:
shows *emeasure lebesgue* (*path-image* (*make-polygonal-path vts*)) = *0*
proof (*induct vts*)
case *Nil*
then show *?case* **by** *auto*
next
case (*Cons a vts*)
then show *?case*
by (*metis linepath-is-negligible make-polygonal-path.simps*(*2*) *negligible-Un negligible-iff-emeasure0 path-image-cons-union polygonal-paths-measurable*)
qed

lemma *polygon-split-path-add-measure*:
fixes *p p1 p2* :: *R-to-R2*
assumes *is-polygon-split-path vts i j cutvts*
assumes *vts1* = (*take i vts*)
vts2 = (*take* (*j* − *i* − *1*) (*drop* (*Suc i*) *vts*))
vts3 = *drop* (*j* − *i*) (*drop* (*Suc i*) *vts*)
x = *vts* ! *i*
y = *vts* ! *j*
p = *make-polygonal-path* (*vts*@[*vts*!*0*])
p1 = *make-polygonal-path* (*x*#(*vts2* @ [*y*] @ (*rev cutvts*) @ [*x*]))
p2 = *make-polygonal-path* (*vts1* @ ([*x*] @ *cutvts* @ [*y*]) @ *vts3* @ [*vts* ! *0*])
defines *M1* ≡ *measure lebesgue* (*path-inside p1*) **and**
M2 ≡ *measure lebesgue* (*path-inside p2*) **and**
M ≡ *measure lebesgue* (*path-inside p*)
shows *M1* + *M2* = *M*
proof−
let *?cut* = *make-polygonal-path* (*x* # *cutvts* @ [*y*])
let *?cut-open-image* = (*path-image ?cut*) − {*x*, *y*}
let *?P* = *path-inside p*
let *?P1* = *path-inside p1*
let *?P2* = *path-inside p2*

let *?M = space lebesgue*
let *?A = sets lebesgue*
let *?μ = emeasure lebesgue*

have *open ?P1*
by (*metis assms(1) assms(3) assms(5) assms(6) assms(8) closed-path-image is-polygon-split-path-def open-inside path-inside-def polygon-def simple-path-def*)
then have *P1-measurable*: *?P1 ∈ ?A* **by** *simp*

have *open ?P2*
by (*metis assms(1) assms(2) assms(4) assms(5) assms(6) assms(9) closed-path-image is-polygon-split-path-def open-inside path-inside-def polygon-def simple-path-def*)
then have *P2-measurable*: *?P2 ∈ ?A* **by** *simp*

have *?P1 ∩ ?P2 = {}*
by (*metis assms(1) assms(2) assms(3) assms(4) assms(5) assms(6) assms(8) assms(9) is-polygon-split-path-def*)
then have *sum-union-finite*: *?μ ?P1 + ?μ ?P2 = ?μ (?P1 ∪ ?P2)*
using *plus-emeasure P1-measurable P2-measurable* **by** *blast*

have *?μ (path-image q) = 0 ⟹ (path-image q) ∈ null-sets lebesgue* **if** *∗*: *path-image q ∈ sets lebesgue* **for** *q*::*real ⇒ (real, 2) vec*
using *null-sets-def ∗* **by** *blast*

have *measure lebesgue ?P1 = ?μ ?P1*
by (*metis assms(1) assms(3) assms(5) assms(6) assms(8) bounded-inside bounded-set-imp-lmeasurable bounded-simple-path-image emeasure-eq-ennreal-measure emeasure-notin-sets ennreal-0 fmeasurableD2 is-polygon-split-path-def measure-zero-top path-inside-def polygon-def*)
moreover have *measure lebesgue ?P2 = ?μ ?P2*
by (*metis Sigma-Algebra.measure-def assms(1) assms(2) assms(4) assms(5) assms(6) assms(9) bounded-inside bounded-path-image bounded-set-imp-lmeasurable emeasure-eq-ennreal-measure emeasure-notin-sets enn2real-top ennreal-0 fmeasurableD2 is-polygon-split-path-def path-inside-def polygon-def simple-path-def*)
ultimately have *?μ (?P1 ∪ ?P2) = M1 + M2*
using *assms(10) assms(11) sum-union-finite* **by** *auto*
moreover have *?μ (?P1 ∪ ?P2) = ?μ ?P*
proof−
have *?μ (path-image ?cut) = 0* **using** *polygonal-path-has-emeasure-0*
by *presburger*
then have (*path-image ?cut*) *∈ null-sets lebesgue* **using** *polygonal-paths-measurable*
by *blast*
moreover have *{x, y} ∈ null-sets lebesgue* **by** *simp*
ultimately have *?cut-open-image ∈ null-sets lebesgue* **using** *measure-Diff-null-set* **by** *auto*
moreover have *?P = ?P1 ∪ ?P2 ∪ ?cut-open-image*
by (*metis assms(1) assms(2) assms(3) assms(4) assms(5) assms(6) assms(7) assms(8) assms(9) is-polygon-split-path-def*)
ultimately show *?thesis*

by (*simp add*: *P1-measurable P2-measurable emeasure-Un-null-set sets.Un*)
qed
ultimately show *?thesis*
by (*smt* (*verit*, *best*) *M1-def M2-def M-def emeasure-eq-ennreal-measure enn2real-ennreal ennreal-neq-top measure-nonneg*)
qed

lemma *polygon-cut-path-to-split-path-vtx0*:
fixes *p* :: *R-to-R2*
assumes *polygon-p*: *polygon p* **and**
i-gt: *i* > *0* **and**
i-lt: *i* < *length vts* **and**
p-is: *p* = *make-polygonal-path* (*vts* @ [*vts* ! *0*]) **and**
cutpath: *cutpath* = *make-polygonal-path* ([*vts*!*0*] @ *cutvts* @ [*vts*!*i*]) **and**
have-cut: *is-polygon-cut-path* (*vts* @ [*vts*!*0*]) *cutpath*
shows *is-polygon-split-path vts 0 i cutvts*
proof −
let *?vts2* = *take* (*i* − *1*) (*drop 1 vts*)
let *?vts3* = *drop i* (*drop 1 vts*)
let *?x* = *vts* ! *0*
let *?y* = *vts* ! *i*

let *?c3-vts* = [*?x*] @ *cutvts* @ [*?y*]
let *?c3* = *cutpath*
let *?c3-rev-vts* = *rev ?c3-vts*
let *?c3-rev* = *make-polygonal-path ?c3-rev-vts*
let *?c3'* = *reversepath ?c3*

let *?p* = *make-polygonal-path* (*vts* @ [*vts* ! *0*])
let *?p1-vts* = *?x* # *?vts2* @ *?c3-rev-vts*
let *?p1* = *make-polygonal-path ?p1-vts*
let *?p1-rot-vts* = *?c3-rev-vts* @ *?vts2* @ [*?y*]
let *?p1-rot* = *make-polygonal-path ?p1-rot-vts*
let *?p2-vts* = *?c3-vts* @ *?vts3* @ [*?x*]
let *?p2* = *make-polygonal-path ?p2-vts*
let *?c1-vts* = *?x* # *?vts2* @ [*?y*]
let *?c1* = *make-polygonal-path ?c1-vts*
let *?c2-vts* = [*?y*] @ *?vts3* @ [*?x*]
let *?c2* = *reversepath* (*make-polygonal-path ?c2-vts*)
let *?c2'-vts* = [*?y*] @ *?vts3* @ [*?x*]
let *?c2'* = (*make-polygonal-path* (*?c2'-vts*))

have *distinct-vts*: *distinct vts*
using *polygon-p p-is*
using *polygon-def simple-polygonal-path-vts-distinct* **by** *force*
have *len-vts-gteq3*: *length vts* ≥ *3*
using *polygon-p p-is polygon-vertices-length-at-least-4* **by** *fastforce*

then have *?x* # *?vts2* @ [*?y*] = *take* (*i*+*1*) (*vts*@ [*vts* ! *0*])

by (*smt* (*verit*, *ccfv-threshold*) *i-gt Cons-nth-drop-Suc Suc-eq-plus1 Suc-pred′ add-less-cancel-left butlast-snoc drop0 drop-drop hd-drop-conv-nth i-lt length-append-singleton length-greater-0-conv less-imp-le-nat linorder-not-less list.size*(*3*) *plus-1-eq-Suc take-Suc-Cons take-all-iff take-butlast take-hd-drop*)
have [*?y*] @ *?vts3* @ [*?x*] = *drop* (*i*) (*vts* @ [*vts* ! *0*])
using *i-gt*
by (*metis* (*no-types*, *lifting*) *Cons-eq-appendI Cons-nth-drop-Suc Suc-eq-plus1 append-Nil diff-is-0-eq′ drop-0 drop-append drop-drop i-lt less-imp-le-nat*)

have *card-gteq*: *card* (*set vts*) ≥ *3*
using *polygon-at-least-3-vertices-wraparound polygon-p p-is*
by (*metis butlast-conv-take butlast-snoc*)
then have *vts* ≠ []
by *auto*
then have *vts-is*: *vts* = *?x* # *?vts2* @ *?y* # *?vts3*
using *split-up-a-list-into-3-parts*[*of 0 vts i*] *i-gt i-lt*
by *auto*

have *elem-prop1*: *last ?c1-vts* = *?y*
by (*metis* (*no-types*, *lifting*) *last.simps snoc-eq-iff-butlast*)
have *elem-prop2*: (*vts* ! *0* # (*rev ?vts3*) @ [*vts* ! *i*]) !
(*length* (*vts* ! *0* # *drop i* (*drop 1 vts*) @ [*vts* ! *i*]) − *1*) = *vts* ! *i*
by (*metis diff-Suc-1 length-Cons length-append-singleton length-rev nth-Cons-Suc nth-append-length*)
have *path-image cutpath* = *path-image ?c3′* **by** *simp*
then have *path-image ?p1* = *path-image* (*?c1* +++ *?c3-rev*)
using *elem-prop1 assms make-polygonal-path-image-append-alt*[*of ?p1 ?p1-vts ?c1 ?c1-vts ?c3-rev ?c3-rev-vts*]
by *simp*
also have ... = *path-image ?c1* ∪ *path-image ?c3-rev*
by (*metis* (*no-types*, *opaque-lifting*) *append-Cons append-Nil elem-prop1 hd-conv-nth last-conv-nth list.discI list.sel*(*1*) *path-image-join polygon-pathfinish polygon-pathstart rev.simps*(*2*) *rev-rev-ident*)
finally have *image-prop*: *path-image ?p1* = *path-image ?c1* ∪ *path-image cutpath*
using *rev-vts-path-image cutpath* **by** *presburger*
have *path-image ?c3′* = *path-image ?c3*
using *cutpath rev-vts-path-image* **by** *force*
then have *path-image-p1*: *path-image ?c1* ∪ *path-image ?c3* = *path-image ?p1*
using *image-prop* **by** *presburger*

have *?p2-vts* = *?c3-vts* @ (*tl ?c2-vts*) **by** *simp*
then have *path-image ?p2* = *path-image* (*?c3* +++ *?c2′*)
using *make-polygonal-path-image-append-alt*[*of ?p2 ?p2-vts ?c3 ?c3-vts ?c2′ ?c2-vts*]
unfolding *assms* **by** *auto*
then have *path-image-p2*: *path-image ?c2* ∪ *path-image ?c3* = *path-image ?p2*
by (*metis* (*no-types*, *opaque-lifting*) *Un-commute append-Cons append-Nil cutpath last-conv-nth nth-Cons-0 path-image-join path-image-reversepath polygon-pathfinish*

polygon-pathstart snoc-eq-iff-butlast)

have *drop 1 vts = take (i − 1) (drop 1 vts) @ [vts ! i] @ drop i (drop 1 vts)*
by (*metis* (*no-types*, *lifting*) *Cons-eq-appendI Cons-nth-drop-Suc Suc-eq-plus1 Suc-pred′ append.simps*(*1*) *append-take-drop-id drop-drop i-gt i-lt*)
then have *vts-is*: *vts @ [vts ! 0] = vts ! 0 # take (i − 1) (drop 1 vts) @ [vts ! i] @ drop i (drop 1 vts) @ [vts ! 0]*
by (*metis* (*no-types*, *opaque-lifting*) *Cons-nth-drop-Suc One-nat-def append.assoc append-Cons drop0 i-lt length-pos-if-in-set nth-mem*)
let *?vts1′ = take (i − 1) (drop 1 vts)*
let *?vts2′ = drop i (drop 1 vts)*
have *path-im-p*: *path-image*
(*make-polygonal-path*
((*vts ! 0 # ?vts1′*) *@ [vts ! i] @ [vts ! i] @ ?vts2′ @ [vts ! 0])*) =
path-image
(*make-polygonal-path*
((*vts ! 0 # ?vts1′*) *@ [vts ! i] @ ?vts2′ @ [vts ! 0]*))
using *make-polygonal-path-image-append-helper*[*of vts ! 0 # ?vts1′ ?vts2′ @ [vts ! 0*]] **by** *auto*
have *path-image*
(*make-polygonal-path*
((*vts ! 0 # ?vts1′*) *@ [vts ! i] @ [vts ! i] @ ?vts2′ @ [vts ! 0]*)) = *path-image* (*make-polygonal-path* ((*vts ! 0 # ?vts1′*) *@ [vts ! i]*) *+++* (*linepath* (*vts ! i*) (*vts ! i*)) *+++ make-polygonal-path* ([*vts ! i] @ ?vts2′ @ [vts ! 0]*))
using *make-polygonal-path-image-append*[*of* (*vts ! 0 # ?vts1′*) *@ [vts ! i] [vts ! i] @ ?vts2′ @ [vts ! 0*]]

by (*smt* (*verit*) *add-2-eq-Suc′ append.assoc append-Cons diff-Suc-1 le-add2 length-Cons length-append-singleton nth-Cons-0 nth-append-length*)
then have *path-image p = path-image* (*make-polygonal-path* ((*vts ! 0 # ?vts1′*) *@ [vts ! i]*) *+++* (*linepath* (*vts ! i*) (*vts ! i*)) *+++ make-polygonal-path* ([*vts ! i] @ ?vts2′ @ [vts ! 0]*))
using *path-im-p p-is vts-is*
by *simp*
then have *path-image p = path-image ?c1 ∪ path-image* (*linepath* (*vts ! i*) (*vts ! i*)) *∪ path-image* (*make-polygonal-path* ([*vts ! i] @ ?vts2′ @ [vts ! 0]*))
by (*metis* (*no-types*, *lifting*) *Un-assoc append-Cons elem-prop1 list.discI nth-Cons-0 path-image-join pathfinish-linepath pathstart-join pathstart-linepath polygon-pathfinish polygon-pathstart last-conv-nth*)
moreover have *... = path-image ?c1 ∪ {vts ! i} ∪ path-image* (*make-polygonal-path* ([*vts ! i] @ ?vts2′ @ [vts ! 0]*))
by *auto*
moreover have *... = path-image ?c1 ∪ path-image* (*make-polygonal-path* ([*vts ! i] @ ?vts2′ @ [vts ! 0]*))
using *vertices-on-path-image* **by** *fastforce*
ultimately have *path-image-p*: *path-image p = path-image ?c1 ∪ path-image ?c2*
using *path-image-reversepath* **by** *blast*

have *simple-path-polygon*: *simple-path* (*make-polygonal-path* (*?x # ?vts2 @ ?y # ?vts3 @ [?x]*))
using *polygon-p p-is vts-is*
using *Cons-eq-appendI append-self-conv2 polygon-def* **by** *auto*
then have *loop-free-polygon*: *loop-free* (*make-polygonal-path* (*?x # ?vts2 @ ?y # ?vts3 @ [?x]*))
unfolding *simple-path-def* **by** *auto*

have *loop-free-p*: *loop-free p*
using *polygon-p p-is* **unfolding** *polygon-def simple-path-def* **by** *auto*

have *sublist-c1*: *sublist* (*?x # ?vts2 @ [?y]*) *vts*
using ‹*vts ! 0 # take* (*i − 1*) (*drop 1 vts*) *@ [vts ! i] = take* (*i + 1*) (*vts @ [vts ! 0]*)› *i-lt* **by** *auto*
then have *sublist-c1*: *sublist* (*?x # ?vts2 @ [?y]*) (*vts@[vts !0]*)
by (*metis* ‹*vts ! 0 # take* (*i − 1*) (*drop 1 vts*) *@ [vts ! i] = take* (*i + 1*) (*vts @ [vts ! 0]*)› *sublist-take*)
then have *loop-free ?c1*
using *sublist-is-loop-free p-is loop-free-p sublist-c1*
by (*metis One-nat-def Suc-1 Suc-eq-plus1 Suc-leI Suc-le-mono* ‹*vts ! 0 # take* (*i − 1*) (*drop 1 vts*) *@ [vts ! i] = take* (*i + 1*) (*vts @ [vts ! 0]*)› *i-gt i-lt length-append-singleton less-imp-le-nat take-i-is-loop-free*)
then have *simple-c1*: *simple-path ?c1*
unfolding *simple-path-def*
using *make-polygonal-path-gives-path* **by** *blast*
have *start-c1*: *pathstart ?c1 = ?x*
using *polygon-pathstart*
by (*metis Cons-eq-appendI list.discI nth-Cons-0*)
have *finish-c1*: *pathfinish ?c1 = ?y*
using *polygon-pathfinish*
by (*metis Cons-eq-appendI diff-Suc-1 length-append-singleton list.discI nth-append-length*)

have *sublist-c2*: *sublist* ([*?y*] *@ ?vts3 @* [*?x*]) (*vts@*[*vts !0*])
by (*metis* ‹[*vts ! i*] *@ drop i* (*drop 1 vts*) *@* [*vts ! 0*] *= drop i* (*vts @* [*vts ! 0*])› *sublist-drop*)
have *i* ≤ *length* (*tl vts*) **using** *i-lt* **by** *fastforce*
then have *loop-free ?c2*
by (*metis* (*no-types*) *Suc-1* ‹[*vts ! i*] *@ drop i* (*drop 1 vts*) *@* [*vts ! 0*] *= drop i* (*vts @* [*vts ! 0*])› ‹*vts ≠* []› *butlast-snoc drop-Suc drop-i-is-loop-free length-butlast length-drop loop-free-p loop-free-reversepath p-is tl-append2*)
then have *simple-c2*: *simple-path ?c2*
unfolding *simple-path-def*
using *make-polygonal-path-gives-path*
using *path-imp-reversepath* **by** *blast*
have *start-c2*: *pathstart ?c2 = ?x*
using *polygon-pathfinish*
by (*metis* (*no-types*, *lifting*) *Nil-is-append-conv last-appendR last-conv-nth path-start-reversepath polygon-pathfinish snoc-eq-iff-butlast*)

have *finish-c2*: *pathfinish ?c2 = ?y*
using *polygon-pathstart* **by** *auto*

have *path-image-int*: *path-image ?c1 ⊆ path-image ?p*
unfolding *path-image-def*
by (*metis Un-upper1 p-is path-image-def path-image-p*)
moreover have *path-image ?p ∩ path-image ?c3 ⊆ {vts ! 0, vts ! i}*
using *have-cut* **unfolding** *is-polygon-cut-path-def*
by (*metis* (*no-types*, *lifting*) *Int-commute append-Cons append-is-Nil-conv cut-path last-appendR last-conv-nth last-snoc not-Cons-self2 nth-Cons-0 polygon-pathfinish polygon-pathstart set-eq-subset*)
ultimately have *vts-subset-c1c3*: *path-image ?c1 ∩ path-image ?c3 ⊆ {?x, ?y}*
by *blast*
have *other-subset1*: *{vts ! 0, vts ! i} ⊆ path-image ?c1*
using *vertices-on-path-image* **by** *fastforce*
have *other-subset2*: *{vts ! 0, vts ! i} ⊆ path-image ?c3*
unfolding *assms* **using** *vertices-on-path-image* **by** *force*
then have *c1-inter-c3*: *path-image ?c1 ∩ path-image ?c3 = {vts ! 0, vts ! i}*
using *vts-subset-c1c3 other-subset1 other-subset2* **by** *blast*
then have *path-image ?c1 ∩ path-image ?c3-rev = {pathstart ?c1, pathstart ?c3-rev}*
by (*metis rev-vts-path-image append-Cons append-Nil cutpath hd-conv-nth list.discI list.sel*(*1*) *polygon-pathstart rev.simps*(*2*) *rev-rev-ident*)

then have *c1-inter-c3'*: *path-image* (*make-polygonal-path* (*vts ! 0 # take* (*i − 1*) (*drop 1 vts*) @ [*vts ! i*])) ∩
path-image (*make-polygonal-path* (*rev* ([*vts ! 0*] @ *cutvts* @ [*vts ! i*])))
⊆ {*pathstart* (*make-polygonal-path* (*vts ! 0 # take* (*i − 1*) (*drop 1 vts*) @ [*vts ! i*])),
pathstart (*make-polygonal-path* (*rev* ([*vts ! 0*] @ *cutvts* @ [*vts ! i*])))}
by *blast*
have *last-is-head*: *last ?c3-rev-vts = hd ?c1-vts* **by** *auto*
have *vts-append*: *vts ! 0 # take* (*i − 1*) (*drop 1 vts*) @ *rev* ([*vts ! 0*] @ *cutvts* @ [*vts ! i*]) =
(*vts ! 0 # take* (*i − 1*) (*drop 1 vts*) @ [*vts ! i*]) @
tl (*rev* ([*vts ! 0*] @ *cutvts* @ [*vts ! i*]))
by *simp*
have *loop-free*: *loop-free* (*make-polygonal-path* (*vts ! 0 # take* (*i − 1*) (*drop 1 vts*) @ [*vts ! i*])) ∧
loop-free (*make-polygonal-path* (*rev* ([*vts ! 0*] @ *cutvts* @ [*vts ! i*])))
by (*metis Suc-eq-plus1 Suc-le-mono Zero-neq-Suc ‹vts ! 0 # take* (*i − 1*) (*drop 1 vts*) @ [*vts ! i*] = *take* (*i + 1*) (*vts* @ [*vts ! 0*])› *cutpath diff-Suc-1 have-cut i-gt i-lt is-polygon-cut-path-def length-append-singleton less-2-cases less-imp-le-nat less-nat-zero-code linorder-le-less-linear loop-free-p p-is rev-vts-is-loop-free simple-path-def take-i-is-loop-free*)
have *last-is-head2*:
last (*vts ! 0 # take* (*i − 1*) (*drop 1 vts*) @ [*vts ! i*]) =
hd (*rev* ([*vts ! 0*] @ *cutvts* @ [*vts ! i*])) **by** *simp*

have *arcs*: *arc* (*make-polygonal-path* (*vts* ! *0* # *take* (*i* − *1*) (*drop 1 vts*) @ [*vts* ! *i*])) ∧
arc (*make-polygonal-path* (*rev* ([*vts* ! *0*] @ *cutvts* @ [*vts* ! *i*])))
using *Nil-is-append-conv append-Cons constant-linepath-is-not-loop-free cutpath finish-c1 have-cut hd-conv-nth is-polygon-cut-path-def last-appendR last-conv-nth last-is-head last-is-head2 last-snoc list.sel*(*1*) *loop-free make-polygonal-path.simps*(*1*) *make-polygonal-path-gives-path polygon-pathfinish polygon-pathstart simple-path-def simple-path-imp-arc loop-free*
by (*smt* (*verit*, *ccfv-SIG*))
then have *loop-free ?p1*
using *loop-free-append*[*of ?p1 ?p1-vts ?c1 ?c1-vts ?c3-rev ?c3-rev-vts*,
OF - - - vts-append loop-free c1-inter-c3′ - last-is-head2 arcs] **using** *last-is-head* **by** *blast*

then have *simple-path ?p1*
unfolding *simple-path-def*
using *make-polygonal-path-gives-path* **by** *blast*
moreover have *closed-path ?p1*
using *polygon-pathstart polygon-pathfinish*
unfolding *closed-path-def*
using *elem-prop1 make-polygonal-path-gives-path*
by (*smt* (*verit*, *best*) *append-is-Nil-conv last-ConsR last-appendR last-conv-nth last-snoc list.discI nth-Cons-0 rev-append singleton-rev-conv*)
ultimately have *polygon-p1*: *polygon ?p1* **unfolding** *polygon-def polygonal-path-def* **by** *fastforce*

have *path-image-int*: *path-image ?c2* ⊆ *path-image* (*make-polygonal-path* (*vts* @ [*vts* ! *0*]))
unfolding *path-image-def* **using** *path-image-p*
by (*simp add*: *p-is path-image-def*)
then have *vts-subset-c2c3*: *path-image ?c2* ∩ *path-image ?c3* ⊆ {*?x*, *?y*}
using *have-cut* **unfolding** *is-polygon-cut-path-def* **using** ‹*path-image* (*make-polygonal-path* (*vts* @ [*vts* ! *0*])) ∩ *path-image cutpath* ⊆ {*vts* ! *0*, *vts* ! *i*}› **by** *auto*
have *other-subset3*: {*vts* ! *0*, *vts* ! *i*} ⊆ *path-image ?c2*
using *vertices-on-path-image* **by** *fastforce*
have *other-subset4*: {*vts* ! *0*, *vts* ! *i*} ⊆ *path-image ?c3*
unfolding *assms* **using** *vertices-on-path-image* **by** *fastforce*
have *c2-inter-c3*: *path-image ?c2* ∩ *path-image ?c3* = {*vts* ! *0*, *vts* ! *i*}
using *vts-subset-c2c3 other-subset3 other-subset4* **by** *blast*
have *path-p2*: *path ?p2*
using *make-polygonal-path-gives-path* **by** *blast*
have *pathfinish ?p2* = *vts* ! *0*
using *polygon-pathfinish*
by (*metis Nil-is-append-conv last-appendR last-conv-nth last-snoc list.discI*)
then have *closed-p2*: *closed-path ?p2*
unfolding *closed-path-def* **using** *polygon-pathstart*
using *path-p2* **by** *auto*

have ([*vts* ! *0*] @ *cutvts* @ [*vts* ! *i*]) @ *drop i* (*drop 1 vts*) @ [*vts* ! *0*] =
([*vts* ! *0*] @ *cutvts* @ [*vts* ! *i*]) @ *tl* ([*vts* ! *i*] @ *drop i* (*drop 1 vts*) @ [*vts* ! *0*])
by *force*
moreover have *loop-free cutpath* ∧
loop-free (*make-polygonal-path* ([*vts* ! *i*] @ *drop i* (*drop 1 vts*) @ [*vts* ! *0*]))
by (*metis* ‹*loop-free* (*reversepath* (*make-polygonal-path* ([*vts* ! *i*] @ *drop i* (*drop 1 vts*) @ [*vts* ! *0*])))› *cutpath loop-free loop-free-reversepath rev-rev-ident rev-vts-is-loop-free reversepath-reversepath*)
moreover have *path-image cutpath* ∩ *path-image* (*make-polygonal-path* ([*vts* ! *i*] @ *drop i* (*drop 1 vts*) @ [*vts* ! *0*]))
⊆ {*pathstart cutpath*,
pathstart (*make-polygonal-path* ([*vts* ! *i*] @ *drop i* (*drop 1 vts*) @ [*vts* ! *0*]))}
using *c2-inter-c3 cutpath polygon-pathstart* **by** *auto*
moreover have *last* ([*vts* ! *i*] @ *drop i* (*drop 1 vts*) @ [*vts* ! *0*]) ≠ *hd* ([*vts* ! *0*] @ *cutvts* @ [*vts* ! *i*]) ⟶
path-image cutpath ∩ *path-image* (*make-polygonal-path* ([*vts* ! *i*] @ *drop i* (*drop 1 vts*) @ [*vts* ! *0*]))
⊆ {*pathstart* (*make-polygonal-path* ([*vts* ! *i*] @ *drop i* (*drop 1 vts*) @ [*vts* ! *0*]))}
by *simp*
moreover have *last* ([*vts* ! *0*] @ *cutvts* @ [*vts* ! *i*]) = *hd* ([*vts* ! *i*] @ *drop i* (*drop 1 vts*) @ [*vts* ! *0*])
by *simp*
moreover have *arc cutpath* ∧ *arc* (*make-polygonal-path* ([*vts* ! *i*] @ *drop i* (*drop 1 vts*) @ [*vts* ! *0*]))
by (*metis* (*no-types*, *lifting*) *arc-simple-path arcs calculation*(*2*) *finish-c1 finish-c2 have-cut is-polygon-cut-path-def make-polygonal-path-gives-path pathfinish-reversepath pathstart-reversepath simple-path-def start-c1 start-c2*)
ultimately have *loop-free ?p2*
using *loop-free-append*[*of ?p2 ?p2-vts ?c3 ?c3-vts ?c2' ?c2'-vts*,
OF - - -] **using** *cutpath* **by** *blast*
then have *polygon-p2*: *polygon ?p2*
using *path-p2 closed-p2* **unfolding** *polygon-def simple-path-def polygonal-path-def*

by *blast*

have *simple-c3*: *simple-path ?c3*
using *have-cut* **unfolding** *is-polygon-cut-path-def* **by** *meson*
have *start-c3*: *pathstart ?c3* = *?x* **unfolding** *assms* **using** *polygon-pathstart* **by** *simp*
have *finish-c3*: *pathfinish ?c3* = *?y* **unfolding** *assms* **using** *polygon-pathfinish* **by** *simp*
have *pathstart cutpath* = *?x* **using** *assms polygon-pathstart* **by** *force*
moreover have *pathfinish cutpath* = *?y* **using** *assms polygon-pathfinish* **by** *simp*
ultimately have *vts-neq*: *vts* ! *0* ≠ *vts* ! *i*
using *have-cut* **unfolding** *is-polygon-cut-path-def* **by** *force*
have *c1-inter-c2*: *path-image ?c1* ∩ *path-image ?c2* = {*vts* ! *0*, *vts* ! *i*}

proof−
obtain *i* **where** *i1*: (*?x # ?vts2 @ [?y] = take i (vts @ [vts!0])*) **and**
i2: (*[?y] @ ?vts3 @ [?x] = drop (i−1) (vts @ [vts!0])*)
by (*metis ‹[vts ! i] @ drop i (drop 1 vts) @ [vts ! 0] = drop i (vts @ [vts ! 0])› ‹vts ! 0 # take (i − 1) (drop 1 vts) @ [vts ! i] = take (i + 1) (vts @ [vts ! 0])› add.commute add-diff-cancel-left′*)
moreover have *1*: *i ≥ 1 ∧ i < length (vts @ [vts!0])*
by (*metis* (*no-types*, *lifting*) *bot-nat-0.extremum less-one Nil-is-append-conv append-Cons calculation diff-is-0-eq drop-Cons′ linorder-not-less list.inject not-Cons-self2 same-append-eq take-all vts-is vts-neq*)
moreover have *2*: *?p = make-polygonal-path (vts @ [vts!0]) ∧ loop-free ?p*
unfolding *polygon-of-def* **using** *p-is polygon-p* **unfolding** *polygon-def simple-path-def* **by** *blast*
ultimately have *path-image ?c1 ∩ path-image (make-polygonal-path ([?y] @ ?vts3 @ [?x])) ⊆ {pathstart ?c1, pathstart (make-polygonal-path ([?y] @ ?vts3 @ [?x]))}*
using *loop-free-split-int*[*of ?p vts @ [vts!0] ?x # ?vts2 @ [?y] i [?y] @ ?vts3 @ [?x] ?c1 make-polygonal-path ([?y] @ ?vts3 @ [?x]) length (vts @ [vts!0]),*
OF 2 i1 i2 - - - 1]
by *presburger*
moreover have *path-image ?c2 = path-image (make-polygonal-path ([?y] @ ?vts3 @ [?x]))* **using** *path-image-reversepath* **by** *fast*
moreover have *pathstart (make-polygonal-path ([?y] @ ?vts3 @ [?x])) = ?y* **using** *polygon-pathstart* **by** *auto*
moreover have *pathstart ?c1 = ?x* **using** *polygon-pathstart* **by** *auto*
ultimately show *?thesis*
using *other-subset1 other-subset3 subset-antisym* **by** *force*
qed

have *non-empty-inter*: *path-image ?c3 ∩ inside(path-image ?c1 ∪ path-image ?c2) ≠ {}*
using *have-cut path-image-p p-is*
unfolding *is-polygon-cut-path-def path-inside-def*
by *fastforce*

have *p1-minus*: *((path-image ?p1) − (path-image ?c3)) = path-image ?c1 − {?x, ?y}*
using *c1-inter-c3 path-image-p1* **by** *blast*
have *p2-minus*: *((path-image ?p2) − (path-image ?c3)) = path-image ?c2 − {?x, ?y}*
using *c2-inter-c3 path-image-p2* **by** *auto*

then have *path-im-intersect-minus*: *((path-image ?p1) − (path-image ?c3)) ∩ ((path-image ?p2) − (path-image (linepath ?x ?y))) = {}*
using *c1-inter-c2 p1-minus p2-minus*
by *blast*
have *((path-image ?p1) − (path-image ?c3)) ∪ ((path-image ?p2) − (path-image ?c3)) ∪ {?x, ?y} = ((path-image ?p1) − (path-image ?c3) ∪ {?x, ?y}) ∪ ((path-image ?p2) − (path-image ?c3) ∪ {?x, ?y})*

by *auto*
then have ((*path-image ?p1*) − (*path-image* (*?c3*))) ∪ ((*path-image ?p2*) − (*path-image* (*?c3*))) ∪ {*?x*, *?y*} = ((*path-image ?c1*) − {*?x*, *?y*} ∪ {*?x*, *?y*}) ∪ ((*path-image ?c2*) − {*?x*, *?y*} ∪ {*?x*, *?y*})
using *p1-minus p2-minus* **by** *simp*
then have ((*path-image ?p1*) − (*path-image* (*?c3*))) ∪ ((*path-image ?p2*) − (*path-image* (*?c3*))) ∪ {*?x*, *?y*} = *path-image ?c1* ∪ *path-image ?c2*
using *other-subset1 other-subset3* **by** *auto*
then have *path-im-intersect-union*: *path-image ?p* = ((*path-image ?p1*) − (*path-image* (*?c3*))) ∪ ((*path-image ?p2*) − (*path-image* (*?c3*))) ∪ {*?x*, *?y*}
using *path-image-p p-is* **by** *auto*

have *inside*(*path-image ?c1* ∪ *path-image ?c3*) ∩ *inside*(*path-image ?c2* ∪ *path-image ?c3*) = {}
using *split-inside-simple-closed-curve-real2*[*OF simple-c1 start-c1 finish-c1 simple-c2 start-c2 finish-c2*
simple-c3 start-c3 finish-c3 vts-neq c1-inter-c2 c1-inter-c3 c2-inter-c3 non-empty-inter]
by *fast*
then have *empty-inter*: *path-inside ?p1* ∩ *path-inside ?p2* = {}
using *path-image-p1 path-image-p2* **unfolding** *path-inside-def*
by *force*
have *inside*(*path-image ?c1* ∪ *path-image ?c3*) ∪ *inside*(*path-image ?c2* ∪ *path-image ?c3*) ∪
(*path-image ?c3* − {*vts* ! *0*, *vts* ! *i*}) = *inside*(*path-image ?c1* ∪ *path-image ?c2*)
using *split-inside-simple-closed-curve-real2*[*OF simple-c1 start-c1 finish-c1 simple-c2 start-c2 finish-c2*
simple-c3 start-c3 finish-c3 vts-neq c1-inter-c2 c1-inter-c3 c2-inter-c3 non-empty-inter]
by *fast*
then have *inside*: *path-inside ?p1* ∪ *path-inside ?p2* ∪ (*path-image ?c3* − {*?x*, *?y*}) = *path-inside p*
using *path-image-p1 path-image-p1 path-image-p* **unfolding** *path-inside-def*
by (*smt* (*z3*) *Diff-cancel Int-Un-distrib2 c1-inter-c2 c1-inter-c3 finish-c1 inf-commute inf-sup-absorb nonempty-simple-path-endless path-image-p2 simple-c1 start-c1*)
have *first-part*: *0* < *length vts* ∧
i < *length vts* ∧
0 < *i*
using *assms*
by *auto*
have *second-part-helper*: *is-polygon-cut-path* (*vts* @ [*vts* ! *0*]) *cutpath* ∧
polygon ?p ∧
polygon ?p1 ∧
polygon ?p2 ∧
path-inside ?p1 ∩ *path-inside ?p2* = {} ∧
path-inside ?p1 ∪ *path-inside ?p2* ∪ (*path-image* (*?c3*) − {*?x*, *?y*}) = *path-inside p*

$\wedge$ ((*path-image ?p1*) − (*path-image* (*?c3*))) ∩ ((*path-image ?p2*) − (*path-image* (*?c3*))) = {}
$\wedge$ *path-image ?p* = ((*path-image ?p1*) − (*path-image* (*?c3*))) ∪ ((*path-image ?p2*) − (*path-image* (*?c3*))) ∪ {*?x*, *?y*}
using *polygon-p p-is polygon-p1 polygon-p2 empty-inter inside have-cut path-im-intersect-minus path-im-intersect-union*
proof−
have {} = *path-image cutpath* ∪ *path-image* (*make-polygonal-path* (*vts* ! *0* # *take* (*i* − *1*) (*drop 1 vts*) @ [*vts* ! *i*])) ∩ *path-image* (*reversepath* (*make-polygonal-path* ([*vts* ! *i*] @ *drop i* (*drop 1 vts*) @ [*vts* ! *0*]))) − *path-image cutpath*
using *c1-inter-c2 c2-inter-c3* **by** *fastforce*
then have {} = (*path-image cutpath* ∪ *path-image* (*make-polygonal-path* (*vts* ! *0* # *take* (*i* − *1*) (*drop 1 vts*) @ [*vts* ! *i*]))) ∩ (*path-image cutpath* ∪ *path-image* (*reversepath* (*make-polygonal-path* ([*vts* ! *i*] @ *drop i* (*drop 1 vts*) @ [*vts* ! *0*])))) − *path-image cutpath*
by *blast*
then show *?thesis*
using *empty-inter have-cut inside polygon-p1 polygon-p2 Int-Diff image-prop p-is path-im-intersect-union path-image-p2 polygon-p*
by *auto*
qed
have *vts-relation*: (*let vts1* = *take 0 vts*; *vts2* = *take* (*i* − *0* − *1*) (*drop* (*Suc 0*) *vts*);
vts3 = *drop* (*i* − *0*) (*drop* (*Suc 0*) *vts*); *x* = *vts* ! *0*; *y* = *vts* ! *i*;
p = *make-polygonal-path* (*vts* @ [*vts* ! *0*]); *p1* = *make-polygonal-path* (*x* # *vts2* @ *?c3-rev-vts*);
p2 = *make-polygonal-path* (*?c3-vts* @ *vts3* @ [*x*]) *in*
vts1 = [] $\wedge$ *vts2* = *?vts2* $\wedge$ *vts3* = *?vts3* $\wedge$ *p* = *?p* $\wedge$ *p1* = *?p1* $\wedge$ *p2* = *?p2*)
by *simp*
have *second-part*: (*let vts1* = *take 0 vts*; *vts2* = *take* (*i* − *0* − *1*) (*drop* (*Suc 0*) *vts*);
vts3 = *drop* (*i* − *0*) (*drop* (*Suc 0*) *vts*); *x* = *vts* ! *0*; *y* = *vts* ! *i*;
p = *make-polygonal-path* (*vts* @ [*vts* ! *0*]); *p1* = *make-polygonal-path* (*x* # *vts2* @ *?c3-rev-vts*);
p2 = *make-polygonal-path* (*vts1* @ *?c3-vts* @ *vts3* @ [*vts* ! *0*])
in is-polygon-cut-path (*vts* @ [*vts* ! *0*]) *cutpath* $\wedge$
polygon p $\wedge$
polygon p1 $\wedge$
polygon p2 $\wedge$
path-inside p1 ∩ *path-inside p2* = {} $\wedge$
path-inside p1 ∪ *path-inside p2* ∪ (*path-image cutpath* − {*x*, *y*}) = *path-inside p*
$\wedge$ ((*path-image p1*) − (*path-image* (*cutpath*))) ∩ ((*path-image p2*) − (*path-image* (*cutpath*))) = {} $\wedge$
path-image p = ((*path-image p1*) − (*path-image* (*cutpath*))) ∪ ((*path-image p2*) − (*path-image* (*cutpath*))) ∪ {*x*, *y*})
using *second-part-helper vts-relation p-is*
by (*metis self-append-conv2*)

show *?thesis*
unfolding *is-polygon-split-path-def*[*of vts 0 i cutvts*]
using *first-part second-part*
by (*smt* (*verit*, *ccfv-threshold*) *append-Cons append-Nil cutpath rev.simps*(*2*) *rev-append rev-is-Nil-conv*)
qed

lemma *polygon-cut-path-to-split-path*:
fixes *p* :: *R-to-R2*
assumes *polygon p*
p = *make-polygonal-path* (*vts* @ [*vts* ! *0*])
is-polygon-cut-path (*vts* @ [*vts*!*0*]) *cutpath*
vts1 ≡ (*take i vts*)
vts2 ≡ (*take* (*j* − *i* − *1*) (*drop* (*Suc i*) *vts*))
vts3 ≡ *drop* (*j* − *i*) (*drop* (*Suc i*) *vts*)
x ≡ *vts* ! *i*
y ≡ *vts* ! *j*
cutpath = *make-polygonal-path* ([*x*] @ *cutvts* @ [*y*])
i < *length vts* ∧ *j* < *length vts* ∧ *i* < *j*
p1 ≡ *make-polygonal-path* (*x*#(*vts2*@([*y*] @ (*rev cutvts*) @ [*x*]))) **and**
p2 ≡ *make-polygonal-path* (*vts1* @ ([*x*] @ *cutvts* @ [*y*]) @ *vts3* @ [(*vts1* @ [*x*]) ! *0*])
shows *is-polygon-split-path vts i j cutvts*
proof−
let *?poly-vts-rot* = *rotate-polygon-vertices* (*vts* @ [*vts* ! *0*]) *i*
let *?vts-rot* = *butlast ?poly-vts-rot*
let *?p-rot* = *make-polygonal-path ?poly-vts-rot*
let *?i-rot* = *j* − *i*
have *rot-poly*: *polygon ?p-rot* **using** *assms*(*1*) *assms*(*2*) *rotation-is-polygon* **by** *blast*
have *i-rot*: *?i-rot* > *0* ∧ *?i-rot* < *length ?poly-vts-rot* − *1*
using *assms*(*10*) *rotate-polygon-vertices-same-length* **by** *fastforce*
have *vtsi*: *vts* ! *i* = *?poly-vts-rot* ! *0*
using *rotated-polygon-vertices*[*of ?poly-vts-rot vts* @ [*vts*!*0*] *i i*]
by (*metis* (*no-types*, *lifting*) *One-nat-def Suc-1 assms*(*10*) *diff-self-eq-0 hd-conv-nth last-snoc length-append-singleton less-imp-le-nat linorder-not-le not-less-eq-eq nth-append take-all-iff take-eq-Nil*)
have *vtsj*: *vts* ! *j* = *?poly-vts-rot* ! *?i-rot*
using *rotated-polygon-vertices*[*of ?poly-vts-rot vts* @ [*vts*!*0*] *i j*]
by (*smt* (*verit*, *ccfv-SIG*) *One-nat-def Suc-1 assms*(*10*) *butlast-snoc hd-append2 hd-conv-nth last-snoc leD length-append-singleton less-Suc-eq-le less-imp-le-nat not-less-eq-eq nth-butlast take-all-iff take-eq-Nil*)
have *is-polygon-cut-path ?poly-vts-rot cutpath*
proof−
have *?poly-vts-rot* ! *0* ≠ *?poly-vts-rot* ! *?i-rot*
using *assms*(*3*) **unfolding** *is-polygon-cut-path-def* **using** *vtsi vtsj*
using *append-Cons append-is-Nil-conv assms*(*7*) *assms*(*8*) *assms*(*9*) *last-appendR last-conv-nth polygon-pathfinish polygon-pathstart*
by *force*

moreover have {*?poly-vts-rot* ! *0*, *?poly-vts-rot* ! *?i-rot*} ⊆ *set* (*?poly-vts-rot* @ [*?poly-vts-rot* ! *0*])
using *assms*(*3*) **unfolding** *is-polygon-cut-path-def* **using** *i-rot vtsi vtsj* **by** *fastforce*
moreover have *path-image cutpath* ∩ *path-image ?p-rot* = {*?poly-vts-rot* ! *0*, *?poly-vts-rot* ! *?i-rot*}
using *polygon-vts-arb-rotation vtsi vtsj assms*(*3*) *is-polygon-cut-path-def*
by (*metis* (*no-types*, *lifting*) *append.assoc append-Cons assms*(*7*) *assms*(*8*) *assms*(*9*) *last-conv-nth nth-Cons-0 polygon-pathfinish polygon-pathstart snoc-eq-iff-butlast*)
moreover have *path-image cutpath* ∩ *path-inside* (*?p-rot*) ≠ {}
using *vtsi vtsj assms*(*3*) *polygon-vts-arb-rotation*
unfolding *is-polygon-cut-path-def path-inside-def* **by** *metis*
ultimately show *?thesis*
unfolding *is-polygon-cut-path-def*
using *rot-poly assms*(*3*) *is-polygon-cut-path-def rotate-polygon-vertices-same-set vtsi vtsj*
by (*metis polygon-vts-arb-rotation*)
qed
then have *rot-cut*: *is-polygon-cut-path* (*?vts-rot* @ [*?vts-rot*!*0*]) *cutpath*
by (*metis butlast-snoc rotate-polygon-vertices-def*)
have *rot-cut-butlast*: *make-polygonal-path ?poly-vts-rot* = *make-polygonal-path* (*?vts-rot* @ [*?vts-rot*!*0*])
by (*metis butlast-snoc rotate-polygon-vertices-def*)
have *split-rot*: *is-polygon-split-path ?vts-rot 0 ?i-rot cutvts*
using *rot-cut rot-cut-butlast*
by (*smt* (*verit*, *ccfv-SIG*) *assms*(*7*) *assms*(*8*) *assms*(*9*) *dual-order.strict-trans i-rot is-polygon-cut-path-def length-butlast nth-butlast polygon-cut-path-to-split-path-vtx0 vtsi vtsj*)

let *?vts1-rot* = *take 0 ?vts-rot*
let *?vts2-rot* = *take* (*j* − *i* − *0* − *1*) (*drop* (*Suc 0*) *?vts-rot*)
let *?vts3-rot* = *drop* (*j* − *i* − *0*) (*drop* (*Suc 0*) *?vts-rot*)
let *?x-rot* = *?vts-rot* ! *0*
let *?y-rot* = *?vts-rot* ! (*j* − *i*)
let *?p1-rot-vts* = *?x-rot* # *?vts2-rot* @ [*?y-rot*] @ (*rev cutvts*) @ [*?x-rot*]
let *?p1-rot* = *make-polygonal-path ?p1-rot-vts*
let *?p2-rot-vts* = *?vts1-rot* @ [*?x-rot*] @ *cutvts* @ [*?y-rot*] @ *?vts3-rot* @ [*?vts-rot* ! *0*]
let *?p2-rot* = *make-polygonal-path ?p2-rot-vts*

let *?p1-vts* = *x* # *vts2* @ [*y*] @ (*rev cutvts*) @ [*x*]
let *?p2-vts* = *vts1* @ [*x*] @ *cutvts* @ [*y*] @ *vts3* @ [(*vts1* @ [*x*]) ! *0*]

have *p2-firstlast*: *hd ?p2-vts* = *last ?p2-vts*
by (*metis* (*no-types*, *lifting*) *append-is-Nil-conv append-self-conv2 hd-append2 hd-conv-nth last-appendR last-snoc list.discI list.sel*(*1*))

have *length* (*drop* (*Suc i*) *vts*) = *length vts* − *i* − *1*
by *simp*

then have *len-prop*: *length* (*drop* (*Suc i*) *vts*) ≥ *j* − *i* − *1*
using *assms*(*9*) *assms*(*10*) *diff-le-mono less-or-eq-imp-le* **by** *presburger*
have *drop-take*: *rotate i vts* = *drop i vts* @ *take i vts*
using *rotate-drop-take*[*of i vts*] *assms*(*10*) *mod-less* **by** *presburger*
then have *drop-take-suc*: *drop* (*Suc 0*) (*rotate i vts*) = *drop* (*Suc i*) *vts* @ *take i vts*
using *assms*(*10*) **by** *simp*
then have *take* (*j* − *Suc i*) (*drop* (*Suc 0*) (*rotate i vts*)) = *take* (*j* − *Suc i*) (*drop* (*Suc i*) *vts*)
using *len-prop* **by** *force*
then have *vts2*: *take* (*j* − *i* − *0* − *1*) (*drop* (*Suc 0*) (*butlast* (*rotate-polygon-vertices* (*vts* @ [*vts* ! *0*]) *i*))) = *vts2*
using *assms*(*5*) **unfolding** *rotate-polygon-vertices-def*
by (*metis Suc-eq-plus1 butlast-snoc diff-diff-left diff-zero*)

have *xy*: *?x-rot* = *x* ∧ *?y-rot* = *y*
using *vtsi vtsj assms* **by** (*metis is-polygon-split-path-def nth-butlast split-rot*)

moreover have *path-image p* = *path-image ?p-rot*
using *assms*(*1*) *assms*(*2*) *polygon-vts-arb-rotation* **by** *auto*
moreover then have *path-inside p* = *path-inside ?p-rot* **unfolding** *path-inside-def* **by** *simp*

moreover have *?p1-rot-vts* = *?p1-vts* **using** *xy vts2* **by** *presburger*
moreover then have *path-image p1* = *path-image ?p1-rot* **using** *assms* **by** *argo*
moreover then have *path-inside p1* = *path-inside ?p1-rot* **unfolding** *path-inside-def* **by** *argo*
moreover have *polygon p1*
using *calculation split-rot assms*(*11*) **unfolding** *is-polygon-split-path-def*
by (*smt* (*verit*, *ccfv-SIG*) *vts2*)

moreover have *?p2-rot-vts* = *rotate-polygon-vertices ?p2-vts i*
proof−
have *butlast* (*vts1* @ [*x*] @ *cutvts* @ [*y*] @ *vts3* @ [(*vts1* @ [*x*]) ! *0*])
= *vts1* @ [*x*] @ *cutvts* @ [*y*] @ *vts3*
by (*simp add*: *butlast-append*)
also have *rotate i* ... = [*x*] @ *cutvts* @ [*y*] @ *vts3* @ *vts1*
using *assms*(*4*)
by (*metis* (*no-types*, *lifting*) *drop-take add-diff-cancel-right′ append.assoc assms*(*10*) *diff-diff-cancel length-append length-drop length-rotate less-imp-le-nat rotate-append*)
finally have *rotate-polygon-vertices ?p2-vts i* = [*x*] @ *cutvts* @ [*y*] @ *vts3* @ *vts1* @ [*x*]
unfolding *rotate-polygon-vertices-def* **by** *simp*
moreover have *?vts3-rot* = *vts3* @ *vts1*
using *assms*(*4*,*6*) **unfolding** *rotate-polygon-vertices-def*
by (*smt* (*verit*, *del-insts*) *One-nat-def Suc-diff-Suc Suc-leI drop-take-suc assms*(*10*) *butlast-snoc diff-is-0-eq diff-zero drop0 drop-append i-rot le-add-diff-inverse len-prop length-drop nat-less-le*)

ultimately show *?thesis* **by** (*simp add*: *xy*)
qed
moreover then have *polygon p2*
using *unrotation-is-polygon*[*of ?p2-vts i p2*] *split-rot assms*(*12*) *p2-firstlast*
unfolding *is-polygon-split-path-def*
by (*smt* (*verit*) *append.assoc*)
moreover then have *path-image p2* = *path-image* (*?p2-rot*)
using *assms*(*12*) *polygon-vts-arb-rotation calculation* **by** *auto*
moreover then have *path-inside p2* = *path-inside ?p2-rot* **unfolding** *path-inside-def*
by *presburger*

ultimately show *is-polygon-split-path vts i j cutvts*
using *split-rot* **unfolding** *is-polygon-split-path-def*
using *One-nat-def assms bot-nat-0.not-eq-extremum butlast-snoc hd-append2 hd-conv-nth hd-take le-add2 length-0-conv length-Cons length-append length-butlast nth-append-length rot-cut-butlast rotate-polygon-vertices-same-length take-eq-Nil*
by (*smt* (*verit*) *append.assoc butlast-conv-take have-wraparound-vertex is-polygon-cut-path-def rotate-polygon-vertices-same-set*)
qed

lemma *good-polygonal-path-implies-polygon-split-path*:
assumes *polygon p*
assumes *p* = *make-polygonal-path* (*vts* @ [*vts*!*0*])
assumes *good-polygonal-path v1 cutvts v2* (*vts* @ [*vts*!*0*])
assumes *i* < *length vts* ∧ *j* < *length vts*
assumes *vts* ! *i* = *v1*
assumes *vts* ! *j* = *v2*
assumes *i* < *j*
shows *is-polygon-split-path vts i j cutvts*
proof−
let *?cutpath* = *make-polygonal-path* ([*v1*] @ *cutvts* @ [*v2*])
let *?p-path* = *make-polygonal-path* (*vts* @ [*vts*!*0*])
have *linepath-subset*: *path-image ?cutpath* ⊆ *path-inside ?p-path* ∪ {*v1*, *v2*}
using *assms*(*3*) **unfolding** *good-polygonal-path-def* **by** *meson*
have *linepath-ends*: *pathstart ?cutpath* = *v1* ∧ *pathfinish ?cutpath* = *v2*
using *polygon-pathfinish polygon-pathstart* **by** *force*
then have *vs-subset1*: {*v1*, *v2*} ⊆ *path-image ?cutpath*
using *vertices-on-path-image* **by** *fastforce*
have *vs-subset2*: {*v1*, *v2*} ⊆ *path-image* (*make-polygonal-path* (*vts* @ [*vts* ! *0*]))
using *assms*(*4*−*6*) *vertices-on-path-image*[*of vts*]
using *vertices-on-path-image* **by** *fastforce*
have *path-inside ?p-path* ∩ *path-image ?p-path* = {}
using *inside-outside-polygon*[*OF assms*(*1*)] *assms*(*2*) **unfolding** *inside-outside-def*
by *blast*
then have *linepath-path*: *path-image ?cutpath* ∩ *path-image* (*make-polygonal-path* (*vts* @ [*vts* ! *0*])) = {*v1*, *v2*}
using *linepath-subset vs-subset1 vs-subset2*
by *blast*
have *?cutpath* (*5* / *10*) ∈ *path-image ?cutpath*

unfolding *path-image-def* **by** *auto*
have *v1-neq-v2*: *v1* ≠ *v2*
using *assms*(*3*) **unfolding** *good-polygonal-path-def*
by *fastforce*
have *not-v1*: *?cutpath* (*0.5*::*real*) = *v1* ⟹ *False*
proof −
assume ∗: *?cutpath* (*0.5*::*real*) = *v1*
then have *?cutpath* (*0.5*::*real*) = *?cutpath 0*
using *linepath-ends* **unfolding** *pathstart-def* **by** *simp*
moreover have *loop-free ?cutpath* **using** *assms* **unfolding** *good-polygonal-path-def*
by *metis*
ultimately show *False* **unfolding** *loop-free-def* **by** *fastforce*
qed
have *not-v2*: *?cutpath* (*0.5*::*real*) = *v2* ⟹ *False*
proof−
assume ∗: *?cutpath* (*0.5*::*real*) = *v2*
then have *?cutpath* (*0.5*::*real*) = *?cutpath 1*
using *linepath-ends* **unfolding** *pathfinish-def* **by** *simp*
moreover have *loop-free ?cutpath* **using** *assms* **unfolding** *good-polygonal-path-def*
by *metis*
ultimately show *False* **unfolding** *loop-free-def* **by** *fastforce*
qed
then have *?cutpath* (*0.5*::*real*) ≠ *v1* ∧ *?cutpath* (*0.5*::*real*) ≠ *v2*
using *not-v1 not-v2* **by** *auto*
then have *linepath-inside*: *path-image ?cutpath* ∩ *path-inside* (*make-polygonal-path*
(*vts* @ [*vts* ! *0*])) ≠ {}
using *linepath-subset*
using ‹*?cutpath* (*5* / *10*) ∈ *path-image ?cutpath*› **by** *blast*
have *is-polygon-cut-path* (*vts* @ [*vts*!*0*]) *?cutpath*
using *assms*(*3*) *assms*(*1*−*2*) **unfolding** *good-polygonal-path-def is-polygon-cut-path-def*
using *linepath-path linepath-inside*
by (*metis linepath-ends make-polygonal-path-gives-path simple-path-def*)
then show *?thesis* **using** *polygon-cut-path-to-split-path assms* **by** *blast*
qed

lemma *good-path-iff*:
good-linepath a b vts ⟷ *good-polygonal-path a* [] *b vts*
unfolding *good-linepath-def good-polygonal-path-def*
using *linepath-loop-free* **by** *auto*

lemma *polygon-cut-iff*: *is-polygon-cut* (*vts* @ [*vts*!*0*]) (*vts*!*i*) (*vts*!*j*)
⟷ *is-polygon-cut-path* (*vts* @ [*vts*!*0*]) (*linepath* (*vts*!*i*) (*vts*!*j*))
unfolding *is-polygon-cut-def is-polygon-cut-path-def*
by (*metis pathfinish-linepath pathstart-linepath simple-path-linepath*)

lemma *polygon-split-iff*: *is-polygon-split vts i j* ⟷ *is-polygon-split-path vts i j* []
unfolding *is-polygon-split-def is-polygon-split-path-def*

by (*smt* (*verit*, *ccfv-threshold*) *append-Cons append-Nil make-polygonal-path.simps*(*3*) *polygon-cut-iff rev.simps*(*1*))

lemma *polygon-cut-to-split-vtx0*:
fixes *p* :: *R-to-R2*
assumes *polygon-p*: *polygon p* **and**
i-gt: *i* > *0* **and**
i-lt: *i* < *length vts* **and**
p-is: *p* = *make-polygonal-path* (*vts* @ [*vts* ! *0*]) **and**
have-cut: *is-polygon-cut* (*vts* @ [*vts*!*0*]) (*vts*!*0*) (*vts*!*i*)
shows *is-polygon-split vts 0 i*
using *have-cut i-gt i-lt p-is polygon-cut-path-to-split-path-vtx0 polygon-cut-iff polygon-p polygon-split-iff*
by *force*

lemma *polygon-cut-to-split*:
fixes *p* :: *R-to-R2*
assumes *is-polygon-cut* (*vts* @ [*vts*!*0*]) (*vts*!*i*) (*vts*!*j*)
i < *length vts* ∧ *j* < *length vts* ∧ *i* < *j*
shows *is-polygon-split vts i j*
by (*metis append-Cons append-Nil assms is-polygon-cut-def make-polygonal-path.simps*(*3*) *polygon-cut-path-to-split-path polygon-cut-iff polygon-split-iff*)

lemma *good-linepath-implies-polygon-split*:
assumes *polygon p*
assumes *p* = *make-polygonal-path* (*vts* @ [*vts*!*0*])
assumes *good-linepath v1 v2* (*vts* @ [*vts*!*0*])
assumes *i* < *length vts* ∧ *j* < *length vts*
assumes *vts* ! *i* = *v1*
assumes *vts* ! *j* = *v2*
assumes *i* < *j*
shows *is-polygon-split vts i j*
using *assms good-path-iff good-polygonal-path-implies-polygon-split-path polygon-split-iff*
by *auto*

end
theory *Triangle-Lemmas*
imports
Polygon-Convex-Lemmas
Integral-Matrix
Affine-Arithmetic.Floatarith-Expression
HOL−Analysis.Topology-Euclidean-Space
HOL−Analysis.Equivalence-Lebesgue-Henstock-Integration
HOL−Analysis.Inner-Product
HOL−Analysis.Line-Segment
HOL−Analysis.Convex-Euclidean-Space
HOL−Analysis.Change-Of-Vars

begin

20 Triangles

definition *elem-triangle* :: *real^2* ⇒ *real^2* ⇒ *real^2* ⇒ *bool* **where**
elem-triangle a b c ⟷
¬ *collinear* {*a*, *b*, *c*}
∧ *integral-vec a* ∧ *integral-vec b* ∧ *integral-vec c*
∧ {*x*. *x* ∈ *convex hull* {*a*, *b*, *c*} ∧ *integral-vec x*} = {*a*, *b*, *c*}

definition *triangle-mat* :: *real^2* ⇒ *real^2* ⇒ *real^2* ⇒ *real^2^2* **where**
triangle-mat a b c = *transpose* (*vector* [*b* − *a*, *c* − *a*])

definition *triangle-linear* :: *real^2* ⇒ *real^2* ⇒ *real^2* ⇒ (*real^2* ⇒ *real^2*) **where**
triangle-linear a b c = (λ*x*. (*triangle-mat a b c*) ∗*v x*)

definition *triangle-affine* :: *real^2* ⇒ *real^2* ⇒ *real^2* ⇒ (*real^2* ⇒ *real^2*) **where**
triangle-affine a b c = (λ*x*. *a* + (*triangle-mat a b c*) ∗*v x*)

abbreviation *unit-square* ≡
(*convex hull* {*vector* [*0*, *0*], *vector* [*0*, *1*], *vector* [*1*, *1*], *vector* [*1*, *0*]})::((*real^2*) *set*)

abbreviation *unit-triangle* ≡
(*convex hull* {*vector* [*0*, *0*], *vector* [*1*, *0*], *vector* [*0*, *1*]})::((*real^2*) *set*)

abbreviation *unit-triangle′* ≡
(*convex hull* {*vector* [*1*, *1*], *vector* [*1*, *0*], *vector* [*0*, *1*]})::((*real^2*) *set*)

lemma *triangle-inside-is-convex-hull-interior*:
assumes *polygon-of p* [*a*, *b*, *c*, *a*]
shows *path-inside p* = *interior* (*convex hull* {*a*, *b*, *c*})
proof−
have *path-image p* = *closed-segment a b* ∪ *closed-segment b c* ∪ *closed-segment c a*
proof−
have *path-image* (*linepath a b*) = *closed-segment a b* **by** *simp*
moreover have *path-image* (*linepath b c*) = *closed-segment b c* **by** *simp*
moreover have *path-image* (*linepath c a*) = *closed-segment c a* **by** *simp*
moreover have *path-image p* = *path-image* (*linepath a b*) ∪ *path-image* (*linepath b c*) ∪ *path-image* (*linepath c a*)
using *calculation assms*(*1*) **unfolding** *polygon-of-def make-polygonal-path.simps*
by (*simp add*: *path-image-join sup-assoc*)
ultimately show *?thesis* **by** *simp*
qed
moreover have *DIM*((*real*, *2*) *vec*) = *2* **by** *simp*
ultimately show *?thesis* **using** *inside-of-triangle*[*of a b c*] **unfolding** *path-inside-def* **by** *presburger*
qed

lemma *triangle-is-convex*:
assumes p = *make-triangle a b c* **and** ¬ *collinear* {a, b, c}
shows *convex* (*path-inside p*) (**is** *convex ?s*)
using *triangle-inside-is-convex-hull-interior assms*(*1*) *assms*(*2*)
using *make-triangle-def polygon-of-def triangle-is-polygon*
by *auto*

lemma *affine-comp-linear-trans*: *triangle-affine a b c* = (λx. x + a) ∘ (*triangle-linear a b c*)
apply (*simp add*: *triangle-affine-def triangle-linear-def*)
by *auto*

lemma *triangle-linear-der*:
fixes *a b c* :: *real^2*
defines T ≡ *triangle-linear a b c*
shows (*T has-derivative T*) (*at x*)
proof−
have *linear T* **using** *T-def* **by** (*simp add*: *triangle-linear-def*)
then have *bounded-linear T* **by** (*simp add*: *linear-linear*)
thus *?thesis* **using** *bounded-linear-imp-has-derivative* **by** *blast*
qed

lemma *triangle-affine-der*:
fixes *a b c* :: *real^2*
assumes S ∈ *sets lebesgue* **and** x ∈ S
defines A ≡ *triangle-affine a b c* **and** T ≡ *triangle-linear a b c*
shows x ∈ S ⟹ (*A has-derivative T*) (*at x within S*)
proof−
assume *xin*: x ∈ S
let *?trans* = λx::*real^2*. x + a
have *comp*: (*?trans* ∘ T) = (λx. (*T x*) + a)
by *auto*
have ∀ x. A x = (*?trans* ∘ T) x **unfolding** *A-def T-def* **using** *affine-comp-linear-trans*
by *auto*
moreover then have *Ax-is*: (⋀x. x ∈ S ⟹ A x = ((λx. x + a) ∘ T) x)
by *auto*
moreover have *trans-der*: (*?trans has-derivative id*) (*at x within S*)
by (*metis* (*full-types*) *add.commute assms*(*2*) *eq-id-iff has-derivative-transform shift-has-derivative-id*)
moreover have *Tder*: (*T has-derivative T*) (*at x within S*) **using** *triangle-linear-der*
by (*simp add*: *T-def bounded-linear-imp-has-derivative triangle-linear-def*)
moreover have *comp-der*: ((*?trans* ∘ T) *has-derivative T*) (*at x within S*)
using *has-derivative-add-const*[*OF Tder*] *comp*
by *simp*
ultimately show (*A has-derivative T*) (*at x within S*)
using *triangle-affine-def triangle-linear-def affine-comp-linear-trans o-apply add.commute vector-derivative-chain-within assms*(*2*) *has-derivative-add-const has-derivative-transform A-def T-def*
by *force*

qed

lemma *triangle-linear-inj*:
fixes *a b c* :: *real^2*
assumes ¬ *collinear* {*a*, *b*, *c*}
defines *L* ≡ *triangle-linear a b c*
shows *inj L*
proof−
let *?M* = *triangle-mat a b c*
let *?m-11* = (*b* − *a*)$*1*
let *?m-12* = (*c* − *a*)$*1*
let *?m-21* = (*b* − *a*)$*2*
let *?m-22* = (*c* − *a*)$*2*
have *det ?M* = *?m-11*∗*?m-22* − *?m-12*∗*?m-21*
unfolding *triangle-mat-def*
by (*metis det-2 det-transpose mult.commute vector-2*(*1*) *vector-2*(*2*))
moreover have *?m-11*∗*?m-22* ≠ *?m-12*∗*?m-21*
proof(*rule ccontr*)
assume ¬ *?m-11*∗*?m-22* ≠ *?m-12*∗*?m-21*
then have *eq*: *?m-11*∗*?m-22* = *?m-12*∗*?m-21* **by** *simp*
{ **assume** ∗: *?m-21* = *0* ∧ *?m-22* ≠ *0*
then have *?m-11* = *0* **using** *eq* **by** *simp*
then have *?m-11* = *0* ∧ *?m-21* = *0* **using** ∗ **by** *auto*
then have *b* − *a* = *0* **by** (*metis* (*no-types*, *opaque-lifting*) *exhaust-2 vec-eq-iff zero-index*)
then have *collinear* {*a*, *b*, *c*} **by** *simp*
then have *False* **using** *assms* **by** *fastforce*
} **moreover**
{ **assume** ∗: *?m-21* ≠ *0* ∧ *?m-22* = *0*
then have *?m-12* = *0* **using** *eq* **by** *simp*
then have *?m-12* = *0* ∧ *?m-22* = *0* **using** ∗ **by** *auto*
then have *c* − *a* = *0* **by** (*metis* (*no-types*, *opaque-lifting*) *exhaust-2 vec-eq-iff zero-index*)
then have *collinear* {*a*, *b*, *c*} **by** (*simp add*: *collinear-3-eq-affine-dependent*)
then have *False* **using** *assms* **by** *fastforce*
} **moreover**
{ **assume** ∗: *?m-21* = *0* ∧ *?m-22* = *0*
{ **assume** *?m-11* = *0*
then have *b* − *a* = *0* **using** ∗
by (*metis* (*no-types*, *opaque-lifting*) *exhaust-2 vec-eq-iff zero-index*)
then have *False* **using** *assms*(*1*) **by** *auto*
} **moreover**
{ **assume** *?m-11* ≠ *0*
then obtain *k* **where** *?m-12* = *k* ∗ *?m-11* **using** *nonzero-divide-eq-eq* **by** *blast*
moreover have *?m-22* = *k* ∗ *?m-21* **using** ∗ **by** *auto*
ultimately have *c* − *a* = *k* ∗*R* (*b* − *a*)
by (*smt* (*verit*, *del-insts*) *exhaust-2 real-scaleR-def vec-eq-iff vector-scaleR-component*)
then have *collinear* {*a*, *b*, *c*}

using *vec-diff-scale-collinear*[*of c a k b*] **by** (*simp add*: *insert-commute*)
then have *False* **using** *assms*(*1*) **by** *fastforce*
}
ultimately have *False* **using** *assms* **by** *fastforce*
} moreover
{ assume ∗: *?m-21* ≠ *0* ∧ *?m-22* ≠ *0*
then have *?m-11*/*?m-21* = *?m-12*/*?m-22* **using** *eq frac-eq-eq* **by** *blast*
then obtain *m* **where** *?m-11* = *m*∗*?m-12* ∧ *?m-21* = *m*∗*?m-22*
using *nonzero-divide-eq-eq* ∗
by (*metis* (*no-types*, *lifting*) *mult.commute times-divide-eq-left*)
then have *b* − *a* = *m* ∗*s* (*c* − *a*)
by (*smt* (*verit*, *del-insts*) *exhaust-2 vec-eq-iff vector-smult-component*)
then have *b* − *a* = *m* ∗*R* (*c* − *a*) **by** (*simp add*: *scalar-mult-eq-scaleR*)
then have *collinear* {*a*, *b*, *c*} **using** *vec-diff-scale-collinear* **by** *auto*
then have *False* **using** *assms* **by** *auto*
}
ultimately show *False* **by** *fastforce*
qed
ultimately have *det ?M* ≠ *0* **by** *linarith*
thus *?thesis* **by** (*simp add*: *L-def inj-matrix-vector-mult invertible-det-nz triangle-linear-def*)
qed

lemma *triangle-affine-inj*:
fixes *a b c* :: *real*^*2*
assumes ¬ *collinear* {*a*, *b*, *c*}
defines *A* ≡ *triangle-affine a b c*
shows *inj A*
proof−
have *inj* (*triangle-linear a b c*) **using** *triangle-linear-inj*[*of a b c*] *assms* **by** *auto*
moreover have *inj* (λ*x*. *x* + *a*) **by** *simp*
moreover have *A* = (λ*x*. *x* + *a*) ∘ (*triangle-linear a b c*)
by (*simp add*: *A-def affine-comp-linear-trans*)
ultimately show *?thesis* **using** *inj-compose* **by** *blast*
qed

lemma *triangle-linear-integrable*:
fixes *a b c* :: *real*^*2*
assumes *S* ∈ *lmeasurable*
defines *T* ≡ *triangle-linear a b c*
shows (λ*x*. *abs* (*det* (*matrix* (*T*)))) *integrable-on S* (**is** (λ*x*. *?c*) *integrable-on S*)
using *integrable-on-const*[*of S ?c*] *assms*(*1*) **by** *blast*

lemma *measure-differentiable-image-eq-affine*:
fixes *a b c* :: *real*^*2*
defines *A* ≡ *triangle-affine a b c* **and** *T* ≡ *triangle-linear a b c*
assumes *S* ∈ *lmeasurable* **and** ¬ *collinear* {*a*, *b*, *c*}
shows *measure lebesgue* (*A* ‘ *S*) = *integral S* (λ*x*. *abs* (*det* (*matrix T*)))
proof−

```
  have ⋀x. x ∈ S ⟹ (A has-derivative T) (at x within S)
    using triangle-affine-der A-def T-def assms(3) by blast
  moreover have inj-on A S
    using A-def assms(3) assms(4) triangle-affine-inj inj-on-subset by blast
  moreover have (λx. abs (det (matrix (T)))) integrable-on S
    by (simp add: T-def assms(3) triangle-linear-integrable)
  ultimately show ?thesis
    using measure-differentiable-image-eq[of - - λx. T] assms(3) by blast
qed

lemma triangle-affine-img:
  fixes a b c :: real^2
  defines A ≡ triangle-affine a b c
  shows convex hull {a, b, c} = A ‘ unit-triangle
proof−
  let ?O = (vector [0, 0])::real^2
  let ?e1 = (vector [1, 0])::real^2
  let ?e2 = (vector [0, 1])::real^2

  let ?translate-a = λx. x + a

  let ?T = triangle-linear a b c

  define al where al = ?T ?O
  define bl where bl = ?T ?e1
  define cl where cl = ?T ?e2

  have a: a = ?translate-a al
  proof−
    have al = ?O
      by (simp add: al-def mat-vec-mult-2 triangle-linear-def)
    then show ?thesis
     by (metis (no-types, opaque-lifting) add-0 mat-vec-mult-2 matrix-vector-mult-0
mult-zero-right zero-index)
  qed
  have b: b = ?translate-a bl
  proof−
    have col1: column 1 (triangle-mat a b c) = b − a
        by (metis column-transpose row-def triangle-mat-def vec-lambda-eta vec-
tor-2(1))
    then have bl = b − a
    using bl-def unfolding triangle-linear-def triangle-mat-def matrix-vector-mult-def
      using matrix-vector-mult-basis[of triangle-mat a b c 1]
      by (simp add: col1 axis-def bl-def mat-vec-mult-2 triangle-linear-def)
    then show ?thesis by simp
  qed
  have c: c = ?translate-a cl
  proof−
    have col2: column 2 (triangle-mat a b c) = c − a
```

by (*metis column-transpose row-def triangle-mat-def vec-lambda-eta vector-2*(*2*))
then have *cl* = *c* − *a*
using *cl-def* **unfolding** *triangle-linear-def triangle-mat-def matrix-vector-mult-def*
using *matrix-vector-mult-basis*[*of triangle-mat a b c 2*]
by (*simp add*: *col2 axis-def cl-def mat-vec-mult-2 triangle-linear-def*)
then show *?thesis* **by** *simp*
qed

have *linear ?T* **using** *triangle-linear-def* **by** *force*
then have *?T ‘ unit-triangle* = *convex hull* {*al*, *bl*, *cl*}
using *convex-hull-linear-image al-def bl-def cl-def* **by** *force*
also have *?translate-a ‘ ...* = *convex hull* {*a*, *b*, *c*}
using *a b c convex-hull-translation*[*of a* {*al*, *bl*, *cl*}]
by (*metis* (*no-types*, *lifting*) *add.commute image-cong image-empty image-insert*)
finally have *?translate-a ‘* (*?T ‘ unit-triangle*) = *convex hull* {*a*, *b*, *c*} **.**
moreover have *?translate-a* ∘ *?T* = *A* **unfolding** *A-def* **using** *affine-comp-linear-trans*
by *auto*
ultimately show *?thesis* **by** *fastforce*
qed

lemma *triangle-affine-e1-e2*:
fixes *a b c* :: *real*^*2*
defines *A* ≡ *triangle-affine a b c*
shows (*triangle-affine a b c*) (*vector* [*0*, *0*]) = *a*
(*triangle-affine a b c*) (*vector* [*1*, *0*]) = *b*
(*triangle-affine a b c*) (*vector* [*0*, *1*]) = *c*
proof−
let *?M* = *triangle-mat a b c*
let *?L* = *triangle-linear a b c*
let *?A* = *triangle-affine a b c*
let *?O* = (*vector* [*0*, *0*])::(*real*^*2*)
let *?e1* = (*vector* [*1*, *0*])::(*real*^*2*)
let *?e2* = (*vector* [*0*, *1*])::(*real*^*2*)

show *?A ?O* = *a*
unfolding *triangle-affine-def triangle-mat-def*
by (*metis* (*no-types*, *opaque-lifting*) *add.right-neutral diff-self mult-zero-right scaleR-left-diff-distrib transpose-matrix-vector vec-scaleR-2 vector-matrix-mult-0*)
show *?A ?e1* = *b*
proof−
have *?L ?e1* = *?M* ∗*v ?e1* **unfolding** *triangle-linear-def* **by** *blast*
also have ... = *vector* [*1*∗(*?M*$*1*$*1*) + *0*∗(*?M*$*1*$*2*), *1*∗(*?M*$*2*$*1*) + *0*∗(*?M*$*2*$*2*)]
unfolding *triangle-linear-def triangle-mat-def*
using *mat-vec-mult-2* **by** *force*
also have ... = *vector* [*1*∗(*b* − *a*)$*1* + *0*∗(*?M*$*1*$*2*), *1*∗(*b* − *a*)$*2* + *0*∗(*?M*$*2*$*2*)]
unfolding *triangle-mat-def transpose-def* **by** *simp*
also have ... = *vector* [(*b* − *a*)$*1*, (*b* − *a*)$*2*] **by** *argo*
also have ... = *b* − *a*

```
    by (smt (verit) exhaust-2 vec-eq-iff vector-2(1) vector-2(2))
   finally show ?thesis unfolding triangle-affine-def triangle-linear-def by simp
  qed
  show ?A ?e2 = c
  proof−
   have ?L ?e2 = ?M *v ?e2 unfolding triangle-linear-def by blast
  also have ... = vector [0*(?M$1$1) + 1*(?M$1$2), 0*(?M$2$1) + 1*(?M$2$2)]
    unfolding triangle-linear-def triangle-mat-def
    using mat-vec-mult-2 by force
   also have ... = vector [0*(?M$1$1) + 1*(c − a)$1, 0*(?M$2$1) + 1*(c −
a)$2]
    unfolding triangle-mat-def transpose-def by simp
   also have ... = vector [(c − a)$1, (c − a)$2] by argo
   also have ... = c − a
    by (smt (verit) exhaust-2 vec-eq-iff vector-2(1) vector-2(2))
   finally show ?thesis unfolding triangle-affine-def triangle-linear-def by simp
  qed
qed

lemma triangle-measure-integral-of-det:
  fixes a b c :: real^2
  defines S ≡ convex hull {a, b, c}
  assumes ¬ collinear {a, b, c}
  shows measure lebesgue S =
        integral unit-triangle (λ(x::real^2). abs (det (matrix (triangle-linear a b
c))))
proof−
  let ?A = triangle-affine a b c
  let ?T = triangle-linear a b c

  have bounded unit-triangle by (simp add: finite-imp-bounded-convex-hull)
  then have lmeasurable-S: unit-triangle ∈ lmeasurable
   using bounded-set-imp-lmeasurable measurable-convex by blast

  have S = ?A ‘ unit-triangle using S-def triangle-affine-img by blast
  then have measure lebesgue S = measure lebesgue (?A ‘ unit-triangle) by blast
  moreover have
   measure lebesgue (?A ‘ unit-triangle)
    = integral unit-triangle (λ(x::real^2). abs (det (matrix ?T)))
    using measure-differentiable-image-eq-affine[OF lmeasurable-S assms(2)] by
auto
  ultimately show ?thesis by auto
qed

lemma triangle-affine-preserves-interior:
  assumes A = triangle-affine a b c and L = triangle-linear a b c
  assumes ¬ collinear {a, b, c}
  shows A ‘ (interior S) = interior (A ‘ S)
proof−
```

let *?trans* = λx::*real^2*. *x* + *a*
have *linear L* **by** (*simp add*: *assms*(*2*) *triangle-linear-def*)
moreover have *surj L*
using *triangle-linear-inj*[*of a b c*] *linear-injective-imp-surjective*[*of L*] *assms calculation*
by *blast*
ultimately have *L*: *interior*(*L ‘ S*) = *L ‘* (*interior S*)
using *interior-surjective-linear-image* **by** *blast*
moreover have *interior*(*?trans ‘ S*) = *?trans ‘* (*interior S*)
using *interior-translation*
by (*metis* (*no-types*, *lifting*) *add.commute image-cong*)
moreover have *A* = *?trans* ∘ *L* **using** *assms triangle-affine-def triangle-linear-def* **by** *fastforce*
ultimately show *?thesis*
by (*smt* (*verit*, *del-insts*) *add.commute image-comp image-cong interior-translation*)
qed

lemma *triangle-affine-preserves-affine-hull*:
assumes *A* = *triangle-affine a b c*
assumes ¬ *collinear* {*a*, *b*, *c*}
shows *A ‘* (*affine hull S*) = *affine hull* (*A ‘ S*)
proof−
let *?L* = *triangle-linear a b c*
have *linear ?L* **by** (*simp add*: *triangle-linear-def*)
then have *?L ‘* (*affine hull S*) = *affine hull* (*?L ‘ S*)
by (*simp add*: *affine-hull-linear-image linear-linear*)
then show *?thesis*
unfolding *assms*(*1*) *triangle-affine-def*
by (*metis affine-hull-translation image-image triangle-linear-def*)
qed

lemma *triangle-measure-convex-hull-measure-path-inside-same*:
assumes *p-triangle*: *p* = *make-triangle a b c*
assumes *elem-triangle*: *elem-triangle a b c*
shows *measure lebesgue* (*convex hull* {*a*, *b*, *c*}) = *measure lebesgue* (*path-inside p*)
(**is** *measure lebesgue ?S* = *measure lebesgue ?I*)
proof−
have *bounded ?S* **by** (*simp add*: *finite-imp-bounded-convex-hull*)
then have *measure lebesgue* (*frontier ?S*) = *measure lebesgue ?S* − *measure lebesgue* (*interior ?S*)
using *measure-frontier*[*of ?S*] **by** *auto*
then have ... = *0*
by (*metis convex-convex-hull negligible-convex-frontier negligible-imp-measure0*)
moreover have *?I* = *interior ?S*
using *assms triangle-is-convex*
by (*metis* (*no-types*, *lifting*) *make-triangle-def convex-polygon-inside-is-convex-hull-interior empty-set insert-absorb2 insert-commute list.simps*(*15*) *elem-triangle-def triangle-is-polygon*)
ultimately show *?thesis* **by** *auto*

qed

lemma *on-triangle-path-image-cases*:
assumes p = *make-triangle a b c*
assumes $d \in$ *path-image p*
shows $d \in$ *path-image* (*linepath a b*) $\vee$ $d \in$ *path-image* (*linepath b c*) $\vee$ $d \in$ *path-image* (*linepath c a*)
using *assms* **unfolding** *make-triangle-def*
by (*metis make-polygonal-path.simps*(*3*) *make-polygonal-path.simps*(*4*) *not-in-path-image-join*)

lemma *on-triangle-frontier-cases*:
fixes *a b c* :: *real^2*
assumes ¬ *collinear* {*a*, *b*, *c*}
assumes $d \in$ *frontier* (*convex hull* {*a*, *b*, *c*})
shows $d \in$ *path-image* (*linepath a b*) $\vee$ $d \in$ *path-image* (*linepath b c*) $\vee$ $d \in$ *path-image* (*linepath c a*)
proof−
let *?p* = *make-triangle a b c*
have *polygon ?p* **by** (*simp add*: *assms*(*1*) *triangle-is-polygon*)
then have *path-image ?p* = *frontier* (*convex hull* {*a*, *b*, *c*})
unfolding *make-triangle-def*
by (*smt* (*verit*, *ccfv-threshold*) *assms*(*1*) *convex-polygon-frontier-is-path-image2 convex-polygon-is-convex-hull empty-set insert-absorb2 insert-commute list.simps*(*15*) *make-triangle-def polygon-convex-iff sup-commute triangle-is-convex*)
thus *?thesis* **using** *on-triangle-path-image-cases assms*(*2*) **by** *blast*
qed

lemma *triangle-path-image-subset-convex*:
assumes p = *make-triangle a b c*
shows *path-image p* ⊆ *convex hull* {*a*, *b*, *c*}
using *polygon-path-image-subset-convex polygon-at-least-3-vertices make-triangle-def*
by (*metis* (*no-types*, *lifting*) *assms empty-set insert-absorb2 insert-commute insert-iff length-pos-if-in-set list.simps*(*15*))

lemma *triangle-convex-hull*:
assumes p = *make-triangle a b c* **and** ¬ *collinear* {*a*, *b*, *c*}
shows *convex hull* {*a*, *b*, *c*} = (*path-image p*) ∪ (*path-inside p*)
using *triangle-is-convex*[*OF assms*(*1*) *assms*(*2*)]
by (*smt* (*z3*) *Un-commute assms*(*1*) *assms*(*2*) *closure-Un-frontier convex-closure convex-polygon-is-convex-hull insert-absorb2 insert-commute inside-outside-def inside-outside-polygon list.set*(*1*) *list.set*(*2*) *make-triangle-def triangle-is-polygon*)

end
theory *Unit-Geometry*
imports
HOL−*Analysis.Polytope*
Polygon-Jordan-Curve
Triangle-Lemmas

begin

21 Measure Setup

lemma *finite-convex-is-measurable*:
fixes $p :: (real\hat{}2)\ set$
assumes $p = convex\ hull\ l$ **and** *finite l*
shows $p \in sets\ lebesgue$
proof−
have *polytope p*
unfolding *polytope-def* **using** *assms* **by** *force*
hence *compact p* **using** *polytope-imp-compact* **by** *auto*
thus *?thesis* **using** *lmeasurable-compact* **by** *blast*
qed

lemma *unit-square-lebesgue*: $unit\text{-}square \in sets\ lebesgue$
using *finite-convex-is-measurable* **by** *auto*

lemma *unit-triangle-lebesgue*: $unit\text{-}triangle \in sets\ lebesgue$
using *finite-convex-is-measurable* **by** *auto*

lemma *unit-triangle-lmeasurable*: $unit\text{-}triangle \in lmeasurable$
by (*simp add*: *bounded-convex-hull bounded-set-imp-lmeasurable unit-triangle-lebesgue*)

22 Unit Triangle

lemma *unit-triangle-vts-not-collinear*:
$\neg$ *collinear* $\{(vector\ [0, 0])::real\hat{}2, vector\ [1, 0], vector\ [0, 1]\}$
(**is** $\neg$ *collinear* $\{?a, ?b, ?c\}$)
proof(*rule ccontr*)
assume $\neg \neg$ *collinear* $\{?a, ?b, ?c\}$
then have *collinear* $\{?a, ?b, ?c\}$ **by** *auto*
then obtain $u :: real\hat{}2$ **where** u: $u \neq 0 \wedge$
$(\forall x{\in}\{?a, ?b, ?c\}.\ \forall y{\in}\{?a, ?b, ?c\}.\ \exists c.\ x - y = c *_R u)$
by (*meson collinear*)
then obtain *c1 c2* **where** *c1*: $?b - ?a = c1 *_R u$ **and** *c2*: $?c - ?a = c2 *_R u$
by *blast*
then have $c1 *_R u = ?b$
by (*metis* (*no-types*, *opaque-lifting*) *diff-zero scaleR-eq-0-iff vector-2*(*1*) *vector-2*(*2*) *vector-minus-component vector-scaleR-component zero-neq-one*)
moreover have $c2 *_R u = ?c$ **using** *c1 c2 calculation* **by** *force*
ultimately have $u\$1 = 0 \wedge u\$2 = 0$
by (*metis scaleR-eq-0-iff vector-2*(*1*) *vector-2*(*2*) *vector-scaleR-component zero-neq-one*)
then have $u = 0$
by (*metis* (*mono-tags*, *opaque-lifting*) *exhaust-2 vec-eq-iff zero-index*)
moreover have $u \neq 0$ **using** u **by** *auto*
ultimately show *False* **by** *auto*

qed

lemma *unit-triangle-convex*:
assumes $p = (\textit{make-polygonal-path}\ [\textit{vector}\ [0,\ 0],\ \textit{vector}\ [1,\ 0],\ \textit{vector}\ [0,\ 1],\ \textit{vector}[0,\ 0]])$
(**is** $p = \textit{make-polygonal-path}\ [?O,\ ?e1,\ ?e2,\ ?O]$)
shows *convex* (*path-inside p*)
proof−
have ¬ *collinear* $\{?O,\ ?e1,\ ?e2\}$ **by** (*simp add*: *unit-triangle-vts-not-collinear*)
thus *?thesis* **using** *triangle-is-convex make-triangle-def assms* **by** *force*
qed

lemma *unit-triangle-char*:
shows $\textit{unit-triangle} = \{x.\ 0 \le x\ \$\ 1 \wedge 0 \le x\ \$\ 2 \wedge x\ \$\ 1 + x\ \$\ 2 \le 1\}$
(**is** *unit-triangle* = *?S*)
proof−
have *unit-triangle* ⊆ *?S*
proof(*rule subsetI*)
fix *x* **assume** $x \in \textit{unit-triangle}$
then obtain *a b c* **where**
$x = a *_R (\textit{vector}\ [0,\ 0]) + b *_R (\textit{vector}\ [1,\ 0]) + c *_R (\textit{vector}\ [0,\ 1])$
$\wedge\ a \ge 0 \wedge b \ge 0 \wedge c \ge 0 \wedge a + b + c = 1$
using *convex-hull-3* **by** *blast*
thus $x \in \{x.\ 0 \le x\ \$\ 1 \wedge 0 \le x\ \$\ 2 \wedge x\ \$\ 1 + x\ \$\ 2 \le 1\}$ **by** *simp*
qed
moreover have *?S* ⊆ *unit-triangle*
proof(*rule subsetI*)
fix *x* **assume** $x \in ?S$
then obtain *b c* **where** *bc*: $x\$1 = b \wedge x\$2 = c \wedge 0 \le b \wedge 0 \le c \wedge b + c \le 1$ **by** *blast*
moreover then obtain *a* **where** $a \ge 0 \wedge a + b + c = 1$ **using** *that*[*of* $1 - b - c$] **by** *argo*
moreover have $a *_R ((\textit{vector}\ [0,\ 0])::(\textit{real}\hat{}2)) = \textit{vector}\ [0,\ 0]$ **by** (*simp add*: *vec-scaleR-2*)
moreover have $x = (a *_R \textit{vector}\ [0,\ 0]) + (b *_R \textit{vector}\ [1,\ 0]) + (c *_R \textit{vector}\ [0,\ 1])$
using *segment-horizontal bc* **by** *fastforce*
ultimately show $x \in \textit{unit-triangle}$ **using** *convex-hull-3* **by** *blast*
qed
ultimately show *?thesis* **by** *blast*
qed

lemma *unit-triangle-interior-char*:
shows $\textit{interior unit-triangle} = \{x.\ 0 < x\ \$\ 1 \wedge 0 < x\ \$\ 2 \wedge x\ \$\ 1 + x\ \$\ 2 < 1\}$
(**is** *interior unit-triangle* = *?S*)
proof−
have *interior unit-triangle* ⊆ *?S*
proof(*rule subsetI*)

fix x **assume** $x \in$ *interior unit-triangle*
moreover have *DIM*(*real^2*) = *2* **by** *simp*
ultimately obtain a b c **where**
$x = a *_R$ (*vector* [*0*, *0*]) + $b *_R$ (*vector* [*1*, *0*]) + $c *_R$ (*vector* [*0*, *1*])
$\wedge$ $a > 0 \wedge b > 0 \wedge c > 0 \wedge a + b + c = 1$
using *interior-convex-hull-3-minimal*[*of* (*vector* [*0*, *0*])::(*real^2*) (*vector* [*1*, *0*])::(*real^2*) (*vector* [*0*, *1*])::(*real^2*)]
using *unit-triangle-vts-not-collinear*
by *auto*
thus $x \in \{x.\ 0 < x\ \$\ 1 \wedge 0 < x\ \$\ 2 \wedge x\ \$\ 1 + x\ \$\ 2 < 1\}$ **by** *simp*
qed
moreover have *?S* $\subseteq$ *interior unit-triangle*
proof(*rule subsetI*)
fix x **assume** $x \in$ *?S*
then obtain b c **where** *bc*: $x\$1 = b \wedge x\$2 = c \wedge 0 < b \wedge 0 < c \wedge b + c < 1$ **by** *blast*
moreover then obtain a **where** $a > 0 \wedge a + b + c = 1$ **using** *that*[*of 1 − b − c*] **by** *argo*
moreover have $a *_R$ ((*vector* [*0*, *0*])::(*real^2*)) = *vector* [*0*, *0*] **by** (*simp add*: *vec-scaleR-2*)
moreover have $x = (a *_R$ *vector* [*0*, *0*]) + ($b *_R$ *vector* [*1*, *0*]) + ($c *_R$ *vector* [*0*, *1*])
using *segment-horizontal bc* **by** *fastforce*
moreover have *DIM*(*real^2*) = *2* **by** *simp*
ultimately show $x \in$ *interior unit-triangle*
using *interior-convex-hull-3-minimal*[*of* (*vector* [*0*, *0*])::(*real^2*) (*vector* [*1*, *0*])::(*real^2*) (*vector* [*0*, *1*])::(*real^2*)]
using *unit-triangle-vts-not-collinear*
by *fast*
qed
ultimately show *?thesis* **by** *blast*
qed

lemma *unit-triangle-is-elementary*: *elem-triangle* (*vector* [*0*, *0*]) (*vector* [*1*, *0*]) (*vector* [*0*, *1*])
(**is** *elem-triangle ?a ?b ?c*)
proof−
let *?UT* = *unit-triangle*
have ¬ *collinear* {*?a*, *?b*, *?c*} **using** *unit-triangle-vts-not-collinear* **by** *auto*
moreover have *integral-vec ?a* $\wedge$ *integral-vec ?b* $\wedge$ *integral-vec ?c*
by (*simp add*: *integral-vec-def is-int-def*)
moreover have {$x \in$ *?UT*. *integral-vec x*} = {*?a*, *?b*, *?c*} (**is** *?UT-integral* = *?abc*)
proof−
have *?UT-integral* $\supseteq$ *?abc* **using** *calculation*(*2*) *hull-subset* **by** *fastforce*
moreover have *?UT-integral* $\subseteq$ *?abc*
proof −
have $\bigwedge x.\ x \in$ *unit-triangle* $\Longrightarrow$ *integral-vec x* $\Longrightarrow$ $x \neq$ *vector* [*0*, *0*] $\Longrightarrow$ $x \neq$ *vector* [*1*, *0*] $\Longrightarrow$ $x \neq$ *vector* [*0*, *1*] $\Longrightarrow$ *False*

```
    proof−
      fix x
      assume ∗: x ∈ unit-triangle
            integral-vec x
             x ≠ vector [0, 0]
            x ≠ vector [1, 0]
            x ≠ vector [0, 1]
      then have x-inset: x ∈{x. 0 ≤ x $ 1 ∧ 0 ≤ x $ 2 ∧ x $ 1 + x $ 2 ≤ 1}
        using unit-triangle-char by auto
      have x $ 1 = 1 ⟹ x $ 2 ≠ 0
        using ∗
        by (smt (verit, del-insts) exhaust-2 vec-eq-iff vector-2(1) vector-2(2))
      then have x $ 1 = 1 ⟹ x $ 1 + x $ 2 > 1 ∨ x $ 2 < 0
        using ∗(2) unfolding integral-vec-def is-int-def
        by linarith
      then have x1-not-1: x$1 = 1 ⟹ False
        using x-inset by simp
      have x $ 1 = 0 ⟹ x $ 2 ≠ 0 ∧ x $ 2 ≠ 1
        using ∗
        by (smt (verit, del-insts) exhaust-2 vec-eq-iff vector-2(1) vector-2(2))
      then have x $ 1 = 0 ⟹ x $ 1 + x $ 2 > 1 ∨ x $ 1 + x $ 2 < 0
        using ∗(2) unfolding integral-vec-def is-int-def
        by auto
      then have x1-not-0: x $ 1 = 0 ⟹ False
        using x-inset by simp
      have x1-not-lt0: x $ 1 < 0 ⟹ False
        using x-inset by auto
      have x1-not-gt1: x $ 1 > 1 ⟹ False
        using x-inset by auto
      then show False using x1-not-0 x1-not-1 x1-not-lt0 x1-not-gt1
        using ∗(2) unfolding integral-vec-def is-int-def
        by force
    qed
    then have ∃ x ∈ ?UT-integral. x ∉ ?abc ∧ integral-vec x ⟹ False
      by blast
    then show ?thesis by blast
  qed
  ultimately show ?thesis unfolding elem-triangle-def by auto
qed

lemma unit-triangles-same-area:
  measure lebesgue unit-triangle' = measure lebesgue unit-triangle
proof−
  let ?a = (vector [1, 1])::real^2
  let ?b = (vector [0, 1])::real^2
  let ?c = (vector [1, 0])::real^2
  let ?A = triangle-affine ?a ?b ?c
```

```
  let ?L = triangle-linear ?a ?b ?c
  have collinear-second-component: ⋀c::real^2. collinear {?a, ?b, c} ⟹ c $ 2 =
1
  proof −
    fix p
    assume collinear {?a, ?b, p}
    then obtain u where u-prop: ∀x∈{vector [1, 1], vector [0, 1], p}.
          ∀y∈{vector [1, 1], vector [0, 1], p}. ∃c. x − y = c *R u
      unfolding collinear-def by auto
    then have c-ab: ∃c. ?a − ?b = c *R u
      by blast
    then have u-2: u $ 2 = 0
      using vector-2
       by (metis cancel-comm-monoid-add-class.diff-cancel diff-zero scaleR-eq-0-iff
vector-minus-component vector-scaleR-component zero-neq-one)
    have u-1: u$1 ≠ 0
      using c-ab vector-2
     by (smt (z3) scaleR-right-diff-distrib vector-minus-component vector-scaleR-component)
    then have (∃c. ?a − p = c *R u) ∧ (∃c. ?b − p = c *R u)
      using u-prop by blast
    then show p $ 2 = 1
      using u-1 u-2
     by (metis eq-iff-diff-eq-0 scaleR-zero-right vector-2(2) vector-minus-component
vector-scaleR-component)
  qed
  have unit-triangle′ = convex hull {?a, ?b, ?c} by (simp add: insert-commute)
  then have ?A ‘ unit-triangle = unit-triangle′ using triangle-affine-img[of ?a ?b
?c] by argo
  moreover have abs (det (matrix ?L)) = 1
  proof−
    have matrix ?L = transpose (vector [?b − ?a, ?c − ?a])
      unfolding triangle-linear-def
      by (simp add: triangle-mat-def)
     also have det ... = det (vector [?b − ?a, ?c − ?a]) using det-transpose by
blast
    also have ... = (?b − ?a)$1 ∗ (?c − ?a)$2 − (?c − ?a)$1 ∗ (?b − ?a)$2
      using det-2 by (metis mult.commute vector-2(1) vector-2(2))
    finally show ?thesis by simp
  qed
  moreover have ¬ collinear {?a, ?b, ?c} using collinear-second-component vec-
tor-2 by force
 ultimately have measure lebesgue unit-triangle′ = integral unit-triangle (λ(x::real^2).
1)
    using triangle-measure-integral-of-det[of ?a ?b ?c]
   by (smt (verit, ccfv-SIG) Henstock-Kurzweil-Integration.integral-cong insert-commute)
  also have ... = measure lebesgue unit-triangle
    by (simp add: lmeasure-integral unit-triangle-lmeasurable)
  finally show ?thesis .
qed
```

23 Unit Square

lemma *convex-hull-4*:
  convex hull {*a*,*b*,*c*,*d*} = { *u* $*_R$ *a* + *v* $*_R$ *b* + *w* $*_R$ *c* + *t* $*_R$ *d* | *u v w t*. 0 ≤ *u* ∧ 0 ≤ *v* ∧ 0 ≤ *w* ∧ 0 ≤ *t* ∧ *u* + *v* + *w* + *t* = 1}
proof −
  have *fin*: *finite* {*a*,*b*,*c*,*d*} *finite* {*b*,*c*,*d*} *finite* {*c*,*d*} *finite* {*d*}
    by *auto*
  have ∗: ⋀*x y z w* ::*real*. *x* + *y* + *z* + *w* = 1 ⟷ *x* = 1 − *y* − *z* − *w*
    by (*auto simp*: *field-simps*)
  show *?thesis*
    unfolding *convex-hull-finite*[*OF fin*(1)]
    unfolding *convex-hull-finite-step*[*OF fin*(2)]
    unfolding *convex-hull-finite-step*[*OF fin*(3)]
    unfolding *convex-hull-finite-step*[*OF fin*(4)]
    unfolding ∗
    apply *auto*
     apply (*smt* (*verit*, *ccfv-threshold*) *add.commute diff-add-cancel diff-diff-eq*)
    subgoal for *v w t*
      apply (*rule exI* [**where** *x*=1 − *v* − *w* − *t*], *simp*)
      apply (*rule exI* [**where** *x*=*v*], *simp*)
      apply (*rule exI* [**where** *x*=*w*], *simp*)
      apply (*rule exI* [**where** *x*=λ*x*. *t*], *simp*)
      done
    done
qed

lemma *unit-square-characterization-helper*:
  fixes *a b* :: *real*
  assumes 0 ≤ *a* ∧ *a* ≤ 1 ∧ 0 ≤ *b* ∧ *b* ≤ 1 **and**
        a ≤ *b*
  obtains *u v w t* **where**
      vector [*a*, *b*] = *u* $*_R$ ((*vector* [0, 0])::*real*^2)
        + *v* $*_R$ (*vector* [0, 1])
        + *w* $*_R$ (*vector* [1, 1])
        + *t* $*_R$ (*vector* [1, 0])
      ∧ 0 ≤ *u* ∧ 0 ≤ *v* ∧ 0 ≤ *w* ∧ 0 ≤ *t* ∧ *u* + *v* + *w* + *t* = 1
proof−
  let *?a* = (*vector* [0, 0])::(*real*^2)
  let *?b* = (*vector* [0, 1])::(*real*^2)
  let *?c* = (*vector* [1, 1])::(*real*^2)
  let *?d* = (*vector* [1, 0])::(*real*^2)
  let *?w* = *a*
  let *?v* = *b* − *a*
  let *?u* = (1 − *?w* − *?v*)::*real*
  let *?t* = 0::*real*
  let *?T* = {*u* $*_R$ *?a* + *v* $*_R$ *?b* + *w* $*_R$ *?c* + *t* $*_R$ *?d* | *u v w t*. 0 ≤ *u* ∧ 0 ≤ *v* ∧ 0 ≤ *w* ∧ 0 ≤ *t* ∧ *u* + *v* + *w* + *t* = 1}
  have *?u* $*_R$ *?a* = 0

by (*smt* (*verit*, *del-insts*) *exhaust-2 scaleR-zero-right vec-eq-iff vector-2*(*1*) *vector-2*(*2*) *zero-index*)
moreover have *?w* $*_R$ *?c* = *vector* [*a*, *a*]
proof−
have (*?w* $*_R$ *?c*)$1 = *a* **by** *simp*
moreover have (*?w* $*_R$ *?c*)$2 = *a* **by** *simp*
ultimately show *?thesis* **by** (*smt* (*verit*) *vec-eq-iff exhaust-2 vector-2*(*1*) *vector-2*(*2*))
qed
moreover have *?v* $*_R$ *?b* = *vector* [*0*, *b* − *a*]
proof−
have (*?v* $*_R$ *?b*)$1 = *0* **by** *fastforce*
moreover have (*?v* $*_R$ *?b*)$2 = *b* − *a* **by** *simp*
ultimately show *?thesis* **by** (*smt* (*verit*) *vec-eq-iff exhaust-2 vector-2*(*1*) *vector-2*(*2*))
qed
ultimately have *?u* $*_R$ *?a* + *?v* $*_R$ *?b* + *?w* $*_R$ *?c* + *?t* $*_R$ *?d* = *vector* [*0*, *b* − *a*] + *vector* [*a*, *a*]
by *fastforce*
also have ... = *vector* [*a*, *b*]
by (*smt* (*verit*, *del-insts*) *diff-add-cancel exhaust-2 vec-eq-iff vector-2*(*1*) *vector-2*(*2*) *vector-add-component*)
finally have *vector* [*a*, *b*] = *?u* $*_R$ *?a* + *?v* $*_R$ *?b* + *?w* $*_R$ *?c* + *?t* $*_R$ *?d* **by** *presburger*
moreover have *0* ≤ *?u* ∧ *?u* ≤ *1* ∧ *0* ≤ *?v* ∧ *?v* ≤ *1* **using** *assms* **by** *simp*
moreover have *0* ≤ *?w* ∧ *?w* ≤ *1* ∧ *0* ≤ *?t* ∧ *?t* ≤ *1* **using** *assms* **by** *simp*
moreover have *?u* + *?v* + *?w* + *?t* = *1* **by** *argo*
ultimately show *?thesis* **using** *that*[*of ?u ?v ?w ?t*] **by** *blast*
qed

lemma *unit-square-characterization*:
unit-square = {*x*. *0* ≤ *x*$*1* ∧ *x*$*1* ≤ *1* ∧ *0* ≤ *x*$*2* ∧ *x*$*2* ≤ *1*} (**is** *unit-square* = *?S*)
proof−
let *?a* = (*vector* [*0*, *0*])::(*real*^*2*)
let *?b* = (*vector* [*0*, *1*])::(*real*^*2*)
let *?c* = (*vector* [*1*, *1*])::(*real*^*2*)
let *?d* = (*vector* [*1*, *0*])::(*real*^*2*)
let *?T* = {*u* $*_R$ *?a* + *v* $*_R$ *?b* + *w* $*_R$ *?c* + *t* $*_R$ *?d* | *u v w t*. *0* ≤ *u* ∧ *0* ≤ *v* ∧ *0* ≤ *w* ∧ *0* ≤ *t* ∧ *u* + *v* + *w* + *t* = *1*}
have *unit-square* = *?T* **using** *convex-hull-4* **by** *blast*
moreover have *?T* ⊆ *?S*
proof(*rule subsetI*)
fix *x*
assume *x* ∈ *?T*
then obtain *u v w t* **where** *x* = *u* $*_R$ *?a* + *v* $*_R$ *?b* + *w* $*_R$ *?c* + *t* $*_R$ *?d* **and**
0 ≤ *u* **and** *0* ≤ *v* **and** *0* ≤ *w* **and** *0* ≤ *t* **and** *u* + *v* + *w* + *t* = *1* **by** *auto*
moreover from *this* **have**
x$*1* = *u* ∗ *0* + *v* ∗ *0* + *w* ∗ *1* + *t* ∗ *1* ∧ *x*$*2* = *u* ∗ *0* + *v* ∗ *1* + *w* ∗ *1* +

$t * 0$ **by** *simp*
ultimately have $0 \leq x\$1 \wedge x\$1 \leq 1 \wedge 0 \leq x\$2 \wedge x\$2 \leq 1$ **by** *linarith*
thus $x \in ?S$ **by** *blast*
qed
moreover have $?S \subseteq ?T$
proof(*rule subsetI*)
fix $x :: real\hat{}2$
assume $*$: $x \in ?S$
{ **assume** $x\$1 < x\$2$
then have $x\$1 \leq x\$2$ **by** *fastforce*
then obtain $u\ v\ w\ t$ **where** $vector\ [x\$1, x\$2] = u *_R ?a + v *_R ?b + w *_R ?c + t *_R ?d \wedge 0 \leq u \wedge 0 \leq v \wedge 0 \leq w \wedge 0 \leq t \wedge u + v + w + t = 1$
using $*$ *unit-square-characterization-helper*[*of* $x\$1\ x\$2$] **by** *blast*
moreover have $x = vector\ [x\$1, x\$2]$
by (*smt* (*verit*, *ccfv-threshold*) *exhaust-2 vec-eq-iff vector-2*(*1*) *vector-2*(*2*))
ultimately have $x \in ?T$ **by** *force*
} **moreover**
{ **assume** $x\$1 \geq x\$2$
then obtain $u\ v\ w\ t$ **where** $**$: $vector\ [x\$2, x\$1] = u *_R ?a + v *_R ?b + w *_R ?c + t *_R ?d \wedge 0 \leq u \wedge 0 \leq v \wedge 0 \leq w \wedge 0 \leq t \wedge u + v + w + t = 1$
using $*$ *unit-square-characterization-helper*[*of* $x\$2\ x\$1$] **by** *blast*
have *x1*: $x\$1 = v + w$ **using** $**$
by (*smt* (*verit*, *ccfv-threshold*) *mult-cancel-left1 real-scaleR-def scaleR-zero-right vector-2*(*2*) *vector-add-component vector-scaleR-component*)
have *x2*: $x\$2 = w + t$ **using** $**$
by (*smt* (*verit*) *mult-cancel-left1 real-scaleR-def scaleR-zero-right vector-2*(*1*) *vector-add-component vector-scaleR-component*)
have $(u *_R ?a + t *_R ?b + w *_R ?c + v *_R ?d)\$1 = w + v$ **by** *auto*
moreover have $(u *_R ?a + t *_R ?b + w *_R ?c + v *_R ?d)\$2 = t + w$ **by** *fastforce*
ultimately have $u *_R ?a + t *_R ?b + w *_R ?c + v *_R ?d = vector\ [w + v, t + w]$
by (*smt* (*verit*) *vec-eq-iff exhaust-2 vector-2*(*1*) *vector-2*(*2*))
also have $... = x$ **using** *x1 x2*
by (*smt* (*verit*, *del-insts*) *add.commute exhaust-2 vec-eq-iff vector-2*(*1*) *vector-2*(*2*))
ultimately have $x \in ?T$
by (*smt* (*verit*, *ccfv-SIG*) $**$ *mem-Collect-eq*)
}
ultimately show $x \in ?T$ **by** *argo*
qed
ultimately show *?thesis* **by** *auto*
qed

lemma *e1e2-basis*:
defines $e1 \equiv (vector\ [1, 0])::(real\hat{}2)$ **and**
$e2 \equiv (vector\ [0, 1])::(real\hat{}2)$
shows $e1 = axis\ 1\ (1::real)$ **and** $e1 \in (Basis::((real\hat{}2)\ set))$ **and**
$e2 = axis\ 2\ (1::real)$ **and** $e2 \in (Basis::((real\hat{}2)\ set))$

proof−
have (*1*::*real*) ∈ *Basis* **by** *simp*
then have *axis 1* (*1*::*real*) ∈ (⋃ *i*. ⋃ *u*∈(*Basis*::(*real set*)). {*axis i u*}) **by** *blast*
moreover show *e1-axis*: *e1* = *axis 1* (*1*::*real*)
unfolding *axis-def vector-def e1-def* **by** *auto*
ultimately show *e1-basis*: *e1* ∈ (*Basis*::((*real*^*2*) *set*)) **by** *simp*

have (*1*::*real*) ∈ *Basis* **by** *simp*
then have *axis 1* (*1*::*real*) ∈ (⋃ *i*. ⋃ *u*∈(*Basis*::(*real set*)). {*axis i u*}) **by** *blast*
moreover show *e2-axis*: *e2* = *axis 2* (*1*::*real*)
unfolding *axis-def vector-def e2-def* **by** *auto*
ultimately show *e2-basis*: *e2* ∈ (*Basis*::((*real*^*2*) *set*)) **by** *simp*
qed

lemma *unit-square-cbox*: *unit-square* = *cbox* (*vector* [*0*, *0*]) (*vector* [*1*, *1*])
proof−
let *?O* = (*vector* [*0*, *0*])::(*real*^*2*)
let *?e1* = (*vector* [*1*, *0*])::(*real*^*2*)
let *?e2* = (*vector* [*0*, *1*])::(*real*^*2*)
let *?I* = (*vector* [*1*, *1*])::(*real*^*2*)
let *?cbox* = {*x*. ∀ *i*∈*Basis*. *?O* · *i* ≤ *x* · *i* ∧ *x* · *i* ≤ *?I* · *i*}

have *unit-square* = {*x*. *0* ≤ *x*$*1* ∧ *x*$*1* ≤ *1* ∧ *0* ≤ *x*$*2* ∧ *x*$*2* ≤ *1*} (**is** *unit-square* = *?S*)
using *unit-square-characterization* **by** *auto*
moreover have *?S* ⊆ *?cbox*
proof(*rule subsetI*)
fix *x*
assume ∗: *x* ∈ *?S*
have *?O* · *?e1* ≤ *x* · *?e1* ∧ *x* · *?e1* ≤ *?I* · *?e1*
using *e1e2-basis*
by (*smt* (*verit*, *del-insts*) ∗ *cart-eq-inner-axis mem-Collect-eq vector-2*(*1*))
moreover have *?O* · *?e2* ≤ *x* · *?e2* ∧ *x* · *?e2* ≤ *?I* · *?e2*
using *e1e2-basis*
by (*smt* (*verit*, *del-insts*) ∗ *cart-eq-inner-axis mem-Collect-eq vector-2*(*2*))
ultimately show *x* ∈ *?cbox*
by (*smt* (*verit*, *best*) ∗ *axis-index cart-eq-inner-axis exhaust-2 mem-Collect-eq vector-2*(*1*) *vector-2*(*2*))
qed
moreover have *?cbox* ⊆ *?S*
proof(*rule subsetI*)
fix *x* :: *real*^*2*
assume ∗: *x* ∈ *?cbox*
then have *0* ≤ *?e1* · *x* **using** *e1e2-basis*
by (*metis* (*no-types*, *lifting*) *cart-eq-inner-axis inner-commute mem-Collect-eq vector-2*(*1*))
moreover have *?e1* · *x* ≤ *1* **using** *e1e2-basis*
by (*smt* (*verit*, *ccfv-SIG*) ∗ *inner-axis inner-commute mem-Collect-eq real-inner-1-right vector-2*(*1*))

```
    moreover have 0 ≤ ?e2 • x
      by (metis (no-types, lifting) ∗ cart-eq-inner-axis e1e2-basis(3) e1e2-basis(4)
inner-commute mem-Collect-eq vector-2(2))
    moreover have ?e2 • x ≤ 1
      by (metis (no-types, lifting) ∗ cart-eq-inner-axis e1e2-basis(3) e1e2-basis(4)
inner-commute mem-Collect-eq vector-2(2))
    moreover have ?e1 • x = x$1
      by (simp add: cart-eq-inner-axis e1e2-basis inner-commute)
    moreover have ?e2 • x = x$2
      by (simp add: cart-eq-inner-axis e1e2-basis inner-commute)
    ultimately show x ∈ ?S by force
  qed
  ultimately show ?thesis unfolding cbox-def by order
qed

lemma unit-square-area: measure lebesgue unit-square = 1
proof−
  let ?e1 = (vector [1, 0])::(real^2)
  let ?e2 = (vector [0, 1])::(real^2)
  have unit-square = cbox (vector [0, 0]) (vector [1, 1]) (is unit-square = cbox
?O ?I)
    using unit-square-cbox by blast
  also have emeasure lborel ... = 1 using emeasure-lborel-cbox-eq
  proof−
    have ?I • ?e1 = (1::real)
      by (simp add: e1e2-basis(1) inner-axis′ inner-commute)
    moreover have ?I • ?e2 = (1::real) by (simp add: e1e2-basis(3) inner-axis′
inner-commute)
    ultimately have basis-dot: ∀ b ∈ Basis. ?I • b = 1
      by (metis (full-types) axis-inverse e1e2-basis(1) e1e2-basis(3) exhaust-2)

    have ?O • ?e1 ≤ ?I • ?e1 by (simp add: e1e2-basis(1) inner-axis)
    moreover have ?O • ?e2 ≤ ?I • ?e2 by (simp add: e1e2-basis(3) inner-axis)
    ultimately have ∀ b ∈ Basis. ?O • b ≤ ?I • b
     by (smt (verit, ccfv-threshold) axis-index cart-eq-inner-axis exhaust-2 insert-iff
vector-2(1) vector-2(2))
    then have emeasure lborel (cbox ?O ?I) = (∏ b∈Basis. (?I − ?O) • b)
      using emeasure-lborel-cbox-eq by auto
    also have ... = (∏ b∈Basis. ?I • b)
      by (smt (verit, del-insts) axis-index diff-zero euclidean-all-zero-iff exhaust-2
inner-axis real-inner-1-right vector-2(1) vector-2(2))
    also have ... = (∏ b∈Basis. (1::real)) using basis-dot by fastforce
    finally show ?thesis by simp
  qed
  finally have emeasure lborel unit-square = 1 .
  moreover have emeasure lborel unit-square = measure lebesgue unit-square
    by (simp add: emeasure-eq-measure2 unit-square-cbox)
  ultimately show ?thesis by fastforce
qed
```

24 Unit Triangle Area is 1/2

lemma *unit-triangle′-char*:
shows *unit-triangle′* = {*x*. *x* $ *1* ≤ *1* ∧ *x* $ *2* ≤ *1* ∧ *x* $ *1* + *x* $ *2* ≥ *1*}
proof −
let *?I* = (*vector* [*1*, *1*])::*real^2*
let *?e1* = (*vector* [*1*, *0*])::*real^2*
let *?e2* = (*vector* [*0*, *1*])::*real^2*
have *unit-triangle′* = {*u* *$_R$* *?I* + *v* *$_R$* *?e1* + *w* *$_R$* *?e2* | *u v w*. *0* ≤ *u* ∧ *0* ≤ *v* ∧ *0* ≤ *w* ∧ *u* + *v* + *w* = *1*}
using *convex-hull-3*[*of ?I ?e1 ?e2*] **by** *auto*
moreover have ⋀*u v w*. *u* *$_R$* *?I* + *v* *$_R$* *?e1* + *w* *$_R$* *?e2* = ((*vector* [*u* + *v*, *u* + *w*])::*real^2*)
proof−
fix *u v w* :: *real*

let *?v-e1* = ((*vector* [*v*, *0*])::*real^2*)
let *?w-e2* = ((*vector* [*0*, *w*])::*real^2*)
let *?u-I* = ((*vector* [*u*, *u*])::*real^2*)

have *u* *$_R$* *?I* = *?u-I* **using** *vec-scaleR-2* **by** *simp*
moreover have *v* *$_R$* *?e1* = *?v-e1* **using** *vec-scaleR-2* **by** *simp*
moreover have *w* *$_R$* *?e2* = *?w-e2* **using** *vec-scaleR-2* **by** *simp*
ultimately have *1*: *u* *$_R$* *?I* + *v* *$_R$* *?e1* + *w* *$_R$* *?e2* = *?u-I* + *?v-e1* + *?w-e2* **by** *argo*
moreover have (*?u-I* + *?v-e1* + *?w-e2*)$*1* = *u* + *v*
using *vector-add-component* **by** *simp*
moreover have (*?u-I* + *?v-e1* + *?w-e2*)$*2* = *u* + *w*
using *vector-add-component* **by** *simp*
ultimately have *?u-I* + *?v-e1* + *?w-e2* = ((*vector* [*u* + *v*, *u* + *w*])::*real^2*)
using *vector-2 exhaust-2* **by** (*smt* (*verit*, *del-insts*) *vec-eq-iff*)
thus *u* *$_R$* *?I* + *v* *$_R$* *?e1* + *w* *$_R$* *?e2* = ((*vector* [*u* + *v*, *u* + *w*])::*real^2*) **using** *1* **by** *argo*
qed
ultimately have *1*: *unit-triangle′* = {(*vector*[*u* + *v*, *u* + *w*])::*real^2* | *u v w*. *0* ≤ *u* ∧ *0* ≤ *v* ∧ *0* ≤ *w* ∧ *u* + *v* + *w* = *1*}
(**is** *unit-triangle′* = *?S*)
by *presburger*
have *unit-triangle′* = {(*vector*[*x*, *y*])::*real^2* | *x y*. *0* ≤ *x* ∧ *x* ≤ *1* ∧ *0* ≤ *y* ∧ *y* ≤ *1* ∧ *x* + *y* ≥ *1*}
(**is** *unit-triangle′* = *?T*)
proof−
have ⋀*x y*::*real*. ∃ *u v w*. *0* ≤ *u* ∧ *0* ≤ *v* ∧ *0* ≤ *w* ∧ *u* + *v* + *w* = *1* ∧ *x* = *u* + *v* ∧ *y* = *u* + *w*
⟹ *0* ≤ *x* ∧ *x* ≤ *1* ∧ *0* ≤ *y* ∧ *y* ≤ *1* ∧ *x* + *y* ≥ *1* **by** *force*
moreover have ∗: ⋀*x y*::*real*. *0* ≤ *x* ∧ *x* ≤ *1* ∧ *0* ≤ *y* ∧ *y* ≤ *1* ∧ *x* + *y* ≥ *1*
⟹ ∃ *u v w*. *0* ≤ *u* ∧ *0* ≤ *v* ∧ *0* ≤ *w* ∧ *u* + *v* + *w* = *1* ∧ *x* = *u* + *v* ∧ *y* = *u* + *w*
proof−

fix $x\ y :: real$
let $?u = y + x - 1$
let $?v = 1 - y$
let $?w = 1 - x$
assume $0 \leq x \wedge x \leq 1 \wedge 0 \leq y \wedge y \leq 1 \wedge 1 \leq x + y$
then have $0 \leq ?u \wedge 0 \leq ?v \wedge 0 \leq ?w \wedge ?u + ?v + ?w = 1 \wedge x = ?u + ?v \wedge y = ?u + ?w$ **by** *argo*
thus $\exists u\ v\ w.\ 0 \leq u \wedge 0 \leq v \wedge 0 \leq w \wedge u + v + w = 1 \wedge x = u + v \wedge y = u + w$ **by** *blast*
qed
ultimately have $\forall x\ y{::}real.\ ((\exists u\ v\ w.\ 0 \leq u \wedge 0 \leq v \wedge 0 \leq w \wedge u + v + w = 1 \wedge x = u + v \wedge y = u + w) \longleftrightarrow (0 \leq x \wedge x \leq 1 \wedge 0 \leq y \wedge y \leq 1 \wedge x + y \geq 1))$
by *metis*
then have $\forall z{::}real\hat{}2.\ ((\exists u\ v\ w.\ 0 \leq u \wedge 0 \leq v \wedge 0 \leq w \wedge u + v + w = 1 \wedge z\$1 = u + v \wedge z\$2 = u + w) \longleftrightarrow (0 \leq z\$1 \wedge z\$1 \leq 1 \wedge 0 \leq z\$2 \wedge z\$2 \leq 1 \wedge z\$1 + z\$2 \geq 1))$ **by** *presburger*
then have $\forall z{::}real\hat{}2.\ ((\exists u\ v\ w.\ 0 \leq u \wedge 0 \leq v \wedge 0 \leq w \wedge u + v + w = 1 \wedge z = vector\ [u + v,\ u + w]) \longleftrightarrow (\exists x\ y.\ 0 \leq x \wedge x \leq 1 \wedge 0 \leq y \wedge y \leq 1 \wedge x + y \geq 1 \wedge z = vector\ [x,\ y]))$
by (*smt* (*verit*) ∗)
moreover have $\forall z{::}real\hat{}2.\ z \in ?S \longleftrightarrow (\exists u\ v\ w.\ 0 \leq u \wedge 0 \leq v \wedge 0 \leq w \wedge u + v + w = 1 \wedge z = vector\ [u + v,\ u + w])$
by *blast*
moreover have $\forall z{::}real\hat{}2.\ z \in ?T \longleftrightarrow (\exists x\ y.\ 0 \leq x \wedge x \leq 1 \wedge 0 \leq y \wedge y \leq 1 \wedge x + y \geq 1 \wedge z = vector\ [x,\ y])$
by *blast*
ultimately have $?S = ?T$ **by** *auto*
then show *?thesis* **using** *1* **by** *auto*
qed
moreover have $\{x.\ 0 \leq x\$1 \wedge x\$1 \leq 1 \wedge 0 \leq x\$2 \wedge x\$2 \leq 1 \wedge x\$1 + x\$2 \geq 1\} \subseteq ?T$
proof(*rule subsetI*)
fix $z :: real\hat{}2$
assume ∗: $z \in \{x.\ 0 \leq x\$1 \wedge x\$1 \leq 1 \wedge 0 \leq x\$2 \wedge x\$2 \leq 1 \wedge x\$1 + x\$2 \geq 1\}$
then obtain $x\ y :: real$ **where** $z = vector[x,\ y] \wedge 0 \leq x$ **using** *forall-vector-2* **by** *fastforce*
moreover from *this* **have** $x \leq 1 \wedge 0 \leq y \wedge y \leq 1 \wedge x + y \geq 1$ **using** ∗ *vector-2*[*of x y*] **by** *simp*
ultimately show $z \in ?T$ **by** *blast*
qed
moreover have $?T \subseteq \{x.\ 0 \leq x\$1 \wedge x\$1 \leq 1 \wedge 0 \leq x\$2 \wedge x\$2 \leq 1 \wedge x\$1 + x\$2 \geq 1\}$
using *vector-2* **by** *force*
ultimately show *?thesis*
by (*smt* (*verit*, *best*) *Collect-cong subset-antisym*)

qed

lemma *unit-square-split-diag*:
shows *unit-square* = *unit-triangle* ∪ *unit-triangle′*
proof−
let *?S* = ({*vector* [*0*, *0*], *vector* [*0*, *1*], *vector* [*1*, *0*]})::((*real*^*2*) *set*)
let *?S′* = ({*vector* [*1*, *1*], *vector* [*0*, *1*], *vector* [*1*, *0*]})::((*real*^*2*) *set*)
have *unit-triangle* ∪ *unit-triangle′* ⊆ *convex hull* (*?S* ∪ *?S′*) **by** (*simp add*: *hull-mono*)
moreover have *convex hull* (*?S* ∪ *?S′*) ⊆ *unit-triangle* ∪ *unit-triangle′*
by (*smt* (*z3*) *Un-commute Un-left-commute Un-upper1 in-mono insert-is-Un mem-Collect-eq subsetI sup.idem unit-square-characterization unit-triangle-char unit-triangle′-char*)
moreover have *unit-square* = *convex hull* (*?S* ∪ *?S′*) **by** (*simp add*: *insert-commute*)
ultimately show *?thesis* **by** *blast*
qed

lemma *unit-triangle-INT-unit-triangle′-measure*:
measure lebesgue (*unit-triangle* ∩ *unit-triangle′*) = *0*
proof −
let *?e1* = (*vector* [*1*, *0*])::*real*^*2*
let *?e2* = (*vector* [*0*, *1*])::*real*^*2*
have *unit-triangle* ∩ *unit-triangle′* = {*x*::(*real*^*2*). *0* ≤ *x* $ *1* ∧ *x* $ *1* ≤ *1* ∧ *0* ≤ *x* $ *2* ∧ *x* $ *2* ≤ *1* ∧ *x* $ *1* + *x* $ *2* = *1*}
(**is** *unit-triangle* ∩ *unit-triangle′* = *?S*)
using *unit-triangle-char unit-triangle′-char*
by *auto*
also have ... = *path-image* (*linepath ?e2 ?e1*)
(**is** ... = *?p*)
proof−
have *?S* ⊆ *?p*
proof(*rule subsetI*)
fix *x* :: *real*^*2*
assume *x* ∈ *?S*
then have ∗: *0* ≤ *1* − *x*$*2* ∧ *x*$*2* = *1* − *x*$*1* ∧ *0* ≤ *x*$*2* ∧ *x*$*2* ≤ *1* **by** *simp*

have *x*$*2* ∗*R* *?e2* + *x*$*1* ∗*R* *?e1* = *vector*[*x*$*1*, *x*$*2*]
proof−
have (*x*$*1* ∗*R* *?e1*)$*1* = *x*$*1* **by** *simp*
moreover have (*x*$*1* ∗*R* *?e1*)$*2* = *0* **by** *auto*
moreover have (*x*$*2* ∗*R* *?e2*)$*1* = *0* **by** *auto*
moreover have (*x*$*2* ∗*R* *?e2*)$*2* = *x*$*2* **by** *fastforce*
ultimately have *x*$*1* ∗*R* *?e1* = *vector* [*x*$*1*, *0*] ∧ *x*$*2* ∗*R* *?e2* = *vector* [*0*, *x*$*2*]
by (*smt* (*verit*, *ccfv-SIG*) *exhaust-2 vec-eq-iff vector-2*(*1*) *vector-2*(*2*))
then have *x*$*1* ∗*R* *?e1* + *x*$*2* ∗*R* *?e2* = *vector* [*x*$*1*, *0*] + *vector* [*0*, *x*$*2*] **by** *auto*
moreover from *this* **have** (*x*$*1* ∗*R* *?e1* + *x*$*2* ∗*R* *?e2*)$*1* = *x*$*1* **by** *auto*
moreover from *calculation* **have** (*x*$*1* ∗*R* *?e1* + *x*$*2* ∗*R* *?e2*)$*2* = *x*$*2*

by *auto*
ultimately show *?thesis*
by (*smt* (*verit*) *add.commute exhaust-2 vec-eq-iff vector-2*(*1*) *vector-2*(*2*))
qed
also have ... = *x*
by (*smt* (*verit*, *best*) *exhaust-2 vec-eq-iff vector-2*(*1*) *vector-2*(*2*))
finally have *x*$*2* $*_R$ *?e2* + *x*$*1* $*_R$ *?e1* = *x* **.**
then have *x* = (λ*x*. (*1* − *x*) $*_R$ *?e2* + *x* $*_R$ *?e1*) (*x*$*1*) ∧ *x*$*1* ∈ {*0..1*}
using ∗ **by** *auto*
thus *x* ∈ *?p* **unfolding** *path-image-def linepath-def* **by** *fast*
qed
moreover have *?p* ⊆ *?S*
proof(*rule subsetI*)
fix *x*
assume ∗: *x* ∈ *?p*
then obtain *t* **where** ∗: *x* = (*1* − *t*) $*_R$ *?e2* + *t* $*_R$ *?e1* ∧ *t* ∈ {*0..1*}
unfolding *path-image-def linepath-def* **by** *blast*
moreover from *this* **have** *x*$*1* = *t* **by** *simp*
moreover from *calculation* **have** *x*$*2* = *1* − *t* **by** *simp*
moreover from *calculation* **have** *0* ≤ *t* ∧ *t* ≤ *1* ∧ *0* ≤ *1* − *t* ∧ *1* − *t* ≤ *1*
by *simp*
ultimately show *x* ∈ *?S* **by** *simp*
qed
ultimately show *?thesis* **by** *blast*
qed
also have *measure lebesgue ?p* = *0* **using** *linepath-has-measure-0* **by** *blast*
finally show *?thesis* **.**
qed

lemma *unit-triangle-area*: *measure lebesgue unit-triangle* = *1*/*2*
proof−
let *?μ* = *measure lebesgue*
have *?μ unit-square* = *?μ unit-triangle* + *?μ unit-triangle*′
using *unit-square-split-diag unit-triangle-INT-unit-triangle*′*-measure*
by (*simp add*: *finite-imp-bounded-convex-hull measurable-convex measure-Un3*)
thus *?thesis* **using** *unit-triangles-same-area unit-square-area* **by** *simp*
qed

end
theory *Elementary-Triangle-Area*
imports
Unit-Geometry

begin

25 Area of Elementary Triangle is 1/2

lemma *nonint-in-square-img-IMP-nonint-triangle-img*:
assumes *A* = *triangle-affine a b c*

assumes $x \in$ *unit-square*
assumes $\neg$ *integral-vec* x
assumes *integral-vec* $(A\ x)$
assumes *elem-triangle* $a\ b\ c$
obtains x' **where** $x' \in$ *unit-triangle* $\wedge \neg$ *integral-vec* $x' \wedge$ *integral-vec* $(A\ x')$
proof−
{ assume $x \in$ *unit-triangle*
then have *?thesis* **using** *assms that* **by** *blast*
} moreover
{ assume $*$: $x \notin$ *unit-triangle*
then have $x \notin \{x.\ 0 \leq x\ \$\ 1 \wedge 0 \leq x\ \$\ 2 \wedge x\ \$\ 1 + x\ \$\ 2 \leq 1\}$
using *unit-triangle-char* **by** *argo*
then have *x2x1-ge-1*: $x\$1 + x\$2 > 1$ **using** *assms*(2) *unit-square-characterization*
by *force*
let *?x'1* $= 1 - x\$1$
let *?x'2* $= 1 - x\$2$
let *?x'* $=$ *vector* $[?x'1,\ ?x'2]$
have *?x'1* $+$ *?x'2* ≤ 1 **using** *x2x1-ge-1* **by** *argo*
then have *?x'* $\in$ *unit-triangle*
using *unit-triangle-char assms*(2) *unit-square-characterization* **by** *auto*
moreover have $\neg$ *integral-vec ?x'*
proof−
have $\neg$ *is-int* $(x\$1) \vee \neg$ *is-int* $(x\$2)$ **using** *assms*(3) **unfolding** *integral-vec-def* **by** *blast*
then have $\neg$ *is-int* $(?x'1) \vee \neg$ *is-int* $(?x'2)$
using *is-int-minus*
by (*metis diff-add-cancel is-int-def minus-diff-eq of-int-1 uminus-add-conv-diff*)
thus *?thesis* **unfolding** *integral-vec-def* **by** *auto*
qed
moreover have *integral-vec* $(A$ *?x'*$)$
proof−
let *?L* $=$ *triangle-linear a b c*
have *A-comp*: $A = (\lambda x.\ x + a) \circ$ *?L* **by** (*simp add*: *affine-comp-linear-trans assms*(1))
then have *Lx-int*: *integral-vec* (*?L x*)
by (*smt* (*verit*, *del-insts*) *assms*(4) *assms*(5) *comp-apply diff-add-cancel diff-minus-eq-add integral-vec-minus integral-vec-sum elem-triangle-def*)

have *linear ?L* **by** (*simp add*: *triangle-linear-def*)
moreover have *?L ?x'* $=$ *?L* $($*vector* $[1,\ 1] - x)$
by (*simp add*: *mat-vec-mult-2 triangle-linear-def*)
ultimately have *?L ?x'* $=$ *?L* $($*vector* $[1,\ 1]) -$ *?L x* **by** (*simp add*: *linear-diff*)
moreover have *integral-vec* $($*?L* $($*vector* $[1,\ 1]))$
proof−
have *?L* $($*vector* $[1,\ 1]) =$ *vector* $[(b - a)\$1 + (c - a)\$1,\ (b - a)\$2 + (c - a)\$2]$
unfolding *triangle-linear-def triangle-mat-def transpose-def* **using** *mat-vec-mult-2*
by *simp*
also have $\ldots = (b - a) + (c - a)$

by (*smt* (*verit*, *del-insts*) *exhaust-2 vec-eq-iff vector-2*(*1*) *vector-2*(*2*) *vector-add-component*)
finally show *?thesis* **using** *assms*(*5*) **unfolding** *elem-triangle-def*
by (*metis ab-group-add-class.ab-diff-conv-add-uminus integral-vec-minus integral-vec-sum*)
qed
ultimately have *integral-vec* (*?L ?x'*)
using *Lx-int integral-vec-sum integral-vec-minus* **by** *force*
then show *?thesis* **using** *A-comp assms*(*5*) *integral-vec-sum elem-triangle-def* **by** *auto*
qed
ultimately have *?thesis* **using** *that* **by** *blast*
}
ultimately show *?thesis* **by** *blast*
qed

lemma *elem-triangle-integral-mat-bij*:
fixes *a b c* :: *real^2*
assumes *elem-triangle a b c*
defines *L* ≡ *triangle-mat a b c*
shows *integral-mat-bij L*
proof−
let *?A* = *triangle-affine a b c*

have *L*: *L* = *transpose* (*vector* [*b* − *a*, *c* − *a*]) (**is** *L* = *transpose* (*vector* [*?w1*, *?w2*]))
unfolding *triangle-mat-def L-def* **by** *auto*

have *integral-vec ?w1* ∧ *integral-vec ?w2*
by (*metis ab-group-add-class.ab-diff-conv-add-uminus assms*(*1*) *integral-vec-minus integral-vec-sum elem-triangle-def*)
then have *L-int-entries*: ∀ *i*∈{*1*, *2*}. ∀ *j*∈{*1*, *2*}. *is-int* (*L*$*i*$*j*)
by (*simp add*: *L-def triangle-mat-def Finite-Cartesian-Product.transpose-def integral-vec-def*)

have *L-integral*: *integral-mat L* **unfolding** *integral-mat-def*
proof(*rule allI*)
fix *v* :: *real^2*
show *integral-vec v* ⟶ *integral-vec* (*L* ∗*v v*)
proof(*rule impI*)
assume *v-int-assm*: *integral-vec v*
let *?Lv* = *L* ∗*v v*

have *?Lv*$*1* = *L*$*1*$*1* ∗ *v*$*1* + *L*$*1*$*2* ∗ *v*$*2* **by** (*simp add*: *mat-vec-mult-2*)
then have *Lv1-int*: *is-int* (*?Lv*$*1*)
using *L-int-entries v-int-assm is-int-sum is-int-mult* **by** (*simp add*: *integral-vec-def*)

have *?Lv*$*2* = *L*$*2*$*1* ∗ *v*$*1* + *L*$*2*$*2* ∗ *v*$*2* **by** (*simp add*: *mat-vec-mult-2*)

then have *Lv2-int*: *is-int* (*?Lv$2*)
using *L-int-entries v-int-assm is-int-sum is-int-mult* **by** (*simp add*: *integral-vec-def*)

show *integral-vec* (*L* ∗*v v*)
by (*simp add*: *Lv1-int Lv2-int integral-vec-def*)
qed
qed
moreover have *integral-mat-surj L*
unfolding *integral-mat-surj-def*
proof(*rule allI*)
fix *v* :: *real^2*
show *integral-vec v* ⟶ (∃ *w*. *integral-vec w* ∧ *L* ∗*v w* = *v*)
proof(*rule impI*)
assume ∗: *integral-vec v*
obtain *w* :: *real^2* **where** *w*: *L* ∗*v w* = *v*
using *triangle-linear-inj assms*(*1*) *full-rank-injective full-rank-surjective*
unfolding *elem-triangle-def L-def triangle-linear-def surj-def*
by (*smt* (*verit*, *best*) *iso-tuple-UNIV-I*)
moreover have *integral-vec w*
proof(*rule ccontr*)
assume ∗∗: ¬ *integral-vec w*
let *?w1* = *w$1*
let *?w2* = *w$2*
let *?w1'* = *w$1* − (*floor* (*w$1*))
let *?w2'* = *w$2* − (*floor* (*w$2*))
let *?w'* = (*vector* [*?w1'*, *?w2'*])::(*real^2*)
have *?w1'* ∈ {*0..1*} ∧ *?w2'* ∈ {*0..1*}
by (*metis add.commute add.right-neutral atLeastAtMost-iff floor-correct floor-frac frac-def of-int-0 real-of-int-floor-add-one-ge*)
then have *?w'* ∈ *unit-square* **using** *unit-square-characterization* **by** *auto*
moreover have ¬ *integral-vec ?w'*
by (*metis* ∗∗ *eq-iff-diff-eq-0 floor-frac floor-of-int frac-def integral-vec-def is-int-def of-int-0 vector-2*(*1*) *vector-2*(*2*))
moreover have *integral-vec* (*?A ?w'*)
proof−
have *?w'* = *vector* [*w$1*, *w$2*] − *vector* [*floor* (*w$1*), *floor* (*w$2*)]
(**is** *?w'* = *vector* [*w$1*, *w$2*] − *?floor-w*)
by (*smt* (*verit*, *del-insts*) *exhaust-2 list.simps*(*8*) *list.simps*(*9*) *vec-eq-iff vector-2*(*1*) *vector-2*(*2*) *vector-minus-component*)
then have *?w'* = *w* − *vector* [*floor* (*w$1*), *floor* (*w$2*)]
by (*smt* (*verit*, *del-insts*) *exhaust-2 vec-eq-iff vector-2*(*1*) *vector-2*(*2*) *vector-minus-component*)
moreover have *?A ?w'* = (*L* ∗*v ?w'*) + *a* **unfolding** *triangle-affine-def L-def* **by** *simp*
ultimately have *?A ?w'* = *v* − (*L* ∗*v ?floor-w*) + *a*
by (*simp add*: *matrix-vector-mult-diff-distrib w*)
moreover have *integral-vec v* ∧ *integral-vec a* ∧ *integral-vec* (*L* ∗*v ?floor-w*)
using ∗ *assms*(*1*) *L-integral integral-mat-integral-vec integral-vec-2*

unfolding *elem-triangle-def*
by *blast*
ultimately show *?thesis*
by (*metis ab-group-add-class.ab-diff-conv-add-uminus integral-vec-minus integral-vec-sum*)
qed
ultimately obtain w'' **where** w'': $w'' \in$ *unit-triangle* $\wedge \neg$ *integral-vec* w'' $\wedge$ *integral-vec* (*?A* w'')
using *nonint-in-square-img-IMP-nonint-triangle-img*[*of ?A a b c ?w'*] *assms*(*1*) **by** *blast*
moreover have *?A* $w'' \notin \{a, b, c\}$
proof−
have *inj ?A* **using** *assms*(*1*) *elem-triangle-def triangle-affine-inj* **by** *auto*
moreover have *?A* (*vector* [*0*, *0*]) = *a*
by (*metis* (*no-types*, *opaque-lifting*) *add.commute add-0 mat-vec-mult-2 matrix-vector-mult-0-right real-scaleR-def scaleR-zero-right triangle-affine-def zero-index*)
moreover have *?A* (*vector* [*1*, *0*]) = *b*
unfolding *triangle-affine-def triangle-mat-def transpose-def*
by (*metis* (*no-types*) *Finite-Cartesian-Product.transpose-def add.commute column-transpose diff-add-cancel e1e2-basis*(*1*) *matrix-vector-mult-basis row-def vec-lambda-eta vector-2*(*1*))
moreover have *?A* (*vector* [*0*, *1*]) = *c*
proof−
have (*?A* (*vector* [*0*, *1*]))$*1* = *c*$*1*
by (*metis L-def L add.commute column-transpose diff-add-cancel e1e2-basis*(*3*) *matrix-vector-mult-basis row-def triangle-affine-def vec-lambda-eta vector-2*(*2*))
moreover have (*?A* (*vector* [*0*, *1*]))$*2* = *c*$*2*
by (*metis add.commute column-transpose diff-add-cancel e1e2-basis*(*3*) *matrix-vector-mult-basis row-def triangle-affine-def triangle-mat-def vec-lambda-eta vector-2*(*2*))
ultimately show *?thesis* **by** (*smt* (*verit*, *ccfv-SIG*) *exhaust-2 vec-eq-iff*)
qed
moreover have $w'' \neq$ *vector* [*0*, *0*] $\wedge$ $w'' \neq$ *vector* [*0*, *1*] $\wedge$ $w'' \neq$ *vector* [*1*, *0*]
using w'' *elem-triangle-def unit-triangle-is-elementary* **by** *blast*
ultimately show *?thesis* **by** (*metis inj-eq insertE singletonD*)
qed
moreover have *?A* ‘ *unit-triangle* = *convex hull* {*a*, *b*, *c*}
using *triangle-affine-img* **by** *blast*
ultimately show *False* **using** *assms* **unfolding** *elem-triangle-def* **by** *blast*
qed
ultimately show $\exists w.$ *integral-vec* $w \wedge L *v w = v$ **by** *auto*
qed
qed
ultimately show *?thesis* **unfolding** *integral-mat-bij-def* **by** *auto*
qed

lemma *elem-triangle-measure-integral-of-1*:

fixes *a b c* :: *real^2*
defines *S* ≡ *convex hull* {*a*, *b*, *c*}
assumes *elem-triangle a b c*
shows *measure lebesgue S* = *integral unit-triangle* (λ(*x*::*real^2*). *1*)
proof−
let *?T* = *triangle-linear a b c*
have *integral-mat-bij* (*matrix ?T*) (**is** *integral-mat-bij ?T-mat*)
by (*simp add*: *assms*(*2*) *elem-triangle-integral-mat-bij triangle-linear-def*)
then have *abs* (*det ?T-mat*) = *1*
using *integral-mat-bij-det-pm1* **by** *fastforce*
thus *?thesis*
using *S-def assms*(*2*) *triangle-measure-integral-of-det elem-triangle-def* **by** *force*
qed

lemma *elem-triangle-area-is-half*:
fixes *a b c* :: *real^2*
assumes *elem-triangle a b c*
defines *S* ≡ *convex hull* {*a*, *b*, *c*}
shows *measure lebesgue S* = *1/2* (**is** *?S-area* = *1/2*)
proof−
have ¬ *collinear* {*a*, *b*, *c*} **using** *elem-triangle-def assms*(*1*) **by** *blast*
then have *measure lebesgue S* = *integral unit-triangle* (λ*x*::*real^2*. *1*)
using *S-def assms*(*1*) *elem-triangle-measure-integral-of-1* **by** *blast*
also have ... = *measure lebesgue unit-triangle*
using *unit-triangle-is-elementary elem-triangle-measure-integral-of-1 unit-triangle-area*
by *metis*
finally show *?thesis* **by** (*simp add*: *unit-triangle-area*)
qed

end
theory *Pick*
imports
Polygon-Splitting
Elementary-Triangle-Area
begin

26 Setup

26.1 Integral Points Cardinality Properties

lemma *bounded-finite*:
fixes *A*:: (*real^2*) *set*
assumes *bounded A*
shows *finite* {*x*::(*real^2*). *integral-vec x* ∧ *x* ∈ *A*} (**is** *finite ?A-int*)
proof−
obtain *M* **where** *M*: ∀ *x* ∈ *A*. *norm x* ≤ *M* **using** *assms bounded-def* **by** (*meson bounded-iff*)

let *?M-bounded-ints* = {*n*. *n* ∈ {−*M*..*M*} ∧ *is-int n*}

let *?M-bounded-int-vecs* = {*v*::(*real*^2). *v*$1 ∈ *?M-bounded-ints* ∧ *v*$2 ∈ *?M-bounded-ints*}

have ∀ *x*::(*real*^2). *norm* (*x*$1) ≤ *norm x* ∧ (*x*$2) ≤ *norm x*
by (*smt* (*verit*, *ccfv-threshold*) *Finite-Cartesian-Product.norm-nth-le real-norm-def*)
then have ∀ *x* ∈ *?A-int*. *norm* (*x*$1) ≤ *M* ∧ *norm* (*x*$2) ≤ *M*
using *M dual-order.trans Finite-Cartesian-Product.norm-nth-le* **by** *blast*
then have ∀ *x* ∈ *?A-int*. *x*$1 ∈ *?M-bounded-ints* ∧ *x*$2 ∈ *?M-bounded-ints*
using *integral-vec-def intervalE* **by** *auto*
then have ∀ *x* ∈ *?A-int*. *x* ∈ *?M-bounded-int-vecs* **by** *blast*
moreover have *finite ?M-bounded-int-vecs*
proof−
obtain *S* :: *int set* **where** *S*: *S* = {*n*. ∃ *m* ∈ *?M-bounded-ints*. *n* = *m*} ∧ (∀ *n* ∈ *S*. *norm n* ≤ *M*)
by (*simp add*: *abs-le-iff*)
then have *finite-S*: *finite S*
by (*metis infinite-int-iff-unbounded le-floor-iff linorder-not-less norm-of-int of-int-abs*)

have *finite-M-bounded-ints*: *finite ?M-bounded-ints*
proof−
let *?f* = λ*n*::*real*. *THE m*::*int*. *n* = *m*
have ∀ *n* ∈ *?M-bounded-ints*. ∃!*m*::*int*. *n* = *m* **using** *is-int-def* **by** *force*
moreover have *inj-on ?f ?M-bounded-ints* **using** *inj-on-def is-int-def* **by** *force*
moreover have *?f* ‘ *?M-bounded-ints* ⊆ *S* **using** *calculation S subsetI* **by** *auto*
ultimately show *?thesis* **using** *finite-imageD finite-S* **by** (*simp add*: *inj-on-finite*)
qed
show *?thesis*
proof−
let *?f* = λ*x*::(*real*^2). (*THE m*::*int*. *m* = *x*$1, *THE n*::*int*. *n* = *x*$2)
have *inj-on ?f ?M-bounded-int-vecs*
unfolding *inj-on-def*
proof *clarify*
fix *x y* :: *real*^2
assume *x1-int*: *is-int* (*x*$1)
assume *x2-int*: *is-int* (*x*$2)
assume *y1-int*: *is-int* (*y*$1)
assume *y2-int*: *is-int* (*y*$2)
assume *x1y1-int-eq*: (*THE m*. *real-of-int m* = *x*$1) = (*THE m*. *real-of-int m* = *y*$1)
assume *x2y2-int-eq*: (*THE n*. *real-of-int n* = *x*$2) = (*THE n*. *real-of-int n* = *y*$2)

have ∃!*m*. *m* = *x*$1
by *blast*
moreover have ∃!*n*. *n* = *y*$1
by *blast*

```
      moreover have (THE m. real-of-int m = x$1) = (THE m. real-of-int m =
y$1)
        using x1y1-int-eq by auto
       ultimately have x1y1: x$1 = y$1
        using x1-int y1-int is-int-def by auto

       have ∃!m. m = x$2
        by blast
       moreover have ∃!n. n = y$2
        by blast
      moreover have (THE m. real-of-int m = x$2) = (THE m. real-of-int m =
y$2)
        using x2y2-int-eq by auto
       ultimately have x2y2: x$2 = y$2
        using x2-int y2-int is-int-def by auto

       show x = y using x1y1 x2y2
        by (metis (no-types, lifting) exhaust-2 vec-eq-iff)
     qed
     moreover have ?f ‘ ?M-bounded-int-vecs ⊆ S × S
     proof(rule subsetI)
       fix mn
       assume mn ∈ ?f ‘ ?M-bounded-int-vecs
       then obtain v where v:
         v ∈ ?M-bounded-int-vecs ∧ ?f v = mn ∧ (∃!m. v$1 = m) ∧ (∃!n. v$2 =
n)
         using is-int-def by auto
       let ?m = fst mn
       let ?n = snd mn

       have ?m = (THE m::int. m = v$1) using v
         by (meson fstI)
       moreover have ∃! m::int. m = v$1 using v is-int-def
         by (metis (no-types, lifting) mem-Collect-eq of-int-eq-iff)
       ultimately have m-in-S: ?m ∈ S
         by (metis (mono-tags, lifting) S mem-Collect-eq theI′ v)

       have ?n = (THE n::int. n = v$2) using v
         by (meson sndI)
       moreover have ∃! n::int. n = v$2 using v is-int-def
         by (metis (no-types, lifting) mem-Collect-eq of-int-eq-iff)
       ultimately have n-in-S: ?n ∈ S
         by (metis (mono-tags, lifting) S mem-Collect-eq theI′ v)

       show mn ∈ S × S using m-in-S n-in-S v by auto
     qed
     ultimately show ?thesis
       by (meson finite-S finite-SigmaI finite-imageD finite-subset)
   qed
```

qed
ultimately show *?thesis*
by (*smt* (*verit*) *finite-subset subsetI*)
qed

lemma *finite-path-image*:
assumes *polygon p*
shows *finite* {*x*. *integral-vec x* ∧ *x* ∈ *path-image p*}
using *bounded-finite inside-outside-polygon*
unfolding *inside-outside-def*
by (*meson assms bounded-simple-path-image polygon-def*)

lemma *finite-path-inside*:
assumes *polygon p*
shows *finite* {*x*. *integral-vec x* ∧ *x* ∈ *path-inside p*}
using *bounded-finite inside-outside-polygon*
unfolding *inside-outside-def*
using *assms* **by** *presburger*

lemma *bounded-finite-inside*:
fixes *B*:: (*real^2*) *set*
assumes *simple-path p*
shows *bounded* (*path-inside p*)
using *assms*
by (*simp add*: *bounded-inside bounded-simple-path-image path-inside-def*)

lemma *finite-integral-points-path-image*:
assumes *simple-path p*
shows *finite* {*x*. *integral-vec x* ∧ *x* ∈ *path-image p*}
using *bounded-finite bounded-simple-path-image assms* **by** *blast*

lemma *finite-integral-points-path-inside*:
assumes *simple-path p*
shows *finite* {*x*. *integral-vec x* ∧ *x* ∈ *path-inside p*}
using *bounded-finite bounded-finite-inside assms* **by** *blast*

27 Pick splitting

lemma *pick-split-path-union-main*:
assumes *is-split*: *is-polygon-split-path vts i j cutvts*
assumes *vts1* = (*take i vts*)
assumes *vts2* = (*take* (*j* − *i* − *1*) (*drop* (*Suc i*) *vts*))
assumes *vts3* = *drop* (*j* − *i*) (*drop* (*Suc i*) *vts*)
assumes *x* = *vts*!*i*
assumes *y* = *vts*!*j*
assumes *cutpath* = *make-polygonal-path* (*x* # *cutvts* @ [*y*])
assumes *p*: *p* = *make-polygonal-path* (*vts*@[*vts*!*0*]) (**is** *p* = *make-polygonal-path ?p-vts*)
assumes *p1*: *p1* = *make-polygonal-path* (*x*#(*vts2* @ [*y*] @ (*rev cutvts*) @ [*x*]))

(**is** *p1 = make-polygonal-path ?p1-vts*)
assumes *p2*: *p2 = make-polygonal-path (vts1 @ ([x] @ cutvts @ [y]) @ vts3 @ [vts ! 0])* (**is** *p2 = make-polygonal-path ?p2-vts*)
assumes *I1*: *I1 = card {x. integral-vec x ∧ x ∈ path-inside p1}*
assumes *B1*: *B1 = card {x. integral-vec x ∧ x ∈ path-image p1}*
assumes *I2*: *I2 = card {x. integral-vec x ∧ x ∈ path-inside p2}*
assumes *B2*: *B2 = card {x. integral-vec x ∧ x ∈ path-image p2}*
assumes *I*: *I = card {x. integral-vec x ∧ x ∈ path-inside p}*
assumes *B*: *B = card {x. integral-vec x ∧ x ∈ path-image p}*
assumes *all-integral-vts*: *all-integral vts*
shows *measure lebesgue (path-inside p1) = I1 + B1/2 − 1*
⟹ measure lebesgue (path-inside p2) = I2 + B2/2 − 1
⟹ measure lebesgue (path-inside p) = I + B/2 − 1
measure lebesgue (path-inside p) = I + B/2 − 1
⟹ measure lebesgue (path-inside p2) = I2 + B2/2 − 1
⟹ measure lebesgue (path-inside p1) = I1 + B1/2 − 1
measure lebesgue (path-inside p) = I + B/2 − 1
⟹ measure lebesgue (path-inside p1) = I1 + B1/2 − 1
⟹ measure lebesgue (path-inside p2) = I2 + B2/2 − 1
proof −
let *?p-im = {x. integral-vec x ∧ x ∈ path-image p}*
let *?p1-im = {x. integral-vec x ∧ x ∈ path-image p1}*
let *?p2-im = {x. integral-vec x ∧ x ∈ path-image p2}*
let *?p-int = {x. integral-vec x ∧ x ∈ path-inside p}*
let *?p1-int = {x. integral-vec x ∧ x ∈ path-inside p1}*
let *?p2-int = {x. integral-vec x ∧ x ∈ path-inside p2}*

have *vts*: *vts = vts1 @ (x # (vts2 @ y # vts3))*
using *assms split-up-a-list-into-3-parts*
using *is-polygon-split-path-def* **by** *blast*
have *polygon p*
using *finite-path-image assms(1) p* **unfolding** *is-polygon-split-path-def*
by (*smt* (*verit*, *best*))
then have *B-finite*: *finite ?p-im*
using *finite-path-image* **by** *auto*
have *polygon-p1*: *polygon p1*
using *finite-path-image assms(1) p1* **unfolding** *is-polygon-split-path-def*
by (*smt* (*verit*) *assms(3) assms(5) assms(6)*)
then have *B1-finite*: *finite ?p1-im*
using *finite-path-image* **by** *auto*
have *polygon-p2*: *polygon p2*
using *finite-path-image assms(1) p1* **unfolding** *is-polygon-split-path-def*
by (*smt* (*verit*) *assms(2) assms(4) assms(5) assms(6) p2*)
then have *B2-finite*: *finite ?p2-im*
using *finite-path-image* **by** *auto*

have *vts-distinct*: *distinct vts*
using *simple-polygonal-path-vts-distinct*
by (*metis ‹polygon p› butlast-snoc p polygon-def*)

then have *x-neq-y*: $x \neq y$
by (*metis assms*(*1*) *assms*(*5*) *assms*(*6*) *index-first index-nth-id is-polygon-split-path-def*)
then have *card-2*: *card* $\{x, y\} = 2$
by *auto*
have *polygon-split-props*: (*is-polygon-cut-path* (*vts*@[*vts*!*0*]) *cutpath* ∧
polygon p ∧ *polygon p1* ∧ *polygon p2* ∧
path-inside p1 ∩ *path-inside p2* = {} ∧
path-inside p1 ∪ *path-inside p2* ∪ (*path-image cutpath* − $\{x, y\}$) = *path-inside p*
∧ ((*path-image p1*) − (*path-image cutpath*)) ∩ ((*path-image p2*) − (*path-image cutpath*)) = {}
∧ *path-image p* = ((*path-image p1*) − (*path-image cutpath*)) ∪ ((*path-image p2*) − (*path-image cutpath*)) ∪ $\{x, y\}$)
using *assms*
by (*meson is-polygon-split-path-def*)
have *measure-sum*: *measure lebesgue* (*path-inside p*) = *measure lebesgue* (*path-inside p1*) + *measure lebesgue* (*path-inside p2*)
using *polygon-split-path-add-measure assms*
by (*smt* (*verit*, *del-insts*))

let *?yx-int* = $\{k.$ *integral-vec k* ∧ *k* ∈ *path-image* (*make-polygonal-path* (*y*#*rev cutvts*@[*x*]))$\}$
let *?xy-int* = $\{k.$ *integral-vec k* ∧ *k* ∈ *path-image cutpath*$\}$
have *yx-int-is-xy-int*: *?yx-int* = *?xy-int*
using *rev-vts-path-image*[*of x # cutvts* @ [*y*]] *assms*(*7*) **by** *simp*
have *x # vts2* @ [*y*] @ *rev cutvts* @ [*x*] = (*x*#*vts2*) @ ([*y*] @ *rev cutvts* @ [*x*]) @ []
by *simp*
then have *sublist* ([*y*]@*rev cutvts*@[*x*]) *?p1-vts*
unfolding *sublist-def* **by** *blast*
then have *subset1*:
?xy-int ⊆ *?p1-im*
using *sublist-integral-subset-integral-on-path p1 yx-int-is-xy-int*
by *force*
have *len-gteq*: *length* (*x # cutvts* @ [*y*]) ≥ *2*
by *auto*
have *sublist-p2*: *sublist* (*x # cutvts* @ [*y*]) *?p2-vts*
unfolding *sublist-def* **by** *auto*
then have *subset2*:
?xy-int ⊆ *?p2-im*
using *sublist-integral-subset-integral-on-path*[*OF len-gteq p2 sublist-p2*]
assms(*7*) **by** *blast*

let *?S1* = *?p1-im* − *?xy-int*
let *?S2* = *?p2-im* − *?xy-int*
have *disjoint-1*: *?S1* ∩ *?S2* = {}
using *polygon-split-props* **by** *blast*

have *integral-xy*: *integral-vec x* ∧ *integral-vec y*
using *all-integral-vts vts*
using *all-integral-def* **by** *auto*
have *nonempty*: *y # rev cutvts* @ [*x*] ≠ []
by *simp*
have *trivial*: *make-polygonal-path* (*y # rev cutvts* @ [*x*]) = *make-polygonal-path* (*y # rev cutvts* @ [*x*])
by *auto*
have *pathstart* (*make-polygonal-path* (*y*#*rev cutvts*@[*x*])) = *y* ∧ *pathfinish* (*make-polygonal-path* (*y*#*rev cutvts*@[*x*])) = *x*
using *polygon-pathstart*[*OF nonempty trivial*] *polygon-pathfinish*[*OF nonempty trivial*]
by (*metis last.simps last-conv-nth nonempty nth-Cons-0 snoc-eq-iff-butlast*)
then have *x-in-y-in*: *x* ∈ *path-image* (*make-polygonal-path* (*y*#*rev cutvts*@[*x*])) ∧ *y* ∈ *path-image* (*make-polygonal-path* (*y*#*rev cutvts*@[*x*]))
unfolding *pathstart-def pathfinish-def path-image-def*
by (*metis* ‹*pathstart* (*make-polygonal-path* (*y # rev cutvts* @ [*x*])) = *y* ∧ *pathfinish* (*make-polygonal-path* (*y # rev cutvts* @ [*x*])) = *x*› *path-image-def pathfinish-in-path-image pathstart-in-path-image*)
then have {*x*, *y*} ⊆ *?yx-int*
using *integral-xy*
by *simp*
then have *disjoint-2*: (*?S1* ∪ *?S2*) ∩ {*x*, *y*} = {}
by (*simp add*: *yx-int-is-xy-int*)
have *path-image p* =
path-image p1 − *path-image cutpath* ∪
(*path-image p2* − *path-image cutpath*) ∪
{*x*, *y*}
using *polygon-split-props* **by** *auto*
then have *set-union*: *?p-im* = (*?S1* ∪ *?S2*) ∪ {*x*, *y*}
using *polygon-split-props integral-xy* **by** *auto*
then have *add-card*: *B* = *card* (*?p1-im* − *?xy-int*) + *card* (*?p2-im* − *?xy-int*) + *card* {*x*, *y*}
using *B-finite* **using** *disjoint-1 disjoint-2*
by (*metis* (*no-types*, *lifting*) *B card-Un-disjoint finite-Un*)
have *sub1*: *card* (*?p1-im* − *?xy-int*) = *B1* − *card ?xy-int*
using *B1-finite B1 subset1*
by (*meson card-Diff-subset finite-subset*)
have *sub2*: *card* (*?p2-im* − *?xy-int*) = *B2* − *card ?xy-int*
using *B2-finite B2 subset2*
by (*meson card-Diff-subset finite-subset*)
have *B*: *B* = (*B1* − *card ?xy-int*) + (*B2* − *card ?xy-int*) + *card* {*x*, *y*}
using *add-card sub1 sub2*
by *auto*
then have *B-sum-h*: *B* = *B1* + *B2* − *2*∗*card ?xy-int* + *2*
using *card-2*
by (*smt* (*verit*, *best*) *B1 B1-finite B2 B2-finite Nat.add-diff-assoc add.commute card-mono diff-diff-left mult-2 subset1 subset2*)
then have *B1* + *B2* = *B* + *2*∗*card ?xy-int* − *2*

by (*metis* (*no-types*, *lifting*) *B1 B1-finite B2 B2-finite add-mono-thms-linordered-semiring*(*1*) *card-mono diff-add-inverse2 le-add2 mult-2 ordered-cancel-comm-monoid-diff-class.add-diff-assoc2 subset1 subset2*)
then have *B-sum*: (*B1* + *B2*)/*2* = *B*/*2* + *card ?xy-int* − *1*
by (*smt* (*verit*) *B-sum-h field-sum-of-halves le-add2 mult-2 nat-1-add-1 of-nat-1 of-nat-add of-nat-diff ordered-cancel-comm-monoid-diff-class.add-diff-assoc2*)
have *casting-h*: ⋀ *A B*:: *nat*. *A* ≥ *B* ⟹ *real* (*A* − *B*) = *real A* − *real B*
by *auto*
have *path-inside p1* ∪ *path-inside p2* ∪ (*path-image cutpath* − {*x*, *y*}) = *path-inside p*
using *polygon-split-props* **by** *auto*
then have *interior-union*: *?p-int* = (*?xy-int* − {*x*, *y*}) ∪ *?p1-int* ∪ *?p2-int*
by *blast*

have *finite-inside-p*: *finite ?p-int*
using *bounded-finite inside-outside-polygon*
by (*simp add*: *polygon-split-props inside-outside-def*)
have *finite-pathimage*: *finite* (*?xy-int* − {*x*, *y*})
using *B1-finite finite-subset subset1* **by** *auto*
have *finite-inside-p1*: *finite ?p1-int*
using *polygon-split-props bounded-finite inside-outside-polygon*
using *finite-Un finite-inside-p interior-union* **by** *auto*
have *finite-inside-p2*: *finite ?p2-int*
using *polygon-split-props bounded-finite inside-outside-polygon*
using *finite-Un finite-inside-p interior-union* **by** *auto*
have *path-image-inside-disjoint1*: (*?xy-int* − {*x*, *y*}) ∩ (*?p1-int*) = {}
using *subset1 inside-outside-polygon*[*OF polygon-p1*]
unfolding *inside-outside-def* **by** *auto*
have *path-image-inside-disjoint2*: (*?xy-int* − {*x*, *y*}) ∩ (*?p2-int*) = {}
using *subset2 inside-outside-polygon*[*OF polygon-p2*]
unfolding *inside-outside-def* **by** *auto*

have (*?xy-int* − {*x*, *y*}) ∩ (*?p1-int* ∪ *?p2-int*) = {}
using *subset2 path-image-inside-disjoint1 path-image-inside-disjoint2*
by *auto*
then have *I-is*: *I* = *card* (*?xy-int* − {*x*, *y*}) +
card (*?p1-int* ∪ *?p2-int*)
using *interior-union I finite-inside-p1 finite-inside-p2*
by (*metis* (*no-types*, *lifting*) *card-Un-disjoint finite-Un finite-pathimage sup-assoc*)

have *disjoint-4*: *?p1-int* ∩ *?p2-int* = {}
using *polygon-split-props* **by** *auto*
then have *I* = *card* (*?xy-int* − {*x*, *y*}) +
I1 + *I2*
using *I-is finite-inside-p1 finite-inside-p2*
by (*simp add*: *I1 I2 card-Un-disjoint*)
have *interior-subset*: (*?xy-int* − {*x*, *y*}) ⊆ *?p-int*
using *interior-union* **by** *auto*
have *x-y-subset*: {*x*, *y*} ⊆ *?xy-int*

using *x-in-y-in rev-vts-path-image*[*of x # cutvts @ [y]*] *assms*(7)
integral-xy
using *yx-int-is-xy-int* **by** *blast*
have *real* (*card* (*?xy-int − {x, y}*)) =
real (*card* (*?xy-int*)) − *real* (*card* {*x, y*})
using *x-y-subset*
by (*metis* (*no-types, lifting*) *B2-finite card-Diff-subset card-mono finite-subset of-nat-diff subset2*)
then have *card-diff*: *real* (*card* (*?xy-int − {x, y}*)) =
real (*card* (*?xy-int*)) − *2*
using *card-2* **by** *auto*
then have *I = I1 + I2 +* (*card* (*?xy-int − {x, y}*))
using *I I1 I2 interior-union finite-inside-p1 finite-inside-p2*
by (*simp add*: *I-is disjoint-4 card-Un-disjoint*)
then have *I = I1 + I2 + real* (*card* (*?xy-int*)) − *2*
using *card-diff*
by *linarith*
then have *I-sum*: *I1 + I2 = I − real* (*card ?xy-int*) + *2*
by *fastforce*

{assume *pick1*: *measure lebesgue* (*path-inside p1*) = *I1 + B1/2 − 1*
assume *pick2*: *measure lebesgue* (*path-inside p2*) = *I2 + B2/2 − 1*
have *measure lebesgue* (*path-inside p*) = *I1 + I2 +* (*B1+B2*)*/2 −2*
using *pick1 pick2 measure-sum* **by** *auto*
then have *measure lebesgue* (*path-inside p*) = *I − real* (*card ?xy-int*) + *2 + B/2 + card ?xy-int − 1 − 2*
using *I-sum B-sum*
by *linarith*
then have *measure lebesgue* (*path-inside p*) = *I + B/2 − 1* **by** *auto*
}
then show *measure lebesgue* (*path-inside p1*) = *I1 + B1/2 − 1 ⟹ measure lebesgue* (*path-inside p2*) = *I2 + B2/2 − 1 ⟹ measure lebesgue* (*path-inside p*) = *I + B/2 − 1*
by *blast*

{assume *pick1*: *measure lebesgue* (*path-inside p*) = *I + B/2 − 1*
assume *pick2*: *measure lebesgue* (*path-inside p2*) = *I2 + B2/2 − 1*
then have *real I + real B / 2 − 1 =* (*measure lebesgue* (*path-inside p1*)) + *I2 + B2/2 −1*
using *measure-sum pick1 pick2* **by** *auto*
then have *measure lebesgue* (*path-inside p*) = *I − real* (*card ?xy-int*) + *2 + B/2 + card ?xy-int − 1 − 2*
using *I-sum B-sum pick1*
by *linarith*
then have *measure lebesgue* (*path-inside p1*) = *I1 + B1/2 − 1*
using *B-sum ‹real I = real* (*I1 + I2*) + *real* (*card* {*k. integral-vec k ∧ k ∈ path-image cutpath*}) − *2› field-sum-of-halves measure-sum of-nat-add*
pick1 pick2 **by** *auto*
}

then show *measure lebesgue (path-inside p) = I + B/2 − 1 ⟹ measure lebesgue (path-inside p2) = I2 + B2/2 − 1 ⟹ measure lebesgue (path-inside p1) = I1 + B1/2 − 1*
by *blast*

{assume *pick1*: *measure lebesgue (path-inside p) = I + B/2 − 1*
assume *pick2*: *measure lebesgue (path-inside p1) = I1 + B1/2 − 1*
then have *real I + real B / 2 − 1 = (measure lebesgue (path-inside p2)) + I1 + B1/2 −1*
using *measure-sum pick1 pick2* **by** *auto*
then have *measure lebesgue (path-inside p) = I − real (card ?xy-int) + 2 + B/2 + card ?xy-int − 1 − 2*
using *I-sum B-sum pick1*
by *linarith*
then have *measure lebesgue (path-inside p2) = I2 + B2/2 − 1*
using *B-sum ‹real I = real (I1 + I2) + real (card {k. integral-vec k ∧ k ∈ path-image cutpath}) − 2› field-sum-of-halves measure-sum of-nat-add*
using *pick2* **by** *auto*
}
then show *measure lebesgue (path-inside p) = I + B/2 − 1 ⟹ measure lebesgue (path-inside p1) = I1 + B1/2 − 1 ⟹ measure lebesgue (path-inside p2) = I2 + B2/2 − 1*
by *blast*
qed

lemma *pick-split-union*:
assumes *is-split*: *is-polygon-split vts i j*
assumes *vts1* = (*take i vts*)
assumes *vts2* = (*take* (*j − i − 1*) (*drop* (*Suc i*) *vts*))
assumes *vts3* = *drop* (*j − i*) (*drop* (*Suc i*) *vts*)
assumes *x* = *vts* ! *i*
assumes *y* = *vts* ! *j*
assumes *p*: *p* = *make-polygonal-path* (*vts*@[*vts*!*0*]) (**is** *p* = *make-polygonal-path ?p-vts*)
assumes *p1*: *p1* = *make-polygonal-path* (*x*#(*vts2*@[*y*, *x*])) (**is** *p1* = *make-polygonal-path ?p1-vts*)
assumes *p2*: *p2* = *make-polygonal-path* (*vts1* @ [*x*, *y*] @ *vts3* @ [*vts* ! *0*]) (**is** *p2* = *make-polygonal-path ?p2-vts*)
assumes *I1*: *I1* = *card* {*x*. *integral-vec x* ∧ *x* ∈ *path-inside p1*}
assumes *B1*: *B1* = *card* {*x*. *integral-vec x* ∧ *x* ∈ *path-image p1*}
assumes *pick1*: *measure lebesgue* (*path-inside p1*) = *I1* + *B1/2* − *1*
assumes *I2*: *I2* = *card* {*x*. *integral-vec x* ∧ *x* ∈ *path-inside p2*}
assumes *B2*: *B2* = *card* {*x*. *integral-vec x* ∧ *x* ∈ *path-image p2*}
assumes *pick2*: *measure lebesgue* (*path-inside p2*) = *I2* + *B2/2* − *1*
assumes *I*: *I* = *card* {*x*. *integral-vec x* ∧ *x* ∈ *path-inside p*}
assumes *B*: *B* = *card* {*x*. *integral-vec x* ∧ *x* ∈ *path-image p*}
assumes *all-integral-vts*: *all-integral vts*
shows *measure lebesgue* (*path-inside p*) = *I* + *B/2* − *1*
measure lebesgue (*path-inside p*) = *measure lebesgue* (*path-inside p1*) +

measure lebesgue (*path-inside p2*)
proof −
let *?p-im* = {*x*. *integral-vec x* ∧ *x* ∈ *path-image p*}
let *?p1-im* = {*x*. *integral-vec x* ∧ *x* ∈ *path-image p1*}
let *?p2-im* = {*x*. *integral-vec x* ∧ *x* ∈ *path-image p2*}
let *?p-int* = {*x*. *integral-vec x* ∧ *x* ∈ *path-inside p*}
let *?p1-int* = {*x*. *integral-vec x* ∧ *x* ∈ *path-inside p1*}
let *?p2-int* = {*x*. *integral-vec x* ∧ *x* ∈ *path-inside p2*}

have *vts*: *vts* = *vts1* @ (*x* # (*vts2* @ *y* # *vts3*))
using *assms split-up-a-list-into-3-parts*
using *is-polygon-split-def* **by** *blast*
have *polygon p*
using *finite-path-image assms*(*1*) *p* **unfolding** *is-polygon-split-def*
by (*smt* (*verit*, *best*))
then have *B-finite*: *finite ?p-im*
using *finite-path-image* **by** *auto*
have *polygon-p1*: *polygon p1*
using *finite-path-image assms*(*1*) *p1* **unfolding** *is-polygon-split-def*
by (*smt* (*verit*) *assms*(*3*) *assms*(*5*) *assms*(*6*))
then have *B1-finite*: *finite ?p1-im*
using *finite-path-image* **by** *auto*
have *polygon-p2*: *polygon p2*
using *finite-path-image assms*(*1*) *p1* **unfolding** *is-polygon-split-def*
by (*smt* (*verit*) *assms*(*2*) *assms*(*4*) *assms*(*5*) *assms*(*6*) *p2*)
then have *B2-finite*: *finite ?p2-im*
using *finite-path-image* **by** *auto*

have *vts-distinct*: *distinct vts*
using *simple-polygonal-path-vts-distinct*
by (*metis* ‹*polygon p*› *butlast-snoc p polygon-def*)
then have *x-neq-y*: *x* ≠ *y*
by (*metis assms*(*1*) *assms*(*5*) *assms*(*6*) *index-first index-nth-id is-polygon-split-def*)
then have *card-2*: *card* {*x*, *y*} = *2*
by *auto*
have *polygon-split-props*: *is-polygon-cut ?p-vts x y* ∧
polygon p ∧ *polygon p1* ∧ *polygon p2* ∧
path-inside p1 ∩ *path-inside p2* = {} ∧
path-inside p1 ∪ *path-inside p2* ∪ (*path-image* (*linepath x y*) − {*x*, *y*})
= *path-inside p* ∧ ((*path-image p1*) − (*path-image* (*linepath x y*))) ∩
((*path-image p2*) − (*path-image* (*linepath x y*))) = {}
∧ *path-image p* = ((*path-image p1*) − (*path-image* (*linepath x y*))) ∪ ((*path-image p2*) − (*path-image* (*linepath x y*))) ∪ {*x*, *y*}
using *assms*
by (*meson is-polygon-split-def*)
have *measure lebesgue* (*path-inside p*) = *measure lebesgue* (*path-inside p1*) + *measure lebesgue* (*path-inside p2*)
using *polygon-split-add-measure assms*
by (*smt* (*verit*, *del-insts*))

then have *measure-sum*: *measure lebesgue* (*path-inside p*) = *I1* + *I2* + (*B1*+*B2*)/*2* −*2*
using *pick1 pick2* **by** *auto*

let *?yx-int* = {*k*. *integral-vec k* ∧ *k* ∈ *path-image* (*linepath y x*)}
let *?xy-int* = {*k*. *integral-vec k* ∧ *k* ∈ *path-image* (*linepath x y*)}
have *yx-int-is-xy-int*: *?yx-int* = *?xy-int*
by (*simp add*: *closed-segment-commute*)

have *sublist* [*y*, *x*] *?p1-vts* **by** (*simp add*: *sublist-Cons-right*)
then have *subset1*:
?xy-int ⊆ *?p1-im*
using *sublist-pair-integral-subset-integral-on-path p1 yx-int-is-xy-int* **by** *blast*
have *subset2*:
?xy-int ⊆ *?p2-im*
using *sublist-pair-integral-subset-integral-on-path p2* **by** *blast*

let *?S1* = *?p1-im* − *?xy-int*
let *?S2* = *?p2-im* − *?xy-int*
have *disjoint-1*: *?S1* ∩ *?S2* = {}
using *polygon-split-props* **by** *blast*

have *integral-xy*: *integral-vec x* ∧ *integral-vec y*
using *all-integral-vts vts*
using *all-integral-def* **by** *auto*
then have {*x*, *y*} ⊆ *?yx-int*
by *simp*
then have *disjoint-2*: (*?S1* ∪ *?S2*) ∩ {*x*, *y*} = {}
by *simp*
have *path-image p* =
path-image p1 − *path-image* (*linepath x y*) ∪
(*path-image p2* − *path-image* (*linepath x y*)) ∪
{*x*, *y*}
using *polygon-split-props* **by** *auto*
then have *set-union*: *?p-im* = (*?S1* ∪ *?S2*) ∪ {*x*, *y*}
using *polygon-split-props integral-xy* **by** *auto*
then have *add-card*: *B* = *card* (*?p1-im* − *?xy-int*) + *card* (*?p2-im* − *?xy-int*) + *card* {*x*, *y*}
using *B-finite* **using** *disjoint-1 disjoint-2*
by (*metis* (*no-types*, *lifting*) *B card-Un-disjoint finite-Un*)
have *sub1*: *card* (*?p1-im* − *?xy-int*) = *B1* − *card ?xy-int*
using *B1-finite B1 subset1*
by (*meson card-Diff-subset finite-subset*)
have *sub2*: *card* (*?p2-im* − *?xy-int*) = *B2* − *card ?xy-int*
using *B2-finite B2 subset2*
by (*meson card-Diff-subset finite-subset*)
have *B*: *B* = (*B1* − *card ?xy-int*) + (*B2* − *card ?xy-int*) + *card* {*x*, *y*}
using *add-card sub1 sub2*
by *auto*

then have *B-sum-h*: $B = B1 + B2 - 2*card\ ?xy\text{-}int + 2$
using *card-2*
by (*smt* (*verit*, *best*) *B1 B1-finite B2 B2-finite Nat.add-diff-assoc add.commute card-mono diff-diff-left mult-2 subset1 subset2*)
then have $B1 + B2 = B + 2*card\ ?xy\text{-}int - 2$
by (*metis* (*no-types*, *lifting*) *B1 B1-finite B2 B2-finite add-mono-thms-linordered-semiring*(*1*) *card-mono diff-add-inverse2 le-add2 mult-2 ordered-cancel-comm-monoid-diff-class.add-diff-assoc2 subset1 subset2*)
then have *B-sum*: $(B1 + B2)/2 = B/2 + card\ ?xy\text{-}int - 1$
by (*smt* (*verit*) *B-sum-h field-sum-of-halves le-add2 mult-2 nat-1-add-1 of-nat-1 of-nat-add of-nat-diff ordered-cancel-comm-monoid-diff-class.add-diff-assoc2*)
have *casting-h*: $\bigwedge A\ B{::}\ nat.\ A \geq B \Longrightarrow real\ (A - B) = real\ A - real\ B$
by *auto*
have $path\text{-}inside\ p1 \cup path\text{-}inside\ p2 \cup (path\text{-}image\ (linepath\ x\ y) - \{x, y\}) = path\text{-}inside\ p$
using *polygon-split-props* **by** *auto*
then have *interior-union*: $?p\text{-}int = (?xy\text{-}int - \{x, y\}) \cup ?p1\text{-}int \cup ?p2\text{-}int$
by *blast*

have *finite-inside-p*: *finite ?p-int*
using *bounded-finite inside-outside-polygon*
by (*simp add*: *polygon-split-props inside-outside-def*)
have *finite-pathimage*: $finite\ (?xy\text{-}int - \{x, y\})$
using *B1-finite finite-subset subset1* **by** *auto*
have *finite-inside-p1*: *finite ?p1-int*
using *polygon-split-props bounded-finite inside-outside-polygon*
using *finite-Un finite-inside-p interior-union* **by** *auto*
have *finite-inside-p2*: *finite ?p2-int*
using *polygon-split-props bounded-finite inside-outside-polygon*
using *finite-Un finite-inside-p interior-union* **by** *auto*
have *path-image-inside-disjoint1*: $(?xy\text{-}int - \{x, y\}) \cap (?p1\text{-}int) = \{\}$
using *subset1 inside-outside-polygon*[*OF polygon-p1*]
unfolding *inside-outside-def* **by** *auto*
have *path-image-inside-disjoint2*: $(?xy\text{-}int - \{x, y\}) \cap (?p2\text{-}int) = \{\}$
using *subset2 inside-outside-polygon*[*OF polygon-p2*]
unfolding *inside-outside-def* **by** *auto*
have $(?xy\text{-}int - \{x, y\}) \cap (?p1\text{-}int \cup ?p2\text{-}int) = \{\}$
using *subset2 path-image-inside-disjoint1 path-image-inside-disjoint2*
by *auto*
then have *I-is*: $I = card\ (?xy\text{-}int - \{x, y\}) +$
$card\ (?p1\text{-}int \cup ?p2\text{-}int)$
using *interior-union I finite-inside-p1 finite-inside-p2*
by (*metis* (*no-types*, *lifting*) *card-Un-disjoint finite-Un finite-pathimage sup-assoc*)

have *disjoint-4*: $?p1\text{-}int \cap ?p2\text{-}int = \{\}$
using *polygon-split-props* **by** *auto*
then have $I = card\ (?xy\text{-}int - \{x, y\}) +$
$I1 + I2$
using *I-is finite-inside-p1 finite-inside-p2*

by (*simp add*: *I1 I2 card-Un-disjoint*)
have *interior-subset*: (*?xy-int* − {*x*, *y*}) ⊆ *?p-int*
using *interior-union* **by** *auto*
have *x-y-subset*: {*x*, *y*} ⊆ *?xy-int*
using *local.set-union* **by** *auto*
have *real* (*card* (*?xy-int* − {*x*, *y*})) =
real (*card* (*?xy-int*)) − *real* (*card* {*x*, *y*})
using *x-y-subset*
by (*metis* (*no-types*, *lifting*) *B2-finite card-Diff-subset card-mono finite-subset of-nat-diff subset2*)
then have *card-diff*: *real* (*card* (*?xy-int* − {*x*, *y*})) =
real (*card* (*?xy-int*)) − *2*
using *card-2* **by** *auto*
then have *I* = *I1* + *I2* + (*card* (*?xy-int* − {*x*, *y*}))
using *I I1 I2 interior-union finite-inside-p1 finite-inside-p2*
by (*simp add*: *I-is disjoint-4 card-Un-disjoint*)
then have *I* = *I1* + *I2* + *real* (*card* (*?xy-int*)) − *2*
using *card-diff*
by *linarith*
then have *I-sum*: *I1* + *I2* = *I* − *real* (*card ?xy-int*) + *2*
by *fastforce*
have *measure lebesgue* (*path-inside p*) = *I* − *real* (*card ?xy-int*) + *2* +
B/*2* + *card ?xy-int* − *1* − *2*
using *measure-sum I-sum B-sum*
by *linarith*
then show *measure lebesgue* (*path-inside p*) = *I* + *B*/*2* − *1* **by** *auto*

show *measure lebesgue* (*path-inside p*) = *measure lebesgue* (*path-inside p1*) + *measure lebesgue* (*path-inside p2*)
using ‹*Sigma-Algebra.measure lebesgue* (*path-inside p*) = *Sigma-Algebra.measure lebesgue* (*path-inside p1*) + *Sigma-Algebra.measure lebesgue* (*path-inside p2*)› **by** *blast*
qed

lemma *pick-split-path-union*:
assumes *is-split*: *is-polygon-split-path vts i j cutvts*
assumes *vts1* = (*take i vts*)
assumes *vts2* = (*take* (*j* − *i* − *1*) (*drop* (*Suc i*) *vts*))
assumes *vts3* = *drop* (*j* − *i*) (*drop* (*Suc i*) *vts*)
assumes *x* = *vts*!*i*
assumes *y* = *vts*!*j*
assumes *cutpath* = *make-polygonal-path* (*x* # *cutvts* @ [*y*])
assumes *p*: *p* = *make-polygonal-path* (*vts*@[*vts*!*0*]) (**is** *p* = *make-polygonal-path ?p-vts*)
assumes *p1*: *p1* = *make-polygonal-path* (*x*#(*vts2* @ [*y*] @ (*rev cutvts*) @ [*x*])) (**is** *p1* = *make-polygonal-path ?p1-vts*)
assumes *p2*: *p2* = *make-polygonal-path* (*vts1* @ ([*x*] @ *cutvts* @ [*y*]) @ *vts3* @ [*vts* ! *0*]) (**is** *p2* = *make-polygonal-path ?p2-vts*)
assumes *I1*: *I1* = *card* {*x*. *integral-vec x* ∧ *x* ∈ *path-inside p1*}

assumes *B1*: *B1* = *card* {*x*. *integral-vec x* ∧ *x* ∈ *path-image p1*}
assumes *pick1*: *measure lebesgue* (*path-inside p1*) = *I1* + *B1*/2 − 1
assumes *I2*: *I2* = *card* {*x*. *integral-vec x* ∧ *x* ∈ *path-inside p2*}
assumes *B2*: *B2* = *card* {*x*. *integral-vec x* ∧ *x* ∈ *path-image p2*}
assumes *pick2*: *measure lebesgue* (*path-inside p2*) = *I2* + *B2*/2 − 1
assumes *I*: *I* = *card* {*x*. *integral-vec x* ∧ *x* ∈ *path-inside p*}
assumes *B*: *B* = *card* {*x*. *integral-vec x* ∧ *x* ∈ *path-image p*}
assumes *all-integral-vts*: *all-integral vts*
shows *measure lebesgue* (*path-inside p*) = *I* + *B*/2 − 1
using *pick-split-path-union-main pick1 pick2*(1) *assms* **by** *blast*

lemma *pick-triangle-basic-split*:
assumes *p* = *make-triangle a b c* **and** *distinct* [*a*, *b*, *c*] **and** ¬ *collinear* {*a*, *b*, *c*} **and**
d-prop: *d* ∈ *path-image* (*linepath a b*) ∧ *d* ∉ {*a*, *b*, *c*}
shows *good-linepath c d* [*a*, *d*, *b*, *c*, *a*]
∧ *path-image* (*make-polygonal-path* [*a*, *d*, *b*, *c*, *a*]) = *path-image p*
proof−
let *?l* = *linepath c d*
let *?L* = *path-image ?l*
let *?P* = *path-image p*
let *?vts′* = [*a*, *d*, *b*, *c*, *a*]
let *?p′* = *make-polygonal-path ?vts′*
let *?P′* = *path-image ?p′*

have *h1*: *path-image* (*make-polygonal-path* [*a*, *b*, *c*, *a*]) = *path-image* (*linepath a b*) ∪ *path-image* (*linepath b c*) ∪ *path-image* (*linepath c a*)
using *polygonal-path-image-linepath-union* **by** (*simp add*: *path-image-join sup.assoc*)
have *h2*: *path-image* (*make-polygonal-path* [*a*, *d*, *b*, *c*, *a*]) = *path-image* (*linepath a d*) ∪ *path-image* (*linepath d b*) ∪ *path-image* (*linepath b c*) ∪ *path-image* (*linepath c a*)
using *polygonal-path-image-linepath-union* **by** (*simp add*: *path-image-join sup.assoc*)
have *h3*: *path-image* (*linepath a b*) = *path-image* (*linepath a d*) ∪ *path-image* (*linepath d b*)
using *path-image-linepath-union d-prop* **by** *auto*

have *1*: *?P′* = *?P*
using *h1 h2 h3*
using *assms*(1) *make-triangle-def* **by** *force*

have {*c*, *d*} = *?L* ∩ *?P*
proof(*rule ccontr*)
have *subs*: {*c*, *d*} ⊆ *?L* ∩ *?P*
using *assms*(1) *vertices-on-path-image* **unfolding** *make-triangle-def*
by (*metis IntD2 IntI assms*(4) *empty-subsetI inf-sup-absorb insert-subset list.discI list.simps*(15) *nth-Cons-0 path-image-cons-union pathfinish-in-path-image pathfinish-linepath pathstart-in-path-image pathstart-linepath*)

assume ∗: {*c*, *d*} ≠ *?L* ∩ *?P*

then obtain z **where** z: $z \neq c \wedge z \neq d \wedge z \in ?L \cap ?P$ **using** *subs* **by** *blast*
then have *cases*:
$z \in$ *path-image* (*linepath a b*) $\vee$ $z \in$ *path-image* (*linepath b c*) $\vee$ $z \in$ *path-image* (*linepath c a*)
using *1 h2 h3* **by** *blast*
{ assume $**$: $z \in$ *path-image* (*linepath a b*)
moreover have $z \in ?L \wedge d \in ?L \wedge d \in$ *path-image* (*linepath a b*) **using** *assms z* **by** *force*
ultimately have $\{z, d\} \subseteq ?L \cap$ *path-image* (*linepath a b*) $\wedge z \neq d$ **using** *z* **by** *blast*
then have *collinear* $\{a, b, c, d\}$ **using** *two-linepath-colinearity-property* **by** *fastforce*
then have *False* **using** *assms(2) assms(3) collinear-4-3* **by** *auto*
} moreover
{ assume $**$: $z \in$ *path-image* (*linepath b c*)
then have *collinear* $\{a, b, c, d\}$ **using** *two-linepath-colinearity-property*[*of z - b c c d*]
by (*smt* (*verit*) $**$ *IntE assms(3) collinear-3-trans d-prop in-path-image-imp-collinear insertCI insert-commute z*)
then have *False* **using** *assms(2) assms(3) collinear-4-3* **by** *auto*
} moreover
{ assume $**$: $z \in$ *path-image* (*linepath c a*)
then have *collinear* $\{a, b, c, d\}$ **using** *two-linepath-colinearity-property*[*of z - c a c d*]
by (*smt* (*verit*) *IntD1 assms(3) collinear-3-trans d-prop in-path-image-imp-collinear insert-commute insert-iff z*)
then have *False* **using** *assms(2) assms(3) collinear-4-3* **by** *auto*
}
ultimately show *False* **using** *cases* **by** *argo*
qed
moreover have $?L \subseteq$ *path-inside p* $\cup$ $?P$
proof−
have *convex hull* $\{a, b, c\}$ = *path-inside p* $\cup$ $?P$
by (*simp add*: *Un-commute assms(1) assms(3) triangle-convex-hull*)
moreover have $?L \subseteq$ *convex hull* $\{a, b, c\}$
by (*smt* (*verit*, *ccfv-threshold*) *assms empty-subsetI hull-insert hull-mono insert-commute insert-mono insert-subset path-image-linepath segment-convex-hull*)
ultimately show *?thesis* **by** *blast*
qed
ultimately have $?L \subseteq$ *path-inside p* $\cup \{c, d\}$ **by** *blast*
then have $?L \subseteq$ *path-inside* $?p' \cup \{c, d\}$ **using** *1* **unfolding** *path-inside-def* **by** *presburger*
then have *2*: *good-linepath c d* $?vts'$ **using** *assms* **unfolding** *good-linepath-def* **by** *auto*

thus *?thesis* **using** *1* **by** *blast*
qed

28 Convex Hull Has Good Linepath

lemma *leq-2-extreme-points-means-collinear*:
fixes *vts* :: *'a*::*euclidean-space set*
assumes *finite vts*
assumes *card* {*v*. *v extreme-point-of* (*convex hull vts*)} ≤ *2*
shows *collinear vts*
using *assms*
by (*metis Krein-Milman-polytope affine-hull-convex-hull collinear-affine-hull-collinear collinear-small extreme-points-of-convex-hull finite-subset*)

lemma *convex-hull-non-extreme-point-in-open-seg*:
assumes *H* = *convex hull vts*
assumes *x* ∈ *H* − {*v*. *v extreme-point-of H*}
shows ∃ *a b*. *a* ∈ *H* ∧ *b* ∈ *H* ∧ *x* ∈ *open-segment a b*
using *assms* **unfolding** *extreme-point-of-def* **by** *blast*

lemma *convex-hull-extreme-points-vertex-split*:
fixes *vts* :: (*real^2*) *set*
assumes *H* = *convex hull vts*
assumes *finite vts*
assumes *card* {*v*. *v extreme-point-of H*} ≥ *4*
assumes {*a*, *b*, *c*} ⊆ {*v*. *v extreme-point-of H*} ∧ *distinct* [*a*, *b*, *c*]
shows *path-image* (*linepath a b*) ∩ *interior H* ≠ {}
∨ *path-image* (*linepath b c*) ∩ *interior H* ≠ {}
∨ *path-image* (*linepath c a*) ∩ *interior H* ≠ {}
proof−
let *?ep* = {*v*. *v extreme-point-of H*}

have *H*: *H* = *convex hull ?ep* **using** *Krein-Milman-polytope assms*(*1*) *assms*(*2*)
by *blast*
let *?H'* = *convex hull* {*a*, *b*, *c*}

have *not-collinear*: ¬ *collinear* {*a*, *b*, *c*}
proof(*rule ccontr*)
assume ¬ ¬ *collinear* {*a*, *b*, *c*}
then have *collinear* {*a*, *b*, *c*} **by** *blast*
then have *a* ∈ *path-image* (*linepath b c*)
∨ *b* ∈ *path-image* (*linepath a c*)
∨ *c* ∈ *path-image* (*linepath a b*)
using *collinear-between-cases* **unfolding** *between-def*
by (*smt* (*verit*, *del-insts*) *between-mem-segment closed-segment-eq collinear-between-cases doubleton-eq-iff path-image-linepath*)
moreover have *a* ≠ *b* ∧ *b* ≠ *c* ∧ *a* ≠ *c* **using** *assms* **by** *simp*
ultimately have *a* ∈ *open-segment b c* ∨ *b* ∈ *open-segment a c* ∨ *c* ∈ *open-segment a b*
using *closed-segment-eq-open* **by** *auto*
moreover have *a extreme-point-of H* ∧ *b extreme-point-of H* ∧ *c extreme-point-of H*

using *assms* **by** *blast*
ultimately show *False* **unfolding** *extreme-point-of-def* **by** *blast*
qed

have *strict-subset*: *interior ?H′ ⊂ interior H*
proof−
have *interior ?H′ ⊆ interior H*
by (*metis H assms*(*4*) *hull-mono interior-mono*)
moreover have *?H′ ⊂ H*
proof−
have *card* {*a*, *b*, *c*} ≤ *3*
by (*metis card.empty card-insert-disjoint collinear-2 finite.emptyI finite-insert insert-absorb nat-le-linear not-collinear numeral-3-eq-3*)
then have *card* (*?ep* − {*a*, *b*, *c*}) ≥ *1*
using *assms*(*3*) *assms*(*4*) **by** *auto*
then obtain *d* **where** *d* ∈ *?ep* − {*a*, *b*, *c*}
by (*metis One-nat-def all-not-in-conv card.empty not-less-eq-eq zero-le*)
thus *?thesis*
by (*metis DiffE H assms*(*4*) *extreme-point-of-convex-hull hull-mono mem-Collect-eq order-less-le*)
qed
ultimately show *?thesis*
by (*metis* (*no-types*, *lifting*) *assms*(*1*) *assms*(*2*) *closure-convex-hull convex-closure-rel-interior convex-convex-hull convex-hull-eq-empty convex-polygon-frontier-is-path-image2 dual-order.strict-iff-order finite.emptyI finite.insertI finite-imp-bounded-convex-hull finite-imp-compact frontier-empty insert-not-empty inside-frontier-eq-interior not-collinear path-inside-def polygon-frontier-is-path-image rel-interior-nonempty-interior sup-bot.right-neutral triangle-convex-hull triangle-is-convex triangle-is-polygon*)
qed
moreover have *interior ?H′ ≠ {}*
by (*metis not-collinear convex-convex-hull convex-hull-eq-empty convex-polygon-frontier-is-path-image2 finite.emptyI finite.insertI finite-imp-bounded-convex-hull frontier-empty insert-not-empty inside-frontier-eq-interior path-inside-def polygon-frontier-is-path-image sup-bot.right-neutral triangle-convex-hull triangle-is-convex triangle-is-polygon*)
ultimately obtain *x y* **where** *xy*: *x ∈ interior ?H′ ∧ y ∈ interior H − interior ?H′* **by** *blast*

let *?l* = *linepath x y*

have *x ∈ interior ?H′ ∧ y ∈ −*(*interior ?H′*) **using** *xy* **by** *blast*
then have *path-image ?l ∩ interior ?H′ ≠ {} ∧ path-image ?l ∩ −*(*interior ?H′*) *≠ {}* **by** *auto*
moreover have *path-connected* (*interior ?H′*) **by** (*simp add*: *convex-imp-path-connected*)
ultimately obtain *z* **where** *z*: *z ∈ path-image ?l ∩ frontier* (*interior ?H′*)
by (*metis Diff-eq Diff-eq-empty-iff all-not-in-conv convex-convex-hull convex-imp-path-connected path-connected-not-frontier-subset path-image-linepath segment-convex-hull*)
moreover have *path-image ?l ⊆ interior H* **using** *xy convex-interior*[*of H*]
by (*metis DiffD1 IntD2 strict-subset assms*(*1*) *closed-segment-subset convex-convex-hull inf.strict-order-iff path-image-linepath*)

ultimately have *z-interior*: $z \in$ *interior* H **by** *blast*

have $z \in$ *frontier* (*interior ?H'*) **using** z **by** *blast*
moreover have *frontier* (*interior ?H'*)
$=$ *path-image* (*linepath a b*) $\cup$ *path-image* (*linepath b c*) $\cup$ *path-image* (*linepath c a*)
proof−
let *?p* = *make-triangle a b c*
have *path-inside ?p* = *interior ?H'*
by (*metis not-collinear bounded-convex-hull bounded-empty bounded-insert convex-convex-hull convex-polygon-frontier-is-path-image2 inside-frontier-eq-interior path-inside-def triangle-convex-hull triangle-is-convex triangle-is-polygon*)
then have *path-image ?p* = *frontier* (*interior ?H'*)
by (*metis not-collinear polygon-frontier-is-path-image triangle-is-polygon*)
moreover have *path-image ?p*
$=$ *path-image* (*linepath a b*) $\cup$ *path-image* (*linepath b c*) $\cup$ *path-image* (*linepath c a*)
by (*metis Un-assoc list.discI make-polygonal-path.simps*(*3*) *make-triangle-def nth-Cons-0 path-image-cons-union*)
ultimately show *?thesis* **by** *presburger*
qed
ultimately show *?thesis* **using** *z-interior* **by** *blast*
qed

lemma *convex-hull-has-vertex-split-helper-wlog*:
assumes p = *make-triangle a b c* **and** *distinct* [a, b, c] **and** ¬ *collinear* {a, b, c} **and**
d-prop: $d \in$ *path-image* (*linepath a b*) $\wedge$ $d \notin$ {a, b, c}
shows *path-image* (*linepath c d*) $\cap$ *path-inside* $p \neq$ {}
proof−
have *good-linepath c d* [a, d, b, c, a]
$\wedge$ *path-image* (*make-polygonal-path* [a, d, b, c, a]) = *path-image* p
using *pick-triangle-basic-split*[*of p a b c d*] *assms* **by** *fast*
thus *?thesis*
unfolding *good-linepath-def*
by (*smt* (*verit*, *del-insts*) *Int-Un-eq*(*4*) *Int-insert-right-if1 Un-insert-right diff-points-path-image-set-property le-iff-inf path-inside-def pathfinish-in-path-image pathfinish-linepath pathstart-in-path-image pathstart-linepath*)
qed

lemma *convex-hull-has-vertex-split-helper*:
assumes p = *make-triangle a b c* **and** *distinct* [a, b, c] **and** ¬ *collinear* {a, b, c} **and**
d-prop: $d \in$ *path-image* $p \wedge d \notin$ {a, b, c}
shows $\exists x\ y.$ {x, y} $\subseteq$ {a, b, c, d} $\wedge$ $x \neq y \wedge$ *path-image* (*linepath x y*) $\cap$ *path-inside* $p \neq$ {}
proof−
{ assume $d \in$ *path-image* (*linepath a b*)
then have *?thesis*

using *convex-hull-has-vertex-split-helper-wlog*[*of p a b c d*] *assms*(*1*) *assms*(*2*) *assms*(*3*) *d-prop*
by *fastforce*
} **moreover**
{ **assume** ∗: $d \in$ *path-image* (*linepath b c*)
let *?p'* = *make-triangle b c a*
have *path-image* (*linepath a d*) ∩ *path-inside ?p'* ≠ {}
using *convex-hull-has-vertex-split-helper-wlog*[*of ?p' b c a d*]
by (*metis* (*no-types*, *opaque-lifting*) ∗ *assms*(*3*) *collinear-2 d-prop distinct-length-2-or-more distinct-singleton insert-absorb2 insert-commute*)
moreover have *path-inside ?p'* = *path-inside p*
unfolding *make-triangle-def*
by (*smt* (*verit*, *best*) *assms*(*1*) *assms*(*3*) *convex-polygon-frontier-is-path-image2 insert-commute make-triangle-def path-inside-def triangle-convex-hull triangle-is-convex triangle-is-polygon*)
ultimately have *?thesis* **using** *assms* **by** *auto*
} **moreover**
{ **assume** ∗: $d \in$ *path-image* (*linepath c a*)
let *?p'* = *make-triangle c a b*
have *path-image* (*linepath b d*) ∩ *path-inside ?p'* ≠ {}
using *convex-hull-has-vertex-split-helper-wlog*[*of ?p' c a b d*]
by (*metis* (*no-types*, *opaque-lifting*) ∗ *assms*(*3*) *collinear-2 d-prop distinct-length-2-or-more distinct-singleton insert-absorb2 insert-commute*)
moreover have *path-inside ?p'* = *path-inside p*
unfolding *make-triangle-def*
by (*smt* (*verit*, *ccfv-SIG*) *assms*(*1*) *assms*(*3*) *convex-polygon-frontier-is-path-image2 insert-commute make-triangle-def path-inside-def triangle-convex-hull triangle-is-convex triangle-is-polygon*)
ultimately have *?thesis* **using** *assms* **by** *auto*
}
ultimately show *?thesis* **using** *on-triangle-path-image-cases assms*(*1*) *d-prop*
by *fast*
qed

lemma *convex-hull-has-vertex-split*:
fixes *vts* :: (*real*^*2*) *set*
assumes *H* = *convex hull vts*
assumes ¬ *collinear vts*
assumes *card vts* > *3*
assumes *finite vts*
shows ∃ *a b*. {*a*, *b*} ⊆ *vts* ∧ *a* ≠ *b* ∧ *path-image* (*linepath a b*) ∩ *interior H* ≠ {}
proof−
let *?ep* = {*v*. *v extreme-point-of H*}
have *ep*: *?ep* ⊆ *vts* **by** (*simp add*: *assms*(*1*) *extreme-points-of-convex-hull*)
have *card-ep*: *card ?ep* ≥ *3*
by (*metis One-nat-def Suc-1 assms*(*1*) *assms*(*2*) *assms*(*3*) *card.infinite leq-2-extreme-points-means-collinear not-less-eq-eq not-less-zero numeral-3-eq-3*)
obtain *a b c* **where** *abc*: {*a*, *b*, *c*} ⊆ *?ep* ∧ *a* ≠ *b* ∧ *b* ≠ *c* ∧ *a* ≠ *c*

proof−
obtain a A **where** $a \in ?ep \wedge A = ?ep - \{a\} \wedge card\ A \geq 2$ **using** *card-ep* **by** *force*
moreover then obtain b B **where** $b \in A \wedge B = A - \{b\} \wedge card\ B \geq 1$
by (*metis Suc-1 Suc-diff-le bot.extremum-uniqueI bot-nat-0.extremum card-Diff-singleton card-eq-0-iff diff-Suc-1 less-Suc-eq-le less-one linorder-not-le subset-emptyI*)
moreover then obtain c C **where** $c \in B \wedge C = B - \{c\} \wedge card\ C \geq 0$
by (*metis One-nat-def bot-nat-0.extremum card.empty equals0I not-less-eq-eq*)
ultimately have $\{a, b, c\} \subseteq ?ep \wedge a \neq b \wedge b \neq c \wedge a \neq c$ **by** *blast*
thus *?thesis* **using** *that* **by** *auto*
qed
{ assume $*$: *card ?ep = 3*
then have *abc*: $?ep = \{a, b, c\}$
by (*metis abc card-3-iff card-gt-0-iff numeral-3-eq-3 order-less-le psubset-card-mono zero-less-Suc*)
obtain d **where** d: $d \in vts \wedge d \neq a \wedge d \neq b \wedge d \neq c$
by (*metis * assms(3) abc ep insertCI nat-less-le subsetI subset-antisym*)
{ assume $d \in interior\ H$
then have $d \in path\text{-}image\ (linepath\ a\ d) \cap interior\ H$ **by** *simp*
then have *?thesis* **using** *ep abc d* **by** *auto*
} moreover
{ assume $***$: $d \notin interior\ H$
let *?p = make-triangle a b c*
have H: $H = convex\ hull\ ?ep$
proof−
have *compact H*
by (*metis assms(1) assms(3) card-eq-0-iff finite-imp-compact-convex-hull gr-implies-not0*)
moreover have *convex H* **using** *convex-convex-hull*[*of vts*] *assms* **by** *blast*
ultimately have $H = closure\ (convex\ hull\ ?ep)$ **using** *Krein-Milman*[*of H*] **by** *fast*
thus *?thesis* **using** *abc* **by** *auto*
qed
then have *interior*: *path-inside ?p = interior H*
using *abc*
by (*metis assms(1,2) affine-hull-convex-hull collinear-affine-hull-collinear convex-convex-hull convex-polygon-frontier-is-path-image2 finite.intros(1) finite-imp-bounded-convex-hull finite-insert inside-frontier-eq-interior path-inside-def triangle-convex-hull triangle-is-convex triangle-is-polygon*)
then have *d-frontier*: $d \in frontier\ H$
by (*metis *** Diff-iff assms(1) UnCI d closure-Un-frontier frontier-def hull-subset in-mono*)
moreover have *path-image ?p = frontier H*
using *convex-polygon-frontier-is-path-image*
by (*metis assms(1,2) H abc affine-hull-convex-hull collinear-affine-hull-collinear convex-polygon-frontier-is-path-image2 triangle-convex-hull triangle-is-convex triangle-is-polygon*)
ultimately have $d \in path\text{-}image\ ?p$ **by** *blast*
moreover have $\neg$ *collinear* $\{a, b, c\}$

by (*metis H assms*(*1*,*2*) *abc affine-hull-convex-hull collinear-affine-hull-collinear*)
moreover then have *distinct* [*a*, *b*, *c*]
by (*metis collinear-2 distinct.simps*(*2*) *distinct-singleton empty-set insert-absorb list.simps*(*15*))
moreover have *d* ∉ {*a*, *b*, *c*} **using** *d* **by** *blast*
ultimately have *?thesis*
using *abc d convex-hull-has-vertex-split-helper*[*of ?p a b c d*]
by (*metis* (*no-types*, *lifting*) *insert-subset interior subset-trans ep*)
}
ultimately have *?thesis* **by** *fast*
} **moreover**
{ **assume** ∗: *card ?ep* ≥ *4*
moreover have {*a*, *b*, *c*} ⊆ *?ep* ∧ *distinct* [*a*, *b*, *c*] **using** *abc* **by** *fastforce*
ultimately have *path-image* (*linepath a b*) ∩ *interior H* ≠ {}
∨ *path-image* (*linepath b c*) ∩ *interior H* ≠ {}
∨ *path-image* (*linepath c a*) ∩ *interior H* ≠ {}
using *convex-hull-extreme-points-vertex-split*[*OF assms*(*1*) *assms*(*4*) ∗] **by** *presburger*
then have *?thesis*
by (*metis* (*no-types*, *lifting*) *ep abc insert-subset subset-trans*)
}
ultimately show *?thesis* **using** *card-ep* **by** *fastforce*
qed

lemma *convex-polygon-has-good-linepath-helper*:
assumes *polygon-of p vts*
assumes *convex* (*path-inside p* ∪ *path-image p*)
assumes *card* (*set vts*) > *3*
obtains *a b* **where** {*a*, *b*} ⊆ *set vts* ∧ *a* ≠ *b* ∧ ¬ *path-image* (*linepath a b*) ⊆ *path-image p*
proof−
let *?H* = *convex hull* (*set vts*)
obtain *a b* **where** *ab*: {*a*, *b*} ⊆ *set vts* ∧ *a* ≠ *b* ∧ *path-image* (*linepath a b*) ∩ *interior ?H* ≠ {}
using *convex-hull-has-vertex-split assms polygon-vts-not-collinear* **unfolding** *polygon-of-def*
by *fastforce*
moreover have *interior ?H* = *path-inside p*
using *assms*(*1*) *assms*(*2*) *convex-polygon-inside-is-convex-hull-interior polygon-convex-iff polygon-of-def*
by *blast*
ultimately have *path-image* (*linepath a b*) ∩ *path-inside p* ≠ {} **by** *simp*
moreover have *path-inside p* ∩ *path-image p* = {} **using** *path-inside-def* **by** *auto*
moreover have *path-image* (*linepath a b*) ⊆ *path-image p* ∪ *path-inside p*
by (*metis ab assms*(*1*) *assms*(*2*) *convex-polygon-is-convex-hull hull-mono path-image-linepath polygon-of-def segment-convex-hull sup-commute*)
ultimately have ¬ *path-image* (*linepath a b*) ⊆ *path-image p* **by** *fast*
thus *?thesis* **using** *ab that* **by** *meson*

qed

lemma *convex-polygon-has-good-linepath*:
assumes *convex* (*path-inside p* ∪ *path-image p*)
assumes *polygon p*
assumes *p* = *make-polygonal-path vts*
assumes *card* (*set vts*) > *3*
shows ∃ *a b*. *good-linepath a b vts*
proof−
let *?T* = *convex hull* (*set vts*)
have *T*: *path-image p* ∪ *path-inside p* = *?T*
by (*metis Un-commute assms*(*1*) *assms*(*2*) *assms*(*3*) *convex-polygon-is-convex-hull*)
obtain *a b* **where** *ab*: *a* ≠ *b* ∧ {*a*, *b*} ⊆ *set vts* ∧ ¬ *path-image* (*linepath a b*) ⊆ *path-image p*
using *convex-polygon-has-good-linepath-helper assms* **unfolding** *polygon-of-def*
by *metis*

let *?S* = *path-image* (*linepath a b*)

have *p-is-frontier*: *frontier ?T* = *path-image p*
using *convex-polygon-frontier-is-path-image assms polygon-of-def polygon-convex-iff*
by *blast*

have *closure ?T* = *?T* **by** (*simp add*: *finite-imp-compact*)
then have *?S* ⊆ *closure ?T* **using** *ab* **by** (*simp add*: *hull-mono segment-convex-hull*)
moreover have *convex ?T* **using** *convex-convex-hull* **by** *auto*
moreover have *convex ?S* **by** *simp*
moreover have *rel-interior ?S* = *open-segment a b*
by (*metis ab path-image-linepath rel-interior-closed-segment*)
moreover have *rel-interior ?T* = *interior ?T*
by (*metis p-is-frontier Diff-empty ab calculation*(*1*) *frontier-def rel-interior-nonempty-interior*)
ultimately have *open-segment a b* ⊆ *interior ?T*
using *subset-rel-interior-convex* **by** (*metis ab p-is-frontier frontier-def rel-frontier-def*)
then have (*open-segment a b*) ∩ *path-image p* = {}
using *p-is-frontier frontier-def* **by** *auto*
then have *closed-segment a b* ∩ *path-image p* = {*a*, *b*}
by (*metis* (*no-types*, *lifting*) *Int-Un-distrib2 Int-absorb2 Un-commute ab assms*(*3*) *closed-segment-eq-open subset-trans sup-bot.right-neutral vertices-on-path-image*)
then have *path-image* (*linepath a b*) ∩ *path-image p* = {*a*, *b*} **by** *simp*
thus *?thesis*
using *ab* **unfolding** *good-linepath-def*
by (*smt* (*verit*, *ccfv-threshold*) *IntI UnCI UnE T assms*(*3*) *hull-mono path-image-linepath segment-convex-hull subset-iff*)
qed

29 Pick's Theorem

definition *integral-inside*:
integral-inside p = {*x*. *integral-vec x* ∧ *x* ∈ *path-inside p*}

definition *integral-boundary*:
integral-boundary p = {*x*. *integral-vec x* ∧ *x* ∈ *path-image p*}

29.1 Pick's Theorem Triangle Case

definition *pick-triangle*:
pick-triangle p a b c ⟷
p = *make-triangle a b c*
∧ *all-integral* [*a*, *b*, *c*]
∧ *distinct* [*a*, *b*, *c*]
∧ ¬ *collinear* {*a*, *b*, *c*}

definition *pick-holds*:
pick-holds p ⟷
(*let I* = *card* {*x*. *integral-vec x* ∧ *x* ∈ *path-inside p*} *in*
let B = *card* {*x*. *integral-vec x* ∧ *x* ∈ *path-image p*} *in*
measure lebesgue (*path-inside p*) = *I* + *B*/*2* − *1*)

lemma *pick-triangle-wlog-helper*:
assumes *pick-triangle p a b c* **and**
I = *card* (*integral-inside p*) **and**
B = *card* (*integral-boundary p*) **and**
integral-inside p = {} **and**
integral-vec d ∧ *d* ∈ *path-image* (*linepath a b*) ∧ *d* ∉ {*a*, *b*, *c*} **and** *d* ∉ {*a*, *b*, *c*} **and**
ih: ⋀*p' a' b' c'*. (*card* (*integral-inside p'*) + *card* (*integral-boundary p'*) < *I* + *B*) ⟹ *pick-triangle p' a' b' c'* ⟹ *pick-holds p'*
shows *measure lebesgue* (*path-inside p*) = *I* + *B*/*2* − *1*
proof−
have *polygon-p*: *polygon p* **using** *triangle-is-polygon assms* **unfolding** *pick-triangle* **by** *presburger*
then have *polygon-of*: *polygon-of p* [*a*, *b*, *c*, *a*]
unfolding *polygon-of-def* **using** *assms* **unfolding** *make-triangle-def pick-triangle* **by** *auto*

let *?p'* = *make-polygonal-path* [*a*, *d*, *b*, *c*, *a*]

have *good-linepath c d* [*a*, *d*, *b*, *c*, *a*] ∧ *path-image* (*make-polygonal-path* [*a*, *d*, *b*, *c*, *a*]) = *path-image p*
using *pick-triangle-basic-split assms* **unfolding** *pick-triangle* **by** *presburger*
then have ∗: *good-linepath d c* [*a*, *d*, *b*, *c*, *a*] ∧ *path-image* (*make-polygonal-path* [*a*, *d*, *b*, *c*, *a*]) = *path-image p*
using *good-linepath-comm* **by** *blast*
have *polygon-new*: *polygon* (*make-polygonal-path* [*a*, *d*, *b*, *c*, *a*])
using *polygon-linepath-split-is-polygon*[*OF polygon-of*, *of 0 a b d* [*a*, *d*, *b*, *c*, *a*]] *assms*
by *force*
have *h1*: *make-polygonal-path* [*a*, *d*, *b*, *c*, *a*] = *make-polygonal-path* ([*a*, *d*, *b*, *c*]

@ [[a, d, b, c] ! 0])
  by *auto*
 have *h2*: *good-linepath d c* ([*a*, *d*, *b*, *c*] @ [[*a*, *d*, *b*, *c*] ! *0*])
  using ∗ **by** *auto*
 have *h3*: (*1*::*nat*) < *length* [*a*, *d*, *b*, *c*] ∧ (*3*::*nat*) < *length* [*a*, *d*, *b*, *c*]
  by *auto*
 then have *polygon-split*: *is-polygon-split* [*a*, *d*, *b*, *c*] *1 3*
  using *good-linepath-implies-polygon-split*[*OF polygon-new h1 h2 h3*] **by** *auto*
 let *?p1* = *make-polygonal-path* (*d* # [*b*] @ [*c*, *d*])
 let *?p2* = *make-polygonal-path* ([*a*] @ [*d*, *c*] @ [] @ [[*a*, *d*, *b*, *c*] ! *0*])
 let *?I1* = *card* {*x*. *integral-vec x* ∧ *x* ∈ *path-inside ?p1*}
 let *?B1* = *card* {*x*. *integral-vec x* ∧ *x* ∈ *path-image ?p1*}
 let *?I2* = *card* {*x*. *integral-vec x* ∧ *x* ∈ *path-inside ?p2*}
 let *?B2* = *card* {*x*. *integral-vec x* ∧ *x* ∈ *path-image ?p2*}
 have *p1-triangle*: *?p1* = *make-triangle d b c*
  unfolding *make-triangle-def* **by** *auto*
 have *p2-triangle*: *?p2* = *make-triangle a d c*
  unfolding *make-triangle-def* **by** *auto*
 have *I-is*: *I* = *card* {*x*. *integral-vec x* ∧ *x* ∈ *path-inside* (*make-polygonal-path* [*a*, *d*, *b*, *c*, *a*])}
  using *path-image-linepath-split*[*of 0* [*a*, *b*, *c*, *a*] *d*] ∗ *assms path-inside-def integral-inside* **by** *presburger*
 have *B-is*: *B* = *card* {*x*. *integral-vec x* ∧ *x* ∈ *path-image* (*make-polygonal-path* [*a*, *d*, *b*, *c*, *a*])}
  using *path-image-linepath-split*[*of 0* [*a*, *b*, *c*, *a*] *d*]
  using ∗ *assms path-inside-def integral-boundary* **by** *presburger*
 have *all-integral-assump*: *all-integral* [*a*, *d*, *b*, *c*]
  using *assms* **unfolding** *all-integral-def pick-triangle* **by** *force*

 have *dist-indh1*: *distinct* [*d*, *b*, *c*]
  using *assms* **unfolding** *pick-triangle* **by** *auto*
 have *coll-indh1*: ¬ *collinear* {*d*, *b*, *c*}
  using *assms pick-triangle*
  by (*smt* (*verit*) *collinear-3-trans dist-indh1 distinct-length-2-or-more in-path-image-imp-collinear insert-commute*)
 have *path-inside-inside*: *path-inside* (*make-polygonal-path* (*d* # [*b*] @ [*c*, *d*])) ⊆ *path-inside p*
  using *polygon-split* **unfolding** *is-polygon-split-def*
  by (*smt* (*verit*) ∗ *One-nat-def Un-iff append-Cons append-Nil diff-Suc-1 drop0 drop-Suc-Cons nth-Cons-0 nth-Cons-Suc numeral-3-eq-3 path-inside-def subsetI take-Suc-Cons take-eq-Nil2*)
 then have *indh1-card1*: *card* {*x*. *integral-vec x* ∧ *x* ∈ *path-inside* (*make-polygonal-path* (*d* # [*b*] @ [*c*, *d*]))}≤ *card* {*x*. *integral-vec x* ∧ *x* ∈ *path-inside p*}
  by (*metis* (*no-types*, *lifting*) *assms*(*4*) *integral-inside Collect-empty-eq card.empty le-zero-eq subsetD*)
 have *indh1-card2*: *card* {*x*. *integral-vec x* ∧ *x* ∈ *path-image* (*make-polygonal-path* (*d* # [*b*] @ [*c*, *d*]))} < *card* {*x*. *integral-vec x* ∧ *x* ∈ *path-image p*}

proof −
have *path-image-union*: *path-image* (*make-polygonal-path* (*d* # [*b*] @ [*c*, *d*])) = *path-image* (*linepath d b*) ∪ *path-image* (*linepath b c*) ∪ *path-image* (*linepath c d*)
using *path-image-cons-union p1-triangle make-triangle-def*
by (*metis* (*no-types*, *lifting*) *inf-sup-aci*(*6*) *list.discI make-polygonal-path.simps*(*3*) *nth-Cons-0*)
have *path-image-db*: *path-image* (*linepath d b*) ⊆ *path-image p*
by (*metis assms*(*5*) *list.discI nth-Cons-0 path-image-cons-union path-image-linepath-union polygon-of polygon-of-def sup.cobounded2 sup.coboundedI1*)
have *path-image-bc*: *path-image* (*linepath b c*) ⊆ *path-image p*
using *assms*(*1*) *linepaths-subset-make-polygonal-path-image*[*of* [*a*, *b*, *c*, *a*] *1*] **unfolding** *pick-triangle make-triangle-def*
by *simp*
have *path-image-cd1*: *path-image* (*linepath c d*) − {*c*, *d*} ⊆ *path-inside p*
using *polygon-split* **unfolding** *is-polygon-split-def*
by (*smt* (*verit*) *One-nat-def* ‹*good-linepath c d* [*a*, *d*, *b*, *c*, *a*] ∧ *path-image* (*make-polygonal-path* [*a*, *d*, *b*, *c*, *a*]) = *path-image p*› *append-Cons append-Nil insert-commute nth-Cons-0 nth-Cons-Suc numeral-3-eq-3 path-image-linepath path-inside-def segment-convex-hull sup.cobounded2*)
have *path-image-cd2*: {*c*, *d*} ⊆ *path-image p*
using *linepaths-subset-make-polygonal-path-image assms*(*1*) **unfolding** *pick-triangle make-triangle-def*
by (*metis* (*no-types*, *lifting*) ‹*good-linepath c d* [*a*, *d*, *b*, *c*, *a*] ∧ *path-image* (*make-polygonal-path* [*a*, *d*, *b*, *c*, *a*]) = *path-image p*› *good-linepath-def subset-trans vertices-on-path-image*)
have *path-image* (*linepath c d*) ⊆ *path-image p* ∪ *path-inside p*
using *path-image-cd1 path-image-cd2* **by** *auto*
moreover have *integral-inside p* = {} **using** *assms* **by** *force*
ultimately have *path-image-cd*: *integral-boundary* (*linepath c d*) ⊆ *integral-boundary p* **unfolding** *integral-inside integral-boundary* **by** *blast*
have *a-neq-d*: *a* ≠ *d*
using *assms*(*5*) **by** *auto*
have *a-neq-c*: *a* ≠ *c*
using *assms*(*1*) **unfolding** *pick-triangle* **by** *simp*
have *a-in-image*: *a* ∈ *path-image p*
using *assms*(*1*) **unfolding** *pick-triangle make-triangle-def* **using** *vertices-on-path-image*
by *fastforce*
have *path-image* (*linepath c d*) ∩ *path-image p* = {*c*, *d*}
using ∗ **unfolding** *good-linepath-def*
by (*smt* (*verit*, *ccfv-SIG*) *One-nat-def h1 insert-commute is-polygon-cut-def is-polygon-split-def nth-Cons-0 nth-Cons-Suc numeral-3-eq-3 path-image-linepath polygon-split segment-convex-hull*)
then have *a-not-in1*: *a* ∉ *path-image* (*linepath c d*)
using *a-neq-c a-neq-d a-in-image* **by** *blast*
have *a-not-in2*: *a* ∉ *path-image* (*linepath d b*)
using *Int-closed-segment assms*(*5*) **by** *auto*
have *a-not-in3*: *a* ∉ *path-image* (*linepath b c*)
by (*metis* (*no-types*, *lifting*) *assms*(*1*) *in-path-image-imp-collinear insert-commute pick-triangle*)

then have $a \notin$ *path-image* (*linepath d b*) $\cup$ *path-image* (*linepath b c*) $\cup$ *path-image* (*linepath c d*)
using *a-not-in1 a-not-in2 a-not-in3* **by** *simp*
then have $a \in$ *integral-boundary p* $\wedge$ $a \notin$ *integral-boundary* (*make-polygonal-path* [*d*, *b*, *c*, *d*])
using *path-image-union* **using** *integral-boundary a-in-image all-integral-assump all-integral-def* **by** *auto*
then have *strict-subset*: *integral-boundary* (*make-polygonal-path* [*d*, *b*, *c*, *d*]) $\subset$ *integral-boundary p*
using *path-image-union path-image-db path-image-bc path-image-cd*
unfolding *integral-boundary* **by** *auto*
have *integral-inside* (*make-polygonal-path* [*d*, *b*, *c*, *d*]) = {}
using *path-inside-inside assms* **unfolding** *integral-inside* **by** *auto*
then show *?thesis* **using** *assms*(*2*−*3*) *strict-subset bounded-finite*
using *finite-path-inside finite-path-image*
by (*simp add*: *integral-boundary polygon-p psubset-card-mono*)
qed
have *fewer-points-p1*: *card* {*x*. *integral-vec x* $\wedge$ *x* $\in$ *path-inside* (*make-polygonal-path* (*d* # [*b*] @ [*c*, *d*]))} +
card {*x*. *integral-vec x* $\wedge$ *x* $\in$ *path-image* (*make-polygonal-path* (*d* # [*b*] @ [*c*, *d*]))}
< *card* {*x*. *integral-vec x* $\wedge$ *x* $\in$ *path-inside p*} +
card {*x*. *integral-vec x* $\wedge$ *x* $\in$ *path-image p*}
using *indh1-card1 indh1-card2* **by** *linarith*
have *indh-1*: *Sigma-Algebra.measure lebesgue* (*path-inside ?p1*) = *real ?I1* + *real ?B1* / *2* − *1*
using *assms fewer-points-p1 p1-triangle all-integral-assump dist-indh1 coll-indh1 all-integral-def*
unfolding *pick-holds pick-triangle integral-inside integral-boundary* **by** *simp*

have *dist-indh2*: *distinct* [*a*, *d*, *c*]
using *assms* **unfolding** *pick-triangle* **by** *auto*
have *coll-indh2*: ¬ *collinear* {*a*, *d*, *c*}
using *assms pick-triangle*
by (*smt* (*verit*) *collinear-3-trans dist-indh2 distinct-length-2-or-more in-path-image-imp-collinear insert-commute*)
have *path-inside-inside*: *path-inside* (*make-polygonal-path* (*a* # [*d*] @ [*c*, *a*])) $\subseteq$ *path-inside p*
using *polygon-split* **unfolding** *is-polygon-split-def*
by (*smt* (*verit*) ∗ *One-nat-def Un-iff append-Cons append-Nil diff-Suc-1 drop0 drop-Suc-Cons nth-Cons-0 nth-Cons-Suc numeral-3-eq-3 path-inside-def subsetI take-Suc-Cons take-eq-Nil2*)
then have *indh2-card1*: *card* {*x*. *integral-vec x* $\wedge$ *x* $\in$ *path-inside* (*make-polygonal-path* (*a* # [*d*] @ [*c*, *a*]))}$\leq$ *card* {*x*. *integral-vec x* $\wedge$ *x* $\in$ *path-inside p*}
by (*metis* (*no-types*, *lifting*) *assms*(*4*) *integral-inside Collect-empty-eq card.empty le-zero-eq subsetD*)
have *indh2-card2*: *card* {*x*. *integral-vec x* $\wedge$ *x* $\in$ *path-image* (*make-polygonal-path* (*a* # [*d*] @ [*c*, *a*]))} < *card* {*x*. *integral-vec x* $\wedge$ *x* $\in$ *path-image p*}

proof −
have *path-image-union*: *path-image* (*make-polygonal-path* (*a* # [*d*] @ [*c*, *a*])) = *path-image* (*linepath a d*) ∪ *path-image* (*linepath d c*) ∪ *path-image* (*linepath c a*)
using *path-image-cons-union p2-triangle make-triangle-def*
by (*metis Un-assoc append.left-neutral append-Cons list.discI make-polygonal-path.simps*(*3*) *nth-Cons-0*)
have *path-image-ad*: *path-image* (*linepath a d*) ⊆ *path-image p*
by (*metis* ‹*good-linepath c d* [*a*, *d*, *b*, *c*, *a*] ∧ *path-image* (*make-polygonal-path* [*a*, *d*, *b*, *c*, *a*]) = *path-image p*› *inf-sup-absorb le-iff-inf list.discI nth-Cons-0 path-image-cons-union*)
have *path-image-ca*: *path-image* (*linepath c a*) ⊆ *path-image p*
using *assms*(*1*) *linepaths-subset-make-polygonal-path-image*[*of* [*a*, *b*, *c*, *a*] *2*] **unfolding** *pick-triangle make-triangle-def*
by *simp*
have *path-image-cd1*: *path-image* (*linepath d c*) − {*c*, *d*} ⊆ *path-inside p*
using *polygon-split* **unfolding** *is-polygon-split-def*
by (*smt* (*verit*) *One-nat-def* ‹*good-linepath c d* [*a*, *d*, *b*, *c*, *a*] ∧ *path-image* (*make-polygonal-path* [*a*, *d*, *b*, *c*, *a*]) = *path-image p*› *append-Cons append-Nil insert-commute nth-Cons-0 nth-Cons-Suc numeral-3-eq-3 path-image-linepath path-inside-def segment-convex-hull sup.cobounded2*)
have *path-image-cd2*: {*c*, *d*} ⊆ *path-image p*
using *linepaths-subset-make-polygonal-path-image assms*(*1*) **unfolding** *pick-triangle make-triangle-def*
by (*metis* (*no-types*, *lifting*) ‹*good-linepath c d* [*a*, *d*, *b*, *c*, *a*] ∧ *path-image* (*make-polygonal-path* [*a*, *d*, *b*, *c*, *a*]) = *path-image p*› *good-linepath-def subset-trans vertices-on-path-image*)
have *path-image* (*linepath d c*) ⊆ *path-image p* ∪ *path-inside p*
using *path-image-cd1 path-image-cd2* **by** *auto*
moreover have *integral-inside p* = {} **using** *assms* **by** *force*
ultimately have *path-image-cd*: *integral-boundary* (*linepath d c*) ⊆ *integral-boundary p* **unfolding** *integral-inside integral-boundary* **by** *blast*
have *b-neq-d*: *b* ≠ *d*
using *assms*(*5*) **by** *auto*
have *b-neq-c*: *b* ≠ *c*
using *assms*(*1*) **unfolding** *pick-triangle* **by** *simp*
have *b-in-image*: *b* ∈ *path-image p*
using *assms*(*1*) **unfolding** *pick-triangle make-triangle-def* **using** *vertices-on-path-image*
by *fastforce*
have *path-image* (*linepath d c*) ∩ *path-image p* = {*d*, *c*}
using ∗ **unfolding** *good-linepath-def*
by (*smt* (*verit*, *ccfv-SIG*) *One-nat-def h1 insert-commute is-polygon-cut-def is-polygon-split-def nth-Cons-0 nth-Cons-Suc numeral-3-eq-3 path-image-linepath polygon-split segment-convex-hull*)
then have *b-not-in1*: *b* ∉ *path-image* (*linepath d c*)
using *b-neq-c b-neq-d b-in-image* **by** *blast*
have *b-not-in2*: *b* ∉ *path-image* (*linepath a d*)
using *Int-closed-segment assms*(*5*) **by** *auto*
have *b-not-in3*: *b* ∉ *path-image* (*linepath c a*)
by (*metis* (*no-types*, *lifting*) *assms*(*1*) *in-path-image-imp-collinear insert-commute*

pick-triangle)
then have $b \notin$ *path-image* (*linepath a d*) $\cup$ *path-image* (*linepath d c*) $\cup$ *path-image* (*linepath c a*)
using *b-not-in1 b-not-in2 b-not-in3* **by** *simp*
then have $b \in$ *integral-boundary p* $\wedge$ $b \notin$ *integral-boundary* (*make-polygonal-path* [*a*, *d*, *c*, *a*])
using *path-image-union* **using** *integral-boundary b-in-image all-integral-assump all-integral-def* **by** *auto*
then have *strict-subset*: *integral-boundary* (*make-polygonal-path* [*a*, *d*, *c*, *a*]) $\subset$ *integral-boundary p*
using *path-image-union path-image-ad path-image-ca path-image-cd*
unfolding *integral-boundary* **by** *auto*
have *integral-inside* (*make-polygonal-path* [*a*, *d*, *c*, *a*]) = {}
using *path-inside-inside assms* **unfolding** *integral-inside* **by** *auto*
then show *?thesis* **using** *assms*(*2*−*3*) *strict-subset bounded-finite*
using *finite-path-inside finite-path-image*
by (*simp add*: *integral-boundary polygon-p psubset-card-mono*)
qed
have *fewer-points-p2*: *card* {*x*. *integral-vec x* $\wedge$ *x* $\in$ *path-inside* (*make-polygonal-path* ([*a*, *d*, *c*, *a*]))} +
card {*x*. *integral-vec x* $\wedge$ *x* $\in$ *path-image* (*make-polygonal-path* ([*a*, *d*, *c*, *a*]))}
< *card* {*x*. *integral-vec x* $\wedge$ *x* $\in$ *path-inside p*} +
card {*x*. *integral-vec x* $\wedge$ *x* $\in$ *path-image p*}
using *indh2-card1 indh2-card2* **by** *simp*
have *indh-2*: *Sigma-Algebra.measure lebesgue* (*path-inside ?p2*) = *real ?I2* + *real ?B2* / *2* − *1*
using *fewer-points-p2* **using** *assms fewer-points-p2 p2-triangle all-integral-assump dist-indh2 coll-indh2 all-integral-def*
unfolding *pick-holds pick-triangle integral-inside integral-boundary* **by** *simp*

have *Sigma-Algebra.measure lebesgue* (*path-inside ?p1*) = *real ?I1* + *real ?B1* / *2* − *1* $\Longrightarrow$
Sigma-Algebra.measure lebesgue (*path-inside ?p2*) = *real ?I2* + *real ?B2* / *2* − *1* $\Longrightarrow$
I = *card* {*x*. *integral-vec x* $\wedge$ *x* $\in$ *path-inside* (*make-polygonal-path* [*a*, *d*, *b*, *c*, *a*])} $\Longrightarrow$
B = *card* {*x*. *integral-vec x* $\wedge$ *x* $\in$ *path-image* (*make-polygonal-path* [*a*, *d*, *b*, *c*, *a*])} $\Longrightarrow$
all-integral [*a*, *d*, *b*, *c*] $\Longrightarrow$
Sigma-Algebra.measure lebesgue (*path-inside* (*make-polygonal-path* [*a*, *d*, *b*, *c*, *a*])) =
real I + *real B* / *2* − *1*
using *pick-split-union*[*OF polygon-split*, *of* [*a*] [*b*] [] *d c ?p'*] **by** *auto*
then have *Sigma-Algebra.measure lebesgue* (*path-inside* (*make-polygonal-path* [*a*, *d*, *b*, *c*, *a*])) =
real I + *real B* / *2* − *1*
using *I-is B-is all-integral-assump indh-1 indh-2* **by** *auto*
thus *measure lebesgue* (*path-inside p*) = *I* + *B*/*2* − *1*

using *path-image-linepath-split*[*of 0* [*a*, *b*, *c*, *a*] *d*] **by** (*metis path-inside-def* ∗)
qed

lemma *pick-triangle-helper*:
assumes *pick-triangle p a b c* **and**
I = *card* (*integral-inside p*) **and**
B = *card* (*integral-boundary p*) **and**
integral-inside p = {} **and**
integral-vec d ∧ *d* ∉ {*a*, *b*, *c*} **and** *d* ∉ {*a*, *b*, *c*} **and**
d ∈ *path-image* (*linepath a b*)
∨ *d* ∈ *path-image* (*linepath b c*)
∨ *d* ∈ *path-image* (*linepath c a*) **and**
ih: ⋀*p′ a′ b′ c′*. (*card* (*integral-inside p′*) + *card* (*integral-boundary p′*) < *I* + *B*) ⟹ *pick-triangle p′ a′ b′ c′* ⟹ *pick-holds p′*
shows *measure lebesgue* (*path-inside p*) = *I* + *B*/*2* − *1*
proof−
{ **assume** *d* ∈ *path-image* (*linepath a b*)
then have *?thesis* **using** *pick-triangle-wlog-helper assms* **by** *blast*
} **moreover**
{ **assume** ∗: *d* ∈ *path-image* (*linepath b c*)
let *?p′* = *make-polygonal-path* (*rotate-polygon-vertices* [*a*, *b*, *c*, *a*] *1*)
let *?I′* = *card* (*integral-inside ?p′*)
let *?B′* = *card* (*integral-boundary ?p′*)

have *p′-p*: *path-image ?p′* = *path-image p* ∧ *path-inside ?p′* = *path-inside p*
unfolding *path-inside-def*
using *assms*(*1*) *make-triangle-def pick-triangle polygon-vts-arb-rotation triangle-is-polygon*
by *auto*

have *rotate-polygon-vertices* [*a*, *b*, *c*, *a*] *1* = [*b*, *c*, *a*, *b*]
unfolding *rotate-polygon-vertices-def* **by** *simp*
then have *pick-triangle-p′*: *pick-triangle ?p′ b c a*
using *assms* **unfolding** *pick-triangle*
by (*smt* (*verit*, *best*) *all-integral-def distinct-length-2-or-more insert-commute list.simps*(*15*) *make-triangle-def*)
then have *measure lebesgue* (*path-inside ?p′*) = *?I′* + *?B′*/*2* − *1*
using *pick-triangle-wlog-helper*[*of ?p′ b c a ?I′ ?B′ d*] *assms*
using *integral-boundary integral-inside* ∗ *insert-commute pick-triangle-p′ p′-p*
by *auto*
moreover have *?I′* = *I* ∧ *?B′* = *B* **using** *p′-p integral-boundary integral-inside assms*(*2*) *assms*(*3*) **by** *presburger*
ultimately have *?thesis* **using** *p′-p* **by** *auto*
} **moreover**
{ **assume** ∗: *d* ∈ *path-image* (*linepath c a*)
let *?p′* = *make-polygonal-path* (*rotate-polygon-vertices* [*a*, *b*, *c*, *a*] *2*)
let *?I′* = *card* (*integral-inside ?p′*)
let *?B′* = *card* (*integral-boundary ?p′*)

have *p'-p*: *path-image ?p' = path-image p ∧ path-inside ?p' = path-inside p*
unfolding *path-inside-def*
using *assms(1) make-triangle-def pick-triangle polygon-vts-arb-rotation triangle-is-polygon*
by *auto*

have *rotate-polygon-vertices [a, b, c, a] 1 = [b, c, a, b]*
unfolding *rotate-polygon-vertices-def* **by** *simp*
also have *rotate-polygon-vertices ... 1 = [c, a, b, c]*
unfolding *rotate-polygon-vertices-def* **by** *simp*
ultimately have *rotate-polygon-vertices [a, b, c, a] 2 = [c, a, b, c]*
by (*metis Suc-1 arb-rotation-as-single-rotation*)
then have *pick-triangle-p'*: *pick-triangle ?p' c a b*
using *assms* **unfolding** *pick-triangle*
by (*smt* (*verit*, *best*) *all-integral-def distinct-length-2-or-more insert-commute list.simps(15) make-triangle-def*)
then have *measure lebesgue* (*path-inside ?p'*) = *?I' + ?B'/2 − 1*
using *pick-triangle-wlog-helper*[*of ?p' c a b ?I' ?B' d*] *assms*
using *integral-boundary integral-inside ∗ insert-commute pick-triangle-p' p'-p*
by *auto*
moreover have *?I' = I ∧ ?B' = B* **using** *p'-p integral-boundary integral-inside assms(2) assms(3)* **by** *presburger*
ultimately have *?thesis* **using** *p'-p* **by** *auto*
}
ultimately show *?thesis* **using** *assms* **by** *blast*
qed

lemma *triangle-3-split-helper*:
fixes *a b* :: *'a*::*euclidean-space*
assumes *a ∈ frontier S*
assumes *b ∈ interior S*
assumes *convex S*
assumes *closed S*
shows *path-image* (*linepath a b*) ∩ *frontier S* = {*a*}
proof−
let *?L = path-image* (*linepath a b*)
have *a ∈ S ∧ b ∈ S* **using** *assms frontier-subset-closed interior-subset* **by** *auto*
then have *?L ⊆ S*
using *assms hull-minimal segment-convex-hull* **by** (*simp add*: *closed-segment-subset*)
then have *?L ⊆ closure S* **using** *assms(4)* **by** *auto*
moreover have *convex ?L* **by** *simp*
moreover have *?L ∩ interior S ≠* {} **using** *assms(2)* **by** *auto*
moreover then have ¬ *?L ⊆ rel-frontier S*
by (*metis DiffE assms(2) interior-subset-rel-interior pathfinish-in-path-image pathfinish-linepath rel-frontier-def subsetD*)
ultimately have *rel-interior ?L ⊆ rel-interior S*
using *subset-rel-interior-convex*[*of ?L S*] *assms* **by** *fastforce*
then have *open-segment a b ⊆ interior S*
by (*metis all-not-in-conv assms(2) empty-subsetI open-segment-eq-empty' path-image-linepath*

rel-interior-closed-segment rel-interior-nonempty-interior)
moreover have *?L = closed-segment a b* **by** *auto*
moreover have *interior S ∩ frontier S = {}* **by** (*simp add*: *frontier-def*)
ultimately have *?L ∩ frontier S ⊆ {a, b}*
by (*smt* (*verit*) *Diff-iff disjoint-iff inf-commute inf-le1 open-segment-def subsetD subsetI*)
moreover have *b ∉ frontier S* **by** (*simp add*: *assms*(*2*) *frontier-def*)
ultimately show *?thesis* **using** *assms*(*1*) **by** *auto*
qed

lemma *unit-triangle-interior-point-not-collinear-e1-e2*:
assumes *p = make-triangle* (*vector* [*0*, *0*]) (*vector* [*1*, *0*]) (*vector* [*0*, *1*])
(**is** *p = make-triangle ?O ?e1 ?e2*)
assumes *z ∈ path-inside p*
shows *¬ collinear* {*?O*, *?e1*, *z*}
proof−
have *path-inside p = interior* (*convex hull* {*?O*, *?e1*, *?e2*})
by (*metis assms*(*1*) *bounded-convex-hull bounded-empty bounded-insert convex-convex-hull convex-polygon-frontier-is-path-image2 inside-frontier-eq-interior path-inside-def triangle-convex-hull triangle-is-convex triangle-is-polygon unit-triangle-vts-not-collinear*)
then have *z ∈ interior* (*convex hull* {*?O*, *?e1*, *?e2*}) **using** *assms* **by** *simp*
then have *z*: *z$1 > 0 ∧ z$2 > 0*
using *assms*(*1*) *assms*(*2*) *unit-triangle-interior-char make-triangle-def* **by** *blast*
have *abc*: *?O$1 = 0 ∧ ?O$2 = 0 ∧ ?e1$2 = 0 ∧ ?e2$1 = 0* **by** *simp*

show *¬ collinear* {*?O*, *?e1*, *z*}
proof(*rule ccontr*)
assume *¬ ¬ collinear* {*?O*, *?e1*, *z*}
then have ∗: *collinear* {*?O*, *?e1*, *z*} **by** *blast*
then obtain *u c1 c2* **where** *u*: *?O − ?e1 = c1 ∗R u ∧ ?e1 − z = c2 ∗R u*
unfolding *collinear-def* **by** *blast*
moreover have *c1 ≠ 0*
proof−
have (*?O − ?e1*)$*1* = −*1* **by** *simp*
moreover have (*?O − ?e1*)$*1* = (*c1 ∗R u*)$*1* **using** *u* **by** *presburger*
ultimately show *?thesis* **by** *force*
qed
moreover have (*?O − ?e1*)$*2* = *0* **by** *simp*
moreover have (*?O − ?e1*)$*2* = (*c1 ∗R u*)$*2* **by** (*simp add*: *calculation*(*1*))
ultimately have *u$2 = 0* **by** *auto*
thus *False*
by (*smt* (*verit*, *ccfv-threshold*) *u abc scaleR-eq-0-iff vector-minus-component vector-scaleR-component z*)
qed
qed

lemma *triangle-interior-point-not-collinear-vertices-wlog-helper*:
assumes *p = make-triangle a b c*
assumes *polygon p*

assumes $z \in$ *path-inside p*
shows $\neg$ *collinear* $\{a, b, z\}$
proof−
let *?O* = (*vector* [*0*, *0*])::(*real^2*)
let *?e1* = (*vector* [*1*, *0*])::(*real^2*)
let *?e2* = (*vector* [*0*, *1*])::(*real^2*)
let *?M* = *triangle-affine a b c*
have *a*: *?M ?O* = *a*
using *triangle-affine-e1-e2* **by** *blast*
have *b*: *?M ?e1* = *b* **using** *triangle-affine-e1-e2* **by** *simp*
have *c*: *?M ?e2* = *c* **using** *triangle-affine-e1-e2* **by** *simp*

have *abc-not-collinear*: $\neg$ *collinear* $\{a, b, c\}$
using *assms polygon-vts-not-collinear* **unfolding** *make-triangle-def polygon-of-def*
by (*metis* (*no-types*, *lifting*) *empty-set insertCI insert-absorb insert-commute list.simps*(*15*))

have *convex hull* $\{a, b, c\}$ = *convex hull* $\{$*?M ?O*, *?M ?e1*, *?M ?e2*$\}$
using *a b c* **by** *simp*
also have ... = *?M* ‘ (*convex hull* $\{$*?O*, *?e1*, *?e2*$\}$)
using *calculation triangle-affine-img* **by** *blast*
also have *interior-preserve*: *interior* ... = *?M* ‘ (*interior* (*convex hull* $\{$*?O*, *?e1*, *?e2*$\}$))
using *triangle-affine-preserves-interior*[*of ?M a b c - convex hull* $\{$*?O*, *?e1*, *?e2*$\}$]
using *abc-not-collinear*
by *presburger*
finally have *z*: *z* $\in$ *?M* ‘ (*interior* (*convex hull* $\{$*?O*, *?e1*, *?e2*$\}$))
using *assms*(*1*) *assms*(*2*) *assms*(*3*) *make-triangle-def polygon-of-def triangle-inside-is-convex-hull-interior*
by *auto*
then obtain *z'* **where** *z'*: *z'* $\in$ *interior* (*convex hull* $\{$*?O*, *?e1*, *?e2*$\}$) $\wedge$ *?M z'* = *z* **by** *fast*
then have $\neg$ *collinear* $\{$*?O*, *?e1*, *z'*$\}$
by (*metis convex-convex-hull convex-polygon-frontier-is-path-image2 finite.intros*(*1*) *finite-imp-bounded-convex-hull finite-insert inside-frontier-eq-interior path-inside-def triangle-convex-hull triangle-is-convex triangle-is-polygon unit-triangle-interior-point-not-collinear-e1-e2 unit-triangle-vts-not-collinear*)
then have *z'-notin*: *z'* $\notin$ *affine hull* $\{$*?O*, *?e1*$\}$ **using** *affine-hull-3-imp-collinear* **by** *blast*
then have *?M z'* $\notin$ *affine hull* $\{$*?M ?O*, *?M ?e1*$\}$
proof−
have *inj ?M* **using** *triangle-affine-inj abc-not-collinear* **by** *blast*
then have *?M z'* $\notin$ *?M* ‘ (*affine hull* $\{$*?O*, *?e1*$\}$) **using** *z'-notin* **by** (*simp add*: *inj-image-mem-iff*)
moreover have *?M* ‘ (*affine hull* $\{$*?O*, *?e1*$\}$) = *affine hull* $\{$*?M ?O*, *?M ?e1*$\}$
using *triangle-affine-preserves-affine-hull*[*of - a b c*] *abc-not-collinear* **by** *simp*
ultimately show *?thesis* **by** *blast*
qed

then have $z \notin$ *affine hull* $\{a, b\}$ **using** $a\ b\ z'$ **by** *argo*
thus *?thesis*
by (*metis interior-preserve z affine-hull-convex-hull affine-hull-nonempty-interior collinear-2 collinear-3-affine-hull collinear-affine-hull-collinear empty-iff insert-absorb2 triangle-affine-img unit-triangle-vts-not-collinear* z')
qed

lemma *triangle-interior-point-not-collinear-vertices*:
assumes $p =$ *make-triangle a b c*
assumes *polygon p*
assumes $z \in$ *path-inside p*
shows $\neg$ *collinear* $\{a, b, z\} \wedge \neg$ *collinear* $\{a, c, z\} \wedge \neg$ *collinear* $\{b, c, z\}$
proof−
let *?p1* $=$ *make-triangle b c a*
let *?p2* $=$ *make-triangle c a b*
have *p1*: *?p1* $=$ *make-polygonal-path* (*rotate-polygon-vertices* $[a, b, c, a]$ *1*)
using *assms* **unfolding** *make-triangle-def rotate-polygon-vertices-def* **by** *fastforce*
have *p2*: *?p2* $=$ *make-polygonal-path* (*rotate-polygon-vertices* $[a, b, c, a]$ *2*)
using *assms* **unfolding** *make-triangle-def rotate-polygon-vertices-def* **by** (*simp add*: *numeral-Bit0*)

have *path-inside ?p1* $=$ *path-inside p* $\wedge$ *path-inside ?p2* $=$ *path-inside p*
using *p1 p2* **unfolding** *path-inside-def*
using *assms*(*1*) *assms*(*2*) *make-triangle-def polygon-vts-arb-rotation* **by** *force*
then have $z \in$ *path-inside ?p1* $\wedge$ $z \in$ *path-inside ?p2* **using** *assms* **by** *force*
moreover have *polygon ?p1* $\wedge$ *polygon ?p2*
using *assms make-triangle-def p1 p2 rotation-is-polygon* **by** *presburger*
ultimately show *?thesis*
using *assms triangle-interior-point-not-collinear-vertices-wlog-helper*
by (*smt* (*verit*, *best*) *insert-commute*)
qed

lemma *triangle-3-split*:
assumes $p =$ *make-triangle a b c*
assumes *polygon p*
assumes $z \in$ *path-inside p*
shows *is-polygon-split-path* $[a, b, c]$ *0 1* $[z]$
is-polygon-split $[a, z, b, c]$ *1 3*
$a \notin$ *path-image* (*make-triangle z b c*) $\cup$ *path-inside* (*make-triangle z b c*)
$b \notin$ *path-image* (*make-triangle a z c*) $\cup$ *path-inside* (*make-triangle a z c*)
$c \notin$ *path-image* (*make-triangle a b z*) $\cup$ *path-inside* (*make-triangle a b z*)
proof−
let *?q* $=$ *make-polygonal-path* $[a, z, b, c, a]$
let *?cutpath* $=$ *make-polygonal-path* $[a, z, b]$
let *?vts* $= [a, b, c, a]$

let *?l1* $=$ *linepath a z*

let *?l2* = *linepath z b*
let *?S* = *path-inside p* ∪ *path-image p*
have *convex* (*path-inside p*)
using *triangle-is-convex assms*(*1*,*2*) *polygon-vts-not-collinear* **unfolding** *make-triangle-def*
by (*simp add*: *polygon-of-def triangle-inside-is-convex-hull-interior*)
then have *convex*: *convex* (*path-inside p* ∪ *path-image p*)
using *polygon-convex-iff assms*(*2*) **by** *simp*
then have *frontier*: *frontier ?S* = *path-image p*
using *convex-polygon-frontier-is-path-image3* **by** (*simp add*: *assms*(*2*) *sup-commute*)
have *interior*: *interior ?S* = *path-inside p*
by (*metis Jordan-inside-outside-real2 closed-path-def* ‹*convex* (*path-inside p*)› *assms*(*2*) *closure-Un-frontier convex-interior-closure interior-open path-inside-def polygon-def*)

have *not-collinear*: ¬ *collinear* {*a*, *b*, *z*} ∧ ¬ *collinear* {*a*, *c*, *z*} ∧ ¬ *collinear* {*b*, *c*, *z*}
using *triangle-interior-point-not-collinear-vertices assms*(*1*) *assms*(*2*) *assms*(*3*) **by** *blast*

have *a* = *pathstart ?cutpath* ∧ *b* = *pathfinish ?cutpath* **by** *simp*
moreover have *a* ≠ *b*
by (*metis assms*(*1*) *assms*(*2*) *constant-linepath-is-not-loop-free make-polygonal-path.simps*(*4*) *make-triangle-def not-loop-free-first-component polygon-def simple-path-def*)
moreover have *polygon p* **by** (*simp add*: *assms*(*2*))
moreover have {*a*, *b*} ⊆ *set ?vts* **by** *force*
moreover have *simple-path ?cutpath*
by (*simp add*: *insert-commute not-collinear not-collinear-loopfree-path simple-path-def*)
moreover have *path-image ?cutpath* ∩ *path-image p* = {*a*, *b*}
proof−
have {*a*, *b*} ⊆ *path-image ?cutpath* ∩ *path-image p*
by (*metis* (*no-types*, *lifting*) *Int-subset-iff Un-subset-iff assms*(*1*) *insert-is-Un list.simps*(*15*) *make-triangle-def vertices-on-path-image*)
moreover have *path-image ?cutpath* ∩ *path-image p* ⊆ {*a*, *b*}
proof−
have *z* ∈ *interior ?S* **using** *assms interior* **by** *fast*
moreover then have *a* ∈ *frontier ?S* ∧ *b* ∈ *frontier ?S*
using *vertices-on-path-image*
using ‹{*a*, *b*} ⊆ *path-image* (*make-polygonal-path* [*a*, *z*, *b*]) ∩ *path-image p*› *frontier* **by** *force*
moreover have *closed ?S* **using** *frontier frontier-subset-eq* **by** *auto*
ultimately have *path-image ?l1* ∩ *path-image p* = {*a*} ∧ *path-image ?l2* ∩ *path-image p* = {*b*}
using *triangle-3-split-helper convex frontier*
by (*metis* (*no-types*, *lifting*) *insert-commute path-image-linepath segment-convex-hull*)
moreover have *path-image ?cutpath* = *path-image ?l1* ∪ *path-image ?l2*
by (*metis list.discI make-polygonal-path.simps*(*3*) *nth-Cons-0 path-image-cons-union*)
ultimately show *?thesis* **by** *blast*
qed

ultimately show *?thesis* **by** *blast*
qed
moreover have *path-image ?cutpath* ∩ *path-inside p* ≠ {}
by (*metis* (*no-types*, *opaque-lifting*) *Int-Un-distrib2 Un-absorb2 Un-empty assms*(*3*) *insert-disjoint*(*2*) *list.simps*(*15*) *vertices-on-path-image*)
ultimately have *cutpath*: *is-polygon-cut-path ?vts ?cutpath*
using *assms* **unfolding** *make-triangle-def is-polygon-cut-path-def* **by** *simp*
thus *1*: *is-polygon-split-path* [*a*, *b*, *c*] *0 1* [*z*]
using *polygon-cut-path-to-split-path assms*(*2*) **by** (*simp add*: *assms*(*1*,*2*) *make-triangle-def*)

let *?l* = *linepath z c*
let *?vts* = [*a*, *z*, *b*, *c*, *a*]

have *c-noton-cutpath*: *c* ∉ *path-image ?cutpath*
by (*smt* (*verit*) *UnE assms*(*1*) *assms*(*2*) *assms*(*3*) *in-path-image-imp-collinear insert-commute make-polygonal-path.simps*(*3*) *neq-Nil-conv nth-Cons-0 path-image-cons-union triangle-interior-point-not-collinear-vertices*)

have *z* ≠ *c*
proof−
have *c* ∈ *path-image p*
by (*metis assms*(*1*) *insert-subset list.simps*(*15*) *make-triangle-def vertices-on-path-image*)
moreover have *path-image p* ∩ *path-inside p* = {}
by (*simp add*: *disjoint-iff inside-def path-inside-def*)
ultimately show *?thesis* **using** *assms*(*3*) **by** *blast*
qed
moreover have *polygon-q*: *polygon ?q*
using *1*
using *nth-Cons-0* **by** (*fastforce simp*: *is-polygon-split-path-def*)

moreover have {*z*, *c*} ⊆ *set ?vts* **by** *force*
moreover have *l-q-int*: *path-image ?l* ∩ *path-image ?q* = {*z*, *c*}
proof−
have {*z*, *c*} ⊆ *path-image ?l* ∩ *path-image ?q*
by (*metis* (*no-types*, *lifting*) *Int-subset-iff calculation*(*3*) *dual-order.trans hull-subset path-image-linepath segment-convex-hull vertices-on-path-image*)
moreover
{ **fix** *x*
assume ∗: *x* ∈ *path-image ?l* ∩ *path-image ?q* ∧ *x* ≠ *z* ∧ *x* ≠ *c*
then have *x* ∈ *path-image ?q* **by** *blast*
then have *x* ∈ *path-image* (*linepath a z*)
∨ *x* ∈ *path-image* (*linepath z b*)
∨ *x* ∈ *path-image* (*linepath b c*)
∨ *x* ∈ *path-image* (*linepath c a*)
by (*metis UnE list.discI make-polygonal-path.simps*(*3*) *nth-Cons-0 path-image-cons-union*)
moreover
{ **assume** *x* ∈ *path-image* (*linepath a z*)
then have *x* ∈ *path-image* (*linepath a z*) ∧ *x* ∈ *path-image* (*linepath z c*)
using ∗ **by** *blast*

moreover have $z \in$ *path-image* (*linepath a z*) $\wedge$ $z \in$ *path-image* (*linepath z c*) **by** *simp*
moreover have $x \neq z$ **using** $*$ **by** *blast*
ultimately have *collinear* $\{a, z, c\}$
by (*smt* (*verit*, *best*) *collinear-3-trans in-path-image-imp-collinear insert-commute*)
then have *False* **using** *not-collinear* **by** (*simp add*: *insert-commute*)
} moreover
{ assume $x \in$ *path-image* (*linepath z b*)
then have $x \in$ *path-image* (*linepath z b*) $\wedge$ $x \in$ *path-image* (*linepath z c*) **using** $*$ **by** *blast*
moreover have $z \in$ *path-image* (*linepath z b*) $\wedge$ $z \in$ *path-image* (*linepath z c*) **by** *simp*
moreover have $x \neq z$ **using** $*$ **by** *blast*
ultimately have *collinear* $\{z, b, c\}$
by (*smt* (*verit*, *best*) *collinear-3-trans in-path-image-imp-collinear insert-commute*)
then have *False* **using** *not-collinear* **by** (*simp add*: *insert-commute*)
} moreover
{ assume $x \in$ *path-image* (*linepath b c*)
then have $x \in$ *path-image* (*linepath b c*) $\wedge$ $x \in$ *path-image* (*linepath z c*) **using** $*$ **by** *blast*
moreover have $c \in$ *path-image* (*linepath b c*) $\wedge$ $z \in$ *path-image* (*linepath z c*) **by** *simp*
moreover have $x \neq c$ **using** $*$ **by** *blast*
ultimately have *collinear* $\{b, z, c\}$
by (*smt* (*verit*, *best*) *collinear-3-trans in-path-image-imp-collinear insert-commute*)
then have *False* **using** *not-collinear* **by** (*simp add*: *insert-commute*)
} moreover
{ assume $x \in$ *path-image* (*linepath c a*)
then have $x \in$ *path-image* (*linepath c a*) $\wedge$ $x \in$ *path-image* (*linepath z c*) **using** $*$ **by** *blast*
moreover have $c \in$ *path-image* (*linepath c a*) $\wedge$ $z \in$ *path-image* (*linepath z c*) **by** *simp*
moreover have $x \neq c$ **using** $*$ **by** *blast*
ultimately have *collinear* $\{a, z, c\}$
by (*smt* (*verit*, *best*) *collinear-3-trans in-path-image-imp-collinear insert-commute*)
then have *False* **using** *not-collinear* **by** (*simp add*: *insert-commute*)
}
ultimately have *False* **by** *blast*
}
ultimately show *?thesis* **by** *blast*
qed
moreover have *path-image ?l* $\cap$ *path-inside ?q* $\neq \{\}$
proof(*rule ccontr*)
let *?p'* = *make-triangle a b z*

assume ¬ *path-image ?l* ∩ *path-inside ?q* ≠ {}
then have *path-image ?l* ∩ *path-inside ?q* = {} **by** *blast*
then have ∗: *rel-interior* (*path-image ?l*) ∩ *path-inside ?q* = {}
by (*meson disjoint-iff rel-interior-subset subset-eq*)

have *path-image ?l* ⊆ *path-image p* ∪ *path-inside p*
by (*metis UnCI assms*(*1*) *assms*(*3*) *empty-subsetI hull-minimal insert-subset list.simps*(*15*) *local.convex make-triangle-def path-image-linepath segment-convex-hull sup-commute vertices-on-path-image*)
then have *path-image ?l* ⊆ *convex hull* {*a*, *b*, *c*}
by (*smt* (*verit*, *best*) *assms*(*1*) *convex-polygon-is-convex-hull cutpath empty-set insertCI insert-absorb insert-commute is-polygon-cut-path-def list.simps*(*15*) *local.convex make-triangle-def sup-commute*)
then have *rel-interior* (*path-image ?l*) ⊆ *interior* (*convex hull* {*a*, *b*, *c*})
by (*smt* (*verit*, *ccfv-threshold*) *Diff-disjoint IntE IntI Un-upper1 assms*(*1*) *assms*(*2*) *assms*(*3*) *calculation*(*4*) *closure-Un-frontier convex-polygon-is-convex-hull convex-segment*(*1*) *dual-order.trans empty-iff empty-set insertCI insert-absorb2 insert-commute interior list.simps*(*15*) *local.convex make-triangle-def path-image-linepath rel-frontier-def rel-interior-nonempty-interior subsetD subset-rel-interior-convex*)
then have *rel-interior*: *rel-interior* (*path-image ?l*) ⊆ *path-inside p*
by (*smt* (*verit*, *best*) *assms*(*1*) *convex-polygon-is-convex-hull cutpath empty-set insertCI insert-absorb insert-commute interior is-polygon-cut-path-def list.simps*(*15*) *local.convex make-triangle-def*)

have (*let vts1* = []; *vts2* = []; *vts3* = [*c*]; *x* = *a*; *y* = *b*; *cutpath* = *?cutpath*; *p* = *make-polygonal-path* ([*a*, *b*, *c*] @ [[*a*, *b*, *c*] ! *0*]); *p1* = *make-polygonal-path* (*x* # *vts2* @ [*y*] @ *rev* [*z*] @ [*x*]); *p2* = *make-polygonal-path* (*vts1* @ ([*x*] @ [*z*] @ [*y*]) @ *vts3* @ [[*a*, *b*, *c*] ! *0*]); *c1* = *make-polygonal-path* (*x* # *vts2* @ [*y*]); *c2* = *make-polygonal-path* (*vts1* @ ([*x*] @ [*z*] @ [*y*]) @ *vts3*)
in is-polygon-cut-path ([*a*, *b*, *c*] @ [[*a*, *b*, *c*] ! *0*]) *?cutpath* ∧
polygon p ∧
polygon p1 ∧
polygon p2 ∧
path-inside p1 ∩ *path-inside p2* = {} ∧
path-inside p1 ∪ *path-inside p2* ∪ (*path-image cutpath* − {*x*, *y*}) = *path-inside p* ∧
(*path-image p1* − *path-image cutpath*) ∩ (*path-image p2* − *path-image ?cutpath*) = {} ∧
path-image p = *path-image p1* − *path-image ?cutpath* ∪ (*path-image p2* − *path-image ?cutpath*) ∪ {*x*, *y*})
using *1* **unfolding** *is-polygon-split-path-def* **by** *fastforce*
then have (*let*
p = *make-polygonal-path* ([*a*, *b*, *c*] @ [[*a*, *b*, *c*] ! *0*]);
p1 = *make-polygonal-path* (*a* # [] @ [*b*] @ *rev* [*z*] @ [*a*]);
p2 = *make-polygonal-path* ([] @ ([*a*] @ [*z*] @ [*b*]) @ [*c*] @ [[*a*, *b*, *c*] ! *0*])
in path-inside p1 ∪ *path-inside p2* ∪ (*path-image ?cutpath* − {*a*, *b*}) =

path-inside p
∧ (*path-image p1* − *path-image ?cutpath*) ∩ (*path-image p2* − *path-image ?cutpath*) = {})
by *meson*
moreover have *?q* = *make-polygonal-path* ([] @ ([*a*] @ [*z*] @ [*b*]) @ [*c*] @ [[*a*, *b*, *c*] ! *0*])
by *simp*
moreover have *?p′* = *make-polygonal-path* (*a* # [] @ [*b*] @ *rev* [*z*] @ [*a*])
unfolding *make-triangle-def* **by** *simp*
moreover have *p* = *make-polygonal-path* ([*a*, *b*, *c*] @ [[*a*, *b*, *c*] ! *0*])
unfolding *assms make-triangle-def* **by** *auto*
ultimately have *path-inside-p*: *path-inside ?p′*
∪ *path-inside ?q*
∪ (*path-image ?cutpath* − {*a*, *b*}) = *path-inside p*
∧ (*path-image ?p′* − *path-image ?cutpath*) ∩ (*path-image ?q* − *path-image ?cutpath*) = {}
using *1* **unfolding** *make-triangle-def is-polygon-split-path-def* **by** *metis*
moreover have *a* ∈ *path-image ?cutpath* ∧ *a* ∉ *path-inside ?p′* ∪ *path-inside ?q*
by (*metis* (*no-types*, *lifting*) *UnI1* ‹*a* = *pathstart* (*make-polygonal-path* [*a*, *z*, *b*]) ∧ *b* = *pathfinish* (*make-polygonal-path* [*a*, *z*, *b*])› *assms*(*1*) *assms*(*2*) *collinear-2 insert-absorb2 insert-commute path-inside-p pathstart-in-path-image triangle-interior-point-not-collinear-vertices-wlog-helper*)
moreover have *b* ∈ *path-image ?cutpath* ∧ *b* ∉ *path-inside ?p′* ∪ *path-inside ?q*
by (*metis UnI1* ‹*a* = *pathstart* (*make-polygonal-path* [*a*, *z*, *b*]) ∧ *b* = *pathfinish* (*make-polygonal-path* [*a*, *z*, *b*])› *assms*(*1*) *assms*(*2*) *collinear-2 insert-absorb2 path-inside-p pathfinish-in-path-image triangle-interior-point-not-collinear-vertices-wlog-helper*)
ultimately have *rel-interior* (*path-image ?l*) ⊆
(*path-inside ?p′* − *path-image ?cutpath*)
∪ (*path-image ?cutpath* − {*a*, *b*})
using *rel-interior* ∗ **by** *blast*
then have *rel-interior* (*path-image ?l*) ⊆ *path-inside ?p′* ∪ *path-image ?cutpath* **by** *blast*
moreover have *path-image ?cutpath* ⊆ *path-image ?p′*
proof−
have *path-image ?cutpath* = *path-image* (*linepath a z*) ∪ *path-image* (*linepath z b*)
by (*metis list.discI make-polygonal-path.simps*(*3*) *nth-Cons-0 path-image-cons-union*)
moreover have *path-image* (*linepath a z*) = *path-image* (*linepath z a*)
∧ *path-image* (*linepath z b*) = *path-image* (*linepath b z*)
by (*simp add*: *insert-commute*)
moreover have *path-image* (*linepath z a*) ⊆ *path-image ?p′*
∧ *path-image* (*linepath b z*) ⊆ *path-image ?p′*
unfolding *make-triangle-def*
by (*metis Un-commute Un-upper2 list.discI nth-Cons-0 path-image-cons-union sup.coboundedI2*)
ultimately show *?thesis* **by** *blast*
qed

ultimately have *rel-interior* (*path-image ?l*) ⊆ *path-inside ?p′* ∪ *path-image ?p′* **by** *fast*
then have *rel-interior* (*path-image ?l*) ⊆ *convex hull* {*a*, *z*, *b*}
unfolding *make-triangle-def*
by (*simp add*: *insert-commute make-triangle-def not-collinear sup-commute triangle-convex-hull*)
then have *closure* (*rel-interior* (*path-image ?l*)) ⊆ *closure* (*convex hull* {*a*, *z*, *b*})
using *closure-mono* **by** *blast*
then have *path-image ?l* ⊆ *convex hull* {*a*, *z*, *b*} **by** (*simp add*: *convex-closure-rel-interior*)
then have *c*: *c* ∈ *path-image ?p′* ∪ *path-inside ?p′*
unfolding *make-triangle-def*
by (*metis* (*no-types*, *lifting*) *IntE insertCI insert-commute l-q-int make-triangle-def not-collinear subsetD triangle-convex-hull*)

moreover have *c* ∉ *path-image ?p′*
proof −
have *c* ∈ *path-image ?q* − *path-image ?cutpath* **using** *c-noton-cutpath l-q-int* **by** *auto*
moreover have (*path-image ?p′* − *path-image ?cutpath*) ∩ (*path-image ?q* − *path-image ?cutpath*) = {}
using *path-inside-p* **by** *fastforce*
ultimately show *?thesis* **by** *blast*
qed
moreover have *c* ∉ *path-inside ?p′*
by (*smt* (*verit*, *ccfv-threshold*) *DiffI IntD1 UnI1 UnI2* ‹*path-image* (*make-polygonal-path* [*a*, *z*, *b*]) ∩ *path-image p* = {*a*, *b*}› ‹*path-image* (*make-polygonal-path* [*a*, *z*, *b*]) ⊆ *path-image* (*make-triangle a b z*)› *assms*(*1*) *assms*(*2*) *calculation*(*2*) *collinear-2 in-mono insert-absorb2 path-inside-p triangle-interior-point-not-collinear-vertices*)
ultimately show *False* **by** *blast*
qed
ultimately have *cutpath*: *is-polygon-cut ?vts z c*
using *assms* **unfolding** *make-triangle-def is-polygon-cut-def* **by** *blast*
thus *2*: *is-polygon-split* [*a*, *z*, *b*, *c*] *1 3*
using *polygon-cut-to-split*
by (*metis One-nat-def append-Cons append-Nil diff-Suc-1 length-Cons length-greater-0-conv lessI list.discI list.size*(*3*) *nth-Cons-0 nth-Cons-Suc numeral-3-eq-3 polygon-cut-to-split zero-less-diff*)

let *?p1* = *make-triangle a z c*
let *?p2* = *make-triangle z b c*
let *?p3* = *make-triangle a b z*

have (*path-image ?p1* − *path-image* (*linepath z c*)) ∩ (*path-image ?p2* − *path-image* (*linepath z c*)) = {}
using *2* **unfolding** *make-triangle-def is-polygon-split-def*
by (*smt* (*z3*) *Int-commute One-nat-def Suc-1 append-Cons append-Nil diff-numeral-Suc diff-zero drop0 drop-Suc-Cons nth-Cons-0 nth-Cons-Suc nth-Cons-numeral pred-numeral-simps*(*3*) *take0 take-Cons-numeral take-Suc-Cons*)

moreover have *a ∉ path-image (linepath z c) ∧ b ∉ path-image (linepath z c)*
by (*metis* (*no-types*, *lifting*) *assms*(*1*) *assms*(*2*) *assms*(*3*) *in-path-image-imp-collinear insert-commute triangle-interior-point-not-collinear-vertices*)
moreover have *a ∈ path-image ?p1 ∧ b ∈ path-image ?p2*
by (*metis insert-subset list.simps*(*15*) *make-triangle-def vertices-on-path-image*)
ultimately have *a ∉ path-image ?p2 ∧ b ∉ path-image ?p1* **by** *auto*
moreover have *a ∉ path-inside ?p2 ∧ b ∉ path-inside ?p1*
proof−
have *a ∉ path-inside p*
by (*metis* (*no-types*, *lifting*) *assms*(*1*) *assms*(*2*) *collinear-2 insertCI insert-absorb triangle-interior-point-not-collinear-vertices*)
moreover have *b ∉ path-inside p*
using *assms*(*1*) *assms*(*2*) *triangle-interior-point-not-collinear-vertices-wlog-helper* **by** *fastforce*
moreover have *path-inside ?p2 ⊆ path-inside ?q*
using *2* **unfolding** *is-polygon-split-def*
by (*smt* (*verit*) *One-nat-def UnCI append-Cons diff-Suc-1 drop0 drop-Suc-Cons make-triangle-def nth-Cons-0 nth-Cons-Suc numeral-3-eq-3 self-append-conv2 subsetI take0 take-Suc-Cons*)
moreover have *path-inside ?p1 ⊆ path-inside ?q*
using *2* **unfolding** *is-polygon-split-def*
by (*smt* (*verit*) *One-nat-def Un-assoc append-Cons diff-Suc-1 drop0 drop-Suc-Cons inf-sup-absorb le-iff-inf make-triangle-def nth-Cons-0 nth-Cons-Suc numeral-3-eq-3 self-append-conv2 sup-commute take0 take-Suc-Cons*)
moreover have *path-inside ?q ⊆ path-inside p*
using *1* **unfolding** *is-polygon-split-path-def*
by (*smt* (*verit*) *One-nat-def Un-subset-iff Un-upper1 append-Cons append-Nil assms*(*1*) *diff-zero drop0 drop-Suc-Cons make-triangle-def nth-Cons-0 nth-Cons-Suc take0*)
ultimately show *?thesis* **by** *blast*
qed
moreover show *a ∉ path-image ?p2 ∪ path-inside ?p2* **using** *calculation* **by** *simp*
ultimately show *b ∉ path-image ?p1 ∪ path-inside ?p1* **by** *simp*

have (*path-image ?p3 − path-image ?cutpath*) ∩ (*path-image ?q − path-image ?cutpath*) = {}
using *1* **unfolding** *make-triangle-def is-polygon-split-path-def*
by (*smt* (*verit*) *One-nat-def append-Cons append-Nil diff-self-eq-0 diff-zero drop0 drop-Suc-Cons nth-Cons-0 nth-Cons-Suc rev-singleton-conv take-0*)
moreover have *c ∈ path-image ?q* **using** *l-q-int* **by** *auto*
ultimately have *c ∉ path-image ?p3* **using** *c-noton-cutpath* **by** *blast*
moreover have *c ∉ path-inside ?p3*
proof−
have *c ∉ path-inside p*
using *assms*(*1*) *assms*(*2*) *triangle-interior-point-not-collinear-vertices* **by** *fastforce*
moreover have *path-inside ?p3 ⊆ path-inside p*
using *1* **unfolding** *is-polygon-split-path-def*

by (*smt* (*z3*) *One-nat-def Un-assoc Un-upper1 append-Cons append-Nil assms*(*1*) *diff-Suc-Suc diff-zero make-triangle-def nth-Cons-0 nth-Cons-Suc rev-singleton-conv take0*)
ultimately show *?thesis* **by** *blast*
qed
ultimately show $c \notin$ *path-image ?p3* $\cup$ *path-inside ?p3* **by** *blast*
qed

lemma *smaller-triangle*:
assumes $\neg$ *collinear* $\{a, b, c\} \wedge \neg$ *collinear* $\{a', b', c'\}$
assumes $p =$ *make-triangle a b c*
assumes $p' =$ *make-triangle a' b' c'*
assumes *path-inside p* $\subseteq$ *path-inside p'*
assumes $\exists d.$ *integral-vec d* $\wedge$ $d \in$ *path-image p'* $\cup$ *path-inside p'* $\wedge$ $d \notin$ *path-image p* $\cup$ *path-inside p*
shows *card* (*integral-inside p*) + *card* (*integral-boundary p*) < *card* (*integral-inside p'*) + *card* (*integral-boundary p'*)
proof−
have *simple-path p* **using** *assms* **unfolding** *make-triangle-def*
using *assms*(*2*) *polygon-def triangle-is-polygon* **by** *presburger*
then have *finite-p*: *finite* (*integral-inside p*) $\wedge$ *finite* (*integral-boundary p*) **using** *assms* **unfolding** *make-triangle-def*
using *integral-boundary integral-inside finite-integral-points-path-image finite-integral-points-path-inside* **by** *metis*
have *simple-path p'* **using** *assms* **unfolding** *make-triangle-def*
using *assms*(*3*) *polygon-def triangle-is-polygon* **by** *presburger*
then have *finite-p'*: *finite* (*integral-inside p'*) $\wedge$ *finite* (*integral-boundary p'*) **using** *assms* **unfolding** *make-triangle-def*
using *integral-boundary integral-inside finite-integral-points-path-image finite-integral-points-path-inside* **by** *metis*

have *polygon p* **using** *assms*(*1*,*2*) *triangle-is-polygon* **by** *blast*
then have *1*: (*integral-inside p*) $\cap$ (*integral-boundary p*) = {}
unfolding *integral-inside integral-boundary* **using** *inside-outside-polygon* **unfolding** *inside-outside-def* **by** *blast*

have *polygon p'* **using** *assms*(*1*,*3*) *triangle-is-polygon* **by** *blast*
then have *2*: (*integral-inside p'*) $\cap$ (*integral-boundary p'*) = {}
unfolding *integral-inside integral-boundary* **using** *inside-outside-polygon* **unfolding** *inside-outside-def* **by** *blast*

have *path-image-subset*: *path-image p* $\subseteq$ *path-image p'* $\cup$ *path-inside p'*
proof−
have *p-frontier*: *path-image p* = *frontier* (*convex hull* $\{a, b, c\}$)
by (*simp add*: *assms*(*1*) *assms*(*2*) *convex-polygon-frontier-is-path-image2 triangle-convex-hull triangle-is-convex triangle-is-polygon*)
have *p'-frontier*: *path-image p'* = *frontier* (*convex hull* $\{a', b', c'\}$)
by (*simp add*: *assms*(*1*) *assms*(*3*) *convex-polygon-frontier-is-path-image2 triangle-convex-hull triangle-is-convex triangle-is-polygon*)

have *p-interior*: *path-inside p* = *interior* (*convex hull* {*a*, *b*, *c*})
by (*simp add*: *bounded-convex-hull p-frontier inside-frontier-eq-interior path-inside-def*)
have *p'-interior*: *path-inside p'* = *interior* (*convex hull* {*a'*, *b'*, *c'*})
by (*simp add*: *bounded-convex-hull p'-frontier inside-frontier-eq-interior path-inside-def*)

have *interior* (*convex hull* {*a*, *b*, *c*}) ⊆ *interior* (*convex hull* {*a'*, *b'*, *c'*})
using *assms p-interior p'-interior* **by** *argo*
moreover have *compact* (*convex hull* {*a*, *b*, *c*}) ∧ *compact* (*convex hull* {*a'*, *b'*, *c'*})
by (*simp add*: *compact-convex-hull*)
ultimately have *frontier* (*convex hull* {*a*, *b*, *c*})
⊆ *interior* (*convex hull* {*a'*, *b'*, *c'*}) ∪ *frontier* (*convex hull* {*a'*, *b'*, *c'*})
by (*smt* (*verit*, *ccfv-threshold*) *Jordan-inside-outside-real2 closed-path-def* ‹*polygon p'*› ‹*polygon p*› *assms*(*1*) *assms*(*2*) *closure-Un closure-Un-frontier closure-convex-hull finite.emptyI finite-imp-compact finite-insert p'-frontier p'-interior p-interior path-inside-def polygon-def subset-trans sup.absorb-iff1 sup-commute triangle-convex-hull*)
then show *?thesis* **using** *p'-frontier p'-interior p-frontier* **by** *blast*
qed

have *card* ((*integral-inside p*) ∪ (*integral-boundary p*)) = *card* (*integral-inside p*) + *card* (*integral-boundary p*)
using *1 finite-p* **by** (*simp add*: *card-Un-disjoint*)
moreover have *card* ((*integral-inside p'*) ∪ (*integral-boundary p'*)) = *card* (*integral-inside p'*) + *card* (*integral-boundary p'*)
using *2 finite-p'* **by** (*simp add*: *card-Un-disjoint*)
moreover have (*integral-inside p*) ∪ (*integral-boundary p*) ⊆ (*integral-inside p'*) ∪ (*integral-boundary p'*)
using *assms path-image-subset* **unfolding** *integral-inside integral-boundary* **by** *blast*
moreover then have (*integral-inside p*) ∪ (*integral-boundary p*) ⊂ (*integral-inside p'*) ∪ (*integral-boundary p'*) **using** *assms* **unfolding** *integral-inside integral-boundary* **by** *blast*
ultimately show *?thesis* **by** (*metis finite-Un finite-p' psubset-card-mono*)
qed

lemma *pick-elem-triangle*:
fixes *p* :: *R-to-R2*
assumes *p-triangle*: *p* = *make-triangle a b c*
assumes *elem-triangle*: *elem-triangle a b c*
assumes *I* = *card* {*x*. *integral-vec x* ∧ *x* ∈ *path-inside p*} **and**
B = *card* {*x*. *integral-vec x* ∧ *x* ∈ *path-image p*}
shows *measure lebesgue* (*path-inside p*) = *I* + *B*/*2* − *1*
proof −
have *polygon-p*: *polygon p*
using *p-triangle triangle-is-polygon elem-triangle*
unfolding *elem-triangle-def* **by** *auto*
then have *path-inside p* ∩ *path-image p* = {}

using *inside-outside-polygon*[*of p*] **unfolding** *inside-outside-def*
by *auto*

let *?p* = *polygon* (*make-polygonal-path* [*a*, *b*, *c*, *a*])
have *a-neq-b*:*a* ≠ *b*
using *elem-triangle* **unfolding** *elem-triangle-def*
by *auto*
have *b-neq-c*: *b* ≠ *c*
using *elem-triangle* **unfolding** *elem-triangle-def*
by *auto*
have *a-neq-c*: *c* ≠ *a*
using *elem-triangle* **unfolding** *elem-triangle-def*
using *collinear-3-eq-affine-dependent* **by** *blast*

have *path-image p* ⊆ *convex hull* {*a*, *b*, *c*}
using *triangle-path-image-subset-convex p-triangle* **by** *auto*
then have
{*x*. *integral-vec x* ∧ *x* ∈ *path-image p*} ⊆ {*x*. *integral-vec x* ∧ *x* ∈ *convex hull* {*a*, *b*, *c*}}
by *auto*
also have ... = {*a*, *b*, *c*}
using *elem-triangle* **unfolding** *elem-triangle-def* **by** *auto*
finally have {*x*. *integral-vec x* ∧ *x* ∈ *path-image p*} ⊆ {*a*, *b*, *c*} **.**
moreover have {*x*. *integral-vec x* ∧ *x* ∈ *path-image p*} ⊇ {*a*, *b*, *c*}

by (*smt* (*verit*) *Collect-mono-iff make-triangle-def* ‹{*x*. *integral-vec x* ∧ *x* ∈ *convex hull* {*a*, *b*, *c*}} = {*a*, *b*, *c*}› *empty-set insert-subset list.simps*(*15*) *mem-Collect-eq p-triangle subsetD vertices-on-path-image*)
ultimately have {*x*. *integral-vec x* ∧ *x* ∈ *path-image p*} = {*a*, *b*, *c*} **by** *auto*
then have *card-2*: *B* = *3*
using *a-neq-b b-neq-c a-neq-c assms*(*4*)
by *simp*

have {*x*. *integral-vec x* ∧ *x* ∈ *path-inside p*} = {}
proof−
have *path-inside p* ⊆ *convex hull* {*a*, *b*, *c*}
by (*smt* (*verit*, *best*) *Diff-insert-absorb make-triangle-def convex-polygon-inside-is-convex-hull-interior empty-iff empty-set insert-Diff-single insert-commute interior-subset list.simps*(*15*) *p-triangle polygon-p elem-triangle elem-triangle-def triangle-is-convex*)
then have
{*x*. *integral-vec x* ∧ *x* ∈ *path-inside p*} ⊆ {*x*. *integral-vec x* ∧ *x* ∈ *convex hull* {*a*, *b*, *c*}}
by *auto*
also have ... = {*a*, *b*, *c*}
using ‹{*x*. *integral-vec x* ∧ *x* ∈ *convex hull* {*a*, *b*, *c*}} = {*a*, *b*, *c*}› **by** *auto*
finally have {*x*. *integral-vec x* ∧ *x* ∈ *path-inside p*} ⊆ {*a*, *b*, *c*} **.**
moreover have
{*x*. *integral-vec x* ∧ *x* ∈ *path-inside p*} ∩ {*x*. *integral-vec x* ∧ *x* ∈ *path-image p*} = {}

```
    using ‹path-inside p ∩ path-image p = {}› by auto
   ultimately show ?thesis
    using ‹{x. integral-vec x ∧ x ∈ path-image p} = {a, b, c}› by auto
 qed
 then have card-1: I = 0
  using assms(3)
  by (metis card.empty)

 have I + B/2 − 1 = 1/2
  using card-1 card-2 assms
  by auto
 then show ?thesis
  using elem-triangle-area-is-half[OF assms(2)] triangle-measure-convex-hull-measure-path-inside-same[OF
assms(1) assms(2)]
  by auto
qed

lemma pick-triangle-lemma:
 fixes p :: R-to-R2
 assumes p = make-triangle a b c and all-integral [a, b, c] and distinct [a, b, c]
and ¬ collinear {a, b, c}
        I = card {x. integral-vec x ∧ x ∈ path-inside p} and
        B = card {x. integral-vec x ∧ x ∈ path-image p}
 shows measure lebesgue (path-inside p) = I + B/2 − 1
 using assms
proof(induction card {x. integral-vec x ∧ x ∈ path-inside p} + card {x. integral-vec
x ∧ x ∈ path-image p} arbitrary: p a b c I B rule:less-induct)
 case less
 have polygon-p: polygon p using triangle-is-polygon[OF less.prems(4)] less.prems(1)
by simp
 then have polygon-of: polygon-of p [a, b, c, a]
  unfolding polygon-of-def using less.prems(1) unfolding make-triangle-def by
auto

 have convex-hull-char: convex hull {a, b, c} = path-inside p ∪ path-image p
  using triangle-convex-hull[OF less.prems(1) less.prems(4)] by auto
 then have interior-convex-hull: {x. integral-vec x ∧ x ∈ path-inside p} ∪ {x.
integral-vec x ∧ x ∈ path-image p} = {x ∈ convex hull {a, b, c}. integral-vec x}
  by auto
 have vts-in-path-image: a ∈ path-image p ∧ b ∈ path-image p ∧  c ∈ path-image
p
  using assms(1) unfolding make-triangle-def using vertices-on-path-image
 by (metis (mono-tags, lifting) insertCI less.prems(1) list.simps(15) make-triangle-def
subset-code(1))
 have integral-vts: integral-vec a ∧ integral-vec b ∧ integral-vec c
  using less.prems(2)
  by (simp add: all-integral-def)
 then have subset: {a, b, c} ⊆ {x. integral-vec x ∧ x ∈ path-image p}
  using vts-in-path-image integral-vts by simp
```

have *finite-integral-on-path-im*: *finite* $\{x.\ integral\text{-}vec\ x \wedge x \in path\text{-}image\ p\}$
using *finite-integral-points-path-image triangle-is-polygon*[*OF less.prems*(*4*)]
unfolding *make-triangle-def polygon-def*
using *less.prems*(*1*) *make-triangle-def* **by** *auto*
have *B-3-if*: $B > 3$ **if** *other-point-in-set*: $\{x.\ integral\text{-}vec\ x \wedge x \in path\text{-}image\ p\} \neq \{a,\ b,\ c\}$
proof −
have $\exists d.\ d \notin \{a,\ b,\ c\} \wedge d \in \{x.\ integral\text{-}vec\ x \wedge x \in path\text{-}image\ p\}$
using *other-point-in-set subset*
by *blast*
then obtain *d* **where** *d-prop*: $d \notin \{a,\ b,\ c\} \wedge d \in \{x.\ integral\text{-}vec\ x \wedge x \in path\text{-}image\ p\}$
by *auto*
then have *subset2*: $\{a,\ b,\ c,\ d\} \subseteq \{x.\ integral\text{-}vec\ x \wedge x \in path\text{-}image\ p\}$
using *d-prop subset* **by** *auto*
have *distinct* $[a,\ b,\ c,\ d]$
using *d-prop*
using *less.prems*(*3*) **by** *auto*
then have *card-is*: *card* $\{a,\ b,\ c,\ d\} = 4$
by *simp*
show *?thesis* **using** *subset2 card-is finite-integral-on-path-im*
by (*metis* (*no-types*, *lifting*) *Suc-le-eq card-mono eval-nat-numeral*(*2*) *less.prems*(*6*) *semiring-norm*(*26*) *semiring-norm*(*27*))
qed
{ assume ∗: $I = 0$
have *finite* $\{x.\ integral\text{-}vec\ x \wedge x \in path\text{-}inside\ p\}$
using *finite-integral-points-path-inside triangle-is-polygon*[*OF less.prems*(*4*)]
unfolding *make-triangle-def*
by (*simp add*: *less.prems*(*1*) *make-triangle-def polygon-def*)
then have *empty-inside*: $\{x.\ integral\text{-}vec\ x \wedge x \in path\text{-}inside\ p\} = \{\}$
using ∗ *less.prems*(*5*) **by** *auto*

{ assume ∗∗: $B = 3$
have $\{x \in convex\ hull\ \{a,\ b,\ c\}.\ integral\text{-}vec\ x\} = \{a,\ b,\ c\}$
using ∗ ∗∗ *less.prems*(*5*−*6*) *B-3-if interior-convex-hull empty-inside*
by *blast*
then have *elem-triangle a b c*
unfolding *elem-triangle-def* **using** *less.prems*(*4*) *integral-vts* **by** *simp*
then have *measure lebesgue* (*path-inside p*) $= I + B/2 - 1$
using *pick-elem-triangle less.prems* **by** *auto*
}
moreover
{ assume ∗: $B > 3$
then obtain *d* **where** *d*: *integral-vec d* $\wedge$ $d \in path\text{-}image\ p \wedge d \notin \{a,\ b,\ c\}$
by (*smt* (*verit*, *del-insts*) *subset finite-integral-on-path-im less.prems*(*3*) *card-3-iff collinear-3-eq-affine-dependent less.prems*(*4*) *less.prems*(*6*) *less-not-refl mem-Collect-eq subsetI subset-antisym*)
have *path-image* (*make-polygonal-path* $[a,\ b,\ c,\ a]$) = *path-image* (*linepath a*

```
b) ∪  path-image (linepath b c) ∪  path-image (linepath c a)
      by (metis (no-types, lifting) list.discI make-polygonal-path.simps(3) nth-Cons-0
path-image-cons-union sup-assoc)
      then have d ∈ path-image (linepath a b)
          ∨ d ∈ path-image (linepath b c)
          ∨ d ∈ path-image (linepath c a)
        using d less.prems(1) unfolding make-triangle-def polygon-of-def
        by blast
      then have measure lebesgue (path-inside p) = I + B/2 − 1
        using pick-triangle-helper less.prems less.hyps empty-inside d
        unfolding pick-holds pick-triangle integral-inside integral-boundary
        apply simp by blast
    }
    ultimately have measure lebesgue (path-inside p) = I + B/2 − 1
      using B-3-if
       by (metis (no-types, lifting) card.empty card-insert-disjoint collinear-2 fi-
nite.emptyI finite.insertI insert-absorb less.prems(4) less.prems(6) numeral-3-eq-3)
  }
  moreover
  { assume ∗: I > 0
    then obtain d where d-inside: integral-vec d ∧ d ∈ path-inside p
      using less.prems(5)
     by (metis (mono-tags, lifting) Collect-empty-eq add-0 canonically-ordered-monoid-add-class.lessE
card-0-eq card-ge-0-finite)
    have a ∈ path-image p
     using vertices-on-path-image polygon-of unfolding polygon-of-def by fastforce
    then have a-inset: a ∈ path-inside p ∪ path-image p
      by fastforce
    have convex-hull-set: convex hull set [a, b, c, a] = path-inside p ∪ path-image
p
      using convex-hull-char
      by (simp add: insert-commute)
     then have ad-linepath-inside: path-image (linepath a d) ⊆ path-inside p ∪
path-image p
        using d-inside convex-polygon-means-linepaths-inside[OF polygon-of con-
vex-hull-set a-inset]
      by blast
    have b ∈ path-image p
     using vertices-on-path-image polygon-of unfolding polygon-of-def by fastforce
    then have b-inset: b ∈ path-inside p ∪ path-image p
      by fastforce
   have bd-linepath-inside: path-image (linepath b d) ⊆ path-inside p ∪ path-image
p
        using d-inside convex-polygon-means-linepaths-inside[OF polygon-of con-
vex-hull-set b-inset]
      by blast
    have c ∈ path-image p
     using vertices-on-path-image polygon-of unfolding polygon-of-def by fastforce
    then have c-inset: c ∈ path-inside p ∪ path-image p
```

by *fastforce*
then have *cd-linepath-inside*: *path-image* (*linepath c d*) ⊆ *path-inside p* ∪ *path-image p*
using *d-inside convex-hull-char convex-polygon-means-linepaths-inside*[*OF polygon-of convex-hull-set c-inset*]
by *blast*

let *?p1* = *make-triangle a d c*
let *?p2* = *make-triangle d b c*
let *?p3* = *make-triangle a b d*

have *triangle-split*:
is-polygon-split-path [*a*, *b*, *c*] *0 1* [*d*]
is-polygon-split [*a*, *d*, *b*, *c*] *1 3*
a ∉ *path-image ?p2* ∪ *path-inside ?p2*
b ∉ *path-image ?p1* ∪ *path-inside ?p1*
c ∉ *path-image ?p3* ∪ *path-inside ?p3*
using *triangle-3-split*[*of p a b c d*] *less.prems d-inside polygon-p* **apply** *fastforce*
using *triangle-3-split*[*of p a b c d*] *less.prems d-inside polygon-p* **apply** *fastforce*
using *triangle-3-split*[*of p a b c d*] *less.prems d-inside polygon-p* **apply** *fastforce*
using *triangle-3-split*[*of p a b c d*] *less.prems d-inside polygon-p* **apply** *fastforce*
using *triangle-3-split*[*of p a b c d*] *less.prems d-inside polygon-p* **by** *fastforce*

let *?q* = *make-polygonal-path* [*a*, *d*, *b*, *c*, *a*]
let *?I1* = *card* (*integral-inside ?p1*)
let *?B1* = *card* (*integral-boundary ?p1*)
let *?I2* = *card* (*integral-inside ?p2*)
let *?B2* = *card* (*integral-boundary ?p2*)
let *?I3* = *card* (*integral-inside ?p3*)
let *?B3* = *card* (*integral-boundary ?p3*)
let *?Iq* = *card* (*integral-inside ?q*)
let *?Bq* = *card* (*integral-boundary ?q*)
have *measure lebesgue* (*path-inside ?p1*) = *?I1* + *?B1*/*2* − *1*
proof−
have *path-inside ?p1* ⊆ *path-inside ?q*
using *triangle-split*(*2*) **unfolding** *is-polygon-split-def*
by (*smt* (*verit*) *One-nat-def Un-assoc Un-upper1 append-Cons append-Nil diff-Suc-Suc diff-zero drop0 drop-Suc-Cons make-triangle-def nth-Cons-0 nth-Cons-Suc numeral-3-eq-3 sup-commute take0 take-Suc-Cons*)
moreover have *path-inside ?q* ⊆ *path-inside p*
using *triangle-split*(*1*) **unfolding** *is-polygon-split-path-def*
by (*smt* (*verit*) *One-nat-def Un-assoc Un-subset-iff append-Cons append-Nil diff-zero drop0 drop-Suc-Cons less.prems*(*1*) *make-triangle-def nth-Cons-0 nth-Cons-Suc sup.cobounded2 take0*)
ultimately have *path-inside ?p1* ⊆ *path-inside p* **by** *blast*
moreover have ¬ *collinear* {*a*, *d*, *c*}
by (*metis d-inside insert-commute less.prems*(*1*) *polygon-p triangle-interior-point-not-collinear-vertices*)
moreover have ¬ *collinear* {*a*, *b*, *c*} **by** (*simp add*: *less.prems*(*4*))
moreover have *integral-vec b*

using *integral-vts* **by** *blast*
moreover have $b \in$ *path-image p*
using *vts-in-path-image* **by** *auto*
ultimately have *card* (*integral-inside ?p1*) + *card* (*integral-boundary ?p1*)
< *card* (*integral-inside p*) + *card* (*integral-boundary p*)
using *smaller-triangle*[*of a d c a b c ?p1 p*] *triangle-split*(*4*) *less.prems*(*1*)
less-imp-le-nat
by *blast*
thus *?thesis*
using *less.hyps*[*of ?p1 a d c*] **unfolding** *integral-inside integral-boundary*
using ‹¬ *collinear* {*a*, *d*, *c*}› *all-integral-def d-inside integral-vts less.prems*(*1*)
less.prems(*3*) *triangle-split*(*3*) *triangle-split*(*5*)
by *fastforce*
qed
moreover have *measure lebesgue* (*path-inside ?p2*) = *?I2* + *?B2*/*2* − *1*
proof−
have *path-inside ?p2* ⊆ *path-inside ?q*
using *triangle-split*(*2*) **unfolding** *is-polygon-split-def*
by (*smt* (*verit*) *One-nat-def Un-assoc Un-upper1 append-Cons append-Nil*
diff-Suc-Suc diff-zero drop0 drop-Suc-Cons make-triangle-def nth-Cons-0 nth-Cons-Suc
numeral-3-eq-3 sup-commute take0 take-Suc-Cons)
moreover have *path-inside ?q* ⊆ *path-inside p*
using *triangle-split*(*1*) **unfolding** *is-polygon-split-path-def*
by (*smt* (*verit*) *One-nat-def Un-assoc Un-subset-iff append-Cons append-Nil*
diff-zero drop0 drop-Suc-Cons less.prems(*1*) *make-triangle-def nth-Cons-0 nth-Cons-Suc*
sup.cobounded2 take0)
ultimately have *path-inside ?p2* ⊆ *path-inside p* **by** *blast*
moreover have ¬ *collinear* {*d*, *b*, *c*}
by (*metis d-inside insert-commute less.prems*(*1*) *polygon-p triangle-interior-point-not-collinear-vertices*)
moreover have ¬ *collinear* {*a*, *b*, *c*} **by** (*simp add*: *less.prems*(*4*))
moreover have *integral-vec a*
using *integral-vts* **by** *blast*
moreover have $a \in$ *path-image p*
using *vts-in-path-image* **by** *auto*
ultimately have *card* (*integral-inside ?p2*) + *card* (*integral-boundary ?p2*)
< *card* (*integral-inside p*) + *card* (*integral-boundary p*)
using *smaller-triangle*[*of d b c a b c ?p2 p*] *triangle-split*(*3*) *less.prems*(*1*)
less-imp-le-nat
by *blast*
thus *?thesis*
using *less.hyps*[*of ?p2 d b c*] **unfolding** *integral-inside integral-boundary*
using ‹¬ *collinear* {*d*, *b*, *c*}› *all-integral-def d-inside integral-vts less.prems*(*1*)
less.prems(*3*) *triangle-split*(*3*) *triangle-split*(*5*)
by *fastforce*
qed
moreover have *measure lebesgue* (*path-inside ?p3*) = *?I3* + *?B3*/*2* − *1*
proof−
have *path-inside ?p3* ⊆ *path-inside p*
using *triangle-split*(*1*) **unfolding** *is-polygon-split-path-def*

by (*smt* (*verit*) *One-nat-def Un-assoc Un-upper1 append-Cons append-Nil diff-Suc-Suc diff-zero less.prems*(*1*) *make-triangle-def nth-Cons-0 nth-Cons-Suc rev-singleton-conv take0*)
moreover have ¬ *collinear* {*a*, *b*, *d*}
by (*metis d-inside less.prems*(*1*) *polygon-p triangle-interior-point-not-collinear-vertices*)
moreover have ¬ *collinear* {*a*, *b*, *c*} **by** (*simp add*: *less.prems*(*4*))
moreover have *integral-vec c*
using *integral-vts* **by** *blast*
moreover have *c* ∈ *path-image p*
using *vts-in-path-image* **by** *auto*
ultimately have *card* (*integral-inside ?p3*) + *card* (*integral-boundary ?p3*) < *card* (*integral-inside p*) + *card* (*integral-boundary p*)
using *smaller-triangle*[*of a b d a b c ?p3 p*] *triangle-split*(*5*) *less.prems*(*1*) *less-imp-le-nat*
by *blast*
thus *?thesis*
using *less.hyps*[*of ?p3 a b d*] **unfolding** *integral-inside integral-boundary*
using ‹¬ *collinear* {*a*, *b*, *d*}› *all-integral-def d-inside integral-vts less.prems*(*1*) *less.prems*(*3*) *triangle-split*(*3*) *triangle-split*(*5*)
by *fastforce*
qed
moreover have *measure lebesgue* (*path-inside ?q*) = *?Iq* + *?Bq*/*2* − *1*
using *pick-split-union*[*OF triangle-split*(*2*),
of [*a*] [*b*] [] *d c ?q ?p2 ?p1 ?I2 ?B2 ?I1 ?B1 ?Iq ?Bq*]
using *calculation*
unfolding *integral-inside integral-boundary make-triangle-def*
using *all-integral-def d-inside less.prems*(*2*) **by** *force*
ultimately have *?case*
using *pick-split-path-union*[*OF triangle-split*(*1*),
of [] [] [*c*] *a b make-polygonal-path* (*a* # [*d*] @ [*b*]) *p ?p3 ?q ?I3 ?B3 ?Iq ?Bq I B*]
unfolding *integral-inside integral-boundary make-triangle-def less.prems*
using *less.prems*(*2*) **by** *force*
}
ultimately show *?case* **by** *blast*
qed

29.2 Pocket properties

definition *index-not-in-set* :: (*real*^*2*) *list* ⇒ (*real*^*2*) *set* ⇒ *nat* ⇒ *bool*
where *index-not-in-set vts A i* ⟷ *i* ∈ {*i*. *i* < *length vts* ∧ *vts* ! *i* ∉ *A*}

definition *min-index-not-in-set*:: (*real*^*2*) *list* ⇒ (*real*^*2*) *set* ⇒ *nat*
where *min-index-not-in-set vts A* = (*LEAST i*. *index-not-in-set vts A i*)

definition *nonzero-index-in-set* :: (*real*^*2*) *list* ⇒ (*real*^*2*) *set* ⇒ *nat* ⇒ *bool*
where
nonzero-index-in-set vts A i ⟷ *i* ∈ {*i*. *0* < *i* ∧ *i* < *length vts* ∧ *vts* ! *i* ∈ *A*}

definition *min-nonzero-index-in-set* :: (*real^2*) *list* ⇒ (*real^2*) *set* ⇒ *nat* **where**
min-nonzero-index-in-set vts A = (*LEAST i. nonzero-index-in-set vts A i*)

definition *construct-pocket-0* :: (*real^2*) *list* ⇒ (*real^2*) *set* ⇒ (*real^2*) *list* **where**
construct-pocket-0 vts A = *take* ((*min-nonzero-index-in-set vts A*) + *1*) *vts*

definition *is-pocket-0* :: (*real^2*) *list* ⇒ (*real^2*) *list* ⇒ *bool* **where**
is-pocket-0 vts vts′ ⟷
polygon (*make-polygonal-path vts*)
∧ (∃ *i*. *vts′* = *take i vts*)
∧ *3* ≤ *length vts′* ∧ *length vts′* < *length vts*
∧ *hd vts′* ∈ *frontier* (*convex hull* (*set vts*)) ∧ *last vts′* ∈ *frontier* (*convex hull* (*set vts*))
∧ *set* (*tl* (*butlast vts′*)) ⊆ *interior* (*convex hull* (*set vts*))

definition *fill-pocket-0* :: (*real^2*) *list* ⇒ *nat* ⇒ (*real^2*) *list* **where**
fill-pocket-0 vts i = (*hd vts*) # (*drop* (*i−1*) *vts*)

lemma *min-nonzero-index-in-set-exists*:
assumes *set* (*tl vts*) ∩ *A* ≠ {}
shows ∃ *i*. *nonzero-index-in-set vts A i*
proof−
obtain *v* **where** *v*: *v* ∈ *A* ∩ *set* (*tl vts*) **using** *assms* **by** *blast*
then obtain *i* **where** (*tl vts*)!*i* = *v* ∧ *i* < *length* (*tl vts*) **by** (*meson IntD2 in-set-conv-nth*)
then obtain *j* **where** *vts*!*j* = *v* ∧ *0* < *j* ∧ *j* < *length vts* **using** *nth-tl* **by** *fastforce*
thus *?thesis* **unfolding** *nonzero-index-in-set-def* **using** *v* **by** *blast*
qed

lemma *min-nonzero-index-in-set-defined*:
assumes *set* (*tl vts*) ∩ *A* ≠ {}
defines *i* ≡ *min-nonzero-index-in-set vts A*
shows *nonzero-index-in-set vts A i* ∧ (∀ *j* < *i*. ¬ *nonzero-index-in-set vts A j*)
proof−
have ∃ *i*. *nonzero-index-in-set vts A i* **using** *assms min-nonzero-index-in-set-exists* **by** *blast*
then have *nonzero-index-in-set vts A i*
using *assms* **unfolding** *min-nonzero-index-in-set-def*
using *LeastI-ex* **by** *blast*
moreover have (∀ *j* < *i*. ¬ *nonzero-index-in-set vts A j*)
by (*metis assms*(*2*) *wellorder-Least-lemma*(*2*) *leD min-nonzero-index-in-set-def*)
ultimately show *?thesis* **by** *blast*
qed

lemma *min-index-not-in-set-exists*:
assumes *set vts* ⊃ *A*
shows ∃ *i*. *index-not-in-set vts A i*

proof−
obtain *v* **where** *v* ∈ *set vts* ∧ *v* ∉ *A* **using** *assms* **by** *blast*
then obtain *i* **where** *i* < *length vts* ∧ *vts* ! *i* ∉ *A* **by** (*metis in-set-conv-nth*)
thus *?thesis* **unfolding** *index-not-in-set-def* **by** *blast*
qed

lemma *min-index-not-in-set-defined*:
assumes *set vts* ⊃ *A*
defines *i* ≡ *min-index-not-in-set vts A*
shows *index-not-in-set vts A i* ∧ (∀ *j* < *i*. ¬ *index-not-in-set vts A j*)
proof−
have ∃ *i*. *index-not-in-set vts A i* **using** *assms min-index-not-in-set-exists* **by** *simp*
then have *index-not-in-set vts A i*
using *assms* **unfolding** *min-index-not-in-set-def*
using *LeastI-ex* **by** *blast*
moreover have (∀ *j* < *i*. ¬ *index-not-in-set vts A j*)
by (*metis assms*(*2*) *wellorder-Least-lemma*(*2*) *leD min-index-not-in-set-def*)
ultimately show *?thesis* **by** *blast*
qed

lemma *min-nonzero-index-in-set-bound*:
assumes *set* (*tl vts*) ∩ *A* ≠ {}
shows *min-nonzero-index-in-set vts A* < *length vts*
using *min-nonzero-index-in-set-defined assms* **unfolding** *nonzero-index-in-set-def* **by** *blast*

lemma *construct-pocket-0-subset-vts*:
assumes *set* (*tl vts*) ∩ *A* ≠ {}
shows *set* (*construct-pocket-0 vts A*) ⊆ *set vts*
proof−
let *?i* = *min-nonzero-index-in-set vts A*
have *nonzero-index-in-set vts A ?i* **using** *min-nonzero-index-in-set-defined assms* **by** *presburger*
then have *?i* < *length vts* **unfolding** *nonzero-index-in-set-def* **by** *blast*
thus *?thesis* **unfolding** *construct-pocket-0-def* **by** (*simp add*: *set-take-subset*)
qed

lemma *min-index-not-in-set-0*:
assumes *set vts* ⊃ *A*
assumes *vts*!*0* ∈ *A*
defines *i* ≡ *min-index-not-in-set vts A*
defines *r* ≡ *i* − *1*
shows *vts*!*r* ∈ *A*
proof−
have ∗: *index-not-in-set vts A i* ∧ (∀ *j*<*i*. ¬ *index-not-in-set vts A j*)
using *min-index-not-in-set-defined*[*of A vts*, *OF assms*(*1*)] **unfolding** *i-def* **by** *blast*
moreover then have *r* < *i*

unfolding *r-def i-def min-index-not-in-set-def index-not-in-set-def*
by (*metis* (*no-types*, *lifting*) *assms*(*2*) *bot-nat-0.not-eq-extremum diff-less mem-Collect-eq zero-less-one*)
ultimately have ¬ *index-not-in-set vts A r* **by** *blast*
thus *?thesis*
unfolding *index-not-in-set-def* **using** *assms* ∗ *index-not-in-set-def less-imp-diff-less*
by *force*
qed

lemma *construct-pocket-0-last-in-set*:
assumes *set* (*tl vts*) ∩ *A* ≠ {}
assumes *vts*!*0* ∈ *A*
defines *p* ≡ *construct-pocket-0 vts A*
shows *last p* ∈ *A*
proof−
let *?i* = *min-nonzero-index-in-set vts A*
have ∗: *nonzero-index-in-set vts A ?i* **using** *assms*(*1*) *min-nonzero-index-in-set-defined*
by *blast*
then have *length p* = *min-nonzero-index-in-set vts A* + *1*
unfolding *p-def construct-pocket-0-def nonzero-index-in-set-def* **by** *simp*
then have *last p* = *p*!*?i*
by (*metis add-diff-cancel-right' last-conv-nth length-0-conv zero-eq-add-iff-both-eq-0 zero-neq-one*)
also have ... = *vts*!*?i*
unfolding *p-def construct-pocket-0-def* **by** *simp*
also have ... ∈ *A* **using** ∗ **unfolding** *nonzero-index-in-set-def* **by** *force*
finally show *?thesis* **.**
qed

lemma *construct-pocket-0-first-last-distinct*:
assumes *card A* ≥ *2*
assumes *A* ⊆ *set vts*
assumes *distinct* (*butlast vts*)
assumes *hd vts* = *last vts*
shows *hd* (*construct-pocket-0 vts A*) ≠ *last* (*construct-pocket-0 vts A*)
proof−
let *?n* = *min-nonzero-index-in-set vts A*
have *set* (*tl vts*) ∩ *A* ≠ {}
by (*metis* (*no-types*, *lifting*) *Diff-cancel Int-commute Int-insert-right-if1 Nat.le-diff-conv2 Suc-1 add-leD1 assms*(*1*) *assms*(*2*) *card.empty card-Diff-singleton inf.orderE list.collapse list.sel*(*2*) *list.set*(*2*) *not-one-le-zero plus-1-eq-Suc subset-insert*)
then have *n-defined*: *nonzero-index-in-set vts A ?n* ∧ (∀ *j* < *?n*. ¬ *nonzero-index-in-set vts A j*)
using *min-nonzero-index-in-set-defined* **by** *presburger*
obtain *a b* **where** *ab*: *a* ≠ *b* ∧ {*a*, *b*} ⊆ *A* **by** (*metis assms*(*1*) *card-2-iff ex-card*)
then obtain *i j* **where** *ij*: *vts*!*i* = *a* ∧ *vts*!*j* = *b* ∧ *i* < *length vts* ∧ *j* < *length vts* ∧ *i* ≠ *j*
by (*metis* (*no-types*, *opaque-lifting*) *assms*(*2*) *in-set-conv-nth insert-subset subsetD*)

have *?thesis* **if** ∗: *?n* < *length vts* − *1*
proof−
have *?n* > *0* **using** *n-defined* **unfolding** *nonzero-index-in-set-def* **by** *blast*
then have *n-bound′*: *?n* > *0* ∧ *?n* < *length* (*butlast vts*) **using** ∗ **by** *fastforce*
then have *hd vts* ≠ *vts*!*?n*
by (*metis assms*(*3*) *distinct-Ex1 hd-conv-nth ij in-set-conv-nth length-0-conv length-pos-if-in-set less-nat-zero-code nth-butlast*)
moreover then have *vts*!*?n* ≠ *last vts* **using** *assms*(*4*) **by** *simp*
moreover have *last* (*construct-pocket-0 vts A*) = *vts*!*?n*
using *n-defined*
unfolding *construct-pocket-0-def*
by (*metis Cons-nth-drop-Suc Suc-eq-plus1 n-bound′* ∗ *last-snoc less-diff-conv list.sel*(*1*) *nth-butlast take-butlast take-hd-drop*)
moreover have *hd* (*construct-pocket-0 vts A*) = *hd vts*
unfolding *construct-pocket-0-def* **by** *force*
ultimately show *?thesis* **by** *presburger*
qed
moreover have *?thesis* **if** ∗: *?n* = *length vts* − *1*
proof−
have {*i*, *j*} ⊆ {*i*. *i* < *length vts* ∧ *vts* ! *i* ∈ *A*} **using** *ij ab* **by** *simp*
moreover have *i* ≠ *0* ∨ *j* ≠ *0* **using** *ij* **by** *argo*
ultimately have *nonzero-index-in-set vts A i* ∨ *nonzero-index-in-set vts A j*
unfolding *nonzero-index-in-set-def* **by** *simp*
then have *?n* = *i* ∨ *?n* = *j*
by (*metis n-defined Suc-diff-1 gr-implies-not-zero ij linorder-cases not-less-eq* ∗)
moreover then have *last* (*construct-pocket-0 vts A*) = *vts*!*?n*
by (*metis Suc-eq-plus1 construct-pocket-0-def hd-drop-conv-nth ij snoc-eq-iff-butlast take-hd-drop*)
ultimately show *?thesis*
by (*metis* (*no-types*, *lifting*) *ij ab Suc-eq-plus1 assms*(*4*) *bot-nat-0.not-eq-extremum hd-conv-nth insert-subset last-conv-nth less-diff-conv list.size*(*3*) *mem-Collect-eq n-defined nat-neq-iff nonzero-index-in-set-def not-less-eq that*)
qed
ultimately show *?thesis* **using** *n-defined* **unfolding** *nonzero-index-in-set-def* **by** *fastforce*
qed

lemma *construct-pocket-is-pocket*:
assumes *polygon* (*make-polygonal-path vts*)
assumes *vts*!*0* ∈ *frontier* (*convex hull* (*set vts*))
assumes *vts*!*1* ∉ *frontier* (*convex hull* (*set vts*))
shows *is-pocket-0 vts* (*construct-pocket-0 vts* (*set vts* ∩ *frontier* (*convex hull* (*set vts*))))
proof−
let *?vts′* = *construct-pocket-0 vts* (*set vts* ∩ *frontier* (*convex hull* (*set vts*)))
have *ex-i*: ∃ *i*. *?vts′* = *take i vts* **unfolding** *construct-pocket-0-def* **by** *blast*
moreover have *3* ≤ *length ?vts′*

by (*smt* (*verit*) *Cons-nth-drop-Suc IntI Int-iff One-nat-def Suc-1 Suc-diff-Suc Suc-lessI add-diff-cancel-right' add-gr-0 append-Nil2 assms*(1) *assms*(2) *assms*(3) *butlast.simps*(1) *butlast.simps*(2) *butlast-conv-take calculation cancel-comm-monoid-add-class.diff-cancel card.empty construct-pocket-0-def construct-pocket-0-first-last-distinct construct-pocket-0-last-in-set convex-hull-two-vts-on-frontier diff-diff-cancel diff-is-0-eq diff-is-0-eq' drop0 empty-iff empty-set have-wraparound-vertex hd-conv-nth hd-drop-conv-nth hd-take id-take-nth-drop last.simps last-conv-nth last-drop last-in-set last-snoc leI le-add2 le-numeral-extra*(4) *le-trans length-0-conv length-greater-0-conv length-take length-tl length-upt less-2-cases less-numeral-extra*(1) *less-numeral-extra*(3) *linorder-not-less list.distinct*(1) *list.sel*(2) *list.sel*(3) *list.size*(3) *min.absorb4 not-gr-zero not-less-eq-eq not-numeral-le-zero nth-mem numeral-3-eq-3 plus-1-eq-Suc polygon-at-least-3-vertices polygon-at-least-3-vertices-wraparound polygon-def pos2 rev.simps*(1) *self-append-conv2 simple-polygonal-path-vts-distinct snoc-eq-iff-butlast subset-iff take-all-iff take-eq-Nil take-hd-drop*)
moreover have *vts'-length*: *length ?vts' < length vts*
by (*metis* (*no-types*, *lifting*) *One-nat-def Suc-1 assms*(1) *calculation*(1) *calculation*(2) *construct-pocket-0-first-last-distinct convex-hull-two-vts-on-frontier have-wraparound-vertex hd-conv-nth inf-le1 last-snoc leI le-add2 le-trans length-take min.absorb4 not-numeral-le-zero numeral-3-eq-3 plus-1-eq-Suc polygon-at-least-3-vertices polygon-def simple-polygonal-path-vts-distinct take-all-iff take-eq-Nil*)
moreover have *hd ?vts' ∈ frontier* (*convex hull* (*set vts*))
by (*metis assms*(2) *bot-nat-0.not-eq-extremum calculation*(1) *calculation*(2) *hd-conv-nth hd-take list.size*(3) *not-numeral-le-zero take-eq-Nil*)
moreover have *last ?vts' ∈ frontier* (*convex hull* (*set vts*))
by (*smt* (*verit*, *ccfv-SIG*) *Cons-nth-drop-Suc Int-iff assms*(1) *assms*(2) *card-length construct-pocket-0-last-in-set drop0 drop-eq-Nil empty-iff have-wraparound-vertex last-drop last-in-set le-add2 le-trans linorder-not-less list.sel*(3) *list.simps*(15) *not-less-eq-eq numeral-3-eq-3 plus-1-eq-Suc polygon-at-least-3-vertices snoc-eq-iff-butlast*)
moreover have *set* (*tl* (*butlast ?vts'*)) ⊆ *interior* (*convex hull* (*set vts*))
proof−
let *?A* = (*set vts* ∩ *frontier* (*convex hull* (*set vts*)))
let *?r* = *min-nonzero-index-in-set vts ?A*
have *nonzero-index-in-set vts ?A ?r*
∧ (∀ *j*<*min-nonzero-index-in-set vts ?A*. ¬ *nonzero-index-in-set vts ?A j*)
by (*metis min-nonzero-index-in-set-defined IntI Nitpick.size-list-simp*(2) *One-nat-def add-leD1 assms*(1) *assms*(2) *calculation*(2) *calculation*(3) *empty-iff empty-set have-wraparound-vertex last-in-set last-snoc last-tl less-one not-one-le-zero nth-mem numeral-3-eq-3 plus-1-eq-Suc*)
then have ∀ *i*. (0 < *i* ∧ *i* < *?r*) ⟶ *vts*!*i* ∉ *?A* **unfolding** *nonzero-index-in-set-def*
by *force*
then have ∀ *i*. (0 < *i* ∧ *i* < *?r*) ⟶ *vts*!*i* ∉ *frontier* (*convex hull* (*set vts*))
using *calculation*(3) *construct-pocket-0-def* **by** *fastforce*
then have ∀ *i*. (0 < *i* ∧ *i* < *?r*) ⟶ *vts*!*i* ∈ *interior* (*convex hull* (*set vts*))
by (*smt* (*verit*, *ccfv-threshold*) *Cons-nth-drop-Suc DiffI IntI One-nat-def add-leD1 assms*(1) *assms*(2) *calculation*(2) *calculation*(3) *closure-subset drop0 dual-order.strict-trans2 empty-iff frontier-def have-wraparound-vertex hull-subset inf.strict-coboundedI2 inf.strict-order-iff last-drop last-in-set last-snoc length-greater-0-conv list.discI list.sel*(3) *min-nonzero-index-in-set-bound nth-mem numeral-3-eq-3 plus-1-eq-Suc subset-eq*)
moreover have *tl* (*butlast ?vts'*) = *drop 1* (*take ?r vts*)
unfolding *construct-pocket-0-def*
by (*metis One-nat-def add-implies-diff antisym-conv2 butlast-take construct-pocket-0-def*

drop-0 drop-Suc linorder-le-cases take-all vts'-length)
moreover have $\forall v \in set\ (drop\ 1\ (take\ ?r\ vts)).\ \exists i.\ 0 < i \wedge i < ?r \wedge vts!i = v$
proof
fix *v* **assume** ∗: $v \in set\ (drop\ 1\ (take\ ?r\ vts))$
then obtain i' **where** i': $(drop\ 1\ (take\ ?r\ vts))!i' = v \wedge i' < ?r - 1$
by (*smt* (*verit*) *Cons-nth-drop-Suc One-nat-def ex-i butlast-conv-take calculation*(2) *drop0 hd-conv-nth hd-take index-less-size-conv length-drop length-take less-imp-le-nat linorder-not-less list.collapse list.sel*(2) *min.absorb4 nth-index take-all-iff take-eq-Nil vts'-length*)
then have $(take\ ?r\ vts)!(i' + 1) = v$
by (*metis* ∗ *add.commute drop-eq-Nil empty-iff empty-set nle-le nth-drop*)
thus $\exists i.\ 0 < i \wedge i < ?r \wedge vts!i = v$
by (*metis add-gr-0 i' less-diff-conv nth-take zero-less-one*)
qed
ultimately show *?thesis* **by** *fastforce*
qed
ultimately show *?thesis* **unfolding** *is-pocket-0-def* **using** *assms*(1) **by** *argo*
qed

lemma *exists-point-above-interior*:
fixes $a :: real\hat{}2$
assumes $a \in interior\ (convex\ hull\ S)$
obtains x **where** $x \in S \wedge x\$2 > a\$2$
proof−
have *False* **if** $\forall x \in S.\ x\$2 \leq a\$2$
proof−
have $S \subseteq \{x.\ x \cdot (vector\ [0,\ 1]) \leq a\$2\}$
proof(*rule subsetI*)
fix *x*
assume $x \in S$
then have $x\$2 \leq a\$2$ **using** *that* **by** *blast*
moreover have $x \cdot (vector\ [0,\ 1]) = x\$1 * 0 + x\$2 * 1$
by (*simp add*: *cart-eq-inner-axis e1e2-basis*(3))
ultimately show $x \in \{x.\ x \cdot (vector\ [0,\ 1]) \leq a\$2\}$ **by** *simp*
qed
then have ∗: $convex\ hull\ S \subseteq \{x.\ x \cdot (vector\ [0,\ 1]) \leq a\$2\}$
proof−
have $S \subseteq \{v.\ vector\ [0,\ 1] \cdot v \leq a\ \$\ 2\}$
by (*simp add*: ‹$S \subseteq \{x.\ x \cdot vector\ [0,\ 1] \leq a\ \$\ 2\}$› *inner-commute*)
then have $convex\ hull\ S \subseteq \{v.\ vector\ [0,\ 1] \cdot v \leq a\ \$\ 2\}$
by (*simp add*: *convex-halfspace-le hull-minimal*)
then show *?thesis*
by (*simp add*: *inner-commute*)
qed
moreover have $a \cdot (vector\ [0,\ 1]) = a\$2$ **by** (*simp add*: *cart-eq-inner-axis e1e2-basis*(3))
moreover have *frontier* $\{x.\ x \cdot ((vector\ [0,\ 1])::(real\hat{}2)) \leq a\$2\}$

$= \{x.\ x \cdot (vector\ [0,\ 1]) = a\$2\}$
using *frontier-halfspace-le*[*of* (*vector* [0, 1])::(*real*^2) *a*$2]
by (*smt* (*verit*) *Collect-cong inner-commute vector-2*(2) *zero-index*)
ultimately have $a \in$ *frontier* $\{x.\ x \cdot (vector\ [0,\ 1]) \leq a\$2\}$ **by** *blast*
thus *False*
by (*metis* (*mono-tags*, *lifting*) *Diff-iff* ∗ *assms frontier-def in-frontier-in-subset in-mono interior-subset*)
qed
thus *?thesis* **using** *that* **by** *fastforce*
qed

lemma *exists-point-above-convex-hull-interior*:
fixes S :: (*real*^2) *set*
assumes $S \neq \{\}$
assumes *compact S*
obtains x **where** $x \in S - (interior\ (convex\ hull\ S)) \wedge (\forall\, y \in interior\ (convex\ hull\ S).\ x\$2 > y\$2)$
proof−
let *?H* = *convex hull S*
let *?e2* = (*vector* [0, 1])::(*real*^2)
let *?f* = (λ*x*. *x*$2)::(*real*^2 ⇒ *real*)
have *continuous-on* {*x*. *True*} *?f* **by** (*simp add*: *continuous-on-component*)
moreover have *compact* (*convex hull S*) **using** *assms*(2) *compact-convex-hull* **by** *blast*
moreover from *calculation* **have** *compact* (*?f*‘*?H*)
using *compact-continuous-image continuous-on-subset* **by** *blast*
ultimately obtain x *max* **where** x: $x \in$ *?H* ∧ *?f x* = *max* ∧ $(\forall\, y \in$ *?H*. $y\$2 \leq$ *max*)
by (*smt* (*verit*) *Collect-mono assms*(1) *convex-hull-eq-empty convex-hull-explicit continuous-attains-sup continuous-on-subset*)

have *?H* ∩ $\{x.$ *?e2* · $x =$ *max*$\} \neq \{\}$
by (*metis* (*mono-tags*, *lifting*) *cart-eq-inner-axis disjoint-iff e1e2-basis*(3) *inner-commute mem-Collect-eq x*)
moreover have *?H* ∩ $\{x.$ *?e2* · $x =$ *max*$\} = \{\}$ **if** $(\forall\, x \in S.\ x\$2 <$ *max*)
proof−
have $S \subseteq \{x.$ *?e2* · $x <$ *max*$\}$
using *that* **by** (*simp add*: *cart-eq-inner-axis e1e2-basis*(3) *inner-commute subset-eq*)
moreover have *convex* $\{x.$ *?e2* · $x <$ *max*$\}$ **by** (*simp add*: *convex-halfspace-lt*)
ultimately show *?thesis* **using** *hull-minimal* **by** *blast*
qed
ultimately have $\exists\, x \in S.\ x\$2 \geq$ *max* **by** *force*
moreover have *?H* ⊆ $\{x.$ *?e2* · $x \leq$ *max*$\}$
using *x*
by (*simp add*: *cart-eq-inner-axis e1e2-basis*(3) *inner-commute subsetI*)
moreover then have *interior ?H* ⊆ $\{x.$ *?e2* · $x <$ *max*$\}$
by (*metis* (*mono-tags*) *convex-empty empty-iff inner-zero-left interior-halfspace-le interior-mono real-inner-1-left separating-hyperplane-set-0 vector-2*(2) *zero-index*)

ultimately have $x \notin$ *interior ?H* $\wedge$ $(\forall y \in$ *interior ?H*. $x\$2 > y\$2)$
by (*smt* (*verit*) *cart-eq-inner-axis e1e2-basis*(*3*) *in-mono inner-commute mem-Collect-eq x*)
thus *?thesis* **using** *that* ‹$\exists x{\in}S$. $max \leq x \$ 2$› x **by** *fastforce*
qed

lemma *flip-function*:
defines $M \equiv$ (*vector* [*vector* [*1*, *0*], *vector* [*0*, $-$*1*]])::(*real^2^2*)
defines $f \equiv \lambda v.\ M *v v$
defines $g \equiv$ ($\lambda v.$ *vector* [$v\$1$, $-v\$2$])::(*real^2* $\Rightarrow$ *real^2*)
shows *inj f f* = *g*
proof−
have *det M* = *M$1$1* ∗ *M$2$2* − *M$1$2* ∗ *M$2$1* **using** *det-2* **by** *blast*
thus *inj f* **by** (*simp add*: *inj-matrix-vector-mult invertible-det-nz f-def M-def*)

have $\bigwedge x.\ f\ x = g\ x$
proof−
fix *x*
have *f x* = *vector* [*M$1$1* ∗ *x$1* + *M$1$2* ∗ *x$2*, *M$2$1* ∗ *x$1* + *M$2$2* ∗ *x$2*]
by (*simp add*: *M-def f-def mat-vec-mult-2*)
also have ... = *vector* [*x$1*, $-$*x$2*] **by** (*simp add*: *M-def*)
finally show *f x* = *g x* **using** *f-def g-def* **by** *blast*
qed
thus *f* = *g* **by** (*simp add*: *f-def g-def*)
qed

lemma *exists-point-below-convex-hull-interior*:
fixes *S* :: (*real^2*) *set*
assumes $S \neq \{\}$
assumes *compact S*
obtains *x* **where** $x \in S - ($*interior* (*convex hull S*)$) \wedge (\forall y \in$ *interior* (*convex hull S*). $x\$2 < y\$2)$
proof−
let *?M* = (*vector* [*vector* [*1*, *0*], *vector* [*0*, $-$*1*]])::(*real^2^2*)
let *?f* = $\lambda v.$ *?M* ∗*v v*
let *?g* = ($\lambda v.$ *vector* [$v\$1$, $-v\$2$])::(*real^2* $\Rightarrow$ *real^2*)

let *?H′* = *?g*‘(*convex hull S*)
let *?S′* = *?g*‘*S*

have *interior*: *?f*‘(*interior* (*convex hull S*)) = *interior* (*convex hull* (*?f*‘*S*))
by (*smt* (*verit*, *best*) *flip-function convex-hull-linear-image interior-injective-linear-image matrix-vector-mul-linear*)
have *hull*: *?H′* = *convex hull ?S′*
proof−
have (∗*v*) (*vector* [*vector* [*1*, *0*], *vector* [*0*, − *1*]]) ‘ (*convex hull S*) = *convex hull* ((∗*v*) (*vector* [*vector* [*1*, *0*], *vector* [*0*, − *1*]]) ‘ *S*::(*real*, *2*) *vec set*)
by (*simp add*: *convex-hull-linear-image*)

then show *?thesis*
by (*simp add*: *flip-function*)
qed
moreover have *compact ?S'*
proof−
have *continuous-on* {*x*. *True*} *?f* **using** *matrix-vector-mult-linear-continuous-on* **by** *blast*
then have *continuous-on* {*x*. *True*} *?g* **using** *flip-function* **by** *simp*
thus *?thesis* **using** *assms*(2) *compact-continuous-image continuous-on-subset flip-function* **by** *blast*
qed
moreover have *?S'* ≠ {} **using** *assms*(1) **by** *blast*
ultimately obtain *x'* **where** *x'*: *x'* ∈ *?S'* − (*interior ?H'*) ∧ (∀ *y* ∈ *interior ?H'*. *x'*$2 > *y*$2)
using *exists-point-above-convex-hull-interior*[*of ?S'*] **by** *auto*
moreover have *?S'* − (*interior ?H'*) = *?f*‘(*S* − (*interior* (*convex hull S*)))
proof−
have *?f*‘(*S* − (*interior* (*convex hull S*))) = *?S'* − *?f*‘(*interior* (*convex hull S*))
by (*metis* (*no-types*, *lifting*) *flip-function*(1) *flip-function*(2) *image-cong image-set-diff*)
thus *?thesis* **using** *flip-function*(2) *interior hull* **by** *auto*
qed
ultimately obtain *x* **where** *?g x* = *x'* ∧ *x* ∈ *S* − *interior* (*convex hull S*)
using *flip-function* **by** *auto*
moreover have (∀ *y*∈*interior* (*convex hull S*). *x* $ *2* < *y* $ *2*)
proof *clarify*
fix *y*
assume *y* ∈ *interior* (*convex hull S*)
then have (*?g x*)$2 > (*?g y*)$2
using *x' interior hull flip-function* **by** (*metis* (*no-types*, *lifting*) *calculation image-eqI*)
thus *x*$2 < *y*$2 **by** *simp*
qed
ultimately show *?thesis* **using** *that* **by** *fast*
qed

lemma *exists-point-above-all*:
fixes *p q* :: *R-to-R2*
defines *H* ≡ *convex hull* (*path-image p* ∪ *path-image q*)
assumes *path p* ∧ *path q*
assumes *p*‘{*0*<..<*1*} ⊆ *interior H*
assumes (*p 0*)$2 = *0* ∧ (*p 1*)$2 = *0*
assumes ∃ *x* ∈ *p*‘{*0*<..<*1*}. *x*$2 ≥ *0*
obtains *x* **where** *x* ∈ *path-image q* ∧ (∀ *y* ∈ *path-image p*. *x*$2 > *y*$2)
proof−
let *?S* = *path-image p* ∪ *path-image q*
let *?H* = *convex hull ?S*
obtain *x* **where** *x*: *x* ∈ *?S* − (*interior ?H*) ∧ (∀ *y* ∈ *interior ?H*. *x*$2 > *y*$2)
by (*metis exists-point-above-convex-hull-interior Un-empty assms*(2) *compact-Un*

compact-path-image path-image-nonempty)
then have $x \notin p`\{0<..<1\}$ **using** *H-def assms*(3) **by** *blast*
moreover have $x \in ?S$ **using** x **by** *blast*
ultimately have $x \in$ *path-image* $q \vee x \in$ (*path-image* p) $-$ $p`\{0<..<1\}$ **by** *blast*
moreover have $\{0..1\} - \{0<..<1\} = \{0::real, 1\}$ **by** *fastforce*
ultimately have $x \in$ *path-image* $q \vee x \in p`\{0, 1\}$
by (*smt* (*verit*, *best*) *image-diff-subset path-image-def subsetD*)
moreover have $x\$2 > (p\ 0)\$2 \wedge x\$2 > (p\ 1)\$2$
using *H-def assms*(3) *assms*(4) *assms*(5) x **by** *fastforce*
ultimately have $x \in$ *path-image* $q \wedge x\$2 > (p\ 0)\$2 \wedge x\$2 > (p\ 1)\$2 \wedge (\forall y \in p`\{0<..<1\}.\ x\$2 > y\$2)$
using *H-def assms*(3) x **by** *auto*
moreover have *path-image* $p = p`\{0<..<1\} \cup \{p\ 0, p\ 1\}$
proof−
have $\{0<..<1\} \cup \{0::real, 1\} = \{0..1\}$ **by** *force*
thus *?thesis* **unfolding** *path-image-def* **by** *blast*
qed
ultimately show *?thesis* **by** (*simp add*: *that*)
qed

lemma *exists-point-below-all*:
fixes $p\ q ::$ *R-to-R2*
defines $H \equiv$ *convex hull* (*path-image* $p \cup$ *path-image* q)
assumes *path* $p \wedge$ *path* q
assumes $p`\{0<..<1\} \subseteq$ *interior* H
assumes $(p\ 0)\$2 = 0 \wedge (p\ 1)\$2 = 0$
assumes $\exists x \in$ *path-image* $p \cup$ *path-image* $q.\ x\$2 < 0$
obtains x **where** $x \in$ *path-image* $q \wedge (\forall y \in$ *path-image* $p.\ x\$2 < y\$2)$
proof−
let *?thesis'* $= \exists x.\ x \in$ *path-image* $q \wedge (\forall y \in$ *path-image* $p.\ x\$2 < y\$2)$
have *?thesis'* **if** $\exists x \in$ *path-image* $p.\ x\$2 < 0$
proof−
have $*$: $\exists x \in p`\{0<..<1\}.\ x\$2 < 0$
proof−
have $(p\ 0)\$2 = 0 \wedge (p\ 1)\$2 = 0$ **by** (*simp add*: *assms*(4))
thus *?thesis*
using *that* **unfolding** *path-image-def*
using *atLeastAtMost-iff less-eq-real-def*
by *fastforce*
qed
let $?S =$ *path-image* $p \cup$ *path-image* q
let $?H =$ *convex hull* $?S$
obtain x **where** x: $x \in ?S - ($*interior* $?H) \wedge (\forall y \in$ *interior* $?H.\ x\$2 < y\$2)$
by (*metis exists-point-below-convex-hull-interior Un-empty assms*(2) *compact-Un compact-path-image path-image-nonempty*)
then have $x \notin p`\{0<..<1\}$ **using** *H-def assms*(3) **by** *blast*
moreover have $x \in ?S$ **using** x **by** *blast*
ultimately have $x \in$ *path-image* $q \vee x \in$ (*path-image* p) $-$ $p`\{0<..<1\}$ **by** *blast*

moreover have $\{0..1\} - \{0<..<1\} = \{0::real,\ 1\}$ **by** *fastforce*
ultimately have $x \in$ *path-image* $q \vee x \in p`\{0,\ 1\}$
by (*smt* (*verit*, *best*) *image-diff-subset path-image-def subsetD*)
moreover have $x\$2 < (p\ 0)\$2 \wedge x\$2 < (p\ 1)\$2$
by (*smt* (*verit*, *ccfv-SIG*) $*$ *H-def assms*(3) *assms*(4) *subset-eq x*)
ultimately have $x\$2 < (p\ 0)\$2 \wedge x\$2 < (p\ 1)\$2 \wedge (\forall y \in p`\{0<..<1\}.\ x\$2 < y\$2)$
using *H-def assms*(3) *x* **by** *blast*
moreover have *path-image* $p = p`\{0<..<1\} \cup \{p\ 0,\ p\ 1\}$
proof −
have $\{0<..<1\} \cup \{0::real,\ 1\} = \{0..1\}$ **by** *force*
thus *?thesis* **unfolding** *path-image-def* **by** *blast*
qed
ultimately have $\forall y \in$ *path-image p*. $x\$2 < y\$2$ **by** *fast*
thus *?thesis* **using** *x* **by** *fast*
qed
moreover then have *?thesis'* **if** $\neg\ (\exists x \in$ *path-image p*. $x\$2 < 0)$ **using** *assms*(5) **by** *fastforce*
ultimately show *?thesis* **using** *that* **by** *blast*
qed

lemma *pocket-fill-line-int-aux*:
fixes *x y z* :: *real^2*
defines $a \equiv y\$1$
assumes $x = 0$
assumes $a > 0 \wedge y\$2 = 0$
assumes $z\$1 < 0 \vee z\$1 > a$
assumes $z\$2 = 0$
assumes *convex A* $\wedge$ *compact A*
assumes $\{x,\ y,\ z\} \subseteq A$
assumes $\{x,\ y\} \subseteq$ *frontier A*
shows $z \in$ *frontier A* $\wedge$ *closed-segment x y* $\subseteq$ *frontier A*
proof(*rule disjE*[*OF assms*(4)])
assume $z\$1 > a$
moreover have *xyz*: $x\$1 = 0 \wedge x\$2 = 0 \wedge y\$1 = a \wedge y\$2 = 0 \wedge z\$2 = 0$
by (*simp add*: *a-def assms*(2) *assms*(3) *assms*(5))
ultimately have *y*: $y \in$ *path-image* (*linepath x z*) (**is** - $\in$ *?L*)
using *segment-horizontal assms*(3) **by** *force*
moreover have *y-neq*: $y \neq x \wedge y \neq z$
by (*metis a-def assms*(2) *assms*(3) *assms*(4) *not-less-iff-gr-or-eq zero-index*)
ultimately have $y \in$ *rel-interior ?L*
by (*metis UnE closed-segment-eq-open closed-segment-idem insert-Diff insert-iff path-image-linepath rel-interior-closed-segment singleton-insert-inj-eq*)
moreover have $?L \subseteq A$ **using** *assms closed-segment-subset* **by** *auto*
moreover have $z \in$ *interior A* $\cup$ *frontier A*
by (*metis Diff-iff UnI1 UnI2 assms*(6) *calculation*(2) *closure-convex-hull convex-hull-eq frontier-def in-mono pathfinish-in-path-image pathfinish-linepath*)
ultimately have $z \in$ *frontier A*
by (*metis* (*no-types*, *lifting*) *Int-iff UnE y y-neq assms*(6) *assms*(8) *com-*

pact-imp-closed insert-subset singletonD triangle-3-split-helper)
moreover have *closed-segment x y* ⊆ *frontier A*
proof(*rule ccontr*)
assume ¬ *closed-segment x y* ⊆ *frontier A*
then obtain *v* **where** *v* ∈ *closed-segment x y* − *frontier A* **by** *blast*
moreover then have *v* ∈ *closed-segment x y* ∩ *interior A*
by (*metis* (*no-types*, *lifting*) *DiffD1 DiffD2 DiffI Int-iff assms*(*6*) *assms*(*7*) *closed-segment-subset closure-convex-hull convex-hull-eq frontier-def insert-subset subsetD*)
moreover from *calculation* **have** *v* ≠ *x* ∧ *v* ≠ *y* **using** *assms*(*8*) **by** *auto*
moreover from *calculation* **have** *v*$*1* < *a*
by (*smt* (*verit*) *DiffD1 a-def assms*(*2*) *assms*(*3*) *exhaust-2 segment-horizontal vec-eq-iff zero-index*)
moreover from *calculation* **have** *y* ∈ *open-segment v z*
by (*smt* (*verit*,*del-insts*) *Diff-iff xyz insert-iff open-segment-def open-segment-idem path-image-linepath segment-horizontal y y-neq*)
ultimately have *y* ∈ *interior A*
by (*metis* (*no-types*, *lifting*) *IntD2 assms*(*6*) *assms*(*7*) *closure-convex-hull convex-hull-eq in-interior-closure-convex-segment insertI2 singletonI subsetD*)
thus *False* **using** *assms*(*8*) *frontier-def* **by** *auto*
qed
ultimately show *z* ∈ *frontier A* ∧ *closed-segment x y* ⊆ *frontier A* **by** *blast*
next
assume ∗: *z*$*1* < *0*
moreover have *xyz*: *x*$*1* = *0* ∧ *x*$*2* = *0* ∧ *y*$*1* = *a* ∧ *y*$*2* = *0* ∧ *z*$*2* = *0*
by (*simp add*: *a-def assms*(*2*) *assms*(*3*) *assms*(*5*))
ultimately have *x*: *x* ∈ *path-image* (*linepath y z*) (**is** - ∈ *?L′*)
using *segment-horizontal assms*(*3*) **by** *force*
moreover have *x-neq*: *y* ≠ *x* ∧ *x* ≠ *z*
by (*metis a-def assms*(*2*) *assms*(*3*) *assms*(*4*) *not-less-iff-gr-or-eq zero-index*)
ultimately have *x* ∈ *rel-interior ?L′*
by (*metis UnE closed-segment-eq-open closed-segment-idem insert-Diff insert-iff path-image-linepath rel-interior-closed-segment singleton-insert-inj-eq*)
moreover have *?L′* ⊆ *A*
proof−
have *y* ∈ *A* ∧ *z* ∈ *A* **using** *assms* **by** *blast*
thus *?thesis* **by** (*simp add*: *assms*(*6*) *closed-segment-subset*)
qed
moreover have *z* ∈ *interior A* ∪ *frontier A*
by (*metis Diff-iff UnI1 UnI2 assms*(*6*) *calculation*(*2*) *closure-convex-hull convex-hull-eq frontier-def in-mono pathfinish-in-path-image pathfinish-linepath*)
ultimately have *z* ∈ *frontier A*
by (*metis* (*no-types*, *lifting*) *Int-iff UnE x x-neq assms*(*6*) *assms*(*8*) *compact-imp-closed insert-subset singletonD triangle-3-split-helper*)
moreover have *closed-segment x y* ⊆ *frontier A*
proof(*rule ccontr*)
assume ¬ *closed-segment x y* ⊆ *frontier A*
then obtain *v* **where** *v* ∈ *closed-segment x y* − *frontier A* **by** *blast*
moreover then have *v* ∈ *closed-segment x y* ∩ *interior A*

by (*metis* (*no-types*, *lifting*) *DiffD1 DiffD2 DiffI Int-iff assms*(6) *assms*(7) *closed-segment-subset closure-convex-hull convex-hull-eq frontier-def insert-subset subsetD*)
moreover from *calculation* **have** $v \neq x \wedge v \neq y$ **using** *assms*(8) **by** *auto*
moreover from *calculation* **have** *v*$1 > 0
by (*smt* (*verit*) *DiffD1 a-def assms*(2) *assms*(3) *exhaust-2 segment-horizontal vec-eq-iff zero-index*)
moreover from *calculation* **have** $x \in$ *open-segment v z*
by (*smt* (*verit*,*del-insts*) *Diff-iff xyz insert-iff open-segment-def open-segment-idem path-image-linepath segment-horizontal x x-neq*)
ultimately have $x \in$ *interior A*
by (*metis* (*no-types*, *lifting*) *IntD2 assms*(6) *assms*(7) *closure-convex-hull convex-hull-eq in-interior-closure-convex-segment insertI2 singletonI subsetD*)
thus *False* **using** *assms*(8) *frontier-def* **by** *auto*
qed
ultimately show $z \in$ *frontier A* $\wedge$ *closed-segment x y* $\subseteq$ *frontier A* **by** *blast*
qed

lemma *axis-dist*:
fixes *a b* :: *real*^2
shows *a*$2 = *b*$2 $\Longrightarrow$ *dist a b* = *dist* (*a*$1) (*b*$1) *a*$1 = *b*$1 $\Longrightarrow$ *dist a b* = *dist* (*a*$2) (*b*$2)
proof−
have *dist a b* = *norm* (*b* − *a*) **by** (*metis dist-commute dist-norm*)
also have ... = *sqrt* ((*b* − *a*) · (*b* − *a*)) **using** *norm-eq-sqrt-inner* **by** *blast*
also have ... = *sqrt* ((*b* − *a*)$1 ∗ (*b* − *a*)$1 + (*b* − *a*)$2 ∗ (*b* − *a*)$2)
by (*simp add*: *inner-vec-def sum-2*)
finally have ∗: *dist a b* = *sqrt* ((*b* − *a*)$1 ∗ (*b* − *a*)$1 + (*b* − *a*)$2 ∗ (*b* − *a*)$2) **.**
show *a*$2 = *b*$2 $\Longrightarrow$ *dist a b* = *dist* (*a*$1) (*b*$1)
a$1 = *b*$1 $\Longrightarrow$ *dist a b* = *dist* (*a*$2) (*b*$2)
apply (*simp add*: ∗ *dist-real-def*)
by (*simp add*: ∗ *dist-real-def*)
qed

lemma *dist-bound-1*:
fixes *a b x* :: *real*^2
assumes *a*$2 = *x*$2
assumes *b* $\in$ *ball x* ε
assumes $\varepsilon <$ *dist a x*
shows *a*$1 < *x*$1 $\Longrightarrow$ *b*$1 > *a*$1 *a*$1 > *x*$1 $\Longrightarrow$ *b*$1 < *a*$1
proof−
have *1*: *dist a x* = *dist* (*a*$1) (*x*$1) **using** *axis-dist assms*(1) **by** *blast*
have *2*: *dist* (*b*$1) (*x*$1) < ε
by (*metis assms*(2) *dist-commute dist-vec-nth-le mem-ball order-le-less-trans*)
show *a*$1 < *x*$1 $\Longrightarrow$ *b*$1 > *a*$1 *a*$1 > *x*$1 $\Longrightarrow$ *b*$1 < *a*$1
apply (*smt* (*verit*, *ccfv-threshold*) *assms*(1) *assms*(3) *1 2 dist-norm real-norm-def*)
by (*smt* (*verit*, *ccfv-threshold*) *assms*(1) *assms*(3) *1 2 dist-norm real-norm-def*)
qed

lemma *dist-bound-2*:
fixes *a b x* :: *real^2*
assumes *a$1 = x$1*
assumes *b ∈ ball x ε*
assumes *ε < dist a x*
shows *a$2 < x$2 ⟹ b$2 > a$2 a$2 > x$2 ⟹ b$2 < a$2*
proof−
have *1*: *dist a x = dist (a$2) (x$2)* **using** *axis-dist assms(1)* **by** *blast*
have *2*: *dist (b$2) (x$2) < ε*
by (*metis assms(2) dist-commute dist-vec-nth-le mem-ball order-le-less-trans*)
show *a$2 < x$2 ⟹ b$2 > a$2 a$2 > x$2 ⟹ b$2 < a$2*
apply (*smt* (*verit*, *ccfv-threshold*) *assms(1) assms(3) 1 2 dist-norm real-norm-def*)
by (*smt* (*verit*, *ccfv-threshold*) *assms(1) assms(3) 1 2 dist-norm real-norm-def*)
qed

lemma *linepath-bound-1*:
fixes *x y* :: *real^2*
shows *a < x$1 ∧ a < y$1 ⟹ ∀ v ∈ path-image* (*linepath x y*). *a < v$1*
x$1 < b ∧ y$1 < b ⟹ ∀ v ∈ path-image (*linepath x y*). *v$1 < b*
proof−
have ∗: *∀ v ∈ path-image* (*linepath x y*). *∃ u ∈ {0..1}*. *v = (1 − u) ∗R x + u ∗R y*
by (*simp add*: *image-iff linepath-def path-image-def*)
have *1*: *∀ u ∈ {0..1}*. *a < ((1 − u) ∗R x + u ∗R y)$1* **if** *a < x$1 ∧ a < y$1*
proof *clarify*
fix *u* **assume** *u ∈ {0..1::real}*
then have ∗: *u ≥ 0 ∧ 1 − u ≥ 0* **by** *simp*
then show *a < ((1 − u) ∗R x + u ∗R y)$1*
by (*smt* (*verit*) *that scaleR-collapse scaleR-left-mono vector-add-component vector-scaleR-component*)
qed
have *2*: *∀ u ∈ {0..1}*. *((1 − u) ∗R x + u ∗R y)$1 < b* **if** *x$1 < b ∧ y$1 < b*
proof *clarify*
fix *u* **assume** *u ∈ {0..1::real}*
then have ∗: *u ≥ 0 ∧ 1 − u ≥ 0* **by** *simp*
then show *((1 − u) ∗R x + u ∗R y)$1 < b*
by (*smt* (*verit*) *that scaleR-collapse scaleR-left-mono vector-add-component vector-scaleR-component*)
qed
show *a < x$1 ∧ a < y$1 ⟹ ∀ v ∈ path-image* (*linepath x y*). *a < v$1* **using** ∗ *1* **by** *fastforce*
show *x$1 < b ∧ y$1 < b ⟹ ∀ v ∈ path-image* (*linepath x y*). *v$1 < b* **using** ∗ *2* **by** *fastforce*
qed

lemma *linepath-bound-2*:
fixes *x y* :: *real^2*
shows *a < x$2 ∧ a < y$2 ⟹ ∀ v ∈ path-image* (*linepath x y*). *a < v$2*

$x\$2 < b \wedge y\$2 < b \Longrightarrow \forall v \in \textit{path-image}\ (\textit{linepath}\ x\ y).\ v\$2 < b$
proof−
have $*$: $\forall v \in \textit{path-image}\ (\textit{linepath}\ x\ y).\ \exists u \in \{0..1\}.\ v = (1 - u) *_R x + u *_R y$
by (*simp add*: *image-iff linepath-def path-image-def*)
have *1*: $\forall u \in \{0..1\}.\ a < ((1 - u) *_R x + u *_R y)\$2$ **if** $a < x\$2 \wedge a < y\$2$
proof *clarify*
fix u **assume** $u \in \{0..1::real\}$
then have $*$: $u \geq 0 \wedge 1 - u \geq 0$ **by** *simp*
then show $a < ((1 - u) *_R x + u *_R y)\$2$
by (*smt* (*verit*) *that scaleR-collapse scaleR-left-mono vector-add-component vector-scaleR-component*)
qed
have *2*: $\forall u \in \{0..1\}.\ ((1 - u) *_R x + u *_R y)\$2 < b$ **if** $x\$2 < b \wedge y\$2 < b$
proof *clarify*
fix u **assume** $u \in \{0..1::real\}$
then have $*$: $u \geq 0 \wedge 1 - u \geq 0$ **by** *simp*
then show $((1 - u) *_R x + u *_R y)\$2 < b$
by (*smt* (*verit*) *that scaleR-collapse scaleR-left-mono vector-add-component vector-scaleR-component*)
qed
show $a < x\$2 \wedge a < y\$2 \Longrightarrow \forall v \in \textit{path-image}\ (\textit{linepath}\ x\ y).\ a < v\$2$ **using** $*$ *1* **by** *fastforce*
show $x\$2 < b \wedge y\$2 < b \Longrightarrow \forall v \in \textit{path-image}\ (\textit{linepath}\ x\ y).\ v\$2 < b$ **using** $*$ *2* **by** *fastforce*
qed

lemma *linepath-int-corner*:
fixes $x\ y\ z :: real\hat{}2$
assumes $x\$2 \neq y\$2$
assumes $y\$2 = z\$2$
shows $\textit{path-image}\ (\textit{linepath}\ x\ y) \cap \textit{path-image}\ (\textit{linepath}\ y\ z) = \{y\}$
(**is** *path-image ?l1* ∩ *path-image ?l2* = $\{y\}$)
proof−
have *1*: $y \in \textit{path-image}\ ?l1 \cap \textit{path-image}\ ?l2$ **by** *simp*

have $\forall t \in \{0..1\}.\ (?l1\ t)\$2 = y\$2 \longrightarrow t = 1$
proof *clarify*
fix $t :: real$
assume *1*: $t \in \{0..1\}$
assume *2*: $(?l1\ t)\$2 = y\$2$

have $(?l1\ t)\$2 = ((1 - t) * (x\$2) + t * (y\$2))$ **by** (*simp add*: *linepath-def*)
thus $t = 1$
by (*smt* (*verit*, *best*) *assms 2 distrib-right inner-real-def mult.commute real-inner-1-right vector-space-over-itself.scale-cancel-left*)
qed
then have $\forall t \in \{0..1\}.\ (?l1\ t)\$2 = y\$2 \longleftrightarrow t = 1$ **by** (*metis linepath-1'*)
moreover have $\forall t \in \{0..1\}.\ (?l2\ t)\$2 = y\$2$

unfolding *linepath-def*
by (*metis* (*no-types*, *lifting*) *assms*(*2*) *segment-degen-1 vector-add-component vector-scaleR-component*)
ultimately have *2*: *path-image ?l1* ∩ *path-image ?l2* ⊆ {*y*}
by (*smt* (*verit*, *best*) *1 IntD1 IntD2 imageE path-defs*(*4*) *singleton-iff subsetI*)

show *?thesis* **using** *1 2* **by** *fastforce*
qed

lemma *linepath-int-vertical*:
fixes *w x y z* :: *real*^*2*
assumes *w*$*1* ≠ *y*$*1*
assumes *w*$*1* = *x*$*1*
assumes *y*$*1* = *z*$*1*
shows *path-image* (*linepath w x*) ∩ *path-image* (*linepath y z*) = {}
using *assms segment-vertical* **by** *fastforce*

lemma *linepath-int-horizontal*:
fixes *w x y z* :: *real*^*2*
assumes *w*$*2* ≠ *y*$*2*
assumes *w*$*2* = *x*$*2*
assumes *y*$*2* = *z*$*2*
shows *path-image* (*linepath w x*) ∩ *path-image* (*linepath y z*) = {}
using *assms segment-horizontal* **by** *fastforce*

lemma *linepath-int-columns*:
fixes *w x y z* :: *real*^*2*
assumes *w*$*1* < *y*$*1* ∧ *w*$*1* < *z*$*1*
assumes *x*$*1* < *y*$*1* ∧ *x*$*1* < *z*$*1*
shows *path-image* (*linepath w x*) ∩ *path-image* (*linepath y z*) = {}
(**is** *path-image ?l1* ∩ *path-image ?l2* = {})
proof−
have ∀ *t1* ∈ {*0*..*1*}. ∀ *t2* ∈ {*0*..*1*}. (*?l2 t2*)$*1* > (*?l1 t1*)$*1*
by (*smt* (*verit*, *ccfv-SIG*) *assms linepath-bound-1 linepath-in-path path-image-linepath*)
thus *?thesis* **by** (*smt* (*verit*, *best*) *disjoint-iff imageE path-image-def*)
qed

lemma *linepath-int-rows*:
fixes *w x y z* :: *real*^*2*
assumes *w*$*2* < *y*$*2* ∧ *w*$*2* < *z*$*2*
assumes *x*$*2* < *y*$*2* ∧ *x*$*2* < *z*$*2*
shows *path-image* (*linepath w x*) ∩ *path-image* (*linepath y z*) = {}
(**is** *path-image ?l1* ∩ *path-image ?l2* = {})
proof−
have ∀ *t1* ∈ {*0*..*1*}. ∀ *t2* ∈ {*0*..*1*}. (*?l2 t2*)$*2* > (*?l1 t1*)$*2*
by (*smt* (*verit*, *ccfv-SIG*) *assms linepath-bound-2 linepath-in-path path-image-linepath*)
thus *?thesis* **by** (*smt* (*verit*, *best*) *disjoint-iff imageE path-image-def*)
qed

lemma *horizontal-segment-at-0*:
assumes $a > 0$
shows *closed-segment* ((*vector* [0, 0])::(*real*^2)) (*vector* [*a*, 0]) = {*x*. *x*$2 = 0 ∧ *x*$1 ∈ {0..*a*}}
(**is** *?l* = *?s*)
proof−
have *?l* ⊆ *?s*
proof(*rule subsetI*)
fix *x*
assume ∗: *x* ∈ *?l*
then have *x*$2 = 0 **using** *segment-horizontal* **by** *auto*
moreover have 0 ≤ *x*$1 ∧ *x*$1 ≤ *a* **using** ∗ *assms segment-horizontal* **by** *force*
ultimately show *x* ∈ *?s* **by** *force*
qed
moreover have *?s* ⊆ *?l*
proof(*rule subsetI*)
fix *x*
assume ∗: *x* ∈ *?s*
then have *x* = (*x*$1 / *a*) ∗$_R$ (*vector* [*a*, 0]) + (1 − (*x*$1 / *a*)) ∗$_R$ (*vector* [0, 0])
proof−
have (*x*$1 / *a*) ∗$_R$ ((*vector* [*a*, 0])::(*real*^2)) = *vector* [*x*$1, 0]
using *vec-scaleR-2 assms* **by** *fastforce*
moreover have (1 − (*x*$1 / *a*)) ∗$_R$ ((*vector* [0, 0])::(*real*^2)) = *vector* [0, 0]
using *vec-scaleR-2* **by** *simp*
moreover have *x* = *vector* [*x*$1, 0]
by (*smt* (*verit*) ∗ *exhaust-2 mem-Collect-eq vec-eq-iff vector-2*(1) *vector-2*(2))
ultimately show *?thesis*
by (*metis add-cancel-right-right scaleR-collapse vec-scaleR-2 vector-2*(2))
qed
moreover have *x*$1 / *a* ∈ {0..1} **using** ∗ *assms* **by** *fastforce*
ultimately show *x* ∈ *?l*
by (*smt* (*verit*, *del-insts*) *add.commute atLeastAtMost-iff mem-Collect-eq closed-segment-def*)
qed
ultimately show *?thesis* **by** *blast*
qed

lemma *horizontal-segment-at-0′*:
fixes *x y* :: *real*^2
assumes $a > 0$
assumes *x*$1 = 0 ∧ *x*$2 = 0 ∧ *y*$1 = *a* ∧ *y*$2 = 0
shows *closed-segment x y* = {*x*. *x*$2 = 0 ∧ *x*$1 ∈ {0..*a*}}
proof−
have *x* = *vector* [0, 0] ∧ *y* = *vector* [*a*, 0]
by (*smt* (*verit*, *best*) *assms*(2) *exhaust-2 vec-eq-iff vector-2*(1) *vector-2*(2))
thus *?thesis* **using** *horizontal-segment-at-0 assms* **by** *presburger*

qed

lemma *pocket-fill-line-int-aux1*:
fixes *p q* :: *R-to-R2*
defines *p0* ≡ *pathstart p*
defines *p1* ≡ *pathfinish p*
defines *q0* ≡ *pathstart q*
defines *q1* ≡ *pathfinish q*
defines *a* ≡ *p1$1*
defines *l* ≡ *closed-segment p0 p1*
assumes *simple-path p*
assumes *simple-path q*
assumes *p0$1 = 0 ∧ p0$2 = 0 ∧ p1$2 = 0*
assumes *a > 0*
assumes *path-image q ∩ {x. x$2 = 0} ⊆ l*
assumes *path-image p ∩ {x. x$2 = 0} ⊆ l*
assumes *∀ v ∈ path-image p. q0$2 ≤ v$2*
assumes *∀ v ∈ path-image p. q1$2 > v$2*
shows *path-image p ∩ path-image q ≠ {}*
proof−
have *p0*: *p0 = 0*
by (*metis* (*mono-tags*, *opaque-lifting*) *assms*(*9*) *exhaust-2 vec-eq-iff zero-index*)
moreover have *p1*: *p1 = vector* [*a*, *0*]
by (*smt* (*verit*) *a-def assms*(*9*) *exhaust-2 vec-eq-iff vector-2*(*1*) *vector-2*(*2*))

obtain *a-x* **where** *a-x*: *∀ v ∈ path-image p ∪ path-image q. a-x < v$1*
proof−
let *?a-x = Inf* ((*λv. v$1*)*`*(*path-image p ∪ path-image q*))
have *compact* (*path-image p ∪ path-image q*)
by (*simp add*: *assms*(*7*) *assms*(*8*) *compact-Un compact-simple-path-image*)
moreover have *continuous-on UNIV* ((*λv. v$1*)::(*real^2 ⇒ real*))
by (*simp add*: *continuous-on-component*)
ultimately have ∗: *compact* ((*λv. v$1*)*`*(*path-image p ∪ path-image q*))
by (*meson compact-continuous-image continuous-on-subset top-greatest*)
then have *∀ x ∈* ((*λv. v$1*)*`*(*path-image p ∪ path-image q*)). *?a-x ≤ x*
by (*simp add*: *assms*(*7*) *assms*(*8*) *bounded-component-cart bounded-has-Inf*(*1*) *bounded-simple-path-image*)
thus *?thesis* **using** *that*[*of ?a-x − 1*] **by** (*smt* (*verit*, *ccfv-SIG*) *assms*(*10*) *imageI*)
qed
obtain *b-x* **where** *b-x*: *∀ v ∈ path-image p ∪ path-image q. b-x > v$1*
proof−
let *?b-x = Sup* ((*λv. v$1*)*`*(*path-image p ∪ path-image q*))
have *compact* (*path-image p ∪ path-image q*)
by (*simp add*: *assms*(*7*) *assms*(*8*) *compact-Un compact-simple-path-image*)
moreover have *continuous-on UNIV* ((*λv. v$1*)::(*real^2 ⇒ real*))
by (*simp add*: *continuous-on-component*)
ultimately have ∗: *compact* ((*λv. v$1*)*`*(*path-image p ∪ path-image q*))
by (*meson compact-continuous-image continuous-on-subset top-greatest*)

then have $\forall x \in ((\lambda v.\ v\$1)\text{`}(\textit{path-image } p \cup \textit{path-image } q)).\ ?b\text{-}x \geq x$
by (*simp add*: *assms*(7) *assms*(8) *bounded-component-cart bounded-has-Sup*(1) *bounded-simple-path-image*)
thus *?thesis* **using** *that*[*of ?b-x + 1*] **by** (*smt* (*verit*, *ccfv-SIG*) *assms*(10) *imageI*)
qed
obtain *b-y* **where** *b-y*: $\forall v \in \textit{path-image } p \cup \textit{path-image } q.\ b\text{-}y > v\$2$
proof−
let *?b-y* = *Sup* $((\lambda v.\ v\$2)\text{`}(\textit{path-image } p \cup \textit{path-image } q))$
have *compact* (*path-image p* ∪ *path-image q*)
by (*simp add*: *assms*(7) *assms*(8) *compact-Un compact-simple-path-image*)
moreover have *continuous-on UNIV* $((\lambda v.\ v\$2)::(real\hat{}2 \Rightarrow real))$
by (*simp add*: *continuous-on-component*)
ultimately have ∗: *compact* $((\lambda v.\ v\$2)\text{`}(\textit{path-image } p \cup \textit{path-image } q))$
by (*meson compact-continuous-image continuous-on-subset top-greatest*)
then have $\forall x \in ((\lambda v.\ v\$2)\text{`}(\textit{path-image } p \cup \textit{path-image } q)).\ ?b\text{-}y \geq x$
by (*simp add*: *assms*(7) *assms*(8) *bounded-component-cart bounded-has-Sup*(1) *bounded-simple-path-image*)
thus *?thesis* **using** *that*[*of ?b-y + 1*] **by** (*smt* (*verit*, *ccfv-SIG*) *assms*(10) *imageI*)
qed

let *?l1* = *linepath p1* (*vector* [*b-x*, *0*])
let *?l2* = *linepath* (*vector* [*b-x*, *0*]) ((*vector* [*b-x*, *b-y*])::(*real*^2))
let *?l3* = *linepath* (*vector* [*b-x*, *b-y*]) ((*vector* [*a-x*, *b-y*])::(*real*^2))
let *?l4* = *linepath* (*vector* [*a-x*, *b-y*]) ((*vector* [*a-x*, *0*])::(*real*^2))
let *?l5* = *linepath* (*vector* [*a-x*, *0*]) *p0*

let *?R′* = *?l1* +++ *?l2* +++ *?l3* +++ *?l4* +++ *?l5*
let *?R* = *p* +++ *?R′*

have *R-y-b*: $\forall v \in \textit{path-image } ?R.\ v\$2 \leq b\text{-}y$
proof−
have $\forall v \in \textit{path-image } ?l1.\ v\$2 \leq b\text{-}y$
by (*metis UnCI assms*(9) *b-y less-eq-real-def p1-def path-image-linepath pathfinish-in-path-image segment-horizontal vector-2*(2))
moreover have $\forall v \in \textit{path-image } ?l2.\ v\$2 \leq b\text{-}y$
by (*smt* (*verit*, *ccfv-SIG*) *UnCI assms*(9) *b-y p0-def path-image-linepath pathstart-in-path-image segment-vertical vector-2*(1) *vector-2*(2))
moreover have $\forall v \in \textit{path-image } ?l3.\ v\$2 \leq b\text{-}y$
by (*simp add*: *segment-horizontal*)
moreover have $\forall v \in \textit{path-image } ?l4.\ v\$2 \leq b\text{-}y$
by (*smt* (*verit*, *best*) *UnCI assms*(9) *b-y p0-def path-image-linepath pathstart-in-path-image segment-vertical vector-2*(1) *vector-2*(2))
moreover have $\forall v \in \textit{path-image } ?l5.\ v\$2 \leq b\text{-}y$
by (*smt* (*verit*) *UnI1 assms*(9) *b-y linepath-image-01 p0-def path-defs*(4) *pathstart-in-path-image segment-horizontal vector-2*(2))
ultimately show *?thesis* **by** (*smt* (*verit*, *best*) *UnCI b-y not-in-path-image-join*)
qed

have *R-y-q0*: $\forall\, v \in$ *path-image ?R*. *v*\$*2* $\geq$ *q0*\$*2*
proof−
have $\forall\, v \in$ *path-image ?l1*. *v*\$*2* $\geq$ *q0*\$*2*
using *assms*(*13*) *assms*(*9*) *p1-def pathfinish-in-path-image segment-horizontal*
by *fastforce*
moreover have $\forall\, v \in$ *path-image ?l2*. *v*\$*2* $\geq$ *q0*\$*2*
by (*smt* (*verit*) *UnCI assms*(*13*) *assms*(*9*) *b-y p1-def path-image-linepath pathfinish-in-path-image segment-vertical vector-2*(*1*) *vector-2*(*2*))
moreover have $\forall\, v \in$ *path-image ?l3*. *v*\$*2* $\geq$ *q0*\$*2*
by (*metis calculation*(*2*) *ends-in-segment*(*2*) *path-image-linepath segment-horizontal vector-2*(*2*))
moreover have $\forall\, v \in$ *path-image ?l4*. *v*\$*2* $\geq$ *q0*\$*2*
by (*smt* (*verit*) *UnCI assms*(*13*) *assms*(*9*) *b-y p1-def path-image-linepath pathfinish-in-path-image segment-vertical vector-2*(*1*) *vector-2*(*2*))
moreover have $\forall\, v \in$ *path-image ?l5*. *v*\$*2* $\geq$ *q0*\$*2*
by (*metis assms*(*13*) *assms*(*9*) *p0-def path-image-linepath pathstart-in-path-image segment-horizontal vector-2*(*2*))
ultimately show *?thesis*
by (*metis assms*(*13*) *not-in-path-image-join*)
qed

have *R-x-a*: $\forall\, v \in$ *path-image ?R*. *v*\$*1* $\geq$ *a-x*
proof−
have $\forall\, v \in$ *path-image ?l1*. *v*\$*2* $\geq$ *a-x*
by (*metis UnCI a-x assms*(*9*) *linorder-le-cases linorder-not-less p0-def path-image-linepath pathstart-in-path-image segment-horizontal vector-2*(*2*))
moreover have $\forall\, v \in$ *path-image ?l2*. *v*\$*2* $\geq$ *a-x*
by (*smt* (*verit*) *UnCI assms*(*9*) *b-y calculation p0-def path-image-linepath pathstart-in-path-image pathstart-linepath segment-vertical vector-2*(*1*) *vector-2*(*2*))
moreover have $\forall\, v \in$ *path-image ?l3*. *v*\$*2* $\geq$ *a-x*
by (*metis calculation*(*2*) *ends-in-segment*(*2*) *path-image-linepath segment-horizontal vector-2*(*2*))
moreover have $\forall\, v \in$ *path-image ?l4*. *v*\$*2* $\geq$ *a-x*
by (*smt* (*verit*) *assms*(*9*) *calculation*(*1*) *calculation*(*3*) *ends-in-segment*(*1*) *path-image-linepath segment-vertical vector-2*(*1*) *vector-2*(*2*))
moreover have $\forall\, v \in$ *path-image ?l5*. *v*\$*2* $\geq$ *a-x*
by (*smt* (*verit*, *del-insts*) *UnCI a-x assms*(*9*) *p0-def path-image-linepath pathstart-in-path-image segment-horizontal vector-2*(*2*))
ultimately show *?thesis*
by (*smt* (*verit*,*del-insts*) *UnCI a-x assms*(*9*) *b-x not-in-path-image-join p1-def path-image-linepath pathfinish-in-path-image segment-horizontal segment-vertical vector-2*(*1*) *vector-2*(*2*))
qed

have *closed*: *closed-path ?R* **using** *assms p0-def* **unfolding** *simple-path-def closed-path-def*
by *simp*
have *simple*: *simple-path ?R*
proof−
have *arc ?R′*

proof −
let *?a* = *p1*
let *?b* = (*vector* [*b-x*, *0*])::(*real^2*)
let *?c* = (*vector* [*b-x*, *b-y*])::(*real^2*)
let *?d* = (*vector* [*a-x*, *b-y*])::(*real^2*)
let *?e* = (*vector* [*a-x*, *0*])::(*real^2*)
let *?f* = *p0*

have *arcs*: *arc ?l1* ∧ *arc ?l2* ∧ *arc ?l3* ∧ *arc ?l4* ∧ *arc ?l5*
by (*smt* (*verit*, *ccfv-SIG*) *UnCI a-x arc-linepath assms*(*9*) *b-x b-y p0-def p1-def pathfinish-in-path-image pathstart-in-path-image vector-2*(*1*) *vector-2*(*2*))

have *l4l5*: *path-image ?l4* ∩ *path-image ?l5* = {*pathfinish ?l4*}
using *linepath-int-corner*[*of ?d ?e ?f*] *arc-simple-path arcs constant-linepath-is-not-loop-free p0 simple-path-def*
by *auto*
have *l3l4*: *path-image ?l3* ∩ *path-image ?l4* = {*pathfinish ?l3*}
using *linepath-int-corner*[*of ?c ?d ?e*]
by (*metis Int-commute arc-simple-path arcs closed-segment-commute linepath-0′ linepath-int-corner path-image-linepath pathfinish-linepath pathstart-def vector-2*(*2*))
have *l2l3*: *path-image ?l2* ∩ *path-image ?l3* = {*pathfinish ?l2*}
using *linepath-int-corner*[*of ?b ?c ?d*]
by (*metis Int-commute arc-simple-path arcs linepath-0′ linepath-int-corner pathfinish-linepath pathstart-def vector-2*(*2*))
have *l1l2*: *path-image ?l1* ∩ *path-image ?l2* = {*pathfinish ?l1*}
using *linepath-int-corner*[*of ?a ?b ?c*]
by (*metis Int-commute arc-distinct-ends arcs assms*(*9*) *closed-segment-commute linepath-int-corner path-image-linepath pathfinish-linepath pathstart-linepath vector-2*(*2*))

have *l3l5*: *path-image ?l3* ∩ *path-image ?l5* = {}
using *linepath-int-horizontal*[*of ?c ?d ?e ?f*]
by (*metis arc-distinct-ends arcs assms*(*9*) *linepath-int-horizontal pathfinish-linepath pathstart-linepath vector-2*(*2*))
have *l2l4*: *path-image ?l2* ∩ *path-image ?l4* = {}
using *linepath-int-vertical*[*of ?b ?c ?d ?e*]
by (*metis arc-distinct-ends arcs linepath-int-vertical pathfinish-linepath pathstart-linepath vector-2*(*1*))
have *l1l3*: *path-image ?l1* ∩ *path-image ?l3* = {}
using *linepath-int-vertical*[*of ?a ?b ?c ?d*]
by (*metis arc-distinct-ends arcs assms*(*9*) *linepath-int-horizontal pathfinish-linepath pathstart-linepath vector-2*(*2*))

have *l2l5*: *path-image ?l2* ∩ *path-image ?l5* = {}
using *linepath-int-columns*[*of ?b ?c ?e ?f*]
by (*smt* (*verit*, *ccfv-threshold*) *Int-commute UnCI a-x b-x linepath-int-columns p0 p0-def pathstart-in-path-image pathstart-join vector-2*(*1*) *verit-comp-simplify1*(*3*))
have *l1l4*: *path-image ?l1* ∩ *path-image ?l4* = {}
using *linepath-int-columns*[*of ?a ?b ?d ?e*]
by (*smt* (*verit*) *UnCI a-x assms*(*9*) *b-x disjoint-iff p1-def path-image-linepath*

pathfinish-in-path-image segment-horizontal segment-vertical vector-2(1) vector-2(2))

have *l1l5*: *path-image ?l1 ∩ path-image ?l5 = {}*
using *linepath-int-columns[of ?a ?b ?e ?f]*
by (*smt* (*verit*) *UnCI a-def a-x assms(10) assms(9) b-x disjoint-iff p1-def path-image-linepath pathfinish-in-path-image segment-horizontal vector-2(1) vector-2(2)*)

have *path-image ?l4 ∩ path-image ?l5 = {pathfinish ?l4}*
using *l4l5* **by** *blast*
moreover have *sf-45*: *pathfinish ?l4 = pathstart ?l5* **by** *simp*
ultimately have *arc* (*?l4 +++ ?l5*)
by (*metis arc-join-eq-alt arcs*)
moreover have *path-image ?l3 ∩ path-image* (*?l4 +++ ?l5*) = {*pathfinish ?l3*}
using *l3l4 l3l5*
by (*metis* (*no-types*, *lifting*) *Int-Un-distrib sf-45 insert-is-Un path-image-join*)
moreover have *sf-345*: *pathfinish ?l3 = pathstart* (*?l4 +++ ?l5*) **by** *simp*
ultimately have *arc* (*?l3 +++ ?l4 +++ ?l5*)
by (*metis arc-join-eq-alt arcs*)
moreover have *path-image ?l2 ∩ path-image* (*?l3 +++ ?l4 +++ ?l5*) = {*pathfinish ?l2*}
using *l2l3 l2l4 l2l5*
by (*smt* (*verit*) *Int-Un-distrib sf-45 sf-345 insert-is-Un path-image-join sup-bot-left*)
moreover have *sf-2345*: *pathfinish ?l2 = pathstart* (*?l3 +++ ?l4 +++ ?l5*) **by** *simp*
ultimately have *arc* (*?l2 +++ ?l3 +++ ?l4 +++ ?l5*)
by (*metis arc-join-eq-alt arcs*)
moreover have *path-image ?l1 ∩ path-image* (*?l2 +++ ?l3 +++ ?l4 +++ ?l5*) = {*pathfinish ?l1*}
proof −
have *path-image* (*?l2 +++ ?l3 +++ ?l4 +++ ?l5*)
= path-image ?l2 ∪ path-image ?l3 ∪ path-image ?l4 ∪ path-image ?l5
by (*simp add*: *path-image-join sup-assoc*)
thus *?thesis* **using** *l1l2 l1l3 l1l4 l1l5* **by** *blast*
qed
moreover have *pathfinish ?l1 = pathstart* (*?l2 +++ ?l3 +++ ?l4 +++ ?l5*) **by** *simp*
ultimately show *arc* (*?l1 +++ ?l2 +++ ?l3 +++ ?l4 +++ ?l5*)
by (*metis arc-join-eq-alt arcs*)
qed
moreover have *loop-free p* **using** *assms(1) assms(7) simple-path-def* **by** *blast*
moreover have *path-image ?R′ ∩ path-image p* = {*p0*, *p1*}
proof −
have *path-image p ∩ path-image ?l2* = {} **using** *b-x segment-vertical* **by** *auto*
moreover have *path-image p ∩ path-image ?l3* = {} **using** *b-y segment-horizontal* **by** *auto*
moreover have *path-image p ∩ path-image ?l4* = {} **using** *a-x segment-vertical*

by *auto*
moreover have *path-image p ∩ path-image ?l1 = {p1}*
proof−
have *p1 ∈ path-image p* **using** *p1-def* **by** *blast*
moreover have *path-image p ∩ path-image ?l1 ⊆ {p1}*
proof(*rule subsetI*)
fix *x* **assume** ∗: *x ∈ path-image p ∩ path-image ?l1*
then have *x$1 ≤ a*
using *a-def assms(10) assms(12) assms(9) l-def linepath-image-01 segment-horizontal* **by** *auto*
moreover have *x$1 ≥ a*
by (*smt* (*verit*) ∗ *Int-iff Un-iff a-def assms(9) b-x linepath-image-01 path-defs(4) segment-horizontal vector-2(1) vector-2(2)*)
moreover have *x$2 = 0* **using** ∗ *assms(9) segment-horizontal* **by** *auto*
ultimately show *x ∈ {p1}* **using** *a-def assms(9) segment-vertical* **by** *fastforce*
qed
ultimately show *?thesis* **by** *auto*
qed
moreover have *path-image p ∩ path-image ?l5 = {p0}*
proof−
have *p0 ∈ path-image p* **using** *p0-def* **by** *blast*
moreover have *path-image p ∩ path-image ?l5 ⊆ {p0}*
proof(*rule subsetI*)
fix *x* **assume** ∗: *x ∈ path-image p ∩ path-image ?l5*
then have *x$1 ≤ 0*
using *R-x-a assms(9) p0-def pathstart-in-path-image segment-horizontal* **by** *fastforce*
moreover have *x$1 ≥ 0*
proof−
have *x ∈ {x. x$2 = 0}* **using** ∗ *assms(9) segment-horizontal* **by** *fastforce*
then have *x ∈ l* **using** ∗ *assms(12)* **by** *auto*
thus *?thesis* **using** *a-def assms(10) assms(9) l-def segment-horizontal* **by** *auto*
qed
moreover have *x$2 = 0* **using** ∗ *assms(9) segment-horizontal* **by** *auto*
ultimately show *x ∈ {p0}* **using** *a-def assms(9) segment-vertical* **by** *fastforce*
qed
ultimately show *?thesis* **by** *auto*
qed
moreover have *path-image ?R′*
= path-image ?l1 ∪ path-image ?l2 ∪ path-image ?l3 ∪ path-image ?l4 ∪ path-image ?l5
by (*simp add*: *Un-assoc path-image-join*)
ultimately show *?thesis* **by** *fast*
qed
moreover have *arc p*
using *a-def arc-simple-path assms(10) assms(7) p0 p0-def p1-def* **by** *fastforce*

ultimately show *?thesis*
by (*metis* (*no-types*, *lifting*) *simple-path-join-loop-eq Int-commute dual-order.refl p0-def p1-def pathfinish-join pathfinish-linepath pathstart-join pathstart-linepath*)
qed

have *inside-outside*: *inside-outside ?R* (*path-inside ?R*) (*path-outside ?R*)
using *closed simple Jordan-inside-outside-real2*
by (*simp add*: *closed-path-def inside-outside-def path-inside-def path-outside-def*)

have *interior-frontier*: *path-inside ?R* = *interior* (*path-inside ?R*)
∧ *frontier* (*path-inside ?R*) = *path-image ?R*
using *inside-outside interior-open* **unfolding** *inside-outside-def* **by** *auto*

have *path-image q* ∩ *path-image ?l1* ⊆ {*p1*}
proof(*rule subsetI*)
fix *x* **assume** ∗: *x* ∈ *path-image q* ∩ *path-image ?l1*
then have *x*$*1* ≤ *a* **using** *a-def assms*(*10*) *assms*(*11*) *assms*(*9*) *l-def segment-horizontal* **by** *auto*
moreover have *x*$*1* ≥ *a*
by (*smt* (*verit*) ∗ *Int-iff Un-iff a-def assms*(*9*) *b-x linepath-image-01 path-defs*(*4*) *segment-horizontal vector-2*(*1*) *vector-2*(*2*))
moreover have *x*$*2* = *0* **using** ∗ *assms*(*9*) *segment-horizontal* **by** *auto*
ultimately show *x* ∈ {*p1*} **using** *a-def assms*(*9*) *segment-vertical* **by** *fastforce*
qed
moreover have *path-image q* ∩ *path-image ?l5* ⊆ {*p0*}
proof(*rule subsetI*)
fix *x* **assume** ∗: *x* ∈ *path-image q* ∩ *path-image ?l5*
then have *x*$*1* ≤ *0*
using *R-x-a assms*(*9*) *p0-def pathstart-in-path-image segment-horizontal* **by** *fastforce*
moreover have *x*$*1* ≥ *0*
using ∗ *a-def assms*(*10*) *assms*(*11*) *assms*(*9*) *l-def segment-horizontal* **by** *auto*
moreover have *x*$*2* = *0* **using** ∗ *assms*(*9*) *segment-horizontal* **by** *auto*
ultimately show *x* ∈ {*p0*} **using** *a-def assms*(*9*) *segment-vertical* **by** *fastforce*
qed
moreover have *?thesis* **if** *p1* ∈ *path-image q* ∩ *path-image ?l1* **using** *p1-def that* **by** *blast*
moreover have *?thesis* **if** *p0* ∈ *path-image q* ∩ *path-image ?l5* **using** *p0-def that* **by** *blast*
moreover have *?thesis* **if**
q-int-l1: *path-image q* ∩ *path-image ?l1* = {} **and**
q-int-l5: *path-image q* ∩ *path-image ?l5* = {}
proof−
have *q-int-l2*: *path-image q* ∩ *path-image ?l2* = {}
using *b-x segment-vertical* **by** *auto*
moreover have *q-int-l3*: *path-image q* ∩ *path-image ?l3* = {}
using *UnCI b-y segment-horizontal* **by** *auto*
moreover have *q-int-l4*: *path-image q* ∩ *path-image ?l4* = {}
using *a-x segment-vertical* **by** *auto*

moreover have *?thesis* **if** *q0 ∈ path-image p* **using** *q0-def that* **by** *blast*
moreover have *path-image q ∩ path-image ?R ≠ {}* **if** *q0 ∉ path-image p*
proof−
have *q0 ∈ path-outside ?R*

proof−
let *?e2′ = (vector [0, −1])::(real^2)*
let *?ray = λd. q0 + d ∗R ?e2′*
have *¬ (∃ d>0. ?ray d ∈ path-image ?R)*
proof−
have *∀ d>0. (?ray d)$2 < q0$2* **by** *auto*
thus *?thesis* **using** *R-y-q0* **by** *fastforce*
qed
moreover have *bounded (path-inside ?R)* **using** *bounded-finite-inside simple*
by *blast*
moreover have *?e2′ ≠ 0* **by** *(metis vector-2(2) zero-index zero-neq-neg-one)*
ultimately have *q0 ∉ path-inside ?R*
using *ray-to-frontier[of path-inside ?R] interior-frontier* **by** *metis*
moreover have *q0 ∉ path-image ?R*
using *that q-int-l1 q-int-l2 q-int-l3 q-int-l4 q-int-l5*
by *(simp add: disjoint-iff not-in-path-image-join pathstart-in-path-image*
q0-def)
ultimately show *?thesis* **using** *inside-outside* **unfolding** *inside-outside-def*
by *blast*
qed
then have *q0 ∈ − (path-inside ?R)*
by *(metis ComplI IntI equals0D inside-Int-outside path-inside-def path-outside-def)*
moreover have *q1 ∈ path-inside ?R*

proof−
let *?e = (vector [q1$1, b-y])::(real^2)*
let *?d1 = (vector [b-x, b-y])::(real^2)*
let *?d2 = (vector [a-x, b-y])::(real^2)*
obtain *ε* **where** *ε: 0 < ε ∧ ε < dist ?e q1 ∧ ε < dist ?e ?d1 ∧ ε < dist ?e*
?d2
proof−
have *?e ≠ q1*
by *(metis UnCI b-y order-less-irrefl pathfinish-in-path-image q1-def*
vector-2(2))
moreover have *?e ≠ ?d1*
by *(smt (verit) UnCI b-x pathfinish-in-path-image q1-def vector-2(1))*
moreover have *?e ≠ ?d2*
by *(metis UnCI a-x order-less-irrefl pathfinish-in-path-image q1-def*
vector-2(1))
ultimately have *0 < dist ?e q1 ∧ 0 < dist ?e ?d1 ∧ 0 < dist ?e ?d2* **by**
simp
then have *0 < Min {dist ?e q1, dist ?e ?d1, dist ?e ?d2}* **by** *auto*
then obtain *ε* **where** *0 < ε ∧ ε < Min {dist ?e q1, dist ?e ?d1, dist ?e*
?d2}

by (*meson field-lbound-gt-zero*)
thus *?thesis* **using** *that* **by** *auto*
qed
then have *?e* ∈ *path-image ?l3*
by (*simp add*: *a-x b-x q1-def segment-horizontal less-eq-real-def pathfinish-in-path-image*)
then have *?e* ∈ *path-image ?R* **by** (*simp add*: *p1-def path-image-join*)
then have *?e* ∈ *frontier* (*path-inside ?R*)
using *inside-outside* **unfolding** *inside-outside-def* **by** *blast*
then obtain *int-p* **where** *int-p*: *int-p* ∈ *ball ?e ε* ∧ *int-p* ∈ *path-inside ?R*
by (*meson ε inside-outside frontier-straddle mem-ball*)

have *int-p-x*: *a-x* < *int-p*$*1* ∧ *int-p*$*1* < *b-x*
by (*metis* (*mono-tags*, *lifting*) *dist-bound-1 UnI2 ε a-x b-x dist-commute int-p pathfinish-in-path-image q1-def vector-2*(*1*) *vector-2*(*2*))
have *int-p*$*2* < *b-y*
proof(*rule ccontr*)
have *int-p*$*2* ≠ *b-y*
proof−
have *int-p*$*2* = *b-y* ⟹ *int-p* ∈ *path-image ?l3*
using *int-p-x* **by** (*simp add*: *segment-horizontal*)
moreover have *int-p* ∈ *path-image ?l3* ⟹ *int-p* ∈ *path-image ?R*
by (*simp add*: *p1-def path-image-join*)
moreover have *path-image ?R* ∩ *path-inside ?R* = {}
using *inside-outside* **unfolding** *inside-outside-def* **by** *blast*
ultimately show *?thesis* **using** *int-p* **by** *fast*
qed
moreover assume ¬ *int-p*$*2* < *b-y*
ultimately have ∗: *int-p*$*2* > *b-y* **by** *simp*

let *?e2* = (*vector* [*0*, *1*])::(*real*^*2*)
let *?ray* = λ*d*. *int-p* + *d* ∗$_R$ *?e2*
have ¬ (∃ *d*>*0*. *?ray d* ∈ *path-image ?R*)
proof−
have ∀ *d*>*0*. (*?ray d*)$*2* > *b-y* **using** ∗ **by** *auto*
thus *?thesis* **using** *R-y-b* **by** *fastforce*
qed
moreover have *bounded* (*path-inside ?R*) **using** *bounded-finite-inside simple* **by** *blast*
moreover have *?e2* ≠ *0* **using** *e1e2-basis*(*4*) **by** *force*
ultimately have *int-p* ∉ *path-inside ?R*
using *ray-to-frontier*[*of path-inside ?R*] *interior-frontier* **by** *metis*
thus *False* **using** *int-p* **by** *blast*
qed
moreover have *int-p*$*2* > *q1*$*2*
proof−
have *dist int-p ?e* < *ε* **using** *ε dist-commute-lessI int-p mem-ball* **by** *blast*
then have *dist* (*int-p*$*2*) (*?e*$*2*) < *ε* **by** (*smt* (*verit*, *best*) *dist-vec-nth-le*)
then have *1*: *int-p*$*2* > *?e*$*2* − *ε* **by** (*simp add*: *dist-real-def*)

have *q1$1 = ?e$1* **by** *simp*
then have *dist q1 ?e = dist (q1$2) (?e$2)* **using** *axis-dist* **by** *blast*
then have *q1$2 < ?e$2 − ε*
by (*smt* (*verit*) *UnCI ε b-y dist-commute dist-real-def pathfinish-in-path-image q1-def vector-2*(*2*))
moreover have *q1$2 < ?e$2* **by** (*simp add*: *b-y pathfinish-in-path-image q1-def*)
moreover have *dist q1 ?e > ε* **by** (*metis ε dist-commute*)
ultimately have *q1$2 < ?e$2 − ε* **by** *presburger*
thus *?thesis* **using** *1* **by** *force*
qed
ultimately have *int-p-y*: *int-p$2 < b-y ∧ int-p$2 > q1$2* **by** *blast*

let *?int-l = linepath int-p q1*

have *path-image ?int-l ∩ path-image p = {}*
proof−
have *∀ x ∈ path-image p. (?int-l 0)$2 > x$2*
by (*smt* (*verit*) *int-p-y assms*(*14*) *linepath-0′*)
moreover have *∀ x ∈ path-image p. (?int-l 1)$2 > x$2*
by (*simp add*: *assms*(*14*) *linepath-1′*)
ultimately have *∀ x ∈ path-image p. ∀ y ∈ path-image ?int-l. y$2 > x$2*
by (*metis assms*(*14*) *linepath-0′ linepath-bound-2*(*1*))
thus *?thesis* **by** *blast*
qed
moreover have *path-image ?int-l ∩ path-image ?l1 = {}*
by (*smt* (*verit*, *best*) *assms*(*14*) *assms*(*9*) *disjoint-iff int-p-y linepath-int-rows p0-def pathstart-in-path-image vector-2*(*2*))
moreover have *path-image ?int-l ∩ path-image ?l2 = {}*
by (*metis UnCI b-x int-p-x linepath-int-columns pathfinish-in-path-image q1-def vector-2*(*1*))
moreover have *path-image ?int-l ∩ path-image ?l3 = {}*
using *int-p-y linepath-int-rows* **by** *auto*
moreover have *path-image ?int-l ∩ path-image ?l4 = {}*
by (*metis UnCI a-x inf-commute int-p-x linepath-int-columns pathfinish-in-path-image q1-def vector-2*(*1*))
moreover have *path-image ?int-l ∩ path-image ?l5 = {}*
by (*smt* (*verit*, *best*) *assms*(*14*) *assms*(*9*) *disjoint-iff int-p-y linepath-int-rows p0-def pathstart-in-path-image vector-2*(*2*))
ultimately have *path-image ?int-l ∩ path-image ?R = {}*
by (*simp add*: *disjoint-iff not-in-path-image-join*)
then have *path-image ?int-l ⊆ path-inside ?R ∨ path-image ?int-l ⊆ path-outside ?R*
by (*smt* (*verit*, *ccfv-SIG*) *convex-imp-path-connected convex-segment*(*1*) *disjoint-insert*(*1*) *insert-Diff inside-outside-def int-p linepath-image-01 local.inside-outside path-connected-not-frontier-subset path-defs*(*4*) *pathstart-in-path-image pathstart-linepath*)
moreover have *?int-l 0 = int-p ∧ int-p ∈ path-inside ?R*
using *int-p* **by** (*simp add*: *linepath-0′*)

ultimately have *path-image ?int-l* $\subseteq$ *path-inside ?R*
using *inside-outside-def local.inside-outside* **by** *auto*
thus *?thesis* **by** *auto*
qed
ultimately have *path-image q* $\cap$ $-$ (*path-inside ?R*) $\neq$ {} $\wedge$ *path-image q* $\cap$ (*path-inside ?R*) $\neq$ {}
unfolding *q0-def q1-def* **by** *fast*
moreover have *path-connected* (*path-image q*)
by (*simp add*: *assms*(*8*) *path-connected-path-image simple-path-imp-path*)
moreover have *path-image ?R* = *frontier* (*path-inside ?R*)
using *inside-outside* **unfolding** *inside-outside-def p0-def path-inside-def* **by** *auto*
ultimately show *?thesis* **by** (*metis Diff-eq Diff-eq-empty-iff path-connected-not-frontier-subset*)
qed
ultimately show *?thesis*
by (*smt* (*verit*, *ccfv-threshold*) *disjoint-iff-not-equal not-in-path-image-join q-int-l1 q-int-l5*)
qed
ultimately show *?thesis* **by** *auto*
qed

lemma *pocket-fill-line-int-aux2*:
fixes *p q* :: *R-to-R2*
fixes *A* :: (*real^2*) *set*
defines *p0* $\equiv$ *pathstart p*
defines *p1* $\equiv$ *pathfinish p*
defines *a* $\equiv$ *p1$1*
defines *l* $\equiv$ *closed-segment p0 p1*
assumes *simple-path p*
assumes *p0$1* = *0* $\wedge$ *p0$2* = *0* $\wedge$ *p1$2* = *0*
assumes *a* > *0*
assumes *convex A* $\wedge$ *compact A*
assumes {*p0*, *p1*} $\subseteq$ *frontier A*
assumes *p* ‘ {*0*<..<*1*} $\subseteq$ *interior A*
shows *path-image p* $\cap$ {*x*. *x$2* = *0*} $\subseteq$ *l*
proof$-$
have *l*: *l* = {*x*. *x$2* = *0* $\wedge$ *x$1* $\in$ {*0*..*a*}}
using *horizontal-segment-at-0'' a-def assms*(*6*) *assms*(*7*) *l-def* **by** *presburger*
have *endpoints*: (*p 0*)*$1* = *0* $\wedge$ (*p 0*)*$2* = *0* $\wedge$ (*p 1*)*$1* = *a* $\wedge$ (*p 1*)*$2* = *0*
by (*metis a-def assms*(*6*) *p0-def p1-def pathfinish-def pathstart-def*)

have *False* **if** $*$: $\exists$ *t* $\in$ {*0*..*1*}. (*p t*)*$2* = *0* $\wedge$ ((*p t*)*$1* > *a* $\vee$ (*p t*)*$1* < *0*)
proof$-$
obtain *t* **where** *t* $\in$ {*0*<..<*1*} $\wedge$ (*p t*)*$2* = *0* $\wedge$ ((*p t*)*$1* > *a* $\vee$ (*p t*)*$1* < *0*)
by (*metis* $*$ *assms*(*7*) *endpoints atLeastAtMost-iff greaterThanLessThan-iff less-eq-real-def linorder-not-le*)
then obtain *x* **where** *x*: *x* $\in$ *p*‘{*0*<..<*1*} $\wedge$ *x$2* = *0* $\wedge$ (*x$1* > *a* $\vee$ *x$1* < *0*)
by *blast*
thus *False*

using *pocket-fill-line-int-aux*[*of p0 p1 x A*]
by (*smt* (*verit*, *del-insts*) *Diff-iff a-def assms*(*10*) *assms*(*6*) *assms*(*7*) *assms*(*8*) *assms*(*9*) *empty-subsetI endpoints exhaust-2 frontier-def frontier-subset-compact insert-subset interior-subset p0-def pathstart-def subset-eq vec-eq-iff zero-index*)
qed
then have $\forall\, t \in \{0..1\}.\ (p\ t)\$2 = 0 \longrightarrow (p\ t)\$1 \in \{0..a\}$ **by** *fastforce*
then have $\forall\, v \in$ *path-image p*. $v\$2 = 0 \longrightarrow v\$1 \in \{0..a\}$ **by** (*simp add*: *imageE path-defs*(*4*))
thus *?thesis* **using** *l* **by** *blast*
qed

lemma *three-points-on-line*:
fixes *a b* :: *'a*::*real-vector*
assumes $A =$ *affine hull* $\{a, b\}$
assumes $a \neq b$
assumes $\{x, y, z\} \subseteq A$
assumes $x \neq y \wedge y \neq z \wedge x \neq z$
shows $x \in$ *open-segment y z* $\vee$ $y \in$ *open-segment x z* $\vee$ $z \in$ *open-segment x y*
proof−
let *?u* $= b - a$

have $*$: $\bigwedge \alpha\ \beta\ \gamma$::*real*. $\alpha \in$ *open-segment* $\beta\ \gamma$
$\Longrightarrow a + \alpha *_R$ *?u* $\in$ *open-segment* $(a + \beta *_R$ *?u*$)\ (a + \gamma *_R$ *?u*$)$
proof−
fix $\alpha\ \beta\ \gamma$ :: *real*
assume $*$: $\alpha \in$ *open-segment* $\beta\ \gamma$

define *x* **where** $x \equiv a + \alpha *_R$ *?u*
define *y* **where** $y \equiv a + \beta *_R$ *?u*
define *z* **where** $z \equiv a + \gamma *_R$ *?u*

obtain *v* **where** *v*: $\alpha = (1 - v) * \beta + v * \gamma \wedge v \in \{0<..<1\}$
by (*metis* (*no-types*, *lifting*) $*$ *imageE in-segment*(*2*) *real-scaleR-def segment-image-interval*(*2*))
then have $x = a + ((1 - v) * \beta + v * \gamma) *_R$ *?u* **using** *x-def* **by** *blast*
also have $\ldots = a + (((1 - v) * \beta) *_R$ *?u*$) + ((v * \gamma) *_R$ *?u*$)$ **by** (*simp add*: *scaleR-left.add*)
also have $\ldots = a + ((1 - v) *_R (\beta *_R$ *?u*$)) + (v *_R (\gamma *_R$ *?u*$))$ **by** *simp*
also have $\ldots = a + ((1 - v) *_R (y - a)) + (v *_R (z - a))$ **by** (*simp add*: *y-def z-def*)
also have $\ldots = a + y - a - v *_R (y - a) + v *_R (z - a)$ **by** (*simp add*: *scaleR-left-diff-distrib*)
also have $\ldots = y - v *_R (y - a) + v *_R (z - a)$ **by** *simp*
also have $\ldots = y - (v *_R y) + (v *_R a) + (v *_R z) - (v *_R a)$ **by** (*simp add*: *scaleR-right-diff-distrib*)
also have $\ldots = (1 - v) *_R y + v *_R z$ **by** (*metis add-diff-cancel diff-add-eq scaleR-collapse*)
finally have $x = (1 - v) *_R y + v *_R z$ **.**
moreover have $0 \leq 1 - v \wedge 1 - v \leq 1$ **using** *v* **by** *fastforce*

ultimately have $x \in$ *closed-segment* y z **using** *in-segment*(*1*) **by** *auto*
moreover have $x \neq y \wedge x \neq z$
by (*metis* $*$ *add-diff-cancel-left' assms*(*2*) *eq-iff-diff-eq-0 in-open-segment-iff-line open-segment-commute open-segment-subsegment scaleR-right-imp-eq x-def y-def z-def*)
ultimately show $a + \alpha *_R$ *?u* $\in$ *open-segment* $(a + \beta *_R$ *?u*$)$ $(a + \gamma *_R$ *?u*$)$
unfolding *open-segment-def* **using** *x-def y-def z-def* **by** *force*
qed

obtain α β γ **where** *xyz*: $x = a + \alpha *_R$ *?u* $\wedge$ $y = a + \beta *_R$ *?u* $\wedge$ $z = a + \gamma *_R$ *?u*
using *affine-hull-2-alt*[*of a b*] *assms*(*1*) *assms*(*3*) **by** *auto*
then have $\alpha \neq \beta \wedge \beta \neq \gamma \wedge \alpha \neq \gamma$ **using** *assms* **by** *blast*
moreover have $\alpha \in$ *closed-segment* β γ $\vee$ $\beta \in$ *closed-segment* α γ $\vee$ $\gamma \in$ *closed-segment* α β
by (*metis atLeastAtMost-iff closed-segment-commute less-eq-real-def less-max-iff-disj linorder-not-less real-Icc-closed-segment*)
ultimately have $\alpha \in$ *open-segment* β γ $\vee$ $\beta \in$ *open-segment* α γ $\vee$ $\gamma \in$ *open-segment* α β
unfolding *open-segment-def* **by** *fast*
thus *?thesis* **using** $*$ *xyz* **by** *presburger*
qed

lemma *pocket-fill-line-int-aux3*:
fixes A :: (*real^2*) *set*
assumes *convex* $A \wedge$ *compact* A
assumes $v \neq 0$
assumes *closed-segment 0 w* $\subseteq$ *frontier* A (**is** *closed-segment ?a ?b* $\subseteq$ -)
assumes $w \cdot v = 0$
assumes $w \neq 0$
shows $(A \subseteq \{x.\ x \cdot v \leq 0\} \vee A \subseteq \{x.\ x \cdot v \geq 0\})$ (**is** $A \subseteq$ *?P1* $\vee$ $A \subseteq$ *?P2*)
proof−
have *frontiers*: *frontier ?P1* $=$ *frontier ?P2* $\wedge$ *frontier ?P1* $\subseteq$ *?P2* $\wedge$ *frontier ?P2* $\subseteq$ *?P1*
by (*smt* (*verit*, *ccfv-threshold*) *Collect-mono assms*(*2*) *frontier-halfspace-component-ge frontier-halfspace-le inner-commute subset-antisym*)
have *frontier*: *frontier ?P1* $= \{x.\ x \cdot v = 0\}$
by (*simp add*: *assms*(*2*) *frontier-halfspace-component-ge frontiers*)

have *?thesis* **if** *interior* $A \neq \{\}$
proof−
have *interior* $A \subseteq$ *?P1* $\vee$ *interior* $A \subseteq$ *?P2*
proof(*rule ccontr*)
assume ¬ (*interior* $A \subseteq$ *?P1* $\vee$ *interior* $A \subseteq$ *?P2*)
then obtain x y **where** *xy*: $x \in$ ((*interior* A) $\cap$ *?P1*) $-$ *?P2* $\wedge$ $y \in$ ((*interior* A) $\cap$ *?P2*) $-$ *?P1*
by *fastforce*
moreover have $x \in$ *frontier ?P1* $\cup$ *interior ?P1* $\wedge$ $y \in$ *frontier ?P2* $\cup$ *interior ?P2*
by (*metis DiffD1 IntD2 Un-Diff-cancel2 frontiers closure-Un-frontier fron-*

tier-def interior-subset sup.orderE xy)
ultimately have *xy'*: $x \in (\mathit{interior}\ A) \cap \mathit{interior}\ ?P1 \wedge y \in (\mathit{interior}\ A) \cap \mathit{interior}\ ?P2$
using *frontiers* **by** *blast*
then have $\mathit{closed\text{-}segment}\ x\ y \cap \mathit{frontier}\ ?P1 \neq \{\}$
by (*metis* (*no-types*, *lifting*) *DiffD1 DiffD2 Int-iff convex-closed-segment convex-imp-path-connected empty-iff ends-in-segment*(*1*) *ends-in-segment*(*2*) *in-mono path-connected-not-frontier-subset xy*)
moreover have $\mathit{closed\text{-}segment}\ x\ y \subseteq \mathit{interior}\ A$
by (*metis convex-interior Int-iff assms*(*1*) *convex-contains-segment xy'*)
ultimately obtain *z* **where** *z*: $z \in \mathit{interior}\ A \cap \mathit{frontier}\ ?P1$ **by** *blast*

have $\mathit{closed\text{-}segment}\ ?a\ ?b \subseteq \mathit{frontier}\ ?P1$
proof(*rule subsetI*)
fix *x*
assume $x \in \mathit{closed\text{-}segment}\ ?a\ ?b$
then obtain *u* **where** $x = (1 - u) *_R\ ?a + u *_R\ ?b \wedge 0 \leq u \wedge u \leq 1$
unfolding *closed-segment-def* **by** *blast*
then have $x \cdot v = u *_R (?b \cdot v)$ **by** *simp*
moreover have $?b \cdot v = 0$ **by** (*simp add*: *assms*(*4*))
ultimately have $x \cdot v = 0$ **by** *simp*
thus $x \in \mathit{frontier}\ ?P1$ **using** *frontier* **by** *blast*
qed
moreover have $z \notin \mathit{closed\text{-}segment}\ ?a\ ?b$ **using** *assms*(*3*) *frontier-def z* **by** *fastforce*
ultimately have $z \in \mathit{frontier}\ ?P1 - \mathit{closed\text{-}segment}\ ?a\ ?b$ **using** *z* **by** *blast*
moreover have $\mathit{collinear}\ \{z, ?a, ?b\}$
proof−
have $\{z, ?a, ?b\} \subseteq \{x.\ x \cdot v = 0\}$
using ‹$\{0--w\} \subseteq \mathit{frontier}\ \{x.\ x \cdot v \leq 0\}$› *frontier z* **by** *auto*
moreover have $\{x.\ x \cdot v = 0\} = \mathit{affine\ hull}\ \{?a, ?b\}$
by (*metis* (*no-types*, *lifting*) *Collect-mono assms*(*2*) *assms*(*5*) *calculation halfplane-frontier-affine-hull inner-commute insert-subset subset-antisym*)
ultimately show *?thesis* **using** *collinear-affine-hull* **by** *auto*
qed
ultimately have $?a \in \mathit{open\text{-}segment}\ z\ ?b \vee ?b \in \mathit{open\text{-}segment}\ z\ ?a$
using *three-points-on-line*[*of* $\{x.\ x \cdot v = 0\}$]
by (*smt* (*verit*) ‹$z \notin \{0--w\}$› *assms*(*5*) *collinear-3-imp-in-affine-hull ends-in-segment*(*1*) *ends-in-segment*(*2*) *hull-redundant hull-subset insert-commute open-closed-segment three-points-on-line*)
moreover have $\mathit{open\text{-}segment}\ z\ ?b \subseteq \mathit{interior}\ A \wedge \mathit{open\text{-}segment}\ z\ ?a \subseteq \mathit{interior}\ A$
proof−
have $\mathit{closed\text{-}segment}\ z\ ?b \subseteq A \wedge \mathit{closed\text{-}segment}\ z\ ?a \subseteq A$
by (*meson IntD1 assms*(*1*) *assms*(*3*) *closed-segment-subset ends-in-segment*(*1*) *ends-in-segment*(*2*) *frontier-subset-compact in-mono interior-subset z*)
then have $\mathit{rel\text{-}interior}\ (\mathit{closed\text{-}segment}\ z\ ?b) \subseteq \mathit{interior}\ A$
$\wedge\ \mathit{rel\text{-}interior}\ (\mathit{closed\text{-}segment}\ z\ ?a) \subseteq \mathit{interior}\ A$
by (*metis IntD1* ‹$z \notin \{0--w\}$› *assms*(*1*) *closure-convex-hull convex-hull-eq*

in-interior-closure-convex-segment order-class.order-eq-iff rel-interior-closed-segment subsetD subset-closed-segment z)
moreover have *rel-interior* (*closed-segment z ?b*) = *open-segment z ?b* ∧ *rel-interior* (*closed-segment z ?a*) = *open-segment z ?a*
by (*metis* ‹*z* ∉ {*0*−−*w*}› *closed-segment-commute ends-in-segment*(*1*) *rel-interior-closed-segment*)
ultimately show *?thesis* **by** *force*
qed
ultimately have *?a* ∈ *interior A* ∨ *?b* ∈ *interior A* **by** *fast*
thus *False* **using** *assms*(*3*) *frontier-def* **by** *auto*
qed
then have *closure* (*interior A*) ⊆ *closure ?P1* ∨ *closure* (*interior A*) ⊆ *closure ?P2*
using *closure-mono* **by** *blast*
moreover have *closed ?P1* ∧ *closed ?P2*
by (*simp add*: *closed-halfspace-component-ge closed-halfspace-component-le*)
moreover have *closure* (*interior A*) = *A*
using *assms*(*1*)
by (*simp add*: *compact-imp-closed convex-closure-interior that*)
ultimately show *?thesis* **using** *closure-closed* **by** *auto*
qed
moreover have *?thesis* **if** *interior A* = {}
proof(*rule ccontr*)
assume ¬ (*A* ⊆ *?P1* ∨ *A* ⊆ *?P2*)
then obtain *x y* **where** *xy*: *x* ∈ (*A* ∩ *?P1*) − *?P2* ∧ *y* ∈ (*A* ∩ *?P2*) − *?P1* **by** *fastforce*
moreover have *x* ∈ *frontier ?P1* ∪ *interior ?P1* ∧ *y* ∈ *frontier ?P2* ∪ *interior ?P2*
by (*metis DiffD1 IntD2 Un-Diff-cancel2 frontiers closure-Un-frontier frontier-def interior-subset sup.orderE xy*)
ultimately have *xy'*: *x* ∈ *A* ∩ *interior ?P1* ∧ *y* ∈ *A* ∩ *interior ?P2* **using** *frontiers* **by** *blast*
have ¬ *collinear* {*?a*, *?b*, *x*, *y*}
proof(*rule ccontr*)
assume ¬ ¬ *collinear* {*?a*, *?b*, *x*, *y*}
then have ∗: *collinear* {*?a*, *?b*, *x*, *y*} **by** *blast*
then have {*?a*, *?b*, *x*, *y*} ⊆ *affine hull* {*?a*, *?b*}
by (*metis assms*(*5*) *collinear-3-imp-in-affine-hull collinear-4-3 hull-subset insert-subset*)
moreover have *affine hull* {*?a*, *?b*} = {*x*. *x* ∙ *v* = *0*}
by (*smt* (*verit*) *DiffE* ∗ *assms*(*2*) *assms*(*4*) *assms*(*5*) *collinear-3-imp-in-affine-hull collinear-4-3 halfplane-frontier-affine-hull inner-commute mem-Collect-eq xy*)
moreover have ... = *frontier ?P1* ∧ ... = *frontier ?P2*
using *frontiers assms*(*2*) *frontier-halfspace-component-ge* **by** *blast*
ultimately show *False* **using** *frontiers xy* **by** *auto*
qed
then obtain *c1 c2 c3* **where** *c123*: ¬ *collinear* {*c1*, *c2*, *c3*} ∧ {*c1*, *c2*, *c3*} ⊆ {*?a*, *?b*, *x*, *y*}
by (*metis assms*(*5*) *collinear-4-3 insert-mono subset-insertI*)

then have *interior* (*convex hull* {*c1*, *c2*, *c3*}) ≠ {}
by (*metis Jordan-inside-outside-real2 closed-path-def make-triangle-def path-inside-def polygon-def polygon-of-def triangle-inside-is-convex-hull-interior triangle-is-polygon*)
moreover have {*c1*, *c2*, *c3*} ⊆ *A*
by (*smt* (*verit*, *del-insts*) *c123 xy' assms*(*1*) *assms*(*3*) *empty-subsetI frontier-subset-compact in-mono inf.orderE insert-absorb insert-mono le-infE subsetI subset-closed-segment*)
ultimately have *interior A* ≠ {}
by (*metis assms*(*1*) *interior-mono subset-empty subset-hull*)
thus *False* **using** *that* **by** *blast*
qed
ultimately show *?thesis* **by** *blast*
qed

lemma *pocket-fill-line-int-aux4*:
fixes *p q* :: *R-to-R2*
fixes *A* :: (*real^2*) *set*
defines *p0* ≡ *pathstart p*
defines *p1* ≡ *pathfinish p*
defines *q0* ≡ *pathstart q*
defines *q1* ≡ *pathfinish q*
defines *a* ≡ *p1$1*
defines *l* ≡ *closed-segment p0 p1*
assumes *simple-path p*
assumes *simple-path q*
assumes *path-image p* ∩ *path-image q* = {}
assumes *p0$1* = *0* ∧ *p0$2* = *0* ∧ *p1$2* = *0*
assumes *a* > *0*
assumes ∀ *v* ∈ *path-image p*. *q0$2* ≤ *v$2*
assumes ∀ *v* ∈ *path-image p*. *q1$2* > *v$2*
assumes *convex A* ∧ *compact A*
assumes {*p0*, *p1*} ⊆ *frontier A*
assumes *p'*{*0*<..<*1*} ⊆ *interior A*
assumes *path-image q* ⊆ *A*
shows *l* ⊆ *frontier A* ∀ *x* ∈ (*path-image p*) ∪ (*path-image q*). *x$2* ≥ *0 q0$2* = *0*
proof−
have *l*: *l* = {*x*. *x$2* = *0* ∧ *x$1* ∈ {*0*..*a*}}
using *horizontal-segment-at-0' a-def assms*(*10*) *assms*(*11*) *l-def* **by** *presburger*
have *endpoints*: (*p 0*)*$1* = *0* ∧ (*p 0*)*$2* = *0* ∧ (*p 1*)*$1* = *a* ∧ (*p 1*)*$2* = *0*
by (*metis a-def assms*(*10*) *p0-def p1-def pathfinish-def pathstart-def*)

have *l* ⊆ *frontier A* **if** ¬ (*path-image q* ∩ {*x*. *x$2* = *0*} ⊆ *l*)
proof−
from *that* **obtain** *x* **where** *x* ∈ *path-image q* ∩ {*x*. *x$2* = *0*} ∧ (*x$1* < *0* ∨ *x$1* > *a*)
by (*smt* (*verit*) *Int-Collect a-def assms*(*10*) *endpoints l-def p0-def pathstart-def segment-horizontal subsetI*)
thus *?thesis*
using *pocket-fill-line-int-aux*[*of p0 p1 x A*] **unfolding** *l-def*

by (*smt* (*verit*, *del-insts*) *IntD2 Int-commute a-def assms*(*11*) *assms*(*14*) *assms*(*15*) *assms*(*17*) *assms*(*10*) *endpoints exhaust-2 frontier-subset-compact insert-subset mem-Collect-eq p0-def pathstart-def subset-eq vec-eq-iff zero-index*)
qed
moreover have *False* **if** (*path-image q* ∩ {*x*. *x*$*2* = *0*} ⊆ *l*)
proof−
have (*path-image p* ∩ {*x*. *x*$*2* = *0*} ⊆ *l*)
using *pocket-fill-line-int-aux2*
by (*metis a-def assms*(*10*) *assms*(*11*) *assms*(*14*) *assms*(*15*) *assms*(*16*) *assms*(*7*) *l-def p0-def p1-def*)
then have *path-image p* ∩ *path-image q* ≠ {}
using *pocket-fill-line-int-aux1*
by (*metis* (*mono-tags*, *lifting*) *assms*(*11*) *assms*(*12*) *assms*(*13*) *assms*(*7*) *assms*(*8*) *endpoints l-def p0-def p1-def pathfinish-def pathstart-def q0-def q1-def that*)
thus *False* **by** (*simp add*: *assms*(*9*))
qed
ultimately show ∗: *l* ⊆ *frontier A* **by** *blast*

show ∀ *x* ∈ (*path-image p*) ∪ (*path-image q*). *x*$*2* ≥ *0*
proof(*rule ccontr*)
assume ¬ (∀ *x* ∈ (*path-image p*) ∪ (*path-image q*). *x*$*2* ≥ *0*)
then have ∃ *x* ∈ (*path-image p*) ∪ (*path-image q*). *x*$*2* < *0* **using** *linorder-not-le* **by** *blast*
then obtain *x* **where** *x*: *x* ∈ ((*path-image p*) ∪ (*path-image q*)) ∩ *A* ∧ *x*$*2* < *0*
using *assms*(*12*) *assms*(*17*) *pathstart-in-path-image q0-def* **by** *fastforce*

let *?v* = (*vector* [*0*, *1*])::(*real*^*2*)
have *1*: *?v* ≠ *0* **by** (*simp add*: *e1e2-basis*(*3*))
have *2*: *closed-segment 0 p1* ⊆ *frontier A*
by (*smt* (*verit*, *del-insts*) ∗ *Int-closed-segment closed-segment-eq doubleton-eq-iff endpoints l-def p0-def pathstart-def segment-vertical zero-index*)
have *3*: *p1* · *?v* = *0* **by** (*metis assms*(*10*) *cart-eq-inner-axis e1e2-basis*(*3*))
have *4*: *p1* ≠ *0* **using** *a-def assms*(*11*) **by** *force*
have ∗: (*A* ⊆ {*x*. *x* · *?v* ≤ *0*} ∨ *A* ⊆ {*x*. *x* · *?v* ≥ *0*})
using *pocket-fill-line-int-aux3*[*OF assms*(*14*) *1 2 3 4*] **by** *blast*
moreover have *q1*$*2* > *0* **using** *assms*(*10*) *assms*(*13*) *p0-def pathstart-in-path-image* **by** *fastforce*
ultimately show *False*
by (*metis* (*no-types*, *lifting*) *IntE x assms*(*17*) *e1e2-basis*(*3*) *inner-axis linorder-not-less mem-Collect-eq pathfinish-in-path-image q1-def real-inner-1-right subsetD*)
qed
moreover have *q0*$*2* ≤ *0* **using** *assms*(*10*) *assms*(*12*) *p1-def* **by** *force*
moreover have *q0* ∈ (*path-image p*) ∪ (*path-image q*)
by (*simp add*: *pathstart-in-path-image q0-def*)
ultimately show *q0*$*2* = *0* **by** *force*
qed

lemma *pocket-fill-line-int-aux5*:
fixes *p q* :: *R-to-R2*
fixes *A* :: (*real^2*) *set*
defines *p0* ≡ *pathstart p*
defines *p1* ≡ *pathfinish p*
defines *q0* ≡ *pathstart q*
defines *q1* ≡ *pathfinish q*
defines *a* ≡ *p1$1*
defines *l* ≡ *closed-segment p0 p1*
assumes *simple-path p*
assumes *simple-path q*
assumes *path-image p* ∩ *path-image q* = {*q0*, *q1*}
assumes *p0$1* = *0* ∧ *p0$2* = *0* ∧ *p1$2* = *0*
assumes *a* > *0*
assumes *A* = *convex hull* (*path-image p* ∪ *path-image q*)
assumes {*p0*, *p1*} ⊆ *frontier A*
assumes *p*‘{*0*<..<*1*} ⊆ *interior A*
assumes *path-image q* ⊆ *A*
assumes ∃ *x* ∈ *p*‘{*0*<..<*1*}. *x$2* ≥ *0*
assumes *q0* = *p1* ∧ *q1* = *p0*
shows *l* ⊆ *frontier A* ∀ *x* ∈ *path-image p* ∪ *path-image q*. *x$2* ≥ *0*
proof−
have *1*: *l* ⊆ *frontier A* **if** ∀ *x* ∈ *path-image p* ∪ *path-image q*. *x$2* ≥ *0*
proof−
have ∀ *x* ∈ *path-image p* ∪ *path-image q*. *x* · (*vector* [*0*, *1*]) ≥ *0*
by (*simp add*: *e1e2-basis*(*3*) *inner-axis that*)
then have ∀ *x* ∈ *A*. *x* · (*vector* [*0*, *1*]) ≥ *0*
by (*smt* (*verit*, *ccfv-threshold*) *convex-cut-aux'* *assms*(*12*) *inner-commute mem-Collect-eq subset-eq*)
then have *A* ⊆ {*x*. *x* · (*vector* [*0*, *1*]) ≥ *0*} **by** *blast*
moreover have *frontier* {*x*. *x* · ((*vector* [*0*, *1*])::(*real^2*)) ≥ *0*} = {*x*. *x* · (*vector* [*0*, *1*]) = *0*}
by (*metis dual-order.refl frontier-halfspace-component-ge not-one-le-zero vector-2*(*2*) *zero-index*)
moreover have *l* ⊆ {*x*. *x* · (*vector* [*0*, *1*]) = *0*}
proof−
have ∀ *x* ∈ *l*. *x$2* = *0* **using** *assms*(*10*) *l-def segment-horizontal* **by** *presburger*
thus *?thesis* **by** (*simp add*: *cart-eq-inner-axis e1e2-basis*(*3*) *subset-eq*)
qed
ultimately show *?thesis*
by (*smt* (*verit*, *best*) *Un-upper1 assms*(*12*) *closed-segment-subset convex-convex-hull hull-subset in-frontier-in-subset l-def p0-def p1-def pathfinish-in-path-image pathstart-in-path-image subset-eq*)
qed
have *2*: *False* **if** *tht*: ¬ (∀ *x* ∈ (*path-image p*) ∪ (*path-image q*). *x$2* ≥ *0*)
proof−
obtain *x tx* **where** *x*: *tx* ∈ {*0..1*} ∧ *q tx* = *x* ∧ (∀ *z* ∈ *path-image p*. *x$2* <

$z\$2$)
using *exists-point-below-all*[*of p q*] *that*
by (*smt* (*verit*, *del-insts*) *tht assms*(10) *assms*(12) *assms*(14) *assms*(7) *assms*(8) *image-iff p0-def p1-def path-image-def pathfinish-def pathstart-def simple-path-imp-path*)
obtain *y ty* **where** *y*: $ty \in \{0..1\} \wedge q\ ty = y \wedge (\forall x \in$ *path-image p*. $y\$2 > x\$2$)
using *exists-point-above-all*[*of p q*]
by (*smt* (*verit*, *del-insts*) *assms*(10) *assms*(12) *assms*(14) *assms*(16) *assms*(7) *assms*(8) *image-iff p0-def p1-def path-image-def pathfinish-def pathstart-def simple-path-imp-path*)

let *?Q* =
$\lambda q'$. *simple-path* $q' \wedge$ *path-image p* $\cap$ *path-image* $q' = \{\}$
$\wedge\ q'\ 0 = q\ tx \wedge q'\ 1 = q\ ty$
$\wedge$ *path-image* $q' \subseteq$ *path-image q*
have $*$: $\bigwedge q'$. *?Q* $q' \Longrightarrow$ *False*
proof−
fix q'
assume $*$: *?Q* q'

have *2*: *simple-path* q' **by** (*simp add*: $*$)
have *3*: *path-image p* $\cap$ *path-image* $q' = \{\}$ **by** (*simp add*: $*$)
have *6*: $\forall v \in$*path-image p*. *pathstart* $q'\ \$\ 2 \leq v\ \$\ 2$
by (*simp add*: $*$ *less-eq-real-def pathstart-def x*)
have *7*: $\forall v \in$*path-image p*. $v\ \$\ 2 <$ *pathfinish* $q'\ \$\ 2$ **by** (*simp add*: $*$ *pathfinish-def y*)
have *11*: *path-image* $q' \subseteq A$ **using** $*$ *assms*(15) **by** *blast*
have $\forall x \in$ (*path-image p*) $\cup$ (*path-image* q'). $x\$2 \geq 0$
using *pocket-fill-line-int-aux4*(2)[*of p*, *OF - 2 3 - - 6 7 - - - 11*]
by (*metis a-def assms*(10) *assms*(11) *assms*(12) *assms*(13) *assms*(14) *assms*(7) *assms*(8) *compact-Un compact-convex-hull compact-simple-path-image convex-convex-hull p0-def p1-def*)
thus *False*
by (*smt* (*verit*) $*$ *UnCI assms*(10) *p0-def pathstart-def pathstart-in-path-image x*)
qed

have *lf*: $(\forall t \in \{0..1\}.\ (q\ t = q0 \vee q\ t = q1) \longrightarrow (t = 0 \vee t = 1))$
using *assms*(8)
unfolding *q0-def q1-def simple-path-def loop-free-def pathstart-def pathfinish-def*
by *fastforce*
have *endpoints*: $q\ tx \neq q0 \wedge q\ ty \neq q0 \wedge q\ tx \neq q1 \wedge q\ ty \neq q1$
by (*metis x y assms*(10) *assms*(17) *order-less-le p0-def pathstart-in-path-image*)

have *tx-neq-ty*: $tx \neq ty$ **using** *pathstart-in-path-image x y* **by** *fastforce*
moreover have *False* **if** $tx < ty$
proof−

have *path-image p ∩ path-image (subpath tx ty q) = {}*
(is *path-image p ∩ path-image ?q' = {}*)
proof−
have *q0 ∉ path-image ?q' ∧ q1 ∉ path-image ?q'*
proof−
have *{tx..ty} ⊆ {0..1}* **using** *x y* **by** *simp*
then have *(∀ t ∈ {tx..ty}. (q t = q0 ∨ q t = q1) ⟶ (t = 0 ∨ t = 1))*
using *lf* **by** *blast*
moreover have *0 ∉ {tx..ty} ∧ 1 ∉ {tx..ty}*
by (*metis atLeastAtMost-iff dual-order.eq-iff endpoints pathfinish-def pathstart-def q0-def q1-def x y*)
moreover have *path-image ?q' = q‘{tx..ty}* **by** (*simp add*: *path-image-subpath that*)
ultimately show *?thesis* **by** *fastforce*
qed
thus *?thesis*
by (*smt* (*verit*, *best*) *Int-empty-right Int-insert-right-if0 assms(9) boolean-algebra-cancel.inf2 inf.absorb-iff1 path-image-subpath-subset x y*)
qed
thus *?thesis* **using** ∗[*of ?q'*]
by (*metis assms(8) tx-neq-ty path-image-subpath-subset pathfinish-def pathfinish-subpath pathstart-def pathstart-subpath simple-path-subpath x y*)
qed
moreover have *False* **if** *ty < tx*
proof−
have *path-image p ∩ path-image (reversepath (subpath tx ty q)) = {}*
(is *path-image p ∩ path-image ?q' = {}*)
proof−
have *q0 ∉ path-image ?q' ∧ q1 ∉ path-image ?q'*
proof−
have *{ty..tx} ⊆ {0..1}* **using** *x y* **by** *simp*
then have *(∀ t ∈ {ty..tx}. (q t = q0 ∨ q t = q1) ⟶ (t = 0 ∨ t = 1))*
using *lf* **by** *blast*
moreover have *0 ∉ {ty..tx} ∧ 1 ∉ {ty..tx}*
by (*metis atLeastAtMost-iff dual-order.eq-iff endpoints pathfinish-def pathstart-def q0-def q1-def x y*)
moreover have *path-image ?q' = q‘{ty..tx}*
by (*simp add*: *path-image-subpath reversepath-subpath that*)
ultimately show *?thesis* **by** *fastforce*
qed
thus *?thesis*
by (*smt* (*verit*) *Int-commute assms(9) inf.absorb-iff2 inf.assoc inf-bot-right insert-disjoint(2) path-image-reversepath path-image-subpath-subset x y*)
qed
thus *?thesis* **using** ∗[*of ?q'*]
by (*metis ∗ assms(8) tx-neq-ty path-image-subpath-commute path-image-subpath-subset pathfinish-def pathfinish-subpath pathstart-def pathstart-subpath reversepath-subpath simple-path-subpath x y*)
qed

```
    ultimately show False by fastforce
  qed
  show l ⊆ frontier A ∀ x ∈ (path-image p) ∪ (path-image q). x$2 ≥ 0
    using 1 2 apply blast
    using 1 2 by blast
qed

lemma pocket-fill-line-int-aux6:
  fixes p q :: R-to-R2
  defines p0 ≡ pathstart p
  defines p1 ≡ pathfinish p
  defines q0 ≡ pathstart q
  defines q1 ≡ pathfinish q
  defines a ≡ p1$1
  assumes simple-path p
  assumes simple-path q
  assumes p0 = 0 ∧ p1$2 = 0
  assumes a > 0
  assumes q0$1 ∈ {0..a} ∧ q0$2 = 0
  assumes ∀ x ∈ path-image p. q1$2 > x$2
  assumes ∀ x ∈ path-image p ∪ path-image q. x$2 ≥ 0
  shows path-image p ∩ path-image q ≠ {}
proof−
  let ?l1 = linepath p1 (vector [a, −1])
  let ?l2 = linepath ((vector [a, −1])::(real^2)) (vector [0, −1])
  let ?l3 = linepath ((vector [0, −1])::(real^2)) 0

  let ?R′ = ?l1 +++ ?l2 +++ ?l3
  let ?R = p +++ ?R′

  have closed: closed-path ?R
  proof−
    have path ?R using assms(6) p1-def simple-path-imp-path by auto
    moreover have pathstart ?R = pathstart p by simp
    moreover have pathfinish ?R = pathfinish ?l3 by simp
    moreover have pathstart p = 0 using assms(8) p0-def by fastforce
    moreover have pathfinish ?l3 = 0 by simp
    ultimately show ?thesis unfolding closed-path-def by presburger
  qed
  have simple: simple-path ?R
  proof−
    have arc ?R′
    proof−
      let ?a = p1
      let ?b = (vector [a, −1])::(real^2)
      let ?c = (vector [0, −1])::(real^2)
      let ?d = 0::(real^2)

      have arcs: arc ?l1 ∧ arc ?l2 ∧ arc ?l3
```

by (*metis arc-linepath assms*(8) *assms*(9) *vector-2*(1) *vector-2*(2) *verit-comp-simplify1*(1) *zero-index zero-neq-neg-one*)

have *l2l3*: *path-image ?l2* ∩ *path-image ?l3* = {*pathfinish ?l2*}
using *linepath-int-corner*[*of ?b ?c ?d*]
by (*metis Int-commute closed-segment-commute linepath-int-corner path-image-linepath pathfinish-linepath vector-2*(2) *zero-index zero-neq-neg-one*)
have *l1l2*: *path-image ?l1* ∩ *path-image ?l2* = {*pathfinish ?l1*}
using *linepath-int-corner*[*of ?a ?b ?c*] **by** (*simp add*: *assms*(8))
have *l1l3*: *path-image ?l1* ∩ *path-image ?l3* = {}
using *linepath-int-vertical*[*of ?a ?b ?c ?d*] *a-def assms*(9) *linepath-int-vertical*
by *auto*

have *path-image ?l2* ∩ *path-image ?l3* = {*pathfinish ?l2*}
using *l2l3* **by** *blast*
moreover have *sf-23*: *pathfinish ?l2* = *pathstart ?l3* **by** *simp*
ultimately have *arc* (*?l2* +++ *?l3*)
by (*metis arc-join-eq-alt arcs*)
moreover have *path-image ?l1* ∩ *path-image* (*?l2* +++ *?l3*) = {*pathfinish ?l1*}
using *l1l2 l1l3*
by (*metis* (*no-types*, *lifting*) *Int-Un-distrib sf-23 insert-is-Un path-image-join*)
moreover have *pathfinish ?l1* = *pathstart* (*?l2* +++ *?l3*) **by** *simp*
ultimately show *arc* (*?l1* +++ *?l2* +++ *?l3*)
by (*metis arc-join-eq-alt arcs*)
qed
moreover have *loop-free p* **using** *assms*(6) *simple-path-def* **by** *blast*
moreover have *path-image ?R'* ∩ *path-image p* = {*p0*, *p1*}
proof−
have *path-image ?l1* ∩ *path-image p* = {*p1*}
proof−
have ∀ *x* ∈ *path-image p*. *x*$2 ≥ *0* **by** (*simp add*: *assms*(12))
moreover have ∀ *x* ∈ *path-image ?l1*. *x*$2 ≤ *0* **using** *a-def assms*(8) *segment-vertical* **by** *force*
ultimately have ∀ *x* ∈ *path-image p* ∩ *path-image ?l1*. *x*$2 = *0* **by** *fastforce*
moreover have ∀ *x* ∈ *path-image ?l1*. *x*$2 = *0* ⟶ *x* = *p1*
by (*metis* (*mono-tags*, *opaque-lifting*) *a-def assms*(8) *exhaust-2 path-image-linepath segment-vertical vec-eq-iff vector-2*(1))
ultimately have ∀ *x* ∈ *path-image p* ∩ *path-image ?l1*. *x* = *p1* **by** *fast*
moreover have *p1* ∈ *path-image ?l1* ∧ *p1* ∈ *path-image p* **using** *p1-def*
by *auto*
ultimately show *?thesis* **by** *blast*
qed
moreover have *path-image ?l2* ∩ *path-image p* = {}
by (*smt* (*verit*, *best*) *segment-horizontal assms*(12) *UnCI disjoint-iff path-image-linepath vector-2*(2))
moreover have *path-image ?l3* ∩ *path-image p* = {*p0*}
proof−
have ∀ *x* ∈ *path-image p*. *x*$2 ≥ *0* **by** (*simp add*: *assms*(12))

moreover have $\forall x \in$ *path-image ?l3*. $x\$2 \leq 0$ **using** *a-def assms*(8) *segment-vertical* **by** *force*
ultimately have $\forall x \in$ *path-image p* $\cap$ *path-image ?l3*. $x\$2 = 0$ **by** *fastforce*
moreover have $\forall x \in$ *path-image ?l3*. $x\$2 = 0 \longrightarrow x =$ *p0*
by (*metis* (*no-types*, *opaque-lifting*) *assms*(8) *exhaust-2 path-image-linepath segment-vertical vec-eq-iff vector-2*(1) *zero-index*)
ultimately have $\forall x \in$ *path-image p* $\cap$ *path-image ?l3*. $x =$ *p0* **by** *fast*
moreover have *p0* $\in$ *path-image ?l3* $\wedge$ *p0* $\in$ *path-image p* **using** *assms*(8) *p0-def* **by** *fastforce*
ultimately show *?thesis* **by** *blast*
qed
ultimately show *?thesis*
by (*smt* (*verit*, *del-insts*) *Int-Un-distrib Int-commute Un-assoc Un-insert-right insert-is-Un path-image-join pathfinish-linepath pathstart-join pathstart-linepath*)
qed
moreover have *arc p*
using *closed-path-def arc-distinct-ends assms*(6) *calculation*(1) *closed p1-def simple-path-imp-arc*
by *force*
ultimately show *?thesis*
by (*metis* (*no-types*, *opaque-lifting*) *Int-commute closed-path-def closed dual-order.refl linepath-0' p0-def p1-def pathfinish-join pathstart-def pathstart-join simple-path-join-loop-eq*)
qed

have *inside-outside*: *inside-outside ?R* (*path-inside ?R*) (*path-outside ?R*)
using *closed simple Jordan-inside-outside-real2*
by (*simp add*: *closed-path-def inside-outside-def path-inside-def path-outside-def*)

have *interior-frontier*: *path-inside ?R* = *interior* (*path-inside ?R*)
$\wedge$ *frontier* (*path-inside ?R*) = *path-image ?R*
using *inside-outside interior-open* **unfolding** *inside-outside-def* **by** *auto*

have *R-y-q1*: $\forall x \in$ *path-image ?R*. $x\$2 <$ *q1*$\$2$
proof−
have ∗: $\forall x \in$ *path-image p*. $x\$2 <$ *q1*$\$2$ **using** *assms*(11) **by** *blast*
moreover have $\forall x \in$ *path-image ?l1*. $x\$2 <$ *q1*$\$2$
using *a-def assms*(8) ∗ *p1-def pathfinish-in-path-image segment-vertical* **by** *fastforce*
moreover have $\forall x \in$ *path-image ?l2*. $x\$2 <$ *q1*$\$2$
using *assms*(8) ∗ *p1-def pathfinish-in-path-image segment-horizontal* **by** *fastforce*
moreover have $\forall x \in$ *path-image ?l3*. $x\$2 <$ *q1*$\$2$
using *assms*(8) ∗ *p1-def pathfinish-in-path-image segment-vertical* **by** *fastforce*
ultimately show *?thesis* **by** (*metis not-in-path-image-join*)
qed
have *R-y-0*: $\forall x \in$ *path-image ?R*. $x\$2 \geq -1$
proof−
have $\forall x \in$ *path-image ?l1*. $x\$2 \geq -1$ **using** *a-def assms*(8) *segment-vertical* **by** *fastforce*

moreover have $\forall x \in$ *path-image ?l2*. $x\$2 \geq -1$ **using** *segment-horizontal* **by** *auto*
moreover have $\forall x \in$ *path-image ?l3*. $x\$2 \geq -1$ **using** *segment-vertical* **by** *auto*
moreover have $\forall x \in$ *path-image p*. $x\$2 \geq -1$ **using** *assms*(*12*) **by** *force*
ultimately show *?thesis* **by** (*metis not-in-path-image-join*)
qed

have *?thesis* **if** *p0* $\in$ *path-image q* $\vee$ *p1* $\in$ *path-image q* **using** *p0-def p1-def that* **by** *blast*
moreover have *?thesis* **if** *p0* $\notin$ *path-image q* $\wedge$ *p1* $\notin$ *path-image q* $\wedge$ *q0* $\notin$ *path-image p*
proof−
have *q-int-l1*: *path-image q* $\cap$ *path-image ?l1* $= \{\}$
proof−
have $\forall x \in$ *path-image q*. $x\$2 \geq 0$ **by** (*simp add*: *assms*(*12*))
moreover have $\forall x \in$ *path-image ?l1*. $x\$2 = 0 \longrightarrow x =$ *p1*
by (*metis* (*mono-tags*, *opaque-lifting*) *a-def assms*(*8*) *exhaust-2 path-image-linepath segment-vertical vec-eq-iff vector-2*(*1*))
ultimately show *?thesis* **using** *that a-def assms*(*8*) *segment-vertical* **by** *fastforce*
qed
moreover have *q-int-l2*: *path-image q* $\cap$ *path-image ?l2* $= \{\}$
by (*smt* (*verit*, *ccfv-threshold*) *UnCI assms*(*12*) *disjoint-iff path-image-linepath segment-horizontal vector-2*(*2*))
moreover have *q-int-l3*: *path-image q* $\cap$ *path-image ?l3* $= \{\}$
proof−
have $\forall x \in$ *path-image q*. $x\$2 \geq 0$ **by** (*simp add*: *assms*(*12*))
moreover have $\forall x \in$ *path-image ?l3*. $x\$2 = 0 \longrightarrow x =$ *p0*
by (*metis* (*no-types*, *opaque-lifting*) *assms*(*8*) *exhaust-2 path-image-linepath segment-vertical vec-eq-iff vector-2*(*1*) *zero-index*)
ultimately show *?thesis* **using** *that a-def assms*(*8*) *segment-vertical* **by** *fastforce*
qed
ultimately have *q0-notin-R*: *q0* $\notin$ *path-image ?R*
using *that* **by** (*simp add*: *disjoint-iff not-in-path-image-join pathstart-in-path-image q0-def*)

have *path-image q* $\cap$ *path-image ?R* $\neq \{\}$
proof−
have *q0* $\in$ *path-inside ?R*
proof−
let *?e* = (*vector* [*q0*$\$1$, -1])::(*real*^*2*)
let *?d1* = (*vector* [*a*, -1])::(*real*^*2*)
let *?d2* = (*vector* [*0*, -1])::(*real*^*2*)

have $0 <$ *q0*$\$1 \wedge$ *q0*$\$1 <$ *a*
by (*smt* (*verit*) *a-def assms*(*10*) *assms*(*8*) *atLeastAtMost-iff exhaust-2 linorder-not-less pathstart-in-path-image q0-def that vec-eq-iff zero-index*)

then have *q0$1 > 0 ∧ a − q0$1 > 0* **by** *simp*
then have *min (min (q0$1) (a − q0$1)) 1 > 0* (**is** *?ε′ > 0*) **by** *linarith*
then have *0 < ?ε′/2 ∧ ?ε′/2 < 1 ∧ ?ε′/2 < q0$1 ∧ ?ε′/2 < a − q0$1* **by** *argo*
then obtain *ε* **where** *ε*: *0 < ε ∧ ε < 1 ∧ ε < q0$1 ∧ ε < a − q0$1* **by** *blast*
moreover have *?e ∈ frontier (path-inside ?R)*
by (*smt* (*verit*, *del-insts*) *UnCI ‹0 < q0 $ 1 ∧ 0 < a − q0 $ 1› interior-frontier p1-def path-image-join path-image-linepath pathfinish-linepath pathstart-join pathstart-linepath segment-horizontal vector-2(1) vector-2(2)*)
ultimately obtain *int-p* **where** *int-p*: *int-p ∈ ball ?e ε ∩ path-inside ?R*
by (*meson inside-outside frontier-straddle mem-ball IntI*)

have *int-p-x*: *int-p$1 > 0 ∧ int-p$1 < a*
proof−
have *int-p$1 > 0*
proof(*rule ccontr*)
assume ¬ *int-p$1 > 0*
moreover have *dist (int-p$1) (q0$1) < q0$1*
by (*smt* (*verit*) *IntE ε dist-commute dist-vec-nth-le int-p mem-ball vector-2(1)*)
ultimately show *False* **using** *dist-real-def* **by** *force*
qed
moreover have *int-p$1 < a*
proof(*rule ccontr*)
assume ¬ *int-p$1 < a*
moreover have *dist (int-p$1) (q0$1) < a − q0$1*
by (*smt* (*verit*) *IntE ε dist-commute dist-vec-nth-le int-p mem-ball vector-2(1)*)
ultimately show *False* **using** *dist-real-def* **by** *force*
qed
ultimately show *?thesis* **by** *blast*
qed
have *int-p-y*: *int-p$2 > −1 ∧ int-p$2 < 0*
proof−
have *int-p$2 > −1*
proof(*rule ccontr*)
assume ∗: ¬ *int-p$2 > −1*
then have *int-p$2 ≤ −1* **by** *simp*
let *?e2′ = (vector [0, −1])::(real^2)*
let *?ray = λd. int-p + d ∗R ?e2′*
have ¬ (∃ *d>0. ?ray d ∈ path-image ?R*)
proof−
have ∀ *d>0. (?ray d)$2 < −1* **using** ∗ **by** *auto*
thus *?thesis* **using** *R-y-0* **by** *force*
qed
moreover have *bounded (path-inside ?R)* **using** *bounded-finite-inside simple* **by** *blast*
moreover have *?e2′ ≠ 0* **by** (*metis vector-2(2) zero-index zero-neq-neg-one*)

ultimately have *int-p* $\notin$ *path-inside ?R*
using *ray-to-frontier*[*of path-inside ?R*] *interior-frontier* **by** *metis*
thus *False* **using** *int-p* **by** *blast*
qed
moreover have *int-p*$*2* < *0*
proof(*rule ccontr*)
assume ¬ *int-p*$*2* < *0*
then have *dist int-p ?e* ≥ *1*
by (*smt* (*verit*, *del-insts*) *dist-real-def dist-vec-nth-le vector-2*(*2*))
thus *False* **by** (*smt* (*verit*, *del-insts*) *IntD1* ε *dist-commute int-p mem-ball*)
qed
ultimately show *?thesis* **by** *blast*
qed

let *?int-l* = *linepath int-p q0*

have *path-image ?int-l* ∩ *path-image ?l1* = {}
using ‹*0* < *q0* $ *1* ∧ *q0* $ *1* < *a*› *a-def int-p-x linepath-int-columns* **by** *auto*
moreover have *path-image ?int-l* ∩ *path-image ?l2* = {}
by (*smt* (*verit*, *best*) *assms*(*10*) *disjoint-iff int-p-y linepath-int-rows vector-2*(*2*))
moreover have *path-image ?int-l* ∩ *path-image ?l3* = {}
by (*smt* (*verit*, *del-insts*) ε *disjoint-iff int-p-x linepath-int-columns vector-2*(*1*) *zero-index*)
moreover have *path-image ?int-l* ∩ *path-image p* = {}
proof−
have ∀ *t* ∈ {*0*..*1*}. (*?int-l t*)$*2* = *0* ⟶ *t* = *1*
unfolding *linepath-def* **using** *assms*(*10*) *int-p-y* **by** *force*
then have ∀ *x* ∈ *path-image ?int-l*. *x*$*2* = *0* ⟶ *x* = *q0*
unfolding *path-image-def* **using** *linepath-1′* **by** *fastforce*
moreover have ∀ *x* ∈ *path-image p*. *x*$*2* ≥ *0* **by** (*simp add*: *assms*(*12*))
moreover have ∀ *x* ∈ *path-image ?int-l*. *x*$*2* ≤ *0*
by (*smt* (*verit*) *assms*(*10*) *int-p-y linepath-bound-2*(*2*))
ultimately show *?thesis* **using** *that* **by** *fastforce*
qed
ultimately have *path-image ?int-l* ∩ *path-image ?R* = {}
by (*simp add*: *disjoint-iff not-in-path-image-join*)

then have *path-image ?int-l* ⊆ *path-inside ?R* ∨ *path-image ?int-l* ⊆ *path-outside ?R*
by (*metis IntD2 IntI convex-imp-path-connected convex-segment*(*1*) *empty-iff int-p interior-frontier path-connected-not-frontier-subset path-image-linepath path-start-in-path-image pathstart-linepath*)
moreover have *?int-l 0* = *int-p* ∧ *int-p* ∈ *path-inside ?R*
using *int-p* **by** (*simp add*: *linepath-0′*)
ultimately have *path-image ?int-l* ⊆ *path-inside ?R*
using *inside-outside-def local.inside-outside* **by** *auto*
thus *?thesis* **by** *auto*

qed
then have *q0* ∈ − (*path-outside ?R*)
by (*metis ComplI IntI equals0D inside-Int-outside path-inside-def path-outside-def*)
moreover have *q1* ∈ *path-outside ?R*
proof−
let *?e2* = (*vector* [*0*, *1*])::(*real*^*2*)
let *?ray* = λ*d*. *q1* + *d* ∗$_R$ *?e2*
have ¬ (∃ *d*>*0*. *?ray d* ∈ *path-image ?R*)
proof−
have ∀ *d*>*0*. (*?ray d*)$*2* > *q1*$*2* **by** *simp*
thus *?thesis* **using** *R-y-q1* **by** *fastforce*
qed
moreover have *bounded* (*path-inside ?R*) **using** *bounded-finite-inside simple*
by *blast*
moreover have *?e2* ≠ *0* **using** *e1e2-basis*(*4*) **by** *force*
ultimately have *q1* ∉ *path-inside ?R*
using *ray-to-frontier*[*of path-inside ?R*] *interior-frontier* **by** *metis*
moreover have *q1* ∉ *path-image ?R* **using** *R-y-q1* **by** *blast*
ultimately show *?thesis* **using** *inside-outside* **unfolding** *inside-outside-def*
by *blast*
qed
ultimately have *path-image q* ∩ − (*path-outside ?R*) ≠ {}
∧ *path-image q* ∩ (*path-outside ?R*) ≠ {}
using *q0-def q1-def* **by** *blast*
moreover have *path-connected* (*path-image q*)
using *assms*(*7*) *path-connected-path-image simple-path-def* **by** *blast*
moreover have *path-image ?R* = *frontier* (*path-outside ?R*)
using *inside-outside* **unfolding** *inside-outside-def p0-def path-inside-def* **by**
blast
ultimately show *?thesis* **by** (*metis Diff-eq Diff-eq-empty-iff path-connected-not-frontier-subset*)
qed
thus *?thesis* **by** (*meson q-int-l1 q-int-l2 q-int-l3 disjoint-iff not-in-path-image-join*)
qed
ultimately show *?thesis* **using** *q0-def* **by** *blast*
qed

lemma *pocket-fill-line-int-aux7*:
fixes *p q* :: *R-to-R2*
fixes *A* :: (*real*^*2*) *set*
defines *p0* ≡ *pathstart p*
defines *p1* ≡ *pathfinish p*
defines *q0* ≡ *pathstart q*
defines *q1* ≡ *pathfinish q*
defines *a* ≡ *p1*$*1*
defines *l* ≡ *open-segment p0 p1*
assumes *simple-path p*
assumes *simple-path q*
assumes *path-image p* ∩ *path-image q* = {*q0*, *q1*}
assumes *p0*$*1* = *0* ∧ *p0*$*2* = *0* ∧ *p1*$*2* = *0*

assumes $a > 0$
assumes $A = convex\ hull\ (path\text{-}image\ p \cup path\text{-}image\ q)$
assumes $\{p0, p1\} \subseteq frontier\ A$
assumes $p`\{0<..<1\} \subseteq interior\ A$
assumes $\exists x \in p`\{0<..<1\}.\ x\$2 \geq 0$
assumes $q0 = p1 \wedge q1 = p0$
shows $path\text{-}image\ q \cap l = \{\}\ closed\text{-}segment\ p0\ p1 \subseteq frontier\ A$
proof−
have *1*: $path\text{-}image\ p \cap path\text{-}image\ q = \{pathstart\ q, pathfinish\ q\}$
by (*simp add*: *assms*(*9*) *q0-def q1-def*)
have *2*: $pathstart\ p\ \$\ 1 = 0 \wedge pathstart\ p\ \$\ 2 = 0 \wedge pathfinish\ p\ \$\ 2 = 0$
using *assms*(*10*) *p0-def p1-def* **by** *blast*
have *3*: $0 < pathfinish\ p\ \$\ 1$ **using** *a-def assms*(*11*) *p1-def* **by** *auto*
have *4*: $A = convex\ hull\ (path\text{-}image\ p \cup path\text{-}image\ q)$ **by** (*simp add*: *assms*(*12*))
have *5*: $\{pathstart\ p, pathfinish\ p\} \subseteq frontier\ A$ **using** *assms*(*13*) *p0-def p1-def*
by *blast*
have *6*: $p\ `\ \{0<..<1\} \subseteq interior\ A$ **using** *assms*(*14*) **by** *blast*
have *7*: $path\text{-}image\ q \subseteq A$ **using** *assms*(*12*) *hull-subset* **by** *force*
have *8*: $\exists x \in p`\{0<..<1\}.\ x\$2 \geq 0$ **using** *assms*(*15*) **by** *blast*
have *9*: $pathstart\ q = pathfinish\ p \wedge pathfinish\ q = pathstart\ p$
using *assms*(*16*) *p0-def p1-def q0-def q1-def* **by** *fastforce*
have $*$: $\forall x \in (path\text{-}image\ p) \cup (path\text{-}image\ q).\ x\$2 \geq 0$
using *pocket-fill-line-int-aux5*(*2*)[*OF assms*(*7*) *assms*(*8*) *1 2 3 4 5 6 7 8 9*] **by**
blast

show $closed\text{-}segment\ p0\ p1 \subseteq frontier\ A$
using *pocket-fill-line-int-aux5*(*1*)[*OF assms*(*7*) *assms*(*8*) *1 2 3 4 5 6 7 8 9*]
unfolding *l-def p0-def p1-def* **by** *blast*
show $path\text{-}image\ q \cap l = \{\}$
proof(*rule ccontr*)
assume $\neg\ path\text{-}image\ q \cap l = \{\}$
then obtain *x tx* **where** *x*: $tx \in \{0..1\} \wedge q\ tx = x \wedge x \in l$
by (*metis* (*no-types*, *lifting*) *disjoint-iff imageE path-image-def*)
obtain *y ty* **where** *y*: $ty \in \{0..1\} \wedge q\ ty = y \wedge (\forall x \in path\text{-}image\ p.\ y\$2 > x\$2)$
using *exists-point-above-all*[*of p q*]
by (*smt* (*verit*, *del-insts*) *4 6 8 assms*(*10*) *assms*(*7*) *assms*(*8*) *p0-def p1-def pathfinish-def pathstart-def simple-path-def image-iff path-image-def*)

have *lf*: $(\forall t \in \{0..1\}.\ (q\ t = q0 \vee q\ t = q1) \longrightarrow (t = 0 \vee t = 1))$
using *assms*(*8*)
unfolding *q0-def q1-def simple-path-def loop-free-def pathstart-def pathfinish-def*
by *fastforce*
have *endpoints*: $q\ tx \neq q0 \wedge q\ ty \neq q0 \wedge q\ tx \neq q1 \wedge q\ ty \neq q1 \wedge tx \neq ty$
proof−
have $(q\ ty)\$2 > 0$ **by** (*metis assms*(*10*) *p0-def pathstart-in-path-image y*)
moreover have $(q\ tx)\$2 = 0$
proof−

```
      have q tx ∈ closed-segment q0 q1
        using assms(16) l-def open-closed-segment open-segment-commute x by
blast
      thus ?thesis by (simp add: assms(10) assms(16) segment-horizontal)
    qed
    moreover have q0 ∉ open-segment q0 q1 ∧ q1 ∉ open-segment q0 q1
      by (simp add: open-segment-def)
    ultimately show ?thesis
      using assms(10) assms(16) l-def open-segment-commute x by auto
  qed

  let ?Q =
    λq'. simple-path q' ∧ path-image p ∩ path-image q' = {}
      ∧ q' 0 = q tx ∧ q' 1 = q ty
      ∧ path-image q' ⊆ path-image q
  have **: ⋀q'. ?Q q' ⟹ False
  proof−
    fix q'
    assume **: ?Q q'
    have 1: simple-path q' by (simp add: **)
    have 2: pathstart p = 0 ∧ pathfinish p $ 2 = 0
      by (metis (mono-tags, lifting) assms(10) exhaust-2 p0-def p1-def vec-eq-iff
zero-index)
    have 3: 0 < pathfinish p $ 1 using a-def assms(11) p1-def by blast
    have 4: pathstart q' $ 1 ∈ {0..pathfinish p $ 1} ∧ pathstart q' $ 2 = 0
    proof−
      have q' 0 ∈ closed-segment p0 p1 using ** l-def open-closed-segment x by
auto
      thus ?thesis
        by (smt (verit) 2 a-def assms(11) atLeastAtMost-iff atLeastatMost-empty
p0-def p1-def pathstart-def pathstart-subpath segment-horizontal zero-index)
    qed
    have 5: ∀x∈path-image p. x $ 2 < pathfinish q' $ 2 by (simp add: **
pathfinish-def y)
    have 6: ∀x∈path-image p ∪ path-image q'. 0 ≤ x $ 2 using * ** by blast
    have path-image p ∩ path-image q' ≠ {}
      using pocket-fill-line-int-aux6[OF assms(7) 1 2 3 4 5 6] by simp
    thus False using ** by blast
  qed

  have False if tx < ty
  proof−
    let ?q' = subpath tx ty q
    have q0 ∉ path-image ?q' ∧ q1 ∉ path-image ?q'
    proof−
      have {tx..ty} ⊆ {0..1} using x y by simp
      then have (∀t ∈ {tx..ty}. (q t = q0 ∨ q t = q1) ⟶ (t = 0 ∨ t = 1))
using lf by blast
      moreover have 0 ∉ {tx..ty} ∧ 1 ∉ {tx..ty}
```

by (*metis atLeastAtMost-iff dual-order.eq-iff endpoints pathfinish-def pathstart-def q0-def q1-def x y*)
moreover have *path-image ?q' = q'{tx..ty}* **by** (*simp add*: *path-image-subpath that*)
ultimately show *?thesis* **by** *fastforce*
qed
then have *?Q ?q'*
by (*smt* (*verit*, *best*) *assms*(*8*) *assms*(*9*) *disjoint-insert*(*1*) *endpoints inf.absorb-iff1 inf-bot-right inf-left-commute path-image-subpath-subset pathfinish-def pathfinish-subpath pathstart-def pathstart-subpath simple-path-subpath x y*)
thus *False* **using** ∗∗ **by** *auto*
qed
moreover have *False* **if** *tx* > *ty*
proof−
let *?q'* = *reversepath* (*subpath ty tx q*)
have *q0* ∉ *path-image ?q'* ∧ *q1* ∉ *path-image ?q'*
proof−
have {*ty..tx*} ⊆ {*0..1*} **using** *x y* **by** *simp*
then have (∀ *t* ∈ {*ty..tx*}. (*q t* = *q0* ∨ *q t* = *q1*) ⟶ (*t* = *0* ∨ *t* = *1*))
using *lf* **by** *blast*
moreover have *0* ∉ {*ty..tx*} ∧ *1* ∉ {*ty..tx*}
by (*metis atLeastAtMost-iff dual-order.eq-iff endpoints pathfinish-def pathstart-def q0-def q1-def x y*)
moreover have *path-image ?q'* = *q'*{*ty..tx*} **by** (*simp add*: *path-image-subpath that*)
ultimately show *?thesis* **by** *fastforce*
qed
then have *?Q ?q'*
by (*smt* (*verit*) *assms*(*8*) *assms*(*9*) *endpoints inf.absorb-iff2 inf.assoc inf-bot-left insert-disjoint*(*2*) *path-image-subpath-subset pathstart-def pathstart-subpath reversepath-def reversepath-subpath simple-path-subpath x y*)
thus *False* **using** ∗∗ **by** *blast*
qed
ultimately show *False* **using** *endpoints* **by** *linarith*
qed
qed

lemma *frontier-injective-linear-image*:
fixes *f* :: *'a*::*euclidean-space* ⇒ *'a*::*euclidean-space*
assumes *linear f inj f*
shows *f* ' (*frontier S*) = *frontier* (*f* ' *S*)
using *interior-injective-linear-image closure-injective-linear-image frontier-def assms*
by (*metis image-set-diff*)

lemma *pocket-fill-line-int-aux8*:
fixes *p q* :: *R-to-R2*
fixes *A* :: (*real^2*) *set*

defines *p0* ≡ *pathstart p*
defines *p1* ≡ *pathfinish p*
defines *q0* ≡ *pathstart q*
defines *q1* ≡ *pathfinish q*
defines *a* ≡ *p1$1*
defines *l* ≡ *open-segment p0 p1*
assumes *simple-path p*
assumes *simple-path q*
assumes *path-image p* ∩ *path-image q* = {*q0*, *q1*}
assumes *p0$1* = *0* ∧ *p0$2* = *0* ∧ *p1$2* = *0*
assumes *a* > *0*
assumes *A* = *convex hull* (*path-image p* ∪ *path-image q*)
assumes {*p0*, *p1*} ⊆ *frontier A*
assumes *p*‘{*0*<..<*1*} ⊆ *interior A*
assumes *q0* = *p1* ∧ *q1* = *p0*
shows *path-image q* ∩ *l* = {} ∧ *l* ⊆ *frontier A*
proof−
have *?thesis* **if** *ex*: ∃ *x* ∈ *p*‘{*0*<..<*1*}. *x$2* ≥ *0*
using *ex a-def assms dual-order.trans l-def p0-def p1-def pocket-fill-line-int-aux7*(*1*) *pocket-fill-line-int-aux7*(*2*) *q0-def q1-def segment-open-subset-closed that*

by (*smt* (*verit*) *a-def assms dual-order.trans l-def p0-def p1-def pocket-fill-line-int-aux7*(*1*) *pocket-fill-line-int-aux7*(*2*) *q0-def q1-def segment-open-subset-closed that*)
moreover have *?thesis* **if** ¬ (∃ *x* ∈ *p*‘{*0*<..<*1*}. *x$2* ≥ *0*)
proof−
let *?M* = (*vector* [*vector* [*1*, *0*], *vector* [*0*, −*1*]])::(*real*^*2*^*2*)
let *?f* = λ*v*. *?M* ∗*v v*
let *?g* = (λ*v*. *vector* [*v$1*, −*v$2*])::(*real*^*2* ⇒ *real*^*2*)
define *p*′ **where** *p*′ ≡ *?f* ∘ *p*
define *q*′ **where** *q*′ ≡ *?f* ∘ *q*
define *A*′ **where** *A*′ ≡ *?f*‘*A*

have *inj*: *inj ?f* **and** *f-eq-g*: *?f* = *?g*
using *flip-function*(*1*) **apply** *blast*
using *flip-function*(*2*) **by** *blast*

have *4*: *pathstart p*′ $ *1* = *0* ∧ *pathstart p*′ $ *2* = *0* ∧ *pathfinish p*′ $ *2* = *0*
by (*smt* (*verit*, *best*) *assms*(*10*) *f-eq-g o-apply p*′*-def p0-def p1-def pathfinish-def pathstart-def vector-2*(*1*) *vector-2*(*2*))
have *startfinish*: *pathstart p*′ = *pathstart p* ∧ *pathfinish p*′ = *pathfinish p*
by (*metis* (*mono-tags*, *opaque-lifting*) *4 assms*(*10*) *exhaust-2 f-eq-g o-apply p*′*-def p0-def p1-def pathfinish-def vec-eq-iff vector-2*(*1*))

have *1*: *simple-path p*′ **using** *inj* **by** (*simp add*: *assms*(*7*) *simple-path-linear-image-eq p*′*-def*)
have *2*: *simple-path q*′ **using** *inj* **by** (*simp add*: *assms*(*8*) *simple-path-linear-image-eq q*′*-def*)
have *3*: *path-image p*′ ∩ *path-image q*′ = {*pathstart q*′, *pathfinish q*′}
proof−

have *path-image p′ ∩ path-image q′ = ?f‘(path-image p ∩ path-image q)*
unfolding *p′-def q′-def* **by** (*simp add*: *image-Int inj path-image-compose*)
also have *... = ?f‘{q0, q1}* **using** *assms(9)* **by** *presburger*
finally show *?thesis*
by (*simp add*: *startfinish pathfinish-compose pathstart-compose q′-def q0-def q1-def*)
qed
have *5*: *0 < pathfinish p′ $ 1*
by (*metis* (*mono-tags*, *lifting*) *a-def assms(11) f-eq-g o-apply p′-def p1-def pathfinish-def vector-2(1)*)
have *6*: *A′ = convex hull* (*path-image p′ ∪ path-image q′*)
proof−
have *path-image* (*?f ∘ p*) = *?f‘*(*path-image p*) **using** *path-image-compose* **by** *blast*
moreover have *path-image* (*?f ∘ q*) = *?f‘*(*path-image q*) **using** *path-image-compose* **by** *blast*
moreover have *?f‘*(*path-image p ∪ path-image q*) = *?f‘*(*path-image p*) ∪ *?f‘*(*path-image q*)
by *blast*
moreover have *A′ = convex hull* (*?f‘*(*path-image p ∪ path-image q*))
by (*simp add*: *assms(12) convex-hull-linear-image A′-def*)
ultimately show *?thesis* **using** *p′-def q′-def A′-def* **by** *argo*
qed
have *7*: {*pathstart p′*, *pathfinish p′*} ⊆ *frontier A′*
using *frontier-injective-linear-image*
by (*smt* (*verit*, *best*) *3 A′-def assms(13) assms(15) assms(9) doubleton-eq-iff image-Int inj inj-image-subset-iff matrix-vector-mul-linear p′-def p0-def p1-def path-image-linear-image pathfinish-compose pathstart-compose q′-def q0-def q1-def*)
have *8*: *p′‘{0<..<1}* ⊆ *interior A′*
proof−
have *?f‘*(*interior A*) = *interior A′* **by** (*simp add*: *A′-def inj interior-injective-linear-image*)
thus *?thesis* **using** *assms(14) p′-def* **by** *auto*
qed
have *9*: ∃ *x* ∈ *p′‘{0<..<1}*. *x$2 ≥ 0*
proof−
have ∃ *x* ∈ *p‘{0<..<1}*. *x$2 < 0*
by (*metis that all-not-in-conv bot.extremum greaterThanLessThan-subseteq-greaterThanLessThan image-is-empty verit-comp-simplify1(3) zero-less-one*)
then obtain *x* **where** *x* ∈ *p‘{0<..<1}* ∧ *x$2 < 0* **by** *presburger*
moreover then have (*?g x*)*$2 > 0* **by** *fastforce*
ultimately show *?thesis* **by** (*smt* (*verit*, *ccfv-threshold*) *f-eq-g image-iff o-apply p′-def*)
qed
have *10*: *pathstart q′ = pathfinish p′ ∧ pathfinish q′ = pathstart p′*
by (*metis* (*mono-tags*, *lifting*) *assms(15) o-apply p′-def p0-def p1-def pathfinish-def pathstart-def q′-def q0-def q1-def*)

have *path-image q′ ∩ open-segment* (*pathstart p′*) (*pathfinish p′*) = {}
using *pocket-fill-line-int-aux7(1)*[*OF 1 2 3 4 5 6 7 8 9 10*] **by** *blast*

then have *path-image* $q' \cap l = \{\}$ **using** *startfinish* **unfolding** *l-def p0-def p1-def* **by** *simp*
moreover have *on-l*: $\bigwedge x.\ x \in l \Longrightarrow ?g\ x \in l$
proof−
fix $x :: real\hat{}2$
assume $x \in l$
moreover then have $x\$2 = 0$ **by** (*metis assms*(6,10) *segment-horizontal open-closed-segment*)
moreover then have $(?g\ x)\$2 = 0$ **by** *simp*
moreover have $(?g\ x)\$1 = x\$1$ **by** *simp*
ultimately show $?g\ x \in l$ **by** (*smt* (*verit*, *ccfv-SIG*) *exhaust-2 vec-eq-iff*)
qed
ultimately have *path-image* $q \cap l = \{\}$
by (*metis* (*no-types*, *lifting*) *disjoint-iff f-eq-g image-eqI path-image-compose* q'*-def*)
moreover have $l \subseteq$ *frontier A*
proof−
have *pathstart* $p' =$ *pathstart* $p \wedge$ *pathfinish* $p' =$ *pathfinish* p
using *startfinish* **by** *auto*
then have $?f`l \subseteq$ *frontier* A'
using *pocket-fill-line-int-aux7*(2)[*OF 1 2 3 4 5 6 7 8 9 10*] *on-l f-eq-g l-def p0-def p1-def segment-open-subset-closed*
by *force*
thus *?thesis*
by (*metis* (*no-types*, *lifting*) A'*-def frontier-injective-linear-image inj inj-image-subset-iff matrix-vector-mul-linear*)
qed
ultimately show *?thesis* **by** *fast*
qed
ultimately show *?thesis* **by** *argo*
qed

lemma *simple-path-linear-image*:
assumes *simple-path p*
assumes *inj f* $\wedge$ *bounded-linear f*
shows *simple-path* $(f \circ p)$
proof−
have *continuous-on* $\{x.\ True\}$ *f* **using** *assms*(2) *linear-continuous-on* **by** *blast*
then have *1*: *path* $(f \circ p)$
by (*metis Collect-cong UNIV-I assms*(1) *continuous-on-subset path-continuous-image simple-path-imp-path top-empty-eq top-greatest top-set-def*)

have *inj-on p* {*0*<..<*1*} **by** (*simp add*: *assms*(1) *simple-path-inj-on*)
then have *inj-on* $(f \circ p)$ {*0*<..<*1*} **by** (*meson assms*(2) *comp-inj-on inj-on-subset top-greatest*)
then have *loop-free* $(f \circ p)$
by (*metis* (*mono-tags*, *lifting*) *assms*(1) *assms*(2) *comp-apply inj-eq loop-free-def simple-path-def*)
thus *?thesis* **using** *1* **unfolding** *simple-path-def* **by** *blast*

qed

lemma *vts-interior*:
fixes *vts*
defines $p \equiv$ *make-polygonal-path vts*
assumes *convex H*
assumes $\forall\, j \in \{0<..<\textit{length vts} - 1\}.\ \textit{vts}!j \notin \textit{frontier } H$
assumes *loop-free p*
assumes $\textit{path-image } p \subseteq H$
assumes $\textit{length vts} \geq 3$
shows $p`\{0<..<1\} \subseteq \textit{interior } H$
proof(*rule subsetI*)
fix *x* **assume** $*$: $x \in p`\{0<..<1\}$
then obtain *t* **where** *t*: $x = p\ t \wedge t \in \{0<..<1\}$ **by** *blast*
then have $x \neq p\ 0 \wedge x \neq p\ 1$ **using** *assms*(4) **unfolding** *loop-free-def* **by** *fastforce*
then have *x-neq*: $x \neq \textit{hd vts} \wedge x \neq \textit{last vts}$
by (*metis assms*(4) *constant-linepath-is-not-loop-free hd-conv-nth last-conv-nth make-polygonal-path.simps*(1) *p-def pathfinish-def pathstart-def polygon-pathfinish polygon-pathstart*)

have $x \in \textit{interior } H$ **if** $**$: $\exists\, i<\textit{length vts}.\ x = \textit{vts}!i$
proof−
obtain *i* **where** *i*: $i < \textit{length vts} \wedge x = \textit{vts}!i$ **using** $**$ **by** *blast*
then have $i \neq 0 \wedge i \neq \textit{length vts} - 1$
by (*metis x-neq gr-implies-not0 hd-conv-nth last-conv-nth list.size*(3))
then have $i \in \{0<..<\textit{length vts} - 1\}$ **using** *i* **by** *fastforce*
then have $\textit{vts}!i \notin \textit{frontier } H$ **using** *assms*(3) **by** *blast*
then have $\textit{vts}!i \in \textit{interior } H$
by (*metis DiffI assms*(5) *closure-subset frontier-def i nth-mem p-def subsetD vertices-on-path-image*)
thus *?thesis* **using** *assms*(3) *i* **by** *blast*
qed
moreover have $x \in \textit{interior } H$ **if** $**$: $\neg\ (\exists\, i<\textit{length vts}.\ x = \textit{vts}!i)$
proof−
have $x \in \textit{path-image } p$ **using** $*$ **unfolding** *path-image-def* **by** *force*
then obtain *i* **where** *i*: $x \in \textit{path-image } (\textit{linepath } (\textit{vts}!i)\ (\textit{vts}!(i+1))) \wedge i < \textit{length vts} - 1$
using *make-polygonal-path-image-property*[*of vts x*] *assms*(6) **unfolding** *p-def* **by** *auto*
moreover then have $x \neq \textit{vts}!i \wedge x \neq \textit{vts}!(i+1)$ **using** $**$ **by** *force*
ultimately have $x \in \textit{open-segment } (\textit{vts}!i)\ (\textit{vts}!(i+1))$ **by** (*simp add*: *open-segment-def*)
moreover then have $x \in \textit{rel-interior } (\textit{path-image } (\textit{linepath } (\textit{vts}!i)\ (\textit{vts}!(i+1))))$
by (*metis empty-iff open-segment-idem path-image-linepath rel-interior-closed-segment*)
moreover have *interior-nonempty*: $\textit{vts}!i \in \textit{interior } H \vee \textit{vts}!(i+1) \in \textit{interior } H$
proof(*rule ccontr*)
assume $\neg\ (\textit{vts}!i \in \textit{interior } H \vee \textit{vts}!(i+1) \in \textit{interior } H)$
then have $\textit{vts}!i \in \textit{frontier } H \wedge \textit{vts}!(i+1) \in \textit{frontier } H$

using *assms*(5) *closure-subset frontier-def i p-def vertices-on-path-image* **by** *fastforce*
thus *False*
by (*metis assms*(3) *i Suc-1 Suc-eq-plus1 add.commute add.right-neutral assms*(6) *eval-nat-numeral*(3) *greaterThanLessThan-iff less-diff-conv linorder-not-le not-gr-zero not-less-eq-eq*)
qed
ultimately have $x \in$ *rel-interior H*
by (*smt* (*verit*, *ccfv-SIG*) *add-diff-inverse-nat assms*(2) *assms*(5) *convex-same-rel-interior-closure-straddle empty-iff i in-interior-closure-convex-segment less-diff-conv less-nat-zero-code nat-diff-split nth-mem open-segment-commute p-def rel-interior-nonempty-interior subset-eq trans-less-add2 vertices-on-path-image*)
moreover have *interior H* $\neq \{\}$ **using** *interior-nonempty* **by** *blast*
ultimately show *?thesis* **using** *rel-interior-nonempty-interior* **by** *blast*
qed
ultimately show $x \in$ *interior H* **by** *blast*
qed

lemma *pocket-fill-line-int-0*:
assumes *polygon-of r vts*
defines $H \equiv$ *convex hull* (*set vts*)
assumes $2 \leq i \wedge i <$ *length vts* $- 1$
defines $a \equiv$ *hd vts*
defines $b \equiv$ *vts*!i
assumes $\{a, b\} \subseteq$ *frontier H*
assumes $\forall j \in \{0<..<i\}$. *vts*!$j \notin$ *frontier H*
assumes $a = 0$
shows *path-image* (*linepath a b*) $\cap$ *path-image r* $= \{a, b\}$
path-image (*linepath a b*) $\subseteq$ *frontier H*
proof−
let *?x* $= (b - a)$
let *?e* $=$ *norm* $(b - a) *_R$ ((*vector* [1, 0])::(*real*^2))
have *norm ?x* $=$ *norm ?e* **by** (*simp add*: *e1e2-basis*(1))
then obtain f **where** f: *orthogonal-transformation* $f \wedge$ *det*(*matrix* f) $= 1 \wedge f$ *?x* $=$ *?e*
using *rotation-exists* **by** (*metis two-le-card*)

have *bij*: *bij* $f \wedge$ *linear* f
using f *orthogonal-transformation-bij orthogonal-transformation-def* **by** *blast*

let *?p-vts* $=$ *take* $(i + 1)$ *vts*
let *?q-vts* $=$ *drop i vts*
let *?p* $=$ *make-polygonal-path ?p-vts*
let *?q* $=$ *make-polygonal-path ?q-vts*

let *?p'* $= f \circ$ *?p*
let *?q'* $= f \circ$ *?q*
let *?H'* $=$ *convex hull* (*path-image ?p'* $\cup$ *path-image ?q'*)

have *vts-split*: *vts* = *?p-vts* @ (*tl ?q-vts*)
by (*metis Suc-eq-plus1 append-take-drop-id drop-Suc tl-drop*)

have *simple-path r* **using** *assms*(*1*) **unfolding** *polygon-of-def polygon-def* **by** *blast*
then have *a-neq-b*: *a* ≠ *b*
using *simple-polygonal-path-vts-distinct*[*of vts*]
by (*metis* (*mono-tags*, *lifting*) *a-def assms*(*1*) *assms*(*3*) *b-def bot-nat-0.extremum-strict butlast-conv-take constant-linepath-is-not-loop-free distinct-nth-eq-iff dual-order.strict-trans2 hd-conv-nth length-butlast make-polygonal-path.simps*(*1*) *nat-neq-iff nth-take polygon-of-def pos2 simple-path-def*)

have *H-r*: *H* = *convex hull* (*path-image r*)
by (*metis* (*no-types*, *lifting*) *H-def Un-subset-iff assms*(*1*) *convex-convex-hull convex-hull-eq convex-hull-of-polygon-is-convex-hull-of-vts hull-mono hull-subset order-antisym-conv polygon-of-def vertices-on-path-image*)
moreover have *r-union*: *path-image r* = (*path-image ?p*) ∪ (*path-image ?q*)
proof−
let *?i* = *i* + *1*
let *?x* = ((*2*::*real*) ^ (*?i* − *1*) − *1*) / *2* ^ (*?i* − *1*)
have *?x* ∈ {*0*..*1*} ∧ *path-image ?p* = *r*‘{*0*..*?x*} ∧ *path-image ?q* = *r*‘{*?x*..*1*}
using *vts-split-path-image*[*of r vts ?p ?p-vts ?q ?q-vts ?i - ?x*]
by (*smt* (*verit*, *ccfv-SIG*) *add.commute add-diff-cancel-left*′ *assms*(*1*) *assms*(*3*) *atLeastAtMost-iff atLeastatMost-empty*′ *image-empty le-add1 less-diff-conv path-image-nonempty polygon-of-def*)
thus *?thesis* **by** (*metis atLeastAtMost-iff image-Un ivl-disj-un-two-touch*(*4*) *path-image-def*)
qed
moreover have *f*‘*H* = *convex hull* (*f*‘(*path-image r*))
using *bij* **by** (*simp add*: *calculation*(*1*) *convex-hull-linear-image*)
ultimately have *H-image*: *?H*′ = *f*‘*H* **by** (*simp add*: *image-Un path-image-compose*)

have *p-image*: *path-image ?p*′ = *f*‘(*path-image ?p*) **using** *path-image-compose* **by** *blast*
have *q-image*: *path-image ?q*′ = *f*‘(*path-image ?q*) **using** *path-image-compose* **by** *blast*

have *pathstart-p*: *pathstart ?p* = *a*
by (*metis Suc-eq-plus1 a-def assms*(*3*) *gr-implies-not0 hd-conv-nth length-tl less-Suc-eq-0-disj list.sel*(*2*) *list.size*(*3*) *nth-take polygon-pathstart take-eq-Nil*)
have *pathfinish-p*: *pathfinish ?p* = *b*
by (*metis* (*no-types*, *lifting*) *H-def H-r add-diff-cancel-right*′ *assms*(*3*) *b-def convex-hull-eq-empty length-take less-add-one less-diff-conv min.absorb4 nth-append one-neq-zero path-image-nonempty polygon-pathfinish set-empty take-eq-Nil vts-split zero-eq-add-iff-both-eq-0*)
then have *pathstart-q*: *pathstart ?q* = *b* **using** *assms*(*3*) *b-def polygon-pathstart* **by** *force*

have *pathstart-p*′: *pathstart ?p*′ = *f a* **using** *pathstart-compose pathstart-p* **by**

blast
have *pathfinish-p'*: *pathfinish ?p' = f b* **using** *pathfinish-compose pathfinish-p* **by** *blast*
have *pathstart-q'*: *pathstart ?q' = f b* **using** *pathstart-compose pathstart-q* **by** *blast*

have *sublist ?p-vts vts* **by** *auto*
then have *lf-p*: *loop-free ?p*
by (*metis add.commute assms*(*1*) *assms*(*3*) *less-diff-conv less-imp-le-nat polygon-def polygon-of-def simple-path-def take-i-is-loop-free trans-le-add2*)
then have *simple-p*: *simple-path ?p*
using *assms* **unfolding** *polygon-of-def*
by (*meson make-polygonal-path-gives-path simple-path-def*)

have *sublist ?q-vts vts* **by** *auto*
then have *lf-q*: *loop-free ?q*
by (*metis* (*no-types*, *lifting*) *Suc-1 Suc-diff-Suc assms*(*1*) *assms*(*3*) *diff-is-0-eq drop-i-is-loop-free less-Suc-eq-le less-zeroE linorder-not-less polygon-def polygon-of-def simple-path-def*)
then have *simple-q*: *simple-path ?q*
using *assms* **unfolding** *polygon-of-def*
by (*meson make-polygonal-path-gives-path simple-path-def*)

have *bounded-linear*: *bounded-linear f* **using** *bij linear-conv-bounded-linear* **by** *blast*
have *1*: *simple-path ?p'*
using *simple-p simple-path-linear-image bij bij-is-inj bounded-linear*
by *blast*
have *2*: *simple-path ?q'*
using *simple-q simple-path-linear-image bij bij-is-inj bounded-linear*
by *blast*
have *3*: *path-image ?p' ∩ path-image ?q' = {pathstart ?q', pathfinish ?q'}*
proof−
have *path-image ?p ∩ path-image ?q ⊆ {pathstart ?q, pathfinish ?q}*
using *loop-free-split-int*[*of r vts ?p-vts i ?q-vts ?p ?q*]
by (*smt* (*verit*, *ccfv-threshold*) *a-def add-diff-cancel-right' assms*(*1*) *assms*(*3*) *constant-linepath-is-not-loop-free drop-eq-Nil have-wraparound-vertex hd-conv-nth insert-commute last-conv-nth last-drop last-snoc le-add2 less-diff-conv lf-q linorder-not-less loop-free-split-int make-polygonal-path.simps*(*1*) *pathstart-p polygon-def polygon-of-def polygon-pathfinish simple-path-def*)
moreover have *pathstart ?q ∈ path-image ?q ∧ pathfinish ?q ∈ path-image ?q* **by** *blast*
moreover have *pathstart ?q ∈ path-image ?p ∧ pathfinish ?q ∈ path-image ?p*
by (*smt* (*verit*, *ccfv-SIG*) *a-def add-diff-cancel-right' assms*(*1*) *assms*(*3*) *b-def constant-linepath-is-not-loop-free drop-eq-Nil have-wraparound-vertex hd-conv-nth last-conv-nth last-drop last-snoc length-take less-add-one less-diff-conv lf-q linorder-not-less list.size*(*3*) *make-polygonal-path.simps*(*1*) *min.absorb4 nth-take pathfinish-in-path-image pathstart-in-path-image pathstart-p pathstart-q polygon-of-def polygon-pathfinish take-eq-Nil zero-eq-add-iff-both-eq-0 zero-neq-one*)

ultimately have *path-image ?p ∩ path-image ?q = {pathstart ?q, pathfinish ?q}* **by** *fast*
moreover have *path-image ?p' ∩ path-image ?q' = f`(path-image ?p ∩ path-image ?q)*
by (*metis bij bij-is-inj image-Int p-image q-image*)
ultimately show *?thesis* **by** (*simp add*: *pathfinish-compose pathstart-compose*)
qed
have *4*: (*pathstart ?p'*)$*1 = 0 ∧* (*pathstart ?p'*)$*2 = 0 ∧* (*pathfinish ?p'*)$*2 = 0*
proof−
have *f ?x = ?e* **using** *f* **by** *blast*
then have *f b − f a = ?e*
by (*metis assms*(*8*) *diff-zero f norm-eq-zero orthogonal-transformation-norm*)
moreover have *f a = 0* **by** (*metis assms*(*8*) *f norm-eq-zero orthogonal-transformation-norm*)
moreover from *calculation* **have** *f b = ?e* **by** *force*
ultimately show *?thesis* **using** *pathfinish-p' pathstart-p'* **by** *auto*
qed
have *5*: (*pathfinish ?p'*)$*1 > 0*
proof−
have *pathfinish ?p' = f b* **using** *pathfinish-p'* **by** *auto*
moreover have *f b = ?e* **using** *assms*(*8*) *f* **by** *auto*
moreover have *?e$1 = norm ?x* **by** *simp*
ultimately show *?thesis* **using** *a-neq-b* **by** *auto*
qed
have *6*: *?H' = convex hull* (*path-image ?p' ∪ path-image ?q'*) **by** *blast*
have *7*: {*pathstart ?p', pathfinish ?p'*} ⊆ *frontier ?H'*
proof−
have {*pathstart ?p, pathfinish ?p*} ⊆ *frontier H*
using *pathstart-p pathfinish-p assms*(*6*) **by** *fastforce*
then have *f`*{*pathstart ?p, pathfinish ?p*} ⊆ *f`*(*frontier H*) **by** *blast*
moreover have *f`*(*frontier H*) = *frontier* (*f`H*)
by (*simp add*: *bij bij-is-inj frontier-injective-linear-image*)
ultimately show *?thesis* **using** *H-image* **by** (*simp add*: *pathfinish-compose pathstart-compose*)
qed
have *8*: *?p'`*{*0<..<1*} ⊆ *interior ?H'*
proof−
have *1*: *convex H* **by** (*simp add*: *H-def*)
have *2*: ∀ *j*∈{*0<..<length ?p-vts − 1*}. *?p-vts ! j ∉ frontier H*
by (*simp add*: *add.commute assms*(*3*) *assms*(*7*) *less-diff-conv*)
have *3*: *loop-free ?p* **using** *lf-p* **by** *blast*
have *4*: *path-image ?p ⊆ H* **using** *H-r hull-subset r-union* **by** *fastforce*
have *5*: *length ?p-vts ≥ 3* **using** *assms*(*3*) **by** *force*
have *?p`*{*0<..<1*} ⊆ *interior H* **using** *vts-interior*[*OF 1 2 3 4 5*] **by** *argo*
moreover have *f`*(*?p`*{*0<..<1*}) = *?p'`*{*0<..<1*} **by** (*meson image-comp*)
moreover have *f`*(*interior H*) = *interior ?H'*
using *H-image interior-injective-linear-image*[*of f H*] **by** (*simp add*: *bij bij-is-inj*)
ultimately show *?thesis* **by** *fast*
qed

have *9*: *pathstart ?q′ = pathfinish ?p′ ∧ pathfinish ?q′ = pathstart ?p′*
by (*metis* (*mono-tags*, *lifting*) *H-def H-r a-def assms*(*1*) *constant-linepath-is-not-loop-free convex-hull-eq-empty drop-eq-Nil have-wraparound-vertex hd-conv-nth last-conv-nth last-drop last-snoc lf-q linorder-not-less make-polygonal-path.simps*(*1*) *path-image-nonempty pathfinish-compose pathfinish-p pathstart-compose pathstart-p pathstart-q polygon-of-def polygon-pathfinish set-empty*)

let *?l* = *open-segment a b*
let *?l′* = *open-segment* (*pathstart ?p′*) (*pathfinish ?p′*)

have ∗: *path-image ?q′* ∩ *open-segment* (*pathstart ?p′*) (*pathfinish ?p′*) = {} ∧ *?l′* ⊆ *frontier ?H′*
using *pocket-fill-line-int-aux8*[*OF 1 2 3 4 5 6 7 8 9*] **by** *blast*
moreover have *l-image*: *?l′* = *f*‘*?l*
proof−
have *f a* = *pathstart ?p′* ∧ *f b* = *pathfinish ?p′* **using** *pathfinish-p′ pathstart-p′* **by** *presburger*
moreover have ⋀*a b*. *f*‘(*open-segment a b*) = *open-segment* (*f a*) (*f b*)
by (*simp add*: *bij bij-is-inj open-segment-linear-image*)
ultimately show *?thesis* **by** *presburger*
qed
moreover have *path-image ?q′* = *f*‘(*path-image ?q*) **using** *q-image* **by** *blast*
ultimately have *path-image ?q* ∩ *?l* = {} **by** *blast*
moreover have *path-image ?p* ∩ *?l* = {}
proof−
from *8* **have** *path-image ?p′* ∩ *?l′* = {}
proof−
have *?p′*‘{*0*<..<*1*} ∩ *?l′* = {}
by (*smt* (*verit*, *ccfv-SIG*) ∗ *8 Diff-disjoint disjoint-iff frontier-def subset-iff*)
moreover have *?p′ 0* ∉ *?l′*
by (*metis* ∗ *9 IntI empty-iff pathfinish-in-path-image pathstart-def*)
moreover have *?p′ 1* ∉ *?l′*
by (*metis* ∗ *9 Int-iff emptyE pathfinish-def pathstart-in-path-image*)
ultimately show *?thesis*
by (*smt* (*verit*, *ccfv-SIG*) ∗ *1 3 9 Int-Un-eq*(*4*) *Un-Diff-cancel Un-iff disjoint-iff insert-commute simple-path-endless*)
qed
thus *?thesis* **using** *l-image bij p-image* **by** *auto*
qed
ultimately have *path-image r* ∩ *?l* = {}
by (*simp add*: *r-union boolean-algebra.conj-disj-distrib inf-commute*)
moreover have *a* ∈ *path-image r* **using** *pathstart-p r-union* **by** *auto*
moreover have *b* ∈ *path-image r* **using** *pathfinish-p r-union* **by** *auto*
moreover have (*path-image* (*linepath a b*)) = *?l* ∪ {*a*, *b*} **by** (*simp add*: *closed-segment-eq-open*)
ultimately show *path-image* (*linepath a b*) ∩ *path-image r* = {*a*, *b*} **by** *auto*

have *l′-frontier*: *?l′* ⊆ *frontier ?H′* **using** ∗ **by** *presburger*
have *?l* ⊆ *frontier H*

proof−
have *?l′ = f‘?l* **using** *l-image* **by** *blast*
moreover have *frontier ?H′ = f‘(frontier H)*
by (*metis H-image bij bij-is-inj frontier-injective-linear-image*)
ultimately have *f‘?l ⊆ f‘(frontier H)* **using** *l′-frontier* **by** *argo*
thus *?thesis* **by** (*simp add*: *bij bij-is-inj inj-image-subset-iff*)
qed
moreover have *closed-segment a b = path-image* (*linepath a b*) **by** *simp*
moreover have *closed-segment a b = ?l ∪ {a, b}* **by** (*simp add*: *closed-segment-eq-open*)
moreover have *a ∈ frontier H ∧ b ∈ frontier H* **using** *assms*(*6*) **by** *auto*
ultimately show *path-image* (*linepath a b*) *⊆ frontier H* **by** *simp*
qed

lemma *linepath-translation*: (*λv. v − a*) *∘* (*linepath x y*) = *linepath* ((*λv. v − a*) *x*) ((*λv. v − a*) *y*)
by (*auto simp*: *linepath-def algebra-simps*)

lemma *linepath-image-translation*:
path-image ((*λv. v − a*) *∘* (*linepath x y*)) = *path-image* (*linepath* ((*λv. v − a*) *x*) ((*λv. v − a*) *y*))
using *linepath-translation* **by** *metis*

lemma *make-polygonal-path-translate*:
assumes *length vts ≥ 1*
shows (*λv. v − a*) *∘* (*make-polygonal-path vts*) = *make-polygonal-path* (*map* (*λv. v − a*) *vts*)
using *assms*
proof(*induct length vts arbitrary*: *vts a*)
case *0*
then show *?case* **by** *linarith*
next
case (*Suc n*)
{ assume *∗*: *Suc n = 1*
then have *make-polygonal-path vts = linepath* (*vts*!*0*) (*vts*!*0*)
by (*metis Cons-nth-drop-Suc One-nat-def Suc.hyps*(*2*) *Suc.prems drop0 drop-eq-Nil less-numeral-extra*(*1*) *make-polygonal-path.simps*(*2*))
then have (*λv. v − a*) *∘* (*make-polygonal-path vts*) = *linepath* ((*vts*!*0*) *− a*) ((*vts*!*0*) *− a*)
by *fastforce*
then have *?case*
by (*metis Cons-nth-drop-Suc One-nat-def Suc.hyps*(*2*) *Suc.prems ∗ drop0 drop-eq-Nil list.map*(*1*) *list.simps*(*9*) *make-polygonal-path.simps*(*2*) *zero-less-one*)
} moreover
{ assume *∗*: *Suc n = 2*
then have *make-polygonal-path vts = linepath* (*vts*!*0*) (*vts*!*1*)
by (*metis* (*no-types*, *lifting*) *Cons-nth-drop-Suc One-nat-def Suc.hyps*(*2*) *Suc-1 diff-Suc-1 drop0 drop-Suc drop-eq-Nil le-numeral-extra*(*4*) *length-tl less-numeral-extra*(*1*) *make-polygonal-path.simps*(*3*) *nth-tl pos2*)
then have (*λv. v − a*) *∘* (*make-polygonal-path vts*) = *linepath* ((*vts*!*0*) *− a*)

((*vts*!*1*) − *a*)
using *linepath-translation* **by** *auto*
then have *?case*
by (*metis* (*no-types*, *lifting*) ∗ *Cons-nth-drop-Suc One-nat-def Suc.hyps*(*2*) *Suc-1 drop0 drop-eq-Nil length-map lessI make-polygonal-path.simps*(*3*) *nat-le-linear nth-map pos2*)
} moreover
{ assume ∗: *Suc n* ≥ *3*
then obtain *h h*′ *t* **where** *vts*: *vts* = *h* # *h*′ # *t*
by (*metis Suc.hyps*(*2*) *Suc-le-length-iff numeral-3-eq-3*)
then have (λ*v*. *v* − *a*) ∘ (*make-polygonal-path* (*h*′ # *t*))
= *make-polygonal-path* (*map* (λ*v*. *v* − *a*) (*h*′ # *t*))
using *Suc.hyps*(*1*) *Suc.hyps*(*2*) ∗ **by** *auto*
moreover have (λ*v*. *v* − *a*) ∘ (*linepath h h*′) = *linepath* (*h* − *a*) (*h*′ − *a*)
using *linepath-translation* **by** *blast*
moreover have *make-polygonal-path vts* = (*linepath h h*′) +++ (*make-polygonal-path* (*h*′ # *t*))
by (*metis* ∗ *Suc.hyps*(*2*) *Suc-le-length-iff vts list.sel*(*3*) *make-polygonal-path.simps*(*4*) *numeral-3-eq-3*)
ultimately have *?case*
by (*smt* (*verit*) *list.discI list.inject list.simps*(*9*) *make-polygonal-path.elims path-compose-join vts*)
}
ultimately show *?case* **using** *Suc.prems* **by** *linarith*
qed

lemma *pocket-fill-line-int*:
assumes *polygon-of r vts*
defines *H* ≡ *convex hull* (*set vts*)
assumes *2* ≤ *i* ∧ *i* < *length vts* − *1*
defines *a* ≡ *hd vts*
defines *b* ≡ *vts*!*i*
assumes {*a*, *b*} ⊆ *frontier H*
assumes ∀ *j* ∈ {*0*<..<*i*}. *vts*!*j* ∉ *frontier H*
shows *path-image* (*linepath a b*) ∩ *path-image r* = {*a*, *b*}
path-image (*linepath a b*) ⊆ *frontier H*
proof−
let *?f* = (λ*v*. *v* − *a*)::(*real*^*2* ⇒ *real*^*2*)
let *?r*′ = *?f* ∘ *r*
let *?vts*′ = *map ?f vts*
let *?H*′ = *convex hull* (*set ?vts*′)
let *?a*′ = *?f a*
let *?b*′ = *?f b*

have *5*: *hd ?vts*′ = *0*
by (*metis One-nat-def a-def assms*(*3*) *cancel-comm-monoid-add-class.diff-cancel lessI list.map-sel*(*1*) *list.size*(*3*) *nat-diff-split-asm not-less-zero*)

have *a*′*b*′: *?a*′ = *hd ?vts*′ ∧ *?b*′ = *?vts*′!*i* **using** *5 assms*(*3*) *b-def* **by** *force*

have *frontier-H'*: *frontier ?H' = ?f ' (frontier H)*
using *frontier-translation*[*of −a H*]
by (*metis* (*no-types*, *lifting*) *H-def convex-hull-translation image-cong list.set-map uminus-add-conv-diff*)

have *simple-path r* **using** *assms*(*1*) *polygon-def polygon-of-def* **by** *blast*
then have *simple-path ?r'* **using** *simple-path-translation-eq*[*of −a r*] **by** *simp*
moreover have *?r' = make-polygonal-path ?vts'*
using *make-polygonal-path-translate assms*(*1*) *assms*(*3*) *polygon-of-def* **by** *auto*
moreover have *closed-path ?r'*
by (*smt* (*verit*, *best*) *closed-path-def add-diff-inverse-nat assms*(*1*) *assms*(*3*) *calculation*(*1*) *calculation*(*2*) *dual-order.refl gr-implies-not0 hd-conv-nth length-map less-Suc-eq-le list.map-disc-iff list.map-sel*(*1*) *nat-diff-split-asm nth-map plus-1-eq-Suc polygon-def polygon-of-def polygon-pathfinish polygon-pathstart simple-path-def*)
ultimately have *1*: *polygon-of ?r' ?vts'*
unfolding *polygon-of-def polygon-def polygon-def polygonal-path-def* **by** *blast*
have *2*: *2 ≤ i ∧ i < length ?vts' − 1* **using** *assms*(*3*) **by** *auto*
have *3*: {*hd ?vts'*, *?vts'*!*i*} ⊆ *frontier ?H'*
using *a'b' frontier-H'*
by (*metis* (*no-types*, *lifting*) *assms*(*6*) *image-empty image-insert image-mono*)
have *4*: ∀ *j* ∈ {*0*<..<*i*}. *?vts'*!*j* ∉ *frontier ?H'*
proof
fix *j* **assume** ∗: *j* ∈ {*0*<..<*i*}
then have *vts*!*j* ∉ *frontier H* **using** *assms*(*7*) **by** *blast*
then have *?f* (*vts*!*j*) ∉ *frontier ?H'* **using** *frontier-H'* **by** *auto*
thus *?vts'*!*j* ∉ *frontier ?H'* **using** *Nat.le-imp-diff-is-add* ∗ *assms*(*3*) **by** *auto*
qed

have *path-image* (*linepath ?a' ?b'*) ∩ *path-image ?r' =* {*?a'*, *?b'*}
using *pocket-fill-line-int-0*(*1*)[*OF 1 2 3 4 5*] *a'b'* **by** *argo*
moreover have {*?a'*, *?b'*} = *?f*‘{*a*, *b*} **by** *simp*
moreover have *path-image* (*linepath ?a' ?b'*) = *?f*‘(*path-image* (*linepath a b*))
using *linepath-image-translation path-image-compose* **by** *blast*
moreover have *path-image ?r' = ?f*‘(*path-image r*) **using** *path-image-compose* **by** *blast*
ultimately have *?f*‘(*path-image* (*linepath a b*)) ∩ *?f*‘(*path-image r*) = *?f*‘{*a*, *b*} **by** *argo*
then have *?f*‘(*path-image* (*linepath a b*) ∩ *path-image r*) = *?f*‘{*a*, *b*} **by** (*simp add*: *image-Int*)
moreover have *bij ?f* **by** (*simp add*: *bij-diff-right*)
ultimately show *path-image* (*linepath a b*) ∩ *path-image r* = {*a*, *b*}
by (*meson bij-is-inj inj-image-eq-iff*)

have *path-image* (*linepath ?a' ?b'*) ⊆ *frontier ?H'*
using *pocket-fill-line-int-0*(*2*)[*OF 1 2 3 4 5*] *a'b'* **by** *argo*
thus *path-image* (*linepath a b*) ⊆ *frontier H*
by (*metis* ‹*bij ?f*› ‹*path-image* (*linepath ?a' ?b'*) = *?f*‘(*path-image* (*linepath a b*))› *bij-betw-imp-inj-on frontier-H' inj-image-subset-iff*)

qed

lemma *path-connected-simple-path-endless*:
assumes *simple-path p*
shows *path-connected* (*path-image p* − {*pathstart p*, *pathfinish p*}) (**is** *path-connected ?S*)
proof−
have *continuous-on* {*0*<..<*1*} *p*
using *assms*(*1*) **unfolding** *simple-path-def path-def*
by (*meson continuous-on-path dual-order.refl greaterThanLessThan-subseteq-atLeastAtMost-iff path-def*)
moreover have *path-connected* {*0*<..<*1*::*real*} **by** *simp*
ultimately have *path-connected* (*p*‘{*0*<..<*1*}) **using** *path-connected-continuous-image* **by** *blast*
thus *?thesis* **using** *simple-path-endless assms* **by** *metis*
qed

lemma *simple-loop-split*:
assumes *simple-path p* ∧ *closed-path p*
assumes *simple-path q*
assumes *path-image q* ∩ *path-image p* = {*q 0*, *q 1*}
assumes *path-image q* ∩ *path-inside p* ≠ {}
shows *q*‘{*0*<..<*1*} ⊆ *path-inside p*
proof−
have *inside-outside*: *inside-outside p* (*path-inside p*) (*path-outside p*)
using *Jordan-inside-outside-real2 closed-path-def assms*(*1*) *inside-outside-def path-inside-def path-outside-def*
by *presburger*

obtain *x* **where** *x*: *x* ∈ *path-image q* ∩ *path-inside p* **using** *assms*(*4*) **by** *blast*
then obtain *tx* **where** *tx* ∈ {*0*..*1*} ∧ *q tx* = *x* **unfolding** *path-image-def* **by** *fast*
moreover then have *tx* ≠ *0* ∧ *tx* ≠ *1*
using *assms*(*3*) *inside-outside x* **unfolding** *inside-outside-def* **by** *auto*
ultimately have *tx*: *tx* ∈ {*0*<..<*1*} ∧ *q tx* = *x* **by** *simp*

have *connected* (*q*‘{*0*<..<*1*})
using *connected-simple-path-endless simple-path-endless assms*(*2*) **by** *metis*
then have *path-connected* (*q*‘{*0*<..<*1*})
using *path-connected-simple-path-endless assms*(*2*) *simple-path-endless* **by** *metis*
moreover have *q*‘{*0*<..<*1*} ∩ *path-inside p* ≠ {} **using** *tx x* **by** *blast*
moreover have *q*‘{*0*<..<*1*} ∩ *frontier* (*path-inside p*) = {}
using *inside-outside* **unfolding** *inside-outside-def*
by (*smt* (*verit*, *del-insts*) *Diff-Int-distrib2 assms*(*2*,*3*) *diff-eq inf-compl-bot-right inf-idem inf-sup-aci*(*1*) *pathfinish-def pathstart-def simple-path-endless*)
ultimately show *?thesis*
using *path-connected-not-frontier-subset*[*of q*‘{*0*<..<*1*} *path-inside p*] **by** *fast*
qed

lemma *pocket-path-interior-aux*:
assumes *simple-path p* ∧ *simple-path q*
assumes *arc p* ∧ *arc q*
assumes *q 0* = *p 1* ∧ *q 1* = *p 0*
assumes *path-image p* ∩ *path-image q* = {*p 0*, *q 0*}
defines *A* ≡ *convex hull* (*path-image p* ∪ *path-image q*)
defines *l* ≡ *linepath* (*p 0*) (*p 1*)
assumes *p*‘{*0*<..<*1*} ⊆ *interior A*
assumes *path-image q* ∩ *path-image l* = {*l 0*, *q 0*}
shows *p*‘{*0*<..<*1*} ∩ *path-inside* (*l* +++ *q*) ≠ {}
simple-path (*l* +++ *q*) ∧ *closed-path* (*l* +++ *q*)
path-image p ∩ *path-image* (*l* +++ *q*) = {*p 0*, *p 1*}
proof−
let *?r* = *l* +++ *q*
let *?Ir* = *path-inside ?r*
let *?Or* = *path-outside ?r*
show *closed-simple-r*: *simple-path ?r* ∧ *closed-path ?r*
using *simple-path-join-loop*[*of l q*] *assms* **unfolding** *pathstart-def pathfinish-def*
by (*metis* (*no-types*, *opaque-lifting*) *closed-path-def arc-linepath arc-simple-path dual-order.refl inf-commute linepath-0′ linepath-1′ pathfinish-def pathfinish-join pathstart-def pathstart-join simple-path-def*)
then have *inside-outside-r*: *inside-outside ?r ?Ir ?Or*
by (*simp add*: *Jordan-inside-outside-real2 closed-path-def inside-outside-def path-inside-def path-outside-def*)

have *l-p-endpoints*: *l 0* = *p 0* ∧ *l 1* = *p 1* **by** (*simp add*: *l-def linepath-0′ linepath-1′*)
have *l-q-endpoints*: *l 0* = *q 1* ∧ *l 1* = *q 0* **by** (*simp add*: *assms*(*3*) *l-p-endpoints*)
have *p-int-l*: *p*‘{*0*<..<*1*} ∩ *path-image l* = {} **using** *assms*(*7*,*8*) **unfolding** *frontier-def* **by** *blast*
have *q-int-l*: *q*‘{*0*<..<*1*} ∩ *path-image l* = {}
by (*metis* (*no-types*, *opaque-lifting*) *assms*(*9*) *Diff-iff Int-Diff all-not-in-conv assms*(*1*) *assms*(*3*) *inf-sup-aci*(*1*) *insert-commute l-def linepath-0′ pathfinish-def pathstart-def simple-path-endless*)
have *interval*: {*0*..*1*::*real*} = {*0*<..<*1*} ∪ {*0*, *1*} **by** *fastforce*
have *lf-l*: *loop-free l*
using *closed-simple-r not-loop-free-first-component simple-path-def* **by** *blast*

let *?p′* = *reversepath p*
let *?s* = *l* +++ *?p′*
let *?Is* = *path-inside ?s*
let *?Os* = *path-outside ?s*
have *arc ?p′* ∧ *arc l*
by (*metis assms*(*2*) *arc-linepath arc-reversepath arc-simple-path l-def pathfinish-def pathstart-def*)
moreover have *p′-int-l*: *path-image ?p′* ∩ *path-image l* = {*?p′ 0*, *l 0*}
proof−

```
    have path-image p ∩ path-image l = {l 0, l 1}
    proof−
      have {l 0, l 1} ⊆ path-image p ∩ path-image l
        using assms(3) assms(4) l-def linepath-0′ linepath-1′ by fastforce
      moreover have path-image p = p‘{0<..<1} ∪ {p 0, p 1}
        using interval unfolding path-image-def by blast
      ultimately show ?thesis using p-int-l l-p-endpoints by simp
    qed
    moreover have ?p′ 0 = l 1 by (simp add: l-def linepath-1′ reversepath-def)
    moreover have path-image p = path-image ?p′ by simp
    ultimately show ?thesis by (metis doubleton-eq-iff)
  qed
  ultimately have closed-simple-s: closed-path ?s ∧ simple-path ?s
      using simple-path-join-loop[of l ?p′] assms unfolding pathstart-def pathfin-
ish-def
    by (metis (no-types, opaque-lifting) closed-path-def dual-order.refl inf-commute
insert-commute linepath-0′ linepath-1′ pathfinish-def pathfinish-join pathfinish-reversepath
pathstart-def pathstart-join pathstart-reversepath simple-path-def)
  then have inside-outside-s: inside-outside ?s ?Is ?Os
      by (simp add: Jordan-inside-outside-real2 closed-path-def inside-outside-def
path-inside-def path-outside-def)

  have r-inside-subset: path-inside ?r ⊆ interior A
  proof−
    have path-image l ⊆ A ∧ path-image q ⊆ A
    by (metis A-def Un-upper2 assms(1) assms(8) compact-Un compact-convex-hull
compact-simple-path-image frontier-subset-compact hull-subset subset-trans)
    thus ?thesis
    by (metis (no-types, lifting) A-def closed-simple-r convex-contains-simple-closed-path-imp-contains-path-ins
convex-convex-hull inside-outside-def inside-outside-r interior-eq interior-mono sub-
set-path-image-join)
  qed
  have s-inside-subset: path-inside ?s ⊆ interior A
  proof−
    have path-image l ⊆ A ∧ path-image p ⊆ A
    by (metis A-def Un-upper1 assms(1) assms(8) compact-Un compact-convex-hull
compact-simple-path-image frontier-subset-compact hull-subset subset-trans)
    thus ?thesis
      by (metis A-def Jordan-inside-outside-real2 closed-path-def closed-simple-s
convex-contains-simple-closed-path-imp-contains-path-inside convex-convex-hull in-
terior-maximal path-image-reversepath path-inside-def subset-path-image-join)
  qed

  have q-outside: q‘{0<..<1} ⊆ path-outside ?s
  proof(rule ccontr)
    let ?ep = {v. v extreme-point-of A}
    assume ¬ q‘{0<..<1} ⊆ path-outside ?s
    then have ∃x ∈ q‘{0<..<1}. x ∈ path-inside ?s ∪ path-image ?s
      using inside-outside-s unfolding inside-outside-def by auto
```

then have $q`\{0<..<1\} \subseteq$ *path-inside ?s*
using *simple-loop-split*[*of p q*]
by (*smt* (*verit*) *DiffE IntI Int-Un-distrib2 closed-path-def UnE* ‹*arc* (*reversepath p*) ∧ *arc l*› *arc-imp-path assms*(*1*) *assms*(*2*) *assms*(*3*) *assms*(*4*) *closed-simple-r closed-simple-s doubleton-eq-iff emptyE inf.commute l-def path-image-join path-image-reversepath path-join-eq pathfinish-join pathfinish-linepath pathstart-join pathstart-linepath simple-loop-split simple-path-endless simple-path-joinE sup-absorb2*)
then have $q`\{0<..<1\} \cap$ *frontier A* $= \{\}$ **using** *frontier-def s-inside-subset* **by** *fastforce*
then have (*path-image p* ∪ *path-image q*) ∩ *frontier A* $= \{p\ 0, p\ 1\}$
by (*smt* (*verit*,*del-insts*) *Diff-disjoint Int-Un-distrib Un-Diff-Int Un-Int-eq*(*3*) *assms*(*1*) *assms*(*3*) *assms*(*4*) *assms*(*7*) *assms*(*8*) *assms*(*9*) *frontier-def inf.commute inf.orderE inf-idem inf-left-commute insert-commute l-p-endpoints pathfinish-def pathstart-def simple-path-endless*)
moreover have *?ep*⊆ *path-image p* ∪ *path-image q*
by (*simp add*: *extreme-points-of-convex-hull A-def*)
moreover have *?ep* ⊆ *frontier A*
using *extreme-point-not-in-interior*
proof−
have *?ep* ∩ *interior A* $= \{\}$
using *extreme-point-not-in-interior* **by** *blast*
thus *?thesis*
by (*smt* (*verit*, *ccfv-SIG*) *A-def Int-Un-distrib2 Un-Diff-cancel assms*(*1*) *calculation*(*2*) *closure-convex-hull compact-Un compact-simple-path-image dual-order.trans frontier-def hull-subset inf.absorb-iff2 inf-commute sup-bot-left*)
qed
ultimately have ∗: *?ep* ⊆ $\{p\ 0, p\ 1\}$ **by** *auto*
have *A* = *path-image l*
proof−
have *convex A* ∧ *compact A*
by (*simp add*: *A-def arc-imp-path assms*(*2*) *compact-Un compact-convex-hull compact-path-image*)
then have *A-ep*: *A* = *convex hull ?ep* **using** *Krein-Milman-Minkowski* **by** *blast*
moreover have *finite ?ep* **using** ∗ *infinite-super* **by** *auto*
moreover have *A* ≠ $\{\}$ **by** (*simp add*: *A-def*)
moreover have ∀ *x*. *A* ≠ $\{x\}$ **using** *assms*(*7*) **by** *fastforce*
ultimately have *card ?ep* ≥ *2* **using** *convex-hull-two-extreme-points* **by** *metis*
then have *?ep* $= \{p\ 0, p\ 1\}$
by (*metis* ∗ *One-nat-def Suc-1 add-leD2 card.empty card-insert-disjoint card-seteq finite.emptyI finite.insertI insert-absorb plus-1-eq-Suc*)
then have *A* = *closed-segment* (*p 0*) (*p 1*) **by** (*metis A-ep segment-convex-hull*)
thus *?thesis* **by** (*simp add*: *l-def*)
qed
then have *interior A* $= \{\}$
by (*metis A-def Diff-eq-empty-iff assms*(*1*) *assms*(*8*) *closure-convex-hull compact-Un compact-simple-path-image double-diff dual-order.refl frontier-def interior-subset*)
thus *False* **using** *inside-outside-def inside-outside-r r-inside-subset* **by** *auto*

qed

let *?e* = *l* (*1*/*2*)
have *l-on-r-frontier*: *path-image l* ⊆ *frontier* (*path-inside ?r*)
using *inside-outside-r* **unfolding** *inside-outside-def*
by (*metis Un-upper1 closed-simple-r* ‹*arc* (*reversepath p*) ∧ *arc l*› *arc-def assms*(*2*) *path-image-join path-join-eq simple-path-def*)
moreover have *path-image l* ⊆ *frontier* (*path-inside ?s*)
using *inside-outside-s* **unfolding** *inside-outside-def*
by (*simp add*: *l-def path-image-join pathstart-def reversepath-def*)
ultimately have *e-frontier*: *?e* ∈ *frontier* (*path-inside ?r*) ∧ *?e* ∈ *frontier* (*path-inside ?s*)
by (*simp add*: *path-defs*(*4*) *subsetD*)

have *e-notin*: *?e* ∉ *path-image p* ∪ *path-image q*
proof−
have *?e* ∉ *path-image p*
proof−
have *?e* ≠ *l 0* ∧ *?e* ≠ *l 1* **using** *lf-l* **unfolding** *loop-free-def* **by** *fastforce*
then have *?e* ≠ *p 0* ∧ *?e* ≠ *p 1* **using** *l-p-endpoints* **by** *simp*
moreover have *?e* ∉ *p*‘{*0*<..<*1*} **using** *p-int-l* **unfolding** *path-image-def* **by** *fastforce*
ultimately show *?thesis* **using** *p-int-l* **unfolding** *path-image-def* **by** *fastforce*
qed
moreover have *?e* ∉ *path-image q*
proof−
have *?e* ≠ *l 0* ∧ *?e* ≠ *l 1* **using** *lf-l* **unfolding** *loop-free-def* **by** *fastforce*
then have *?e* ≠ *q 0* ∧ *?e* ≠ *q 1* **using** *l-q-endpoints* **by** *simp*
moreover have *?e* ∉ *q*‘{*0*<..<*1*} **using** *q-int-l* **unfolding** *path-image-def* **by** *fastforce*
ultimately show *?thesis* **using** *q-int-l* **unfolding** *path-image-def* **by** *fastforce*
qed
ultimately show *?thesis* **by** *blast*
qed
obtain ε **where** ε: ε > *0* ∧ *ball ?e* ε ∩ *path-image p* = {} ∧ *ball ?e* ε ∩ *path-image q* = {}
proof−
have *?e* ∉ *path-image p* **using** *e-notin* **by** *simp*
moreover have *compact* (*path-image p*) **by** (*simp add*: *assms*(*2*) *compact-arc-image*)
moreover have *?e* ∉ *path-image q* **using** *e-notin* **by** *simp*
moreover have *compact* (*path-image q*) **by** (*simp add*: *assms*(*2*) *compact-arc-image*)
ultimately obtain *ε1 ε2* **where**
ε1 > *0* ∧ *ball ?e ε1* ∩ *path-image p* = {} ∧ *ε2* > *0* ∧ *ball ?e ε2* ∩ *path-image q* = {}
by (*meson assms*(*1*) *not-on-path-ball simple-path-imp-path*)
thus *?thesis* **using** *that*[*of min ε1 ε2*] **by** (*simp add*: *disjoint-iff*)
qed

obtain *z-r* **where** *z-r*: *z-r* ∈ *ball ?e* ε ∩ *path-inside ?r*

by (*metis e-frontier ε all-not-in-conv disjoint-iff frontier-straddle mem-ball*)
obtain *z-s* **where** *z-s*: *z-s* ∈ *ball ?e ε* ∩ *path-inside ?s*
by (*metis e-frontier ε all-not-in-conv disjoint-iff frontier-straddle mem-ball*)

have *z-s-in-r*: *z-s* ∈ *path-inside ?r*
proof−
let *?l-z* = *linepath z-r z-s*
have *z-r* ∈ *interior A* ∧ *z-s* ∈ *interior A*
using *r-inside-subset s-inside-subset z-r z-s* **by** *blast*
then have *path-image ?l-z* ⊆ *interior A* **by** (*simp add*: *A-def closed-segment-subset*)
then have *1*: *path-image ?l-z* ∩ *path-image l* = {}
by (*smt* (*verit*) *Diff-iff assms*(*8*) *disjoint-iff frontier-def subsetD*)

have *convex* (*ball ?e ε*) **by** *simp*
then have *path-image ?l-z* ⊆ *ball ?e ε*
by (*metis IntD1 closed-segment-subset path-image-linepath z-r z-s*)
then have *2*: *path-image ?l-z* ∩ *path-image q* = {} **using** *ε* **by** *blast*

show *?thesis*
by (*smt* (*verit*, *best*) *1 2 IntI Int-Un-distrib Int-Un-distrib2 Jordan-inside-outside-real2 closed-path-def ε* ‹*path-image* (*linepath z-r z-s*) ⊆ *ball* (*l* (*1* / *2*)) *ε*› *arc-def assms*(*2*) *closed-simple-r emptyE in-mono inf.assoc le-iff-inf path-connected-not-frontier-subset path-connected-path-image path-image-join path-inside-def path-join-path-ends path-linepath pathfinish-in-path-image pathfinish-linepath pathstart-in-path-image pathstart-linepath sup.order-iff z-r*)
qed

let *?xq* = *q* (*1*/*2*)
let *?z* = *z-s*

let *?v* = *?xq* − *?z*
let *?ray* = λ*d*. *?z* + *d* ∗$_R$ *?v*
let *?rayline* = *linepath ?z ?xq*
have *z-ray*: *?z* = *?ray 0* **by** *simp*
have *xq-ray*: *?xq* = *?ray 1* **by** *simp*
have *xq-rayline*: *?xq* = *?rayline 1* **unfolding** *linepath-def* **by** *simp*
have *?xq* ∈ *path-image ?r*
by (*metis* (*mono-tags*, *opaque-lifting*) *Un-iff atLeastAtMost-iff imageI l-q-endpoints less-eq-real-def path-defs*(*4*) *path-image-join pathfinish-def pathstart-def pos-half-less zero-less-divide-1-iff zero-less-numeral zero-less-one*)
then have *xq-frontier*: *?xq* ∈ *frontier* (*path-inside ?r*)
using *inside-outside-r* **unfolding** *inside-outside-def* **by** *auto*
have *xq-neq-z*: *?xq* ≠ *?z*
proof−
have *?xq* ∈ *path-image ?r*
proof−
have *q* (*1* / *2*) ∈ *path-image q*
by (*simp add*: *path-defs*(*4*))
thus *?thesis*

by (*simp add*: *l-q-endpoints path-image-join pathfinish-def pathstart-def*)
qed
thus *?thesis* **using** *z-s-in-r inside-outside-r* **unfolding** *inside-outside-def* **by** *blast*
qed
then have *v-neq-0*: *?v* ≠ *0* **by** *simp*

have *bounded* (*path-inside ?r*) **using** *inside-outside-r* **unfolding** *inside-outside-def* **by** *blast*
moreover have *?z* ∈ *interior* (*path-inside ?r*)
by (*metis inside-outside-def inside-outside-r interior-eq z-s-in-r*)
ultimately obtain *d* **where** *d*: *0* < *d* ∧ *?ray d* ∈ *frontier* (*path-inside ?r*)
∧ (∀ *e* ∈ {*0*..<*d*}. *?ray e* ∈ *interior* (*path-inside ?r*))
using *ray-to-frontier*[*of path-inside ?r ?z ?v*] **by** (*metis atLeastLessThan-iff v-neq-0*)

have *interior-inside-r*: *interior* (*path-inside ?r*) = *path-inside ?r*
by (*meson inside-outside-def inside-outside-r interior-eq*)
have *d-leq-1*: *d* ≤ *1*
proof(*rule ccontr*)
assume ¬ *d* ≤ *1*
then have *d* > *1* **by** *simp*
moreover have *?ray 1* ∈ *frontier* (*path-inside ?r*) **using** *xq-ray xq-frontier* **by** *argo*
ultimately show *False* **using** *d* **unfolding** *frontier-def* **by** *fastforce*
qed

have *z-inside*: *?z* ∈ *path-inside ?s* **using** *z-s* **by** *blast*
moreover have *?rayline d* ∈ *path-outside ?s*
proof−
have *?rayline d* ∉ *path-image l* **if** *d* < *1*
proof−
have *?rayline 0* ∈ *interior A*
using *r-inside-subset* **by** (*simp add*: *linepath-0′ subsetD z-s-in-r*)
moreover have *path-image ?rayline* ⊆ *closure A*
proof−
have *closure A* = *A*
using *A-def assms*(*1*) *closure-convex-hull compact-Un compact-simple-path-image* **by** *blast*
moreover have *?rayline 0* ∈ *A* **using** ‹*?rayline 0* ∈ *interior A*› *interior-subset* **by** *blast*
moreover have *?rayline 1* ∈ *A*
using *path-image-def A-def hull-subset xq-rayline* **by** *fastforce*
ultimately show *?thesis*
by (*metis A-def closed-segment-subset convex-convex-hull linepath-0′ linepath-1′ path-image-linepath*)
qed
moreover have ¬ *path-image ?rayline* ⊆ *rel-frontier A*
proof−

have *path-image ?rayline ∩ interior A ≠ {}*
using ‹*?rayline 0 ∈ interior A*› **unfolding** *path-image-def* **by** *fastforce*
moreover have *interior A ∩ rel-frontier A = {}*
using *rel-frontier-def rel-interior-nonempty-interior* **by** *auto*
ultimately show *?thesis* **by** *blast*
qed
ultimately have *rel-interior (path-image ?rayline) ⊆ rel-interior A*
using *subset-rel-interior-convex*[*of path-image ?rayline A*] **by** (*simp add*: *A-def*)
moreover have *interior A = rel-interior A*
using ‹*?rayline 0 ∈ interior A*› *rel-interior-nonempty-interior* **by** *auto*
moreover have *?rayline d ∈ ?rayline‘{0<..<1}* **using** *that d* **by** *simp*
ultimately show *?thesis*
by (*smt* (*verit*, *del-insts*) *DiffD1 DiffD2 Un-iff xq-neq-z arc-linepath arc-simple-path assms*(*8*) *closed-segment-eq-open frontier-def path-image-linepath pathfinish-linepath pathstart-linepath rel-interior-closed-segment simple-path-endless subset-eq*)
qed
moreover have *?rayline d ∉ path-image l* **if** *d = 1*
using *that q-int-l* **unfolding** *linepath-def* **by** (*simp add*: *disjoint-iff*)
moreover have *?rayline d ∈ path-image ?r*
by (*metis* (*no-types*, *lifting*) *add-diff-eq d diff-add-eq inside-outside-def inside-outside-r linepath-def scale-left-diff-distrib scale-one scale-right-diff-distrib*)
ultimately show *?thesis*
by (*smt* (*verit*, *ccfv-SIG*) *d-leq-1 Diff-iff Int-iff closed-path-def ‹arc (reversepath p) ∧ arc l› arc-def assms*(*1*) *assms*(*3*) *assms*(*9*) *closed-simple-r insert-commute l-def l-p-endpoints not-in-path-image-join path-join-eq pathfinish-join pathfinish-linepath pathstart-join pathstart-linepath q-outside simple-path-def simple-path-endless subsetD*)
qed
moreover have *?z ∈ ?rayline‘{0..d}*
using *z-ray* **unfolding** *linepath-def*
by (*smt* (*verit*, *del-insts*) *add.commute atLeastAtMost-iff cancel-comm-monoid-add-class.diff-cancel d diff-zero image-iff less-eq-real-def segment-degen-1*)
moreover have *?rayline d ∈ ?rayline‘{0..d}* **by** (*simp add*: *d less-eq-real-def*)
ultimately have *?rayline‘{0..d} ∩ path-inside ?s ≠ {} ∧ ?rayline‘{0..d} ∩ path-outside ?s ≠ {}*
by *blast*
then have *?rayline‘{0..d} ∩ path-inside ?s ≠ {} ∧ ?rayline‘{0..d} ∩ − path-inside ?s ≠ {}*
using *inside-outside-s* **unfolding** *inside-outside-def* **by** (*meson ComplI disjoint-iff*)
moreover have *path-connected (?rayline‘{0..d})*
proof−
have *?rayline‘{0..d} = path-image (subpath 0 d ?rayline)* **by** (*simp add*: *d path-image-subpath*)
moreover have *path (subpath 0 d ?rayline)* **using** *d d-leq-1* **by** *auto*
ultimately show *?thesis* **by** (*metis path-connected-path-image*)
qed
ultimately have *?rayline‘{0..d} ∩ frontier (path-inside ?s) ≠ {}*

using *path-connected-frontier*[*of ?rayline‘*{*0..d*} *path-inside ?s*] **by** (*metis disjoint-iff*)
then have *?rayline‘*{*0..d*} ∩ *path-image ?s* ≠ {} **using** *inside-outside-s* **unfolding** *inside-outside-def* **by** *argo*
moreover have *?rayline 0* ∉ *path-image ?s*
proof−
have *?xq* ≠ *p 0*
by (*metis* (*full-types*) *disjoint-iff greaterThanLessThan-iff imageI l-p-endpoints pathstart-def pathstart-in-path-image pos-half-less q-int-l zero-less-divide-1-iff zero-less-numeral zero-less-one*)
moreover have *?xq* ≠ *p 1*
by (*metis* (*full-types*) *disjoint-iff greaterThanLessThan-iff imageI l-p-endpoints pathfinish-def pathfinish-in-path-image pos-half-less q-int-l zero-less-divide-1-iff zero-less-numeral zero-less-one*)
moreover have *?xq* ∉ *p‘*{*0<..<1*}
proof−
have *?xq* ∈ *q‘*{*0<..<1*} **by** *fastforce*
thus *?thesis* **by** (*metis assms*(*1*,*3*,*4*) *Diff-iff Int-iff pathfinish-def pathstart-def simple-path-endless*)
qed
moreover have *?xq* ∉ *path-image l*
by (*metis disjoint-iff greaterThanLessThan-iff imageI pos-half-less q-int-l zero-less-divide-1-iff zero-less-numeral zero-less-one*)
ultimately show *?thesis*
by (*metis* (*no-types*, *lifting*) *ComplD UnI1 z-inside inside-outside-def inside-outside-s linepath-0′*)
qed
moreover have *?rayline d* ∉ *path-image ?s*
using ‹*?rayline d* ∈ *path-outside ?s*› *inside-outside-def inside-outside-s* **by** *auto*
moreover have {*0..d*} = {*0<..<d*} ∪ {*0*, *d*} **using** *d* **by** *fastforce*
ultimately have *?rayline‘*{*0<..<d*} ∩ *path-image ?s* ≠ {} **unfolding** *path-image-def* **by** *blast*
moreover have *?rayline‘*{*0<..<d*} = *?ray‘*{*0<..<d*}
unfolding *linepath-def* **by** (*auto simp*: *algebra-simps*)
moreover have *?ray‘*{*0<..<d*} ⊆ *path-inside ?r* **using** *d interior-inside-r* **by** *fastforce*
ultimately have *path-image ?s* ∩ *path-inside ?r* ≠ {} **by** *blast*
moreover have *path-image l* ∩ *path-inside ?r* = {}
by (*metis* (*no-types*, *opaque-lifting*) *Diff-disjoint Int-assoc l-on-r-frontier frontier-def inf.orderE inf-bot-left inf-sup-aci*(*1*) *interior-inside-r*)
moreover have *p‘*{*0<..<1*} = *path-image ?s* − *path-image l*
proof−
have *path-image ?s* = *path-image p* ∪ *path-image l*
by (*simp add*: *l-p-endpoints path-image-join pathfinish-def sup-commute*)
moreover have *p‘*{*0<..<1*} = *path-image p* − {*p 0*, *p 1*}
by (*metis assms*(*1*) *pathfinish-def pathstart-def simple-path-endless*)
ultimately have *path-image ?s* = *p‘*{*0<..<1*} ∪ {*p 0*, *p 1*} ∪ *path-image l*
using *assms*(*3*) *assms*(*9*) *l-p-endpoints* **by** *auto*
moreover have *p 1* ∈ *path-image l* ∧ *p 0* ∈ *path-image l* **by** (*simp add*: *l-def*)

ultimately show *?thesis* **using** *p-int-l* **by** *blast*
qed
ultimately show *p‘{0<..<1} ∩ path-inside (l +++ q) ≠ {}* **by** *auto*

show *path-image p ∩ path-image (l +++ q) = {p 0, p 1}*
by (*smt* (*verit*, *best*) *Int-Un-distrib Un-absorb assms(1) assms(3) assms(4) closed-simple-r insert-commute l-p-endpoints p'-int-l path-image-join path-image-reversepath path-join-path-ends reversepath-def simple-path-imp-path*)
qed

lemma *pocket-path-interior*:
assumes *simple-path p ∧ simple-path q*
assumes *arc p ∧ arc q*
assumes *q 0 = p 1 ∧ q 1 = p 0*
assumes *path-image p ∩ path-image q = {p 0, q 0}*
defines *A ≡ convex hull (path-image p ∪ path-image q)*
defines *l ≡ linepath (p 0) (p 1)*
assumes *p‘{0<..<1} ⊆ interior A*
assumes *path-image q ∩ path-image l = {l 0, q 0}*
shows *p‘{0<..<1} ⊆ path-inside (l +++ q)*
using *pocket-path-interior-aux*[*of p q*] *simple-loop-split*[*of l +++ q p*] *assms*
by (*metis* (*no-types*, *lifting*) *DiffE disjoint-iff simple-path-endless*)

lemma *pocket-path-good*:
assumes *polygon* (*make-polygonal-path vts*)
assumes *vts!0 ∈ frontier* (*convex hull* (*set vts*))
assumes *vts!1 ∉ frontier* (*convex hull* (*set vts*))
assumes ¬ *convex* (*path-image* (*make-polygonal-path vts*) ∪ *path-inside* (*make-polygonal-path vts*))
defines *pocket-path-vts ≡ construct-pocket-0 vts* (*set vts ∩ frontier* (*convex hull* (*set vts*)))
defines *pocket ≡ make-polygonal-path* (*pocket-path-vts @* [*pocket-path-vts!0*])
defines *filled-vts ≡ fill-pocket-0 vts* (*length pocket-path-vts*)
defines *filled-p ≡ make-polygonal-path filled-vts*
defines *a ≡ hd pocket-path-vts*
defines *b ≡ last pocket-path-vts*
defines *good-pocket-path-vts ≡ tl* (*butlast pocket-path-vts*)
shows *polygon filled-p*
is-polygon-split-path (*butlast filled-vts*) *0 1 good-pocket-path-vts*
polygon pocket
card (*set pocket-path-vts*) < *card* (*set vts*)
card (*set filled-vts*) < *card* (*set vts*)
proof−
let *?p = make-polygonal-path vts*
let *?A = set vts ∩ frontier* (*convex hull* (*set vts*))
let *?filled-vts-tl = tl filled-vts*
let *?filled-p-tl = make-polygonal-path ?filled-vts-tl*
let *?pocket-vts = pocket-path-vts @* [*pocket-path-vts!0*]

let *?pocket-path* = *make-polygonal-path pocket-path-vts*
let *?l* = *linepath a b*

let *?r* = *min-nonzero-index-in-set vts ?A*
have *int-A-nonempty*: *set* (*tl vts*) ∩ *?A* ≠ {}
by (*metis* (*mono-tags*, *lifting*) *IntI Nitpick.size-list-simp*(*2*) *Suc-eq-plus1 assms*(*1*) *assms*(*2*) *card-length empty-iff have-wraparound-vertex last-in-set last-tl le-add1 le-trans not-less-eq-eq numeral-3-eq-3 polygon-at-least-3-vertices snoc-eq-iff-butlast*)
then have *r-defined*: *nonzero-index-in-set vts ?A ?r* ∧ (∀ *i* < *?r*. ¬ *nonzero-index-in-set vts ?A i*)
using *min-nonzero-index-in-set-defined*[*of vts ?A*] **by** *fast*

have *two-vts-on-frontier*: *2* ≤ *card ?A*
by (*metis convex-hull-two-vts-on-frontier One-nat-def Suc-1 add-leD2 assms*(*1*) *numeral-3-eq-3 plus-1-eq-Suc polygon-at-least-3-vertices*)
moreover have *frontier-vts-subset*: *?A* ⊆ *set vts* **by** *force*
moreover have *distinct-vts*: *distinct* (*butlast vts*)
using *assms*(*1*) *polygon-def simple-polygonal-path-vts-distinct* **by** *blast*
moreover have *hd-last-vts*: *hd vts* = *last vts*
by (*metis assms*(*1*) *have-wraparound-vertex hd-conv-nth snoc-eq-iff-butlast*)
ultimately have *a-neq-b*: *a* ≠ *b*
using *a-def b-def construct-pocket-0-first-last-distinct pocket-path-vts-def* **by** *presburger*
have *length filled-vts* ≥ *2*
unfolding *filled-vts-def fill-pocket-0-def*
by (*smt* (*verit*, *best*) *One-nat-def Suc-1 Suc-diff-Suc a-def a-neq-b b-def construct-pocket-0-def diff-is-0-eq diff-zero hd-Nil-eq-last length-drop length-greater-0-conv length-tl list.sel*(*3*) *not-less-eq-eq pocket-path-vts-def sublist-length-le sublist-take*)
moreover have *filled-vts-0*: *a* = *filled-vts*!*0*
unfolding *filled-vts-def fill-pocket-0-def a-def pocket-path-vts-def construct-pocket-0-def*
by *auto*
moreover have *filled-vts-1*: *b* = *filled-vts*!*1*
by (*smt* (*verit*, *del-insts*) *filled-vts-def fill-pocket-0-def b-def pocket-path-vts-def construct-pocket-0-def Cons-nth-drop-Suc Nitpick.size-list-simp*(*2*) *a-def a-neq-b add.right-neutral drop0 drop-eq-Nil hd-Nil-eq-last last-conv-nth length-take length-tl linorder-not-less list.sel*(*3*) *min.absorb4 nat-le-linear not-less-eq-eq nth-drop nth-take plus-1-eq-Suc take-all-iff zero-less-diff*)
ultimately have *filled-vts*: *filled-vts* = [*a*, *b*] @ *tl ?filled-vts-tl*
by (*metis* (*no-types*, *lifting*) *Nitpick.size-list-simp*(*2*) *One-nat-def Suc-1 append-Nil append-eq-Cons-conv length-greater-0-conv list.collapse not-less-eq-eq nth-Cons-0 nth-tl order-less-le-trans pos2*)

have *1*: *polygon-of ?p vts* **unfolding** *polygon-of-def* **using** *assms*(*1*) **by** *blast*
have *2*: *2* ≤ *?r* ∧ *?r* < *length vts* − *1*
proof−
have *?r* ≠ *0* ∧ *?r* ≠ *1*
using *assms*(*2*,*3*) *min-nonzero-index-in-set-def nonzero-index-in-set-def r-defined*

by *fastforce*
then have *1*: *?r* ≥ *2* **by** *simp*

have ∃ *i* ∈ {*0*<..<*length vts* − *1*}. *vts*!*i* ∈ *frontier* (*convex hull* (*set vts*))
proof−
have *card* ((*set vts*) ∩ *frontier* (*convex hull* (*set vts*))) ≥ *2*
using *two-vts-on-frontier* **by** *blast*
then obtain *v* **where** *v* ∈ *set vts* ∧ *v* ∈ *frontier* (*convex hull set vts*) ∧ *v* ≠ *hd vts*
by (*metis hd-last-vts Int-iff a-neq-b assms*(*2*) *b-def construct-pocket-0-last-in-set convex-hull-empty empty-set fill-pocket-0-def filled-vts-0 filled-vts-def frontier-empty hd-conv-nth int-A-nonempty last-in-set nth-Cons-0 pocket-path-vts-def*)
thus *?thesis*
by (*metis hd-last-vts assms*(*1*) *in-set-conv-nth diff-Suc-1 gr0-implies-Suc greaterThanLessThan-iff have-wraparound-vertex last-conv-nth le-eq-less-or-eq less-Suc-eq-le less-one nat.simps*(*3*) *nat-le-linear snoc-eq-iff-butlast*)
qed
then have *2*: *?r* < *length vts* − *1*
using *r-defined*
unfolding *min-nonzero-index-in-set-def nonzero-index-in-set-def*
by (*smt* (*verit*, *del-insts*) *Int-iff add.commute add-diff-cancel-left' add-diff-inverse-nat greaterThanLessThan-iff less-imp-diff-less mem-Collect-eq nat-less-le nth-mem*)
show *?thesis* **using** *1 2* **by** *blast*
qed
have *ab*: *a* = *hd vts* ∧ *b* = *vts*!*?r*
by (*metis* (*no-types*, *lifting*) *2 Suc-1 int-A-nonempty ab-semigroup-add-class.add-ac*(*1*) *add-Suc-right b-def construct-pocket-0-def fill-pocket-0-def filled-vts-0 filled-vts-def hd-drop-conv-nth last-snoc le-add-diff-inverse2 min-nonzero-index-in-set-bound nth-Cons-0 plus-1-eq-Suc pocket-path-vts-def take-hd-drop*)
have *3*: {*hd vts*, *vts* ! *?r*} ⊆ *frontier* (*convex hull set vts*)
using *ab assms*(*1*) *assms*(*2*) *assms*(*3*) *b-def construct-pocket-is-pocket is-pocket-0-def pocket-path-vts-def*
by *fastforce*
have *4*: ∀ *j*∈{*0*<..<*?r*}. *vts* ! *j* ∉ *frontier* (*convex hull set vts*)
using *r-defined* **unfolding** *nonzero-index-in-set-def* **by** *fastforce*

have *l-int-p*: *path-image* (*linepath* (*hd vts*) (*vts* ! *?r*)) ∩ *path-image ?p* = {*hd vts*, *vts* ! *?r*}
using *pocket-fill-line-int*[*OF 1 2 3 4*] **by** *blast*
have *l-frontier*: *path-image* (*linepath* (*hd vts*) (*vts* ! *?r*)) ⊆ *frontier* (*convex hull* (*set vts*))
using *pocket-fill-line-int*[*OF 1 2 3 4*] **by** *blast*

have *path-image ?filled-p-tl* ∩ *path-image ?l* = {*a*, *b*}
proof−
have *path-image* (*linepath* (*hd vts*) (*vts* ! *?r*)) ∩ *path-image ?p* = {*hd vts*, *vts* ! *?r*}
using *pocket-fill-line-int*[*OF 1 2 3 4*] **by** *blast*

moreover have *path-image ?filled-p-tl* ⊆ *path-image ?p*
proof −
have *sublist ?filled-vts-tl vts* **by** (*simp add*: *fill-pocket-0-def filled-vts-def*)
thus *?thesis* **using** ‹*2* ≤ *length filled-vts*› *sublist-path-image-subset* **by** *auto*
qed
moreover have *a* ∈ *path-image ?filled-p-tl* ∧ *b* ∈ *path-image ?filled-p-tl*
by (*smt* (*verit*, *best*) *Cons-nth-drop-Suc Diff-insert-absorb One-nat-def Suc-1* ‹*2* ≤ *length filled-vts*› *drop0 drop-eq-Nil fill-pocket-0-def filled-vts-0 filled-vts-1 filled-vts-def hd-last-vts last-drop last-in-set linorder-not-le list.sel*(*3*) *not-less-eq-eq nth-Cons-0 order-less-le-trans pathstart-in-path-image polygon-pathstart pos2 subset-Diff-insert vertices-on-path-image*)
ultimately show *?thesis* **using** *ab* **by** *auto*
qed
moreover have *hd-filled*: *hd ?filled-vts-tl* = *last* [*a*, *b*]
unfolding *filled-vts-def fill-pocket-0-def pocket-path-vts-def construct-pocket-0-def*
by (*metis construct-pocket-0-def fill-pocket-0-def filled-vts filled-vts-def hd-append2 last-ConsL last-ConsR list.sel*(*1*) *list.sel*(*3*) *list.simps*(*3*) *pocket-path-vts-def tl-append2*)
moreover have *last-filled*: *last ?filled-vts-tl* = *hd* [*a*, *b*]
unfolding *filled-vts-def fill-pocket-0-def pocket-path-vts-def construct-pocket-0-def*
using *r-defined a-def assms*(*1*) *assms*(*2*) *assms*(*3*) *construct-pocket-is-pocket hd-last-vts is-pocket-0-def pocket-path-vts-def*
by *fastforce*
moreover have *loop-free ?filled-p-tl*
proof −
have *sublist ?filled-vts-tl vts*
unfolding *filled-vts-def fill-pocket-0-def pocket-path-vts-def construct-pocket-0-def*
using *r-defined*
by *force*
thus *?thesis*
by (*smt* (*verit*, *del-insts*) *Nitpick.size-list-simp*(*2*) *Suc-1* ‹*2* ≤ *length filled-vts*› ‹*b* = *filled-vts* ! *1*› *a-neq-b assms*(*1*) *diff-is-0-eq dual-order.strict-trans1 last-conv-nth last-filled le-antisym length-greater-0-conv length-tl list.sel*(*1*) *list.size*(*3*) *not-less-eq-eq nth-tl polygon-def pos2 simple-path-def sublist-is-loop-free sublist-length-le*)
qed
moreover have *loop-free ?l* **using** *a-neq-b linepath-loop-free* **by** *blast*
moreover have *filled-vts*: *filled-vts* = [*a*, *b*] @ *tl ?filled-vts-tl* **using** *filled-vts* **by** *blast*
moreover have *arc ?l*
by (*smt* (*verit*) *arc-linepath calculation*(*5*) *constant-linepath-is-not-loop-free*)
moreover have *arc ?filled-p-tl*
by (*smt* (*verit*) *arc-simple-path calculation*(*2*) *calculation*(*3*) *calculation*(*4*) *calculation*(*7*) *hd-Nil-eq-last hd-conv-nth last.simps last-conv-nth list.discI list.sel*(*1*) *make-polygonal-path-gives-path pathfinish-linepath pathstart-linepath polygon-pathfinish polygon-pathstart simple-path-def*)
moreover have *?l* = *make-polygonal-path* [*a*, *b*]
using *make-polygonal-path.simps* **by** *presburger*
ultimately have *lf-filled*: *loop-free filled-p*
by (*smt* (*verit*) *Nat.add-diff-assoc One-nat-def Suc-pred′ add-Suc-shift append-butlast-last-id arc-distinct-ends butlast.simps*(*2*) *filled-p-def hd-Nil-eq-last hd-conv-nth*

inf-sup-aci(*1*) *last-ConsR less-numeral-extra*(*1*) *list.sel*(*1*) *list.simps*(*3*) *list.size*(*3*) *list.size*(*4*) *loop-free-append nth-append-length order-eq-refl plus-1-eq-Suc polygon-pathfinish polygon-pathstart*)
show *polygon-filled-p*: *polygon filled-p*
unfolding *polygon-def*
by (*metis closed-path-def UNIV-def append-is-Nil-conv filled-p-def filled-vts hd-append2 last.simps last-conv-nth last-filled lf-filled list.discI list.exhaust-sel make-polygonal-path-gives-path nth-Cons-0 polygon-pathfinish polygon-pathstart polygonal-path-def rangeI simple-path-def*)

have {*a*, *b*} ⊆ *set filled-vts*
using *filled-vts* **by** (*smt* (*verit*) *UnCI empty-set list.simps*(*15*) *set-append subset-iff*)
moreover have *pocket-path*: *?pocket-path* = *make-polygonal-path* ([*a*] @ *good-pocket-path-vts* @ [*b*])
by (*metis* (*no-types*, *lifting*) *a-def a-neq-b append-Cons append-Nil append-butlast-last-id b-def good-pocket-path-vts-def hd-Nil-eq-last hd-conv-nth last-conv-nth length-butlast list.collapse list.size*(*3*) *tl-append2*)
moreover have *path-image ?pocket-path* ⊆ *path-inside filled-p* ∪ {*a*, *b*}
proof−
let *?*p = *?pocket-path*
let *?*q = *?filled-p-tl*
let *?H* = *convex hull* (*path-image ?*p ∪ *path-image ?*q)
have *b*: *pocket-path-vts* = *take* (*?r* + *1*) *vts*
unfolding *pocket-path-vts-def construct-pocket-0-def* **by** *blast*
moreover then have *c′*: *?filled-vts-tl* = *drop ?r vts* **unfolding** *filled-vts-def fill-pocket-0-def*
using *2* **by** *fastforce*
ultimately have *vts* = *pocket-path-vts* @ *tl ?filled-vts-tl*
by (*metis Suc-eq-plus1 append-take-drop-id drop-Suc tl-drop*)
then have *path-image ?p* = *path-image ?*p ∪ *path-image ?*q
by (*metis Suc-1 a-def a-neq-b b-def diff-is-0-eq hd-Nil-eq-last hd-conv-nth hd-filled last.simps last-conv-nth last-filled list.discI list.sel*(*1*) *make-polygonal-path-image-append-alt not-less-eq-eq path-image-join polygon-pathfinish polygon-pathstart*)
moreover have *convex hull* (*path-image ?p*) = *convex hull* (*set vts*)
by (*metis* (*no-types*, *lifting*) *1 Un-subset-iff convex-hull-of-polygon-is-convex-hull-of-vts hull-Un-subset hull-mono subset-antisym vertices-on-path-image*)
ultimately have *H-eq*: *?H* = *convex hull* (*set vts*) **by** *presburger*

have *a*: *?p* = *make-polygonal-path vts* ∧ *loop-free ?p*
using *assms*(*1*) *polygon-def simple-path-def* **by** *blast*
have *c*: *?filled-vts-tl* = *drop* ((*?r* + *1*) − *1*) *vts* **using** *c′* **by** *simp*
have *h*: *1* ≤ *?r* + *1* ∧ *?r* + *1* < *length vts* **using** *2* **by** *linarith*
have *path-image ?*p ∩ *path-image ?*q ⊆ {*?*p *0*, *?*q *0*}
using *loop-free-split-int*[*OF a b c - - - h*] **by** (*simp add*: *pathstart-def*)
moreover have *?*p *0* ∈ *path-image ?*p ∧ *?*p *0* ∈ *path-image ?*q
by (*metis a-def a-neq-b b-def hd-Nil-eq-last hd-conv-nth hd-filled last.simps last-conv-nth last-filled list.sel*(*1*) *pathfinish-in-path-image pathstart-def pathstart-in-path-image polygon-pathfinish polygon-pathstart*)

moreover have *?q 0 ∈ path-image ?p ∧ ?q 0 ∈ path-image ?q*
by (*metis a-def a-neq-b b-def hd-Nil-eq-last hd-conv-nth hd-filled last.simps last-conv-nth last-filled list.sel(1) pathfinish-in-path-image pathstart-def pathstart-in-path-image polygon-pathfinish polygon-pathstart*)
ultimately have *4*: *path-image ?p ∩ path-image ?q = {?p 0, ?q 0}* **by** *fastforce*

have *1*: *simple-path ?p ∧ simple-path ?q*
by (*metis* (*no-types*, *lifting*) *One-nat-def Suc-1 Suc-le-eq ‹arc ?filled-p-tl› arc-simple-path assms(1) assms(2) assms(3) construct-pocket-is-pocket is-pocket-0-def le-add2 make-polygonal-path-gives-path numeral-3-eq-3 order-le-less-trans plus-1-eq-Suc pocket-path-vts-def polygon-def simple-path-def sublist-is-loop-free sublist-take*)
have *2*: *arc ?p ∧ arc ?q*
by (*metis 1 ‹arc ?filled-p-tl› a-def a-neq-b b-def hd-Nil-eq-last hd-conv-nth last-conv-nth polygon-pathfinish polygon-pathstart simple-path-cases*)
have *3*: *?q 0 = ?p 1 ∧ ?q 1 = ?p 0*
by (*metis 1 a-def append-Cons b-def constant-linepath-is-not-loop-free filled-vts hd-conv-nth last-conv-nth last-filled list.sel(1) list.sel(3) make-polygonal-path.simps(1) pathfinish-def pathstart-def polygon-pathfinish polygon-pathstart simple-path-def*)
have *5*: *?p ‘ {0<..<1} ⊆ interior ?H*
proof −
have *∀ j ∈ {0<..<?r}. vts!j ∉ frontier* (*convex hull* (*set vts*))
by (*smt* (*verit*, *del-insts*) *Int-iff dual-order.strict-trans greaterThanLessThan-iff int-A-nonempty mem-Collect-eq min-nonzero-index-in-set-defined nonzero-index-in-set-def nth-mem*)
moreover have *?r = length pocket-path-vts − 1* **using** *b h* **by** *auto*
moreover have *∀ j < ?r. vts!j = pocket-path-vts!j* **using** *b* **by** *auto*
ultimately have *∀ j ∈ {0<..<length pocket-path-vts − 1}. pocket-path-vts!j ∉ frontier ?H*
using *H-eq* **by** *simp*
moreover have *loop-free ?pocket-path* **using** *1 simple-path-def* **by** *auto*
ultimately show *?thesis*
by (*metis vts-interior Un-subset-iff assms(1) assms(2) assms(3) construct-pocket-is-pocket convex-convex-hull hull-subset is-pocket-0-def pocket-path-vts-def*)
qed
have *6*: *path-image* (*linepath* (*?p 0*) (*?p 1*)) ⊆ *frontier ?H*
by (*metis l-frontier H-eq 3 a-def a-neq-b ab b-def hd-Nil-eq-last hd-conv-nth hd-filled last.simps last-filled list.discI list.sel(1) pathstart-def polygon-pathstart*)
have *7*: *path-image ?q ∩ path-image* (*linepath* (*?p 0*) (*?p 1*)) = {*linepath* (*?p 0*) (*?p 1*) *0*, *?q 0*}
by (*metis 3 ‹path-image* (*make-polygonal-path* (*tl filled-vts*)) ∩ *path-image* (*linepath a b*) = {*a*, *b*}› *a-def a-neq-b b-def hd-Nil-eq-last hd-filled last.simps last-conv-nth last-filled linepath-0′ list.sel(1) pathfinish-def polygon-pathfinish*)
have *?p ‘ {0<..<1} ⊆ path-inside* (*linepath* (*?p 0*) (*?p 1*) *+++ ?q*)
using *pocket-path-interior*[*OF 1 2 3 4 5 6 7*] **by** *blast*
then have *?p‘{0<..<1} ⊆ path-inside filled-p*
by (*smt* (*verit*) *3 ‹2 ≤ length filled-vts› a-def a-neq-b b-def filled-p-def filled-vts-0 hd-Nil-eq-last hd-filled last.simps last-filled length-greater-0-conv list.discI list.sel(1) list.sel(3) make-polygonal-path.elims nth-Cons-0 order-less-le-trans path-start-def polygon-pathstart pos2*)

moreover have *?p 0 = a ∧ ?p 1 = b*
by (*metis 3 a-def a-neq-b b-def hd-Nil-eq-last hd-conv-nth hd-filled last.simps last-filled list.discI list.sel(1) pathstart-def polygon-pathstart*)
ultimately show *?thesis*
by (*metis 1 Diff-subset-conv a-def a-neq-b b-def hd-Nil-eq-last hd-conv-nth last-conv-nth polygon-pathfinish polygon-pathstart simple-path-endless sup-commute*)
qed
moreover have *loop-free-pocket-path*: *loop-free ?pocket-path*
proof−
have *sublist pocket-path-vts vts*
by (*simp add*: *construct-pocket-0-def pocket-path-vts-def*)
moreover have *loop-free ?p*
using *assms(1) polygon-def simple-path-def* **by** *blast*
moreover have *length pocket-path-vts ≥ 2*
by (*metis Suc-1 a-def a-neq-b b-def diff-is-0-eq′ hd-Nil-eq-last hd-conv-nth last-conv-nth not-less-eq-eq*)
moreover have *length vts ≥ 2*
by (*meson calculation(1) calculation(3) le-trans sublist-length-le*)
ultimately show *?thesis* **using** *sublist-is-loop-free* **by** *blast*
qed
ultimately have *good-polygonal-path*: *good-polygonal-path a good-pocket-path-vts b filled-vts*
by (*metis a-neq-b filled-p-def good-polygonal-path-def*)

have *filled-vts-as-butlast*: *filled-vts* = (*butlast filled-vts*) @ [(*butlast filled-vts*)!*0*]
by (*metis Nitpick.size-list-simp(2) append.right-neutral butlast-conv-take filled-p-def filled-vts have-wraparound-vertex length-butlast length-tl less-Suc-eq-0-disj list.discI list.sel(2) list.sel(3) nth-butlast polygon-filled-p*)
then have *filled-p-as-butlast*:
filled-p = *make-polygonal-path* ((*butlast filled-vts*) @ [(*butlast filled-vts*)!*0*])
unfolding *filled-p-def filled-vts-def* **by** *argo*
have *le*: *0* < (*1*::*nat*) **by** *simp*

have *filled-0-a*: (*butlast filled-vts*) ! *0* = *a*
by (*metis append-Cons append-Nil butlast.simps(2) filled-vts nth-Cons-0 filled-vts-0*)
have *filled-1-b*: (*butlast filled-vts*) ! *1* = *b*
by (*metis* (*no-types*, *opaque-lifting*) *filled-vts-1 filled-vts-as-butlast a-neq-b append-Cons append-Nil butlast-conv-take filled-0-a filled-vts length-butlast less-one linorder-not-le nat-less-le nth-append-length nth-butlast take0*)

have *01*: *0* < *length* (*butlast filled-vts*) ∧ *1* < *length* (*butlast filled-vts*)
by (*metis One-nat-def Suc-lessI filled-vts-1 filled-vts-as-butlast a-neq-b append-eq-Cons-conv filled-0-a length-greater-0-conv nth-Cons-Suc nth-append-length*)
show *is-split-path*:
is-polygon-split-path (*butlast filled-vts*) *0 1 good-pocket-path-vts*
using *good-polygonal-path-implies-polygon-split-path*
[*OF polygon-filled-p filled-p-as-butlast - 01 filled-0-a filled-1-b le*]
using *good-polygonal-path filled-vts-as-butlast*

by *presburger*

have *polygon-pocket-rev*: *polygon* (*make-polygonal-path* (*a*#([] @ [*b*] @ (*rev good-pocket-path-vts*) @ [*a*])))
unfolding *is-polygon-split-path-def*
by (*smt* (*verit*) *01 One-nat-def add-diff-cancel-left' add-diff-cancel-right' filled-0-a filled-1-b is-polygon-split-path-def is-split-path nth-butlast plus-1-eq-Suc take0*)
moreover have *rev-pocket-vts*: *rev ?pocket-vts* = *a*#([] @ [*b*] @ (*rev good-pocket-path-vts*) @ [*a*])
by (*smt* (*verit*) *a-def a-neq-b append.left-neutral append-Cons append-butlast-last-id b-def good-pocket-path-vts-def hd-Nil-eq-last hd-append2 hd-conv-nth last-conv-nth length-butlast list.collapse list.size*(*3*) *rev.simps*(*1*) *rev.simps*(*2*) *rev-append*)
ultimately show *polygon pocket*
by (*metis polygon-pocket-rev rev-vts-is-polygon polygon-of-def pocket-def rev-rev-ident*)

have *card* (*set vts*) = *length* (*butlast vts*)
using *distinct-vts*
by (*smt* (*verit*, *ccfv-threshold*) *Suc-n-not-le-n Un-insert-right append-Nil2 assms*(*1*) *butlast-conv-take distinct-card dual-order.strict-trans have-wraparound-vertex hd-conv-nth hd-in-set hd-take insert-absorb length-0-conv length-butlast less-eq-Suc-le linorder-linear list.set*(*2*) *not-numeral-le-zero numeral-3-eq-3 polygon-at-least-3-vertices-wraparound polygon-vertices-length-at-least-4 set-append*)
then have *set pocket-path-vts* ⊂ *set vts*
unfolding *pocket-path-vts-def construct-pocket-0-def*
using *r-defined*
by (*smt* (*verit*, *ccfv-threshold*) *Cons-nth-drop-Suc One-nat-def Suc-diff-Suc Suc-le-lessD add-diff-cancel-right' assms*(*1*) *assms*(*2*) *assms*(*3*) *butlast-conv-take butlast-snoc card-length construct-pocket-0-def construct-pocket-is-pocket drop0 fill-pocket-0-def filled-vts-def is-pocket-0-def is-polygon-split-path-def is-split-path leD le-less-Suc-eq length-butlast length-drop length-greater-0-conv list.inject numeral-3-eq-3 plus-1-eq-Suc pocket-path-vts-def polygon-at-least-3-vertices-wraparound psubsetI set-take-subset take-eq-Nil add-eq-0-iff-both-eq-0 add-gr-0 cancel-comm-monoid-add-class.diff-cancel diff-zero dual-order.strict-trans filled-p-def length-Cons length-tl less-imp-diff-less list.sel*(*3*) *list.size*(*3*) *not-less-eq-eq polygon-filled-p zero-less-one zero-neq-one*)
thus *card* (*set pocket-path-vts*) < *card* (*set vts*) **by** (*simp add*: *psubset-card-mono*)

have *card* (*set vts*) = *card* (*set* (*butlast vts*))
by (*smt* (*verit*,*del-insts*) *Cons-nth-drop-Suc List.finite-set One-nat-def Suc-1 Suc-le-lessD two-vts-on-frontier distinct-vts hd-last-vts frontier-vts-subset butlast.simps*(*1*) *butlast-conv-take card-insert-if card-length card-mono distinct-card drop0 drop-eq-Nil dual-order.trans last-in-set last-tl length-butlast length-greater-0-conv length-tl list.collapse list.sel*(*3*) *list.simps*(*15*) *set-take-subset verit-la-disequality*)
moreover have *length good-pocket-path-vts* ≥ *1*
unfolding *good-pocket-path-vts-def pocket-path-vts-def construct-pocket-0-def*
using *convex-hull-of-nonconvex-polygon-strict-subset*[*OF - assms*(*4*), *of vts*]
using *Suc-le-eq assms*(*1*) *assms*(*2*) *assms*(*3*) *construct-pocket-0-def construct-pocket-is-pocket*

is-pocket-0-def numeral-3-eq-3
by *auto*
ultimately show *card* (*set filled-vts*) < *card* (*set vts*)

unfolding *filled-vts-def fill-pocket-0-def good-pocket-path-vts-def pocket-path-vts-def*
by (*smt* (*verit*) *Nitpick.size-list-simp*(*2*) *Suc-1 Suc-diff-Suc Suc-n-not-le-n* ‹*2* ≤ *length filled-vts*› *distinct-vts hd-last-vts card-length diff-is-0-eq diff-less distinct-card drop-eq-Nil fill-pocket-0-def filled-vts-def insert-absorb last-drop last-in-set le leI le-less-Suc-eq length-Cons length-butlast length-drop length-tl less-imp-diff-less list.simps*(*15*) *order-less-le-trans pocket-path-vts-def*)
qed

29.3 Arbitrary Polygon Case

lemma *pick-rotate*:
assumes *polygon-of p vts*
assumes *all-integral vts*
obtains *p′ vts′* **where** *polygon-of p′ vts′*
∧ *vts′*!*0* ∈ *frontier* (*convex hull* (*set vts′*))
∧ *path-image p′* = *path-image p*
∧ *all-integral vts′*
∧ *set vts′* = *set vts*
proof−
obtain *v* **where** *v*: *v* ∈ *set vts* ∩ *frontier* (*convex hull* (*set vts*))
proof−
obtain *v* **where** *v* ∈ *set vts* ∧ *v extreme-point-of* (*convex hull* (*set vts*))
using *assms* **unfolding** *polygon-of-def*
by (*metis List.finite-set card.empty convex-convex-hull convex-hull-eq-empty extreme-point-exists-convex extreme-point-of-convex-hull finite-imp-compact-convex-hull not-numeral-le-zero polygon-at-least-3-vertices*)
then have *v* ∈ *set vts* ∧ *v* ∈ *frontier* (*convex hull* (*set vts*))
by (*metis Krein-Milman-frontier List.finite-set convex-convex-hull extreme-point-of-convex-hull finite-imp-compact-convex-hull*)
thus *?thesis* **using** *that* **by** *blast*
qed
obtain *i* **where** *i*: *vts*!*i* = *v* ∧ *i* < *length vts* **by** (*meson IntE in-set-conv-nth v*)
let *?vts-rotated* = *rotate-polygon-vertices vts i*
let *?p-rotated* = *make-polygonal-path ?vts-rotated*
have *same-set*: *set vts* = *set ?vts-rotated*
using *assms* **unfolding** *polygon-of-def*
using *rotate-polygon-vertices-same-set*
by *force*
moreover have ∗: *?vts-rotated*!*0* ∈ *frontier* (*convex hull* (*set ?vts-rotated*))
proof−
have *?vts-rotated*!*0* = *vts*!*i*
using *assms* **unfolding** *polygon-of-def*
by (*metis add-leD2 diff-self-eq-0 have-wraparound-vertex hd-conv-nth i last-snoc less-nat-zero-code list.size*(*3*) *nat-le-linear numeral-Bit0 polygon-vertices-length-at-least-4 rotated-polygon-vertices*)

moreover have *vts!i ∈ frontier (convex hull (set vts))* **using** *v i* **by** *blast*
ultimately show *?thesis* **using** *same-set* **by** *argo*
qed
moreover have *polygon ?p-rotated*
using *rotation-is-polygon assms* **unfolding** *polygon-of-def* **by** *blast*
moreover have *all-integral ?vts-rotated*
using *rotate-polygon-vertices-same-set assms*
unfolding *all-integral-def polygon-of-def* **by** *blast*
moreover have *path-image ?p-rotated = path-image p*
using *assms* **unfolding** *polygon-of-def* **using** *polygon-vts-arb-rotation* **by** *force*
moreover then have *path-inside ?p-rotated = path-inside p* **unfolding** *path-inside-def*
by *simp*
ultimately show *?thesis* **using** *polygon-of-def that* **by** *blast*
qed

lemma *pick-unrotated*:
fixes *p :: R-to-R2*
assumes *polygon*: *polygon p*
assumes *polygonal-path*: *p = make-polygonal-path vts*
assumes *int-vertices*: *all-integral vts*
assumes *I-is*: *I = card {x. integral-vec x ∧ x ∈ path-inside p}*
assumes *B-is*: *B = card {x. integral-vec x ∧ x ∈ path-image p}*
assumes *vts!0 ∈ frontier (convex hull (set vts))*
shows *measure lebesgue (path-inside p) = I + B/2 − 1*
using *assms*
proof (*induct card (set vts) arbitrary*: *vts p I B rule*: *less-induct*)
case *less*
have *B-finite*: *finite {x. integral-vec x ∧ x ∈ path-image p}*
using *finite-path-image less(2)* **by** *auto*
have *set vts ⊆ {x. integral-vec x ∧ x ∈ path-image p}*
using *less(3) vertices-on-path-image[of vts] less(4)*
unfolding *all-integral-def*
by *auto*
then have *card-vts*: *card (set vts) ≥ 3*
using *polygon-at-least-3-vertices[OF less(2) less(3)] card-mono order-trans*
by *blast*
have *vts-wraparound*: *vts ! 0 = vts ! (length vts − 1)*
using *less(2−3) polygon-pathstart polygon-pathfinish*
unfolding *polygon-def closed-path-def*
by (*metis diff-0-eq-0 length-0-conv*)
then have *vts-is*: *vts = (butlast vts) @ [vts ! 0]*
by (*metis butlast-conv-take have-wraparound-vertex less.prems(1) less.prems(2)*)
have *same-set*: *set vts = set (butlast (vts))*
by (*metis ListMem-iff Un-insert-right append.right-neutral butlast.simps(2) constant-linepath-is-not-loop-free elem hd-conv-nth insert-absorb less.prems(1) less.prems(2) list.collapse list.simps(15) make-polygonal-path.simps(2) polygon-def set-append simple-path-def vts-is*)
have *distinct-butlast-vts*: *distinct (butlast vts)*
using *simple-polygonal-path-vts-distinct less(2−3)*

unfolding *polygon-def*
by *auto*
have *card-butlast-vts*: *card* (*set vts*) = *card* (*set* (*butlast vts*))
using *vts-wraparound*
by (*smt* (*verit*, *best*) *List.finite-set butlast-conv-take card-distinct card-length card-mono card-vts diff-is-0-eq diff-less distinct-butlast-vts distinct-card drop-rev dual-order.strict-trans1 le-SucE length-append-singleton length-greater-0-conv less-numeral-extra*(*1*) *less-numeral-extra*(*4*) *nth-eq-iff-index-eq one-less-numeral-iff order-class.order-eq-iff semiring-norm*(*77*) *set-drop-subset set-rev vts-is*)
then have *card-set-len-butlast*: *card* (*set vts*) = *length* (*butlast vts*)
using *distinct-butlast-vts*
by (*metis distinct-card*)
{ assume *triangle*: *card* (*set vts*) = *3*
then have *length* (*butlast vts*) = *3*
using *card-set-len-butlast*
by *auto*
then have *butlast vts* = [*vts* ! *0*, *vts* ! *1*, *vts* ! *2*]
by (*metis* (*no-types*, *lifting*) *Cons-nth-drop-Suc One-nat-def Suc-1 card-set-len-butlast card-vts drop0 drop-eq-Nil lessI nth-append numeral-3-eq-3 one-less-numeral-iff semiring-norm*(*77*) *vts-is zero-less-numeral*)
then have *vts-is*: *vts* = [*vts* ! *0*, *vts* ! *1*, *vts* ! *2*, *vts* ! *0*]
using *vts-is* **by** *auto*
then have *p-make-triangle*: *p* = *make-triangle* (*vts* ! *0*) (*vts* ! *1*) (*vts* ! *2*)
using *less*(*3*) **unfolding** *make-triangle-def* **by** *simp*
then have *not-collinear*: ¬ *collinear* {*vts* ! *0*, *vts* ! *1*, *vts* ! *2*}
using *vts-is less*(*2*) *polygon-vts-not-collinear*[*of p vts*] **unfolding** *polygon-of-def make-triangle-def*
by (*smt* (*verit*, *ccfv-threshold*) *insert-absorb2 insert-commute list.set*(*1*) *list.simps*(*15*))
have *all-integral*: *all-integral* [*vts* ! *0*, *vts* ! *1*, *vts* ! *2*]
using *less.prems*(*3*) *vts-is* **unfolding** *all-integral-def*
by (*simp add*: ‹*butlast vts* = [*vts* ! *0*, *vts* ! *1*, *vts* ! *2*]› *in-set-butlastD*)
have *distinct*: *distinct* [*vts* ! *0*, *vts* ! *1*, *vts* ! *2*]
using ‹*butlast vts* = [*vts* ! *0*, *vts* ! *1*, *vts* ! *2*]› *distinct-butlast-vts* **by** *presburger*
have *pick-triangle*: *pick-triangle p* (*vts* ! *0*) (*vts* ! *1*) (*vts* ! *2*)
using *pick-triangle p-make-triangle less*(*2*) *not-collinear all-integral distinct*
by *simp*
then have *?case*
using *pick-triangle-lemma*[*OF p-make-triangle all-integral distinct not-collinear*] *less.prems*(*4*−*5*)
by *blast*
} moreover
{ assume *non-triangle*: *card* (*set vts*) > *3*
{ assume *convex*: *convex* (*path-image p* ∪ *path-inside p*)
then obtain *a b* **where** *good-linepath a b vts*
using *convex-polygon-has-good-linepath non-triangle*
by (*metis inf-sup-aci*(*5*) *less.prems*(*1*) *less.prems*(*2*))
then have *ab-prop*: *a* ≠ *b* ∧ {*a*, *b*} ⊆ *set vts* ∧ *path-image* (*linepath a b*) ⊆ *path-inside p* ∪ {*a*, *b*}

unfolding *good-linepath-def less.prems*(*2*) **by** *presburger*
then have *ab-prop-restate*: $a \neq b \wedge a \in set\ (butlast\ vts) \wedge b \in set\ (butlast\ vts)$
using *same-set*
by *simp*
have *good-linepath-ab*: *good-linepath a b* ((*butlast vts*) @ [(*butlast vts*) ! *0*])
using *ab-prop vts-is* **unfolding** *good-linepath-def*
using *ab-prop-restate empty-set hd-append2 hd-conv-nth insert-absorb insert-not-empty less.prems*(*2*) *same-set*
by (*smt* (*verit*))
then have *good-linepath-ba*: *good-linepath b a* ((*butlast vts*) @ [(*butlast vts*) ! *0*])
using *good-linepath-comm good-linepath-def* **by** *blast*
obtain *i1 j1* **where** *ij-prop*: $i1 < length\ (butlast\ vts) \wedge j1 < length\ (butlast\ vts) \wedge$
$butlast\ vts\ !\ i1 = a \wedge$
$butlast\ vts\ !\ j1 = b \wedge i1 \neq j1$
using *ab-prop-restate*
by (*metis distinct-Ex1 distinct-butlast-vts*)
have *i-lt-then*: $i1 < j1 \Longrightarrow is\text{-}polygon\text{-}split\ (butlast\ vts)\ i1\ j1$
using *good-linepath-implies-polygon-split*[*OF less*(*2*), *of butlast vts*] *vts-is same-set*
using *ij-prop good-linepath-ab good-linepath-ba*
by (*metis ab-prop-restate length-pos-if-in-set less.prems*(*2*) *nth-butlast*)
have *j-lt-then*: $j1 < i1 \Longrightarrow is\text{-}polygon\text{-}split\ (butlast\ vts)\ j1\ i1$
using *good-linepath-implies-polygon-split*[*OF less*(*2*), *of butlast vts*] *vts-is same-set*
using *ij-prop good-linepath-ab good-linepath-ba*
by (*metis ab-prop-restate length-pos-if-in-set less.prems*(*2*) *nth-butlast*)
obtain *i j* **where** *polygon-split*: *is-polygon-split* (*butlast vts*) *i j*
using *i-lt-then j-lt-then ij-prop*
by (*meson nat-neq-iff*)
then have *ij-prop*: $i < length\ (butlast\ vts) \wedge j < length\ (butlast\ vts) \wedge i < j$
unfolding *is-polygon-split-def*
by *blast*

have *p-is*: $p = make\text{-}polygonal\text{-}path\ (butlast\ vts\ @\ [butlast\ vts\ !\ 0])$
using *less*(*3*) *vts-is*
by (*metis length-greater-0-conv nth-butlast same-set set-empty*)

let *?vts1* = *take i* (*butlast vts*)
let *?vts2* = *take* ($j - i - 1$) (*drop* (*Suc i*) (*butlast vts*))
let *?vts3* = *drop* ($j - i$) (*drop* (*Suc i*) (*butlast vts*))

let *?vtsp1* = (*butlast vts* ! *i* # *?vts2* @ [*butlast vts* ! *j*, *butlast vts* ! *i*])
have *finite-butlast*: *finite* (*set* (*butlast vts*))
by *blast*
have *vtsp1-subset*: *set ?vtsp1* $\subseteq$ *set* (*butlast vts*)

using *ij-prop*
by (*smt* (*verit*, *del-insts*) *Un-commute append-Cons append-Nil dual-order.trans insert-subset list.simps*(15) *nth-mem set-append set-drop-subset set-take-subset*)

let *?p1* = *make-polygonal-path ?vtsp1*
let *?I1* = *card* {*x*. *integral-vec x* ∧ *x* ∈ *path-inside ?p1*}
let *?B1* = *card* {*x*. *integral-vec x* ∧ *x* ∈ *path-image ?p1*}
have *polygon-p1*: *polygon ?p1*
using *polygon-split* **unfolding** *is-polygon-split-def* **by** *metis*

let *?vtsp2* = *?vts1* @ [*butlast vts* ! *i*, *butlast vts* ! *j*] @ *?vts3* @ [*butlast vts* ! *0*]
let *?p2* = *make-polygonal-path ?vtsp2*
have *polygon-p2*: *polygon ?p2*
using *polygon-split* **unfolding** *is-polygon-split-def* **by** *metis*

have *j-neq*: *j* ≠ *i* + *1*
by (*smt* (*verit*, *ccfv-SIG*) *One-nat-def Suc-n-not-le-n Suc-numeral add-Suc-shift add-implies-diff cancel-ab-semigroup-add-class.diff-right-commute length-Cons length-append list.size*(3) *numeral-3-eq-3 plus-1-eq-Suc polygon-p1 polygon-vertices-length-at-least-4 semiring-norm*(2) *semiring-norm*(8) *take-eq-Nil*)
have *subset1*: *set* (*take i* (*butlast vts*)) ⊆ *set* (*butlast vts*)
using *ij-prop* **by** (*meson set-take-subset*)
have *subset2*: *set* ([*butlast vts* ! *i*, *butlast vts* ! *j*]) ⊆ *set* (*butlast vts*)
using *ij-prop* **by** *simp*
have *subset3*: *set* (*take i* (*butlast vts*) @
[*butlast vts* ! *i*, *butlast vts* ! *j*]) ⊆ *set* (*butlast vts*)
using *subset1 subset2* **by** *auto*
have *subset4*: *set* (*drop* (*j* − *i*) (*drop* (*Suc i*) (*butlast vts*)) @ [*butlast vts* ! *0*])
⊆ *set* (*butlast vts*)
using *ij-prop set-drop-subset*
by (*metis* (*no-types*, *opaque-lifting*) *Un-commute append-Cons append-Nil card-set-len-butlast drop0 drop-drop drop-eq-Nil2 hd-append2 hd-conv-nth in-set-conv-decomp insert-subset linorder-not-less list.simps*(15) *non-triangle not-less-eq not-less-iff-gr-or-eq numeral-3-eq-3 same-set set-append snoc-eq-iff-butlast vts-is*)
then have *main-subset*: *set ?vtsp2* ⊆ *set* (*butlast vts*)
using *subset3 subset4* **by** *simp*

have *subset-p1*: *set ?vtsp1* ⊂ *set* (*butlast vts*)
using *ij-prop distinct-butlast-vts*
proof−
have *card* (*set ?vtsp2*) ≥ *3*
using *polygon-p2 polygon-at-least-3-vertices* **by** *blast*
moreover have *set ?vtsp1* ∩ *set ?vtsp2* = {*vts*!*i*, *vts*!*j*}
proof−
have *set ?vts2* ∩ *set ?vts3* = {}
by (*metis append-take-drop-id diff-le-self distinct-append distinct-butlast-vts set-take-disj-set-drop-if-distinct*)
moreover have *set ?vts2* ∩ *set ?vts1* = {}

proof−
have *set ?vts2 ⊆ set (drop (i + 1) vts)*
by (*metis add.commute drop-butlast in-set-butlastD in-set-takeD plus-1-eq-Suc subset-code(1)*)
moreover have *set (drop (i + 1) vts) ∩ set ?vts1 ⊆ {last vts}*
proof−
have *set (drop (i + 1) (butlast vts)) ∩ set ?vts1 = {}*
by (*simp add*: *Int-commute set-take-disj-set-drop-if-distinct distinct-butlast-vts*)
moreover have *set (drop (i + 1) vts) = set (drop (i + 1) (butlast vts)) ∪ {last vts}*
proof−
have *drop (i + 1) vts = (drop (i + 1) ((butlast vts) @ [last vts]))*
by (*metis last-snoc vts-is*)
thus *?thesis* **using** *ij-prop* **by** *force*
qed
ultimately show *?thesis* **by** *blast*
qed
moreover have *last vts ∉ set ?vts2*
by (*metis card-set-len-butlast card-vts distinct-butlast-vts dual-order.strict-trans1 in-set-takeD index-nth-id last-snoc nth-butlast numeral-3-eq-3 set-drop-if-index vts-is zero-less-Suc*)
ultimately show *?thesis* **by** *force*
qed
moreover have *vts!i ∈ set ?vtsp1* **by** (*metis ij-prop list.set-intros(1) nth-butlast*)
moreover have *vts!j ∈ set ?vtsp1* **using** *ij-prop nth-butlast* **by** *fastforce*
moreover have *vts!i ∈ set ?vtsp2*
by (*metis UnCI ij-prop list.set-intros(1) nth-butlast set-append*)
moreover have *vts!j ∈ set ?vtsp2* **using** *ij-prop nth-butlast* **by** *force*
moreover have *set ?vtsp1 = set ?vts2 ∪ {vts!i, vts!j}*
by (*smt (verit, ccfv-SIG) Un-insert-right empty-set ij-prop insert-absorb2 insert-commute list.simps(15) nth-butlast set-append*)
moreover have *set ?vtsp2 = set ?vts1 ∪ set ?vts3 ∪ {vts!i, vts!j, vts!0}*
proof−
have *vts!i = (butlast vts)!i* **by** (*metis ij-prop nth-butlast*)
moreover have *vts!j = (butlast vts)!j* **by** (*metis ij-prop nth-butlast*)
moreover have *vts!0 = (butlast vts)!0*
by (*metis ij-prop leD length-greater-0-conv nth-butlast take-all-iff take-eq-Nil*)
ultimately show *?thesis* **by** *force*
qed
moreover have *vts!0 ∉ set ?vts2*
by (*metis distinct-butlast-vts in-set-conv-decomp in-set-takeD index-nth-id length-pos-if-in-set nth-butlast same-set set-drop-if-index vts-is zero-less-Suc*)
ultimately show *?thesis* **by** *blast*
qed
ultimately have *card (set ?vtsp2) > card (set ?vtsp1 ∩ set ?vtsp2)*
by (*smt (verit, del-insts) card-length empty-set leI le-trans length-Cons*

list.simps(15) list.size(3) not-less-eq-eq numeral-3-eq-3)
then have $\exists v.\ v \in set\ ?vtsp2 \wedge v \notin (set\ ?vtsp1 \cap set\ ?vtsp2)$
by (*smt* (*verit*) *Int-lower2 Orderings.order-eq-iff less-not-refl subset-code(1)*)
then obtain *v* **where** $v \in set\ ?vtsp2 - set\ ?vtsp1$ **by** *blast*
thus *?thesis*
by (*metis main-subset Diff-eq-empty-iff length-pos-if-in-set less-numeral-extra(3) list.set(1) list.size(3) psubsetI vtsp1-subset*)
qed
then have *card* (*set ?vtsp1*) < *card* (*set* (*butlast vts*))
using *card-subset-eq*[*OF finite-butlast*]
by (*meson finite-butlast psubset-card-mono*)
then have *card-lt-p1*: *card* (*set ?vtsp1*) < *card* (*set vts*)
using *same-set* **by** *argo*
have *set ?vtsp1* ⊆ *set vts*
using *ij-prop*
using *same-set subset-p1* **by** *blast*
then have *all-integral-p1*: *all-integral ?vtsp1*
using *less(4)* **unfolding** *all-integral-def*
by *blast*

obtain *p1' vtsp1'* **where** *p1-rot*: *polygon-of p1' vtsp1'*
∧ *vtsp1'*!0 ∈ *frontier* (*convex hull* (*set vtsp1'*))
∧ *path-image p1' = path-image ?p1*
∧ *all-integral vtsp1'*
∧ *set vtsp1' = set ?vtsp1*
using *pick-rotate less polygon-p1* **unfolding** *polygon-of-def*
using *all-integral-p1*
by *blast*

let *?I1' = card* {*x. integral-vec x* ∧ *x* ∈ *path-inside p1'*}
let *?B1' = card* {*x. integral-vec x* ∧ *x* ∈ *path-image p1'*}

have *measure lebesgue* (*path-inside p1'*) = *real ?I1'* + *real ?B1'* / *2* − *1*
using *less(1) polygon-split card-lt-p1 p1-rot* **unfolding** *polygon-of-def* **by** *force*
then have *indh1*: *Sigma-Algebra.measure lebesgue* (*path-inside ?p1*) = *real ?I1* + *real ?B1* / *2* − *1*
using *p1-rot* **unfolding** *path-inside-def* **by** *metis*

have *vts* ! (*i*+*1*) ∉ *set* (*take i* (*butlast vts*))
using *distinct-butlast-vts j-neq ij-prop*
proof−
have *i* + *1* < *length vts* − *2* **using** *distinct-butlast-vts j-neq ij-prop* **by** *fastforce*
then have *vts* ! (*i*+*1*) = (*butlast vts*) ! (*i*+*1*) **by** (*simp add*: *nth-butlast*)
moreover then have ∀ *j* < *i* + *1*. (*butlast vts*) ! *j* ≠ (*butlast vts*) ! (*i*+*1*)
using *distinct-butlast-vts distinct-nth-eq-iff ij-prop* **by** *fastforce*
moreover have *set* (*take i* (*butlast vts*)) = {*vts*!*j* | *j*. *j* < *i*}
proof−

have *set* (*take i* (*butlast vts*)) ⊆ {*vts*!*j* | *j*. *j* < *i*}
by (*smt* (*verit*, *ccfv-SIG*) *dual-order.strict-trans ij-prop in-set-conv-nth length-take mem-Collect-eq min.absorb4 nth-butlast nth-take subsetI*)
moreover have {*vts*!*j* | *j*. *j* < *i*} ⊆ *set* (*take i* (*butlast vts*))
by (*smt* (*verit*, *del-insts*) *dual-order.strict-trans ij-prop in-set-conv-nth length-take mem-Collect-eq min.absorb4 nth-butlast nth-take subsetI*)
ultimately show *?thesis* **by** *blast*
qed
ultimately show *?thesis*
by (*metis* (*no-types*, *lifting*) *add.commute ij-prop in-set-conv-nth length-take min.absorb4 nth-take trans-less-add2*)
qed
moreover have *vts* ! (*i*+*1*) ≠ *butlast vts* ! *i*
by (*metis* (*no-types*, *lifting*) *ij-prop add.commute add-cancel-right-right distinct-butlast-vts distinct-nth-eq-iff less-trans-Suc nth-append plus-1-eq-Suc vts-is zero-neq-one*)
moreover have *vts* ! (*i*+*1*) ≠ *butlast vts* ! *j*
by (*metis* (*no-types*, *lifting*) *add.commute distinct-butlast-vts distinct-nth-eq-iff ij-prop j-neq less-trans-Suc nth-append plus-1-eq-Suc vts-is*)
ultimately have *vts* ! (*i*+*1*) ∉ *set* (*take i* (*butlast vts*) @
[*butlast vts* ! *i*, *butlast vts* ! *j*]) **by** *force*
moreover have *vts* ! (*i*+*1*) ∉ *set* (*drop* (*j* − *i*) (*drop* (*Suc i*) (*butlast vts*)) @ [*butlast vts* ! *0*])
proof−
have *vts* ! (*i*+*1*) ∉ *set* (*drop* (*j* − *i* + *Suc i*) (*butlast vts*))
by (*metis* (*no-types*, *lifting*) *add.commute distinct-butlast-vts ij-prop index-nth-id less-add-same-cancel2 less-trans-Suc nth-append plus-1-eq-Suc set-drop-if-index vts-is zero-less-diff*)
moreover have *vts* ! (*i*+*1*) ≠ *butlast vts* ! *0*
by (*metis* (*no-types*, *lifting*) *ij-prop Nil-is-append-conv add.commute distinct-butlast-vts distinct-nth-eq-iff length-greater-0-conv less-trans-Suc list.discI nat.distinct*(*1*) *nth-append plus-1-eq-Suc same-set set-empty vts-is*)
ultimately show *?thesis* **by** *simp*
qed
ultimately have *vts* ! (*i*+*1*) ∉ *set* (*take i* (*butlast vts*) @
[*butlast vts* ! *i*, *butlast vts* ! *j*] @
drop (*j* − *i*) (*drop* (*Suc i*) (*butlast vts*)) @ [*butlast vts* ! *0*])
by *auto*
then have *subset-butlast-p2*: *set ?vtsp2* ⊂ *set* (*butlast vts*)
using *main-subset ij-prop*
by (*metis* (*no-types*, *lifting*) *antisym-conv2 length-butlast less-diff-conv nth-mem same-set*)
then have *card-lt-p2*: *card* (*set ?vtsp2*) < *card* (*set vts*)
using *card-subset-eq*[*OF finite-butlast*]
by (*metis finite-butlast psubset-card-mono same-set*)
have *subset-p2*: *set ?vtsp2* ⊂ *set vts*
using *subset-butlast-p2 same-set*
by *presburger*
then have *all-integral-p2*: *all-integral ?vtsp2*

using *less*(*4*) **unfolding** *all-integral-def*
by *blast*

let *?p2* = *make-polygonal-path* (*take i* (*butlast vts*) @ [*butlast vts* ! *i*, *butlast vts* ! *j*] @
drop (*j* − *i*) (*drop* (*Suc i*) (*butlast vts*)) @ [*butlast vts* ! *0*])
let *?I2* = *card* {*x*. *integral-vec x* ∧ *x* ∈ *path-inside ?p2*}
let *?B2* = *card* {*x*. *integral-vec x* ∧ *x* ∈ *path-image ?p2*}
have *polygon-p2*: *polygon ?p2*
using *polygon-split* **unfolding** *is-polygon-split-def* **by** *metis*

have *vtsp2-0*: *?vtsp2*!*0* ∈ *frontier* (*convex hull* (*set ?vtsp2*))
proof−
have *?vtsp2*!*0* = *vts*!*0*
by (*metis* (*no-types*, *lifting*) *append-Cons ij-prop length-greater-0-conv less-nat-zero-code nat-neq-iff nth-append nth-append-length nth-butlast nth-take take-eq-Nil*)
then have *?vtsp2*!*0* ∈ *frontier* (*convex hull* (*set vts*)) **using** *less* **by** *argo*
moreover have *?vtsp2*!*0* ∈ (*convex hull* (*set ?vtsp2*))
by (*meson append-is-Nil-conv hull-inc length-greater-0-conv neq-Nil-conv nth-mem*)
moreover have *convex hull* (*set ?vtsp2*) ⊆ *convex hull* (*set vts*)
by (*metis hull-mono main-subset same-set*)
ultimately show *?thesis* **using** *in-frontier-in-subset* **by** *blast*
qed

have *indh2*: *Sigma-Algebra.measure lebesgue* (*path-inside ?p2*) = *real ?I2* + *real ?B2* / *2* − *1*
using *less*(*1*)[*OF card-lt-p2 polygon-p2 - all-integral-p2 - - vtsp2-0*] *polygon-split*
by *blast*

have *all-integral* (*butlast vts*) ⟹
Sigma-Algebra.measure lebesgue (*path-inside p*) = *real* (*card* {*x*. *integral-vec x* ∧ *x* ∈ *path-inside p*}) + *real* (*card* {*x*. *integral-vec x* ∧ *x* ∈ *path-image p*}) / *2* − *1*
using *pick-split-union*
[*OF polygon-split*, *of ?vts1 ?vts2 ?vts3 butlast vts* ! *i butlast vts* ! *j p ?p1 ?p2 ?I1 ?B1 ?I2 ?B2*]
using *indh1 indh2 p-is*
by *blast*
then have *?case*
using *less*(*4*−*6*) **unfolding** *all-integral-def*
using *same-set* **by** *presburger*
} moreover
{ assume *non-convex*: ¬ (*convex* (*path-image p* ∪ *path-inside p*))
let *?vts-ch* = *set vts* ∩ *frontier* (*convex hull* (*set vts*))
have *finite-vts*: *finite* (*set vts*)
using *less*
by *force*

have *subset-ch*: *?vts-ch* ⊂ *set vts*
using *vts-subset-frontier*
using *less.prems*(*1*) *less.prems*(*2*) *non-convex polygon-of-def* **by** *blast*
then have *card-ch*: *card* (*?vts-ch*) < *card* (*set vts*)
using *finite-vts*
by (*simp add*: *psubset-card-mono*)

let *?vts-ch-list* = *filter* (λ*v*. *v* ∈ *?vts-ch*) *vts*

let *?r-idx* = *min-index-not-in-set vts ?vts-ch*
let *?r* = *?r-idx* − *1*
let *?rotated-vts* = *rotate-polygon-vertices vts ?r*
let *?pr* = *make-polygonal-path ?rotated-vts*

have *subset-ch-list*: *set ?vts-ch-list* ⊂ *set vts* **using** *subset-ch* **by** *auto*
then have *r-defined*: *index-not-in-set vts ?vts-ch ?r-idx*
∧ (∀ *j* < *?r-idx*. ¬ *index-not-in-set vts ?vts-ch j*)
using *min-index-not-in-set-defined*[*of ?vts-ch vts*] **by** *fastforce*

have *pr-image*: *path-image p* = *path-image ?pr*
using *polygon-vts-arb-rotation less* **by** *blast*
then have *measure lebesgue* (*path-inside ?pr*) = *measure lebesgue* (*path-inside p*)
unfolding *path-inside-def* **by** *presburger*
have *rotated-vts-set*: *set ?rotated-vts* = *set vts*
using *less.prems*(*1*) *less.prems*(*2*) *rotate-polygon-vertices-same-set* **by** *auto*
then have *card* (*set ?rotated-vts*) = *card* (*set vts*) **by** *argo*
have *polygon-rotation*: *polygon ?pr* **using** *rotation-is-polygon less* **by** *blast*

let *?pocket-path-vts* = *construct-pocket-0 ?rotated-vts ?vts-ch*

let *?a* = *hd ?pocket-path-vts*
let *?b* = *last ?pocket-path-vts*
let *?l* = *linepath ?a ?b*

have *vts*!*0* ∈ *?vts-ch*
by (*metis IntI length-greater-0-conv less.prems*(*6*) *nth-mem snoc-eq-iff-butlast vts-is*)
then have *vts-r*: *vts*!*?r* ∈ *?vts-ch*
using *min-index-not-in-set-0 subset-ch* **by** *presburger*
moreover have *rotated-0*: *?rotated-vts*!*0* = *vts*!*?r*
using *rotated-polygon-vertices*[*of ?rotated-vts vts ?r ?r*]
by (*metis* (*no-types*, *lifting*) *Suc-1 Suc-leI card-gt-0-iff card-set-len-butlast diff-is-0-eq*′ *finite-vts hd-conv-nth index-not-in-set-def le-refl length-butlast less-imp-diff-less mem-Collect-eq r-defined set-empty snoc-eq-iff-butlast vts-is zero-less-diff*)
ultimately have *rotated-0-in*: *?rotated-vts*!*0* ∈ *?vts-ch* **by** *presburger*
then have *b-in*: *?b* ∈ *set vts*

using *construct-pocket-0-last-in-set*[*of ?rotated-vts ?vts-ch*]
by (*smt* (*verit*, *ccfv-threshold*) *Int-iff One-nat-def closed-path-def Suc-leI card-0-eq card-set-len-butlast empty-iff finite-vts last-conv-nth last-in-set last-tl length-butlast length-greater-0-conv length-tl list.size*(*3*) *polygon-def polygon-pathfinish polygon-pathstart polygon-rotation rotate-polygon-vertices-same-length set-empty*)

have *2* ≤ *card ?vts-ch*
using *convex-hull-two-vts-on-frontier*
by (*metis One-nat-def Suc-1 add-leD2 card-vts numeral-3-eq-3 plus-1-eq-Suc*)
moreover have *?vts-ch* ⊆ *set ?rotated-vts*
using *less.prems*(*1*) *less.prems*(*2*) *rotate-polygon-vertices-same-set* **by** *force*
moreover have *distinct* (*butlast ?rotated-vts*)
using *polygon-def polygon-rotation simple-polygonal-path-vts-distinct* **by** *blast*
moreover have *hd-last-rotated*: *hd ?rotated-vts* = *last ?rotated-vts*
by (*metis have-wraparound-vertex hd-conv-nth polygon-rotation snoc-eq-iff-butlast*)
ultimately have *a-neq-b*: *?a* ≠ *?b*
using *construct-pocket-0-first-last-distinct*
by (*smt* (*verit*) *Collect-cong Int-def mem-Collect-eq set-filter*)

let *?pocket-vts* = *?pocket-path-vts* @ [*?rotated-vts*!*0*]

let *?pocket-good-path-vts* = *tl* (*butlast ?pocket-path-vts*)

let *?filled-vts* = *fill-pocket-0 ?rotated-vts* (*length ?pocket-path-vts*)
let *?filled-vts-tl* = *tl ?filled-vts*
let *?filled-p-tl* = *make-polygonal-path ?filled-vts-tl*
let *?filled-p* = *make-polygonal-path ?filled-vts*
let *?pocket-path* = *make-polygonal-path ?pocket-path-vts*
let *?pocket* = *make-polygonal-path ?pocket-vts*

have *non-convex-rot*: ¬ *convex* (*path-image ?pr* ∪ *path-inside ?pr*)
using *non-convex* **by** (*simp add*: *path-inside-def pr-image*)

have *0*: *?rotated-vts*!*0* ∈ *frontier* (*convex hull* (*set ?rotated-vts*))
using *less.prems*(*1*) *less.prems*(*2*) *rotate-polygon-vertices-same-set rotated-0-in* **by** *fastforce*
have *1*: *?rotated-vts*!*1* ∉ *frontier* (*convex hull* (*set ?rotated-vts*))
proof−
have *?rotated-vts*!*1* = *vts*!(*?r* + *1*)
using *rotated-polygon-vertices*[*of ?rotated-vts vts ?r ?r* + *1*]
by (*smt* (*verit*, *ccfv-threshold*) *Suc-1 Suc-leI card-gt-0-iff card-set-len-butlast diff-is-0-eq*′ *finite-vts hd-conv-nth index-not-in-set-def le-refl length-butlast less-imp-diff-less mem-Collect-eq r-defined set-empty snoc-eq-iff-butlast vts-is zero-less-diff Suc-diff-Suc add.commute add-diff-cancel-left*′ *bot-nat-0.not-eq-extremum less-imp-le-nat plus-1-eq-Suc*)
also have ... ∉ *frontier* (*convex hull* (*set ?rotated-vts*))
using *r-defined* **unfolding** *index-not-in-set-def*

by (*smt* (*verit*, *best*) *Int-iff Suc-leI add.commute add-diff-inverse-nat bot-nat-0.not-eq-extremum diff-is-0-eq' mem-Collect-eq nat-less-le nth-mem plus-1-eq-Suc rotated-vts-set vts-r zero-less-diff*)
finally show *?thesis* **.**
qed
then have *split*:
is-polygon-split-path (*butlast ?filled-vts*) *0 1 ?pocket-good-path-vts*
and *polygon-filled-p*: *polygon ?filled-p*
and *polygon-pocket*: *polygon ?pocket*
and *pocket-path-vts-card*: *card* (*set ?pocket-path-vts*) < *card* (*set vts*)
and *filled-vts-card*: *card* (*set ?filled-vts*) < *card* (*set vts*)
using *pocket-path-good*[*OF - 0 1 non-convex-rot*] *polygon-rotation rotated-vts-set* **apply** *argo*
using *pocket-path-good*[*OF - 0 1 non-convex-rot*] *polygon-rotation rotated-vts-set* **apply** *argo*
using *pocket-path-good*[*OF - 0 1 non-convex-rot*] *polygon-rotation rotated-vts-set*
apply (*metis add-gr-0 construct-pocket-0-def nth-take zero-less-one*)
using *pocket-path-good*[*OF - 0 1 non-convex-rot*] *polygon-rotation rotated-vts-set* **apply** *argo*
using *pocket-path-good*[*OF - 0 1 non-convex-rot*] *polygon-rotation rotated-vts-set* **by** *argo*

have *vts-0-frontier*: *?rotated-vts*!*0* ∈ *frontier* (*convex hull* (*set vts*))
using *rotated-0-in* **by** *simp*
have *filled-0*: *?filled-vts*!*0* = *?rotated-vts*!*0*
by (*metis convex-hull-empty empty-set fill-pocket-0-def frontier-empty hd-conv-nth length-pos-if-in-set less.prems*(*6*) *less-numeral-extra*(*3*) *list.size*(*3*) *nth-Cons-0 rotated-vts-set*)
have *pocket-0*: *?pocket-vts*!*0* = *?rotated-vts*!*0*
unfolding *construct-pocket-0-def*
by (*simp add*: *less-numeral-extra*(*1*) *nth-append trans-less-add2*)

have *subset-pocket-path-vts*: *set ?pocket-path-vts* ⊆ *set vts*
using *construct-pocket-0-subset-vts*
by (*metis construct-pocket-0-def less.prems*(*1*) *less.prems*(*2*) *rotate-polygon-vertices-same-set set-take-subset*)
moreover have *set ?pocket-good-path-vts* ⊆ *set ?pocket-path-vts*
by (*smt* (*verit*, *best*) *butlast-conv-take list.exhaust-sel list.sel*(*2*) *set-subset-Cons set-take-subset subset-trans*)
ultimately have *subset-pocket-good-path*: *set ?pocket-good-path-vts* ⊆ *set vts*
by *blast*
then have *subset-pocket*: *set ?pocket-vts* ⊆ *set vts*
by (*metis* (*mono-tags*, *lifting*) *have-wraparound-vertex less.prems*(*1*) *less.prems*(*2*) *polygon-rotation rotate-polygon-vertices-same-set set-append subset-code*(*1*) *subset-pocket-path-vts sup.bounded-iff*)
have *set ?filled-vts* ⊆ *set ?rotated-vts*
unfolding *fill-pocket-0-def*
by (*metis b-in hd-in-set insert-subset length-pos-if-in-set less-numeral-extra*(*3*)

list.simps(*15*) *list.size*(*3*) *rotated-vts-set set-drop-subset*)
then have *subset-filled*: *set ?filled-vts* ⊆ *set vts*
using *rotated-vts-set* **by** *blast*

have *taut1*: *?filled-p* = *make-polygonal-path ?filled-vts* **by** *blast*
have *all-integral-filled-vts*: *all-integral ?filled-vts*
using *subset-filled less* **by** (*meson all-integral-def subset-iff*)
have *taut2*: *card* (*integral-inside ?filled-p*) = *card* {*x*. *integral-vec x* ∧ *x* ∈ *path-inside ?filled-p*}
unfolding *integral-inside* **by** *blast*
have *taut3*: *card* (*integral-boundary ?filled-p*) = *card* {*x*. *integral-vec x* ∧ *x* ∈ *path-image ?filled-p*}
unfolding *integral-boundary* **by** *blast*
have *filled-vts-0-frontier*: *?filled-vts*!*0* ∈ *frontier* (*convex hull* (*set ?filled-vts*))
proof−
have *?filled-vts*!*0* ∈ *frontier* (*convex hull set vts*)
using *filled-0 vts-0-frontier* **by** *presburger*
moreover have *?filled-vts*!*0* ∈ *convex hull* (*set ?filled-vts*)
by (*metis have-wraparound-vertex hull-inc in-set-conv-decomp polygon-filled-p*)
moreover have *set ?filled-vts* ⊆ *set vts* **using** *subset-filled* **by** *force*
ultimately show *?thesis* **using** *in-frontier-in-subset-convex-hull* **by** *blast*
qed

have *ih-filled*: *measure lebesgue* (*path-inside ?filled-p*)
= *card* (*integral-inside ?filled-p*) + ((*card* (*integral-boundary ?filled-p*)) / *2*) − *1*
using *less*(*1*)[*OF filled-vts-card polygon-filled-p taut1 all-integral-filled-vts taut2 taut3 filled-vts-0-frontier*]
by *blast*

have *set ?pocket-path-vts* ⊂ *set vts*
using *pocket-path-vts-card subset-pocket-path-vts* **by** *force*
moreover have *pocket-path-set*: *set ?pocket-path-vts* = *set ?pocket-vts*
by (*smt* (*verit*) *Nil-is-append-conv rotated-0 a-neq-b append-Cons append-Nil hd-Nil-eq-last hd-append2 hd-conv-nth hd-in-set insert-absorb list.simps*(*15*) *pocket-0 rev-append set-append set-rev*)
ultimately have *set ?pocket-vts* ⊂ *set vts* **by** *blast*
then have *pocket-vts-card*: *card* (*set ?pocket-vts*) < *card* (*set vts*)
by (*meson finite-vts psubset-card-mono*)
have *all-integral-pocket-vts*: *all-integral ?pocket-vts*
using *subset-pocket less* **unfolding** *all-integral-def* **by** *blast*
have *taut1*: *?pocket* = *make-polygonal-path ?pocket-vts* **by** *blast*
have *taut2*: *card* (*integral-inside ?pocket*) = *card* {*x*. *integral-vec x* ∧ *x* ∈ *path-inside ?pocket*}
unfolding *integral-inside* **by** *blast*
have *taut3*: *card* (*integral-boundary ?pocket*) = *card* {*x*. *integral-vec x* ∧ *x* ∈ *path-image ?pocket*}

unfolding *integral-boundary* **by** *blast*
have *pocket-vts-0-frontier*: *?pocket-vts!0 ∈ frontier (convex hull (set ?pocket-vts))*
proof−
have *?pocket-vts!0 ∈ frontier (convex hull set vts)*
using *pocket-0 vts-0-frontier* **by** *presburger*
moreover have *?pocket-vts!0 ∈ convex hull (set ?pocket-vts)*
by (*smt* (*verit*, *del-insts*) *hull-inc in-set-conv-decomp pocket-0*)
moreover have *set ?pocket-vts ⊆ set vts* **using** *subset-pocket* **by** *force*
ultimately show *?thesis* **using** *in-frontier-in-subset-convex-hull* **by** *blast*
qed

have *ih-pocket*: *measure lebesgue (path-inside ?pocket) = card (integral-inside ?pocket) + ((card (integral-boundary ?pocket)) / 2) − 1*
using *less*(*1*)[*OF pocket-vts-card polygon-pocket taut1 all-integral-pocket-vts taut2 taut3 pocket-vts-0-frontier*]
by *blast*

let *?i = 0*::*nat*
let *?j = 1*::*nat*
let *?vts = butlast ?filled-vts*
let *?vts1 = []*
let *?vts2 = []*
let *?vts3 = butlast (drop 2 ?filled-vts)*
let *?cutvts = ?pocket-good-path-vts*
let *?p = ?filled-p*
let *?p1 = make-polygonal-path (?a # ?vts2 @ [?b] @ rev ?cutvts @ [?a])*
let *?p2 = ?pr*
let *?I1 = card {x. integral-vec x ∧ x ∈ path-inside ?p1}*
let *?B1 = card {x. integral-vec x ∧ x ∈ path-image ?p1}*
let *?I2 = card {x. integral-vec x ∧ x ∈ path-inside ?p2}*
let *?B2 = card {x. integral-vec x ∧ x ∈ path-image ?p2}*
let *?I = card {x. integral-vec x ∧ x ∈ path-inside ?p}*
let *?B = card {x. integral-vec x ∧ x ∈ path-image ?p}*

have *rev ?pocket-vts = (?a # ?vts2 @ [?b] @ rev ?cutvts @ [?a])*
by (*smt* (*verit*) *a-neq-b append-Nil append-butlast-last-id hd-Nil-eq-last hd-append2 hd-conv-nth last-conv-nth length-butlast list.collapse list.size*(*3*) *pocket-0 rev.simps*(*2*) *rev-append rev-rev-ident snoc-eq-iff-butlast*)
then have *pocket-rev-image*: *path-image ?pocket = path-image ?p1*
using *polygon-at-least-3-vertices polygon-pocket card-length*
by (*smt* (*verit*, *best*) *One-nat-def Suc-1 le-add2 le-trans numeral-3-eq-3 plus-1-eq-Suc rev-vts-path-image polygon-at-least-3-vertices polygon-pocket card-length*)
then have *pocket-rev-inside*: *path-inside ?pocket = path-inside ?p1*
unfolding *path-inside-def* **by** *argo*

have *split'*: *is-polygon-split-path ?vts ?i ?j ?cutvts* **using** *split* **by** *blast*
have *0*: *?vts1 = take ?i ?vts* **by** *auto*
have *1*: *?vts2 = take (?j − ?i − 1) (drop (Suc ?i) ?vts)* **by** *simp*
have *2*: *?vts3 = drop (?j − ?i) (drop (Suc ?i) ?vts)*

by (*metis* (*no-types*, *lifting*) *One-nat-def Suc-1 diff-zero drop-butlast drop-drop plus-1-eq-Suc*)
have *3*: *?a* = *?vts* ! *?i*
by (*smt* (*verit*) *Nil-is-append-conv pocket-path-set filled-0 hd-conv-nth is-polygon-split-path-def length-greater-0-conv list.distinct*(*1*) *nth-append nth-butlast pocket-0 set-empty split'*)
have *4*: *?b* = *?vts* ! *?j*
proof−
have *?b* = *?filled-vts*!*1*
unfolding *construct-pocket-0-def fill-pocket-0-def*
by (*smt* (*verit*) *Suc-eq-plus1 a-neq-b construct-pocket-0-def diff-Suc-1 diff-is-0-eq' drop-eq-Nil hd-conv-nth hd-drop-conv-nth hd-last-rotated last-conv-nth length-take linorder-not-less min.absorb4 nat-le-linear not-less-eq-eq nth-Cons' nth-take one-neq-zero take-all-iff take-eq-Nil*)
thus *?thesis* **by** (*metis is-polygon-split-path-def nth-butlast split'*)
qed
have *5*: *?pocket-path* = *make-polygonal-path* (*?a* # *?cutvts* @ [*?b*])
by (*smt* (*verit*, *ccfv-SIG*) *a-neq-b butlast.simps*(*2*) *butlast-tl hd-Cons-tl hd-Nil-eq-last last.simps snoc-eq-iff-butlast*)
have *6*: *?p* = *make-polygonal-path* (*?vts* @ [*?vts*!*0*])
by (*metis* (*no-types*, *lifting*) *butlast-conv-take have-wraparound-vertex is-polygon-split-path-def nth-butlast polygon-filled-p split'*)
have *7*: *?p1* = *make-polygonal-path* (*?a* # *?vts2* @ [*?b*] @ *rev ?cutvts* @ [*?a*]) **by** *blast*
have *8*: *?p2* = *make-polygonal-path* (*?vts1* @ ([*?a*] @ *?cutvts* @ [*?b*]) @ *?vts3* @ [*?vts*!*0*])
proof−
have *?rotated-vts* = *?vts1* @ ([*?a*] @ *?cutvts* @ [*?b*]) @ *?vts3* @ [*?vts*!*0*]
unfolding *construct-pocket-0-def fill-pocket-0-def*
by (*smt* (*verit*) *3 Suc-1 hd-last-rotated a-neq-b append-Cons append-Nil append-butlast-last-id append-take-drop-id construct-pocket-0-def drop-Suc drop-drop drop-eq-Nil fill-pocket-0-def hd-Nil-eq-last hd-append2 hd-conv-nth last-conv-nth last-drop length-Cons length-take length-tl linorder-not-less list.collapse list.sel*(*3*) *list.size*(*3*) *min.absorb4 plus-1-eq-Suc take-all-iff*)
thus *?thesis* **by** *argo*
qed
have *9*: *?I1* = *card* {*x*. *integral-vec x* ∧ *x* ∈ *path-inside ?p1*} **by** *blast*
have *10*: *?B1* = *card* {*x*. *integral-vec x* ∧ *x* ∈ *path-image ?p1*} **by** *blast*
have *11*: *?I2* = *card* {*x*. *integral-vec x* ∧ *x* ∈ *path-inside ?p2*} **by** *blast*
have *12*: *?B2* = *card* {*x*. *integral-vec x* ∧ *x* ∈ *path-image ?p2*} **by** *blast*
have *13*: *?I* = *card* {*x*. *integral-vec x* ∧ *x* ∈ *path-inside ?p*} **by** *blast*
have *14*: *?B* = *card* {*x*. *integral-vec x* ∧ *x* ∈ *path-image ?p*} **by** *blast*
have *15*: *all-integral ?vts*
using *subset-filled less*
unfolding *all-integral-def*
by (*metis* (*no-types*, *lifting*) *all-integral-def all-integral-filled-vts in-set-butlastD*)
have *16*: *measure lebesgue* (*path-inside ?p*) = *?I* + *?B*/*2* − *1*
using *ih-filled* **unfolding** *integral-inside integral-boundary* **by** *blast*
have *17*: *measure lebesgue* (*path-inside ?p1*) = *?I1* + *?B1*/*2* − *1*

using *ih-pocket* **unfolding** *integral-inside integral-boundary* **using** *pocket-rev-image pocket-rev-inside* **by** *force*
have *measure lebesgue* (*path-inside ?p2*) = *?I2* + *?B2*/*2* − *1*
using *pick-split-path-union-main*(*3*)
[*OF split' 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17*] *less*(*5*−*6*) **by** *blast*
moreover have *?I2* = *I* **using** *less*(*5*) *pr-image path-inside-def* **by** *presburger*
moreover have *?B2* = *B* **using** *less*(*6*) *pr-image path-image-def* **by** *presburger*
ultimately have *?case* **by** (*simp add*: *path-inside-def pocket-rev-inside pr-image*)
}
ultimately have *?case* **by** *blast*
}
ultimately show *?case* **using** *card-vts* **by** *linarith*
qed

theorem *pick*:
fixes *p* :: *R-to-R2*
assumes *polygon p*
assumes *p* = *make-polygonal-path vts*
assumes *all-integral vts*
assumes *I* = *card* {*x*. *integral-vec x* ∧ *x* ∈ *path-inside p*}
assumes *B* = *card* {*x*. *integral-vec x* ∧ *x* ∈ *path-image p*}
shows *measure lebesgue* (*path-inside p*) = *I* + *B*/*2* − *1*
proof−
obtain *p' vts'* **where** *polygon-of p' vts'*
∧ *vts'*!*0* ∈ *frontier* (*convex hull* (*set vts'*))
∧ *path-image p'* = *path-image p*
∧ *all-integral vts'*
∧ *set vts'* = *set vts*
using *pick-rotate assms* **unfolding** *polygon-of-def* **by** *blast*
thus *?thesis* **using** *assms pick-unrotated* **unfolding** *path-inside-def polygon-of-def* **by** *fastforce*
qed

end

References

[1] B. Grünbaum and G. C. Shephard. Pick's theorem. *The American Mathematical Monthly*, 100(2):150–161, 1993.

[2] J. Harrison. A formal proof of Pick's theorem. *Math. Struct. Comput. Sci.*, 21(4):715–729, 2011.