Poincaré Disc Model

Danijela Simić Filip Marić Pierre Boutry

April 9, 2025

Abstract

We describe formalization of the Poincaré disc model of hyperbolic geometry within the Isabelle/HOL proof assistant. The model is defined within the extended complex plane (one dimensional complex projective space $\mathbb{C}P^1$), formalized in the AFP entry "Complex Geometry" [6]. Points, lines, congruence of pairs of points, betweenness of triples of points, circles, and isometries are defined within the model. It is shown that the model satisfies all Tarski's axioms except the Euclid's axiom. It is shown that it satisfies its negation and the limiting parallels axiom (which proves it to be a model of hyperbolic geometry).

Contents

1 Introduction

Poincaré disc is a model of hyperbolic geometry. That fact has been a mathematical folklore for more than 100 years. However, up to the best of our knowledge, fully precise, formal proofs of this fact are lacking. In this paper we present a formalization of the Poincaré disc model in Isabelle/HOL, introduce its basic notions (h-points, h-lines, h-congruence, h-isometries, h-betweenness) and prove that it models Tarski's axioms except for Euclid's axiom. We shown that is satisfies the negation of Euclid's axiom, and, moreover, the existence of limiting parallels axiom. The model is defined within the extended complex plane, which has been described quite precisely by Schwerdfeger [8] and formalized in the previous work of the first two authors [5].

Related work. In 1840 Lobachevsky [3] published developments about non-Euclidean geometry. Hyperbolic geometry is studied through many of its models. The concept of a projective disc model was introduced by Klein while Poincaré investigated the half-plane model proposed by Liouville and Beltrami and primarily studied the isometries of the hyperbolic plane that preserve orientation. In this paper, we focus on the formalization of the latter.

Regarding non-Euclidean geometry, Makarios showed the independence of Euclid's axiom [4]. He did so by formalizing that the Klein–Beltrami model is a model of Tarski's axioms at the exception of Euclid's axiom. Latter Coghetto formalized the Klein-Beltrami model within Mizar [1, 2].

2 Background theories

2.1 Hyperbolic Functions

In this section hyperbolic cosine and hyperbolic sine functions are introduced and some of their properties needed for further development are proved.

theory *Hyperbolic-Functions*
  imports *Complex-Main Complex-Geometry.More-Complex*
begin

lemma *arcosh-eq-iff*:
  fixes $x\ y::real$
  assumes $x \geq 1\ y \geq 1$
  shows $arcosh\ x = arcosh\ y \longleftrightarrow x = y$
  by (*smt* (*verit*, *best*) *arcosh-less-iff-real arcosh-real-strict-mono assms*)

lemma *cosh-gt-1* [*simp*]:
  fixes $x ::real$
  assumes $x > 0$
  shows $cosh\ x > 1$
  using *assms cosh-real-strict-mono* **by** *force*

lemma *cosh-eq-iff*:
  fixes $x\ y::real$
  assumes $x \geq 0\ y \geq 0$
  shows $cosh\ x = cosh\ y \longleftrightarrow x = y$
  by (*simp add*: *assms*)

lemma *arcosh-mono*:
  fixes $x\ y::real$
  assumes $x \geq 1\ y \geq 1$
  shows $arcosh\ x \geq arcosh\ y \longleftrightarrow x \geq y$
  using *arcosh-less-iff-real assms linorder-not-le* **by** *blast*

lemma *arcosh-add*:
  fixes $x\ y::real$
  assumes $x \geq 1\ y \geq 1$
  shows $arcosh\ x + arcosh\ y = arcosh\ (x*y + sqrt((x^2 - 1)*(y^2 - 1)))$
proof−
  have $sqrt\ ((x^2 - 1) * (y^2 - 1)) \geq 0$

```
    using assms
    by simp
  moreover
  have x * y ≥ 1
    using assms
    by (smt mult-le-cancel-left1)
  ultimately
  have **: x * y + sqrt ((x^2 − 1) * (y^2 − 1)) ≥ 1
    by linarith
  hence 1: 0 ≤ (x * y + sqrt ((x^2 − 1) * (y^2 − 1)))^2 − 1
      by simp

  have 2: x * sqrt (y^2 − 1) + y * sqrt (x^2 − 1) ≥ 0
    using assms
    by simp

  have (x*sqrt(y^2 − 1)+y*sqrt(x^2 −1))^2 = (sqrt((x*y+sqrt((x^2−1)*(y^2−1)))^2 − 1))^2
    using assms
  proof (subst real-sqrt-pow2)
    show 0 ≤ (x * y + sqrt ((x^2 − 1) * (y^2 − 1)))^2 − 1
      by fact
  next
    have (x * sqrt (y^2 − 1))^2 = x^2 * (y^2 − 1)
      by (simp add: ‹y ≥ 1› power-mult-distrib)
    moreover
    have (y * sqrt (x^2 − 1))^2 = y^2 * (x^2 − 1)
      using assms by (simp add: power-mult-distrib)
    ultimately show (x * sqrt (y^2 − 1) + y * sqrt (x^2 − 1))^2 = (x * y + sqrt ((x^2 − 1) * (y^2 − 1)))^2 − 1
      using assms
      unfolding power2-sum
      apply (simp add: real-sqrt-mult power-mult-distrib)
      apply (simp add: field-simps)
      done
  qed
  hence sqrt ((x * y + sqrt ((x^2 − 1) * (y^2 − 1)))^2 − 1) = x * sqrt (y^2 − 1) + y * sqrt (x^2 − 1)
    using power2-eq-iff-nonneg[OF 2 real-sqrt-ge-zero[OF 1]]
    by simp
  thus ?thesis
    using assms **
    apply (simp add: arcosh-real-def)
    apply (subst ln-mult-pos[symmetric])
     apply (smt one-le-power real-sqrt-ge-zero)
    apply (smt (verit) one-le-power real-sqrt-ge-zero)
    by (smt (verit) distrib-right mult.commute real-sqrt-mult)
qed

lemma arcosh-double:
  fixes x :: real
  assumes x ≥ 1
  shows 2 * arcosh x = arcosh (2*x^2 − 1)
  by (smt arcosh-add arcosh-mono assms one-power2 power2-eq-square real-sqrt-abs)

end
```

3 Tarski axioms

In this section we introduce axioms of Tarski [7] trough a series of locales.

```
theory Tarski
imports Main
begin
```

The first locale assumes all Tarski axioms except for the Euclid's axiom and the continuity axiom and corresponds to absolute geometry.

```
locale TarskiAbsolute =
```

fixes *cong* :: $'p \Rightarrow 'p \Rightarrow 'p \Rightarrow 'p \Rightarrow bool$
fixes *betw* :: $'p \Rightarrow 'p \Rightarrow 'p \Rightarrow bool$
assumes *cong-reflexive*: $cong\ x\ y\ y\ x$
assumes *cong-transitive*: $cong\ x\ y\ z\ u \wedge cong\ x\ y\ v\ w \longrightarrow cong\ z\ u\ v\ w$
assumes *cong-identity*: $cong\ x\ y\ z\ z \longrightarrow x = y$
assumes *segment-construction*: $\exists\ z.\ betw\ x\ y\ z \wedge cong\ y\ z\ a\ b$
assumes *five-segment*: $x \neq y \wedge betw\ x\ y\ z \wedge betw\ x'\ y'\ z' \wedge cong\ x\ y\ x'\ y' \wedge cong\ y\ z\ y'\ z' \wedge cong\ x\ u\ x'\ u' \wedge cong\ y\ u\ y'\ u' \longrightarrow cong\ z\ u\ z'\ u'$
assumes *betw-identity*: $betw\ x\ y\ x \longrightarrow x = y$
assumes *Pasch*: $betw\ x\ u\ z \wedge betw\ y\ v\ z \longrightarrow (\exists\ a.\ betw\ u\ a\ y \wedge betw\ x\ a\ v)$
assumes *lower-dimension*: $\exists\ a.\ \exists\ b.\ \exists\ c.\ \neg\ betw\ a\ b\ c \wedge \neg\ betw\ b\ c\ a \wedge \neg\ betw\ c\ a\ b$
assumes *upper-dimension*: $cong\ x\ u\ x\ v \wedge cong\ y\ u\ y\ v \wedge cong\ z\ u\ z\ v \wedge u \neq v \longrightarrow betw\ x\ y\ z \vee betw\ y\ z\ x \vee betw\ z\ x\ y$
begin

The following definitions are used to specify axioms in the following locales.

Point p is on line ab.

definition *on-line* **where**
$on\text{-}line\ p\ a\ b \longleftrightarrow betw\ p\ a\ b \vee betw\ a\ p\ b \vee betw\ a\ b\ p$

Point p is on ray ab.

definition *on-ray* **where**
$on\text{-}ray\ p\ a\ b \longleftrightarrow betw\ a\ p\ b \vee betw\ a\ b\ p$

Point p is inside angle abc.

definition *in-angle* **where**
$in\text{-}angle\ p\ a\ b\ c \longleftrightarrow b \neq a \wedge b \neq c \wedge p \neq b \wedge (\exists\ x.\ betw\ a\ x\ c \wedge x \neq a \wedge x \neq c \wedge on\text{-}ray\ p\ b\ x)$

Ray $r_a r_b$ meets the line $l_a l_b$.

definition *ray-meets-line* **where**
$ray\text{-}meets\text{-}line\ ra\ rb\ la\ lb \longleftrightarrow (\exists\ x.\ on\text{-}ray\ x\ ra\ rb \wedge on\text{-}line\ x\ la\ lb)$

end

The second locales adds the negation of Euclid's axiom and limiting parallels and corresponds to hyperbolic geometry.

locale *TarskiHyperbolic* = *TarskiAbsolute* +
assumes *euclid-negation*: $\exists\ a\ b\ c\ d\ t.\ betw\ a\ d\ t \wedge betw\ b\ d\ c \wedge a \neq d \wedge (\forall\ x\ y.\ betw\ a\ b\ x \wedge betw\ a\ c\ y \longrightarrow \neg\ betw\ x\ t\ y)$
assumes *limiting-parallels*: $\neg\ on\text{-}line\ a\ x1\ x2 \Longrightarrow$
$(\exists\ a1\ a2.\ \neg\ on\text{-}line\ a\ a1\ a2\ \wedge$
$\neg\ ray\text{-}meets\text{-}line\ a\ a1\ x1\ x2\ \wedge$
$\neg\ ray\text{-}meets\text{-}line\ a\ a2\ x1\ x2\ \wedge$
$(\forall\ a'.\ in\text{-}angle\ a'\ a1\ a\ a2 \longrightarrow ray\text{-}meets\text{-}line\ a\ a'\ x1\ x2))$

The third locale adds the continuity axiom and corresponds to elementary hyperbolic geometry.

locale *ElementaryTarskiHyperbolic* = *TarskiHyperbolic* +
assumes *continuity*: $[\![\exists\ a.\ \forall\ x.\ \forall\ y.\ \varphi\ x \wedge \psi\ y \longrightarrow betw\ a\ x\ y]\!] \Longrightarrow \exists\ b.\ \forall\ x.\ \forall\ y.\ \varphi\ x \wedge \psi\ y \longrightarrow betw\ x\ b\ y$

end

4 H-lines in the Poincaré model

theory *Poincare-Lines*
imports *Complex-Geometry.Unit-Circle-Preserving-Moebius Complex-Geometry.Circlines-Angle*
begin

4.1 Definition and basic properties of h-lines

H-lines in the Poincaré model are either line segments passing trough the origin or segments (within the unit disc) of circles that are perpendicular to the unit circle. Algebraically these are circlines that are represented by Hermitean matrices of the form

$$H = \begin{pmatrix} A & B \\ \overline{B} & A \end{pmatrix},$$

for $A \in \mathbb{R}$, and $B \in \mathbb{C}$, and $|B|^2 > A^2$, where the circline equation is the usual one: $z^* H z = 0$, for homogenous coordinates z.

definition *is-poincare-line-cmat* :: *complex-mat* ⇒ *bool* **where**
  [*simp*]: *is-poincare-line-cmat H* ⟷
    (*let* (*A*, *B*, *C*, *D*) = *H*
      in hermitean (*A*, *B*, *C*, *D*) ∧ *A* = *D* ∧ (*cmod B*)^2 > (*cmod A*)^2)

lift-definition *is-poincare-line-clmat* :: *circline-mat* ⇒ *bool* **is** *is-poincare-line-cmat*
  done

We introduce the predicate that checks if a given complex matrix is a matrix of a h-line in the Poincaré model, and then by means of the lifting package lift it to the type of non-zero Hermitean matrices, and then to circlines (that are equivalence classes of such matrices).

lift-definition *is-poincare-line* :: *circline* ⇒ *bool* **is** *is-poincare-line-clmat*
proof (*transfer*, *transfer*)
  fix *H1 H2* :: *complex-mat*
  assume *hh*: *hermitean H1* ∧ *H1* ≠ *mat-zero hermitean H2* ∧ *H2* ≠ *mat-zero*
  assume *circline-eq-cmat H1 H2*
  thus *is-poincare-line-cmat H1* ⟷ *is-poincare-line-cmat H2*
    using *hh*
    by (*cases H1*, *cases H2*) (*auto simp*: *norm-mult power-mult-distrib*)
qed

lemma *is-poincare-line-mk-circline*:
  assumes (*A*, *B*, *C*, *D*) ∈ *hermitean-nonzero*
  shows *is-poincare-line* (*mk-circline A B C D*) ⟷ (*cmod B*)^2 > (*cmod A*)^2 ∧ *A* = *D*
  using *assms*
  by (*transfer*, *transfer*, *auto simp add*: *Let-def*)

Abstract characterisation of *is-poincare-line* predicate: H-lines in the Poincaré model are real circlines (circlines with the negative determinant) perpendicular to the unit circle.

lemma *is-poincare-line-iff*:
  shows *is-poincare-line H* ⟷ *circline-type H* = −*1* ∧ *perpendicular H unit-circle*
  unfolding *perpendicular-def*
proof (*simp*, *transfer*, *transfer*)
  fix *H*
  assume *hh*: *hermitean H* ∧ *H* ≠ *mat-zero*
  obtain *A B C D* **where** ∗: *H* = (*A*, *B*, *C*, *D*)
    by (*cases H*, *auto*)
  have ∗∗: *is-real A is-real D C* = *cnj B*
    using *hh* ∗ *hermitean-elems*
    by *auto*
  hence (*Re A* = *Re D* ∧ *cmod A* ∗ *cmod A* < *cmod B* ∗ *cmod B*) =
      (*Re A* ∗ *Re D* < *Re B* ∗ *Re B* + *Im B* ∗ *Im B* ∧ (*Re D* = *Re A* ∨ *Re A* ∗ *Re D* = *Re B* ∗ *Re B* + *Im B* ∗ *Im B*))
    using ∗
    by (*smt cmod-power2 power2-eq-square zero-power2*)+
  thus *is-poincare-line-cmat H* ⟷
      circline-type-cmat H = − *1* ∧ *cos-angle-cmat* (*of-circline-cmat H*) *unit-circle-cmat* = *0*
    using ∗ ∗∗
    by (*auto simp add*: *sgn-1-neg complex-eq-if-Re-eq cmod-square power2-eq-square simp del*: *pos-oriented-cmat-def*)
qed

The *x-axis* is an h-line.

lemma *is-poincare-line-x-axis* [*simp*]:
  shows *is-poincare-line x-axis*
  by (*transfer*, *transfer*) (*auto simp add*: *hermitean-def mat-adj-def mat-cnj-def*)

The *unit-circle* is not an h-line.

lemma *not-is-poincare-line-unit-circle* [*simp*]:
  shows ¬ *is-poincare-line unit-circle*
  by (*transfer*, *transfer*, *simp*)

4.1.1 Collinear points

Points are collinear if they all belong to an h-line.

definition *poincare-collinear* :: *complex-homo set* $\Rightarrow$ *bool* **where**
poincare-collinear S $\longleftrightarrow$ ($\exists$ *p*. *is-poincare-line p* $\wedge$ *S* $\subseteq$ *circline-set p*)

4.1.2 H-lines and inversion

Every h-line in the Poincaré model contains the inverse (wrt. the unit circle) of each of its points (note that at most one of them belongs to the unit disc).

lemma *is-poincare-line-inverse-point*:
assumes *is-poincare-line H u* $\in$ *circline-set H*
shows *inversion u* $\in$ *circline-set H*
using *assms*
unfolding *is-poincare-line-iff circline-set-def perpendicular-def inversion-def*
apply *simp*
proof (*transfer*, *transfer*)
fix *u H*
assume *hh*: *hermitean H* $\wedge$ *H* $\neq$ *mat-zero u* $\neq$ *vec-zero* **and**
aa: *circline-type-cmat H* = $-$ *1* $\wedge$ *cos-angle-cmat* (*of-circline-cmat H*) *unit-circle-cmat* = *0* *on-circline-cmat-cvec H u*
obtain *A B C D u1 u2* **where** $*$: *H* = (*A*, *B*, *C*, *D*) *u* = (*u1*, *u2*)
by (*cases H*, *cases u*, *auto*)
have *is-real A is-real D C* = *cnj B*
using $*$ *hh hermitean-elems*
by *auto*
moreover
have *A* = *D*
using *aa*(*1*) $*$ ‹*is-real A*› ‹*is-real D*›
by (*auto simp del*: *pos-oriented-cmat-def simp add*: *complex.expand split*: *if-split-asm*)
thus *on-circline-cmat-cvec H* (*conjugate-cvec* (*reciprocal-cvec u*))
using *aa*(*2*) $*$
by (*simp add*: *vec-cnj-def field-simps*)
qed

Every h-line in the Poincaré model and is invariant under unit circle inversion.

lemma *circline-inversion-poincare-line*:
assumes *is-poincare-line H*
shows *circline-inversion H* = *H*
proof$-$
obtain *u v w* **where** $*$: *u* $\neq$ *v v* $\neq$ *w u* $\neq$ *w* {*u*, *v*, *w*} $\subseteq$ *circline-set H*
using *assms is-poincare-line-iff*[*of H*]
using *circline-type-neg-card-gt3*[*of H*]
by *auto*
hence {*inversion u*, *inversion v*, *inversion w*} $\subseteq$ *circline-set* (*circline-inversion H*)
{*inversion u*, *inversion v*, *inversion w*} $\subseteq$ *circline-set H*
using *is-poincare-line-inverse-point*[*OF assms*]
by *auto*
thus *?thesis*
using $*$ *unique-circline-set*[*of inversion u inversion v inversion w*]
by (*metis insert-subset inversion-involution*)
qed

4.1.3 Classification of h-lines into Euclidean segments and circles

If an h-line contains zero, than it also contains infinity (the inverse point of zero) and is by definition an Euclidean line.

lemma *is-poincare-line-trough-zero-trough-infty* [*simp*]:
assumes *is-poincare-line l* **and** 0_h $\in$ *circline-set l*
shows ∞_h $\in$ *circline-set l*
using *is-poincare-line-inverse-point*[*OF assms*]
by *simp*

lemma *is-poincare-line-trough-zero-is-line*:

assumes *is-poincare-line l* **and** $0_h \in$ *circline-set l*
shows *is-line l*
using *assms*
using *inf-in-circline-set is-poincare-line-trough-zero-trough-infty*
by *blast*

If an h-line does not contain zero, than it also does not contain infinity (the inverse point of zero) and is by definition an Euclidean circle.

lemma *is-poincare-line-not-trough-zero-not-trough-infty* [*simp*]:
assumes *is-poincare-line l*
assumes $0_h \notin$ *circline-set l*
shows $\infty_h \notin$ *circline-set l*
using *assms*
using *is-poincare-line-inverse-point*[*OF assms*(*1*), *of* ∞_h]
by *auto*

lemma *is-poincare-line-not-trough-zero-is-circle*:
assumes *is-poincare-line l* $0_h \notin$ *circline-set l*
shows *is-circle l*
using *assms*
using *inf-in-circline-set is-poincare-line-not-trough-zero-not-trough-infty*
by *auto*

4.1.4 Points on h-line

Each h-line in the Poincaré model contains at least two different points within the unit disc.

First we prove an auxiliary lemma.

lemma *ex-is-poincare-line-points'*:
assumes *i12*: *i1* $\in$ *circline-set H* $\cap$ *unit-circle-set*
i2 $\in$ *circline-set H* $\cap$ *unit-circle-set*
i1 $\neq$ *i2*
assumes *a*: *a* $\in$ *circline-set H a* $\notin$ *unit-circle-set*
shows $\exists$ *b*. *b* $\neq$ *i1* $\wedge$ *b* $\neq$ *i2* $\wedge$ *b* $\neq$ *a* $\wedge$ *b* $\neq$ *inversion a* $\wedge$ *b* $\in$ *circline-set H*
proof−
have *inversion a* $\notin$ *unit-circle-set*
using ‹*a* $\notin$ *unit-circle-set*›
unfolding *unit-circle-set-def circline-set-def*
by (*metis inversion-id-iff-on-unit-circle inversion-involution mem-Collect-eq*)

have *a* $\neq$ *inversion a*
using ‹*a* $\notin$ *unit-circle-set*› *inversion-id-iff-on-unit-circle*[*of a*]
unfolding *unit-circle-set-def circline-set-def*
by *auto*

have *a* $\neq$ *i1 a* $\neq$ *i2 inversion a* $\neq$ *i1 inversion a* $\neq$ *i2*
using *assms* ‹*inversion a* $\notin$ *unit-circle-set*›
by *auto*

then obtain *b* **where** *cr2*: *cross-ratio b i1 a i2* = *of-complex 2*
using ‹*i1* $\neq$ *i2*›
using *ex-cross-ratio*[*of i1 a i2*]
by *blast*

have *distinct-b*: *b* $\neq$ *i1 b* $\neq$ *i2 b* $\neq$ *a*
using ‹*i1* $\neq$ *i2*› ‹*a* $\neq$ *i1*› ‹*a* $\neq$ *i2*›
using *ex1-cross-ratio*[*of i1 a i2*]
using *cross-ratio-0*[*of i1 a i2*] *cross-ratio-1*[*of i1 a i2*] *cross-ratio-inf*[*of i1 i2 a*]
using *cr2*
by *auto*

hence *b* $\in$ *circline-set H*
using *assms four-points-on-circline-iff-cross-ratio-real*[*of b i1 a i2*] *cr2*
using *unique-circline-set*[*of i1 i2 a*]
by *auto*

moreover

have *b* ≠ *inversion a*
proof (*rule ccontr*)
assume ∗: ¬ *?thesis*
have *inversion i1* = *i1 inversion i2* = *i2*
using *i12*
unfolding *unit-circle-set-def*
by *auto*
hence *cross-ratio* (*inversion a*) *i1 a i2* = *cross-ratio a i1* (*inversion a*) *i2*
using ∗ *cross-ratio-inversion*[*of i1 a i2 b*] ‹*a* ≠ *i1*› ‹*a* ≠ *i2*› ‹*i1* ≠ *i2*› ‹*b* ≠ *i1*›
using *four-points-on-circline-iff-cross-ratio-real*[*of b i1 a i2*]
using *i12 distinct-b conjugate-id-iff*[*of cross-ratio b i1 a i2*]
using *i12 a* ‹*b* ∈ *circline-set H*›
by *auto*
hence *cross-ratio* (*inversion a*) *i1 a i2* ≠ *of-complex 2*
using *cross-ratio-commute-13*[*of inversion a i1 a i2*]
using *reciprocal-id-iff*
using *of-complex-inj*
by *force*
thus *False*
using ∗ *cr2*
by *simp*
qed

ultimately
show *?thesis*
using *assms* ‹*b* ≠ *i1*› ‹*b* ≠ *i2*› ‹*b* ≠ *a*›
by *auto*
qed

Now we can prove the statement.

lemma *ex-is-poincare-line-points*:
assumes *is-poincare-line H*
shows ∃ *u v*. *u* ∈ *unit-disc* ∧ *v* ∈ *unit-disc* ∧ *u* ≠ *v* ∧ {*u*, *v*} ⊆ *circline-set H*
proof−
obtain *u v w* **where** ∗: *u* ≠ *v v* ≠ *w u* ≠ *w* {*u*, *v*, *w*} ⊆ *circline-set H*
using *assms is-poincare-line-iff*[*of H*]
using *circline-type-neg-card-gt3*[*of H*]
by *auto*

have ¬ {*u*, *v*, *w*} ⊆ *unit-circle-set*
using *unique-circline-set*[*of u v w*] ∗
by (*metis assms insert-subset not-is-poincare-line-unit-circle unit-circle-set-def*)

hence *H* ≠ *unit-circle*
unfolding *unit-circle-set-def*
using ∗
by *auto*

show *?thesis*
proof (*cases* (*u* ∈ *unit-disc* ∧ *v* ∈ *unit-disc*) ∨
(*u* ∈ *unit-disc* ∧ *w* ∈ *unit-disc*) ∨
(*v* ∈ *unit-disc* ∧ *w* ∈ *unit-disc*))
case *True*
thus *?thesis*
using ∗
by *auto*
next
case *False*

have ∃ *a b*. *a* ≠ *b* ∧ *a* ≠ *inversion b* ∧ *a* ∈ *circline-set H* ∧ *b* ∈ *circline-set H* ∧ *a* ∉ *unit-circle-set* ∧ *b* ∉ *unit-circle-set*
proof (*cases* (*u* ∈ *unit-circle-set* ∧ *v* ∈ *unit-circle-set*) ∨
(*u* ∈ *unit-circle-set* ∧ *w* ∈ *unit-circle-set*) ∨
(*v* ∈ *unit-circle-set* ∧ *w* ∈ *unit-circle-set*))

```
  case True
  then obtain i1 i2 a where *:
    i1 ∈ unit-circle-set ∩ circline-set H i2 ∈ unit-circle-set ∩ circline-set H
    a ∈ circline-set H a ∉ unit-circle-set
    i1 ≠ i2 i1 ≠ a i2 ≠ a
    using * ‹¬ {u, v, w} ⊆ unit-circle-set›
    by auto
  then obtain b where b ∈ circline-set H b ≠ i1 b ≠ i2 b ≠ a b ≠ inversion a
    using ex-is-poincare-line-points'[of i1 H i2 a]
    by blast

  hence b ∉ unit-circle-set
    using * ‹H ≠ unit-circle› unique-circline-set[of i1 i2 b]
    unfolding unit-circle-set-def
    by auto

  thus ?thesis
    using * ‹b ∈ circline-set H› ‹b ≠ a› ‹b ≠ inversion a›
    by auto
next
  case False
  then obtain f g h where
    *: f ≠ g f ∈ circline-set H f ∉ unit-circle-set
             g ∈ circline-set H g ∉ unit-circle-set
             h ∈ circline-set H h ≠ f h ≠ g
    using *
    by auto
  show ?thesis
  proof (cases f = inversion g)
    case False
    thus ?thesis
      using *
      by auto
  next
    case True
    show ?thesis
    proof (cases h ∈ unit-circle-set)
      case False
      thus ?thesis
        using * ‹f = inversion g›
        by auto
    next
      case True
      obtain m where cr2: cross-ratio m h f g = of-complex 2
        using ex-cross-ratio[of h f g] * ‹f ≠ g› ‹h ≠ f› ‹h ≠ g›
        by auto
      hence m ≠ h m ≠ f m ≠ g
        using ‹h ≠ f› ‹h ≠ g› ‹f ≠ g›
        using ex1-cross-ratio[of h f g]
        using cross-ratio-0[of h f g] cross-ratio-1[of h f g] cross-ratio-inf[of h g f]
        using cr2
        by auto
      hence m ∈ circline-set H
        using four-points-on-circline-iff-cross-ratio-real[of m h f g] cr2
        using ‹h ≠ f› ‹h ≠ g› ‹f ≠ g› *
        using unique-circline-set[of h f g]
        by auto

      show ?thesis
      proof (cases m ∈ unit-circle-set)
        case False
        thus ?thesis
          using ‹m ≠ f› ‹m ≠ g› ‹f = inversion g› * ‹m ∈ circline-set H›
          by auto
      next
        case True
```

then obtain *n* **where** $n \neq h$ $n \neq m$ $n \neq f$ $n \neq$ *inversion f* $n \in$ *circline-set H*
using *ex-is-poincare-line-points'*[*of h H m f*] $*$ ‹$m \in$ *circline-set H*› ‹$h \in$ *unit-circle-set*› ‹$m \neq h$›
by *auto*
hence $n \notin$ *unit-circle-set*
using $*$ ‹$H \neq$ *unit-circle*› *unique-circline-set*[*of m n h*]
using ‹$m \neq h$› ‹$m \in$ *unit-circle-set*› ‹$h \in$ *unit-circle-set*› ‹$m \in$ *circline-set H*›
unfolding *unit-circle-set-def*
by *auto*

thus *?thesis*
using $*$ ‹$n \in$ *circline-set H*› ‹$n \neq f$› ‹$n \neq$ *inversion f*›
by *auto*
qed
qed
qed
qed
then obtain *a b* **where** *ab*: $a \neq b$ $a \neq$ *inversion b* $a \in$ *circline-set H* $b \in$ *circline-set H* $a \notin$ *unit-circle-set* $b \notin$ *unit-circle-set*
by *blast*
have $\forall$ *x*. $x \in$ *circline-set H* $\wedge$ $x \notin$ *unit-circle-set* $\longrightarrow$ ($\exists$ *x'*. $x' \in$ *circline-set H* $\cap$ *unit-disc* $\wedge$ ($x' = x \vee x' =$ *inversion x*))
proof *safe*
fix *x*
assume *x*: $x \in$ *circline-set H* $x \notin$ *unit-circle-set*
show $\exists$ *x'*. $x' \in$ *circline-set H* $\cap$ *unit-disc* $\wedge$ ($x' = x \vee x' =$ *inversion x*)
proof (*cases* $x \in$ *unit-disc*)
case *True*
thus *?thesis*
using *x*
by *auto*
next
case *False*
hence $x \in$ *unit-disc-compl*
using *x* *in-on-out-univ*[*of ounit-circle*]
unfolding *unit-circle-set-def unit-disc-def unit-disc-compl-def*
by *auto*
hence *inversion x* $\in$ *unit-disc*
using *inversion-unit-disc-compl*
by *blast*
thus *?thesis*
using *is-poincare-line-inverse-point*[*OF assms, of x*] *x*
by *auto*
qed
qed
then obtain *a' b'* **where**
$*$: $a' \in$ *circline-set H* $a' \in$ *unit-disc* $b' \in$ *circline-set H* $b' \in$ *unit-disc* **and**
$**$: $a' = a \vee a' =$ *inversion a* $b' = b \vee b' =$ *inversion b*
using *ab*
by *blast*
have $a' \neq b'$
using ‹$a \neq b$› ‹$a \neq$ *inversion b*› $**$ $*$
by (*metis inversion-involution*)
thus *?thesis*
using $*$
by *auto*
qed
qed

4.1.5 H-line uniqueness

There is no more than one h-line that contains two different h-points (in the disc).

lemma *unique-is-poincare-line*:
assumes *in-disc*: $u \in$ *unit-disc* $v \in$ *unit-disc* $u \neq v$
assumes *pl*: *is-poincare-line l1 is-poincare-line l2*
assumes *on-l*: $\{u, v\} \subseteq$ *circline-set l1* $\cap$ *circline-set l2*
shows $l1 = l2$

proof −
  have *u* ≠ *inversion u v* ≠ *inversion u*
    using *in-disc*
    using *inversion-noteq-unit-disc*[*of u v*]
    using *inversion-noteq-unit-disc*[*of u u*]
    by *auto*
  thus *?thesis*
    using *on-l*
    using *unique-circline-set*[*of u inversion u v*] ‹*u* ≠ *v*›
    using *is-poincare-line-inverse-point*[*of l1 u*]
    using *is-poincare-line-inverse-point*[*of l2 u*]
    using *pl*
    by *auto*
qed

For the rest of our formalization it is often useful to consider points on h-lines that are not within the unit disc. Many lemmas in the rest of this section will have such generalizations.

There is no more than one h-line that contains two different and not mutually inverse points (not necessary in the unit disc).

lemma *unique-is-poincare-line-general*:
  assumes *different*: *u* ≠ *v u* ≠ *inversion v*
  assumes *pl*: *is-poincare-line l1 is-poincare-line l2*
  assumes *on-l*: {*u*, *v*} ⊆ *circline-set l1* ∩ *circline-set l2*
  shows *l1* = *l2*
proof (*cases u* ≠ *inversion u*)
  case *True*
  thus *?thesis*
    using *unique-circline-set*[*of u inversion u v*]
    using *assms*
    using *is-poincare-line-inverse-point* **by** *force*
next
  case *False*
  show *?thesis*
  proof (*cases v* ≠ *inversion v*)
    case *True*
    thus *?thesis*
      using *unique-circline-set*[*of u inversion v v*]
      using *assms*
      using *is-poincare-line-inverse-point* **by** *force*
  next
    case *False*

    have *on-circline unit-circle u on-circline unit-circle v*
      using ‹¬ *u* ≠ *inversion u*› ‹¬ *v* ≠ *inversion v*›
      using *inversion-id-iff-on-unit-circle*
      by *fastforce*+
    thus *?thesis*
      using *pl on-l* ‹*u* ≠ *v*›
      unfolding *circline-set-def*
      apply *simp*
    proof (*transfer*, *transfer*, *safe*)
      fix *u1 u2 v1 v2 A1 B1 C1 D1 A2 B2 C2 D2* :: *complex*
      let *?u* = (*u1*, *u2*) **and** *?v* = (*v1*, *v2*) **and** *?H1* = (*A1*, *B1*, *C1*, *D1*) **and** *?H2* = (*A2*, *B2*, *C2*, *D2*)
      assume ∗: *?u* ≠ *vec-zero ?v* ≠ *vec-zero*
        on-circline-cmat-cvec unit-circle-cmat ?u on-circline-cmat-cvec unit-circle-cmat ?v
        is-poincare-line-cmat ?H1 is-poincare-line-cmat ?H2
        hermitean ?H1 ?H1 ≠ *mat-zero hermitean ?H2 ?H2* ≠ *mat-zero*
        on-circline-cmat-cvec ?H1 ?u on-circline-cmat-cvec ?H1 ?v
        on-circline-cmat-cvec ?H2 ?u on-circline-cmat-cvec ?H2 ?v
        ¬ (*u1*, *u2*) $\approx_v$ (*v1*, *v2*)
      have ∗∗: *A1* = *D1 A2* = *D2 C1* = *cnj B1 C2* = *cnj B2 is-real A1 is-real A2*
        using ‹*is-poincare-line-cmat ?H1*› ‹*is-poincare-line-cmat ?H2*›
        using ‹*hermitean ?H1*› ‹*?H1* ≠ *mat-zero*› ‹*hermitean ?H2*› ‹*?H2* ≠ *mat-zero*›
        using *hermitean-elems*
        by *auto*

have *uv*: $u1 \neq 0$ $u2 \neq 0$ $v1 \neq 0$ $v2 \neq 0$
using $*(1-4)$
by (*auto simp add*: *vec-cnj-def*)

have *u*: $cor\ ((Re\ (u1/u2))^2) + cor\ ((Im\ (u1/u2))^2) = 1$
using ‹*on-circline-cmat-cvec unit-circle-cmat ?u*› *uv*
apply (*subst of-real-add*[*symmetric*])
apply (*subst complex-mult-cnj*[*symmetric*])
apply (*simp add*: *vec-cnj-def mult.commute*)
done

have *v*: $cor\ ((Re\ (v1/v2))^2) + cor\ ((Im\ (v1/v2))^2) = 1$
using ‹*on-circline-cmat-cvec unit-circle-cmat ?v*› *uv*
apply (*subst of-real-add*[*symmetric*])
apply (*subst complex-mult-cnj*[*symmetric*])
apply (*simp add*: *vec-cnj-def mult.commute*)
done

have
$A1 * (cor\ ((Re\ (u1/u2))^2) + cor\ ((Im\ (u1/u2))^2) + 1) + cor\ (Re\ B1) * cor(2 * Re\ (u1/u2)) + cor\ (Im\ B1) * cor(2 * Im\ (u1/u2)) = 0$
$A2 * (cor\ ((Re\ (u1/u2))^2) + cor\ ((Im\ (u1/u2))^2) + 1) + cor\ (Re\ B2) * cor(2 * Re\ (u1/u2)) + cor\ (Im\ B2) * cor(2 * Im\ (u1/u2)) = 0$
$A1 * (cor\ ((Re\ (v1/v2))^2) + cor\ ((Im\ (v1/v2))^2) + 1) + cor\ (Re\ B1) * cor(2 * Re\ (v1/v2)) + cor\ (Im\ B1) * cor(2 * Im\ (v1/v2)) = 0$
$A2 * (cor\ ((Re\ (v1/v2))^2) + cor\ ((Im\ (v1/v2))^2) + 1) + cor\ (Re\ B2) * cor(2 * Re\ (v1/v2)) + cor\ (Im\ B2) * cor(2 * Im\ (v1/v2)) = 0$
using *circline-equation-quadratic-equation*[*of A1 u1/u2 B1 D1 Re* (*u1/u2*) *Im* (*u1* / *u2*) *Re B1 Im B1*]
using *circline-equation-quadratic-equation*[*of A2 u1/u2 B2 D2 Re* (*u1/u2*) *Im* (*u1* / *u2*) *Re B2 Im B2*]
using *circline-equation-quadratic-equation*[*of A1 v1/v2 B1 D1 Re* (*v1/v2*) *Im* (*v1* / *v2*) *Re B1 Im B1*]
using *circline-equation-quadratic-equation*[*of A2 v1/v2 B2 D2 Re* (*v1/v2*) *Im* (*v1* / *v2*) *Re B2 Im B2*]
using ‹*on-circline-cmat-cvec ?H1 ?u*› ‹*on-circline-cmat-cvec ?H2 ?u*›
using ‹*on-circline-cmat-cvec ?H1 ?v*› ‹*on-circline-cmat-cvec ?H2 ?v*›
using ∗∗ *uv*
by (*simp-all add*: *vec-cnj-def field-simps*)

hence
$A1 + cor\ (Re\ B1) * cor(Re\ (u1/u2)) + cor\ (Im\ B1) * cor(Im\ (u1/u2)) = 0$
$A1 + cor\ (Re\ B1) * cor(Re\ (v1/v2)) + cor\ (Im\ B1) * cor(Im\ (v1/v2)) = 0$
$A2 + cor\ (Re\ B2) * cor(Re\ (u1/u2)) + cor\ (Im\ B2) * cor(Im\ (u1/u2)) = 0$
$A2 + cor\ (Re\ B2) * cor(Re\ (v1/v2)) + cor\ (Im\ B2) * cor(Im\ (v1/v2)) = 0$
using *u v*
by *simp-all algebra+*

hence
$cor\ (Re\ A1 + Re\ B1 * Re\ (u1/u2) + Im\ B1 * Im\ (u1/u2)) = 0$
$cor\ (Re\ A2 + Re\ B2 * Re\ (u1/u2) + Im\ B2 * Im\ (u1/u2)) = 0$
$cor\ (Re\ A1 + Re\ B1 * Re\ (v1/v2) + Im\ B1 * Im\ (v1/v2)) = 0$
$cor\ (Re\ A2 + Re\ B2 * Re\ (v1/v2) + Im\ B2 * Im\ (v1/v2)) = 0$
using ‹*is-real A1*› ‹*is-real A2*›
by *simp-all*

hence
$Re\ A1 + Re\ B1 * Re\ (u1/u2) + Im\ B1 * Im\ (u1/u2) = 0$
$Re\ A1 + Re\ B1 * Re\ (v1/v2) + Im\ B1 * Im\ (v1/v2) = 0$
$Re\ A2 + Re\ B2 * Re\ (u1/u2) + Im\ B2 * Im\ (u1/u2) = 0$
$Re\ A2 + Re\ B2 * Re\ (v1/v2) + Im\ B2 * Im\ (v1/v2) = 0$
using *of-real-eq-0-iff*
by *blast+*

moreover

have $Re(u1/u2) \neq Re(v1/v2) \lor Im(u1/u2) \neq Im(v1/v2)$
proof (*rule ccontr*)
assume ¬ *?thesis*

hence *u1/u2 = v1/v2*
using *complex-eqI* **by** *blast*
thus *False*
using *uv* ‹¬ (*u1*, *u2*) $\approx_v$ (*v1*, *v2*)›
using ∗(*1*) ∗(*2*) *complex-cvec-eq-mix*[*OF* ∗(*1*) ∗(*2*)]
by (*auto simp add*: *field-simps*)
qed

moreover

have *Re A1* ≠ *0* ∨ *Re B1* ≠ *0* ∨ *Im B1* ≠ *0*
using ‹*?H1* ≠ *mat-zero*› ∗∗
by (*metis complex-cnj-zero complex-of-real-Re mat-zero-def of-real-0*)

ultimately

obtain *k* **where**
k: *Re A2* = *k* ∗ *Re A1 Re B2* = *k* ∗ *Re B1 Im B2* = *k* ∗ *Im B1*
using *linear-system-homogenous-3-2*[*of* λ*x y z*. *1* ∗ *x* + *Re* (*u1* / *u2*) ∗ *y* + *Im* (*u1* / *u2*) ∗ *z 1 Re* (*u1*/*u2*) *Im* (*u1*/*u2*)
λ*x y z*. *1* ∗ *x* + *Re* (*v1* / *v2*) ∗ *y* + *Im* (*v1* / *v2*) ∗ *z 1 Re* (*v1*/*v2*) *Im* (*v1*/*v2*)
Re A2 Re B2 Im B2 Re A1 Re B1 Im B1]
by (*auto simp add*: *field-simps*)

have *Re A2* ≠ *0* ∨ *Re B2* ≠ *0* ∨ *Im B2* ≠ *0*
using ‹*?H2* ≠ *mat-zero*› ∗∗
by (*metis complex-cnj-zero complex-of-real-Re mat-zero-def of-real-0*)
hence *k* ≠ *0*
using *k*
by *auto*

show *circline-eq-cmat ?H1 ?H2*
using ∗∗ *k* ‹*k* ≠ *0*›
by (*auto simp add*: *vec-cnj-def*) (*rule-tac x*=*k* **in** *exI*, *auto simp add*: *complex.expand*)
qed
qed
qed

The only h-line that goes trough zero and a non-zero point on the x-axis is the x-axis.

lemma *is-poincare-line-0-real-is-x-axis*:
assumes *is-poincare-line l* 0_h ∈ *circline-set l*
x ∈ *circline-set l* ∩ *circline-set x-axis x* ≠ 0_h *x* ≠ ∞_h
shows *l* = *x-axis*
using *assms*
using *is-poincare-line-trough-zero-trough-infty*[*OF assms*(*1*−*2*)]
using *unique-circline-set*[*of x* 0_h ∞_h]
by *auto*

The only h-line that goes trough zero and a non-zero point on the y-axis is the y-axis.

lemma *is-poincare-line-0-imag-is-y-axis*:
assumes *is-poincare-line l* 0_h ∈ *circline-set l*
y ∈ *circline-set l* ∩ *circline-set y-axis y* ≠ 0_h *y* ≠ ∞_h
shows *l* = *y-axis*
using *assms*
using *is-poincare-line-trough-zero-trough-infty*[*OF assms*(*1*−*2*)]
using *unique-circline-set*[*of y* 0_h ∞_h]
by *auto*

4.1.6 H-isometries preserve h-lines

H-isometries are defined as homographies (actions of Möbius transformations) and antihomographies (compositions of actions of Möbius transformations with conjugation) that fix the unit disc (map it onto itself). They also map h-lines onto h-lines

We prove a bit more general lemma that states that all Möbius transformations that fix the unit circle (not necessary the unit disc) map h-lines onto h-lines

```
lemma unit-circle-fix-preserve-is-poincare-line [simp]:
  assumes unit-circle-fix M is-poincare-line H
  shows is-poincare-line (moebius-circline M H)
  using assms
  unfolding is-poincare-line-iff
proof (safe)
  let ?H' = moebius-ocircline M (of-circline H)
  let ?U' = moebius-ocircline M ounit-circle
  assume ++: unit-circle-fix M perpendicular H unit-circle
  have ounit: ounit-circle = moebius-ocircline M ounit-circle ∨
              ounit-circle = moebius-ocircline M (opposite-ocircline ounit-circle)
    using ++(1) unit-circle-fix-iff[of M]
    by (simp add: inj-of-ocircline moebius-circline-ocircline)

  show perpendicular (moebius-circline M H) unit-circle
  proof (cases pos-oriented ?H')
    case True
    hence *: of-circline (of-ocircline ?H') = ?H'
      using of-circline-of-ocircline-pos-oriented
      by blast
    from ounit show ?thesis
    proof
      assume **: ounit-circle = moebius-ocircline M ounit-circle
      show ?thesis
        using ++
        unfolding perpendicular-def
        by (simp, subst moebius-circline-ocircline, subst *, subst **) simp
    next
      assume **: ounit-circle = moebius-ocircline M (opposite-ocircline ounit-circle)
      show ?thesis
        using ++
        unfolding perpendicular-def
        by (simp, subst moebius-circline-ocircline, subst *, subst **) simp
    qed
  next
    case False
    hence *: of-circline (of-ocircline ?H') = opposite-ocircline ?H'
      by (metis of-circline-of-ocircline pos-oriented-of-circline)
    from ounit show ?thesis
    proof
      assume **: ounit-circle = moebius-ocircline M ounit-circle
      show ?thesis
        using ++
        unfolding perpendicular-def
        by (simp, subst moebius-circline-ocircline, subst *, subst **) simp
    next
      assume **: ounit-circle = moebius-ocircline M (opposite-ocircline ounit-circle)
      show ?thesis
        using ++
        unfolding perpendicular-def
        by (simp, subst moebius-circline-ocircline, subst *, subst **) simp
    qed
  qed
qed simp

lemma unit-circle-fix-preserve-is-poincare-line-iff [simp]:
  assumes unit-circle-fix M
  shows is-poincare-line (moebius-circline M H) ⟷ is-poincare-line H
  using assms
  using unit-circle-fix-preserve-is-poincare-line[of M H]
  using unit-circle-fix-preserve-is-poincare-line[of moebius-inv M moebius-circline M H]
  by (auto simp del: unit-circle-fix-preserve-is-poincare-line)
```

Since h-lines are preserved by transformations that fix the unit circle, so is collinearity.

```
lemma unit-disc-fix-preserve-poincare-collinear [simp]:
  assumes unit-circle-fix M poincare-collinear A
```

shows *poincare-collinear* (*moebius-pt M ‘ A*)
using *assms*
unfolding *poincare-collinear-def*
by (*auto*, *rule-tac x=moebius-circline M p* **in** *exI*, *auto*)

lemma *unit-disc-fix-preserve-poincare-collinear-iff* [*simp*]:
assumes *unit-circle-fix M*
shows *poincare-collinear* (*moebius-pt M ‘ A*) $\longleftrightarrow$ *poincare-collinear A*
using *assms*
using *unit-disc-fix-preserve-poincare-collinear*[*of M A*]
using *unit-disc-fix-preserve-poincare-collinear*[*of moebius-inv M moebius-pt M ‘ A*]
by (*auto simp del*: *unit-disc-fix-preserve-poincare-collinear*)

lemma *unit-disc-fix-preserve-poincare-collinear3* [*simp*]:
assumes *unit-disc-fix M*
shows *poincare-collinear* {*moebius-pt M u*, *moebius-pt M v*, *moebius-pt M w*} $\longleftrightarrow$
poincare-collinear {*u*, *v*, *w*}
using *assms unit-disc-fix-preserve-poincare-collinear-iff*[*of M* {*u*, *v*, *w*}]
by *simp*

Conjugation is also an h-isometry and it preserves h-lines.

lemma *is-poincare-line-conjugate-circline* [*simp*]:
assumes *is-poincare-line H*
shows *is-poincare-line* (*conjugate-circline H*)
using *assms*
by (*transfer*, *transfer*, *auto simp add*: *mat-cnj-def hermitean-def mat-adj-def*)

lemma *is-poincare-line-conjugate-circline-iff* [*simp*]:
shows *is-poincare-line* (*conjugate-circline H*) $\longleftrightarrow$ *is-poincare-line H*
using *is-poincare-line-conjugate-circline*[*of conjugate-circline H*]
by *auto*

Since h-lines are preserved by conjugation, so is collinearity.

lemma *conjugate-preserve-poincare-collinear* [*simp*]:
assumes *poincare-collinear A*
shows *poincare-collinear* (*conjugate ‘ A*)
using *assms*
unfolding *poincare-collinear-def*
by *auto* (*rule-tac x=conjugate-circline p* **in** *exI*, *auto*)

lemma *conjugate-conjugate* [*simp*]: *conjugate ‘ conjugate ‘ A = A*
by (*auto simp add*: *image-iff*)

lemma *conjugate-preserve-poincare-collinear-iff* [*simp*]:
shows *poincare-collinear* (*conjugate ‘ A*) $\longleftrightarrow$ *poincare-collinear A*
using *conjugate-preserve-poincare-collinear*[*of A*]
using *conjugate-preserve-poincare-collinear*[*of conjugate ‘ A*]
by (*auto simp del*: *conjugate-preserve-poincare-collinear*)

4.1.7 Mapping h-lines to x-axis

Each h-line in the Poincaré model can be mapped onto the x-axis (by a unit-disc preserving Möbius transformation).

lemma *ex-unit-disc-fix-is-poincare-line-to-x-axis*:
assumes *is-poincare-line l*
shows $\exists$ *M*. *unit-disc-fix M* $\wedge$ *moebius-circline M l = x-axis*
proof−
from *assms* **obtain** *u v* **where** *u* $\neq$ *v u* $\in$ *unit-disc v* $\in$ *unit-disc* **and** {*u*, *v*} $\subseteq$ *circline-set l*
using *ex-is-poincare-line-points*
by *blast*
then obtain *M* **where** ∗: *unit-disc-fix M moebius-pt M u* = 0_h *moebius-pt M v* $\in$ *positive-x-axis*
using *ex-unit-disc-fix-to-zero-positive-x-axis*[*of u v*]
by *auto*
moreover
hence {0_h, *moebius-pt M v*} $\subseteq$ *circline-set x-axis*

unfolding *positive-x-axis-def*
by *auto*
moreover
have *moebius-pt M v* $\neq$ *0*$_h$
using ‹*u* $\neq$ *v*› *
by (*metis moebius-pt-neq-I*)
moreover
have *moebius-pt M v* $\neq \infty_h$
using ‹*unit-disc-fix M*› ‹*v* $\in$ *unit-disc*›
using *unit-disc-fix-discI*
by *fastforce*
ultimately
show *?thesis*
using ‹*is-poincare-line l*› ‹{*u*, *v*} $\subseteq$ *circline-set l*› ‹*unit-disc-fix M*›
using *is-poincare-line-0-real-is-x-axis*[*of moebius-circline M l moebius-pt M v*]
by (*rule-tac x=M* **in** *exI*, *force*)
qed

When proving facts about h-lines, without loss of generality it can be assumed that h-line is the x-axis (if the property being proved is invariant under Möbius transformations that fix the unit disc).

lemma *wlog-line-x-axis*:
assumes *is-line*: *is-poincare-line H*
assumes *x-axis*: *P x-axis*
assumes *preserves*: $\bigwedge$ *M*. ⟦*unit-disc-fix M*; *P* (*moebius-circline M H*)⟧ $\Longrightarrow$ *P H*
shows *P H*
using *assms*
using *ex-unit-disc-fix-is-poincare-line-to-x-axis*[*of H*]
by *auto*

4.2 Construction of the h-line between the two given points

Next we show how to construct the (unique) h-line between the two given points in the Poincaré model

Geometrically, h-line can be constructed by finding the inverse point of one of the two points and by constructing the circle (or line) trough it and the two given points.
Algebraically, for two given points u and v in $\mathbb{C}$, the h-line matrix coefficients can be $A = i \cdot (u\overline{v} - v\overline{u})$ and $B = i \cdot (v(|u|^2 + 1) - u(|v|^2 + 1))$.
We need to extend this to homogenous coordinates. There are several degenerate cases.
- If $\{z, w\} = \{0_h, \infty_h\}$ then there is no unique h-line (any line trough zero is an h-line).
- If z and w are mutually inverse, then the construction fails (both geometric and algebraic).
- If z and w are different points on the unit circle, then the standard construction fails (only geometric).
- None of this problematic cases occur when z and w are inside the unit disc.
We express the construction algebraically, and construct the Hermitean circline matrix for the two points given in homogenous coordinates. It works correctly in all cases except when the two points are the same or are mutually inverse.

definition *mk-poincare-line-cmat* :: *real* $\Rightarrow$ *complex* $\Rightarrow$ *complex-mat* **where**
[*simp*]: *mk-poincare-line-cmat A B* = (*cor A*, *B*, *cnj B*, *cor A*)

lemma *mk-poincare-line-cmat-zero-iff*:
mk-poincare-line-cmat A B = *mat-zero* $\longleftrightarrow$ *A* = *0* $\wedge$ *B* = *0*
by *auto*

lemma *mk-poincare-line-cmat-hermitean*
[*simp*]: *hermitean* (*mk-poincare-line-cmat A B*)
by *simp*

lemma *mk-poincare-line-cmat-scale*:
cor k $*_{sm}$ *mk-poincare-line-cmat A B* = *mk-poincare-line-cmat* (*k* * *A*) (*k* * *B*)
by *simp*

definition *poincare-line-cvec-cmat* :: *complex-vec* $\Rightarrow$ *complex-vec* $\Rightarrow$ *complex-mat* **where**
[*simp*]: *poincare-line-cvec-cmat z w* =
(*let* (*z1*, *z2*) = *z*;

```
        (w1, w2) = w;
        nom = w1*cnj w2*(z1*cnj z1 + z2*cnj z2) − z1*cnj z2*(w1*cnj w1 + w2*cnj w2);
        den = z1*cnj z2*cnj w1*w2 − w1*cnj w2*cnj z1*z2
     in if den ≠ 0 then
          mk-poincare-line-cmat (Re(i*den)) (i*nom)
        else if z1*cnj z2 ≠ 0 then
          mk-poincare-line-cmat 0 (i*z1*cnj z2)
        else if w1*cnj w2 ≠ 0 then
          mk-poincare-line-cmat 0 (i*w1*cnj w2)
        else
          mk-poincare-line-cmat 0 i)

lemma poincare-line-cvec-cmat-AeqD:
  assumes poincare-line-cvec-cmat z w = (A, B, C, D)
  shows A = D
  using assms
  by (cases z, cases w) (auto split: if-split-asm)

lemma poincare-line-cvec-cmat-hermitean [simp]:
  shows hermitean (poincare-line-cvec-cmat z w)
  by (cases z, cases w) (auto split: if-split-asm simp del: mk-poincare-line-cmat-def)

lemma poincare-line-cvec-cmat-nonzero [simp]:
  assumes z ≠ vec-zero w ≠ vec-zero
  shows  poincare-line-cvec-cmat z w ≠ mat-zero
proof−

  obtain z1 z2 w1 w2 where *: z = (z1, z2) w = (w1, w2)
    by (cases z, cases w, auto)

  let ?den = z1*cnj z2*cnj w1*w2 − w1*cnj w2*cnj z1*z2
  show ?thesis
  proof (cases ?den ≠ 0)
    case True
    have is-real (i * ?den)
      using eq-cnj-iff-real[of i *?den]
      by (simp add: field-simps)
    hence Re (i * ?den) ≠ 0
      using ‹?den ≠ 0›
      by (metis complex-i-not-zero complex-surj mult-eq-0-iff zero-complex.code)
    thus ?thesis
      using * ‹?den ≠ 0›
      by (simp del: mk-poincare-line-cmat-def mat-zero-def add: mk-poincare-line-cmat-zero-iff)
  next
    case False
    thus ?thesis
      using *
      by (simp del: mk-poincare-line-cmat-def mat-zero-def add: mk-poincare-line-cmat-zero-iff)
  qed
qed

lift-definition poincare-line-hcoords-clmat :: complex-homo-coords ⇒ complex-homo-coords ⇒ circline-mat is poincare-line-cvec-cmat
  using poincare-line-cvec-cmat-hermitean poincare-line-cvec-cmat-nonzero
  by simp

lift-definition poincare-line :: complex-homo ⇒ complex-homo ⇒ circline is poincare-line-hcoords-clmat
proof transfer
  fix za zb wa wb
  assume za ≠ vec-zero zb ≠ vec-zero wa ≠ vec-zero wb ≠ vec-zero
  assume za ≈v zb wa ≈v wb
  obtain za1 za2 zb1 zb2 wa1 wa2 wb1 wb2 where
  *: (za1, za2) = za (zb1, zb2) = zb
    (wa1, wa2) = wa (wb1, wb2) = wb
    by (cases za, cases zb, cases wa, cases wb, auto)
  obtain kz kw where
    **: kz ≠ 0 kw ≠ 0 zb1 = kz * za1 zb2 = kz * za2 wb1 = kw * wa1 wb2 = kw * wa2
```

```
  using ‹za ≈v zb› ‹wa ≈v wb› *[symmetric]
  by auto

let ?nom = λ z1 z2 w1 w2. w1*cnj w2*(z1*cnj z1 + z2*cnj z2) − z1*cnj z2*(w1*cnj w1 + w2*cnj w2)
let ?den = λ z1 z2 w1 w2. z1*cnj z2*cnj w1*w2 − w1*cnj w2*cnj z1*z2

show circline-eq-cmat (poincare-line-cvec-cmat za wa)
                      (poincare-line-cvec-cmat zb wb)
proof−
  have ∃ k. k ≠ 0 ∧
         poincare-line-cvec-cmat (zb1, zb2) (wb1, wb2) = cor k *sm poincare-line-cvec-cmat (za1, za2) (wa1, wa2)
  proof (cases ?den za1 za2 wa1 wa2 ≠ 0)
    case True
    hence ?den zb1 zb2 wb1 wb2 ≠ 0
      using **
      by (simp add: field-simps)

    let ?k = kz * cnj kz * kw * cnj kw

    have ?k ≠ 0
      using **
      by simp

    have is-real ?k
      using eq-cnj-iff-real[of ?k]
      by auto

    have cor (Re ?k) = ?k
      using ‹is-real ?k›
      using complex-of-real-Re
      by blast

    have Re ?k ≠ 0
      using ‹?k ≠ 0› ‹cor (Re ?k) = ?k›
      by (metis of-real-0)

    have arg1: Re (i * ?den zb1 zb2 wb1 wb2) = Re ?k * Re (i * ?den za1 za2 wa1 wa2)
      apply (subst **)+
      apply (subst Re-mult-real[symmetric, OF ‹is-real ?k›])
      apply (rule arg-cong[where f=Re])
      apply (simp add: field-simps)
      done
    have arg2: i * ?nom zb1 zb2 wb1 wb2 = ?k * i * ?nom za1 za2 wa1 wa2
      using **
      by (simp add: field-simps)
    have mk-poincare-line-cmat (Re (i*?den zb1 zb2 wb1 wb2)) (i*?nom zb1 zb2 wb1 wb2) =
          cor (Re ?k) *sm mk-poincare-line-cmat (Re (i*?den za1 za2 wa1 wa2)) (i*?nom za1 za2 wa1 wa2)
      using ‹cor (Re ?k) = ?k› ‹is-real ?k›
      apply (subst mk-poincare-line-cmat-scale)
      apply (subst arg1, subst arg2)
      apply (subst ‹cor (Re ?k) = ?k›)+
      apply simp
      done
     thus ?thesis
      using ‹?den za1 za2 wa1 wa2 ≠ 0› ‹?den zb1 zb2 wb1 wb2 ≠ 0›
      using ‹Re ?k ≠ 0› ‹cor (Re ?k) = ?k›
      by (rule-tac x=Re ?k in exI, simp)
  next
    case False
    hence ?den zb1 zb2 wb1 wb2 = 0
      using **
      by (simp add: field-simps)
    show ?thesis
    proof (cases za1*cnj za2 ≠ 0)
      case True
      hence zb1*cnj zb2 ≠ 0
```

```
        using **
        by (simp add: field-simps)

      let ?k = kz * cnj kz

      have ?k ≠ 0 is-real ?k
        using **
        using eq-cnj-iff-real[of ?k]
        by auto
      thus ?thesis
        using ‹za1 * cnj za2 ≠ 0› ‹zb1 * cnj zb2 ≠ 0›
        using ‹¬ (?den za1 za2 wa1 wa2 ≠ 0)› ‹?den zb1 zb2 wb1 wb2 = 0› **
        by (rule-tac x=Re (kz * cnj kz) in exI, auto simp add: complex.expand)
    next
      case False
      hence zb1 * cnj zb2 = 0
        using **
        by (simp add: field-simps)
      show ?thesis
      proof (cases wa1 * cnj wa2 ≠ 0)
        case True
        hence wb1*cnj wb2 ≠ 0
          using **
          by (simp add: field-simps)

        let ?k = kw * cnj kw

        have ?k ≠ 0 is-real ?k
          using **
          using eq-cnj-iff-real[of ?k]
          by auto

        thus ?thesis
          using ‹¬ (za1 * cnj za2 ≠ 0)›
          using ‹wa1 * cnj wa2 ≠ 0› ‹wb1 * cnj wb2 ≠ 0›
          using ‹¬ (?den za1 za2 wa1 wa2 ≠ 0)› ‹?den zb1 zb2 wb1 wb2 = 0› **
          by (rule-tac x=Re (kw * cnj kw) in exI)
            (auto simp add: complex.expand)
      next
        case False
        hence wb1 * cnj wb2 = 0
          using **
          by (simp add: field-simps)
        thus ?thesis
          using ‹¬ (za1 * cnj za2 ≠ 0)› ‹zb1 * cnj zb2 = 0›
          using ‹¬ (wa1 * cnj wa2 ≠ 0)› ‹wb1 * cnj wb2 = 0›
          using ‹¬ (?den za1 za2 wa1 wa2 ≠ 0)› ‹?den zb1 zb2 wb1 wb2 = 0› **
          by simp
      qed
    qed
  qed
  thus ?thesis
    using *[symmetric]
    by simp
 qed
qed
```

4.2.1 Correctness of the construction

For finite points, our definition matches the classic algebraic definition for points in ℂ (given in ordinary, not homogenous coordinates).

```
lemma poincare-line-non-homogenous:
  assumes u ≠ ∞h v ≠ ∞h u ≠ v u ≠ inversion v
  shows let u' = to-complex u;  v' = to-complex v;
          A = i * (u' * cnj v' − v' * cnj u');
          B = i * (v' * ((cmod u')^2 + 1) − u' * ((cmod v')^2 + 1))
```

```
      in poincare-line u v = mk-circline A B (cnj B) A
  using assms
  unfolding unit-disc-def disc-def inversion-def
  apply (simp add: Let-def)
proof (transfer, transfer, safe)
  fix u1 u2 v1 v2
  assume uv: (u1, u2) ≠ vec-zero (v1, v2) ≠ vec-zero
         ¬ (u1, u2) ≈v ∞v ¬ (v1, v2) ≈v ∞v
         ¬ (u1, u2) ≈v (v1, v2) ¬ (u1, u2) ≈v conjugate-cvec (reciprocal-cvec (v1, v2))
  let ?u = to-complex-cvec (u1, u2) and ?v = to-complex-cvec (v1, v2)
  let ?A = i ∗ (?u ∗ cnj ?v − ?v ∗ cnj ?u)
  let ?B = i ∗ (?v ∗ ((cor (cmod ?u))^2 + 1) − ?u ∗ ((cor (cmod ?v))^2 + 1))
  let ?C = − (i ∗ (cnj ?v ∗ ((cor (cmod ?u))^2 + 1) − cnj ?u ∗ ((cor (cmod ?v))^2 + 1)))
  let ?D = ?A
  let ?H = (?A, ?B, ?C, ?D)


  let ?den = u1 ∗ cnj u2 ∗ cnj v1 ∗ v2 − v1 ∗ cnj v2 ∗ cnj u1 ∗ u2

  have u2 ≠ 0 v2 ≠ 0
    using uv
    using inf-cvec-z2-zero-iff
    by blast+

  have ¬ (u1, u2) ≈v (cnj v2, cnj v1)
    using uv(6)
    by (simp add: vec-cnj-def)
  moreover
  have (cnj v2, cnj v1) ≠ vec-zero
    using uv(2)
    by auto
  ultimately
  have ∗: u1 ∗ cnj v1 ≠ u2 ∗ cnj v2 u1 ∗ v2 ≠ u2 ∗ v1
    using uv(5) uv(1) uv(2) ‹u2 ≠ 0› ‹v2 ≠ 0›
    using complex-cvec-eq-mix
    by blast+

  show circline-eq-cmat (poincare-line-cvec-cmat (u1, u2) (v1, v2))
                  (mk-circline-cmat ?A ?B ?C ?D)
  proof (cases ?den ≠ 0)
    case True

    let ?nom = v1 ∗ cnj v2 ∗ (u1 ∗ cnj u1 + u2 ∗ cnj u2) − u1 ∗ cnj u2 ∗ (v1 ∗ cnj v1 + v2 ∗ cnj v2)
    let ?H' = mk-poincare-line-cmat (Re (i ∗ ?den)) (i ∗ ?nom)

    have circline-eq-cmat ?H ?H'
    proof−
      let ?k = (u2 ∗ cnj v2) ∗ (v2 ∗ cnj u2)
      have is-real ?k
        using eq-cnj-iff-real
        by fastforce
      hence cor (Re ?k) = ?k
        using complex-of-real-Re
        by blast

      have Re (i ∗ ?den) = Re ?k ∗ ?A
      proof−
        have ?A = cnj ?A
          by (simp add: field-simps)
        hence is-real ?A
          using eq-cnj-iff-real
          by fastforce
        moreover
        have i ∗ ?den = cnj (i ∗ ?den)
          by (simp add: field-simps)
        hence is-real (i ∗ ?den)
```

```
          using eq-cnj-iff-real
          by fastforce
        hence cor (Re (i ∗ ?den)) = i ∗ ?den
          using complex-of-real-Re
          by blast
        ultimately
        show ?thesis
          using ‹cor (Re ?k) = ?k›
          by (simp add: field-simps)
      qed

      moreover
      have i ∗ ?nom = Re ?k ∗ ?B
        using ‹cor (Re ?k) = ?k› ‹u2 ≠ 0› ‹v2 ≠ 0› complex-mult-cnj-cmod[symmetric]
        by (auto simp add: field-simps)

      moreover
      have ?k ≠ 0
        using ‹u2 ≠ 0› ‹v2 ≠ 0›
        by simp
      hence Re ?k ≠ 0
        using ‹is-real ?k›
        by (metis ‹cor (Re ?k) = ?k› of-real-0)

      ultimately
      show ?thesis
        by simp (rule-tac x=Re ?k in exI, simp add: mult.commute)
    qed

    moreover

    have poincare-line-cvec-cmat (u1, u2) (v1, v2) = ?H′
      using ‹?den ≠ 0›
      unfolding poincare-line-cvec-cmat-def
      by (simp add: Let-def)

    moreover

    hence hermitean ?H′ ∧ ?H′ ≠ mat-zero
      by (metis mk-poincare-line-cmat-hermitean poincare-line-cvec-cmat-nonzero uv(1) uv(2))

    hence hermitean ?H ∧ ?H ≠ mat-zero
      using ‹circline-eq-cmat ?H ?H′›
      using circline-eq-cmat-hermitean-nonzero[of ?H′ ?H] symp-circline-eq-cmat
      unfolding symp-def
      by metis

    hence mk-circline-cmat ?A ?B ?C ?D = ?H
      by simp

    ultimately

    have circline-eq-cmat (mk-circline-cmat ?A ?B ?C ?D)
                      (poincare-line-cvec-cmat (u1, u2) (v1, v2))
      by simp
    thus ?thesis
      using symp-circline-eq-cmat
      unfolding symp-def
      by blast
  next
    case False

    let ?d = v1 ∗ (u1 ∗ cnj u1 / (u2 ∗ cnj u2) + 1) / v2 − u1 ∗ (v1 ∗ cnj v1 / (v2 ∗ cnj v2) + 1) / u2
    let ?cd = cnj v1 ∗ (u1 ∗ cnj u1 / (u2 ∗ cnj u2) + 1) / cnj v2 − cnj u1 ∗ (v1 ∗ cnj v1 / (v2 ∗ cnj v2) + 1) / cnj
u2
```

```
have cnj ?d = ?cd
  by (simp add: mult.commute)

let ?d1 = (v1 / v2) * (cnj u1 / cnj u2) − 1
let ?d2 = u1 / u2 − v1 / v2

have **: ?d = ?d1 * ?d2
  using ‹¬ ?den ≠ 0› ‹u2 ≠ 0› ‹v2 ≠ 0›
  by(simp add: field-simps)

hence ?d ≠ 0
  using ‹¬ ?den ≠ 0› ‹u2 ≠ 0› ‹v2 ≠ 0› *
  by auto (simp add: field-simps)+

have is-real ?d1
proof−
  have cnj ?d1 = ?d1
    using ‹¬ ?den ≠ 0› ‹u2 ≠ 0› ‹v2 ≠ 0› *
    by (simp add: field-simps)
  thus ?thesis
    using eq-cnj-iff-real
    by blast
qed

show ?thesis
proof (cases u1 * cnj u2 ≠ 0)
  case True
  let ?nom = u1 * cnj u2
  let ?H' = mk-poincare-line-cmat 0 (i * ?nom)

  have circline-eq-cmat ?H ?H'
  proof−

    let ?k = (u1 * cnj u2) / ?d

    have is-real ?k
    proof−
      have is-real ((u1 * cnj u2) / ?d2)
      proof−
        let ?rhs = (u2 * cnj u2) / (1 − (v1*u2)/(u1*v2))

        have 1: (u1 * cnj u2) / ?d2 = ?rhs
          using ‹¬ ?den ≠ 0› ‹u2 ≠ 0› ‹v2 ≠ 0› * ‹u1 * cnj u2 ≠ 0›
          by (simp add: field-simps)
        moreover
        have cnj ?rhs = ?rhs
        proof−
          have cnj (1 − v1 * u2 / (u1 * v2)) = 1 − v1 * u2 / (u1 * v2)
            using ‹¬ ?den ≠ 0› ‹u2 ≠ 0› ‹v2 ≠ 0› * ‹u1 * cnj u2 ≠ 0›
            by (simp add: field-simps)
          moreover
          have cnj (u2 * cnj u2) = u2 * cnj u2
            by simp
          ultimately
          show ?thesis
            by simp
        qed

        ultimately

        show ?thesis
          using eq-cnj-iff-real
          by fastforce
      qed

      thus ?thesis
```

using ∗∗ ‹*is-real ?d1*›
by (*metis complex-cnj-divide divide-divide-eq-left′ eq-cnj-iff-real*)
qed

have *?k* ≠ *0*
using ‹*?d* ≠ *0*› ‹*u1* ∗ *cnj u2* ≠ *0*›
by *simp*

have *cnj ?k* = *?k*
using ‹*is-real ?k*›
using *eq-cnj-iff-real* **by** *blast*

have *Re ?k* ≠ *0*
using ‹*?k* ≠ *0*› ‹*is-real ?k*›
by (*metis complex.expand zero-complex.simps*(*1*) *zero-complex.simps*(*2*))

have *u1* ∗ *cnj u2* = *?k* ∗ *?d*
using ‹*?d* ≠ *0*›
by *simp*

moreover

hence *cnj u1* ∗ *u2* = *cnj ?k* ∗ *cnj ?d*
by (*metis complex-cnj-cnj complex-cnj-mult*)
hence *cnj u1* ∗ *u2* = *?k* ∗ *?cd*
using ‹*cnj ?k* = *?k*› ‹*cnj ?d* = *?cd*›
by *metis*

ultimately

show *?thesis*
using ‹~ *?den* ≠ *0*› ‹*u1* ∗ *cnj u2* ≠ *0*› ‹*u2* ≠ *0*› ‹*v2* ≠ *0*› ‹*Re ?k* ≠ *0*› ‹*is-real ?k*› ‹*?d* ≠ *0*›
using *complex-mult-cnj-cmod*[*symmetric*, *of u1*]
using *complex-mult-cnj-cmod*[*symmetric*, *of v1*]
using *complex-mult-cnj-cmod*[*symmetric*, *of u2*]
using *complex-mult-cnj-cmod*[*symmetric*, *of v2*]
apply (*simp add*: *power-divide norm-mult norm-divide*)
apply (*rule-tac x*=*Re ?k* **in** *exI*)
apply *simp*
apply (*simp add*: *field-simps*)
done
qed

moreover

have *poincare-line-cvec-cmat* (*u1*, *u2*) (*v1*, *v2*) = *?H′*
using ‹¬ *?den* ≠ *0*› ‹*u1* ∗ *cnj u2* ≠ *0*›
unfolding *poincare-line-cvec-cmat-def*
by (*simp add*: *Let-def*)

moreover

hence *hermitean ?H′* ∧ *?H′* ≠ *mat-zero*
by (*metis mk-poincare-line-cmat-hermitean poincare-line-cvec-cmat-nonzero uv*(*1*) *uv*(*2*))

hence *hermitean ?H* ∧ *?H* ≠ *mat-zero*
using ‹*circline-eq-cmat ?H ?H′*›
using *circline-eq-cmat-hermitean-nonzero*[*of ?H′ ?H*] *symp-circline-eq-cmat*
unfolding *symp-def*
by *metis*

hence *mk-circline-cmat ?A ?B ?C ?D* = *?H*
by *simp*

ultimately

```
  have circline-eq-cmat (mk-circline-cmat ?A ?B ?C ?D)
                        (poincare-line-cvec-cmat (u1, u2) (v1, v2))
    by simp
  thus ?thesis
    using symp-circline-eq-cmat
    unfolding symp-def
    by blast
next
  case False
  show ?thesis
  proof (cases v1 * cnj v2 ≠ 0)
    case True
    let ?nom = v1 * cnj v2
    let ?H' = mk-poincare-line-cmat 0 (i * ?nom)

    have circline-eq-cmat ?H ?H'
    proof-
      let ?k = (v1 * cnj v2) / ?d

      have is-real ?k
      proof-
      have is-real ((v1 * cnj v2) / ?d2)
      proof-
        let ?rhs = (v2 * cnj v2) / ((u1*v2)/(u2*v1) - 1)

        have 1: (v1 * cnj v2) / ?d2 = ?rhs
          using ‹¬ ?den ≠ 0› ‹u2 ≠ 0› ‹v2 ≠ 0› * ‹v1 * cnj v2 ≠ 0›
          by (simp add: field-simps)
        moreover
        have cnj ?rhs = ?rhs
        proof-
          have cnj (u1 * v2 / (u2 * v1) - 1) = u1 * v2 / (u2 * v1) - 1
            using ‹¬ ?den ≠ 0› ‹u2 ≠ 0› ‹v2 ≠ 0› * ‹v1 * cnj v2 ≠ 0›
            by (simp add: field-simps)
          moreover
          have cnj (v2 * cnj v2) = v2 * cnj v2
            by simp
          ultimately
          show ?thesis
            by simp
        qed

        ultimately

        show ?thesis
          using eq-cnj-iff-real
          by fastforce
      qed

      thus ?thesis
        using ** ‹is-real ?d1›
        by (metis complex-cnj-divide divide-divide-eq-left' eq-cnj-iff-real)
    qed

    have ?k ≠ 0
      using ‹?d ≠ 0› ‹v1 * cnj v2 ≠ 0›
      by simp

    have cnj ?k = ?k
      using ‹is-real ?k›
      using eq-cnj-iff-real by blast

    have Re ?k ≠ 0
      using ‹?k ≠ 0› ‹is-real ?k›
      by (metis complex.expand zero-complex.simps(1) zero-complex.simps(2))
```

```
    have v1 * cnj v2 = ?k * ?d
      using ‹?d ≠ 0›
      by simp

    moreover

    hence cnj v1 * v2 = cnj ?k * cnj ?d
      by (metis complex-cnj-cnj complex-cnj-mult)
    hence cnj v1 * v2 = ?k * ?cd
      using ‹cnj ?k = ?k› ‹cnj ?d = ?cd›
      by metis

    ultimately

    show ?thesis
      using ‹~ ?den ≠ 0› ‹v1 * cnj v2 ≠ 0› ‹u2 ≠ 0› ‹v2 ≠ 0› ‹Re ?k ≠ 0› ‹is-real ?k› ‹?d ≠ 0›
      using complex-mult-cnj-cmod[symmetric, of u1]
      using complex-mult-cnj-cmod[symmetric, of v1]
      using complex-mult-cnj-cmod[symmetric, of u2]
      using complex-mult-cnj-cmod[symmetric, of v2]
      apply (simp add: power-divide norm-mult norm-divide)
      apply (rule-tac x=Re ?k in exI)
      apply simp
      apply (simp add: field-simps)
      done
    qed

    moreover

    have poincare-line-cvec-cmat (u1, u2) (v1, v2) = ?H'
      using ‹¬ ?den ≠ 0› ‹¬ u1 * cnj u2 ≠ 0› ‹v1 * cnj v2 ≠ 0›
      unfolding poincare-line-cvec-cmat-def
      by (simp add: Let-def)

    moreover

    hence hermitean ?H' ∧ ?H' ≠ mat-zero
      by (metis mk-poincare-line-cmat-hermitean poincare-line-cvec-cmat-nonzero uv(1) uv(2))

    hence hermitean ?H ∧ ?H ≠ mat-zero
      using ‹circline-eq-cmat ?H ?H'›
      using circline-eq-cmat-hermitean-nonzero[of ?H' ?H] symp-circline-eq-cmat
      unfolding symp-def
      by metis

    hence mk-circline-cmat ?A ?B ?C ?D = ?H
      by simp

    ultimately

    have circline-eq-cmat (mk-circline-cmat ?A ?B ?C ?D)
                         (poincare-line-cvec-cmat (u1, u2) (v1, v2))
      by simp
    thus ?thesis
      using symp-circline-eq-cmat
      unfolding symp-def
      by blast
  next
    case False
    hence False
      using ‹¬ ?den ≠ 0› ‹¬ u1 * cnj u2 ≠ 0› uv
      by (simp add: ‹u2 ≠ 0› ‹v2 ≠ 0›)
    thus ?thesis
      by simp
  qed
qed
```

```
  qed
qed
```

Our construction (in homogenous coordinates) always yields an h-line that contain two starting points (this also holds for all degenerate cases except when points are the same).

```
lemma poincare-line [simp]:
  assumes z ≠ w
  shows on-circline (poincare-line z w) z
        on-circline (poincare-line z w) w
proof−
  have on-circline (poincare-line z w) z ∧ on-circline (poincare-line z w) w
    using assms
  proof (transfer, transfer)
    fix z w
    assume vz: z ≠ vec-zero w ≠ vec-zero
    obtain z1 z2 w1 w2 where
    zw: (z1, z2) = z (w1, w2) = w
      by (cases z, cases w, auto)

    let ?den = z1∗cnj z2∗cnj w1∗w2 − w1∗cnj w2∗cnj z1∗z2
    have ∗: cor (Re (i ∗ ?den)) = i ∗ ?den
    proof−
      have cnj ?den = −?den
        by auto
      hence is-imag ?den
        using eq-minus-cnj-iff-imag[of ?den]
        by simp
      thus ?thesis
        using complex-of-real-Re[of i ∗ ?den]
        by simp
    qed
    show on-circline-cmat-cvec (poincare-line-cvec-cmat z w) z ∧
          on-circline-cmat-cvec (poincare-line-cvec-cmat z w) w
      unfolding poincare-line-cvec-cmat-def mk-poincare-line-cmat-def
      apply (subst zw[symmetric])+
      unfolding Let-def prod.case
      apply (subst ∗)+
      by (auto simp add: vec-cnj-def field-simps)
  qed
  thus on-circline (poincare-line z w) z on-circline (poincare-line z w) w
    by auto
qed

lemma poincare-line-circline-set [simp]:
  assumes z ≠ w
  shows z ∈ circline-set (poincare-line z w)
        w ∈ circline-set (poincare-line z w)
  using assms
  by (auto simp add: circline-set-def)
```

When the points are different, the constructed line matrix always has a negative determinant

```
lemma poincare-line-type:
  assumes z ≠ w
  shows circline-type (poincare-line z w) = −1
proof−
  have ∃ a b. a ≠ b ∧ {a, b} ⊆ circline-set (poincare-line z w)
    using poincare-line[of z w] assms
    unfolding circline-set-def
    by (rule-tac x=z in exI, rule-tac x=w in exI, simp)
  thus ?thesis
    using circline-type[of poincare-line z w]
    using circline-type-pos-card-eq0[of poincare-line z w]
    using circline-type-zero-card-eq1[of poincare-line z w]
    by auto
qed
```

The constructed line is an h-line in the Poincaré model (in all cases when the two points are different)

lemma *is-poincare-line-poincare-line* [*simp*]:
  assumes $z \neq w$
  shows *is-poincare-line* (*poincare-line z w*)
  using *poincare-line-type*[*of z w*, *OF assms*]
proof (*transfer*, *transfer*)
  fix *z w*
  assume *vz*: $z \neq$ *vec-zero* $w \neq$ *vec-zero*
  obtain *A B C D* **where** ∗: *poincare-line-cvec-cmat z w* = (*A*, *B*, *C*, *D*)
    by (*cases poincare-line-cvec-cmat z w*) *auto*
  assume *circline-type-cmat* (*poincare-line-cvec-cmat z w*) = − *1*
  thus *is-poincare-line-cmat* (*poincare-line-cvec-cmat z w*)
    using *vz* ∗
    using *poincare-line-cvec-cmat-hermitean*[*of z w*]
    using *poincare-line-cvec-cmat-nonzero*[*of z w*]
    using *poincare-line-cvec-cmat-AeqD*[*of z w A B C D*]
    using *hermitean-elems*[*of A B C D*]
    using *cmod-power2*[*of D*] *cmod-power2*[*of C*]
    unfolding *is-poincare-line-cmat-def*
    by (*simp del*: *poincare-line-cvec-cmat-def add*: *sgn-1-neg power2-eq-square*)
qed

When the points are different, the constructed h-line between two points also contains their inverses

lemma *poincare-line-inversion*:
  assumes $z \neq w$
  shows *on-circline* (*poincare-line z w*) (*inversion z*)
      on-circline (*poincare-line z w*) (*inversion w*)
  using *assms*
  using *is-poincare-line-poincare-line*[*OF* ‹$z \neq w$›]
  using *is-poincare-line-inverse-point*
  unfolding *circline-set-def*
  by *auto*

When the points are different, the onstructed h-line between two points contains the inverse of its every point

lemma *poincare-line-inversion-full*:
  assumes $u \neq v$
  assumes *on-circline* (*poincare-line u v*) *x*
  shows *on-circline* (*poincare-line u v*) (*inversion x*)
  using *is-poincare-line-inverse-point*[*of poincare-line u v x*]
  using *is-poincare-line-poincare-line*[*OF* ‹$u \neq v$›] *assms*
  unfolding *circline-set-def*
  by *simp*

4.2.2 Existence of h-lines

There is an h-line trough every point in the Poincaré model

lemma *ex-poincare-line-one-point*:
  shows ∃ *l*. *is-poincare-line l* ∧ *z* ∈ *circline-set l*
proof (*cases* $z = 0_h$)
  case *True*
  thus *?thesis*
    by (*rule-tac x=x-axis* **in** *exI*) *simp*
next
  case *False*
  thus *?thesis*
    by (*rule-tac x=poincare-line* 0_h *z* **in** *exI*) *auto*
qed

lemma *poincare-collinear-singleton* [*simp*]:
  assumes *u* ∈ *unit-disc*
  shows *poincare-collinear* {*u*}
  using *assms*
  using *ex-poincare-line-one-point*[*of u*]
  by (*auto simp add*: *poincare-collinear-def*)

There is an h-line trough every two points in the Poincaré model

lemma *ex-poincare-line-two-points*:
  assumes $z \neq w$
  shows $\exists$ *l. is-poincare-line l* $\wedge$ *z* $\in$ *circline-set l* $\wedge$ *w* $\in$ *circline-set l*
  using *assms*
  by (*rule-tac x=poincare-line z w* **in** *exI*, *simp*)

lemma *poincare-collinear-doubleton* [*simp*]:
  assumes *u* $\in$ *unit-disc v* $\in$ *unit-disc*
  shows *poincare-collinear* {*u*, *v*}
  using *assms*
  using *ex-poincare-line-one-point*[*of u*]
  using *ex-poincare-line-two-points*[*of u v*]
  by (*cases u* = *v*) (*simp-all add*: *poincare-collinear-def*)

4.2.3 Uniqueness of h-lines

The only h-line between two points is the one obtained by the line-construction.

First we show this only for two different points inside the disc.

lemma *unique-poincare-line*:
  assumes *in-disc*: *u* $\neq$ *v u* $\in$ *unit-disc v* $\in$ *unit-disc*
  assumes *on-l*: *u* $\in$ *circline-set l v* $\in$ *circline-set l is-poincare-line l*
  shows *l* = *poincare-line u v*
  using *assms*
  using *unique-is-poincare-line*[*of u v l poincare-line u v*]
  unfolding *circline-set-def*
  by *auto*

The assumption that the points are inside the disc can be relaxed.

lemma *unique-poincare-line-general*:
  assumes *in-disc*: *u* $\neq$ *v u* $\neq$ *inversion v*
  assumes *on-l*: *u* $\in$ *circline-set l v* $\in$ *circline-set l is-poincare-line l*
  shows *l* = *poincare-line u v*
  using *assms*
  using *unique-is-poincare-line-general*[*of u v l poincare-line u v*]
  unfolding *circline-set-def*
  by *auto*

The explicit line construction enables us to prove that there exists a unique h-line through any given two h-points (uniqueness part was already shown earlier).

First we show this only for two different points inside the disc.

lemma *ex1-poincare-line*:
  assumes *u* $\neq$ *v u* $\in$ *unit-disc v* $\in$ *unit-disc*
  shows $\exists$! *l. is-poincare-line l* $\wedge$ *u* $\in$ *circline-set l* $\wedge$ *v* $\in$ *circline-set l*
proof (*rule ex1I*)
  let *?l* = *poincare-line u v*
  show *is-poincare-line ?l* $\wedge$ *u* $\in$ *circline-set ?l* $\wedge$ *v* $\in$ *circline-set ?l*
    using *assms*
    unfolding *circline-set-def*
    by *auto*
next
  fix *l*
  assume *is-poincare-line l* $\wedge$ *u* $\in$ *circline-set l* $\wedge$ *v* $\in$ *circline-set l*
  thus *l* = *poincare-line u v*
    using *unique-poincare-line assms*
    by *auto*
qed

The assumption that the points are in the disc can be relaxed.

lemma *ex1-poincare-line-general*:
  assumes *u* $\neq$ *v u* $\neq$ *inversion v*
  shows $\exists$! *l. is-poincare-line l* $\wedge$ *u* $\in$ *circline-set l* $\wedge$ *v* $\in$ *circline-set l*
proof (*rule ex1I*)

let *?l* = *poincare-line u v*
show *is-poincare-line ?l* ∧ *u* ∈ *circline-set ?l* ∧ *v* ∈ *circline-set ?l*
using *assms*
unfolding *circline-set-def*
by *auto*
next
fix *l*
assume *is-poincare-line l* ∧ *u* ∈ *circline-set l* ∧ *v* ∈ *circline-set l*
thus *l* = *poincare-line u v*
using *unique-poincare-line-general assms*
by *auto*
qed

4.2.4 Some consequences of line uniqueness

H-line *uv* is the same as the h-line *vu*.

lemma *poincare-line-sym*:
assumes *u* ∈ *unit-disc v* ∈ *unit-disc u* ≠ *v*
shows *poincare-line u v* = *poincare-line v u*
using *assms*
using *unique-poincare-line*[*of u v poincare-line v u*]
by *simp*

lemma *poincare-line-sym-general*:
assumes *u* ≠ *v u* ≠ *inversion v*
shows *poincare-line u v* = *poincare-line v u*
using *assms*
using *unique-poincare-line-general*[*of u v poincare-line v u*]
by *simp*

Each h-line is the h-line constructed out of its two arbitrary different points.

lemma *ex-poincare-line-points*:
assumes *is-poincare-line H*
shows ∃ *u v*. *u* ∈ *unit-disc* ∧ *v* ∈ *unit-disc* ∧ *u* ≠ *v* ∧ *H* = *poincare-line u v*
using *assms*
using *ex-is-poincare-line-points*
using *unique-poincare-line*[**where** *l*=*H*]
by *fastforce*

If an h-line contains two different points on x-axis/y-axis then it is the x-axis/y-axis.

lemma *poincare-line-0-real-is-x-axis*:
assumes *x* ∈ *circline-set x-axis x* ≠ 0_h *x* ≠ ∞_h
shows *poincare-line* 0_h *x* = *x-axis*
using *assms*
using *is-poincare-line-0-real-is-x-axis*[*of poincare-line* 0_h *x x*]
by *auto*

lemma *poincare-line-0-imag-is-y-axis*:
assumes *y* ∈ *circline-set y-axis y* ≠ 0_h *y* ≠ ∞_h
shows *poincare-line* 0_h *y* = *y-axis*
using *assms*
using *is-poincare-line-0-imag-is-y-axis*[*of poincare-line* 0_h *y y*]
by *auto*

lemma *poincare-line-x-axis*:
assumes *x* ∈ *unit-disc y* ∈ *unit-disc x* ∈ *circline-set x-axis y* ∈ *circline-set x-axis x* ≠ *y*
shows *poincare-line x y* = *x-axis*
using *assms*
using *unique-poincare-line*
by *auto*

lemma *poincare-line-minus-one-one* [*simp*]:
shows *poincare-line* (*of-complex* (−*1*)) (*of-complex 1*) = *x-axis*
proof−
have 0_h ∈ *circline-set* (*poincare-line* (*of-complex* (−*1*)) (*of-complex 1*))

```
    unfolding circline-set-def
    by simp (transfer, transfer, simp add: vec-cnj-def)
  hence poincare-line 0h (of-complex 1) = poincare-line (of-complex (−1)) (of-complex 1)
    by (metis is-poincare-line-poincare-line is-poincare-line-trough-zero-trough-infty not-zero-on-unit-circle of-complex-inj
of-complex-one one-neq-neg-one one-on-unit-circle poincare-line-0-real-is-x-axis poincare-line-circline-set(2) reciprocal-involution
reciprocal-one reciprocal-zero unique-circline-01inf')
  thus ?thesis
    using poincare-line-0-real-is-x-axis[of of-complex 1]
    by auto
qed
```

4.2.5 Transformations of constructed lines

Unit dics preserving Möbius transformations preserve the h-line construction

```
lemma unit-disc-fix-preserve-poincare-line [simp]:
  assumes unit-disc-fix M u ∈ unit-disc v ∈ unit-disc u ≠ v
  shows poincare-line (moebius-pt M u) (moebius-pt M v) = moebius-circline M (poincare-line u v)
proof (rule unique-poincare-line[symmetric])
  show moebius-pt M u ≠ moebius-pt M v
    using ‹u ≠ v›
    by auto
next
  show moebius-pt M u ∈ circline-set (moebius-circline M (poincare-line u v))
       moebius-pt M v ∈ circline-set (moebius-circline M (poincare-line u v))
    unfolding circline-set-def
    using moebius-circline[of M poincare-line u v] ‹u ≠ v›
    by auto
next
  from assms(1) have unit-circle-fix M
    by simp
  thus is-poincare-line (moebius-circline M (poincare-line u v))
    using unit-circle-fix-preserve-is-poincare-line assms
    by auto
next
  show moebius-pt M u ∈ unit-disc moebius-pt M v ∈ unit-disc
    using assms(2−3) unit-disc-fix-iff[OF assms(1)]
    by auto
qed
```

Conjugate preserve the h-line construction

```
lemma conjugate-preserve-poincare-line [simp]:
  assumes u ∈ unit-disc v ∈ unit-disc u ≠ v
  shows poincare-line (conjugate u) (conjugate v) = conjugate-circline (poincare-line u v)
proof−
  have conjugate u ≠ conjugate v
    using ‹u ≠ v›
    by (auto simp add: conjugate-inj)
  moreover
  have conjugate u ∈ unit-disc conjugate v ∈ unit-disc
    using assms
    by auto
  moreover
  have conjugate u ∈ circline-set (conjugate-circline (poincare-line u v))
       conjugate v ∈ circline-set (conjugate-circline (poincare-line u v))
    using ‹u ≠ v›
    by simp-all
  moreover
  have is-poincare-line (conjugate-circline (poincare-line u v))
    using is-poincare-line-poincare-line[OF ‹u ≠ v›]
    by simp
  ultimately
  show ?thesis
    using unique-poincare-line[of conjugate u conjugate v conjugate-circline (poincare-line u v)]
    by simp
qed
```

4.2.6 Collinear points and h-lines

```
lemma poincare-collinear3-poincare-line-general:
  assumes poincare-collinear {a, a1, a2} a1 ≠ a2 a1 ≠ inversion a2
  shows a ∈ circline-set (poincare-line a1 a2)
  using assms
  using poincare-collinear-def unique-poincare-line-general
  by auto

lemma poincare-line-poincare-collinear3-general:
  assumes a ∈ circline-set (poincare-line a1 a2) a1 ≠ a2
  shows poincare-collinear {a, a1, a2}
  using assms
  unfolding poincare-collinear-def
  by (rule-tac x=poincare-line a1 a2 in exI, simp)


lemma poincare-collinear3-poincare-lines-equal-general:
  assumes poincare-collinear {a, a1, a2} a ≠ a1 a ≠ a2 a ≠ inversion a1 a ≠ inversion a2
  shows poincare-line a a1 = poincare-line a a2
  using assms
  using unique-poincare-line-general[of a a2 poincare-line a a1]
  by (simp add: insert-commute poincare-collinear3-poincare-line-general)
```

4.2.7 Points collinear with 0_h

```
lemma poincare-collinear-zero-iff:
  assumes of-complex y' ∈ unit-disc and of-complex z' ∈ unit-disc and
        y' ≠ z' and y' ≠ 0 and z' ≠ 0
  shows poincare-collinear {0h, of-complex y', of-complex z'} ⟷
       y'*cnj z' = cnj y'*z' (is ?lhs ⟷ ?rhs)
proof−
  have of-complex y' ≠ of-complex z'
    using assms
    using of-complex-inj
    by blast
  show ?thesis
  proof
    assume ?lhs
    hence 0h ∈ circline-set (poincare-line (of-complex y') (of-complex z'))
      using unique-poincare-line[of of-complex y' of-complex z']
      using assms ‹of-complex y' ≠ of-complex z'›
      unfolding poincare-collinear-def
      by auto
    moreover
    let ?mix = y' * cnj z' − cnj y' * z'
    have is-real (i * ?mix)
      using eq-cnj-iff-real[of ?mix]
      by auto
    hence y' * cnj z' = cnj y' * z' ⟷ Re (i * ?mix) = 0
      using complex.expand[of i * ?mix 0]
      by (metis complex-i-not-zero eq-iff-diff-eq-0 mult-eq-0-iff zero-complex.simps(1) zero-complex.simps(2))
    ultimately
    show ?rhs
      using ‹y' ≠ z'› ‹y' ≠ 0› ‹z' ≠ 0›
      unfolding circline-set-def
       by simp (transfer, transfer, auto simp add: vec-cnj-def split: if-split-asm, metis Re-complex-of-real Re-mult-real
Im-complex-of-real)
  next
    assume ?rhs
    thus ?lhs
      using assms ‹of-complex y' ≠ of-complex z'›
      unfolding poincare-collinear-def
      unfolding circline-set-def
      apply (rule-tac x=poincare-line (of-complex y') (of-complex z') in exI)
      apply auto
      apply (transfer, transfer, simp add: vec-cnj-def)
```

```
    done
  qed
qed

lemma poincare-collinear-zero-polar-form:
  assumes poincare-collinear {0h, of-complex x, of-complex y} and
          x ≠ 0 and y ≠ 0 and of-complex x ∈ unit-disc and of-complex y ∈ unit-disc
  shows ∃ φ rx ry. x = cor rx ∗ cis φ ∧ y = cor ry ∗ cis φ ∧ rx ≠ 0 ∧ ry ≠ 0
proof−
  from ‹x ≠ 0› ‹y ≠ 0› obtain φ φ' rx ry where
    polar: x = cor rx ∗ cis φ y = cor ry ∗ cis φ' and  φ = Arg x φ' = Arg y
    by (metis cmod-cis)
  hence rx ≠ 0 ry ≠ 0
    using ‹x ≠ 0› ‹y ≠ 0›
    by auto
  have of-complex y ∈ circline-set (poincare-line 0h (of-complex x))
    using assms
    using unique-poincare-line[of 0h of-complex x]
    unfolding poincare-collinear-def
    unfolding circline-set-def
    using of-complex-zero-iff
    by fastforce
  hence cnj x ∗ y = x ∗ cnj y
    using ‹x ≠ 0› ‹y ≠ 0›
    unfolding circline-set-def
    by simp (transfer, transfer, simp add: vec-cnj-def field-simps)
  hence cis(φ' − φ) = cis(φ − φ')
    using polar ‹rx ≠ 0› ‹ry ≠ 0›
    by (simp add: cis-mult)
  hence sin (φ' − φ) = 0
    using cis-diff-cis-opposite[of φ' − φ]
    by simp
  then obtain k :: int where φ' − φ = k ∗ pi
    using sin-zero-iff-int2[of φ' − φ]
    by auto
  hence ∗: φ' = φ + k ∗ pi
    by simp
  show ?thesis
  proof (cases even k)
    case True
    then obtain k' where k = 2∗k'
      using evenE by blast
    hence cis φ = cis φ'
      using ∗ cos-periodic-int sin-periodic-int
      by (simp add: cis.ctr field-simps)
    thus ?thesis
      using polar ‹rx ≠ 0› ‹ry ≠ 0›
      by (rule-tac x=φ in exI, rule-tac x=rx in exI, rule-tac x=ry in exI) simp
  next
    case False
    then obtain k' where k = 2∗k' + 1
      using oddE by blast
    hence cis φ = − cis φ'
      using ∗ cos-periodic-int sin-periodic-int
      by (simp add: cis.ctr complex-minus field-simps)
    thus ?thesis
      using polar ‹rx ≠ 0› ‹ry ≠ 0›
      by (rule-tac x=φ in exI, rule-tac x=rx in exI, rule-tac x=−ry in exI) simp
  qed
qed

end
theory Poincare-Lines-Ideal-Points
imports Poincare-Lines
begin
```

4.3 Ideal points of h-lines

Ideal points of an h-line are points where the h-line intersects the unit disc.

4.3.1 Calculation of ideal points

We decided to define ideal points constructively, i.e., we calculate the coordinates of ideal points for a given h-line explicitly. Namely, if the h-line is determined by A and B, the two intersection points are

$$\frac{B}{|B|^2}\left(-A \pm i \cdot \sqrt{|B|^2 - A^2}\right).$$

definition *calc-ideal-point1-cvec* :: *complex* ⇒ *complex* ⇒ *complex-vec* **where**
[*simp*]: *calc-ideal-point1-cvec A B* =
(*let discr* = *Re* ((*cmod B*)^2 − (*Re A*)^2) *in*
(*B*∗(−*A* − i∗*sqrt*(*discr*)), (*cmod B*)^2))

definition *calc-ideal-point2-cvec* :: *complex* ⇒ *complex* ⇒ *complex-vec* **where**
[*simp*]: *calc-ideal-point2-cvec A B* =
(*let discr* = *Re* ((*cmod B*)^2 − (*Re A*)^2) *in*
(*B*∗(−*A* + i∗*sqrt*(*discr*)), (*cmod B*)^2))

definition *calc-ideal-points-cmat-cvec* :: *complex-mat* ⇒ *complex-vec set* **where**
[*simp*]: *calc-ideal-points-cmat-cvec H* =
(*if is-poincare-line-cmat H then*
let (*A*, *B*, *C*, *D*) = *H*
in {*calc-ideal-point1-cvec A B*, *calc-ideal-point2-cvec A B*}
else
{(−*1*, *1*), (*1*, *1*)})

lift-definition *calc-ideal-points-clmat-hcoords* :: *circline-mat* ⇒ *complex-homo-coords set* **is** *calc-ideal-points-cmat-cvec*
by (*auto simp add*: *Let-def split*: *if-split-asm*)

lift-definition *calc-ideal-points* :: *circline* ⇒ *complex-homo set* **is** *calc-ideal-points-clmat-hcoords*
proof *transfer*
fix *H1 H2*
assume *hh*: *hermitean H1* ∧ *H1* ≠ *mat-zero hermitean H2* ∧ *H2* ≠ *mat-zero*
obtain *A1 B1 C1 D1 A2 B2 C2 D2* **where** ∗: *H1* = (*A1*, *B1*, *C1*, *D1*) *H2* = (*A2*, *B2*, *C2*, *D2*)
by (*cases H1*, *cases H2*, *auto*)
assume *circline-eq-cmat H1 H2*
then obtain *k* **where** *k*: *k* ≠ *0 H2* = *cor k* $*_{sm}$ *H1*
by *auto*
thus *rel-set* ($\approx_v$) (*calc-ideal-points-cmat-cvec H1*) (*calc-ideal-points-cmat-cvec H2*)
proof (*cases is-poincare-line-cmat H1*)
case *True*
hence *is-poincare-line-cmat H2*
using *k* ∗ *hermitean-mult-real*[*of H1 k*] *hh*
by (*auto simp add*: *power2-eq-square norm-mult*)
have ∗∗: *sqrt* (|*k*| ∗ *cmod B1* ∗ (|*k*| ∗ *cmod B1*) − *k* ∗ *Re D1* ∗ (*k* ∗ *Re D1*)) =
|*k*| ∗ *sqrt*(*cmod B1* ∗ *cmod B1* − *Re D1* ∗ *Re D1*)
proof−
have |*k*| ∗ *cmod B1* ∗ (|*k*| ∗ *cmod B1*) − *k* ∗ *Re D1* ∗ (*k* ∗ *Re D1*) =
k^2 ∗ (*cmod B1* ∗ *cmod B1* − *Re D1* ∗ *Re D1*)
by (*simp add*: *power2-eq-square field-simps*)
thus *?thesis*
by (*simp add*: *real-sqrt-mult*)
qed
show *?thesis*
using ‹*is-poincare-line-cmat H1*› ‹*is-poincare-line-cmat H2*›
using ∗ *k*
apply (*simp add*: *Let-def*)
apply *safe*
apply (*simp add*: *power2-eq-square rel-set-def norm-mult*)
apply *safe*
apply (*cases k* > *0*)
apply (*rule-tac x*=(*cor k*)^2 **in** *exI*)

```
          apply (subst **)
          apply (simp add: power2-eq-square field-simps)
         apply (erule notE, rule-tac x=(cor k)^2 in exI)
         apply (subst **)
         apply (simp add: power2-eq-square field-simps)
        apply (cases k > 0)
        apply (erule notE, rule-tac x=(cor k)^2 in exI)
        apply (subst **)
        apply (simp add: power2-eq-square field-simps)
        apply (rule-tac x=(cor k)^2 in exI)
        apply (subst **)
        apply (simp add: power2-eq-square field-simps)
       apply (cases k > 0)
        apply (rule-tac x=(cor k)^2 in exI)
        apply (subst **)
        apply (simp add: power2-eq-square field-simps)
       apply (erule notE, rule-tac x=(cor k)^2 in exI)
       apply (subst **)
       apply (simp add: power2-eq-square field-simps)
      apply (rule-tac x=(cor k)^2 in exI)
      apply (cases k > 0)
       apply (erule notE, rule-tac x=(cor k)^2 in exI)
       apply (subst **)
       apply (simp add: power2-eq-square field-simps)
      apply (subst **)
      apply (simp add: power2-eq-square field-simps)
      done
  next
    case False
    hence ¬ is-poincare-line-cmat H2
      using k * hermitean-mult-real[of H1 k] hh
      by (auto simp add: power2-eq-square norm-mult)
    have rel-set (≈_v) {(− 1, 1), (1, 1)} {(− 1, 1), (1, 1)}
      by (simp add: rel-set-def)
    thus ?thesis
      using ‹¬ is-poincare-line-cmat H1› ‹¬ is-poincare-line-cmat H2›
      using *
      by (auto simp add: Let-def)
  qed
qed
```

Correctness of the calculation

We show that for every h-line its two calculated ideal points are different and are on the intersection of that line and the unit circle.

Calculated ideal points are on the unit circle

```
lemma calc-ideal-point-1-unit:
  assumes is-real A (cmod B)^2 > (cmod A)^2
  assumes (z1, z2) = calc-ideal-point1-cvec A B
  shows z1 * cnj z1 = z2 * cnj z2
proof−
  let ?discr = Re ((cmod B)^2 − (Re A)^2)
  have ?discr > 0
    using assms
    by (simp add: cmod-power2)
  have (B*(−A − i*sqrt(?discr))) * cnj (B*(−A − i*sqrt(?discr))) = (B * cnj B) * (A^2 + cor (abs ?discr))
    using ‹is-real A› eq-cnj-iff-real[of A]
    by (simp add: field-simps power2-eq-square)
  also have ... = (B * cnj B) * (cmod B)^2
    using ‹?discr > 0›
    using assms
    using complex-of-real-Re[of (cmod B)^2 − (Re A)^2] complex-of-real-Re[of A] ‹is-real A›
    by (simp add: power2-eq-square)
  also have ... = (cmod B)^2 * cnj ((cmod B)^2)
    using complex-cnj-complex-of-real complex-mult-cnj-cmod
```

by *presburger*
finally show *?thesis*
using *assms*
by *simp*
qed

lemma *calc-ideal-point-2-unit*:
assumes *is-real A* $(cmod\ B)^2 > (cmod\ A)^2$
assumes $(z1, z2) = $ *calc-ideal-point2-cvec A B*
shows $z1 * cnj\ z1 = z2 * cnj\ z2$
proof−
let $?discr = Re\ ((cmod\ B)^2 - (Re\ A)^2)$
have $?discr > 0$
using *assms*
by (*simp add*: *cmod-power2*)
have $(B*(-A + i*sqrt(?discr))) * cnj\ (B*(-A + i*sqrt(?discr))) = (B * cnj\ B) * (A^2 + cor\ (abs\ ?discr))$
using ‹*is-real A*› *eq-cnj-iff-real*[*of A*]
by (*simp add*: *field-simps power2-eq-square*)
also have ... $= (B * cnj\ B) * (cmod\ B)^2$
using ‹$?discr > 0$›
using *assms*
using *complex-of-real-Re*[*of* $(cmod\ B)^2 - (Re\ A)^2$] *complex-of-real-Re*[*of A*] ‹*is-real A*›
by (*simp add*: *power2-eq-square*)
also have ... $= (cmod\ B)^2 * cnj\ ((cmod\ B)^2)$
using *complex-cnj-complex-of-real complex-mult-cnj-cmod*
by *presburger*
finally show *?thesis*
using *assms*
by *simp*
qed

lemma *calc-ideal-points-on-unit-circle*:
shows $\forall\ z \in$ *calc-ideal-points H*. $z \in$ *circline-set unit-circle*
unfolding *circline-set-def*
apply *simp*
proof (*transfer*, *transfer*)
fix *H*
assume *hh*: *hermitean H* $\wedge$ $H \neq$ *mat-zero*
obtain *A B C D* **where** ∗: $H = (A, B, C, D)$
by (*cases H*, *auto*)
have $\forall\ (z1, z2) \in$ *calc-ideal-points-cmat-cvec H*. $z1 * cnj\ z1 = z2 * cnj\ z2$
using *hermitean-elems*[*of A B C D*]
unfolding *calc-ideal-points-cmat-cvec-def*
using *calc-ideal-point-1-unit*[*of A B*]
using *calc-ideal-point-2-unit*[*of A B*]
using *hh* ∗
apply (*cases calc-ideal-point1-cvec A B*, *cases calc-ideal-point2-cvec A B*)
apply (*auto simp add*: *Let-def simp del*: *calc-ideal-point1-cvec-def calc-ideal-point2-cvec-def*)
done
thus *Ball* (*calc-ideal-points-cmat-cvec H*) (*on-circline-cmat-cvec unit-circle-cmat*)
using *on-circline-cmat-cvec-unit*
by (*auto simp del*: *on-circline-cmat-cvec-def calc-ideal-points-cmat-cvec-def*)
qed

Calculated ideal points are on the h-line

lemma *calc-ideal-point1-sq*:
assumes $(z1, z2) = $ *calc-ideal-point1-cvec A B is-real A* $(cmod\ B)^2 > (cmod\ A)^2$
shows $z1 * cnj\ z1 + z2 * cnj\ z2 = 2 * (B * cnj\ B)^2$
proof−
let $?discr = Re\ ((cmod\ B)^2 - (Re\ A)^2)$
have $?discr > 0$
using *assms*
by (*simp add*: *cmod-power2*)
have $z1 * cnj\ z1 = (B * cnj\ B) * (-A + i*sqrt(?discr))*(-A - i*sqrt(?discr))$
using *assms eq-cnj-iff-real*[*of A*]
by (*simp*)

also have ... = $(B * cnj\ B) * (A^2 + ?discr)$
  using *complex-of-real-Re*[*of A*] ‹*is-real A*› ‹*?discr* > *0*›
  by (*simp add*: *power2-eq-square field-simps*)
finally
have $z1 * cnj\ z1 = (B * cnj\ B)^2$
  using *complex-of-real-Re*[*of* $(cmod\ B)^2 - (Re\ A)^2$] *complex-of-real-Re*[*of A*] ‹*is-real A*›
  using *complex-mult-cnj-cmod*[*of B*]
  by (*simp add*: *power2-eq-square*)
moreover
have $z2 * cnj\ z2 = (B * cnj\ B)^2$
  using *assms*
  by *simp*
ultimately
show *?thesis*
  by *simp*
qed

lemma *calc-ideal-point2-sq*:
  assumes $(z1, z2) =$ *calc-ideal-point2-cvec A B is-real A* $(cmod\ B)^2 > (cmod\ A)^2$
  shows $z1 * cnj\ z1 + z2 * cnj\ z2 = 2 * (B * cnj\ B)^2$
proof−
  let *?discr* = $Re\ ((cmod\ B)^2 - (Re\ A)^2)$
  have *?discr* > *0*
    using *assms*
    by (*simp add*: *cmod-power2*)
  have $z1 * cnj\ z1 = (B * cnj\ B) * (-A + i*sqrt(?discr))*(-A - i*sqrt(?discr))$
    using *assms eq-cnj-iff-real*[*of A*]
    by *simp*
  also have ... = $(B * cnj\ B) * (A^2 + ?discr)$
    using *complex-of-real-Re*[*of A*] ‹*is-real A*› ‹*?discr* > *0*›
    by (*simp add*: *power2-eq-square field-simps*)
  finally
  have $z1 * cnj\ z1 = (B * cnj\ B)^2$
    using *complex-of-real-Re*[*of* $(cmod\ B)^2 - (Re\ A)^2$] *complex-of-real-Re*[*of A*] ‹*is-real A*›
    using *complex-mult-cnj-cmod*[*of B*]
    by (*simp add*: *power2-eq-square*)
  moreover
  have $z2 * cnj\ z2 = (B * cnj\ B)^2$
    using *assms*
    by *simp*
  ultimately
  show *?thesis*
    by *simp*
qed

lemma *calc-ideal-point1-mix*:
  assumes $(z1, z2) =$ *calc-ideal-point1-cvec A B is-real A* $(cmod\ B)^2 > (cmod\ A)^2$
  shows $B * cnj\ z1 * z2 + cnj\ B * z1 * cnj\ z2 = -\ 2 * A * (B * cnj\ B)^2$
proof−
  have $B*cnj\ z1 + cnj\ B*z1 = -2*A*B*cnj\ B$
    using *assms eq-cnj-iff-real*[*of A*]
    by (*simp*, *simp add*: *field-simps*)
  moreover
  have $cnj\ z2 = z2$
    using *assms*
    by *simp*
  hence $B*cnj\ z1*z2 + cnj\ B*z1*cnj\ z2 = (B*cnj\ z1 + cnj\ B*z1)*z2$
    by (*simp add*: *field-simps*)
  ultimately
  have $B*cnj\ z1*z2 + cnj\ B*z1*cnj\ z2 = -2*A*(B*\ cnj\ B)*z2$
    by *simp*
  also have ... = $-2*A*(B * cnj\ B)^2$
    using *assms*
    using *complex-mult-cnj-cmod*[*of B*]
    by (*simp add*: *power2-eq-square*)
  finally

show *?thesis*
  .
qed

lemma *calc-ideal-point2-mix*:
  assumes $(z1, z2) = calc\text{-}ideal\text{-}point2\text{-}cvec\ A\ B$ *is-real A* $(cmod\ B)^2 > (cmod\ A)^2$
  shows $B * cnj\ z1 * z2 + cnj\ B * z1 * cnj\ z2 = - 2 * A * (B * cnj\ B)^2$
proof−
  have $B{*}cnj\ z1 + cnj\ B{*}z1 = -2{*}A{*}B{*}cnj\ B$
    using *assms eq-cnj-iff-real*[*of A*]
    by (*simp*, *simp add*: *field-simps*)
  moreover
  have $cnj\ z2 = z2$
    using *assms*
    by *simp*
  hence $B{*}cnj\ z1{*}z2 + cnj\ B{*}z1{*}cnj\ z2 = (B{*}cnj\ z1 + cnj\ B{*}z1){*}z2$
    by (*simp add*: *field-simps*)
  ultimately
  have $B{*}cnj\ z1{*}z2 + cnj\ B{*}z1{*}cnj\ z2 = -2{*}A{*}(B{*}\ cnj\ B){*}z2$
    by *simp*
  also have $\ldots = -2{*}A{*}(B * cnj\ B)^2$
    using *assms*
    using *complex-mult-cnj-cmod*[*of B*]
    by (*simp add*: *power2-eq-square*)
  finally
  show *?thesis*
    .
qed

lemma *calc-ideal-point1-on-circline*:
  assumes $(z1, z2) = calc\text{-}ideal\text{-}point1\text{-}cvec\ A\ B$ *is-real A* $(cmod\ B)^2 > (cmod\ A)^2$
  shows $A{*}z1{*}cnj\ z1 + B{*}cnj\ z1{*}z2 + cnj\ B{*}z1{*}cnj\ z2 + A{*}z2{*}cnj\ z2 = 0$ (**is** *?lhs = 0*)
proof−
  have $?lhs = A * (z1 * cnj\ z1 + z2 * cnj\ z2) + (B * cnj\ z1 * z2 + cnj\ B * z1 * cnj\ z2)$
    by (*simp add*: *field-simps*)
  also have $... = 2{*}A{*}(B{*}cnj\ B)^2 + (-2{*}A{*}(B{*}cnj\ B)^2)$
    using *calc-ideal-point1-sq*[*OF assms*]
    using *calc-ideal-point1-mix*[*OF assms*]
    by *simp*
  finally
  show *?thesis*
    by *simp*
qed

lemma *calc-ideal-point2-on-circline*:
  assumes $(z1, z2) = calc\text{-}ideal\text{-}point2\text{-}cvec\ A\ B$ *is-real A* $(cmod\ B)^2 > (cmod\ A)^2$
  shows $A{*}z1{*}cnj\ z1 + B{*}cnj\ z1{*}z2 + cnj\ B{*}z1{*}cnj\ z2 + A{*}z2{*}cnj\ z2 = 0$ (**is** *?lhs = 0*)
proof−
  have $?lhs = A * (z1 * cnj\ z1 + z2 * cnj\ z2) + (B * cnj\ z1 * z2 + cnj\ B * z1 * cnj\ z2)$
    by (*simp add*: *field-simps*)
  also have $... = 2{*}A{*}(B{*}cnj\ B)^2 + (-2{*}A{*}(B{*}cnj\ B)^2)$
    using *calc-ideal-point2-sq*[*OF assms*]
    using *calc-ideal-point2-mix*[*OF assms*]
    by *simp*
  finally
  show *?thesis*
    by *simp*
qed

lemma *calc-ideal-points-on-circline*:
  assumes *is-poincare-line H*
  shows $\forall\ z \in calc\text{-}ideal\text{-}points\ H.\ z \in circline\text{-}set\ H$
  using *assms*
  unfolding *circline-set-def*
  apply *simp*
proof (*transfer*, *transfer*)

fix *H*
assume *hh*: *hermitean H* ∧ *H* ≠ *mat-zero*
obtain *A B C D* **where** ∗: *H* = (*A*, *B*, *C*, *D*)
by (*cases H*, *auto*)
obtain *z11 z12 z21 z22* **where** ∗∗: (*z11*, *z12*) = *calc-ideal-point1-cvec A B* (*z21*, *z22*) = *calc-ideal-point2-cvec A B*
by (*cases calc-ideal-point1-cvec A B*, *cases calc-ideal-point2-cvec A B*) *auto*

assume *is-poincare-line-cmat H*
hence ∀ (*z1*, *z2*) ∈ *calc-ideal-points-cmat-cvec H*. *A*∗*z1*∗*cnj z1* + *B*∗*cnj z1*∗*z2* + *C*∗*z1*∗*cnj z2* + *D*∗*z2*∗*cnj z2* = *0*
using ∗ ∗∗ *hh*
using *hermitean-elems*[*of A B C D*]
using *calc-ideal-point1-on-circline*[*of z11 z12 A B*]
using *calc-ideal-point2-on-circline*[*of z21 z22 A B*]
by (*auto simp del*: *calc-ideal-point1-cvec-def calc-ideal-point2-cvec-def*)
thus *Ball* (*calc-ideal-points-cmat-cvec H*) (*on-circline-cmat-cvec H*)
using *on-circline-cmat-cvec-circline-equation* ∗
by (*auto simp del*: *on-circline-cmat-cvec-def calc-ideal-points-cmat-cvec-def simp add*: *field-simps*)
qed

Calculated ideal points of an h-line are different

lemma *calc-ideal-points-cvec-different* [*simp*]:
assumes $(cmod\ B)^2 > (cmod\ A)^2$ *is-real A*
shows ¬ (*calc-ideal-point1-cvec A B* $\approx_v$ *calc-ideal-point2-cvec A B*)
using *assms*
by (*auto*) (*auto simp add*: *cmod-def*)

lemma *calc-ideal-points-different*:
assumes *is-poincare-line H*
shows ∃ *i1* ∈ (*calc-ideal-points H*). ∃ *i2* ∈ (*calc-ideal-points H*). *i1* ≠ *i2*
using *assms*
proof (*transfer*, *transfer*)
fix *H*
assume *hh*: *hermitean H* ∧ *H* ≠ *mat-zero is-poincare-line-cmat H*
obtain *A B C D* **where** ∗: *H* = (*A*, *B*, *C*, *D*)
by (*cases H*, *auto*)
hence *is-real A* **using** *hh hermitean-elems* **by** *auto*
thus ∃*i1*∈*calc-ideal-points-cmat-cvec H*. ∃*i2*∈*calc-ideal-points-cmat-cvec H*. ¬ *i1* $\approx_v$ *i2*
using ∗ *hh calc-ideal-points-cvec-different*[*of A B*]
apply (*rule-tac x*=*calc-ideal-point1-cvec A B* **in** *bexI*)
apply (*rule-tac x*=*calc-ideal-point2-cvec A B* **in** *bexI*)
by *auto*
qed

lemma *two-calc-ideal-points* [*simp*]:
assumes *is-poincare-line H*
shows *card* (*calc-ideal-points H*) = *2*
proof−
have ∃ *x* ∈ *calc-ideal-points H*. ∃ *y* ∈ *calc-ideal-points H*. ∀ *z* ∈ *calc-ideal-points H*. *z* = *x* ∨ *z* = *y*
by (*transfer*, *transfer*, *case-tac H*, *simp del*: *calc-ideal-point1-cvec-def calc-ideal-point2-cvec-def*)
then obtain *x y* **where** ∗: *calc-ideal-points H* = {*x*, *y*}
by *auto*
moreover
have *x* ≠ *y*
using *calc-ideal-points-different*[*OF assms*] ∗
by *auto*
ultimately
show *?thesis*
by *auto*
qed

4.3.2 Ideal points

Next we give a genuine definition of ideal points – these are the intersections of the h-line with the unit circle

definition *ideal-points* :: *circline* ⇒ *complex-homo set* **where**
ideal-points H = *circline-intersection H unit-circle*

Ideal points are on the unit circle and on the h-line

lemma *ideal-points-on-unit-circle*:
  shows ∀ *z* ∈ *ideal-points H*. *z* ∈ *circline-set unit-circle*
  unfolding *ideal-points-def circline-intersection-def circline-set-def*
  by *simp*

lemma *ideal-points-on-circline*:
  shows ∀ *z* ∈ *ideal-points H*. *z* ∈ *circline-set H*
  unfolding *ideal-points-def circline-intersection-def circline-set-def*
  by *simp*

For each h-line there are exactly two ideal points

lemma *two-ideal-points*:
  assumes *is-poincare-line H*
  shows *card* (*ideal-points H*) = *2*
proof−
  have *H* ≠ *unit-circle*
    using *assms not-is-poincare-line-unit-circle*
    by *auto*
  let *?int* = *circline-intersection H unit-circle*
  obtain *i1 i2* **where** *i1* ∈ *?int i2* ∈ *?int i1* ≠ *i2*
    using *calc-ideal-points-on-circline*[*OF assms*]
    using *calc-ideal-points-on-unit-circle*[*of H*]
    using *calc-ideal-points-different*[*OF assms*]
    unfolding *circline-intersection-def circline-set-def*
    by *auto*
  thus *?thesis*
    unfolding *ideal-points-def*
    using *circline-intersection-at-most-2-points*[*OF* ‹*H* ≠ *unit-circle*›]
    using *card-geq-2-iff-contains-2-elems*[*of ?int*]
    by *auto*
qed

They are exactly the two points that our calculation finds

lemma *ideal-points-unique*:
  assumes *is-poincare-line H*
  shows *ideal-points H* = *calc-ideal-points H*
proof−
  have *calc-ideal-points H* ⊆ *ideal-points H*
    using *calc-ideal-points-on-circline*[*OF assms*]
    using *calc-ideal-points-on-unit-circle*[*of H*]
    unfolding *ideal-points-def circline-intersection-def circline-set-def*
    by *auto*
  moreover
  have *H* ≠ *unit-circle*
    using *not-is-poincare-line-unit-circle assms*
    by *auto*
  hence *finite* (*ideal-points H*)
    using *circline-intersection-at-most-2-points*[*of H unit-circle*]
    unfolding *ideal-points-def*
    by *auto*
  ultimately
  show *?thesis*
    using *card-subset-eq*[*of ideal-points H calc-ideal-points H*]
    using *two-calc-ideal-points*[*OF assms*]
    using *two-ideal-points*[*OF assms*]
    by *auto*
qed

For each h-line we can obtain two different ideal points

lemma *obtain-ideal-points*:
  assumes *is-poincare-line H*
  obtains *i1 i2* **where** *i1* ≠ *i2 ideal-points H* = {*i1*, *i2*}
  using *two-ideal-points*[*OF assms*] *card-eq-2-iff-doubleton*[*of ideal-points H*]
  by *blast*

Ideal points of each h-line constructed from two points in the disc are different than those two points

lemma *ideal-points-different*:
  assumes $u \in$ *unit-disc* $v \in$ *unit-disc* $u \neq v$
  assumes *ideal-points* (*poincare-line* u v) $= \{i1, i2\}$
  shows $i1 \neq i2$ $u \neq i1$ $u \neq i2$ $v \neq i1$ $v \neq i2$
proof−
  have $i1 \in$ *ocircline-set ounit-circle* $i2 \in$ *ocircline-set ounit-circle*
    using *assms*(*3*) *assms*(*4*) *ideal-points-on-unit-circle is-poincare-line-poincare-line*
    by *fastforce*+
  thus $u \neq i1$ $u \neq i2$ $v \neq i1$ $v \neq i2$
    using *assms*(*1*−*2*)
    using *disc-inter-ocircline-set*[*of ounit-circle*]
    unfolding *unit-disc-def*
    by *auto*
  show $i1 \neq i2$
    using *assms*
    by (*metis doubleton-eq-iff is-poincare-line-poincare-line obtain-ideal-points*)
qed

H-line is uniquely determined by its ideal points

lemma *ideal-points-line-unique*:
  assumes *is-poincare-line* H *ideal-points* $H = \{i1, i2\}$
  shows $H =$ *poincare-line* $i1$ $i2$
  by (*smt assms*(*1*) *assms*(*2*) *calc-ideal-points-on-unit-circle circline-set-def ex-poincare-line-points ideal-points-different*(*1*) *ideal-points-on-circline ideal-points-unique insertI1 insert-commute inversion-unit-circle mem-Collect-eq unique-poincare-line-general*)

Ideal points of some special h-lines

Ideal points of *x-axis*

lemma *ideal-points-x-axis*
  [*simp*]: *ideal-points x-axis* $= \{$*of-complex* (-1), *of-complex* $1\}$
proof (*subst ideal-points-unique*, *simp*)
  have *calc-ideal-points-clmat-hcoords x-axis-clmat* $= \{$*of-complex-hcoords* (-1), *of-complex-hcoords* $1\}$
    by *transfer auto*
  thus *calc-ideal-points x-axis* $= \{$*of-complex* (-1), *of-complex* $1\}$
    by (*simp add*: *calc-ideal-points.abs-eq of-complex.abs-eq x-axis-def*)
qed

Ideal points are proportional vectors only if h-line is a line segment passing trough zero

lemma *ideal-points-proportional*:
  assumes *is-poincare-line* H *ideal-points* $H = \{i1, i2\}$ *to-complex* $i1 =$ *cor* k $*$ *to-complex* $i2$
  shows $0_h \in$ *circline-set* H
proof−
  have $i1 \neq i2$
    using ‹*ideal-points* $H = \{i1, i2\}$›
    using ‹*is-poincare-line* H› *ex-poincare-line-points ideal-points-different*(*1*) **by** *blast*

  have $i1 \in$ *circline-set unit-circle* $i2 \in$ *circline-set unit-circle*
    using *assms calc-ideal-points-on-unit-circle ideal-points-unique*
    by *blast*+

  hence *cmod* (*cor* k) $= 1$
    using ‹*to-complex* $i1 =$ *cor* k $*$ *to-complex* $i2$›
   by (*metis* (*mono-tags*, *lifting*) *circline-set-unit-circle imageE mem-Collect-eq mult.right-neutral norm-mult to-complex-of-complex unit-circle-set-def*)
  hence $k = -1$
    using ‹*to-complex* $i1 =$ *cor* k $*$ *to-complex* $i2$› ‹$i1 \neq i2$›
    using ‹$i1 \in$ *circline-set unit-circle*› ‹$i2 \in$ *circline-set unit-circle*›
    by (*smt* (*verit*, *best*) *mult-cancel-right1 norm-of-real not-inf-on-unit-circle''* *of-complex-to-complex of-real-1*)

   have $\forall$ $i1 \in$ *calc-ideal-points* H. $\forall$ $i2 \in$ *calc-ideal-points* H. *is-poincare-line* H $\wedge$ $i1 \neq i2$ $\wedge$ *to-complex* $i1 = -$ *to-complex* $i2$ $\longrightarrow$
          $0_h \in$ *circline-set* H
    unfolding *circline-set-def*
  proof (*simp*, *transfer*, *transfer*, *safe*)

```
    fix A B C D i11 i12 i21 i22 k
    assume H:hermitean (A, B, C, D) (A, B, C, D) ≠ mat-zero
    assume line: is-poincare-line-cmat (A, B, C, D)
    assume i1: (i11, i12) ∈ calc-ideal-points-cmat-cvec (A, B, C, D)
    assume i2:(i21, i22) ∈ calc-ideal-points-cmat-cvec (A, B, C, D)
    assume ¬ (i11, i12) ≈v (i21, i22)
    assume opposite: to-complex-cvec (i11, i12) = − to-complex-cvec (i21, i22)


    let ?discr = sqrt ((cmod B)^2 − (Re D)^2)
    let ?den = (cmod B)^2
    let ?i1 = B ∗ (− D − i ∗ ?discr)
    let ?i2 = B ∗ (− D + i ∗ ?discr)

    have i11 = ?i1 ∨ i11 = ?i2 i12 = ?den
      i21 = ?i1 ∨ i21 = ?i2 i22 = ?den
      using i1 i2 H line
      by (auto split: if-split-asm)
    hence i: i11 = ?i1 ∧ i21 = ?i2 ∨ i11 = ?i2 ∧ i21 = ?i1
      using ‹¬ (i11, i12) ≈v (i21, i22)›
      by auto

    have ?den ≠ 0
      using line
      by auto

    hence i11 = − i21
      using opposite ‹i12 = ?den› ‹i22 = ?den›
      by (simp add: nonzero-neg-divide-eq-eq2)

    hence ?i1 = − ?i2
      using i
      by (metis add.inverse-inverse)

    hence D = 0
      using ‹?den ≠ 0›
      by (simp add: field-simps)

    thus on-circline-cmat-cvec (A, B, C, D) 0v
      by (simp add: vec-cnj-def)
  qed

  thus ?thesis
    using assms ‹k = −1›
    using calc-ideal-points-different ideal-points-unique
    by fastforce
qed
```

Transformations of ideal points

Möbius transformations that fix the unit disc when acting on h-lines map their ideal points to ideal points.

```
lemma ideal-points-moebius-circline [simp]:
  assumes unit-circle-fix M is-poincare-line H
  shows ideal-points (moebius-circline M H) = (moebius-pt M) ‘ (ideal-points H) (is ?I' = ?M ‘ ?I)
proof−
  obtain i1 i2 where ∗: i1 ≠ i2 ?I = {i1, i2}
    using assms(2)
    by (rule obtain-ideal-points)
  let ?Mi1 = ?M i1 and ?Mi2 = ?M i2
  have ?Mi1 ∈ ?M ‘ (circline-set H)
       ?Mi2 ∈ ?M ‘ (circline-set H)
       ?Mi1 ∈ ?M ‘ (circline-set unit-circle)
       ?Mi2 ∈ ?M ‘ (circline-set unit-circle)
    using ∗
    unfolding ideal-points-def circline-intersection-def circline-set-def
    by blast+
```

hence *?Mi1* ∈ *?I'*
?Mi2 ∈ *?I'*
using *unit-circle-fix-iff*[*of M*] *assms*
unfolding *ideal-points-def circline-intersection-def circline-set-def*
by (*metis mem-Collect-eq moebius-circline*)+
moreover
have *?Mi1* ≠ *?Mi2*
using *bij-moebius-pt*[*of M*] ∗
using *moebius-pt-invert* **by** *blast*
moreover
have *is-poincare-line* (*moebius-circline M H*)
using *assms unit-circle-fix-preserve-is-poincare-line*
by *simp*
ultimately
have *?I'* = {*?Mi1*, *?Mi2*}
using *two-ideal-points*[*of moebius-circline M H*]
using *card-eq-2-doubleton*[*of ?I' ?Mi1 ?Mi2*]
by *simp*
thus *?thesis*
using ∗(*2*)
by *auto*
qed

lemma *ideal-points-poincare-line-moebius* [*simp*]:
assumes *unit-disc-fix M* *u* ∈ *unit-disc v* ∈ *unit-disc u* ≠ *v*
assumes *ideal-points* (*poincare-line u v*) = {*i1*, *i2*}
shows *ideal-points* (*poincare-line* (*moebius-pt M u*) (*moebius-pt M v*)) = {*moebius-pt M i1*, *moebius-pt M i2*}
using *assms*
by *auto*

Conjugation also maps ideal points to ideal points

lemma *ideal-points-conjugate* [*simp*]:
assumes *is-poincare-line H*
shows *ideal-points* (*conjugate-circline H*) = *conjugate* ‘ (*ideal-points H*) (**is** *?I'* = *?M* ‘ *?I*)
proof−
obtain *i1 i2* **where** ∗: *i1* ≠ *i2* *?I* = {*i1*, *i2*}
using *assms*
by (*rule obtain-ideal-points*)
let *?Mi1* = *?M i1* **and** *?Mi2* = *?M i2*
have *?Mi1* ∈ *?M* ‘ (*circline-set H*)
?Mi2 ∈ *?M* ‘ (*circline-set H*)
?Mi1 ∈ *?M* ‘ (*circline-set unit-circle*)
?Mi2 ∈ *?M* ‘ (*circline-set unit-circle*)
using ∗
unfolding *ideal-points-def circline-intersection-def circline-set-def*
by *blast*+
hence *?Mi1* ∈ *?I'*
?Mi2 ∈ *?I'*
unfolding *ideal-points-def circline-intersection-def circline-set-def*
using *circline-set-conjugate-circline circline-set-def conjugate-unit-circle-set*
by *blast*+
moreover
have *?Mi1* ≠ *?Mi2*
using ‹*i1* ≠ *i2*›
by (*auto simp add*: *conjugate-inj*)
moreover
have *is-poincare-line* (*conjugate-circline H*)
using *assms*
by *simp*
ultimately
have *?I'* = {*?Mi1*, *?Mi2*}
using *two-ideal-points*[*of conjugate-circline H*]
using *card-eq-2-doubleton*[*of ?I' ?Mi1 ?Mi2*]
by *simp*
thus *?thesis*
using ∗(*2*)

by *auto*
qed

lemma *ideal-points-poincare-line-conjugate* [*simp*]:
assumes $u \in$ *unit-disc* $v \in$ *unit-disc* $u \neq v$
assumes *ideal-points* (*poincare-line* *u v*) = {*i1*, *i2*}
shows *ideal-points* (*poincare-line* (*conjugate u*) (*conjugate v*)) = {*conjugate i1*, *conjugate i2*}
using *assms*
by *auto*

end
theory *Poincare-Distance*
imports *Poincare-Lines-Ideal-Points Hyperbolic-Functions*
begin

5 H-distance in the Poincaré model

Informally, the *h-distance* between the two h-points is defined as the absolute value of the logarithm of the cross ratio between those two points and the two ideal points.

abbreviation *Re-cross-ratio* **where** *Re-cross-ratio z u v w* ≡ *Re* (*to-complex* (*cross-ratio z u v w*))

definition *calc-poincare-distance* :: *complex-homo* ⇒ *complex-homo* ⇒ *complex-homo* ⇒ *complex-homo* ⇒ *real* **where**
[*simp*]: *calc-poincare-distance u i1 v i2* = *abs* (*ln* (*Re-cross-ratio u i1 v i2*))

definition *poincare-distance-pred* :: *complex-homo* ⇒ *complex-homo* ⇒ *real* ⇒ *bool* **where**
[*simp*]: *poincare-distance-pred u v d* ⟷
$(u = v \wedge d = 0) \vee (u \neq v \wedge (\forall$ *i1 i2*. *ideal-points* (*poincare-line u v*) = {*i1*, *i2*} ⟶ *d* = *calc-poincare-distance u i1 v i2*))

definition *poincare-distance* :: *complex-homo* ⇒ *complex-homo* ⇒ *real* **where**
poincare-distance u v = (*THE d*. *poincare-distance-pred u v d*)

We shown that the described cross-ratio is always finite, positive real number.

lemma *distance-cross-ratio-real-positive*:
assumes $u \in$ *unit-disc* **and** $v \in$ *unit-disc* **and** $u \neq v$
shows ∀ *i1 i2*. *ideal-points* (*poincare-line u v*) = {*i1*, *i2*} ⟶
cross-ratio u i1 v i2 $\neq \infty_h$ ∧ *is-real* (*to-complex* (*cross-ratio u i1 v i2*)) ∧ *Re-cross-ratio u i1 v i2* > *0*
(**is** *?P u v*)
proof (*rule wlog-positive-x-axis*[*OF assms*])
fix *x*
assume ∗: *is-real x* *0* < *Re x* *Re x* < *1*
hence $x \neq -1$ $x \neq 1$
by *auto*
hence ∗∗: *of-complex x* $\neq \infty_h$ *of-complex x* $\neq 0_h$ *of-complex x* ≠ *of-complex* (−*1*) *of-complex 1* ≠ *of-complex x*
of-complex x ∈ *circline-set x-axis*
using ∗
unfolding *circline-set-x-axis*
by (*auto simp add*: *of-complex-inj*)

have ∗∗∗: 0_h ≠ *of-complex* (−*1*) 0_h ≠ *of-complex 1*
by (*metis of-complex-zero-iff zero-neq-neg-one*, *simp*)

have ∗∗∗∗: $-\ x - 1 \neq 0$ $x - 1 \neq 0$
using ‹$x \neq -1$› ‹$x \neq 1$›
by (*metis add.inverse-inverse eq-iff-diff-eq-0*, *simp*)

have *poincare-line* 0_h (*of-complex x*) = *x-axis*
using ∗∗
by (*simp add*: *poincare-line-0-real-is-x-axis*)
thus *?P* 0_h (*of-complex x*)
using ∗ ∗∗ ∗∗∗ ∗∗∗∗
using *cross-ratio-not-inf*[*of* 0_h *of-complex 1 of-complex* (−*1*) *of-complex x*]
using *cross-ratio-not-inf*[*of* 0_h *of-complex* (−*1*) *of-complex 1 of-complex x*]
using *cross-ratio-real*[*of 0* −*1 x 1*] *cross-ratio-real*[*of 0 1 x* −*1*]

```
    apply (auto simp add: poincare-line-0-real-is-x-axis doubleton-eq-iff circline-set-x-axis)
    apply (subst cross-ratio, simp-all, subst Re-complex-div-gt-0, simp, subst mult-neg-neg, simp-all)+
    done
next
  fix M u v
  let ?Mu = moebius-pt M u and ?Mv = moebius-pt M v
  assume ∗: unit-disc-fix M u ∈ unit-disc v ∈ unit-disc u ≠ v
           ?P ?Mu ?Mv
  show ?P u v
  proof safe
    fix i1 i2
    let ?cr = cross-ratio u i1 v i2
    assume ∗∗: ideal-points (poincare-line u v) = {i1, i2}
    have i1 ≠ u i1 ≠ v i2 ≠ u i2 ≠ v i1 ≠ i2
      using ideal-points-different[OF ∗(2−3), of i1 i2] ∗∗ ‹u ≠ v›
      by auto
    hence 0 < Re (to-complex ?cr) ∧ is-real (to-complex ?cr) ∧ ?cr ≠ ∞h
      using ∗ ∗∗
      apply (erule-tac x=moebius-pt M i1 in allE)
      apply (erule-tac x=moebius-pt M i2 in allE)
      apply (subst (asm) ideal-points-poincare-line-moebius[of M u v i1 i2], simp-all)
      done
    thus 0 < Re (to-complex ?cr) is-real (to-complex ?cr) ?cr = ∞h =⇒ False
      by simp-all
  qed
qed
```

Next we can show that for every different points from the unit disc there is exactly one number that satisfies the h-distance predicate.

```
lemma distance-unique:
  assumes u ∈ unit-disc and v ∈ unit-disc
  shows ∃! d. poincare-distance-pred u v d
proof (cases u = v)
  case True
  thus ?thesis
    by auto
next
  case False
  obtain i1 i2 where ∗: i1 ≠ i2 ideal-points (poincare-line u v) = {i1, i2}
    using obtain-ideal-points[OF is-poincare-line-poincare-line] ‹u ≠ v›
    by blast
  let ?d = calc-poincare-distance u i1 v i2
  show ?thesis
  proof (rule ex1I)
    show poincare-distance-pred u v ?d
      using ∗ ‹u ≠ v›
    proof (simp del: calc-poincare-distance-def, safe)
      fix i1' i2'
      assume {i1, i2} = {i1', i2'}
      hence ∗∗: (i1' = i1 ∧ i2' = i2) ∨ (i1' = i2 ∧ i2' = i1)
        using doubleton-eq-iff[of i1 i2 i1' i2']
        by blast
      have all-different: u ≠ i1 u ≠ i2 v ≠ i1 v ≠ i2 u ≠ i1' u ≠ i2' v ≠ i1' v ≠ i2' i1 ≠ i2
        using ideal-points-different[OF assms, of i1 i2] ∗ ∗∗ ‹u ≠ v›
        by auto

      show calc-poincare-distance u i1 v i2 = calc-poincare-distance u i1' v i2'
      proof−
        let ?cr = cross-ratio u i1 v i2
        let ?cr' = cross-ratio u i1' v i2'

        have Re (to-complex ?cr) > 0 is-real (to-complex ?cr)
             Re (to-complex ?cr') > 0 is-real (to-complex ?cr')
          using False distance-cross-ratio-real-positive[OF assms(1−2)] ∗ ∗∗
          by auto
```

thus *?thesis*
using ∗∗
using *cross-ratio-not-zero cross-ratio-not-inf all-different*
by *auto* (*subst cross-ratio-commute-24*, *subst reciprocal-real*, *simp-all add*: *ln-div*)
qed
qed
next
fix *d*
assume *poincare-distance-pred u v d*
thus *d* = *?d*
using ∗ ‹*u* ≠ *v*›
by *auto*
qed
qed

lemma *poincare-distance-satisfies-pred* [*simp*]:
assumes *u* ∈ *unit-disc* **and** *v* ∈ *unit-disc*
shows *poincare-distance-pred u v* (*poincare-distance u v*)
using *distance-unique*[*OF assms*] *theI'*[*of poincare-distance-pred u v*]
unfolding *poincare-distance-def*
by *blast*

lemma *poincare-distance-I*:
assumes *u* ∈ *unit-disc* **and** *v* ∈ *unit-disc* **and** *u* ≠ *v* **and** *ideal-points* (*poincare-line u v*) = {*i1*, *i2*}
shows *poincare-distance u v* = *calc-poincare-distance u i1 v i2*
using *assms*
using *poincare-distance-satisfies-pred*[*OF assms*(*1*−*2*)]
by *simp*

lemma *poincare-distance-refl* [*simp*]:
assumes *u* ∈ *unit-disc*
shows *poincare-distance u u* = *0*
using *assms*
using *poincare-distance-satisfies-pred*[*OF assms assms*]
by *simp*

Unit disc preserving Möbius transformations preserve h-distance.

lemma *unit-disc-fix-preserve-poincare-distance* [*simp*]:
assumes *unit-disc-fix M* **and** *u* ∈ *unit-disc* **and** *v* ∈ *unit-disc*
shows *poincare-distance* (*moebius-pt M u*) (*moebius-pt M v*) = *poincare-distance u v*
proof (*cases u* = *v*)
case *True*
have *moebius-pt M u* ∈ *unit-disc moebius-pt M v* ∈ *unit-disc*
using *unit-disc-fix-iff*[*OF assms*(*1*), *symmetric*] *assms*
by *blast*+
thus *?thesis*
using *assms* ‹*u* = *v*›
by *simp*
next
case *False*
obtain *i1 i2* **where** ∗: *ideal-points* (*poincare-line u v*) = {*i1*, *i2*}
using ‹*u* ≠ *v*›
by (*rule obtain-ideal-points*[*OF is-poincare-line-poincare-line*[*of u v*]])
let *?Mu* = *moebius-pt M u* **and** *?Mv* = *moebius-pt M v* **and** *?Mi1* = *moebius-pt M i1* **and** *?Mi2* = *moebius-pt M i2*

have ∗∗: *?Mu* ∈ *unit-disc ?Mv* ∈ *unit-disc*
using *assms*
using *unit-disc-fix-iff*
by *blast*+

have ∗∗∗: *?Mu* ≠ *?Mv*
using ‹*u* ≠ *v*›
by *simp*

have *poincare-distance u v* = *calc-poincare-distance u i1 v i2*
using *poincare-distance-I*[*OF assms*(*2*−*3*) ‹*u* ≠ *v*› ∗]

```
    by auto
  moreover
  have unit-circle-fix M
    using assms
    by simp
  hence ++: ideal-points (poincare-line ?Mu ?Mv) = {?Mi1, ?Mi2}
    using ‹u ≠ v› assms ∗
    by simp
  have poincare-distance ?Mu ?Mv = calc-poincare-distance ?Mu ?Mi1 ?Mv ?Mi2
    by (rule poincare-distance-I[OF ∗∗ ∗∗∗ ++])
  moreover
  have calc-poincare-distance ?Mu ?Mi1 ?Mv ?Mi2 = calc-poincare-distance u i1 v i2
    using ideal-points-different[OF assms(2−3) ‹u ≠ v› ∗]
    unfolding calc-poincare-distance-def
    by (subst moebius-preserve-cross-ratio[symmetric], simp-all)
  ultimately
  show ?thesis
    by simp
qed
```

Knowing ideal points for x-axis, we can easily explicitly calculate distances.

```
lemma poincare-distance-x-axis-x-axis:
  assumes x ∈ unit-disc and y ∈ unit-disc and x ∈ circline-set x-axis and y ∈ circline-set x-axis
  shows poincare-distance x y =
          (let x′ = to-complex x; y′ = to-complex y
            in abs (ln (Re (((1 + x′) ∗ (1 − y′)) / ((1 − x′) ∗ (1 + y′))))))
proof−
  obtain x′ y′ where ∗: x = of-complex x′ y = of-complex y′
    using inf-or-of-complex[of x] inf-or-of-complex[of y] ‹x ∈ unit-disc› ‹y ∈ unit-disc›
    by auto

  have cmod x′ < 1 cmod y′ < 1
    using ‹x ∈ unit-disc› ‹y ∈ unit-disc› ∗
    by (metis unit-disc-iff-cmod-lt-1)+
  hence ∗∗: x′ ≠ 1 x′ ≠ 1 y′ ≠ −1 y′ ≠ 1
    by auto

  have 1 + y′ ≠ 0
    using ∗∗
    by (metis add.left-cancel add-neg-numeral-special(7))

  show ?thesis
  proof (cases x = y)
    case True
    thus ?thesis
      using assms(1−2)
      using unit-disc-iff-cmod-lt-1[of to-complex x] ∗ ∗∗ ‹1 + y′ ≠ 0›
      by auto

  next
    case False
    hence poincare-line x y = x-axis
      using poincare-line-x-axis[OF assms]
      by simp
    hence ideal-points (poincare-line x y) = {of-complex (−1), of-complex 1}
      by simp
    hence poincare-distance x y = calc-poincare-distance x (of-complex (−1)) y (of-complex 1)
      using poincare-distance-I assms ‹x ≠ y›
      by auto
    also have ... = abs (ln (Re (((x′ + 1) ∗ (y′ − 1)) / ((x′ − 1) ∗ (y′ + 1)))))
      using ∗ ‹cmod x′ < 1› ‹cmod y′ < 1›
      by (simp, transfer, transfer, auto)
    finally
    show ?thesis
      using ∗
        by (metis (no-types, lifting) add.commute minus-diff-eq minus-divide-divide mult-minus-left mult-minus-right
```

to-complex-of-complex)
  qed
qed

lemma *poincare-distance-zero-x-axis*:
  assumes $x \in$ *unit-disc* **and** $x \in$ *circline-set x-axis*
  shows *poincare-distance* 0_h x = (*let* x' = *to-complex* x *in abs* (*ln* (*Re* ((*1* − x') / (*1* + x')))))
  using *assms*
  using *poincare-distance-x-axis-x-axis*[*of* 0_h x]
  by (*simp add*: *Let-def*)

lemma *poincare-distance-zero*:
  assumes $x \in$ *unit-disc*
  shows *poincare-distance* 0_h x = (*let* x' = *to-complex* x *in abs* (*ln* (*Re* ((*1* − *cmod* x') / (*1* + *cmod* x'))))) (**is** *?P x*)
proof (*cases* $x = 0_h$)
  case *True*
  thus *?thesis*
    by *auto*
next
  case *False*
  show *?thesis*
  proof (*rule wlog-rotation-to-positive-x-axis*)
    show $x \in$ *unit-disc* $x \neq 0_h$ **by** *fact+*
  next
    fix φ u
    assume $u \in$ *unit-disc* $u \neq 0_h$ *?P* (*moebius-pt* (*moebius-rotation* φ) u)
    thus *?P u*
      using *unit-disc-fix-preserve-poincare-distance*[*of moebius-rotation* φ 0_h u]
      by (*cases* $u = \infty_h$) (*simp-all add*: *Let-def*)
  next
    fix x
    assume *is-real x* $0 <$ *Re x Re x* < 1
    thus *?P* (*of-complex x*)
      using *poincare-distance-zero-x-axis*[*of of-complex x*]
      by *simp* (*auto simp add*: *circline-set-x-axis cmod-eq-Re complex-is-Real-iff*)
  qed
qed

lemma *poincare-distance-zero-opposite* [*simp*]:
  assumes *of-complex z* $\in$ *unit-disc*
  shows *poincare-distance* 0_h (*of-complex* (− z)) = *poincare-distance* 0_h (*of-complex z*)
proof−
  have $*$: *of-complex* ($-z$) $\in$ *unit-disc*
    using *assms*
    by *auto*
  show *?thesis*
    using *poincare-distance-zero*[*OF assms*]
    using *poincare-distance-zero*[*OF* $*$]
    by *simp*
qed

5.1 Distance explicit formula

Instead of the h-distance itself, very frequently its hyperbolic cosine is analyzed.

abbreviation *cosh-dist u v* $\equiv$ *cosh* (*poincare-distance u v*)

lemma *cosh-poincare-distance-cross-ratio-average*:
  assumes $u \in$ *unit-disc* $v \in$ *unit-disc* $u \neq v$ *ideal-points* (*poincare-line u v*) = {*i1*, *i2*}
  shows *cosh-dist u v* =
        ((*Re-cross-ratio u i1 v i2*) + (*Re-cross-ratio v i1 u i2*)) / *2*
proof−
  let *?cr* = *cross-ratio u i1 v i2*
  let *?crRe* = *Re* (*to-complex ?cr*)
  have *?cr* $\neq \infty_h$ *is-real* (*to-complex ?cr*) *?crRe* > 0
    using *distance-cross-ratio-real-positive*[*OF assms*(*1*−*3*)] *assms*(*4*)
    by *simp-all*

  then obtain *cr* **where** $*$: *cross-ratio u i1 v i2* $=$ *of-complex cr cr* $\neq$ *0 is-real cr Re cr* $>$ *0*
    using *inf-or-of-complex*[*of cross-ratio u i1 v i2*]
    by (*smt to-complex-of-complex zero-complex.simps*(*1*))
  thus *?thesis*
    using $*$
    using *assms cross-ratio-commute-13*[*of v i1 u i2*]
    unfolding *poincare-distance-I*[*OF assms*] *calc-poincare-distance-def cosh-def*
    by (*cases Re cr* $\geq$ *1*)
       (*auto simp add*: *ln-div*[*of 0*] *exp-minus field-simps Re-divide power2-eq-square complex.expand*)
qed

definition *poincare-distance-formula'* :: *complex* $\Rightarrow$ *complex* $\Rightarrow$ *real* **where**
[*simp*]: *poincare-distance-formula' u v* $= 1 + 2 * ((\mathit{cmod}\ (u - v))^2\ /\ ((1 - (\mathit{cmod}\ u)^2) * (1 - (\mathit{cmod}\ v)^2)))$

Next we show that the following formula expresses h-distance between any two h-points (note that the ideal points do not figure anymore).

definition *poincare-distance-formula* :: *complex* $\Rightarrow$ *complex* $\Rightarrow$ *real* **where**
  [*simp*]: *poincare-distance-formula u v* $=$ *arcosh* (*poincare-distance-formula' u v*)

lemma *blaschke-preserve-distance-formula* [*simp*]:
  assumes *of-complex k* $\in$ *unit-disc u* $\in$ *unit-disc v* $\in$ *unit-disc*
  shows *poincare-distance-formula* (*to-complex* (*moebius-pt* (*blaschke k*) *u*)) (*to-complex* (*moebius-pt* (*blaschke k*) *v*)) $=$
         poincare-distance-formula (*to-complex u*) (*to-complex v*)
proof (*cases k* $=$ *0*)
  case *True*
  thus *?thesis*
    by *simp*
next
  case *False*
  obtain *u' v'* **where** $*$: *u'* $=$ *to-complex u v'* $=$ *to-complex v*
    by *auto*

  have *cmod u'* $<$ *1 cmod v'* $<$ *1 cmod k* $<$ *1*
    using *assms* $*$
    using *inf-or-of-complex*[*of u*] *inf-or-of-complex*[*of v*]
    by *auto*

  obtain *nu du nv dv d kk ddu ddv* **where**
    $**$: $nu = u' - k$ $du = 1 - \mathit{cnj}\ k *u'$ $nv = v' - k$ $dv = 1 - \mathit{cnj}\ k * v'$
        $d = u' - v'$ $ddu = 1 - u' * \mathit{cnj}\ u'$ $ddv = 1 - v' * \mathit{cnj}\ v'$ $kk = 1 - k * \mathit{cnj}\ k$
    by *auto*

  have *d*: $nu * dv - nv * du = d * kk$
    by (*subst* $**$)+ (*simp add*: *field-simps*)
  have *ddu*: $du * \mathit{cnj}\ du - nu * \mathit{cnj}\ nu = ddu * kk$
    by (*subst* $**$)+ (*simp add*: *field-simps*)
  have *ddv*: $dv * \mathit{cnj}\ dv - nv * \mathit{cnj}\ nv = ddv * kk$
    by (*subst* $**$)+ (*simp add*: *field-simps*)

  have *du* $\neq$ *0*
  proof (*rule ccontr*)
    assume $\neg$ *?thesis*
    hence *cmod* $(1 - \mathit{cnj}\ k * u') = 0$
      using ‹$du = 1 - \mathit{cnj}\ k * u'$›
      by *auto*
    hence *cmod* $(\mathit{cnj}\ k * u') = 1$
      by *auto*
    thus *False*
      using ‹*cmod k* $<$ *1*› ‹*cmod u'* $<$ *1*›
      using *mult-strict-mono*[*of cmod k 1 cmod u' 1*]
      by (*simp add*: *norm-mult*)
  qed

  have *dv* $\neq$ *0*
  proof (*rule ccontr*)
    assume $\neg$ *?thesis*

```
  hence cmod (1 − cnj k ∗ v′) = 0
    using ‹dv = 1 − cnj k ∗ v′›
    by auto
  hence cmod (cnj k ∗ v′) = 1
    by auto
  thus False
    using ‹cmod k < 1› ‹cmod v′ < 1›
    using mult-strict-mono[of cmod k 1 cmod v′ 1]
    by (simp add: norm-mult)
qed

have kk ≠ 0
proof (rule ccontr)
  assume ¬ ?thesis
  hence cmod (1 − k ∗ cnj k) = 0
    using ‹kk = 1 − k ∗ cnj k›
    by auto
  hence cmod (k ∗ cnj k) = 1
    by auto
  thus False
    using ‹cmod k < 1›
    using mult-strict-mono[of cmod k 1 cmod k 1]
    using complex-mod-sqrt-Re-mult-cnj by auto
qed

note nz = ‹du ≠ 0› ‹dv ≠ 0› ‹kk ≠ 0›


have nu / du − nv / dv = (nu∗dv − nv∗du) / (du ∗ dv)
  using nz
  by (simp add: field-simps)
hence (cmod (nu/du − nv/dv))^2 = cmod ((d∗kk) / (du∗dv) ∗ (cnj ((d∗kk) / (du∗dv)))) (is ?lhs = -)
  unfolding complex-mod-mult-cnj[symmetric]
  by (subst (asm) d) simp
also have ... = cmod ((d∗cnj d∗kk∗kk) / (du∗cnj du∗dv∗cnj dv))
  by (simp add: field-simps norm-mult norm-divide)
finally have 1: ?lhs = cmod ((d∗cnj d∗kk∗kk) / (du∗cnj du∗dv∗cnj dv)) .

have (1 − ((cmod nu) / (cmod du))^2)∗(1 − ((cmod nv) / (cmod dv))^2) =
      (1 − cmod((nu ∗ cnj nu) / (du ∗ cnj du)))∗(1 − cmod((nv ∗ cnj nv) / (dv ∗ cnj dv))) (is ?rhs = -)
  by (metis norm-divide complex-mod-mult-cnj power-divide)
also have ... = cmod(((du∗cnj du − nu∗cnj nu) / (du ∗ cnj du)) ∗ ((dv∗cnj dv − nv∗cnj nv) / (dv ∗ cnj dv)))
proof−
  have u′ ≠ 1 / cnj k v′ ≠ 1 / cnj k
    using ‹cmod u′ < 1› ‹cmod v′ < 1› ‹cmod k < 1›
    by (auto simp add: False norm-divide)
  moreover
  have cmod k ≠ 1
    using ‹cmod k < 1›
    by linarith
  ultimately
  have cmod (nu/du) < 1 cmod (nv/dv) < 1
    using ∗∗(1−4)
    using unit-disc-fix-discI[OF blaschke-unit-disc-fix[OF ‹cmod k < 1›] ‹u ∈ unit-disc›] ‹u′ = to-complex u›
    using unit-disc-fix-discI[OF blaschke-unit-disc-fix[OF ‹cmod k < 1›] ‹v ∈ unit-disc›] ‹v′ = to-complex v›
    using inf-or-of-complex[of u] ‹u ∈ unit-disc› inf-or-of-complex[of v] ‹v ∈ unit-disc›
    using moebius-pt-blaschke[of k u′] using moebius-pt-blaschke[of k v′]
    by auto
  hence (cmod (nu/du))^2 < 1 (cmod (nv/dv))^2 < 1
    by (simp-all add: cmod-def)
  hence cmod (nu ∗ cnj nu / (du ∗ cnj du)) < 1  cmod (nv ∗ cnj nv / (dv ∗ cnj dv)) < 1
    by (metis complex-mod-mult-cnj norm-divide power-divide)+
  moreover
  have is-real (nu ∗ cnj nu / (du ∗ cnj du)) is-real (nv ∗ cnj nv / (dv ∗ cnj dv))
    using eq-cnj-iff-real[of nu ∗ cnj nu / (du ∗ cnj du)]
    using eq-cnj-iff-real[of nv ∗ cnj nv / (dv ∗ cnj dv)]
```

by (*auto simp add*: *mult.commute*)
moreover
have *Re* (*nu* ∗ *cnj nu* / (*du* ∗ *cnj du*)) ≥ *0* *Re* (*nv* ∗ *cnj nv* / (*dv* ∗ *cnj dv*)) ≥ *0*
using ‹*du* ≠ *0*› ‹*dv* ≠ *0*›
unfolding *complex-mult-cnj-cmod*
by *simp-all*
ultimately
have *1* − *cmod* (*nu* ∗ *cnj nu* / (*du* ∗ *cnj du*)) = *cmod* (*1* − *nu* ∗ *cnj nu* / (*du* ∗ *cnj du*))
1 − *cmod* (*nv* ∗ *cnj nv* / (*dv* ∗ *cnj dv*)) = *cmod* (*1* − *nv* ∗ *cnj nv* / (*dv* ∗ *cnj dv*))
by (*simp-all add*: *cmod-def*)
thus *?thesis*
using *nz*
by (*simp add*: *diff-divide-distrib norm-mult*)
qed
also have ... = *cmod*(((*ddu* ∗ *kk*) / (*du* ∗ *cnj du*)) ∗ ((*ddv* ∗ *kk*) / (*dv* ∗ *cnj dv*)))
by (*subst ddu*, *subst ddv*, *simp*)
also have ... = *cmod*((*ddu*∗*ddv*∗*kk*∗*kk*) / (*du*∗*cnj du*∗*dv*∗*cnj dv*))
by (*simp add*: *field-simps*)
finally have *2*: *?rhs* = *cmod*((*ddu*∗*ddv*∗*kk*∗*kk*) / (*du*∗*cnj du*∗*dv*∗*cnj dv*))
.

have *?lhs* / *?rhs* =
cmod ((*d*∗*cnj d*∗*kk*∗*kk*) / (*du*∗*cnj du*∗*dv*∗*cnj dv*)) / *cmod*((*ddu*∗*ddv*∗*kk*∗*kk*) / (*du*∗*cnj du*∗*dv*∗*cnj dv*))
by (*subst 1*, *subst 2*, *simp*)
also have ... = *cmod* ((*d*∗*cnj d*)/(*ddu*∗*ddv*))
using *nz* **by** (*simp add*: *norm-mult norm-divide*)
also have ... = $(cmod\ d)^2$ / $((1 - (cmod\ u')^2){*}(1 - (cmod\ v')^2))$
proof−
have $(cmod\ u')^2 < 1$ $(cmod\ v')^2 < 1$
using ‹*cmod u*′ < *1*› ‹*cmod v*′ < *1*›
by (*simp-all add*: *cmod-def*)
hence *cmod* (*1* − *u*′ ∗ *cnj u*′) = $1 - (cmod\ u')^2$ *cmod* (*1* − *v*′ ∗ *cnj v*′) = $1 - (cmod\ v')^2$
by (*auto simp add*: *cmod-eq-Re cmod-power2 power2-eq-square*[*symmetric*])
thus *?thesis*
using *nz*
by (*simp add*: ∗∗(*6*) ∗∗(*7*) *norm-divide norm-mult power2-eq-square*)
qed
finally
have *3*: *?lhs* / *?rhs* = $(cmod\ d)^2$ / $((1 - (cmod\ u')^2){*}(1 - (cmod\ v')^2))$ **.**

have *cmod k* ≠ *1* *u*′ ≠ *1* / *cnj k* *v*′ ≠ *1* / *cnj k* *u* ≠ ∞_h *v* ≠ ∞_h
using ‹*cmod k* < *1*› ‹*u* ∈ *unit-disc*› ‹*v* ∈ *unit-disc*› ∗ ‹*k* ≠ *0*› ∗∗ ‹*kk* ≠ *0*› *nz*
by *auto*
thus *?thesis* **using** *assms*
using ∗ ∗∗ *3*
using *moebius-pt-blaschke*[*of k u*′]
using *moebius-pt-blaschke*[*of k v*′]
by (*simp add*: *norm-divide*)
qed

To prove the equivalence between the h-distance definition and the distance formula, we shall employ the without loss of generality principle. Therefore, we must show that the distance formula is preserved by h-isometries.

Rotation preserve *poincare-distance-formula*.

lemma *rotation-preserve-distance-formula* [*simp*]:
assumes *u* ∈ *unit-disc* *v* ∈ *unit-disc*
shows *poincare-distance-formula* (*to-complex* (*moebius-pt* (*moebius-rotation* φ) *u*)) (*to-complex* (*moebius-pt* (*moebius-rotation* φ) *v*)) =
poincare-distance-formula (*to-complex u*) (*to-complex v*)
using *assms*
using *inf-or-of-complex*[*of u*] *inf-or-of-complex*[*of v*]
by (*auto simp*: *norm-mult*)

Unit disc fixing Möbius preserve *poincare-distance-formula*.

lemma *unit-disc-fix-preserve-distance-formula* [*simp*]:
assumes *unit-disc-fix M* *u* ∈ *unit-disc* *v* ∈ *unit-disc*

shows *poincare-distance-formula* (*to-complex* (*moebius-pt M u*)) (*to-complex* (*moebius-pt M v*)) =
poincare-distance-formula (*to-complex u*) (*to-complex v*) (**is** *?P′ u v M*)
proof −
have ∀ *u* ∈ *unit-disc*. ∀ *v* ∈ *unit-disc*. *?P′ u v M* (**is** *?P M*)
proof (*rule wlog-unit-disc-fix*[*OF assms*(*1*)])
fix *k*
assume *cmod k* < *1*
hence *of-complex k* ∈ *unit-disc*
by *simp*
thus *?P* (*blaschke k*)
using *blaschke-preserve-distance-formula*
by *simp*
next
fix *φ*
show *?P* (*moebius-rotation φ*)
using *rotation-preserve-distance-formula*
by *simp*
next
fix *M1 M2*
assume ∗: *?P M1* **and** ∗∗: *?P M2* **and** *u11*: *unit-disc-fix M1 unit-disc-fix M2*
thus *?P* (*M1* + *M2*)
by (*auto simp del*: *poincare-distance-formula-def*)
qed
thus *?thesis*
using *assms*
by *simp*
qed

The equivalence between the two h-distance representations.

lemma *poincare-distance-formula*:
assumes *u* ∈ *unit-disc* **and** *v* ∈ *unit-disc*
shows *poincare-distance u v* = *poincare-distance-formula* (*to-complex u*) (*to-complex v*) (**is** *?P u v*)
proof (*rule wlog-x-axis*)
fix *x*
assume ∗: *is-real x 0* ≤ *Re x Re x* < *1*
show *?P* 0_h (*of-complex x*) (**is** *?lhs* = *?rhs*)
proof −
have *of-complex x* ∈ *unit-disc of-complex x* ∈ *circline-set x-axis cmod x* < *1*
using ∗ *cmod-eq-Re*
by (*auto simp add*: *circline-set-x-axis*)
hence *?lhs* = |*ln* (*Re* ((*1* − *x*) / (*1* + *x*)))|
using *poincare-distance-zero-x-axis*[*of of-complex x*]
by *simp*
moreover
have *?rhs* = |*ln* (*Re* ((*1* − *x*) / (*1* + *x*)))|
proof −
let *?x* = *1* + *2* ∗ (*cmod x*)^2 / (*1* − (*cmod x*)^2)
have *0* ≤ *2* ∗ (*cmod x*)^2 / (*1* − (*cmod x*)^2)
by (*smt* ‹*cmod x* < *1*› *divide-nonneg-nonneg norm-ge-zero power-le-one zero-le-power2*)
hence *arcosh-real-gt*: *1* ≤ *?x*
by *auto*
have *?rhs* = *arcosh ?x*
by *simp*
also have ... = *ln* ((*1* + (*cmod x*)^2) / (*1* − (*cmod x*)^2) + *2* ∗ (*cmod x*) / (*1* − (*cmod x*)^2))
proof −
have *1* − (*cmod x*)^2 > *0*
using ‹*cmod x* < *1*›
by (*smt norm-not-less-zero one-power2 power2-eq-imp-eq power-mono*)
hence *1*: *?x* = (*1* + (*cmod x*)^2) / (*1* − (*cmod x*)^2)
by (*simp add*: *field-simps*)
have *2*: *?x*^{2} − *1* = (*4* ∗ (*cmod x*)^2) / (*1* − (*cmod x*)^2)^2
using ‹*1* − (*cmod x*)^2 > *0*›
apply (*subst 1*)
unfolding *power-divide*
by (*subst divide-diff-eq-iff*, *simp*, *simp add*: *power2-eq-square field-simps*)
show *?thesis*

```
        using ‹1 − (cmod x)^2 > 0›
        apply (subst arcosh-real-def[OF arcosh-real-gt])
        apply (subst 2)
        apply (subst 1)
        apply (subst real-sqrt-divide)
        apply (subst real-sqrt-mult)
        apply simp
        done
      qed
      also have ... = ln (((1 + (cmod x))^2) / (1 − (cmod x)^2))
        apply (subst add-divide-distrib[symmetric])
        apply (simp add: field-simps power2-eq-square)
        done
      also have ... = ln ((1 + cmod x) / (1 − (cmod x)))
        using ‹cmod x < 1›
        using square-diff-square-factored[of 1 cmod x]
        by (simp add: power2-eq-square)
      also have ... = |ln (Re ((1 − x) / (1 + x)))|
      proof−
        have ∗: Re ((1 − x) / (1 + x)) ≤ 1 Re ((1 − x) / (1 + x)) > 0
          using ‹is-real x› ‹Re x ≥ 0› ‹Re x < 1›
          using complex-is-Real-iff
          by auto
        hence |ln (Re ((1 − x) / (1 + x)))| = − ln (Re ((1 − x) / (1 + x)))
          by auto
        hence |ln (Re ((1 − x) / (1 + x)))| = ln (Re ((1 + x) / (1 − x)))
          using ln-div[of 1 Re ((1 − x)/(1 + x))] ∗ ‹is-real x›
          by (fastforce simp: complex-is-Real-iff)
        moreover
        have ln ((1 + cmod x) / (1 − cmod x)) = ln ((1 + Re x) / (1 − Re x))
          using ‹Re x ≥ 0› ‹is-real x›
          using cmod-eq-Re by auto
        moreover
        have (1 + Re x) / (1 − Re x) = Re ((1 + x) / (1 − x))
          using ‹is-real x› ‹Re x < 1›
          by (smt Re-divide-real eq-iff-diff-eq-0 minus-complex.simps one-complex.simps plus-complex.simps)
        ultimately
        show ?thesis
          by simp
      qed
      finally
      show ?thesis
        .
    qed
    ultimately
    show ?thesis
      by simp
  qed
next
  fix M u v
  assume ∗: unit-disc-fix M  u ∈ unit-disc v ∈ unit-disc
  assume ?P (moebius-pt M u) (moebius-pt M v)
  thus ?P u v
    using ∗(1−3)
    by (simp del: poincare-distance-formula-def)
next
  show u ∈ unit-disc v ∈ unit-disc
    by fact+
qed
```

Some additional properties proved easily using the distance formula.

poincare-distance is symmetric.

```
lemma poincare-distance-sym:
  assumes u ∈ unit-disc and v ∈ unit-disc
  shows poincare-distance u v = poincare-distance v u
```

using *assms*
using *poincare-distance-formula*[*OF assms*(*1*) *assms*(*2*)]
using *poincare-distance-formula*[*OF assms*(*2*) *assms*(*1*)]
by (*simp add*: *mult.commute norm-minus-commute*)

lemma *poincare-distance-formula'-ge-1*:
assumes $u \in$ *unit-disc* **and** $v \in$ *unit-disc*
shows $1 \leq$ *poincare-distance-formula'* (*to-complex u*) (*to-complex v*)
using *unit-disc-cmod-square-lt-1*[*OF assms*(*1*)] *unit-disc-cmod-square-lt-1*[*OF assms*(*2*)]
by *auto*

poincare-distance is non-negative.

lemma *poincare-distance-ge0*:
assumes $u \in$ *unit-disc* **and** $v \in$ *unit-disc*
shows *poincare-distance u v* ≥ 0
using *poincare-distance-formula'-ge-1 assms* **by** (*simp add*: *poincare-distance-formula*)

lemma *cosh-dist*:
assumes $u \in$ *unit-disc* **and** $v \in$ *unit-disc*
shows *cosh-dist u v* = *poincare-distance-formula'* (*to-complex u*) (*to-complex v*)
using *poincare-distance-formula*[*OF assms*] *poincare-distance-formula'-ge-1*[*OF assms*]
by *simp*

poincare-distance is zero only if the two points are equal.

lemma *poincare-distance-eq-0-iff*:
assumes $u \in$ *unit-disc* **and** $v \in$ *unit-disc*
shows *poincare-distance u v* = *0* $\longleftrightarrow$ *u* = *v*
using *assms*
apply *auto*
using *poincare-distance-formula'-ge-1*[*OF assms*]
using *unit-disc-cmod-square-lt-1*[*OF assms*(*1*)] *unit-disc-cmod-square-lt-1*[*OF assms*(*2*)]
apply (*simp add*: *poincare-distance-formula*)
by (*simp add*: *unit-disc-to-complex-inj*)

Conjugate preserve *poincare-distance-formula*.

lemma *conjugate-preserve-poincare-distance* [*simp*]:
assumes $u \in$ *unit-disc* **and** $v \in$ *unit-disc*
shows *poincare-distance* (*conjugate u*) (*conjugate v*) = *poincare-distance u v*
proof−
obtain *u' v'* **where** $*$: *u* = *of-complex u' v* = *of-complex v'*
using *assms inf-or-of-complex*[*of u*] *inf-or-of-complex*[*of v*]
by *auto*

have $**$: *conjugate u* $\in$ *unit-disc conjugate v* $\in$ *unit-disc*
using $*$ *assms*
by *auto*

show *?thesis*
using $*$
using *poincare-distance-formula*[*OF assms*]
using *poincare-distance-formula*[*OF* $**$]
by (*metis complex-cnj-diff complex-mod-cnj conjugate-of-complex poincare-distance-def poincare-distance-formula'-def poincare-distance-formula-def to-complex-of-complex*)
qed

5.2 Existence and uniqueness of points with a given distance

lemma *ex-x-axis-poincare-distance-negative'*:
fixes *d* :: *real*
assumes $d \geq 0$
shows *let* $z = (1 - exp\ d) / (1 + exp\ d)$
in is-real z $\wedge$ *Re z* $\leq 0 \wedge$ *Re z* $> -1 \wedge$
of-complex z $\in$ *unit-disc* $\wedge$ *of-complex z* $\in$ *circline-set x-axis* $\wedge$
poincare-distance 0_h (*of-complex z*) = *d*
proof−

```
  have exp d ≥ 1
    using assms
    using one-le-exp-iff[of d, symmetric]
    by blast

  hence 1 + exp d ≠ 0
    by linarith

  let ?z = (1 − exp d) / (1 + exp d)

  have ?z ≤ 0
    using ‹exp d ≥ 1›
    by (simp add: divide-nonpos-nonneg)

  moreover

  have ?z > −1
    using exp-gt-zero[of d]
    by (smt divide-less-eq-1-neg nonzero-minus-divide-right)

  moreover

  hence abs ?z < 1
    using ‹?z ≤ 0›
    by simp
  hence cmod ?z < 1
    by (metis norm-of-real)
  hence of-complex ?z ∈ unit-disc
    by simp

  moreover
  have of-complex ?z ∈ circline-set x-axis
    unfolding circline-set-x-axis
    by simp

  moreover
  have (1 − ?z) / (1 + ?z) = exp d
  proof−
    have 1 + ?z = 2 / (1 + exp d)
      using ‹1 + exp d ≠ 0›
      by (subst add-divide-eq-iff, auto)
    moreover
    have 1 − ?z = 2 ∗ exp d / (1 + exp d)
      using ‹1 + exp d ≠ 0›
      by (subst diff-divide-eq-iff, auto)
    ultimately
    show ?thesis
      using ‹1 + exp d ≠ 0›
      by simp
  qed

  ultimately
  show ?thesis
    using poincare-distance-zero-x-axis[of of-complex ?z]
    using ‹d ≥ 0› ‹exp d ≥ 1›
    by simp (simp add: cmod-eq-Re)
qed

lemma ex-x-axis-poincare-distance-negative:
  assumes d ≥ 0
  shows ∃ z. is-real z ∧ Re z ≤ 0 ∧ Re z > −1 ∧
            of-complex z ∈ unit-disc ∧ of-complex z ∈ circline-set x-axis ∧
            poincare-distance 0ₕ (of-complex z) = d (is ∃ z. ?P z)
  using ex-x-axis-poincare-distance-negative'[OF assms]
  unfolding Let-def
  by blast
```

For each real number d there is exactly one point on the positive x-axis such that h-distance between 0 and that point is d.

lemma *unique-x-axis-poincare-distance-negative*:
  assumes $d \geq 0$
  shows $\exists!\ z.\ \textit{is-real}\ z \wedge Re\ z \leq 0 \wedge Re\ z > -1 \wedge$
    $\textit{poincare-distance}\ 0_h\ (\textit{of-complex}\ z) = d$ (**is** $\exists!\ z.\ ?P\ z$)
proof−
  let $?z = (1 - exp\ d)\ /\ (1 + exp\ d)$

  have $?P\ ?z$
    using *ex-x-axis-poincare-distance-negative*′[*OF assms*]
    unfolding *Let-def*
    by *blast*

  moreover

  have $\forall\ z'.\ ?P\ z' \longrightarrow z' = ?z$
  proof−
    let $?g = \lambda\ x'.\ |ln\ (Re\ ((1 - x')\ /\ (1 + x')))|$
    let $?A = \{x.\ \textit{is-real}\ x \wedge Re\ x > -1 \wedge Re\ x \leq 0\}$
    have *inj-on* (*poincare-distance* 0_h ∘ *of-complex*) *?A*
    proof (*rule comp-inj-on*)
      show *inj-on of-complex ?A*
        using *of-complex-inj*
        unfolding *inj-on-def*
        by *blast*
    next
      show *inj-on* (*poincare-distance* 0_h) (*of-complex* ‘ *?A*) (**is** *inj-on ?f* (*of-complex* ‘ *?A*))
      proof (*subst inj-on-cong*)
        have ∗: *of-complex* ‘ *?A* =
            $\{z.\ z \in \textit{unit-disc} \wedge z \in \textit{circline-set x-axis} \wedge Re\ (\textit{to-complex}\ z) \leq 0\}$ (**is** - = *?B*)
          by (*auto simp add*: *cmod-eq-Re circline-set-x-axis*)

        fix *x*
        assume $x \in$ *of-complex* ‘ *?A*
        hence $x \in\ ?B$
          using ∗
          by *simp*
        thus *poincare-distance* 0_h x = (*?g* ∘ *to-complex*) x
          using *poincare-distance-zero-x-axis*
          by (*simp add*: *Let-def*)
      next
        have ∗: *to-complex* ‘ *of-complex* ‘ *?A* = *?A*
          by (*auto simp add*: *image-iff*)

        show *inj-on* (*?g* ∘ *to-complex*) (*of-complex* ‘ *?A*)
        proof (*rule comp-inj-on*)
          show *inj-on to-complex* (*of-complex* ‘ *?A*)
            unfolding *inj-on-def*
            by *auto*
        next
          have *inj-on ?g ?A*
            unfolding *inj-on-def*
          proof(*safe*)
            fix *x y*
            assume *hh*: *is-real x is-real y* $-\ 1 < Re\ x\ Re\ x \leq 0$
              $-\ 1 < Re\ y\ Re\ y \leq 0\ |ln\ (Re\ ((1 - x)\ /\ (1 + x)))| = |ln\ (Re\ ((1 - y)\ /\ (1 + y)))|$

            have *is-real* $((1 - x)/(1 + x))$
              using ‹*is-real x*› *div-reals*[*of* $1{-}x$ $1{+}x$]
              by *auto*
            have *is-real* $((1 - y)/(1 + y))$
              using ‹*is-real y*› *div-reals*[*of* $1{-}y$ $1{+}y$]
              by *auto*

            have $Re\ (1 + x) > 0$

```
            using ‹− 1 < Re x› by auto
          hence 1 + x ≠ 0
            by force
          have Re (1 − x) ≥ 0
            using ‹Re x ≤ 0› by auto
          hence Re ((1 − x)/(1 + x)) > 0
            using Re-divide-real ‹0 < Re (1 + x)› complex-eq-if-Re-eq hh(1) hh(4) by auto
          have Re(1 − x) ≥ Re ( 1 + x)
            using hh by auto
          hence Re ((1 − x)/(1 + x)) ≥ 1
            using ‹Re (1 + x) > 0› ‹is-real ((1 − x)/(1 + x))›
          by (smt Re-divide-real arg-0-iff hh(1) le-divide-eq-1-pos one-complex.simps(2) plus-complex.simps(2))

          have Re (1 + y) > 0
            using ‹− 1 < Re y› by auto
          hence 1 + y ≠ 0
            by force
          have Re (1 − y) ≥ 0
            using ‹Re y ≤ 0› by auto
          hence Re ((1 − y)/(1 + y)) > 0
            using Re-divide-real ‹0 < Re (1 + y)› complex-eq-if-Re-eq hh by auto
          have Re(1 − y) ≥ Re ( 1 + y)
            using hh by auto
          hence Re ((1 − y)/(1 + y)) ≥ 1
            using ‹Re (1 + y) > 0› ‹is-real ((1 − y)/(1 + y))›
            by (smt Re-divide-real arg-0-iff hh le-divide-eq-1-pos one-complex.simps(2) plus-complex.simps(2))

          have ln (Re ((1 − x) / (1 + x))) = ln (Re ((1 − y) / (1 + y)))
            using ‹Re ((1 − y)/(1 + y)) ≥ 1› ‹Re ((1 − x)/(1 + x)) ≥ 1› hh
            by auto
          hence Re ((1 − x) / (1 + x)) = Re ((1 − y) / (1 + y))
            using ‹Re ((1 − y)/(1 + y)) > 0› ‹Re ((1 − x)/(1 + x)) > 0›
            by auto
          hence (1 − x) / (1 + x) = (1 − y) / (1 + y)
            using ‹is-real ((1 − y)/(1 + y))› ‹is-real ((1 − x)/(1 + x))›
            using complex-eq-if-Re-eq by blast
          hence (1 − x) ∗ (1 + y) = (1 − y) ∗ (1 + x)
            using ‹1 + y ≠ 0› ‹1 + x ≠ 0›
            by (simp add:field-simps)
          thus x = y
            by (simp add:field-simps)
        qed
        thus inj-on ?g (to-complex ‘ of-complex ‘ ?A)
          using ∗
          by simp
      qed
    qed
  qed
  thus ?thesis
    using ‹?P ?z›
    unfolding inj-on-def
    by auto
qed
ultimately
show ?thesis
  by blast
qed

lemma ex-x-axis-poincare-distance-positive:
  assumes d ≥ 0
  shows ∃ z. is-real z ∧ Re z ≥ 0 ∧ Re z < 1 ∧
            of-complex z ∈ unit-disc ∧ of-complex z ∈ circline-set x-axis ∧
            poincare-distance 0ₕ (of-complex z) = d (is ∃ z. is-real z ∧ Re z ≥ 0 ∧ Re z < 1 ∧ ?P z)
proof−
  obtain z where ∗: is-real z Re z ≤ 0 Re z > −1 ?P z
    using ex-x-axis-poincare-distance-negative[OF assms]
```

by *auto*
hence ∗∗: *of-complex* $z \in$ *unit-disc* *of-complex* $z \in$ *circline-set x-axis*
by (*auto simp add*: *cmod-eq-Re*)
have *is-real* $(-z) \wedge Re\ (-z) \geq 0 \wedge Re\ (-z) < 1 \wedge\ ?P\ (-z)$
using ∗ ∗∗
by (*simp add*: *circline-set-x-axis*)
thus *?thesis*
by *blast*
qed

lemma *unique-x-axis-poincare-distance-positive*:
assumes $d \geq 0$
shows $\exists!\ z.$ *is-real* $z \wedge Re\ z \geq 0 \wedge Re\ z < 1 \wedge$
poincare-distance 0_h (*of-complex* z) $= d$ (**is** $\exists!\ z.$ *is-real* $z \wedge Re\ z \geq 0 \wedge Re\ z < 1 \wedge\ ?P\ z$)
proof−
obtain z **where** ∗: *is-real* z $Re\ z \leq 0$ $Re\ z > -1$ $?P\ z$
using *unique-x-axis-poincare-distance-negative*[*OF assms*]
by *auto*
hence ∗∗: *of-complex* $z \in$ *unit-disc* *of-complex* $z \in$ *circline-set x-axis*
by (*auto simp add*: *cmod-eq-Re circline-set-x-axis*)
show *?thesis*
proof
show *is-real* $(-z) \wedge Re\ (-z) \geq 0 \wedge Re\ (-z) < 1 \wedge\ ?P\ (-z)$
using ∗ ∗∗
by *simp*
next
fix z'
assume *is-real* $z' \wedge Re\ z' \geq 0 \wedge Re\ z' < 1 \wedge\ ?P\ z'$
hence *is-real* $(-z') \wedge Re\ (-z') \leq 0 \wedge Re\ (-z') > -1 \wedge\ ?P\ (-z')$
by (*auto simp add*: *circline-set-x-axis cmod-eq-Re*)
hence $-z' = z$
using *unique-x-axis-poincare-distance-negative*[*OF assms*] ∗
by *blast*
thus $z' = -z$
by *auto*
qed
qed

Equal distance implies that segments are isometric - this means that congruence could be defined either by two segments having the same distance or by requiring existence of an isometry that maps one segment to the other.

lemma *poincare-distance-eq-ex-moebius*:
assumes *in-disc*: $u \in$ *unit-disc* **and** $v \in$ *unit-disc* **and** $u' \in$ *unit-disc* **and** $v' \in$ *unit-disc*
assumes *poincare-distance* $u\ v =$ *poincare-distance* $u'\ v'$
shows $\exists\ M.$ *unit-disc-fix* $M \wedge$ *moebius-pt* $M\ u = u' \wedge$ *moebius-pt* $M\ v = v'$ (**is** $?P'\ u\ v\ u'\ v'$)
proof (*cases* $u = v$)
case *True*
thus *?thesis*
using *assms poincare-distance-eq-0-iff*[*of* $u'\ v'$]
by (*simp add*: *unit-disc-fix-transitive*)
next
case *False*
have $\forall\ u'\ v'.\ u \neq v \wedge u' \in$ *unit-disc* $\wedge\ v' \in$ *unit-disc* $\wedge$ *poincare-distance* $u\ v =$ *poincare-distance* $u'\ v' \longrightarrow$
$?P'\ u'\ v'\ u\ v$ (**is** $?P\ u\ v$)
proof (*rule wlog-positive-x-axis*[**where** $P{=}?P$])
fix x
assume *is-real* x $0 < Re\ x$ $Re\ x < 1$
hence *of-complex* $x \in$ *unit-disc* *of-complex* $x \in$ *circline-set x-axis*
unfolding *circline-set-x-axis*
by (*auto simp add*: *cmod-eq-Re*)

show $?P\ 0_h$ (*of-complex* x)
proof *safe*
fix $u'\ v'$
assume $0_h \neq$ *of-complex* x **and** *in-disc*: $u' \in$ *unit-disc* $v' \in$ *unit-disc* **and**
poincare-distance 0_h (*of-complex* x) $=$ *poincare-distance* $u'\ v'$
hence $u' \neq v'$ *poincare-distance* $u'\ v' > 0$

using *poincare-distance-eq-0-iff*[*of* 0_h *of-complex x*] ‹*of-complex x* ∈ *unit-disc*›
using *poincare-distance-ge0*[*of* 0_h *of-complex x*]
by *auto*
then obtain *M* **where** *M*: *unit-disc-fix M moebius-pt M u′* = 0_h *moebius-pt M v′* ∈ *positive-x-axis*
using *ex-unit-disc-fix-to-zero-positive-x-axis*[*of u′ v′*] *in-disc*
by *auto*

then obtain *Mv′* **where** *Mv′*: *moebius-pt M v′* = *of-complex Mv′*
using *inf-or-of-complex*[*of moebius-pt M v′*] *in-disc unit-disc-fix-iff*[*of M*]
by (*metis image-eqI inf-notin-unit-disc*)

have *moebius-pt M v′* ∈ *unit-disc*
using *M*(*1*) ‹*v′* ∈ *unit-disc*›
by *auto*

have *Re Mv′* > *0 is-real Mv′ Re Mv′* < *1*
using *M Mv′ of-complex-inj* ‹*moebius-pt M v′* ∈ *unit-disc*›
unfolding *positive-x-axis-def circline-set-x-axis*
using *cmod-eq-Re*
by *auto fastforce*

have *poincare-distance* 0_h (*moebius-pt M v′*) = *poincare-distance u′ v′*
using *M*(*1*)
using *in-disc*
by (*subst M*(*2*)[*symmetric*], *simp*)

have *Mv′* = *x*
using ‹*poincare-distance* 0_h (*moebius-pt M v′*) = *poincare-distance u′ v′*› *Mv′*
using ‹*poincare-distance* 0_h (*of-complex x*) = *poincare-distance u′ v′*›
using *unique-x-axis-poincare-distance-positive*[*of poincare-distance u′ v′*]
‹*poincare-distance u′ v′* > *0*›
using ‹*Re Mv′* > *0*› ‹*Re Mv′* < *1*› ‹*is-real Mv′*›
using ‹*is-real x*› ‹*Re x* > *0*› ‹*Re x* < *1*›
unfolding *positive-x-axis-def*
by *auto*

thus *?P′ u′ v′* 0_h (*of-complex x*)
using *M Mv′*
by *auto*
qed
next
show *u* ∈ *unit-disc v* ∈ *unit-disc u* ≠ *v*
by *fact+*
next
fix *M u v*
let *?Mu* = *moebius-pt M u* **and** *?Mv* = *moebius-pt M v*
assume *1*: *unit-disc-fix M u* ∈ *unit-disc v* ∈ *unit-disc u* ≠ *v*
hence *2*: *?Mu* ≠ *?Mv ?Mu* ∈ *unit-disc ?Mv* ∈ *unit-disc*
by *auto*
assume *3*: *?P* (*moebius-pt M u*) (*moebius-pt M v*)
show *?P u v*
proof *safe*
fix *u′ v′*
assume *4*: *u′* ∈ *unit-disc v′* ∈ *unit-disc poincare-distance u v* = *poincare-distance u′ v′*
hence *poincare-distance ?Mu ?Mv* = *poincare-distance u v*
using *1*
by *simp*
then obtain *M′* **where** *5*: *unit-disc-fix M′ moebius-pt M′ u′* = *?Mu moebius-pt M′ v′* = *?Mv*
using *2 3 4*
by *auto*
let *?M* = (−*M*) + *M′*
have *unit-disc-fix ?M* ∧ *moebius-pt ?M u′* = *u* ∧ *moebius-pt ?M v′* = *v*
using *5* ‹*unit-disc-fix M*›
using *unit-disc-fix-moebius-comp*[*of* −*M M′*]
using *unit-disc-fix-moebius-inv*[*of M*]
by *simp*

thus $\exists M.$ *unit-disc-fix M* $\wedge$ *moebius-pt M u'* $=$ *u* $\wedge$ *moebius-pt M v'* $=$ *v*
by *blast*
qed
qed
then obtain *M* **where** *unit-disc-fix M* $\wedge$ *moebius-pt M u'* $=$ *u* $\wedge$ *moebius-pt M v'* $=$ *v*
using *assms* ‹*u* $\neq$ *v*›
by *blast*
hence *unit-disc-fix* $(-M)$ $\wedge$ *moebius-pt* $(-M)$ *u* $=$ *u'* $\wedge$ *moebius-pt* $(-M)$ *v* $=$ *v'*
using *unit-disc-fix-moebius-inv*[*of M*]
by *auto*
thus *?thesis*
by *blast*
qed

lemma *unique-midpoint-x-axis*:
assumes *x*: *is-real x* $-1 <$ *Re x Re x* < 1 **and**
y: *is-real y* $-1 <$ *Re y Re y* < 1 **and**
$x \neq y$
shows $\exists!$ *z*. $-1 <$ *Re z* $\wedge$ *Re z* < 1 $\wedge$ *is-real z* $\wedge$ *poincare-distance* (*of-complex z*) (*of-complex x*) $=$ *poincare-distance* (*of-complex z*) (*of-complex y*) (**is** $\exists!$ *z*. *?R z* (*of-complex x*) (*of-complex y*))
proof−
let *?x* $=$ *of-complex x* **and** *?y* $=$ *of-complex y*
let *?P* $= \lambda$ *x y*. $\exists!$ *z*. *?R z x y*
have $\forall$ *x*. $-1 <$ *Re x* $\wedge$ *Re x* < 1 $\wedge$ *is-real x* $\wedge$ *of-complex x* $\neq$ *?y* $\longrightarrow$ *?P* (*of-complex x*) *?y* (**is** *?Q* (*of-complex y*))
proof (*rule wlog-real-zero*)
show *?y* $\in$ *unit-disc*
using *y*
by (*simp add*: *cmod-eq-Re*)
next
show *is-real* (*to-complex ?y*)
using *y*
by *simp*
next
show *?Q* 0_h
proof (*rule allI*, *rule impI*, (*erule conjE*)+)
fix *x*
assume *x*: $-1 <$ *Re x Re x* < 1 *is-real x*
let *?x* $=$ *of-complex x*
assume *?x* $\neq 0_h$
hence $x \neq 0$
by *auto*
hence *Re x* $\neq 0$
using *x*
using *complex-neq-0*
by *auto*

have $*$: $\forall$ *a*. $-1 < a \wedge a < 1 \longrightarrow$
(*poincare-distance* (*of-complex* (*cor a*)) *?x* $=$ *poincare-distance* (*of-complex* (*cor a*)) 0_h $\longleftrightarrow$
(*Re x*) $* a * a - 2 * a +$ *Re x* $= 0$)
proof (*rule allI*, *rule impI*)
fix *a* :: *real*
assume $-1 < a \wedge a < 1$
hence *of-complex* (*cor a*) $\in$ *unit-disc*
by *auto*
moreover
have $(a -$ *Re x*$)^2$ / $((1 - a^2) * (1 - ($*Re x*$)^2)) = a^2$ / $(1 - a^2) \longleftrightarrow$
(*Re x*) $* a * a - 2 * a +$ *Re x* $= 0$ (**is** *?lhs* $\longleftrightarrow$ *?rhs*)
proof−
have $1 - a^2 \neq 0$
using ‹$-1 < a \wedge a < 1$›
by (*metis cancel-comm-monoid-add-class.diff-cancel diff-eq-diff-less less-numeral-extra*(*4*) *power2-eq-1-iff right-minus-eq*)
hence *?lhs* $\longleftrightarrow (a -$ *Re x*$)^2$ / $(1 - ($*Re x*$)^2) = a^2$
by (*smt divide-cancel-right divide-divide-eq-left mult.commute*)
also have ... $\longleftrightarrow (a -$ *Re x*$)^2 = a^2 * (1 - ($*Re x*$)^2)$
proof−

```
        have 1 − (Re x)^2 ≠ 0
          using x
          by (smt power2-eq-1-iff)
        thus ?thesis
          by (simp add: divide-eq-eq)
      qed
      also have ... ⟷ a^2 ∗ (Re x)^2 − 2∗a∗Re x + (Re x)^2 = 0
        by (simp add: power2-diff field-simps)
      also have ... ⟷ Re x ∗ (a^2 ∗ Re x − 2 ∗ a + Re x) = 0
        by (simp add: power2-eq-square field-simps)
      also have ... ⟷ ?rhs
        using ‹Re x ≠ 0›
        by (simp add: mult.commute mult.left-commute power2-eq-square)
      finally
      show ?thesis
        .
    qed
    moreover
    have arcosh (1 + 2 ∗ ((a − Re x)^2 / ((1 − a^2) ∗ (1 − (Re x)^2)))) = arcosh (1 + 2 ∗ a^2 / (1 − a^2)) ⟷ ?lhs
      using ‹−1 < a ∧ a < 1› x mult-left-cancel[of 2::real (a − Re x)^2 / ((1 − a^2) ∗ (1 − (Re x)^2)) a^2 / (1 − a^2)]
      by (subst arcosh-eq-iff, simp-all add: square-le-1)
    ultimately
    show poincare-distance (of-complex (cor a)) (of-complex x) = poincare-distance (of-complex (cor a)) 0_h ⟷
          (Re x) ∗ a ∗ a − 2 ∗ a + Re x = 0
      using x
      by (auto simp add: poincare-distance-formula cmod-eq-Re)
  qed

  show ?P ?x 0_h
  proof
    let ?a = (1 − sqrt(1 − (Re x)^2)) / (Re x)
    let ?b = (1 + sqrt(1 − (Re x)^2)) / (Re x)

    have is-real ?a
      by simp
    moreover
    have 1 − (Re x)^2 > 0
      using x
      by (smt power2-eq-1-iff square-le-1)
    have |?a| < 1
    proof (cases Re x > 0)
      case True
      have (1 − Re x)^2 < 1 − (Re x)^2
        using ‹Re x > 0› x
        by (simp add: power2-eq-square field-simps)
      hence 1 − Re x < sqrt (1 − (Re x)^2)
        using real-less-rsqrt by fastforce
      thus ?thesis
        using ‹1 − (Re x)^2 > 0› ‹Re x > 0›
        by simp
    next
      case False
      hence Re x < 0
        using ‹Re x ≠ 0›
        by simp

      have 1 + Re x > 0
        using ‹Re x > −1›
        by simp
      hence 2∗Re x + 2∗Re x∗Re x < 0
        using ‹Re x < 0›
     by (metis comm-semiring-class.distrib mult.commute mult-2-right mult-less-0-iff one-add-one zero-less-double-add-iff-zero-less-si
      hence (1 + Re x)^2 < 1 − (Re x)^2
        by (simp add: power2-eq-square field-simps)
      hence 1 + Re x < sqrt (1 − (Re x)^2)
        using ‹1 − (Re x)^2 > 0›
```

using *real-less-rsqrt* **by** *blast*
thus *?thesis*
using ‹*Re x* < *0*›
by (*simp add*: *field-simps*)
qed
hence $-1 < ?a$ $?a < 1$
by *linarith*+
moreover
have $(Re\ x) * ?a * ?a - 2 * ?a + Re\ x = 0$
using ‹$Re\ x \neq 0$› ‹$1 - (Re\ x)^2 > 0$›
by (*simp add*: *field-simps power2-eq-square*)
ultimately
show $-1 < Re\ (cor\ ?a) \wedge Re\ (cor\ ?a) < 1 \wedge$ *is-real* $?a \wedge$ *poincare-distance* (*of-complex* $?a$) (*of-complex* x) = *poincare-distance* (*of-complex* $?a$) 0_h
using ∗
by *auto*

fix *z*
assume ∗∗: $-1 < Re\ z \wedge Re\ z < 1 \wedge$ *is-real* $z \wedge$
poincare-distance (*of-complex z*) (*of-complex x*) = *poincare-distance* (*of-complex z*) 0_h
hence $Re\ x * Re\ z * Re\ z - 2 * Re\ z + Re\ x = 0$
using ∗[*rule-format*, *of Re z*] *x*
by *auto*
moreover
have $sqrt\ (4 - 4 * Re\ x * Re\ x) = 2 * sqrt(1 - Re\ x * Re\ x)$
proof−
have $sqrt\ (4 - 4 * Re\ x * Re\ x) = sqrt(4 * (1 - Re\ x * Re\ x))$
by *simp*
thus *?thesis*
by (*simp only*: *real-sqrt-mult*, *simp*)
qed
moreover
have $(2 - 2 * sqrt\ (1 - Re\ x * Re\ x)) / (2 * Re\ x) = ?a$
proof−
have $(2 - 2 * sqrt\ (1 - Re\ x * Re\ x)) / (2 * Re\ x) =$
$(2 * (1 - sqrt\ (1 - Re\ x * Re\ x))) / (2 * Re\ x)$
by *simp*
thus *?thesis*
by (*subst* (*asm*) *mult-divide-mult-cancel-left*) (*auto simp add*: *power2-eq-square*)
qed
moreover
have $(2 + 2 * sqrt\ (1 - Re\ x * Re\ x)) / (2 * Re\ x) = ?b$
proof−
have $(2 + 2 * sqrt\ (1 - Re\ x * Re\ x)) / (2 * Re\ x) =$
$(2 * (1 + sqrt\ (1 - Re\ x * Re\ x))) / (2 * Re\ x)$
by *simp*
thus *?thesis*
by (*subst* (*asm*) *mult-divide-mult-cancel-left*) (*auto simp add*: *power2-eq-square*)
qed
ultimately
have $Re\ z = ?a \vee Re\ z = ?b$
using *discriminant-nonneg*[*of Re x* −*2 Re x Re z*] *discrim-def*[*of Re x* −*2 Re x*]
using ‹$Re\ x \neq 0$› ‹$-1 < Re\ x$› ‹$Re\ x < 1$› ‹$1 - (Re\ x)^2 > 0$›
by (*auto simp add:power2-eq-square*)
have $|?b| > 1$
proof (*cases Re x* > *0*)
case *True*
have $(Re\ x - 1)^2 < 1 - (Re\ x)^2$
using ‹$Re\ x > 0$› *x*
by (*simp add*: *power2-eq-square field-simps*)
hence $Re\ x - 1 < sqrt\ (1 - (Re\ x)^2)$
using *real-less-rsqrt*
by *simp*
thus *?thesis*
using ‹$1 - (Re\ x)^2 > 0$› ‹$Re\ x > 0$›
by *simp*

```
        next
          case False
          hence Re x < 0
            using ‹Re x ≠ 0›
            by simp
          have 1 + Re x > 0
            using ‹Re x > −1›
            by simp
          hence 2∗Re x + 2∗Re x∗Re x < 0
            using ‹Re x < 0›
         by (metis comm-semiring-class.distrib mult.commute mult-2-right mult-less-0-iff one-add-one zero-less-double-add-iff-zero-less-si
          hence 1 − (Re x)^2 > (− 1 − (Re x))^2
            by (simp add: field-simps power2-eq-square)
          hence sqrt (1 − (Re x)^2) > −1 − Re x
            using real-less-rsqrt
            by simp
          thus ?thesis
            using ‹Re x < 0›
            by (simp add: field-simps)
        qed
        hence ?b < −1 ∨ ?b > 1
          by auto

        hence Re z = ?a
          using ‹Re z = ?a ∨ Re z = ?b› ∗∗
          by auto
        thus z = ?a
          using ∗∗ complex-of-real-Re
          by fastforce
      qed
    qed
  next
    fix a u
    let ?M = moebius-pt (blaschke a)
    let ?Mu = ?M u
    assume u ∈ unit-disc is-real a cmod a < 1
    assume ∗: ?Q ?Mu
    show ?Q u
    proof (rule allI, rule impI, (erule conjE)+)
      fix x
      assume x: −1 < Re x Re x < 1 is-real x of-complex x ≠ u
      let ?Mx = ?M (of-complex x)
      have of-complex x ∈ unit-disc
        using x cmod-eq-Re
        by auto
      hence ?Mx ∈ unit-disc
        using ‹is-real a› ‹cmod a < 1› blaschke-unit-disc-fix[of a]
        using unit-disc-fix-discI
        by blast
      hence ?Mx ≠ ∞h
        by auto
      moreover
      have of-complex x ∈ circline-set x-axis
        using x
        by auto
      hence ?Mx ∈ circline-set x-axis
        using blaschke-real-preserve-x-axis[OF ‹is-real a› ‹cmod a < 1›, of of-complex x]
        by auto
      hence −1 < Re (to-complex ?Mx) ∧ Re (to-complex ?Mx) < 1 ∧ is-real (to-complex ?Mx)
        using ‹?Mx ≠ ∞h› ‹?Mx ∈ unit-disc›
        unfolding circline-set-x-axis
        by (auto simp add: cmod-eq-Re)
      moreover
      have ?Mx ≠ ?Mu
        using ‹of-complex x ≠ u›
        by simp
```

ultimately
have *?P ?Mx ?Mu*
  using $*$[*rule-format*, *of to-complex ?Mx*] ‹*?Mx* $\neq \infty_h$›
  by *simp*
then obtain *Mz* **where**
  ?R Mz ?Mx ?Mu
  by *blast*
have *of-complex Mz* $\in$ *unit-disc of-complex Mz* $\in$ *circline-set x-axis*
  using ‹*?R Mz ?Mx ?Mu*›
  using *cmod-eq-Re*
  by *auto*

let *?Minv* = $-$ (*blaschke a*)
let *?z* = *moebius-pt ?Minv* (*of-complex Mz*)
have *?z* $\in$ *unit-disc*
  using ‹*of-complex Mz* $\in$ *unit-disc*› ‹*cmod a* < *1*›
  by *auto*
moreover
have *?z* $\in$ *circline-set x-axis*
  using ‹*of-complex Mz* $\in$ *circline-set x-axis*›
  using *blaschke-real-preserve-x-axis* ‹*is-real a*› ‹*cmod a* < *1*›
  by *fastforce*
ultimately
have *z1*: $-$*1* < *Re* (*to-complex ?z*) *Re* (*to-complex ?z*) < *1 is-real* (*to-complex ?z*)
  using *inf-or-of-complex*[*of ?z*]
  unfolding *circline-set-x-axis*
  by (*auto simp add*: *cmod-eq-Re*)

have *z2*: *poincare-distance ?z* (*of-complex x*) = *poincare-distance ?z u*
  using ‹*?R Mz ?Mx ?Mu*› ‹*cmod a* < *1*› ‹*?z* $\in$ *unit-disc*› ‹*of-complex x* $\in$ *unit-disc*› ‹*u* $\in$ *unit-disc*›
by (*metis blaschke-preserve-distance-formula blaschke-unit-disc-fix moebius-pt-comp-inv-right poincare-distance-formula uminus-moebius-def unit-disc-fix-discI unit-disc-iff-cmod-lt-1*)
show *?P* (*of-complex x*) *u*
proof
  show *?R* (*to-complex ?z*) (*of-complex x*) *u*
    using *z1 z2* ‹*?z* $\in$ *unit-disc*› *inf-or-of-complex*[*of ?z*]
    by *auto*
next
  fix *z'*
  assume *?R z'* (*of-complex x*) *u*
  hence *of-complex z'* $\in$ *unit-disc of-complex z'* $\in$ *circline-set x-axis*
    by (*auto simp add*: *cmod-eq-Re*)
  let *?Mz'* = *?M* (*of-complex z'*)
  have *?Mz'* $\in$ *unit-disc ?Mz'* $\in$ *circline-set x-axis*
    using ‹*of-complex z'* $\in$ *unit-disc*› ‹*of-complex z'* $\in$ *circline-set x-axis*› ‹*cmod a* < *1*› ‹*is-real a*›
    using *blaschke-unit-disc-fix unit-disc-fix-discI*
    using *blaschke-real-preserve-x-axis circline-set-x-axis*
    by *blast+*
  hence $-$*1* < *Re* (*to-complex ?Mz'*) *Re* (*to-complex ?Mz'*) < *1 is-real* (*to-complex ?Mz'*)
    unfolding *circline-set-x-axis*
    by (*auto simp add*: *cmod-eq-Re*)
  moreover
  have *poincare-distance ?Mz' ?Mx* = *poincare-distance ?Mz' ?Mu*
    using ‹*?R z'* (*of-complex x*) *u*›
    using ‹*cmod a* < *1*› ‹*of-complex x* $\in$ *unit-disc*› ‹*of-complex z'* $\in$ *unit-disc*› ‹*u* $\in$ *unit-disc*›
    by *auto*
  ultimately
  have *?R* (*to-complex ?Mz'*) *?Mx ?Mu*
    using ‹*?Mz'* $\in$ *unit-disc*› *inf-or-of-complex*[*of ?Mz'*]
    by *auto*
  hence *?Mz'* = *of-complex Mz*
    using ‹*?P ?Mx ?Mu*› ‹*?R Mz ?Mx ?Mu*›
  by (*metis* ‹*moebius-pt* (*blaschke a*) (*of-complex z'*) $\in$ *unit-disc*› ‹*of-complex Mz* $\in$ *unit-disc*› *to-complex-of-complex unit-disc-to-complex-inj*)
  thus *z'* = *to-complex ?z*
    by (*simp add*: *moebius-pt-invert*)

qed
qed
qed
thus *?thesis*
using *assms*
by (*metis to-complex-of-complex*)
qed

5.3 Triangle inequality

lemma *poincare-distance-formula-zero-sum*:
assumes $u \in \mathit{unit\text{-}disc}$ **and** $v \in \mathit{unit\text{-}disc}$
shows $\mathit{poincare\text{-}distance}\ u\ 0_h + \mathit{poincare\text{-}distance}\ 0_h\ v =$
$(\mathit{let}\ u' = \mathit{cmod}\ (\mathit{to\text{-}complex}\ u);\ v' = \mathit{cmod}\ (\mathit{to\text{-}complex}\ v)$
$\mathit{in}\ \mathit{arcosh}\ (((1 + u'^2) * (1 + v'^2) + 4 * u' * v') / ((1 - u'^2) * (1 - v'^2))))$
proof−
obtain $u'\ v'$ **where** *uv*: $u' = \mathit{to\text{-}complex}\ u\ \ v' = \mathit{to\text{-}complex}\ v$
by *auto*
have *uv'*: $u = \mathit{of\text{-}complex}\ u'\ \ v = \mathit{of\text{-}complex}\ v'$
using *uv assms inf-or-of-complex*[*of u*] *inf-or-of-complex*[*of v*]
by *auto*

let $?u' = \mathit{cmod}\ u'$ **and** $?v' = \mathit{cmod}\ v'$

have *disc*: $?u'^2 < 1\ \ ?v'^2 < 1$
using *unit-disc-cmod-square-lt-1*[*OF* ‹$u \in \mathit{unit\text{-}disc}$›]
using *unit-disc-cmod-square-lt-1*[*OF* ‹$v \in \mathit{unit\text{-}disc}$›] *uv*
by *auto*
thm *arcosh-add*
have $\mathit{arcosh}\ (1 + 2 * ?u'^2 / (1 - ?u'^2)) + \mathit{arcosh}\ (1 + 2 * ?v'^2 / (1 - ?v'^2)) =$
$\mathit{arcosh}\ (((1 + ?u'^2) * (1 + ?v'^2) + 4 * ?u' * ?v') / ((1 - ?u'^2) * (1 - ?v'^2)))$ (**is** $\mathit{arcosh}\ ?ll + \mathit{arcosh}\ ?rr =$ $\mathit{arcosh}\ ?r$)
proof (*subst arcosh-add*)
show $?ll \geq 1\ \ ?rr \geq 1$
using *disc*
by *auto*
next
show $\mathit{arcosh}\ ((1 + 2 * ?u'^2 / (1 - ?u'^2)) * (1 + 2 * ?v'^2 / (1 - ?v'^2)) +$
$\mathit{sqrt}\ (((1 + 2 * ?u'^2 / (1 - ?u'^2))^2 - 1) * ((1 + 2 * ?v'^2 / (1 - ?v'^2))^2 - 1))) =$
$\mathit{arcosh}\ ?r$ (**is** $\mathit{arcosh}\ ?l = -$)
proof−
have $1 + 2 * ?u'^2 / (1 - ?u'^2) = (1 + ?u'^2) / (1 - ?u'^2)$
using *disc*
by (*subst add-divide-eq-iff*, *simp-all*)
moreover
have $1 + 2 * ?v'^2 / (1 - ?v'^2) = (1 + ?v'^2) / (1 - ?v'^2)$
using *disc*
by (*subst add-divide-eq-iff*, *simp-all*)
moreover
have $\mathit{sqrt}\ (((1 + 2 * ?u'^2 / (1 - ?u'^2))^2 - 1) * ((1 + 2 * ?v'^2 / (1 - ?v'^2))^2 - 1)) =$
$(4 * ?u' * ?v') / ((1 - ?u'^2) * (1 - ?v'^2))$ (**is** $\mathit{sqrt}\ ?s = ?t$)
proof−
have $?s = ?t^2$
using *disc*
apply (*subst add-divide-eq-iff*, *simp*)+
apply (*subst power-divide*)+
apply *simp*
apply (*subst divide-diff-eq-iff*, *simp*)+
apply (*simp add*: *power2-eq-square field-simps*)
done
thus *?thesis*
using *disc*
by *simp*
qed
ultimately
have $?l = ?r$

```
        using disc
        by simp (subst add-divide-distrib, simp)
      thus ?thesis
        by simp
    qed
  qed
  thus ?thesis
    using uv' assms
    using poincare-distance-formula
    by (simp add: Let-def)
qed

lemma poincare-distance-triangle-inequality:
  assumes u ∈ unit-disc and v ∈ unit-disc and w ∈ unit-disc
  shows poincare-distance u v + poincare-distance v w ≥ poincare-distance u w (is ?P' u v w)
proof−
  have ∀ w. w ∈ unit-disc ⟶ ?P' u v w  (is ?P v u)
  proof (rule wlog-x-axis[where P=?P])
    fix x
    assume is-real x 0 ≤ Re x Re x < 1
    hence of-complex x ∈ unit-disc
      by (simp add: cmod-eq-Re)

    show ?P 0_h (of-complex x)
    proof safe
      fix w
      assume w ∈ unit-disc
      then obtain w' where w: w = of-complex w'
        using inf-or-of-complex[of w]
        by auto

      let ?x = cmod x and ?w = cmod w' and ?xw = cmod (x − w')

      have disc: ?x^2 < 1 ?w^2 < 1
        using unit-disc-cmod-square-lt-1[OF ‹of-complex x ∈ unit-disc›]
        using unit-disc-cmod-square-lt-1[OF ‹w ∈ unit-disc›] w
        by auto

      have poincare-distance (of-complex x) 0_h + poincare-distance 0_h w =
            arcosh (((1 + ?x^2) * (1 + ?w^2) + 4 * ?x * ?w) / ((1 − ?x^2) * (1 − ?w^2))) (is - = arcosh ?r1)
        using poincare-distance-formula-zero-sum[OF ‹of-complex x ∈ unit-disc› ‹w ∈ unit-disc›] w
        by (simp add: Let-def)
      moreover
      have poincare-distance (of-complex x) (of-complex w') =
             arcosh (((1 − ?x^2) * (1 − ?w^2) + 2 * ?xw^2) / ((1 − ?x^2) * (1 − ?w^2))) (is - = arcosh ?r2)
        using disc
        using poincare-distance-formula[OF ‹of-complex x ∈ unit-disc› ‹w ∈ unit-disc›] w
        by (subst add-divide-distrib) simp
      moreover
      have *: (1 − ?x^2) * (1 − ?w^2) + 2 * ?xw^2 ≤ (1 + ?x^2) * (1 + ?w^2) + 4 * ?x * ?w
      proof−
        have (cmod (x − w'))^2 ≤ (cmod x + cmod w')^2
          using norm-triangle-ineq4[of x w']
          by (simp add: power-mono)
        thus ?thesis
          by (simp add: field-simps power2-sum)
      qed
      have arcosh ?r1 ≥ arcosh ?r2
      proof (subst arcosh-mono)
        show ?r1 ≥ 1
          using disc
          by (smt * le-divide-eq-1-pos mult-pos-pos zero-le-power2)
      next
        show ?r2 ≥ 1
          using disc
          by simp
```

```
      next
        show ?r1 ≥ ?r2
          using disc
          using *
          by (subst divide-right-mono, simp-all)
      qed
      ultimately
      show poincare-distance (of-complex x) w ≤ poincare-distance (of-complex x) 0h + poincare-distance 0h w
        using ‹of-complex x ∈ unit-disc› ‹w ∈ unit-disc› w
        using poincare-distance-formula
        by simp
    qed
  next
    show v ∈ unit-disc u ∈ unit-disc
      by fact+
  next
    fix M u v
    assume *: unit-disc-fix M u ∈ unit-disc v ∈ unit-disc
    assume **: ?P (moebius-pt M u) (moebius-pt M v)
    show ?P u v
    proof safe
      fix w
      assume w ∈ unit-disc
      thus ?P' v u w
        by (metis * ** unit-disc-fix-discI unit-disc-fix-preserve-poincare-distance)
    qed
  qed
  thus ?thesis
    using assms
    by auto
qed

end
theory Poincare-Circles
  imports Poincare-Distance
begin
```

6 H-circles in the Poincaré model

Circles consist of points that are at the same distance from the center.

```
definition poincare-circle :: complex-homo ⇒ real ⇒ complex-homo set where
  poincare-circle z r = {z'. z' ∈ unit-disc ∧ poincare-distance z z' = r}
```

Each h-circle in the Poincaré model is represented by an Euclidean circle in the model — the center and radius of that euclidean circle are determined by the following formulas.

```
definition poincare-circle-euclidean :: complex-homo ⇒ real ⇒ euclidean-circle where
  poincare-circle-euclidean z r =
      (let R = (cosh r − 1) / 2;
           z' = to-complex z;
           cz = 1 − (cmod z')^2;
           k = cz * R + 1
        in (z' / k, cz * sqrt(R * (R + 1)) / k))
```

That Euclidean circle has a positive radius and is always fully within the disc.

```
lemma poincare-circle-in-disc:
  assumes r > 0 and z ∈ unit-disc and (ze, re) = poincare-circle-euclidean z r
  shows cmod ze < 1 re > 0 ∀ x ∈ circle ze re. cmod x < 1
proof−
  let ?R = (cosh r − 1) / 2
  let ?z' = to-complex z
  let ?cz = 1 − (cmod ?z')^2
  let ?k = ?cz * ?R + 1
  let ?ze = ?z' / ?k
  let ?re = ?cz * sqrt(?R * (?R + 1)) / ?k
```

from ‹$z \in$ *unit-disc*›
obtain z' **where** z': $z = \textit{of-complex } z'$
  using *inf-or-of-complex*[*of z*]
  by *auto*

hence $z' = ?z'$
  by *simp*

obtain *cz* **where** *cz*: $cz = (1 - (cmod\ z')^2)$
  by *auto*

have $cz > 0$ $cz \leq 1$
  using ‹$z \in$ *unit-disc*› z' *cz*
  using *unit-disc-cmod-square-lt-1*
  by *fastforce*+

obtain *R* **where** *R*: $R = ?R$
  by *blast*

have $R > 0$
  using *cosh-gt-1*[*of r*] ‹$r > 0$›
  by (*subst R*) *simp*

obtain *k* **where** *k*: $k = cz * R + 1$
  by *auto*

have $k > 1$
  using *k* ‹$R > 0$› ‹$cz > 0$›
  by *simp*

hence $cmod\ k = k$
  by *simp*

let $?RR = cz * sqrt(R * (R + 1)) / k$

have $cmod\ z' + cz * sqrt(R * (R + 1)) < k$
proof−
  have $((R+1)-R)^2 > 0$
    by *simp*
  hence $(R+1)^2 - 2*R*(R+1) + R^2 > 0$
    unfolding *power2-diff*
    by (*simp add*: *field-simps*)
  hence $(R+1)^2 + 2*R*(R+1) + R^2 - 4*R*(R+1) > 0$
    by *simp*
  hence $(2*R+1)^2 / 4 > R*(R+1)$
    using *power2-sum*[*of R+1 R*]
    by (*simp add*: *field-simps*)
  hence $sqrt(R*(R+1)) < (2*R+1) / 2$
    using ‹$R > 0$›
    by (*smt arith-geo-mean-sqrt power-divide real-sqrt-four real-sqrt-pow2 zero-le-mult-iff*)
  hence $sqrt(R*(R+1)) - R < 1/2$
    by (*simp add*: *field-simps*)
  hence $(1 + (cmod\ z')) * (sqrt(R*(R+1)) - R) < (1 + (cmod\ z')) * (1 / 2)$
    by (*subst mult-strict-left-mono*, *simp*, *smt norm-not-less-zero*, *simp*)
  also have ... < 1
    using ‹$z \in$ *unit-disc*› z'
    by *auto*
  finally have $(1 - cmod\ z') * ((1 + cmod\ z') * (sqrt(R*(R+1)) - R)) < (1 - cmod\ z') * 1$
    using ‹$z \in$ *unit-disc*› z'
    by (*subst mult-strict-left-mono*, *simp-all*)
  hence $cz * (sqrt\ (R*(R+1)) - R) < 1 - cmod\ z'$
    using *square-diff-square-factored*[*of 1 cmod z'*]
    by (*subst cz*, *subst* (*asm*) *mult.assoc*[*symmetric*], *simp add*: *power2-eq-square field-simps*)
  hence $cmod\ z' + cz * sqrt(R*(R+1)) < 1 + R * cz$
    by (*simp add*: *field-simps*)

thus *?thesis*
  using *k*
  by (*simp add*: *field-simps*)
qed
hence *cmod z′ / k + cz ∗ sqrt*(*R ∗* (*R + 1*)) / *k < 1*
  using ‹*k > 1*›
  unfolding *add-divide-distrib*[*symmetric*]
  by *simp*
hence *cmod* (*z′ / k*) + *cz ∗ sqrt*(*R ∗* (*R + 1*)) / *k < 1*
  using ‹*k > 1*›
  by *simp*
hence *cmod ?ze + ?re < 1*
  using *k cz* ‹*R = ?R*› *z′*
  by *simp*

moreover

have *cz ∗ sqrt*(*R ∗* (*R + 1*)) / *k > 0*
  using ‹*cz > 0*› ‹*R > 0*› ‹*k > 1*›
  by *auto*
hence *?re > 0*
  using *k cz* ‹*R = ?R*› *z′*
  by *simp*

moreover

have *cmod ?ze < 1*
  using ‹*cmod ?ze + ?re < 1*› ‹*?re > 0*›
  by *simp*

moreover

have *ze = ?ze re = ?re*
  using ‹(*ze*, *re*) = *poincare-circle-euclidean z r*›
  unfolding *poincare-circle-euclidean-def Let-def*
  by *simp-all*

moreover

have ∀ *x ∈ circle ze re*. *cmod x ≤ cmod ze + re*
  using *norm-triangle-ineq2*[*of - ze*]
  unfolding *circle-def*
  by (*smt mem-Collect-eq*)

ultimately

show *cmod ze < 1 re > 0* ∀ *x ∈ circle ze re*. *cmod x < 1*
  by *auto*
qed

The connection between the points on the h-circle and its corresponding Euclidean circle.

lemma *poincare-circle-is-euclidean-circle*:
  assumes *z ∈ unit-disc* **and** *r > 0*
  shows *let* (*Ze*, *Re*) = *poincare-circle-euclidean z r*
      in of-complex ‘ (*circle Ze Re*) = *poincare-circle z r*
proof−
  {
    fix *x*
    let *?z = to-complex z*

    from *assms* **obtain** *z′* **where** *z′*: *z = of-complex z′ cmod z′ < 1*
      using *inf-or-of-complex*[*of z*]
      by *auto*

    have ∗: ⋀ *x*. *cmod x < 1* ⟹ *1* − (*cmod x*)^2 > *0*
     by (*metis less-iff-diff-less-0 minus-diff-eq mult.left-neutral neg-less-0-iff-less norm-mult-less norm-power power2-eq-square*)

let $?R = (cosh\ r - 1) / 2$
obtain R **where** R: $R = ?R$
  by *blast*

let $?cx = 1 - (cmod\ x)^2$ **and** $?cz = 1 - (cmod\ z')^2$ **and** $?czx = (cmod\ (z' - x))^2$

let $?k = 1 + R * ?cz$
obtain k **where** k: $k = ?k$
  by *blast*
have $R > 0$
  using *R cosh-gt-1*[*OF* ‹$r > 0$›]
  by *simp*

hence $k > 1$
  using *assms* z' k $*$[*of* z']
  by *auto*
hence $**$: $cor\ k \neq 0$
  by (*smt of-real-eq-0-iff*)

have $of\text{-}complex\ x \in poincare\text{-}circle\ z\ r \longleftrightarrow cmod\ x < 1 \wedge poincare\text{-}distance\ z\ (of\text{-}complex\ x) = r$
  unfolding *poincare-circle-def*
  by *auto*
also have ... $\longleftrightarrow cmod\ x < 1 \wedge poincare\text{-}distance\text{-}formula'\ ?z\ x = cosh\ r$
  using *poincare-distance-formula*[*of z of-complex x*] *cosh-dist*[*of z of-complex x*]
  unfolding *poincare-distance-formula-def*
  using *assms*
  using *arcosh-cosh-real*
  by *auto*
also have ... $\longleftrightarrow cmod\ x < 1 \wedge ?czx / (?cz * ?cx) = ?R$
  using z'
  by (*simp add*: *field-simps*)
also have ... $\longleftrightarrow cmod\ x < 1 \wedge ?czx = ?R * ?cx * ?cz$
  using *assms* z' $*$[*of* z'] $*$[*of* x]
  using *nonzero-divide-eq-eq*[*of* $(1 - (cmod\ x)^2) * (1 - (cmod\ z')^2)$ $(cmod\ (z' - x))^2$ $?R$]
  by (*auto*, *simp add*: *field-simps*)
also have ... $\longleftrightarrow cmod\ x < 1 \wedge (z' - x) * (cnj\ z' - cnj\ x) = R * ?cz * (1 - x * cnj\ x)$ (**is** - $\longleftrightarrow$ - $\wedge$ $?l = ?r$)
proof−
  let $?l = (z' - x) * (cnj\ z' - cnj\ x)$ **and** $?r = R * (1 - Re\ (z' * cnj\ z')) * (1 - x * cnj\ x)$
  have *is-real* $?l$
    using *eq-cnj-iff-real*[*of* $?l$]
    by *simp*
  moreover
  have *is-real* $?r$
    using *eq-cnj-iff-real*[*of* $1 - x * cnj\ x$]
    using *Im-complex-of-real*[*of* $R * (1 - Re\ (z' * cnj\ z'))$]
    by *simp*
  ultimately
  show *?thesis*
    apply (*subst R*[*symmetric*])
    apply (*subst cmod-square*)+
    apply (*subst complex-eq-if-Re-eq*, *simp-all add*: *field-simps*)
    done
qed
also have ... $\longleftrightarrow cmod\ x < 1 \wedge z' * cnj\ z' - x * cnj\ z' - cnj\ x * z' + x * cnj\ x = R * ?cz - R * ?cz * x * cnj\ x$
  unfolding *right-diff-distrib left-diff-distrib*
  by (*simp add*: *field-simps*)
also have ... $\longleftrightarrow cmod\ x < 1 \wedge k * (x * cnj\ x) - x * cnj\ z' - cnj\ x * z' + z' * cnj\ z' = R * ?cz$ (**is** - $\longleftrightarrow$ - $\wedge$ $?lhs = ?rhs$)
  by (*subst k*) (*auto simp add*: *field-simps*)
also have ... $\longleftrightarrow cmod\ x < 1 \wedge (k * x * cnj\ x - x * cnj\ z' - cnj\ x * z' + z' * cnj\ z') / k = (R * ?cz) / k$
  using $**$
  by (*auto simp add*: *Groups.mult-ac*(*1*))
also have ... $\longleftrightarrow cmod\ x < 1 \wedge x * cnj\ x - x * cnj\ z' / k - cnj\ x * z' / k + z' * cnj\ z' / k = (R * ?cz) / k$
  using $**$

unfolding *add-divide-distrib diff-divide-distrib*
by *auto*
also have ... $\longleftrightarrow$ *cmod* $x < 1 \wedge (x - z'/k) * cnj(x - z'/k) = (R * ?cz) / k + (z' / k) * cnj(z' / k) - z' * cnj\ z' / k$
by (*auto simp add*: *field-simps diff-divide-distrib*)
also have ... $\longleftrightarrow$ *cmod* $x < 1 \wedge (cmod\ (x - z'/k))^2 = (R * ?cz) / k + (cmod\ z')^2 / k^2 - (cmod\ z')^2 / k$
apply (*subst complex-mult-cnj-cmod*)+
apply (*subst complex-eq-if-Re-eq*)
apply (*simp-all add*: *power-divide*)
done
also have ... $\longleftrightarrow$ *cmod* $x < 1 \wedge (cmod\ (x - z'/k))^2 = (R * ?cz * k + (cmod\ z')^2 - (cmod\ z')^2 * k) / k^2$
using $**$
unfolding *add-divide-distrib diff-divide-distrib*
by (*simp add*: *power2-eq-square*)
also have ... $\longleftrightarrow$ *cmod* $x < 1 \wedge (cmod\ (x - z'/k))^2 = ?cz^2 * R * (R + 1) / k^2$ (**is** - $\longleftrightarrow$ - $\wedge$ $?a^2 = ?b$)
proof−
have $*$: $R * (1 - (cmod\ z')^2) * k + (cmod\ z')^2 - (cmod\ z')^2 * k = (1 - (cmod\ z')^2)^2 * R * (R + 1)$
by (*subst k*)+ (*simp add*: *field-simps power2-diff*)
thus *?thesis*
by (*subst* $*$, *simp*)
qed
also have ... $\longleftrightarrow$ *cmod* $x < 1 \wedge cmod\ (x - z'/k) = ?cz * sqrt\ (R * (R + 1)) / k$
using ‹$R > 0$› $*$[*of* z'] $**$ ‹$k > 1$› ‹$z \in$ *unit-disc*› z'
using *real-sqrt-unique*[*of ?a ?b*, *symmetric*]
by (*auto simp add*: *real-sqrt-divide real-sqrt-mult power-divide power-mult-distrib*)
finally
have *of-complex* $x \in$ *poincare-circle* $z\ r \longleftrightarrow$ *cmod* $x < 1 \wedge x \in$ *circle* (z'/k) $(?cz * sqrt(R * (R{+}1)) / k)$
unfolding *circle-def* z' k R
by *simp*
hence *of-complex* $x \in$ *poincare-circle* $z\ r \longleftrightarrow$ (*let* (*Ze*, *Re*) = *poincare-circle-euclidean* $z\ r$ *in cmod* $x < 1 \wedge x \in$ *circle Ze Re*)
unfolding *poincare-circle-euclidean-def Let-def circle-def*
using z' R k
by (*simp add*: *field-simps*)
hence *of-complex* $x \in$ *poincare-circle* $z\ r \longleftrightarrow$ (*let* (*Ze*, *Re*) = *poincare-circle-euclidean* $z\ r$ *in* $x \in$ *circle Ze Re*)
using *poincare-circle-in-disc*[*OF* ‹$r > 0$› ‹$z \in$ *unit-disc*›]
by *auto*
} note $*$ = *this*
show *?thesis*
unfolding *Let-def*
proof *safe*
fix *Ze Re x*
assume *poincare-circle-euclidean* $z\ r$ = (*Ze*, *Re*) $x \in$ *circle Ze Re*
thus *of-complex* $x \in$ *poincare-circle* $z\ r$
using $*$[*of* x]
by *simp*
next
fix *Ze Re x*
assume $**$: *poincare-circle-euclidean* $z\ r$ = (*Ze*, *Re*) $x \in$ *poincare-circle* $z\ r$
then obtain x' where x': x = *of-complex* x'
unfolding *poincare-circle-def*
using *inf-or-of-complex*[*of* x]
by *auto*
hence $x' \in$ *circle Ze Re*
using $*$[*of* x'] $**$
by *simp*
thus $x \in$ *of-complex* ‘ *circle Ze Re*
using x'
by *auto*
qed
qed

6.1 Intersection of circles in special positions

Two h-circles centered at the x-axis intersect at mutually conjugate points

lemma *intersect-poincare-circles-x-axis*:

```
  assumes z: is-real z1 and is-real z2 and r1 > 0 and r2 > 0 and
           −1 < Re z1 and Re z1 < 1 and −1 < Re z2 and Re z2 < 1 and
           z1 ≠ z2
  assumes x1: x1 ∈ poincare-circle (of-complex z1) r1 ∩ poincare-circle (of-complex z2) r2 and
         x2: x2 ∈ poincare-circle (of-complex z1) r1 ∩ poincare-circle (of-complex z2) r2 and
            x1 ≠ x2
  shows x1 = conjugate x2
proof−
  have in-disc: of-complex z1 ∈ unit-disc of-complex z2 ∈ unit-disc
    using assms
    by (auto simp add: cmod-eq-Re)

  obtain x1' x2' where x': x1 = of-complex x1' x2 = of-complex x2'
    using x1 x2
    using inf-or-of-complex[of x1] inf-or-of-complex[of x2]
    unfolding poincare-circle-def
    by auto

  obtain Ze1 Re1 where 1: (Ze1, Re1) = poincare-circle-euclidean (of-complex z1) r1
    by (metis poincare-circle-euclidean-def)
  obtain Ze2 Re2 where 2: (Ze2, Re2) = poincare-circle-euclidean (of-complex z2) r2
    by (metis poincare-circle-euclidean-def)
  have circle: x1' ∈ circle Ze1 Re1 ∩ circle Ze2 Re2  x2' ∈ circle Ze1 Re1 ∩ circle Ze2 Re2
    using poincare-circle-is-euclidean-circle[of of-complex z1 r1]
    using poincare-circle-is-euclidean-circle[of of-complex z2 r2]
    using assms 1 2 ‹of-complex z1 ∈ unit-disc› ‹of-complex z2 ∈ unit-disc› x'
    by auto (metis image-iff of-complex-inj)+

  have is-real Ze1 is-real Ze2
    using 1 2 ‹is-real z1› ‹is-real z2›
    by (simp-all add: poincare-circle-euclidean-def Let-def)

  have Re1 > 0 Re2 > 0
    using 1 2 in-disc ‹r1 > 0› ‹r2 > 0›
    using poincare-circle-in-disc(2)[of r1 of-complex z1 Ze1 Re1]
    using poincare-circle-in-disc(2)[of r2 of-complex z2 Ze2 Re2]
    by auto

  have Ze1 ≠ Ze2
  proof (rule ccontr)
    assume ¬ ?thesis
    hence eq: Ze1 = Ze2 Re1 = Re2
      using circle(1)
      unfolding circle-def
      by auto

    let ?A = Ze1 − Re1 and ?B = Ze1 + Re1
    have ?A ∈ circle Ze1 Re1 ?B ∈ circle Ze1 Re1
      using ‹Re1 > 0›
      unfolding circle-def
      by simp-all
    hence of-complex ?A ∈ poincare-circle (of-complex z1) r1 of-complex ?B ∈ poincare-circle (of-complex z1) r1
         of-complex ?A ∈ poincare-circle (of-complex z2) r2 of-complex ?B ∈ poincare-circle (of-complex z2) r2
      using eq
      using poincare-circle-is-euclidean-circle[OF ‹of-complex z1 ∈ unit-disc› ‹r1 > 0›]
      using poincare-circle-is-euclidean-circle[OF ‹of-complex z2 ∈ unit-disc› ‹r2 > 0›]
      using 1 2
      by auto blast+
    hence poincare-distance (of-complex z1) (of-complex ?A) = poincare-distance (of-complex z1) (of-complex ?B)
         poincare-distance (of-complex z2) (of-complex ?A) = poincare-distance (of-complex z2) (of-complex ?B)
         −1 < Re (Ze1 − Re1) Re (Ze1 − Re1) < 1 −1 < Re (Ze1 + Re1) Re (Ze1 + Re1) < 1
      using ‹is-real Ze1› ‹is-real Ze2›
      unfolding poincare-circle-def
      by (auto simp add: cmod-eq-Re)
    hence z1 = z2
      using unique-midpoint-x-axis[of Ze1 − Re1 Ze1 + Re1]
```

using ‹*is-real Ze1*› ‹*is-real z1*› ‹*is-real z2*› ‹$Re1 > 0$› ‹$-1 < Re\ z1$› ‹$Re\ z1 < 1$› ‹$-1 < Re\ z2$› ‹$Re\ z2 < 1$›
by *auto*
thus *False*
using ‹$z1 \neq z2$›
by *simp*
qed

hence $*$: $(Re\ x1')^2 + (Im\ x1')^2 - 2 * Re\ x1' * Ze1 + Ze1 * Ze1 - cor\ (Re1 * Re1) = 0$
$(Re\ x1')^2 + (Im\ x1')^2 - 2 * Re\ x1' * Ze2 + Ze2 * Ze2 - cor\ (Re2 * Re2) = 0$
$(Re\ x2')^2 + (Im\ x2')^2 - 2 * Re\ x2' * Ze1 + Ze1 * Ze1 - cor\ (Re1 * Re1) = 0$
$(Re\ x2')^2 + (Im\ x2')^2 - 2 * Re\ x2' * Ze2 + Ze2 * Ze2 - cor\ (Re2 * Re2) = 0$
using *circle-equation*[*of Re1 Ze1*] *circle-equation*[*of Re2 Ze2*] *circle*
using *eq-cnj-iff-real*[*of Ze1*] ‹*is-real Ze1*› ‹$Re1 > 0$›
using *eq-cnj-iff-real*[*of Ze2*] ‹*is-real Ze2*› ‹$Re2 > 0$›
using *complex-add-cnj*[*of x1'*] *complex-add-cnj*[*of x2'*]
using *distrib-left*[*of Ze1 x1' cnj x1'*] *distrib-left*[*of Ze2 x1' cnj x1'*]
using *distrib-left*[*of Ze1 x2' cnj x2'*] *distrib-left*[*of Ze2 x2' cnj x2'*]
by (*auto simp add*: *complex-mult-cnj power2-eq-square field-simps*)

hence $- 2 * Re\ x1' * Ze1 + Ze1 * Ze1 - cor\ (Re1 * Re1) = - 2 * Re\ x1' * Ze2 + Ze2 * Ze2 - cor\ (Re2 * Re2)$
$- 2 * Re\ x2' * Ze1 + Ze1 * Ze1 - cor\ (Re1 * Re1) = - 2 * Re\ x2' * Ze2 + Ze2 * Ze2 - cor\ (Re2 * Re2)$
by (*smt add-diff-cancel-right' add-diff-eq eq-iff-diff-eq-0 minus-diff-eq mult-minus-left of-real-minus*)+
hence $2 * Re\ x1' * (Ze2 - Ze1) = (Ze2 * Ze2 - cor\ (Re2 * Re2)) - (Ze1 * Ze1 - cor\ (Re1 * Re1))$
$2 * Re\ x2' * (Ze2 - Ze1) = (Ze2 * Ze2 - cor\ (Re2 * Re2)) - (Ze1 * Ze1 - cor\ (Re1 * Re1))$
by *simp-all* (*simp add*: *field-simps*)+
hence $2 * Re\ x1' * (Ze2 - Ze1) = 2 * Re\ x2' * (Ze2 - Ze1)$
by *simp*
hence $Re\ x1' = Re\ x2'$
using ‹$Ze1 \neq Ze2$›
by *simp*
moreover
hence $(Im\ x1')^2 = (Im\ x2')^2$
using $*(1)$ $*(3)$
by (*simp add*: ‹*is-real Ze1*› *complex-eq-if-Re-eq*)
hence $Im\ x1' = Im\ x2' \lor Im\ x1' = -Im\ x2'$
using *power2-eq-iff*
by *blast*
ultimately
show *?thesis*
using *x'* ‹$x1 \neq x2$›
using *complex.expand*
by (*metis cnj.code complex-surj conjugate-of-complex*)
qed

Two h-circles of the same radius centered at mutually conjugate points intersect at the x-axis

lemma *intersect-poincare-circles-conjugate-centers*:
assumes *in-disc*: $z1 \in$ *unit-disc* $z2 \in$ *unit-disc* **and**
$z1 \neq z2$ **and** $z1 =$ *conjugate* $z2$ **and** $r > 0$ **and**
u: $u \in$ *poincare-circle z1 r* $\cap$ *poincare-circle z2 r*
shows *is-real* (*to-complex u*)
proof−
obtain *z1e r1e z2e r2e* **where**
euclidean: (*z1e*, *r1e*) = *poincare-circle-euclidean z1 r*
(*z2e*, *r2e*) = *poincare-circle-euclidean z2 r*
by (*metis poincare-circle-euclidean-def*)
obtain *z1' z2'* **where** *z'*: *z1* = *of-complex z1' z2* = *of-complex z2'*
using *inf-or-of-complex*[*of z1*] *inf-or-of-complex*[*of z2*] *in-disc*
by *auto*
obtain *u'* **where** *u'*: *u* = *of-complex u'*
using *u inf-or-of-complex*[*of u*]
by (*auto simp add*: *poincare-circle-def*)
have *z1'* = *cnj z2'*
using ‹*z1* = *conjugate z2*› *z'*
by (*auto simp add*: *of-complex-inj*)
moreover
let $?cz = 1 - (cmod\ z2')^2$

let *?den* = *?cz* ∗ (*cosh r* − *1*) / *2* + *1*
have *?cz* > *0*
using *in-disc z′*
by (*simp add*: *cmod-def*)
hence *?den* ≥ *1*
using *cosh-gt-1*[*OF* ‹*r* > *0*›]
by *auto*
hence *?den* ≠ *0*
by *simp*
hence *cor ?den* ≠ *0*
using *of-real-eq-0-iff*
by *blast*
ultimately
have *r1e* = *r2e z1e* = *cnj z2e z1e* ≠ *z2e*
using *z′ euclidean* ‹*z1* ≠ *z2*›
unfolding *poincare-circle-euclidean-def Let-def*
by *simp-all metis*

hence *u′* ∈ *circle* (*cnj z2e*) *r2e* ∩ *circle z2e r2e z2e* ≠ *cnj z2e*
using *euclidean u u′*
using *poincare-circle-is-euclidean-circle*[*of z1 r*]
using *poincare-circle-is-euclidean-circle*[*of z2 r*]
using *in-disc* ‹*r* > *0*›
by *auto* (*metis image-iff of-complex-inj*)+
hence $(cmod\ (u' - z2e))^2 = (cmod(u' - cnj\ z2e))^2$
by (*simp add*: *circle-def*)
hence (*u′* − *z2e*) ∗ (*cnj u′* − *cnj z2e*) = (*u′* − *cnj z2e*) ∗ (*cnj u′* − *z2e*)
by (*metis complex-cnj-cnj complex-cnj-diff complex-norm-square*)
hence (*z2e* − *cnj z2e*) ∗ (*u′* − *cnj u′*) = *0*
by (*simp add*: *field-simps*)
thus *?thesis*
using *u′* ‹*z2e* ≠ *cnj z2e*› *eq-cnj-iff-real*[*of u′*]
by *simp*
qed

6.2 Congruent triangles

For every pair of triangles such that its three pairs of sides are pairwise equal there is an h-isometry (a unit disc preserving Möbius transform, eventually composed with a conjugation) that maps one triangle onto the other.

lemma *unit-disc-fix-f-congruent-triangles*:
assumes
in-disc: *u* ∈ *unit-disc v* ∈ *unit-disc w* ∈ *unit-disc* **and**
in-disc′: *u′* ∈ *unit-disc v′* ∈ *unit-disc w′* ∈ *unit-disc* **and**
d: *poincare-distance u v* = *poincare-distance u′ v′*
poincare-distance v w = *poincare-distance v′ w′*
poincare-distance u w = *poincare-distance u′ w′*
shows
∃ *M*. *unit-disc-fix-f M* ∧ *M u* = *u′* ∧ *M v* = *v′* ∧ *M w* = *w′*
proof (*cases u* = *v* ∨ *u* = *w* ∨ *v* = *w*)
case *True*
thus *?thesis*
using *assms*
using *poincare-distance-eq-0-iff*[*of u′ v′*]
using *poincare-distance-eq-0-iff*[*of v′ w′*]
using *poincare-distance-eq-0-iff*[*of u′ w′*]
using *poincare-distance-eq-ex-moebius*[*of v w v′ w′*]
using *poincare-distance-eq-ex-moebius*[*of u w u′ w′*]
using *poincare-distance-eq-ex-moebius*[*of u v u′ v′*]
by (*metis unit-disc-fix-f-def*)
next
case *False*

have ∀ *w u′ v′ w′*. *w* ∈ *unit-disc* ∧ *u′* ∈ *unit-disc* ∧ *v′* ∈ *unit-disc* ∧ *w′* ∈ *unit-disc* ∧ *w* ≠ *u* ∧ *w* ≠ *v* ∧
poincare-distance u v = *poincare-distance u′ v′* ∧
poincare-distance v w = *poincare-distance v′ w′* ∧

poincare-distance u w = *poincare-distance* u' w' $\longrightarrow$
($\exists$ M. *unit-disc-fix-f* M $\wedge$ M u = u' $\wedge$ M v = v' $\wedge$ M w = w') (**is** *?P u v*)
proof (*rule wlog-positive-x-axis*[**where** *P*=*?P*])
show $v \in$ *unit-disc* $u \in$ *unit-disc*
by *fact+*
next
show $u \neq v$
using *False*
by *simp*
next
fix x
assume x: *is-real* x $0 <$ *Re* x *Re* $x < 1$

hence *of-complex* $x \neq 0_h$
using *of-complex-zero-iff*[*of* x]
by (*auto simp add*: *complex.expand*)

show *?P* 0_h (*of-complex* x)
proof *safe*
fix w u' v' w'
assume *in-disc*: $w \in$ *unit-disc* $u' \in$ *unit-disc* $v' \in$ *unit-disc* $w' \in$ *unit-disc*
assume *poincare-distance* 0_h (*of-complex* x) = *poincare-distance* u' v'
then obtain M' **where** M': *unit-disc-fix* M' *moebius-pt* M' u' = 0_h *moebius-pt* M' v' = (*of-complex* x)
using *poincare-distance-eq-ex-moebius*[*of* u' v' 0_h *of-complex* x] *in-disc* x
by (*auto simp add*: *cmod-eq-Re*)

let *?w* = *moebius-pt* M' w'
have *?w* $\in$ *unit-disc*
using ‹*unit-disc-fix* M'› ‹$w' \in$ *unit-disc*›
by *simp*

assume $w \neq 0_h$ $w \neq$ *of-complex* x
hence *dist-gt-0*: *poincare-distance* 0_h $w > 0$ *poincare-distance* (*of-complex* x) $w > 0$
using *poincare-distance-eq-0-iff*[*of* 0_h w] *in-disc poincare-distance-ge0*[*of* 0_h w]
using *poincare-distance-eq-0-iff*[*of of-complex* x w] *in-disc poincare-distance-ge0*[*of of-complex* x w]
using x
by (*simp-all add*: *cmod-eq-Re*)

assume *poincare-distance* (*of-complex* x) w = *poincare-distance* v' w'
poincare-distance 0_h w = *poincare-distance* u' w'
hence *poincare-distance* 0_h *?w* = *poincare-distance* 0_h w
poincare-distance (*of-complex* x) *?w* = *poincare-distance* (*of-complex* x) w
using $M'(1)$ $M'(2)$[*symmetric*] $M'(3)$[*symmetric*] *in-disc*
using *unit-disc-fix-preserve-poincare-distance*[*of* M' u' w']
using *unit-disc-fix-preserve-poincare-distance*[*of* M' v' w']
by *simp-all*
hence *?w* $\in$ *poincare-circle* 0_h (*poincare-distance* 0_h w) $\cap$ *poincare-circle* (*of-complex* x) (*poincare-distance* (*of-complex* x) w)
$w \in$ *poincare-circle* 0_h (*poincare-distance* 0_h w) $\cap$ *poincare-circle* (*of-complex* x) (*poincare-distance* (*of-complex* x) w)
using ‹*?w* $\in$ *unit-disc*› ‹$w \in$ *unit-disc*›
unfolding *poincare-circle-def*
by *simp-all*
hence *?w* = w $\vee$ *?w* = *conjugate* w
using *intersect-poincare-circles-x-axis*[*of* 0 x *poincare-distance* 0_h w *poincare-distance* (*of-complex* x) w *?w* w] x
using ‹*of-complex* $x \neq 0_h$› *dist-gt-0*
using *poincare-distance-eq-0-iff*
by *auto*
thus $\exists$ M. *unit-disc-fix-f* M $\wedge$ M 0_h = u' $\wedge$ M (*of-complex* x) = v' $\wedge$ M w = w'
proof
assume *moebius-pt* M' w' = w
thus *?thesis*
using M'
using *moebius-pt-invert*[*of* M' u' 0_h]
using *moebius-pt-invert*[*of* M' v' *of-complex* x]
using *moebius-pt-invert*[*of* M' w' w]

apply (*rule-tac x=moebius-pt* $(-M')$ **in** *exI*)
apply (*simp add*: *unit-disc-fix-f-def*)
apply (*rule-tac x*=$-M'$ **in** *exI*, *simp*)
done
next
let *?M* = *moebius-pt* $(-M')$ ∘ *conjugate*
assume *moebius-pt* M' w' = *conjugate w*
hence *?M w* = w'
using *moebius-pt-invert*[*of* M' w' *conjugate w*]
by *simp*
moreover
have *?M* 0_h = u' *?M* (*of-complex x*) = v'
using *moebius-pt-invert*[*of* M' u' 0_h]
using *moebius-pt-invert*[*of* M' v' *of-complex x*]
using M' ‹*is-real x*› *eq-cnj-iff-real*[*of x*]
by *simp-all*
moreover
have *unit-disc-fix-f ?M*
using ‹*unit-disc-fix* M'›
unfolding *unit-disc-fix-f-def*
by (*rule-tac x*=$-M'$ **in** *exI*, *simp*)
ultimately
show *?thesis*
by *blast*
qed
qed
next
fix *M u v*
assume *1*: *unit-disc-fix M u* ∈ *unit-disc v* ∈ *unit-disc*
let *?Mu* = *moebius-pt M u* **and** *?Mv* = *moebius-pt M v*
assume *2*: *?P ?Mu ?Mv*
show *?P u v*
proof *safe*
fix *w* u' v' w'
let *?Mw* = *moebius-pt M w* **and** *?Mu'* = *moebius-pt M* u' **and** *?Mv'* = *moebius-pt M* v' **and** *?Mw'* = *moebius-pt M* w'
assume *w* ∈ *unit-disc* u' ∈ *unit-disc* v' ∈ *unit-disc* w' ∈ *unit-disc* $w \neq u$ $w \neq v$
poincare-distance u v = *poincare-distance* u' v'
poincare-distance v w = *poincare-distance* v' w'
poincare-distance u w = *poincare-distance* u' w'
then obtain M' **where** M': *unit-disc-fix-f* M' M' *?Mu* = *?Mu'* M' *?Mv* = *?Mv'* M' *?Mw* = *?Mw'*
using *1 2*[*rule-format*, *of ?Mw ?Mu' ?Mv' ?Mw'*]
by *auto*

let *?M* = *moebius-pt* $(-M)$ ∘ M' ∘ *moebius-pt M*
show ∃ *M*. *unit-disc-fix-f M* ∧ *M u* = u' ∧ *M v* = v' ∧ *M w* = w'
proof (*rule-tac x=?M* **in** *exI*, *safe*)
show *unit-disc-fix-f ?M*
using $M'(1)$ ‹*unit-disc-fix M*›
by (*subst unit-disc-fix-f-comp*, *subst unit-disc-fix-f-comp*, *simp-all*)
next
show *?M u* = u' *?M v* = v' *?M w* = w'
using M'
by *auto*
qed
qed
qed
thus *?thesis*
using *assms False*
by *auto*
qed

end
theory *Poincare-Between*
imports *Poincare-Distance*
begin

7 H-betweenness in the Poincaré model

7.1 H-betwenness expressed by a cross-ratio

The point v is h-between u and w if the cross-ratio between the pairs u and w and v and inverse of v is real and negative.

definition *poincare-between* :: *complex-homo* ⇒ *complex-homo* ⇒ *complex-homo* ⇒ *bool* **where**
  poincare-between u v w ⟷
    u = *v* ∨ *v* = *w* ∨
    (*let cr* = *cross-ratio u v w* (*inversion v*)
     in is-real (*to-complex cr*) ∧ *Re* (*to-complex cr*) < *0*)

7.1.1 H-betwenness is preserved by h-isometries

Since they preserve cross-ratio and inversion, h-isometries (unit disc preserving Möbius transformations and conjugation) preserve h-betweeness.

lemma *unit-disc-fix-moebius-preserve-poincare-between* [*simp*]:
  assumes *unit-disc-fix M* **and** *u* ∈ *unit-disc* **and** *v* ∈ *unit-disc* **and** *w* ∈ *unit-disc*
  shows *poincare-between* (*moebius-pt M u*) (*moebius-pt M v*) (*moebius-pt M w*) ⟷
    poincare-between u v w
proof (*cases u* = *v* ∨ *v* = *w*)
  case *True*
  thus *?thesis*
    using *assms*
    unfolding *poincare-between-def*
    by *auto*
next
  case *False*
  moreover
  hence *moebius-pt M u* ≠ *moebius-pt M v* ∧ *moebius-pt M v* ≠ *moebius-pt M w*
    by *auto*
  moreover
  have *v* ≠ *inversion v w* ≠ *inversion v*
    using *inversion-noteq-unit-disc*[*of v w*]
    using *inversion-noteq-unit-disc*[*of v v*]
    using ‹*v* ∈ *unit-disc*› ‹*w* ∈ *unit-disc*›
    by *auto*
  ultimately
  show *?thesis*
    using *assms*
    using *unit-circle-fix-moebius-pt-inversion*[*of M v*, *symmetric*]
    unfolding *poincare-between-def*
    by (*simp del*: *unit-circle-fix-moebius-pt-inversion*)
qed

lemma *conjugate-preserve-poincare-between* [*simp*]:
  assumes *u* ∈ *unit-disc* **and** *v* ∈ *unit-disc* **and** *w* ∈ *unit-disc*
  shows *poincare-between* (*conjugate u*) (*conjugate v*) (*conjugate w*) ⟷
    poincare-between u v w
proof (*cases u* = *v* ∨ *v* = *w*)
  case *True*
  thus *?thesis*
    using *assms*
    unfolding *poincare-between-def*
    by *auto*
next
  case *False*
  moreover
  hence *conjugate u* ≠ *conjugate v* ∧ *conjugate v* ≠ *conjugate w*
    using *conjugate-inj* **by** *blast*
  moreover
  have *v* ≠ *inversion v w* ≠ *inversion v*
    using *inversion-noteq-unit-disc*[*of v w*]
    using *inversion-noteq-unit-disc*[*of v v*]
    using ‹*v* ∈ *unit-disc*› ‹*w* ∈ *unit-disc*›

by *auto*
ultimately
show *?thesis*
using *assms*
using *conjugate-cross-ratio*[*of v w inversion v u*]
unfolding *poincare-between-def*
by (*metis conjugate-id-iff conjugate-involution inversion-def inversion-sym o-apply*)
qed

7.1.2 Some elementary properties of h-betwenness

lemma *poincare-between-nonstrict* [*simp*]:
shows *poincare-between u u v* **and** *poincare-between u v v*
by (*simp-all add*: *poincare-between-def*)

lemma *poincare-between-sandwich*:
assumes $u \in$ *unit-disc* **and** $v \in$ *unit-disc*
assumes *poincare-between u v u*
shows $u = v$
proof (*rule ccontr*)
assume ¬ *?thesis*
thus *False*
using *assms*
using *inversion-noteq-unit-disc*[*of v u*]
using *cross-ratio-1*[*of v u inversion v*]
unfolding *poincare-between-def Let-def*
by *auto*
qed

lemma *poincare-between-rev*:
assumes $u \in$ *unit-disc* **and** $v \in$ *unit-disc* **and** $w \in$ *unit-disc*
shows *poincare-between u v w* $\longleftrightarrow$ *poincare-between w v u*
using *assms*
using *inversion-noteq-unit-disc*[*of v w*]
using *inversion-noteq-unit-disc*[*of v u*]
using *cross-ratio-commute-13*[*of u v w inversion v*]
using *cross-ratio-not-inf*[*of w inversion v v u*]
using *cross-ratio-not-zero*[*of w v u inversion v*]
using *inf-or-of-complex*[*of cross-ratio w v u* (*inversion v*)]
unfolding *poincare-between-def*
by (*auto simp add*: *Let-def Im-complex-div-eq-0 Re-divide divide-less-0-iff*)

7.1.3 H-betwenness and h-collinearity

Three points can be in an h-between relation only when they are h-collinear.

lemma *poincare-between-poincare-collinear* [*simp*]:
assumes *in-disc*: $u \in$ *unit-disc* $v \in$ *unit-disc* $w \in$ *unit-disc*
assumes *betw*: *poincare-between u v w*
shows *poincare-collinear* {*u*, *v*, *w*}
proof (*cases* $u = v \lor v = w$)
case *True*
thus *?thesis*
using *assms*
by *auto*
next
case *False*
hence *distinct*: *distinct* [*u*, *v*, *w*, *inversion v*]
using *in-disc inversion-noteq-unit-disc*[*of v v*] *inversion-noteq-unit-disc*[*of v u*] *inversion-noteq-unit-disc*[*of v w*]
using *betw poincare-between-sandwich*[*of w v*]
by (*auto simp add*: *poincare-between-def Let-def*)

then obtain *H* **where** ∗: {*u*, *v*, *w*, *inversion v*} ⊆ *circline-set H*
using *assms*
unfolding *poincare-between-def*
using *four-points-on-circline-iff-cross-ratio-real*[*of u v w inversion v*]
by *auto*

hence H = *poincare-line u v*
using *assms distinct*
using *unique-circline-set*[*of u v inversion v*]
using *poincare-line*[*of u v*] *poincare-line-inversion*[*of u v*]
unfolding *circline-set-def*
by *auto*
thus *?thesis*
using $*$ *assms False*
unfolding *poincare-collinear-def*
by (*rule-tac x=poincare-line u v* **in** *exI*) *simp*
qed

lemma *poincare-between-poincare-line-uvz*:
assumes $u \neq v$ **and** $u \in$ *unit-disc* **and** $v \in$ *unit-disc* **and**
$z \in$ *unit-disc* **and** *poincare-between u v z*
shows $z \in$ *circline-set* (*poincare-line u v*)
using *assms*
using *poincare-between-poincare-collinear*[*of u v z*]
using *unique-poincare-line*[*OF assms*(*1*−*3*)]
unfolding *poincare-collinear-def*
by *auto*

lemma *poincare-between-poincare-line-uzv*:
assumes $u \neq v$ **and** $u \in$ *unit-disc* **and** $v \in$ *unit-disc* **and**
$z \in$ *unit-disc poincare-between u z v*
shows $z \in$ *circline-set* (*poincare-line u v*)
using *assms*
using *poincare-between-poincare-collinear*[*of u z v*]
using *unique-poincare-line*[*OF assms*(*1*−*3*)]
unfolding *poincare-collinear-def*
by *auto*

7.1.4 H-betweeness on Euclidean segments

If the three points lie on an h-line that is a Euclidean line (e.g., if it contains zero), h-betweenness can be characterized much simpler than in the definition.

lemma *poincare-between-x-axis-u0v*:
assumes *is-real* u' **and** $u' \neq 0$ **and** $v' \neq 0$
shows *poincare-between* (*of-complex* u') 0_h (*of-complex* v') $\longleftrightarrow$ *is-real* $v' \wedge Re\ u' * Re\ v' < 0$
proof−
have $Re\ u' \neq 0$
using ‹*is-real* u'› ‹$u' \neq 0$›
using *complex-eq-if-Re-eq*
by *auto*
have *nz*: *of-complex* $u' \neq 0_h$ *of-complex* $v' \neq 0_h$
by (*simp-all add*: ‹$u' \neq 0$› ‹$v' \neq 0$›)
hence $0_h \neq$ *of-complex* v'
by *metis*

let *?cr* = *cross-ratio* (*of-complex* u') 0_h (*of-complex* v') ∞_h
have *is-real* (*to-complex ?cr*) $\wedge$ *Re* (*to-complex ?cr*) $< 0 \longleftrightarrow$ *is-real* $v' \wedge Re\ u' * Re\ v' < 0$
using *cross-ratio-0inf*[*of* v' u'] ‹$v' \neq 0$› ‹$u' \neq 0$› ‹*is-real* u'›
by (*metis Re-complex-div-lt-0 Re-mult-real complex-cnj-divide divide-cancel-left eq-cnj-iff-real to-complex-of-complex*)
thus *?thesis*
unfolding *poincare-between-def inversion-zero*
using ‹*of-complex* $u' \neq 0_h$› ‹$0_h \neq$ *of-complex* v'›
by *simp*
qed

lemma *poincare-between-u0v*:
assumes $u \in$ *unit-disc* **and** $v \in$ *unit-disc* **and** $u \neq 0_h$ **and** $v \neq 0_h$
shows *poincare-between u* 0_h *v* $\longleftrightarrow$ ($\exists\ k < 0$. *to-complex u* = *cor k* $*$ *to-complex v*) (**is** *?P u v*)
proof (*cases u* = *v*)
case *True*
thus *?thesis*
using *assms*

using *inf-or-of-complex*[*of v*]
using *poincare-between-sandwich*[*of u* 0_h]
by *auto*
next
case *False*
have $\forall$ *u*. *u* $\in$ *unit-disc* $\wedge$ *u* $\neq$ 0_h $\longrightarrow$ *?P u v* (**is** *?P′ v*)
proof (*rule wlog-rotation-to-positive-x-axis*)
fix φ *v*
let *?M* = *moebius-pt* (*moebius-rotation* φ)
assume *1*: *v* $\in$ *unit-disc v* $\neq$ 0_h
assume *2*: *?P′* (*?M v*)
show *?P′ v*
proof (*rule allI*, *rule impI*, (*erule conjE*)+)
fix *u*
assume *3*: *u* $\in$ *unit-disc u* $\neq$ 0_h
have *poincare-between* (*?M u*) 0_h (*?M v*) $\longleftrightarrow$ *poincare-between u* 0_h *v*
using ‹*u* $\in$ *unit-disc*› ‹*v* $\in$ *unit-disc*›
using *unit-disc-fix-moebius-preserve-poincare-between unit-disc-fix-rotation zero-in-unit-disc*
by *fastforce*
thus *?P u v*
using *1 2*[*rule-format*, *of ?M u*] *3*
using *inf-or-of-complex*[*of u*] *inf-or-of-complex*[*of v*]
by *auto*
qed
next
fix *x*
assume *1*: *is-real x 0* < *Re x Re x* < *1*
hence *x* $\neq$ *0*
by *auto*
show *?P′* (*of-complex x*)
proof (*rule allI*, *rule impI*, (*erule conjE*)+)
fix *u*
assume *2*: *u* $\in$ *unit-disc u* $\neq$ 0_h
then obtain *u′* **where** *u* = *of-complex u′*
using *inf-or-of-complex*[*of u*]
by *auto*
show *?P u* (*of-complex x*)
using *1 2* ‹*x* $\neq$ *0*› ‹*u* = *of-complex u′*›
using *poincare-between-rev*[*of u* 0_h *of-complex x*]
using *poincare-between-x-axis-u0v*[*of x u′*] ‹*is-real x*›
apply (*auto simp add*: *cmod-eq-Re*)
apply (*rule-tac x*=*Re u′* / *Re x* **in** *exI*, *simp add*: *divide-neg-pos algebra-split-simps*)
using *mult-neg-pos mult-pos-neg*
by *blast*
qed
qed *fact*+
thus *?thesis*
using *assms*
by *auto*
qed

lemma *poincare-between-u0v-polar-form*:
assumes *x* $\in$ *unit-disc* **and** *y* $\in$ *unit-disc* **and** *x* $\neq$ 0_h **and** *y* $\neq$ 0_h **and**
to-complex x = *cor rx* ∗ *cis* φ *to-complex y* = *cor ry* ∗ *cis* φ
shows *poincare-between x* 0_h *y* $\longleftrightarrow$ *rx* ∗ *ry* < *0* (**is** *?P x y rx ry*)
proof−
from *assms* **have** *rx* $\neq$ *0 ry* $\neq$ *0*
using *inf-or-of-complex*[*of x*] *inf-or-of-complex*[*of y*]
by *auto*

have ($\exists$ *k*<*0*. *cor rx* = *cor k* ∗ *cor ry*) = (*rx* ∗ *ry* < *0*)
proof
assume $\exists$ *k*<*0*. *cor rx* = *cor k* ∗ *cor ry*
then obtain *k* **where** *k* < *0 cor rx* = *cor k* ∗ *cor ry*
by *auto*
hence *rx* = *k* ∗ *ry*

```
      using of-real-eq-iff
      by fastforce
    thus rx ∗ ry < 0
      using ‹k < 0› ‹rx ≠ 0› ‹ry ≠ 0›
      by (smt divisors-zero mult-nonneg-nonpos mult-nonpos-nonpos zero-less-mult-pos2)
  next
    assume rx ∗ ry < 0
    hence rx = (rx/ry)∗ry rx / ry < 0
      using ‹rx ≠ 0› ‹ry ≠ 0›
      by (auto simp add: divide-less-0-iff algebra-split-simps)
    thus ∃ k<0. cor rx = cor k ∗ cor ry
      using ‹rx ≠ 0› ‹ry ≠ 0›
      by (rule-tac x=rx / ry in exI, simp)
  qed
  thus ?thesis
    using assms
    using poincare-between-u0v[OF assms(1−4)]
    by auto
qed

lemma poincare-between-x-axis-0uv:
  fixes x y :: real
  assumes −1 < x and x < 1 and x ≠ 0
  assumes −1 < y and y < 1 and y ≠ 0
  shows poincare-between 0h (of-complex x) (of-complex y) ⟷
        (x < 0 ∧ y < 0 ∧ y ≤ x) ∨ (x > 0 ∧ y > 0 ∧ x ≤ y) (is ?lhs ⟷ ?rhs)
proof (cases x = y)
  case True
  thus ?thesis
    using assms
    unfolding poincare-between-def
    by auto
next
  case False
  let ?x = of-complex x and ?y = of-complex y

  have ?x ∈ unit-disc ?y ∈ unit-disc
    using assms
    by auto

  have distinct: distinct [0h, ?x, ?y, inversion ?x]
    using ‹x ≠ 0› ‹y ≠ 0› ‹x ≠ y› ‹?x ∈ unit-disc› ‹?y ∈ unit-disc›
    using inversion-noteq-unit-disc[of ?x ?y]
    using inversion-noteq-unit-disc[of ?x ?x]
    using inversion-noteq-unit-disc[of ?x 0h]
    using of-complex-inj[of x y]
    by (metis distinct-length-2-or-more distinct-singleton of-complex-zero-iff of-real-eq-0-iff of-real-eq-iff zero-in-unit-disc)

  let ?cr = cross-ratio 0h ?x ?y (inversion ?x)
  have Re (to-complex ?cr) = x^2 ∗ (x∗y − 1) / (x ∗ (y − x))
    using ‹x ≠ 0› ‹x ≠ y›
    unfolding inversion-def
    by simp (transfer, transfer, auto simp add: vec-cnj-def power2-eq-square field-simps split: if-split-asm)
  moreover
  {
    fix a b :: real
    assume b ≠ 0
    hence a < 0 ⟷ b^2 ∗ a < (0::real)
     by (metis mult.commute mult-eq-0-iff mult-neg-pos mult-pos-pos not-less-iff-gr-or-eq not-real-square-gt-zero power2-eq-square)
  }
  hence x^2 ∗ (x∗y − 1) < 0
    using assms
    by (smt minus-mult-minus mult-le-cancel-left1)
  moreover
  have x ∗ (y − x) > 0 ⟷ ?rhs
    using ‹x ≠ 0› ‹y ≠ 0› ‹x ≠ y›
```

by (*smt mult-le-0-iff*)
ultimately
have ∗: *Re* (*to-complex ?cr*) < *0* ⟷ *?rhs*
by (*simp add*: *divide-less-0-iff*)

show *?thesis*
proof
assume *?lhs*
have *is-real* (*to-complex ?cr*) *Re* (*to-complex ?cr*) < *0*
using ‹*?lhs*› *distinct*
unfolding *poincare-between-def Let-def*
by *auto*
thus *?rhs*
using ∗
by *simp*
next
assume *?rhs*
hence *Re* (*to-complex ?cr*) < *0*
using ∗
by *simp*
moreover
have {0_h, *of-complex* (*cor x*), *of-complex* (*cor y*), *inversion* (*of-complex* (*cor x*))} ⊆ *circline-set x-axis*
using ‹*x* ≠ *0*› *is-real-inversion*[*of cor x*]
using *inf-or-of-complex*[*of inversion ?x*]
by (*auto simp del*: *inversion-of-complex*)
hence *is-real* (*to-complex ?cr*)
using *four-points-on-circline-iff-cross-ratio-real*[*OF distinct*]
by *auto*
ultimately
show *?lhs*
using *distinct*
unfolding *poincare-between-def Let-def*
by *auto*
qed
qed

lemma *poincare-between-0uv*:
assumes *u* ∈ *unit-disc* **and** *v* ∈ *unit-disc* **and** *u* ≠ 0_h **and** *v* ≠ 0_h
shows *poincare-between* 0_h *u v* ⟷
(*let u′* = *to-complex u*; *v′* = *to-complex v in Arg u′* = *Arg v′* ∧ *cmod u′* ≤ *cmod v′*) (**is** *?P u v*)
proof (*cases u* = *v*)
case *True*
thus *?thesis*
by *simp*
next
case *False*
have ∀ *v*. *v* ∈ *unit-disc* ∧ *v* ≠ 0_h ∧ *v* ≠ *u* ⟶ (*poincare-between* 0_h *u v* ⟷ (*let u′* = *to-complex u*; *v′* = *to-complex v in Arg u′* = *Arg v′* ∧ *cmod u′* ≤ *cmod v′*)) (**is** *?P′ u*)
proof (*rule wlog-rotation-to-positive-x-axis*)
show *u* ∈ *unit-disc u* ≠ 0_h
by *fact*+
next
fix *x*
assume ∗: *is-real x 0* < *Re x Re x* < *1*
hence *of-complex x* ∈ *unit-disc of-complex x* ≠ 0_h *of-complex x* ∈ *circline-set x-axis*
unfolding *circline-set-x-axis*
by (*auto simp add*: *cmod-eq-Re*)
show *?P′* (*of-complex x*)
proof *safe*
fix *v*
assume *v* ∈ *unit-disc v* ≠ 0_h *v* ≠ *of-complex x poincare-between* 0_h (*of-complex x*) *v*
hence *v* ∈ *circline-set x-axis*
using *poincare-between-poincare-line-uvz*[*of* 0_h *of-complex x v*]
using *poincare-line-0-real-is-x-axis*[*of of-complex x*]
using ‹*of-complex x* ≠ 0_h› ‹*v* ≠ 0_h› ‹*v* ≠ *of-complex x*› ‹*of-complex x* ∈ *unit-disc*› ‹*of-complex x* ∈ *circline-set x-axis*›

```
        by auto
      obtain v' where v = of-complex v'
        using ‹v ∈ unit-disc›
        using inf-or-of-complex[of v]
        by auto
      hence **: v = of-complex v' -1 < Re v' Re v' < 1 Re v' ≠ 0 is-real v'
        using ‹v ∈ unit-disc› ‹v ≠ 0h› ‹v ∈ circline-set x-axis› of-complex-inj[of v']
        unfolding circline-set-x-axis
        by (auto simp add: cmod-eq-Re real-imag-0)
      show let u' = to-complex (of-complex x); v' = to-complex v in Arg u' = Arg v' ∧ cmod u' ≤ cmod v'
        using poincare-between-x-axis-0uv[of Re x Re v'] * **
        using ‹poincare-between 0h (of-complex x) v›
        using arg-complex-of-real-positive[of Re x] arg-complex-of-real-negative[of Re x]
        using arg-complex-of-real-positive[of Re v'] arg-complex-of-real-negative[of Re v']
        by (auto simp add: cmod-eq-Re)
    next
      fix v
      assume v ∈ unit-disc v ≠ 0h v ≠ of-complex x
      then obtain v' where **: v = of-complex v' v' ≠ 0 v' ≠ x
        using inf-or-of-complex[of v]
        by auto blast
      assume let u' = to-complex (of-complex x); v' = to-complex v in Arg u' = Arg v' ∧ cmod u' ≤ cmod v'
      hence ***: Re x < 0 ∧ Re v' < 0 ∧ Re v' ≤ Re x ∨ 0 < Re x ∧ 0 < Re v' ∧ Re x ≤ Re v' is-real v'
        using arg-pi-iff[of x] arg-pi-iff[of v']
        using arg-0-iff[of x] arg-0-iff[of v']
        using * **
        by (smt cmod-Re-le-iff to-complex-of-complex)+
      have -1 < Re v' Re v' < 1 Re v' ≠ 0 is-real v'
        using ‹v ∈ unit-disc› ** ‹is-real v'›
        by (auto simp add: cmod-eq-Re complex-eq-if-Re-eq)
      thus poincare-between 0h (of-complex x) v
        using poincare-between-x-axis-0uv[of Re x Re v'] * ** ***
        by simp
    qed
  next
    fix φ u
    assume u ∈ unit-disc u ≠ 0h
    let ?M = moebius-rotation φ
    assume *: ?P' (moebius-pt ?M u)
    show ?P' u
    proof (rule allI, rule impI, (erule conjE)+)
      fix v
      assume v ∈ unit-disc v ≠ 0h v ≠ u
      have moebius-pt ?M v ≠ moebius-pt ?M u
        using ‹v ≠ u›
        by auto
      obtain u' v' where v = of-complex v' u = of-complex u' v' ≠ 0 u' ≠ 0
        using inf-or-of-complex[of u] inf-or-of-complex[of v]
        using ‹v ∈ unit-disc› ‹u ∈ unit-disc› ‹v ≠ 0h› ‹u ≠ 0h›
        by auto
      thus ?P u v
        using *[rule-format, of moebius-pt ?M v]
        using ‹moebius-pt ?M v ≠ moebius-pt ?M u›
        using unit-disc-fix-moebius-preserve-poincare-between[of ?M 0h u v]
        using ‹v ∈ unit-disc› ‹u ∈ unit-disc› ‹v ≠ 0h› ‹u ≠ 0h›
        using arg-mult-eq[of cis φ u' v']
        by simp (auto simp add: arg-mult norm-mult)
    qed
  qed
  thus ?thesis
    using assms False
    by auto
qed

lemma poincare-between-y-axis-0uv:
  fixes x y :: real
```

assumes $-1 < x$ **and** $x < 1$ **and** $x \neq 0$
assumes $-1 < y$ **and** $y < 1$ **and** $y \neq 0$
shows *poincare-between* 0_h (*of-complex* (i ∗ x)) (*of-complex* (i ∗ y)) ⟷
($x < 0 \wedge y < 0 \wedge y \leq x$) ∨ ($x > 0 \wedge y > 0 \wedge x \leq y$) (**is** *?lhs* ⟷ *?rhs*)
using *assms*
using *poincare-between-0uv*[*of of-complex* (i ∗ x) *of-complex* (i ∗ y)]
using *arg-pi2-iff*[*of* i ∗ *cor x*] *arg-pi2-iff*[*of* i ∗ *cor y*]
using *arg-minus-pi2-iff*[*of* i ∗ *cor x*] *arg-minus-pi2-iff*[*of* i ∗ *cor y*]
apply (*simp add*: *norm-mult*)
apply (*smt* (*verit*, *best*))
done

lemma *poincare-between-x-axis-uvw*:
fixes *x y z* :: *real*
assumes $-1 < x$ **and** $x < 1$
assumes $-1 < y$ **and** $y < 1$ **and** $y \neq x$
assumes $-1 < z$ **and** $z < 1$ **and** $z \neq x$
shows *poincare-between* (*of-complex x*) (*of-complex y*) (*of-complex z*) ⟷
($y < x \wedge z < x \wedge z \leq y$) ∨ ($y > x \wedge z > x \wedge y \leq z$) (**is** *?lhs* ⟷ *?rhs*)
proof (*cases* $x = 0 \vee y = 0 \vee z = 0$)
case *True*
thus *?thesis*
proof (*cases* $x = 0$)
case *True*
thus *?thesis*
using *poincare-between-x-axis-0uv assms*
by *simp*
next
case *False*
show *?thesis*
proof (*cases* $z = 0$)
case *True*
thus *?thesis*
using *poincare-between-x-axis-0uv assms poincare-between-rev*
by (*smt norm-of-real of-complex-zero of-real-0 poincare-between-nonstrict*(*2*) *unit-disc-iff-cmod-lt-1*)
next
case *False*
have $y = 0$
using ‹$x \neq 0$› ‹$z \neq 0$› ‹$x = 0 \vee y = 0 \vee z = 0$›
by *simp*

have *poincare-between* (*of-complex x*) 0_h (*of-complex z*) = (*is-real z* ∧ $x * z < 0$)
using ‹$x \neq 0$› ‹$z \neq 0$› *poincare-between-x-axis-u0v*
by *auto*
moreover
have $x * z < 0$ ⟷ *?rhs*
using *True* ‹$x \neq 0$› ‹$z \neq 0$›
by (*smt zero-le-mult-iff*)
ultimately
show *?thesis*
using ‹$y = 0$›
by *auto*
qed
qed
next
case *False*
thus *?thesis*
proof (*cases* $z = y$)
case *True*
thus *?thesis*
using *assms*
unfolding *poincare-between-def*
by *auto*
next
case *False*
let *?x* = *of-complex x* **and** *?y* = *of-complex y* **and** *?z* = *of-complex z*

```
  have ?x ∈ unit-disc ?y ∈ unit-disc ?z ∈ unit-disc
    using assms
    by auto

  have distinct: distinct [?x, ?y, ?z, inversion ?y]
    using ‹y ≠ x› ‹z ≠ x› False ‹?x ∈ unit-disc› ‹?y ∈ unit-disc› ‹?z ∈ unit-disc›
    using inversion-noteq-unit-disc[of ?y ?y]
    using inversion-noteq-unit-disc[of ?y ?x]
    using inversion-noteq-unit-disc[of ?y ?z]
    using of-complex-inj[of x y]  of-complex-inj[of y z]  of-complex-inj[of x z]
    by auto

  have cor y * cor x ≠ 1
    using assms
    by (smt minus-mult-minus mult-less-cancel-left2 mult-less-cancel-right2 of-real-1 of-real-eq-iff of-real-mult)

  let ?cr = cross-ratio ?x ?y ?z (inversion ?y)
  have Re (to-complex ?cr) = (x − y) * (z*y − 1)/ ((x*y − 1)*(z − y))
  proof−
    have ⋀y x z. ⟦y ≠ x; z ≠ x; z ≠ y; cor y * cor x ≠ 1; x ≠ 0; y ≠ 0; z ≠ 0⟧ ⟹
          (y * y + y * (y * (x * z)) − (y * x + y * (y * (y * z)))) /
          (y * y + y * (y * (x * z)) − (y * z + y * (y * (y * x)))) =
          (y + y * (x * z) − (x + y * (y * z))) / (y + y * (x * z) − (z + y * (y * x)))
    by (metis (no-types, opaque-lifting) ab-group-add-class.ab-diff-conv-add-uminus distrib-left mult-divide-mult-cancel-left-if
mult-minus-right)
    thus ?thesis
      using ‹y ≠ x› ‹z ≠ x› False ‹¬ (x = 0 ∨ y = 0 ∨ z = 0)›
      using ‹cor y * cor x ≠ 1›
      unfolding inversion-def
      by (transfer, transfer, auto simp add: vec-cnj-def power2-eq-square field-simps split: if-split-asm)
  qed

  moreover
  have (x*y − 1) < 0
    using assms
    by (smt minus-mult-minus mult-less-cancel-right2 zero-less-mult-iff)
  moreover
  have (z*y − 1) < 0
    using assms
    by (smt minus-mult-minus mult-less-cancel-right2 zero-less-mult-iff)
  moreover
  have (x − y) / (z − y) < 0 ⟷ ?rhs
    using ‹y ≠ x› ‹z ≠ x› False ‹¬ (x = 0 ∨ y = 0 ∨ z = 0)›
    by (smt divide-less-cancel divide-nonneg-nonpos divide-nonneg-pos divide-nonpos-nonneg divide-nonpos-nonpos)
  ultimately
  have *: Re (to-complex ?cr) < 0 ⟷ ?rhs
    by (smt algebra-split-simps(24) minus-divide-left zero-less-divide-iff zero-less-mult-iff)
  show ?thesis
  proof
    assume ?lhs
    have is-real (to-complex ?cr) Re (to-complex ?cr) < 0
      using ‹?lhs› distinct
      unfolding poincare-between-def Let-def
      by auto
    thus ?rhs
      using *
      by simp
  next
    assume ?rhs
    hence Re (to-complex ?cr) < 0
      using *
      by simp
    moreover
     have {of-complex (cor x), of-complex (cor y), of-complex (cor z), inversion (of-complex (cor y))} ⊆ circline-set
x-axis
```

using ‹¬ (*x* = *0* ∨ *y* = *0* ∨ *z* = *0*)› *is-real-inversion*[*of cor y*]
using *inf-or-of-complex*[*of inversion ?y*]
by (*auto simp del*: *inversion-of-complex*)
hence *is-real* (*to-complex ?cr*)
using *four-points-on-circline-iff-cross-ratio-real*[*OF distinct*]
by *auto*
ultimately
show *?lhs*
using *distinct*
unfolding *poincare-between-def Let-def*
by *auto*
qed
qed
qed

7.1.5 H-betweenness and h-collinearity

For three h-collinear points at least one of the three possible h-betweeness relations must hold.

lemma *poincare-collinear3-between*:
assumes *u* ∈ *unit-disc* **and** *v* ∈ *unit-disc* **and** *w* ∈ *unit-disc*
assumes *poincare-collinear* {*u*, *v*, *w*}
shows *poincare-between u v w* ∨ *poincare-between u w v* ∨ *poincare-between v u w* (**is** *?P′ u v w*)
proof (*cases u*=*v*)
case *True*
thus *?thesis*
using *assms*
by *auto*
next
case *False*
have ∀ *w*. *w* ∈ *unit-disc* ∧ *poincare-collinear* {*u*, *v*, *w*} ⟶ *?P′ u v w* (**is** *?P u v*)
proof (*rule wlog-positive-x-axis*[**where** *P*=*?P*])
fix *x*
assume *x*: *is-real x 0* < *Re x Re x* < *1*
hence *x* ≠ *0*
using *complex.expand*[*of x 0*]
by *auto*
hence ∗: *poincare-line* 0_h (*of-complex x*) = *x-axis*
using *x poincare-line-0-real-is-x-axis*[*of of-complex x*]
unfolding *circline-set-x-axis*
by *auto*
have *of-complex x* ∈ *unit-disc*
using *x*
by (*auto simp add*: *cmod-eq-Re*)
have *of-complex x* ≠ 0_h
using ‹*x* ≠ *0*›
by *auto*
show *?P* 0_h (*of-complex x*)
proof *safe*
fix *w*
assume *w* ∈ *unit-disc*
assume *poincare-collinear* {0_h, *of-complex x*, *w*}
hence *w* ∈ *circline-set x-axis*
using ∗ *unique-poincare-line*[*of* 0_h *of-complex x*] ‹*of-complex x* ∈ *unit-disc*› ‹*x* ≠ *0*› ‹*of-complex x* ≠ 0_h›
unfolding *poincare-collinear-def*
by *auto*
then obtain *w′* **where** *w′*: *w* = *of-complex w′ is-real w′*
using ‹*w* ∈ *unit-disc*›
using *inf-or-of-complex*[*of w*]
unfolding *circline-set-x-axis*
by *auto*
hence −*1* < *Re w′ Re w′* < *1*
using ‹*w* ∈ *unit-disc*›
by (*auto simp add*: *cmod-eq-Re*)
assume *1*: ¬ *poincare-between* (*of-complex x*) 0_h *w*
hence *w* ≠ 0_h *w′* ≠ *0*
using *w′*

```
      unfolding poincare-between-def
      by auto
    hence Re w′ ≠ 0
      using w′ complex.expand[of w′ 0]
      by auto

    have Re w′ ≥ 0
      using 1 poincare-between-x-axis-u0v[of x w′] ‹Re x > 0› ‹is-real x› ‹x ≠ 0› ‹w′ ≠ 0› w′
      using mult-pos-neg
      by force

    moreover

    assume ¬ poincare-between 0h (of-complex x) w
    hence Re w′ < Re x
      using poincare-between-x-axis-0uv[of Re x Re w′]
      using w′ x ‹−1 < Re w′› ‹Re w′ < 1› ‹Re w′ ≠ 0›
      by auto

    ultimately
    show poincare-between 0h w (of-complex x)
      using poincare-between-x-axis-0uv[of Re w′ Re x]
      using w′ x ‹−1 < Re w′› ‹Re w′ < 1› ‹Re w′ ≠ 0›
      by auto
  qed
 next
  show u ∈ unit-disc v ∈ unit-disc u ≠ v
    by fact+
 next
  fix M u v
  assume 1: unit-disc-fix M u ∈ unit-disc v ∈ unit-disc u ≠ v
  let ?Mu = moebius-pt M u and ?Mv = moebius-pt M v
  assume 2: ?P ?Mu ?Mv
  show ?P u v
  proof safe
    fix w
    assume w ∈ unit-disc poincare-collinear {u, v, w} ¬ poincare-between u v w ¬ poincare-between v u w
    thus poincare-between u w v
      using 1 2[rule-format, of moebius-pt M w]
      by simp
  qed
 qed
 thus ?thesis
  using assms
  by simp
qed

lemma poincare-collinear3-iff:
 assumes u ∈ unit-disc v ∈ unit-disc  w ∈ unit-disc
 shows poincare-collinear {u, v, w} ⟷ poincare-between u v w ∨ poincare-between v u w ∨ poincare-between v w u
 using assms
 by (metis poincare-collinear3-between insert-commute poincare-between-poincare-collinear poincare-between-rev)
```

7.2 Some properties of betweenness

```
lemma poincare-between-transitivity:
 assumes a ∈ unit-disc and x ∈ unit-disc and b ∈ unit-disc and y ∈ unit-disc and
     poincare-between a x b and poincare-between a b y
 shows poincare-between x b y
proof(cases a = b)
 case True
 thus ?thesis
  using assms
  using poincare-between-sandwich by blast
next
 case False
```

```
have ∀ x. ∀ y. poincare-between a x b ∧ poincare-between a b y ∧ x ∈ unit-disc
           ∧ y ∈ unit-disc ⟶ poincare-between x b y (is ?P a b)
proof (rule wlog-positive-x-axis[where P=?P])
  show a ∈ unit-disc
    using assms by simp
next
  show b ∈ unit-disc
    using assms by simp
next
  show a ≠ b
    using False by simp
next
  fix M u v
  assume ∗: unit-disc-fix M u ∈ unit-disc v ∈ unit-disc u ≠ v
          ∀ x y. poincare-between (moebius-pt M u) x (moebius-pt M v) ∧
              poincare-between (moebius-pt M u) (moebius-pt M v) y ∧
              x ∈ unit-disc ∧ y ∈ unit-disc ⟶
              poincare-between x (moebius-pt M v) y
  show ∀ x y. poincare-between u x v ∧ poincare-between u v y ∧ x ∈ unit-disc ∧ y ∈ unit-disc
          ⟶ poincare-between x v y
  proof safe
    fix x y
    assume poincare-between u x v poincare-between u v y  x ∈ unit-disc y ∈ unit-disc

    have poincare-between (moebius-pt M u) (moebius-pt M x) (moebius-pt M v)
      using ‹poincare-between u x v› ‹unit-disc-fix M› ‹x ∈ unit-disc› ‹u ∈ unit-disc› ‹v ∈ unit-disc›
      by simp
    moreover
    have poincare-between (moebius-pt M u) (moebius-pt M v) (moebius-pt M y)
      using ‹poincare-between u v y› ‹unit-disc-fix M› ‹y ∈ unit-disc› ‹u ∈ unit-disc› ‹v ∈ unit-disc›
      by simp
    moreover
    have (moebius-pt M x) ∈ unit-disc
      using ‹unit-disc-fix M› ‹x ∈ unit-disc› by simp
    moreover
    have (moebius-pt M y) ∈ unit-disc
      using ‹unit-disc-fix M› ‹y ∈ unit-disc› by simp
    ultimately
    have poincare-between (moebius-pt M x) (moebius-pt M v) (moebius-pt M y)
      using ∗ by blast
    thus poincare-between x v y
      using ‹y ∈ unit-disc› ∗ ‹x ∈ unit-disc› by simp
  qed
next
  fix x
  assume xx: is-real x 0 < Re x Re x < 1
  hence of-complex x ∈ unit-disc
    using cmod-eq-Re by auto
  hence of-complex x ≠ ∞h
    by simp
  have  of-complex x ≠ 0h
    using xx by auto
  have of-complex x ∈ circline-set x-axis
    using xx by simp
  show ∀ m n. poincare-between 0h m (of-complex x) ∧ poincare-between 0h (of-complex x) n ∧
        m ∈ unit-disc ∧ n ∈ unit-disc ⟶ poincare-between m (of-complex x) n
  proof safe
    fix m n
    assume ∗∗: poincare-between 0h m (of-complex x) poincare-between 0h (of-complex x) n
             m ∈ unit-disc  n ∈ unit-disc
    show poincare-between m (of-complex x) n
    proof(cases m = 0h)
      case True
      thus ?thesis
        using ∗∗ by auto
    next
```

case *False*
hence $m \in$ *circline-set x-axis*
using *poincare-between-poincare-line-uzv*[*of* 0_h *of-complex x m*]
using *poincare-line-0-real-is-x-axis*[*of of-complex x*]
using ‹*of-complex x* ∈ *unit-disc*› ‹*of-complex x* ≠ ∞_h› ‹*of-complex x* ≠ 0_h›
using ‹*of-complex x* ∈ *circline-set x-axis*› ‹*m* ∈ *unit-disc*› **(*1*)
by *simp*
then obtain m' **where** $m =$ *of-complex* m' *is-real* m'
using *inf-or-of-complex*[*of m*] ‹*m* ∈ *unit-disc*›
unfolding *circline-set-x-axis*
by *auto*
hence *Re* $m' \leq$ *Re x*
using ‹*poincare-between* 0_h *m* (*of-complex x*)› *xx* ‹*of-complex x* ≠ 0_h›
using *False* ** ‹*of-complex x* ∈ *unit-disc*›
using *cmod-Re-le-iff poincare-between-0uv* **by** *auto*

have $n \neq 0_h$
using **(*2*, *4*) ‹*of-complex x* ≠ 0_h› ‹*of-complex x* ∈ *unit-disc*›
using *poincare-between-sandwich* **by** *fastforce*
have *n* ∈ *circline-set x-axis*
using *poincare-between-poincare-line-uvz*[*of* 0_h *of-complex x n*]
using *poincare-line-0-real-is-x-axis*[*of of-complex x*]
using ‹*of-complex x* ∈ *unit-disc*› ‹*of-complex x* ≠ ∞_h› ‹*of-complex x* ≠ 0_h›
using ‹*of-complex x* ∈ *circline-set x-axis*› ‹*n* ∈ *unit-disc*› **(*2*)
by *simp*
then obtain n' **where** $n =$ *of-complex* n' *is-real* n'
using *inf-or-of-complex*[*of n*] ‹*n* ∈ *unit-disc*›
unfolding *circline-set-x-axis*
by *auto*
hence *Re x* ≤ *Re* n'
using ‹*poincare-between* 0_h (*of-complex x*) *n*› *xx* ‹*of-complex x* ≠ 0_h›
using *False* ** ‹*of-complex x* ∈ *unit-disc*› ‹$n \neq 0_h$›
using *cmod-Re-le-iff poincare-between-0uv*
by (*metis Re-complex-of-real arg-0-iff rcis-cmod-Arg rcis-zero-arg to-complex-of-complex*)

have *poincare-between* (*of-complex* m') (*of-complex x*) (*of-complex* n')
using ‹*Re x* ≤ *Re* n'› ‹*Re* m' ≤ *Re x*›
using *poincare-between-x-axis-uvw*[*of Re* m' *Re x Re* n']
using ‹*is-real* n'› ‹*is-real* m'› ‹*n* ∈ *unit-disc*› ‹*n* = *of-complex* n'›
using *xx* ‹*m* = *of-complex* m'› ‹*m* ∈ *unit-disc*›
by (*smt complex-of-real-Re norm-of-real poincare-between-def unit-disc-iff-cmod-lt-1*)

thus *?thesis*
using ‹*n* = *of-complex* n'› ‹*m* = *of-complex* m'›
by *auto*
qed
qed
qed
thus *?thesis*
using *assms*
by *blast*
qed

7.3 Poincare between - sum distances

Another possible definition of the h-betweenness relation is given in terms of h-distances between pairs of points. We prove it as a characterization equivalent to our cross-ratio based definition.

lemma *poincare-between-sum-distances-x-axis-u0v*:
assumes *of-complex* u' ∈ *unit-disc of-complex* v' ∈ *unit-disc*
assumes *is-real* u' $u' \neq 0$ $v' \neq 0$
shows *poincare-distance* (*of-complex* u') 0_h + *poincare-distance* 0_h (*of-complex* v') = *poincare-distance* (*of-complex* u') (*of-complex* v') ⟷
is-real v' ∧ *Re* u' ∗ *Re* v' < *0* (**is** *?P* u' v' ⟷ *?Q* u' v')
proof−
have *Re* $u' \neq 0$

using ‹*is-real* u'› ‹$u' \neq 0$›
using *complex-eq-if-Re-eq*
by *simp*

let $?u = cmod\ u'$ **and** $?v = cmod\ v'$ **and** $?uv = cmod\ (u' - v')$
have *disc*: $?u^2 < 1$ $?v^2 < 1$
using *unit-disc-cmod-square-lt-1*[*OF assms*(*1*)]
using *unit-disc-cmod-square-lt-1*[*OF assms*(*2*)]
by *auto*
have *poincare-distance* (*of-complex* u') 0_h + *poincare-distance* 0_h (*of-complex* v') =
arcosh $(((1 + ?u^2) * (1 + ?v^2) + 4 * ?u * ?v) / ((1 - ?u^2) * (1 - ?v^2)))$ (**is** - = *arcosh ?r1*)
using *poincare-distance-formula-zero-sum*[*OF assms*(*1*−*2*)]
by (*simp add*: *Let-def*)
moreover
have *poincare-distance* (*of-complex* u') (*of-complex* v') =
arcosh $(((1 - ?u^2) * (1 - ?v^2) + 2 * ?uv^2) / ((1 - ?u^2) * (1 - ?v^2)))$ (**is** - = *arcosh ?r2*)
using *disc*
using *poincare-distance-formula*[*OF assms*(*1*−*2*)]
by (*subst add-divide-distrib*) *simp*
moreover
have *arcosh ?r1* = *arcosh ?r2* ⟷ *?Q* u' v'
proof
assume *arcosh ?r1* = *arcosh ?r2*
hence *?r1* = *?r2*
proof (*subst* (*asm*) *arcosh-eq-iff*)
show $?r1 \geq 1$
proof−
have $(1 - ?u^2) * (1 - ?v^2) \leq (1 + ?u^2) * (1 + ?v^2) + 4 * ?u * ?v$
by (*simp add*: *field-simps*)
thus *?thesis*
using *disc*
by *simp*
qed
next
show $?r2 \geq 1$
using *disc*
by *simp*
qed
hence $(1 + ?u^2) * (1 + ?v^2) + 4 * ?u * ?v = (1 - ?u^2) * (1 - ?v^2) + 2 * ?uv^2$
using *disc*
by *auto*
hence $(cmod\ (u' - v'))^2 = (cmod\ u' + cmod\ v')^2$
by (*simp add*: *field-simps power2-eq-square*)
hence ∗: $Re\ u' * Re\ v' + |Re\ u'| * sqrt\ ((Im\ v')^2 + (Re\ v')^2) = 0$
using ‹*is-real* u'›
unfolding *cmod-power2 cmod-def*
by (*simp add*: *field-simps*) (*simp add*: *power2-eq-square field-simps*)
hence $sqrt\ ((Im\ v')^2 + (Re\ v')^2) = |Re\ v'|$
using ‹$Re\ u' \neq 0$› ‹$v' \neq 0$›
by (*smt complex-neq-0 mult.commute mult-cancel-right mult-minus-left real-sqrt-gt-0-iff*)
hence $Im\ v' = 0$
by (*smt Im-eq-0 norm-complex-def*)
moreover
hence $Re\ u' * Re\ v' = -\ |Re\ u'| * |Re\ v'|$
using ∗
by *simp*
hence $Re\ u' * Re\ v' < 0$
using ‹$Re\ u' \neq 0$› ‹$v' \neq 0$›
by (*simp add*: ‹*is-real* v'› *complex-eq-if-Re-eq*)
ultimately
show *?Q* u' v'
by *simp*
next
assume *?Q* u' v'
hence *is-real* v' $Re\ u' * Re\ v' < 0$
by *auto*

have *?r1* = *?r2*
proof (*cases Re u′ > 0*)
case *True*
hence *Re v′ < 0*
using ‹*Re u′ ∗ Re v′ < 0*›
by (*smt zero-le-mult-iff*)
show *?thesis*
using *disc* ‹*is-real u′*› ‹*is-real v′*›
using ‹*Re u′ > 0*› ‹*Re v′ < 0*›
unfolding *cmod-power2 cmod-def*
by *simp* (*simp add*: *power2-eq-square field-simps*)
next
case *False*
hence *Re u′ < 0*
using ‹*Re u′ ≠ 0*›
by *simp*
hence *Re v′ > 0*
using ‹*Re u′ ∗ Re v′ < 0*›
by (*smt zero-le-mult-iff*)
show *?thesis*
using *disc* ‹*is-real u′*› ‹*is-real v′*›
using ‹*Re u′ < 0*› ‹*Re v′ > 0*›
unfolding *cmod-power2 cmod-def*
by *simp* (*simp add*: *power2-eq-square field-simps*)
qed
thus *arcosh ?r1* = *arcosh ?r2*
by *metis*
qed
ultimately
show *?thesis*
by *simp*
qed

Different proof of the previous theorem relying on the cross-ratio definition, and not the distance formula. We suppose that this could be also used to prove the triangle inequality.

lemma *poincare-between-sum-distances-x-axis-u0v-different-proof*:
assumes *of-complex u′ ∈ unit-disc of-complex v′ ∈ unit-disc*
assumes *is-real u′ u′ ≠ 0 v′ ≠ 0 is-real v′*
shows *poincare-distance* (*of-complex u′*) 0_h + *poincare-distance* 0_h (*of-complex v′*) = *poincare-distance* (*of-complex u′*) (*of-complex v′*) ⟷
Re u′ ∗ Re v′ < 0 (**is** *?P u′ v′* ⟷ *?Q u′ v′*)
proof−
have −*1 < Re u′ Re u′ < 1 Re u′ ≠ 0*
using *assms*
by (*auto simp add*: *cmod-eq-Re complex-eq-if-Re-eq*)
have −*1 < Re v′ Re v′ < 1 Re v′ ≠ 0*
using *assms*
by (*auto simp add*: *cmod-eq-Re complex-eq-if-Re-eq*)

have |*ln* (*Re* ((*1* − *u′*) / (*1* + *u′*)))| + |*ln* (*Re* ((*1* − *v′*) / (*1* + *v′*)))| =
|*ln* (*Re* ((*1* + *u′*) ∗ (*1* − *v′*) / ((*1* − *u′*) ∗ (*1* + *v′*))))| ⟷ *Re u′ ∗ Re v′ < 0* (**is** |*ln ?a1*| + |*ln ?a2*| = |*ln ?a3*| ⟷ -)
proof−
have *1*: *0 < ?a1 ln ?a1 > 0* ⟷ *Re u′ < 0*
using ‹*Re u′ < 1*› ‹*Re u′ >* −*1*› ‹*is-real u′*›
using *complex-is-Real-iff*
by *auto*
have *2*: *0 < ?a2 ln ?a2 > 0* ⟷ *Re v′ < 0*
using ‹*Re v′ < 1*› ‹*Re v′ >* −*1*› ‹*is-real v′*›
using *complex-is-Real-iff*
by *auto*
have *3*: *0 < ?a3 ln ?a3 > 0* ⟷ *Re v′ < Re u′*
using ‹*Re u′ < 1*› ‹*Re u′ >* −*1*› ‹*is-real u′*›
using ‹*Re v′ < 1*› ‹*Re v′ >* −*1*› ‹*is-real v′*›
using *complex-is-Real-iff*
by *auto* (*simp add*: *field-simps*)+

```
show ?thesis
proof
  assume *: Re u' * Re v' < 0
  show |ln ?a1| + |ln ?a2| = |ln ?a3|
  proof (cases Re u' > 0)
    case True
    hence Re v' < 0
      using *
      by (smt mult-nonneg-nonneg)
    show ?thesis
      using 1 2 3 ‹Re u' > 0› ‹Re v' < 0›
      using ‹Re u' < 1› ‹Re u' > −1› ‹is-real u'›
      using ‹Re v' < 1› ‹Re v' > −1› ‹is-real v'›
      using complex-is-Real-iff
      using ln-div ln-mult
      by simp
  next
    case False
    hence Re v' > 0 Re u' < 0
      using *
      by (smt zero-le-mult-iff)+
    show ?thesis
      using 1 2 3 ‹Re u' < 0› ‹Re v' > 0›
      using ‹Re u' < 1› ‹Re u' > −1› ‹is-real u'›
      using ‹Re v' < 1› ‹Re v' > −1› ‹is-real v'›
      using complex-is-Real-iff
      using ln-div ln-mult
      by simp
  qed
next
  assume *: |ln ?a1| + |ln ?a2| = |ln ?a3|
  {
    assume Re u' > 0 Re v' > 0
    hence False
      using * 1 2 3
      using ‹Re u' < 1› ‹Re u' > −1› ‹is-real u'›
      using ‹Re v' < 1› ‹Re v' > −1› ‹is-real v'›
      using complex-is-Real-iff
      using ln-mult ln-div
      by (cases Re v' < Re u') auto
  }
  moreover
  {
    assume Re u' < 0 Re v' < 0
    hence False
      using * 1 2 3
      using ‹Re u' < 1› ‹Re u' > −1› ‹is-real u'›
      using ‹Re v' < 1› ‹Re v' > −1› ‹is-real v'›
      using complex-is-Real-iff
      using ln-mult ln-div
      by (cases Re v' < Re u') auto
  }
  ultimately
  show Re u' * Re v' < 0
    using ‹Re u' ≠ 0› ‹Re v' ≠ 0›
    by (smt divisors-zero mult-le-0-iff)
qed
qed
thus ?thesis
  using assms
  apply (subst poincare-distance-sym, simp, simp)
  apply (subst poincare-distance-zero-x-axis, simp, simp add: circline-set-x-axis)
  apply (subst poincare-distance-zero-x-axis, simp, simp add: circline-set-x-axis)
  apply (subst poincare-distance-x-axis-x-axis, simp, simp, simp add: circline-set-x-axis, simp add: circline-set-x-axis)
  apply simp
  done
```

qed

lemma *poincare-between-sum-distances*:
  assumes $u \in$ *unit-disc* **and** $v \in$ *unit-disc* **and** $w \in$ *unit-disc*
  shows *poincare-between u v w* $\longleftrightarrow$
      poincare-distance u v + *poincare-distance v w* = *poincare-distance u w* (**is** *?P′ u v w*)
proof (*cases u = v*)
  case *True*
  thus *?thesis*
    using *assms*
    by *simp*
next
  case *False*
  have ∀ *w*. *w* ∈ *unit-disc* ⟶ (*poincare-between u v w* ⟷ *poincare-distance u v* + *poincare-distance v w* = *poincare-distance u w*) (**is** *?P u v*)
  proof (*rule wlog-positive-x-axis*)
    fix *x*
    assume *is-real x 0 < Re x Re x < 1*
    have *of-complex x* ∈ *circline-set x-axis*
      using ‹*is-real x*›
      by (*auto simp add*: *circline-set-x-axis*)

    have *of-complex x* ∈ *unit-disc*
      using ‹*is-real x*› ‹*0 < Re x*› ‹*Re x < 1*›
      by (*simp add*: *cmod-eq-Re*)

    have *x* ≠ *0*
      using ‹*is-real x*› ‹*Re x > 0*›
      by *auto*

    show *?P* (*of-complex x*) 0_h
    proof (*rule allI*, *rule impI*)
      fix *w*
      assume *w* ∈ *unit-disc*
      then obtain *w′* **where** *w = of-complex w′*
        using *inf-or-of-complex*[*of w*]
        by *auto*

      show *?P′* (*of-complex x*) 0_h *w*
      proof (*cases w =* 0_h)
        case *True*
        thus *?thesis*
          by *simp*
      next
        case *False*
        hence *w′* ≠ *0*
          using ‹*w = of-complex w′*›
          by *auto*

        show *?thesis*
          using ‹*is-real x*› ‹*x* ≠ *0*› ‹*w = of-complex w′*› ‹*w′* ≠ *0*›
          using ‹*of-complex x* ∈ *unit-disc*› ‹*w* ∈ *unit-disc*›
          apply *simp*
          apply (*subst poincare-between-x-axis-u0v*, *simp-all*)
          apply (*subst poincare-between-sum-distances-x-axis-u0v*, *simp-all*)
          done
      qed
    qed
  next
    show *v* ∈ *unit-disc u* ∈ *unit-disc*
      using *assms*
      by *auto*
  next
    show *v* ≠ *u*
      using ‹*u* ≠ *v*›
      by *simp*

next
  fix *M u v*
  assume $*$: *unit-disc-fix M u* $\in$ *unit-disc v* $\in$ *unit-disc u* $\neq$ *v* **and**
      $**$: *?P* (*moebius-pt M v*) (*moebius-pt M u*)
  show *?P v u*
  proof (*rule allI*, *rule impI*)
    fix *w*
    assume *w* $\in$ *unit-disc*
    hence *moebius-pt M w* $\in$ *unit-disc*
      using ‹*unit-disc-fix M*›
      by *auto*
    thus *?P′ v u w*
      using ‹*u* $\in$ *unit-disc*› ‹*v* $\in$ *unit-disc*› ‹*w* $\in$ *unit-disc*› ‹*unit-disc-fix M*›
      using $**$[*rule-format*, *of moebius-pt M w*]
      by *auto*
  qed
qed
thus *?thesis*
  using *assms*
  by *simp*
qed

7.4 Some more properties of h-betweenness.

Some lemmas proved earlier are proved almost directly using the sum of distances characterization.

lemma *unit-disc-fix-moebius-preserve-poincare-between′*:
  assumes *unit-disc-fix M* **and** *u* $\in$ *unit-disc* **and** *v* $\in$ *unit-disc* **and** *w* $\in$ *unit-disc*
  shows *poincare-between* (*moebius-pt M u*) (*moebius-pt M v*) (*moebius-pt M w*) $\longleftrightarrow$
      poincare-between u v w
  using *assms*
  using *poincare-between-sum-distances*
  by *simp*

lemma *conjugate-preserve-poincare-between′*:
  assumes *u* $\in$ *unit-disc v* $\in$ *unit-disc w* $\in$ *unit-disc*
  shows *poincare-between* (*conjugate u*) (*conjugate v*) (*conjugate w*) $\longleftrightarrow$ *poincare-between u v w*
  using *assms*
  using *poincare-between-sum-distances*
  by *simp*

There is a unique point on a ray on the given distance from the given starting point

lemma *unique-poincare-distance-on-ray*:
  assumes *d* $\geq$ *0 u* $\neq$ *v u* $\in$ *unit-disc v* $\in$ *unit-disc*
  assumes *y* $\in$ *unit-disc poincare-distance u y* = *d poincare-between u v y*
  assumes *z* $\in$ *unit-disc poincare-distance u z* = *d poincare-between u v z*
  shows *y* = *z*
proof−
  have $\forall$ *d y z. d* $\geq$ *0* $\wedge$
      y $\in$ *unit-disc* $\wedge$ *poincare-distance u y* = *d* $\wedge$ *poincare-between u v y* $\wedge$
      z $\in$ *unit-disc* $\wedge$ *poincare-distance u z* = *d* $\wedge$ *poincare-between u v z* $\longrightarrow$ *y* = *z* (**is** *?P u v*)
  proof (*rule wlog-positive-x-axis*[**where** *P*=*?P*])
    fix *x*
    assume *x*: *is-real x 0* < *Re x Re x* < *1*
    hence *x* $\neq$ *0*
      using *complex.expand*[*of x 0*]
      by *auto*
    hence $*$: *poincare-line* 0_h (*of-complex x*) = *x-axis*
      using *x poincare-line-0-real-is-x-axis*[*of of-complex x*]
      unfolding *circline-set-x-axis*
      by *auto*
    have *of-complex x* $\in$ *unit-disc*
      using *x*
      by (*auto simp add*: *cmod-eq-Re*)
    have *Arg x* = *0*
      using *x*

```
      using arg-0-iff by blast
    show ?P 0_h (of-complex x)
    proof safe
      fix y z
      assume y ∈ unit-disc z ∈ unit-disc
      then obtain y' z' where yz: y = of-complex y' z = of-complex z'
        using inf-or-of-complex[of y] inf-or-of-complex[of z]
        by auto
      assume betw: poincare-between 0_h (of-complex x) y  poincare-between 0_h (of-complex x) z
      hence y ≠ 0_h z ≠ 0_h
        using ‹x ≠ 0› ‹of-complex x ∈ unit-disc› ‹y ∈ unit-disc›
        using poincare-between-sandwich[of 0_h of-complex x]
        using of-complex-zero-iff[of x]
        by force+

      hence Arg y' = 0 cmod y' ≥ cmod x Arg z' = 0 cmod z' ≥ cmod x
        using poincare-between-0uv[of of-complex x y] poincare-between-0uv[of of-complex x z]
        using ‹of-complex x ∈ unit-disc› ‹x ≠ 0› ‹Arg x = 0› ‹y ∈ unit-disc› ‹z ∈ unit-disc› betw yz
        by (simp-all add: Let-def)
      hence *: is-real y' is-real z' Re y' > 0 Re z' > 0
        using arg-0-iff[of y'] arg-0-iff[of z'] x ‹y ≠ 0_h› ‹z ≠ 0_h› yz
        by auto
      assume poincare-distance 0_h z = poincare-distance 0_h y 0 ≤ poincare-distance 0_h y
      thus y = z
        using * yz ‹y ∈ unit-disc› ‹z ∈ unit-disc›
        using unique-x-axis-poincare-distance-positive[of poincare-distance 0_h y]
        by (auto simp add: cmod-eq-Re unit-disc-to-complex-inj)
    qed
  next
    show u ∈ unit-disc v ∈ unit-disc u ≠ v
      by fact+
  next
    fix M u v
    assume *: unit-disc-fix M u ∈ unit-disc v ∈ unit-disc u ≠ v
    assume **: ?P (moebius-pt M u) (moebius-pt M v)
    show ?P u v
    proof safe
      fix d y z
      assume ***: 0 ≤ poincare-distance u y
             y ∈ unit-disc poincare-between u v y
             z ∈ unit-disc poincare-between u v z
             poincare-distance u z = poincare-distance u y
      let ?Mu = moebius-pt M u and ?Mv = moebius-pt M v and ?My = moebius-pt M y and ?Mz = moebius-pt M z
      have ?Mu ∈ unit-disc ?Mv ∈ unit-disc ?My ∈ unit-disc ?Mz ∈ unit-disc
        using ‹u ∈ unit-disc› ‹v ∈ unit-disc› ‹y ∈ unit-disc› ‹z ∈ unit-disc›
        using ‹unit-disc-fix M›
        by auto
      hence ?My = ?Mz
        using * ***
        using **[rule-format, of poincare-distance ?Mu ?My ?My ?Mz]
        by simp
      thus y = z
        using bij-moebius-pt[of M]
        unfolding bij-def inj-on-def
        by blast
    qed
  qed
  thus ?thesis
    using assms
    by auto
qed

end
theory Poincare-Lines-Axis-Intersections
  imports Poincare-Between
begin
```

8 Intersection of h-lines with the x-axis in the Poincaré model

8.1 Betweeness of x-axis intersection

The intersection point of the h-line determined by points u and v and the x-axis is between u and v, then u and v are in the opposite half-planes (one must be in the upper, and the other one in the lower half-plane).

lemma *poincare-between-x-axis-intersection*:
  assumes $u \in$ *unit-disc* **and** $v \in$ *unit-disc* **and** $z \in$ *unit-disc* **and** $u \neq v$
  assumes $u \notin$ *circline-set x-axis* **and** $v \notin$ *circline-set x-axis*
  assumes $z \in$ *circline-set* (*poincare-line u v*) $\cap$ *circline-set x-axis*
  shows *poincare-between u z v* $\longleftrightarrow$ *Arg* (*to-complex u*) $*$ *Arg* (*to-complex v*) $<$ *0*
proof−
  have $\forall$ *u v*. $u \in$ *unit-disc* $\wedge$ $v \in$ *unit-disc* $\wedge$ $u \neq v$ $\wedge$
      $u \notin$ *circline-set x-axis* $\wedge$ $v \notin$ *circline-set x-axis* $\wedge$
      $z \in$ *circline-set* (*poincare-line u v*) $\cap$ *circline-set x-axis* $\longrightarrow$
      (*poincare-between u z v* $\longleftrightarrow$ *Arg* (*to-complex u*) $*$ *Arg* (*to-complex v*) $<$ *0*) (**is** *?P z*)
  proof (*rule wlog-real-zero*)
    show *?P* 0_h
    proof ((*rule allI*)+, *rule impI*, (*erule conjE*)+)
      fix *u v*
      assume $*$: $u \in$ *unit-disc* $v \in$ *unit-disc* $u \neq v$
          $u \notin$ *circline-set x-axis* $v \notin$ *circline-set x-axis*
          $0_h \in$ *circline-set* (*poincare-line u v*) $\cap$ *circline-set x-axis*
      obtain u' v' **where** *uv*: u = *of-complex* u' v = *of-complex* v'
        using $*$ *inf-or-of-complex*[*of u*] *inf-or-of-complex*[*of v*]
        by *auto*

      hence $u \neq 0_h$ $v \neq 0_h$ $u' \neq 0$ $v' \neq 0$
        using $*$
        by *auto*

      hence *Arg* $u' \neq 0$ *Arg* $v' \neq 0$
        using $*$ *arg-0-iff*[*of* u'] *arg-0-iff*[*of* v']
        unfolding *circline-set-x-axis uv*
        by *auto*

      have *poincare-collinear* $\{0_h, u, v\}$
        using $*$
        unfolding *poincare-collinear-def*
        by (*rule-tac x=poincare-line u v* **in** *exI*, *simp*)
      have ($\exists k<0.$ u' = *cor* $k * v'$) $\longleftrightarrow$ (*Arg* $u' *$ *Arg* $v' < 0$) (**is** *?lhs* $\longleftrightarrow$ *?rhs*)
      proof
        assume *?lhs*
        then obtain k **where** $k < 0$ u' = *cor* $k * v'$
          by *auto*
        thus *?rhs*
          using *arg-mult-real-negative*[*of k* v'] *arg-uminus-opposite-sign*[*of* v']
          using ‹$u' \neq 0$› ‹$v' \neq 0$› ‹*Arg* $u' \neq 0$› ‹*Arg* $v' \neq 0$›
          by (*auto simp add*: *mult-neg-pos mult-pos-neg*)
      next
        assume *?rhs*
        obtain *ru rv* φ **where** *polar*: u' = *cor ru* $*$ *cis* φ v' = *cor rv* $*$ *cis* φ
          using ‹*poincare-collinear* $\{0_h, u, v\}$› *poincare-collinear-zero-polar-form*[*of* u' v'] *uv* $*$ ‹$u' \neq 0$› ‹$v' \neq 0$›
          by *auto*
        have *ru* $*$ *rv* < 0
          using *polar* ‹*?rhs*› ‹$u' \neq 0$› ‹$v' \neq 0$›
          using *arg-mult-real-negative*[*of ru cis* φ] *arg-mult-real-positive*[*of ru cis* φ]
          using *arg-mult-real-negative*[*of rv cis* φ] *arg-mult-real-positive*[*of rv cis* φ]
          apply (*cases ru* $>$ *0*)
          apply (*cases rv* $>$ *0*, *simp*, *simp add*: *mult-pos-neg*)
          apply (*cases rv* $>$ *0*, *simp add*: *mult-neg-pos*, *simp*)
          done
        thus *?lhs*
          using *polar*
          by (*rule-tac x=ru / rv* **in** *exI*, *auto simp add*: *divide-less-0-iff mult-less-0-iff*)

```
      qed
      thus poincare-between u 0_h v = (Arg (to-complex u) * Arg (to-complex v) < 0)
        using poincare-between-u0v[of u v] * ‹u ≠ 0_h› ‹v ≠ 0_h› uv
        by simp
    qed
  next
    fix a z
    assume 1: is-real a cmod a < 1 z ∈ unit-disc
    assume 2: ?P (moebius-pt (blaschke a) z)
    show ?P z
    proof ((rule allI)+, rule impI, (erule conjE)+)
      fix u v
      let ?M = moebius-pt (blaschke a)
      let ?Mu = ?M u
      let ?Mv = ?M v
      assume *: u ∈ unit-disc v ∈ unit-disc u ≠ v u ∉ circline-set x-axis v ∉ circline-set x-axis
      hence u ≠ ∞_h v ≠ ∞_h
        by auto

      have **: ⋀ x y :: real. x * y < 0 ⟷ sgn (x * y) < 0
        by simp

      assume z ∈ circline-set (poincare-line u v) ∩ circline-set x-axis
      thus poincare-between u z v = (Arg (to-complex u) * Arg (to-complex v) < 0)
        using * 1 2[rule-format, of ?Mu ?Mv] ‹cmod a < 1› ‹is-real a› blaschke-unit-disc-fix[of a]
        using inversion-noteq-unit-disc[of of-complex a u] ‹u ≠ ∞_h›
        using inversion-noteq-unit-disc[of of-complex a v] ‹v ≠ ∞_h›
        apply auto
        apply (subst (asm) **, subst **, subst (asm) sgn-mult, subst sgn-mult, simp)
        apply (subst (asm) **, subst (asm) **, subst (asm) sgn-mult, subst (asm) sgn-mult, simp)
        done
    qed
  next
    show z ∈ unit-disc by fact
  next
    show is-real (to-complex z)
      using assms inf-or-of-complex[of z]
      by (auto simp add: circline-set-x-axis)
  qed
  thus ?thesis
    using assms
    by simp
qed
```

8.2 Check if an h-line intersects the x-axis

```
lemma x-axis-intersection-equation:
  assumes
   H = mk-circline A B C D and
   (A, B, C, D) ∈ hermitean-nonzero
 shows of-complex z ∈ circline-set x-axis ∩ circline-set H ⟷
        A*z^2 + 2*Re B*z + D = 0 ∧ is-real z (is ?lhs ⟷ ?rhs)
proof−
  have ?lhs ⟷ A*z^2 + (B + cnj B)*z + D = 0 ∧ z = cnj z
    using assms
    using circline-equation-x-axis[of z]
    using circline-equation[of H A B C D z]
    using hermitean-elems
    by (auto simp add: power2-eq-square field-simps)
  thus ?thesis
    using eq-cnj-iff-real[of z]
    using hermitean-elems[of A B C D]
    by (simp add: complex-add-cnj complex-eq-if-Re-eq)
qed
```

Check if an h-line intersects x-axis within the unit disc - this could be generalized to checking if an arbitrary circline intersects the x-axis, but we do not need that.

definition *intersects-x-axis-cmat* :: *complex-mat* ⇒ *bool* **where**
  [*simp*]: *intersects-x-axis-cmat* $H = (let\ (A,\ B,\ C,\ D) = H\ in\ A = 0 \lor (Re\ B)^2 > (Re\ A)^2)$

lift-definition *intersects-x-axis-clmat* :: *circline-mat* ⇒ *bool* **is** *intersects-x-axis-cmat*
  done

lift-definition *intersects-x-axis* :: *circline* ⇒ *bool* **is** *intersects-x-axis-clmat*
proof (*transfer*)
  fix *H1 H2*
  assume *hh*: *hermitean H1* ∧ *H1* ≠ *mat-zero* **and** *hermitean H2* ∧ *H2* ≠ *mat-zero*
  obtain *A1 B1 C1 D1 A2 B2 C2 D2* **where** ∗: $H1 = (A1,\ B1,\ C1,\ D1)$ $H2 = (A2,\ B2,\ C2,\ D2)$
    by (*cases H1*, *cases H2*, *auto*)
  assume *circline-eq-cmat H1 H2*
  then obtain *k* **where** *k*: $k \neq 0 \land H2 = cor\ k *_{sm} H1$
    by *auto*
  show *intersects-x-axis-cmat H1* = *intersects-x-axis-cmat H2*
  proof−
    have $k \neq 0 \Longrightarrow (Re\ A1)^2 < (Re\ B1)^2 \longleftrightarrow (k * Re\ A1)^2 < (k * Re\ B1)^2$
      by (*smt mult-strict-left-mono power2-eq-square semiring-normalization-rules*(*13*) *zero-less-power2*)
    thus *?thesis*
      using ∗ *k*
      by *auto*
  qed
qed

lemma *intersects-x-axis-mk-circline*:
  assumes *is-real A* **and** $A \neq 0 \lor B \neq 0$
  shows *intersects-x-axis* (*mk-circline A B* (*cnj B*) *A*) $\longleftrightarrow A = 0 \lor (Re\ B)^2 > (Re\ A)^2$
proof−
  let *?H* = (*A*, *B*, (*cnj B*), *A*)
  have *hermitean ?H*
    using ‹*is-real A*›
    unfolding *hermitean-def mat-adj-def mat-cnj-def*
    using *eq-cnj-iff-real*
    by *auto*
  moreover
  have *?H* ≠ *mat-zero*
    using *assms*
    by *auto*
  ultimately
  show *?thesis*
    by (*transfer*, *transfer*, *auto simp add*: *Let-def*)
qed

lemma *intersects-x-axis-iff*:
  assumes *is-poincare-line H*
  shows (∃ *x* ∈ *unit-disc*. *x* ∈ *circline-set H* ∩ *circline-set x-axis*) ⟷ *intersects-x-axis H*
proof−
  obtain *Ac B C Dc* **where** ∗: *H* = *mk-circline Ac B C Dc hermitean* (*Ac*, *B*, *C*, *Dc*) (*Ac*, *B*, *C*, *Dc*) ≠ *mat-zero*
    using *ex-mk-circline*[*of H*]
    by *auto*
  hence $(cmod\ B)^2 > (cmod\ Ac)^2$ $Ac = Dc$
    using *assms*
    using *is-poincare-line-mk-circline*
    by *auto*

  hence *H* = *mk-circline* (*Re Ac*) *B* (*cnj B*) (*Re Ac*) *hermitean* (*cor* (*Re Ac*), *B*, (*cnj B*), *cor* (*Re Ac*)) (*cor* (*Re Ac*), *B*, (*cnj B*), *cor* (*Re Ac*)) ≠ *mat-zero*
    using *hermitean-elems*[*of Ac B C Dc*] ∗
    by *auto*
  then obtain *A* **where**
    ∗: *H* = *mk-circline* (*cor A*) *B* (*cnj B*) (*cor A*) (*cor A*, *B*, (*cnj B*), *cor A*) ∈ *hermitean-nonzero*
    by *auto*

  show *?thesis*

proof (*cases* $A = 0$)
case *True*
thus *?thesis*
using $*$
using *x-axis-intersection-equation*[*OF* $*(1-2)$, *of* 0]
using *intersects-x-axis-mk-circline*[*of cor A B*]
by *auto*
next
case *False*
show *?thesis*
proof
assume $\exists\ x \in$ *unit-disc*. $x \in$ *circline-set* $H \cap$ *circline-set x-axis*
then obtain x **where** $**$: *of-complex* $x \in$ *unit-disc of-complex* $x \in$ *circline-set* $H \cap$ *circline-set x-axis*
by (*metis inf-or-of-complex inf-notin-unit-disc*)
hence *is-real* x
unfolding *circline-set-x-axis*
using *of-complex-inj*
by *auto*
hence *eq*: $A * (Re\ x)^2 + 2 * Re\ B * Re\ x + A = 0$
using $**$
using *x-axis-intersection-equation*[*OF* $*(1-2)$, *of Re x*]
by *simp*
hence $(2 * Re\ B)^2 - 4 * A * A \geq 0$
using *discriminant-iff*[*of A - 2 * Re B A*]
using *discrim-def*[*of A 2 * Re B A*] *False*
by *auto*
hence $(Re\ B)^2 \geq (Re\ A)^2$
by (*simp add*: *power2-eq-square*)
moreover
have $(Re\ B)^2 \neq (Re\ A)^2$
proof (*rule ccontr*)
assume $\neg$ *?thesis*
hence $Re\ B = Re\ A \vee Re\ B = -\ Re\ A$
using *power2-eq-iff* **by** *blast*
hence $A * (Re\ x)^2 + \ A * 2 * Re\ x + A = 0 \vee A * (Re\ x)^2 - A * 2 * Re\ x + A = 0$
using *eq*
by *auto*
hence $A * ((Re\ x)^2 + \ 2 * Re\ x + 1) = 0 \vee A * ((Re\ x)^2 - 2 * Re\ x + 1) = 0$
by (*simp add*: *field-simps*)
hence $(Re\ x)^2 + 2 * Re\ x + 1 = 0 \vee (Re\ x)^2 - 2 * Re\ x + 1 = 0$
using $\langle A \neq 0\rangle$
by *simp*
hence $(Re\ x + 1)^2 = 0 \vee (Re\ x - 1)^2 = 0$
by (*simp add*: *power2-sum power2-diff field-simps*)
hence $Re\ x = -1 \vee Re\ x = 1$
by *auto*
thus *False*
using ‹*is-real x*› ‹*of-complex x* $\in$ *unit-disc*›
by (*auto simp add*: *cmod-eq-Re*)
qed
ultimately
show *intersects-x-axis H*
using *intersects-x-axis-mk-circline*
using $*$
by *auto*
next
assume *intersects-x-axis H*
hence $(Re\ B)^2 > (Re\ A)^2$
using $*$ *False*
using *intersects-x-axis-mk-circline*
by *simp*
hence *discr*: $(2 * Re\ B)^2 - 4 * A * A > 0$
by (*simp add*: *power2-eq-square*)
then obtain *x1 x2* **where**
eqs: $A * x1^2 + 2 * Re\ B * x1 + A = 0\ A * x2^2 + 2 * Re\ B * x2 + A = 0\ x1 \neq x2$
using *discriminant-pos-ex*[*OF* $\langle A \neq 0\rangle$, *of* $2 * Re\ B\ A$]

```
      using discrim-def[of A 2 * Re B A]
      by auto
    hence x1 * x2 = 1
      using viette2[OF ‹A ≠ 0›, of 2 * Re B A x1 x2] discr ‹A ≠ 0›
      by auto
    have abs x1 ≠ 1 abs x2 ≠ 1
      using eqs discr ‹x1 * x2 = 1›
      by (auto simp add: abs-if power2-eq-square)
    hence abs x1 < 1 ∨ abs x2 < 1
      using ‹x1 * x2 = 1›
      by (smt mult-le-cancel-left1 mult-minus-right)
    thus ∃ x ∈ unit-disc. x ∈ circline-set H ∩ circline-set x-axis
      using x-axis-intersection-equation[OF *(1−2), of x1]
      using x-axis-intersection-equation[OF *(1−2), of x2]
      using eqs
      by auto
  qed
 qed
qed
```

8.3 Check if a Poincaré line intersects the y-axis

```
definition intersects-y-axis-cmat :: complex-mat ⇒ bool where
 [simp]: intersects-y-axis-cmat H = (let (A, B, C, D) = H in A = 0 ∨ (Im B)^2 > (Re A)^2)

lift-definition intersects-y-axis-clmat :: circline-mat ⇒ bool is intersects-y-axis-cmat
 done

lift-definition intersects-y-axis :: circline ⇒ bool is intersects-y-axis-clmat
proof (transfer)
 fix H1 H2
 assume hh: hermitean H1 ∧ H1 ≠ mat-zero and hermitean H2 ∧ H2 ≠ mat-zero
 obtain A1 B1 C1 D1 A2 B2 C2 D2 where *: H1 = (A1, B1, C1, D1) H2 = (A2, B2, C2, D2)
  by (cases H1, cases H2, auto)
 assume circline-eq-cmat H1 H2
 then obtain k where k: k ≠ 0 ∧ H2 = cor k *sm H1
  by auto
 show intersects-y-axis-cmat H1 = intersects-y-axis-cmat H2
 proof−
  have k ≠ 0 ⟹ (Re A1)^2 < (Im B1)^2 ⟷ (k * Re A1)^2 < (k * Im B1)^2
   by (smt mult-strict-left-mono power2-eq-square semiring-normalization-rules(13) zero-less-power2)
  thus ?thesis
   using * k
   by auto
 qed
qed

lemma intersects-x-axis-intersects-y-axis [simp]:
 shows intersects-x-axis (moebius-circline (moebius-rotation (pi/2)) H) ⟷ intersects-y-axis H
 unfolding moebius-rotation-def moebius-similarity-def
 by simp (transfer, transfer, auto simp add: mat-adj-def mat-cnj-def)

lemma intersects-y-axis-iff:
 assumes is-poincare-line H
 shows (∃ y ∈ unit-disc. y ∈ circline-set H ∩ circline-set y-axis) ⟷ intersects-y-axis H (is ?lhs ⟷ ?rhs)
proof−
 let ?R = moebius-rotation (pi / 2)
 let ?H′ = moebius-circline ?R H
 have 1: is-poincare-line ?H′
  using assms
  using unit-circle-fix-preserve-is-poincare-line[OF - assms, of ?R]
  by simp

 show ?thesis
 proof
  assume ?lhs
```

```
    then obtain y where y ∈ unit-disc y ∈ circline-set H ∩ circline-set y-axis
      by auto
    hence moebius-pt ?R y ∈ unit-disc ∧ moebius-pt ?R y ∈ circline-set ?H' ∩ circline-set x-axis
      using rotation-pi-2-y-axis
     by (metis Int-iff circline-set-moebius-circline-E moebius-circline-comp-inv-left moebius-pt-comp-inv-left unit-disc-fix-discI
unit-disc-fix-rotation)
    thus ?rhs
      using intersects-x-axis-iff[OF 1]
      using intersects-x-axis-intersects-y-axis[of H]
      by auto
  next
    assume intersects-y-axis H
    hence intersects-x-axis ?H'
      using intersects-x-axis-intersects-y-axis[of H]
      by simp
    then obtain x where *: x ∈ unit-disc x ∈ circline-set ?H' ∩ circline-set x-axis
      using intersects-x-axis-iff[OF 1]
      by auto
    let ?y = moebius-pt (−?R) x
    have ?y ∈ unit-disc ∧ ?y ∈ circline-set H ∩ circline-set y-axis
      using * rotation-pi-2-y-axis[symmetric]
     by (metis Int-iff circline-set-moebius-circline-E moebius-pt-comp-inv-left moebius-rotation-uminus uminus-moebius-def
unit-disc-fix-discI unit-disc-fix-rotation)
    thus ?lhs
      by auto
  qed
qed
```

8.4 Intersection point of a Poincaré line with the x-axis in the unit disc

```
definition calc-x-axis-intersection-cvec :: complex ⇒ complex ⇒ complex-vec where
 [simp]: calc-x-axis-intersection-cvec A B =
    (let discr = (Re B)^2 − (Re A)^2 in
        (−Re(B) + sgn (Re B) * sqrt(discr), A))


definition calc-x-axis-intersection-cmat-cvec :: complex-mat ⇒ complex-vec where [simp]:
  calc-x-axis-intersection-cmat-cvec H =
    (let (A, B, C, D) = H in
       if A ≠ 0 then
          calc-x-axis-intersection-cvec A B
       else
          (0, 1)
    )

lift-definition calc-x-axis-intersection-clmat-hcoords :: circline-mat ⇒ complex-homo-coords is calc-x-axis-intersection-cmat-cvec
  by (auto split: if-split-asm)

lift-definition calc-x-axis-intersection :: circline ⇒ complex-homo is calc-x-axis-intersection-clmat-hcoords
proof transfer
  fix H1 H2
  assume *: hermitean H1 ∧ H1 ≠ mat-zero hermitean H2 ∧ H2 ≠ mat-zero
  obtain A1 B1 C1 D1 A2 B2 C2 D2 where hh: H1 = (A1, B1, C1, D1) H2 = (A2, B2, C2, D2)
    by (cases H1, cases H2, auto)
  assume circline-eq-cmat H1 H2
  then obtain k where k: k ≠ 0 H2 = cor k *sm H1
    by auto

  have calc-x-axis-intersection-cvec A1 B1 ≈v calc-x-axis-intersection-cvec A2 B2
    using hh k
    apply simp
    apply (rule-tac x=cor k in exI)
    apply auto
    apply (simp add: sgn-mult power-mult-distrib)
    apply (subst right-diff-distrib[symmetric])
    apply (subst real-sqrt-mult)
```

by (*simp add*: *real-sgn-eq right-diff-distrib*)

thus *calc-x-axis-intersection-cmat-cvec H1* $\approx_v$
calc-x-axis-intersection-cmat-cvec H2
using *hh k*
by (*auto simp del*: *calc-x-axis-intersection-cvec-def*)
qed

lemma *calc-x-axis-intersection-in-unit-disc*:
assumes *is-poincare-line H intersects-x-axis H*
shows *calc-x-axis-intersection H* $\in$ *unit-disc*
proof (*cases is-line H*)
case *True*
thus *?thesis*
using *assms*
unfolding *unit-disc-def disc-def*
by *simp* (*transfer*, *transfer*, *auto simp add*: *vec-cnj-def*)
next
case *False*
thus *?thesis*
using *assms*
unfolding *unit-disc-def disc-def*
proof (*simp*, *transfer*, *transfer*)
fix *H*
assume *hh*: *hermitean H* $\wedge$ *H* $\neq$ *mat-zero*
then obtain *A B D* **where** $*$: *H* = (*A*, *B*, *cnj B*, *D*) *is-real A is-real D*
using *hermitean-elems*
by (*cases H*) *blast*
assume *is-poincare-line-cmat H*
hence $*$: *H* = (*A*, *B*, *cnj B*, *A*) *is-real A*
using $*$
by *auto*

assume $\neg$ *circline-A0-cmat H*
hence *A* $\neq$ *0*
using $*$
by *simp*

assume *intersects-x-axis-cmat H*
hence $(Re\ B)^2 > (Re\ A)^2$
using $*$ ‹*A* $\neq$ *0*›
by (*auto simp add*: *power2-eq-square complex.expand*)

hence *Re B* $\neq$ *0*
by *auto*

have *Re A* $\neq$ *0*
using ‹*is-real A*› ‹*A* $\neq$ *0*›
by (*auto simp add*: *complex.expand*)

have $sqrt((Re\ B)^2 - (Re\ A)^2) < sqrt((Re\ B)^2)$
using ‹*Re A* $\neq$ *0*›
by (*subst real-sqrt-less-iff*) *auto*
also have ... = *sgn* (*Re B*) $*$ (*Re B*)
by (*smt mult-minus-right nonzero-eq-divide-eq real-sgn-eq real-sqrt-abs*)
finally
have *1*: $sqrt((Re\ B)^2 - (Re\ A)^2) < sgn\ (Re\ B) * (Re\ B)$
.

have *2*: $(Re\ B)^2 - (Re\ A)^2 < sgn\ (Re\ B) * (Re\ B) * sqrt((Re\ B)^2 - (Re\ A)^2)$
using ‹$(Re\ B)^2 > (Re\ A)^2$›
using *mult-strict-right-mono*[*OF 1*, *of sqrt* $((Re\ B)^2 - (Re\ A)^2)$]
by *simp*

have *3*: $(Re\ B)^2 - 2*sgn\ (Re\ B)*Re\ B*sqrt((Re\ B)^2 - (Re\ A)^2) + (Re\ B)^2 - (Re\ A)^2 < (Re\ A)^2$

```
      using mult-strict-left-mono[OF 2, of 2]
      by (simp add: field-simps)

    have (sgn (Re B))^2 = 1
      using ‹Re B ≠ 0›
      by (simp add: sgn-if)

    hence (−Re B + sgn (Re B) ∗ sqrt((Re B)^2 − (Re A)^2))^2 < (Re A)^2
      using ‹(Re B)^2 > (Re A)^2› 3
      by (simp add: power2-diff field-simps)

    thus in-ocircline-cmat-cvec unit-circle-cmat (calc-x-axis-intersection-cmat-cvec H)
      using ∗ ‹(Re B)^2 > (Re A)^2›
      by (auto simp add: vec-cnj-def power2-eq-square split: if-split-asm)
  qed
qed


lemma calc-x-axis-intersection:
  assumes is-poincare-line H and intersects-x-axis H
  shows calc-x-axis-intersection H ∈ circline-set H ∩ circline-set x-axis
proof (cases is-line H)
  case True
  thus ?thesis
    using assms
    unfolding circline-set-def
    by simp (transfer, transfer, auto simp add: vec-cnj-def)
next
  case False
  thus ?thesis
    using assms
    unfolding circline-set-def
  proof (simp, transfer, transfer)
    fix H
    assume hh: hermitean H ∧ H ≠ mat-zero
    then obtain A B D where ∗: H = (A, B, cnj B, D) is-real A is-real D
      using hermitean-elems
      by (cases H) blast
    assume is-poincare-line-cmat H
    hence ∗: H = (A, B, cnj B, A) is-real A
      using ∗
      by auto
    assume ¬ circline-A0-cmat H
    hence A ≠ 0
      using ∗
      by auto

    assume intersects-x-axis-cmat H
    hence (Re B)^2 > (Re A)^2
      using ∗ ‹A ≠ 0›
      by (auto simp add: power2-eq-square complex.expand)

    hence Re B ≠ 0
      by auto

    show on-circline-cmat-cvec H (calc-x-axis-intersection-cmat-cvec H) ∧
        on-circline-cmat-cvec x-axis-cmat (calc-x-axis-intersection-cmat-cvec H) (is ?P1 ∧ ?P2)
    proof
      show on-circline-cmat-cvec H (calc-x-axis-intersection-cmat-cvec H)
      proof (cases circline-A0-cmat H)
        case True
        thus ?thesis
          using ∗ ‹is-poincare-line-cmat H› ‹intersects-x-axis-cmat H›
          by (simp add: vec-cnj-def)
      next
        case False
```

```
let ?x = calc-x-axis-intersection-cvec A B
let ?nom = fst ?x and ?den = snd ?x
have x: ?x = (?nom, ?den)
  by simp

hence on-circline-cmat-cvec H (calc-x-axis-intersection-cvec A B)
proof (subst *, subst x, subst on-circline-cmat-cvec-circline-equation)
  have (sgn(Re B))^2 = 1
    using ‹Re B ≠ 0› sgn-pos zero-less-power2 by fastforce
  have (sqrt((Re B)^2 − (Re A)^2))^2 = (Re B)^2 − (Re A)^2
    using ‹(Re B)^2 > (Re A)^2›
    by simp

  have (−(Re B) + sgn(Re B)∗sqrt((Re B)^2 − (Re A)^2))^2 =
       (−(Re B))^2 + (sgn(Re B)∗sqrt((Re B)^2 − (Re A)^2))^2 − 2∗(Re B)∗sgn(Re B)∗sqrt((Re B)^2 − (Re A)^2)
    by (simp add: power2-diff)
  also have ... = (Re B)∗(Re B) + (sgn(Re B)∗sqrt((Re B)^2 − (Re A)^2))^2 − 2∗(Re B)∗sgn(Re B)∗sqrt((Re B)^2
− (Re A)^2)
    by (simp add: power2-eq-square)
  also have ... = (Re B)∗(Re B) + (sgn(Re B))^2∗(sqrt((Re B)^2 − (Re A)^2))^2 − 2∗(Re B)∗sgn(Re B)∗sqrt((Re
B)^2 − (Re A)^2)
    by (simp add: power-mult-distrib)
  also have ... = (Re B)∗(Re B) + (Re B)^2 − (Re A)^2 − 2∗(Re B)∗sgn(Re B)∗sqrt((Re B)^2 − (Re A)^2)
    using ‹(sqrt((Re B)^2 − (Re A)^2))^2 = (Re B)^2 − (Re A)^2› ‹(sgn(Re B))^2 = 1›
    by simp
  finally have (−(Re B) + sgn(Re B)∗sqrt((Re B)^2 − (Re A)^2))^2 =
            (Re B)∗(Re B) + (Re B)^2 − (Re A)^2 − 2∗(Re B)∗sgn(Re B)∗sqrt((Re B)^2 − (Re A)^2)
    by simp

  have is-real ?nom is-real ?den
    using ‹is-real A›
    by simp+
  hence cnj (?nom) = ?nom cnj (?den) = ?den
    by (simp add:eq-cnj-iff-real)+
  hence A∗?nom∗(cnj (?nom)) + B∗?den∗(cnj (?nom)) + (cnj B)∗(cnj (?den))∗?nom + A∗?den∗(cnj (?den))
      = A∗?nom∗?nom + B∗?den∗?nom + (cnj B)∗?den∗?nom + A∗?den∗?den
    by auto
  also have ... = A∗?nom∗?nom + (B + (cnj B))∗?den∗?nom + A∗?den∗?den
    by (simp add:field-simps)
  also have ... = A∗?nom∗?nom + 2∗(Re B)∗?den∗?nom + A∗?den∗?den
    by (simp add:complex-add-cnj)
  also have ... = A∗?nom^2 + 2∗(Re B)∗?den∗?nom + A∗?den∗?den
    by (simp add:power2-eq-square)
  also have ... = A∗(−(Re B) + sgn(Re B)∗sqrt((Re B)^2 − (Re A)^2))^2
               + 2∗(Re B)∗A∗(−(Re B) + sgn(Re B)∗sqrt((Re B)^2 − (Re A)^2)) + A∗A∗A
    unfolding calc-x-axis-intersection-cvec-def
    by auto
  also have ... = A∗((Re B)∗(Re B) + (Re B)^2 − (Re A)^2 − 2∗(Re B)∗sgn(Re B)∗sqrt((Re B)^2 − (Re A)^2))
          + 2∗(Re B)∗A∗(−(Re B) + sgn(Re B)∗sqrt((Re B)^2 − (Re A)^2)) + A∗A∗A
    using ‹(−(Re B) + sgn(Re B)∗sqrt((Re B)^2 − (Re A)^2))^2 =
            (Re B)∗(Re B) + (Re B)^2 − (Re A)^2 − 2∗(Re B)∗sgn(Re B)∗sqrt((Re B)^2 − (Re A)^2)›
    by simp
  also have ... = A∗((Re B)∗(Re B) + (Re B)^2 − A^2 − 2∗(Re B)∗sgn(Re B)∗sqrt((Re B)^2 − (Re A)^2))
          + 2∗(Re B)∗A∗(−(Re B) + sgn(Re B)∗sqrt((Re B)^2 − (Re A)^2)) + A∗A∗A
    using ‹is-real A›
    by simp
  also have ... = 0
    apply (simp add:field-simps)
    by (simp add: power2-eq-square)
  finally have A∗?nom∗(cnj (?nom)) + B∗?den∗(cnj (?nom)) + (cnj B)∗(cnj (?den))∗?nom + A∗?den∗(cnj
(?den)) = 0
    by simp
  thus circline-equation A B (cnj B) A ?nom ?den
    by simp
qed
thus ?thesis
```

```
        using ∗ ‹is-poincare-line-cmat H› ‹intersects-x-axis-cmat H›
        by (simp add: vec-cnj-def)
      qed
    next
      show on-circline-cmat-cvec x-axis-cmat (calc-x-axis-intersection-cmat-cvec H)
        using ∗ ‹is-poincare-line-cmat H› ‹intersects-x-axis-cmat H› ‹is-real A›
        using eq-cnj-iff-real[of A]
        by (simp add: vec-cnj-def)
    qed
  qed
qed

lemma unique-calc-x-axis-intersection:
  assumes is-poincare-line H and H ≠ x-axis
  assumes x ∈ unit-disc and x ∈ circline-set H ∩ circline-set x-axis
  shows x = calc-x-axis-intersection H
proof−
  have ∗: intersects-x-axis H
    using assms
    using intersects-x-axis-iff[OF assms(1)]
    by auto
  show x = calc-x-axis-intersection H
    using calc-x-axis-intersection[OF assms(1) ∗]
    using calc-x-axis-intersection-in-unit-disc[OF assms(1) ∗]
    using assms
    using unique-is-poincare-line[of x calc-x-axis-intersection H H x-axis]
    by auto
qed
```

8.5 Check if an h-line intersects the positive part of the x-axis

```
definition intersects-x-axis-positive-cmat :: complex-mat ⇒ bool where
  [simp]: intersects-x-axis-positive-cmat H = (let (A, B, C, D) = H in Re A ≠ 0 ∧ Re B / Re A < −1)

lift-definition intersects-x-axis-positive-clmat :: circline-mat ⇒ bool is intersects-x-axis-positive-cmat
  done

lift-definition intersects-x-axis-positive :: circline ⇒ bool is intersects-x-axis-positive-clmat
proof (transfer)
  fix H1 H2
  assume hh: hermitean H1 ∧ H1 ≠ mat-zero and hermitean H2 ∧ H2 ≠ mat-zero
  obtain A1 B1 C1 D1 A2 B2 C2 D2 where ∗: H1 = (A1, B1, C1, D1) H2 = (A2, B2, C2, D2)
    by (cases H1, cases H2, auto)
  assume circline-eq-cmat H1 H2
  then obtain k where k ≠ 0 ∧ H2 = cor k ∗sm H1
    by auto
  thus intersects-x-axis-positive-cmat H1 = intersects-x-axis-positive-cmat H2
    using ∗
    by simp
qed

lemma intersects-x-axis-positive-mk-circline:
  assumes is-real A and A ≠ 0 ∨ B ≠ 0
  shows intersects-x-axis-positive (mk-circline A B (cnj B) A) ←→ Re B / Re A < −1
proof−
  let ?H = (A, B, (cnj B), A)
  have hermitean ?H
    using ‹is-real A›
    unfolding hermitean-def mat-adj-def mat-cnj-def
    using eq-cnj-iff-real
    by auto
  moreover
  have ?H ≠ mat-zero
    using assms
    by auto
  ultimately
```

show *?thesis*
by (*transfer*, *transfer*, *auto simp add*: *Let-def*)
qed

lemma *intersects-x-axis-positive-intersects-x-axis* [*simp*]:
assumes *intersects-x-axis-positive H*
shows *intersects-x-axis H*
proof−
have $\bigwedge$ *a aa*. $[\![$ *Re a* $\neq$ *0*; *Re aa* / *Re a* < − *1*; ¬ $(Re\ a)^2 < (Re\ aa)^2$ $]\!] \Longrightarrow$ *aa* = *0* ∧ *a* = *0*
by (*smt less-divide-eq-1-pos one-less-power pos2 power2-minus power-divide zero-less-power2*)
thus *?thesis*
using *assms*
apply *transfer*
apply *transfer*
apply (*auto simp add*: *hermitean-def mat-adj-def mat-cnj-def*)
done
qed

lemma *add-less-abs-positive-iff*:
fixes *a b* :: *real*
assumes *abs b* < *abs a*
shows *a* + *b* > *0* ⟷ *a* > *0*
using *assms*
by *auto*

lemma *calc-x-axis-intersection-positive-abs'*:
fixes *A B* :: *real*
assumes $B^2 > A^2$ **and** *A* ≠ *0*
shows *abs* (*sgn*(*B*) ∗ *sqrt*($B^2 - A^2$) / *A*) < *abs*(−*B*/*A*)
proof−
from *assms* **have** *B* ≠ *0*
by *auto*

have $B^2 - A^2 < B^2$
using ‹*A* ≠ *0*›
by *auto*
hence *sqrt* ($B^2 - A^2$) < *abs B*
using *real-sqrt-less-iff*[*of* $B^2 - A^2$ B^2]
by *simp*
thus *?thesis*
using *assms* ‹*B* ≠ *0*›
by (*simp add*: *abs-mult divide-strict-right-mono*)
qed

lemma *calc-intersect-x-axis-positive-lemma*:
assumes $B^2 > A^2$ **and** *A* ≠ *0*
shows (−*B* + *sgn B* ∗ *sqrt*($B^2 - A^2$)) / *A* > *0* ⟷ −*B*/*A* > *1*
proof−
have (−*B* + *sgn B* ∗ *sqrt*($B^2 - A^2$)) / *A* = −*B* / *A* + (*sgn B* ∗ *sqrt*($B^2 - A^2$)) / *A*
using *assms*
by (*simp add*: *field-simps*)
moreover
have −*B* / *A* + (*sgn B* ∗ *sqrt*($B^2 - A^2$)) / *A* > *0* ⟷ − *B* / *A* > *0*
using *add-less-abs-positive-iff*[*OF calc-x-axis-intersection-positive-abs'*[*OF assms*]]
by *simp*
moreover
hence $(B/A)^2 > 1$
using *assms*
by (*simp add*: *power-divide*)
hence *B*/*A* > *1* ∨ *B*/*A* < −*1*
by (*smt one-power2 pos2 power2-minus power-0 power-strict-decreasing zero-power2*)
hence −*B* / *A* > *0* ⟷ −*B* / *A* > *1*
by *auto*
ultimately
show *?thesis*

```
    using assms
    by auto
qed

lemma intersects-x-axis-positive-iff':
  assumes is-poincare-line H
  shows intersects-x-axis-positive H ⟷
        calc-x-axis-intersection H ∈ unit-disc ∧ calc-x-axis-intersection H ∈ circline-set H ∩ positive-x-axis (is ?lhs ⟷
?rhs)
proof
  let ?x = calc-x-axis-intersection H
  assume ?lhs
  hence ?x ∈ circline-set x-axis ?x ∈ circline-set H ?x ∈ unit-disc
    using calc-x-axis-intersection-in-unit-disc[OF assms] calc-x-axis-intersection[OF assms]
    by auto
  moreover
  have Re (to-complex ?x) > 0
    using ‹?lhs› assms
  proof (transfer, transfer)
    fix H
    assume hh: hermitean H ∧ H ≠ mat-zero
    obtain A B C D where *: H = (A, B, C, D)
      by (cases H, auto)
    assume intersects-x-axis-positive-cmat H
    hence **: Re B / Re A < − 1 Re A ≠ 0
      using *
      by auto
    have (Re B)² > (Re A)²
      using **
      by (smt divide-less-eq-1-neg divide-minus-left less-divide-eq-1-pos real-sqrt-abs real-sqrt-less-iff right-inverse-eq)
    have is-real A A ≠ 0
      using hh hermitean-elems * ‹Re A ≠ 0› complex.expand[of A 0]
      by auto
    have (cmod B)² > (cmod A)²
      using ‹(Re B)² > (Re A)²› ‹is-real A›
      by (smt cmod-power2 power2-less-0 zero-power2)
    have ***: 0 < (− Re B + sgn (Re B) * sqrt ((Re B)² − (Re A)²)) / Re A
      using calc-intersect-x-axis-positive-lemma[of Re A Re B] ** ‹(Re B)² > (Re A)²›
      by auto

    assume is-poincare-line-cmat H
    hence A = D
      using * hh
      by simp

    have Re ((cor (sgn (Re B)) * cor (sqrt ((Re B)² − (Re A)²)) − cor (Re B)) / A) = (sgn (Re B) * sqrt ((Re B)²
− (Re A)²) − Re B) / Re D
      using ‹is-real A› ‹A = D›
      by (metis (no-types, lifting) Re-complex-of-real complex-of-real-Re of-real-diff of-real-divide of-real-mult)
    thus 0 < Re (to-complex-cvec (calc-x-axis-intersection-cmat-cvec H))
      using * hh ** *** ‹(cmod B)² > (cmod A)²› ‹(Re B)² > (Re A)²› ‹A ≠ 0› ‹A = D›
      by simp
  qed
  ultimately
  show ?rhs
    unfolding positive-x-axis-def
    by auto
next
  let ?x = calc-x-axis-intersection H
  assume ?rhs
  hence Re (to-complex ?x) > 0  ?x ≠ ∞h  ?x ∈ circline-set x-axis ?x ∈ unit-disc ?x ∈ circline-set H
    unfolding positive-x-axis-def
    by auto
  hence intersects-x-axis H
    using intersects-x-axis-iff[OF assms]
    by auto
```

thus *?lhs*
using ‹*Re* (*to-complex ?x*) > *0*› *assms*
proof (*transfer*, *transfer*)
fix *H*
assume *hh*: *hermitean H* ∧ *H* ≠ *mat-zero*
obtain *A B C D* **where** ∗: *H* = (*A*, *B*, *C*, *D*)
by (*cases H*, *auto*)
assume $0 < Re$ (*to-complex-cvec* (*calc-x-axis-intersection-cmat-cvec H*)) *intersects-x-axis-cmat H is-poincare-line-cmat H*
hence ∗∗: $A \neq 0$ $0 < Re\ ((cor\ (sgn\ (Re\ B)) * cor\ (sqrt\ ((Re\ B)^2 - (Re\ A)^2)) - cor\ (Re\ B))\ /\ A)$ $A = D$ *is-real A* $(Re\ B)^2 > (Re\ A)^2$
using ∗ *hh hermitean-elems*
by (*auto split*: *if-split-asm*)

have *Re A* ≠ *0*
using *complex.expand*[*of A 0*] ‹*A* ≠ *0*› ‹*is-real A*›
by *auto*

have $Re\ ((cor\ (sgn\ (Re\ B)) * cor\ (sqrt\ ((Re\ B)^2 - (Re\ D)^2)) - cor\ (Re\ B))\ /\ D) = (sgn\ (Re\ B) * sqrt\ ((Re\ B)^2 - (Re\ D)^2) - Re\ B)\ /\ Re\ D$
using ‹*is-real A*› ‹*A* = *D*›
by (*metis* (*no-types*, *lifting*) *Re-complex-of-real complex-of-real-Re of-real-diff of-real-divide of-real-mult*)

thus *intersects-x-axis-positive-cmat H*
using ∗ ∗∗ ‹*Re A* ≠ *0*›
using *calc-intersect-x-axis-positive-lemma*[*of Re A Re B*]
by *simp*
qed
qed

lemma *intersects-x-axis-positive-iff*:
assumes *is-poincare-line H* **and** *H* ≠ *x-axis*
shows *intersects-x-axis-positive H* ⟷
(∃ *x*. *x* ∈ *unit-disc* ∧ *x* ∈ *circline-set H* ∩ *positive-x-axis*) (**is** *?lhs* ⟷ *?rhs*)
proof
assume *?lhs*
thus *?rhs*
using *intersects-x-axis-positive-iff'*[*OF assms*(*1*)]
by *auto*
next
assume *?rhs*
then obtain *x* **where** *x* ∈ *unit-disc x* ∈ *circline-set H* ∩ *positive-x-axis*
by *auto*
thus *?lhs*
using *unique-calc-x-axis-intersection*[*OF assms*, *of x*]
using *intersects-x-axis-positive-iff'*[*OF assms*(*1*)]
unfolding *positive-x-axis-def*
by *auto*
qed

8.6 Check if an h-line intersects the positive part of the y-axis

definition *intersects-y-axis-positive-cmat* :: *complex-mat* ⇒ *bool* **where**
[*simp*]: *intersects-y-axis-positive-cmat H* = (*let* (*A*, *B*, *C*, *D*) = *H in Re A* ≠ *0* ∧ *Im B* / *Re A* < −*1*)

lift-definition *intersects-y-axis-positive-clmat* :: *circline-mat* ⇒ *bool* **is** *intersects-y-axis-positive-cmat*
done

lift-definition *intersects-y-axis-positive* :: *circline* ⇒ *bool* **is** *intersects-y-axis-positive-clmat*
proof (*transfer*)
fix *H1 H2*
assume *hh*: *hermitean H1* ∧ *H1* ≠ *mat-zero* **and** *hermitean H2* ∧ *H2* ≠ *mat-zero*
obtain *A1 B1 C1 D1 A2 B2 C2 D2* **where** ∗: *H1* = (*A1*, *B1*, *C1*, *D1*) *H2* = (*A2*, *B2*, *C2*, *D2*)
by (*cases H1*, *cases H2*, *auto*)
assume *circline-eq-cmat H1 H2*
then obtain *k* **where** $k \neq 0 \wedge H2 = cor\ k *_{sm} H1$

```
    by auto
  thus intersects-y-axis-positive-cmat H1 = intersects-y-axis-positive-cmat H2
    using *
    by simp
qed

lemma intersects-x-axis-positive-intersects-y-axis-positive [simp]:
  shows intersects-x-axis-positive (moebius-circline (moebius-rotation (−pi/2)) H) ⟷ intersects-y-axis-positive H
  using hermitean-elems
  unfolding moebius-rotation-def moebius-similarity-def
  by simp (transfer, transfer, auto simp add: mat-adj-def mat-cnj-def)

lemma intersects-y-axis-positive-iff:
  assumes is-poincare-line H H ≠ y-axis
  shows (∃ y ∈ unit-disc. y ∈ circline-set H ∩ positive-y-axis) ⟷ intersects-y-axis-positive H (is ?lhs ⟷ ?rhs)
proof−
  let ?R = moebius-rotation (−pi / 2)
  let ?H' = moebius-circline ?R H
  have 1: is-poincare-line ?H'
    using assms
    using unit-circle-fix-preserve-is-poincare-line[OF - assms(1), of ?R]
    by simp

  have 2: moebius-circline ?R H ≠ x-axis
  proof (rule ccontr)
    assume ¬ ?thesis
    hence H = moebius-circline (moebius-rotation (pi/2)) x-axis
      using moebius-circline-comp-inv-left[of ?R H]
      by auto
    thus False
      using ‹H ≠ y-axis›
      by auto
  qed

  show ?thesis
  proof
    assume ?lhs
    then obtain y where y ∈ unit-disc y ∈ circline-set H ∩ positive-y-axis
      by auto
    hence moebius-pt ?R y ∈ unit-disc moebius-pt ?R y ∈ circline-set ?H' ∩ positive-x-axis
      using rotation-minus-pi-2-positive-y-axis
      by auto
    thus ?rhs
      using intersects-x-axis-positive-iff[OF 1 2]
      using intersects-x-axis-positive-intersects-y-axis-positive[of H]
      by auto
  next
    assume intersects-y-axis-positive H
    hence intersects-x-axis-positive ?H'
      using intersects-x-axis-positive-intersects-y-axis-positive[of H]
      by simp
    then obtain x where *: x ∈ unit-disc x ∈ circline-set ?H' ∩ positive-x-axis
      using intersects-x-axis-positive-iff[OF 1 2]
      by auto
    let ?y = moebius-pt (−?R) x
    have ?y ∈ unit-disc ∧ ?y ∈ circline-set H ∩ positive-y-axis
      using * rotation-minus-pi-2-positive-y-axis[symmetric]
      by (metis Int-iff circline-set-moebius-circline-E image-eqI moebius-pt-comp-inv-image-left moebius-rotation-uminus
uminus-moebius-def unit-disc-fix-discI unit-disc-fix-rotation)
    thus ?lhs
      by auto
  qed
qed
```

8.7 Position of the intersection point in the unit disc

Check if the intersection point of one h-line with the x-axis is located more outward the edge of the disc than the intersection point of another h-line.

definition *outward-cmat* :: *complex-mat* ⇒ *complex-mat* ⇒ *bool* **where**
[*simp*]: *outward-cmat H1 H2* = (*let* (*A1*, *B1*, *C1*, *D1*) = *H1*; (*A2*, *B2*, *C2*, *D2*) = *H2*
in −*Re B1*/*Re A1* ≤ −*Re B2*/*Re A2*)
lift-definition *outward-clmat* :: *circline-mat* ⇒ *circline-mat* ⇒ *bool* **is** *outward-cmat*
done
lift-definition *outward* :: *circline* ⇒ *circline* ⇒ *bool* **is** *outward-clmat*
apply *transfer*
apply *simp*
apply (*case-tac circline-mat1*, *case-tac circline-mat2*, *case-tac circline-mat3*, *case-tac circline-mat4*)
apply *simp*
apply (*erule-tac exE*)+
apply (*erule-tac conjE*)+
apply *simp*
done

lemma *outward-mk-circline*:
assumes *is-real A1* **and** *is-real A2* **and** *A1* ≠ *0* ∨ *B1* ≠ *0* **and** *A2* ≠ *0* ∨ *B2* ≠ *0*
shows *outward* (*mk-circline A1 B1* (*cnj B1*) *A1*) (*mk-circline A2 B2* (*cnj B2*) *A2*) ⟷ − *Re B1* / *Re A1* ≤ − *Re B2* / *Re A2*
proof−
let *?H1* = (*A1*, *B1*, (*cnj B1*), *A1*)
let *?H2* = (*A2*, *B2*, (*cnj B2*), *A2*)
have *hermitean ?H1 hermitean ?H2*
using ‹*is-real A1*› ‹*is-real A2*›
unfolding *hermitean-def mat-adj-def mat-cnj-def*
using *eq-cnj-iff-real*
by *auto*
moreover
have *?H1* ≠ *mat-zero ?H2* ≠ *mat-zero*
using *assms*
by *auto*
ultimately
show *?thesis*
by (*transfer*, *transfer*, *auto simp add*: *Let-def*)
qed

lemma *calc-x-axis-intersection-fun-mono*:
fixes *x1 x2* :: *real*
assumes *x1* > *1* **and** *x2* > *x1*
shows $x1 - sqrt(x1^2 - 1) > x2 - sqrt(x2^2 - 1)$
using *assms*
proof−
have ∗: $sqrt(x1^2 - 1) + sqrt(x2^2 - 1) > 0$
using *assms*
by (*smt one-less-power pos2 real-sqrt-gt-zero*)

have $sqrt(x1^2 - 1) < x1$
using *real-sqrt-less-iff*[*of* $x1^2 - 1$ $x1^2$] ‹*x1* > *1*›
by *auto*
moreover
have $sqrt(x2^2 - 1) < x2$
using *real-sqrt-less-iff*[*of* $x2^2 - 1$ $x2^2$] ‹*x1* > *1*› ‹*x2* > *x1*›
by *auto*
ultimately
have $sqrt(x1^2 - 1) + sqrt(x2^2 - 1) < x1 + x2$
by *simp*
hence $(x1 + x2) / (sqrt(x1^2 - 1) + sqrt(x2^2 - 1)) > 1$
using ∗
using *less-divide-eq-1-pos*[*of* $sqrt(x1^2 - 1) + sqrt(x2^2 - 1)$ *x1* + *x2*]
by *simp*
hence $(x2^2 - x1^2) / (sqrt(x1^2 - 1) + sqrt(x2^2 - 1)) > x2 - x1$
using ‹*x2* > *x1*›

```
    using mult-less-cancel-left-pos[of x2 − x1 1 (x2 + x1) / (sqrt(x1^2 − 1) + sqrt(x2^2 − 1))]
    by (simp add: power2-eq-square field-simps)
  moreover
  have (x2^2 − x1^2) = (sqrt(x1^2 − 1) + sqrt(x2^2 − 1)) ∗ ((sqrt(x2^2 − 1) − sqrt(x1^2 − 1)))
    using ‹x1 > 1› ‹x2 > x1›
    by (simp add: field-simps)
  ultimately
  have sqrt(x2^2 − 1) − sqrt(x1^2 − 1) > x2 − x1
    using ∗
    by simp
  thus ?thesis
    by simp
qed

lemma calc-x-axis-intersection-mono:
  fixes a1 b1 a2 b2 :: real
  assumes −b1/a1 > 1 and a1 ≠ 0 and −b2/a2 ≥ −b1/a1 and a2 ≠ 0
  shows (−b1 + sgn b1 ∗ sqrt(b1^2 − a1^2)) / a1 ≥ (−b2 + sgn b2 ∗ sqrt(b2^2 − a2^2)) / a2 (is ?lhs ≥ ?rhs)
proof−
  have ?lhs = −b1/a1 − sqrt((−b1/a1)^2 − 1)
  proof (cases b1 > 0)
    case True
    hence a1 < 0
      using assms
      by (smt divide-neg-pos)
    thus ?thesis
      using ‹b1 > 0› ‹a1 < 0›
      by (simp add: real-sqrt-divide field-simps)
  next
    case False
    hence b1 < 0
      using assms
      by (cases b1 = 0) auto
    hence a1 > 0
      using assms
      by (smt divide-pos-neg)
    thus ?thesis
      using ‹b1 < 0› ‹a1 > 0›
      by (simp add: real-sqrt-divide field-simps)
  qed

  moreover

  have ?rhs = −b2/a2 − sqrt((−b2/a2)^2 − 1)
  proof (cases b2 > 0)
    case True
    hence a2 < 0
      using assms
      by (smt divide-neg-pos)
    thus ?thesis
      using ‹b2 > 0› ‹a2 < 0›
      by (simp add: real-sqrt-divide field-simps)
  next
    case False
    hence b2 < 0
      using assms
      by (cases b2 = 0) auto
    hence a2 > 0
      using assms
      by (smt divide-pos-neg)
    thus ?thesis
      using ‹b2 < 0› ‹a2 > 0›
      by (simp add: real-sqrt-divide field-simps)
  qed

  ultimately
```

show *?thesis*
using *calc-x-axis-intersection-fun-mono*[*of* $-b1/a1$ $-b2/a2$]
using *assms*
by (*cases* $-b1/a1=-b2/a2$, *auto*)
qed

lemma *outward*:
assumes *is-poincare-line H1* **and** *is-poincare-line H2*
assumes *intersects-x-axis-positive H1* **and** *intersects-x-axis-positive H2*
assumes *outward H1 H2*
shows *Re* (*to-complex* (*calc-x-axis-intersection H1*)) $\geq$ *Re* (*to-complex* (*calc-x-axis-intersection H2*))
proof−
have *intersects-x-axis H1 intersects-x-axis H2*
using *assms*
by *auto*
thus *?thesis*
using *assms*
proof (*transfer*, *transfer*)
fix *H1 H2*
assume *hh*: *hermitean H1* $\wedge$ *H1* $\neq$ *mat-zero* *hermitean H2* $\wedge$ *H2* $\neq$ *mat-zero*
obtain *A1 B1 C1 D1 A2 B2 C2 D2* **where** ∗: *H1* = (*A1*, *B1*, *C1*, *D1*) *H2* = (*A2*, *B2*, *C2*, *D2*)
by (*cases H1*, *cases H2*, *auto*)
have *is-real A1 is-real A2*
using *hermitean-elems* ∗ *hh*
by *auto*
assume *1*: *intersects-x-axis-positive-cmat H1 intersects-x-axis-positive-cmat H2*
assume *2*: *intersects-x-axis-cmat H1 intersects-x-axis-cmat H2*
assume *3*: *is-poincare-line-cmat H1 is-poincare-line-cmat H2*
assume *4*: *outward-cmat H1 H2*
have *A1* $\neq$ *0 A2* $\neq$ *0*
using ∗ ‹*is-real A1*› ‹*is-real A2*› *1 complex.expand*[*of A1 0*] *complex.expand*[*of A2 0*]
by *auto*
hence (*sgn* (*Re B2*) ∗ *sqrt* ((*Re B2*)^2 − (*Re A2*)^2) − *Re B2*) / *Re A2*
$\leq$ (*sgn* (*Re B1*) ∗ *sqrt* ((*Re B1*)^2 − (*Re A1*)^2) − *Re B1*) / *Re A1*
using *calc-x-axis-intersection-mono*[*of Re B1 Re A1 Re B2 Re A2*]
using *1 4* ∗
by *simp*
moreover
have (*sgn* (*Re B2*) ∗ *sqrt* ((*Re B2*)^2 − (*Re A2*)^2) − *Re B2*) / *Re A2* =
Re ((*cor* (*sgn* (*Re B2*)) ∗ *cor* (*sqrt* ((*Re B2*)^2 − (*Re A2*)^2)) − *cor* (*Re B2*)) / *A2*)
using ‹*is-real A2*› ‹*A2* $\neq$ *0*›
by (*simp add*: *Re-divide-real*)
moreover
have (*sgn* (*Re B1*) ∗ *sqrt* ((*Re B1*)^2 − (*Re A1*)^2) − *Re B1*) / *Re A1* =
Re ((*cor* (*sgn* (*Re B1*)) ∗ *cor* (*sqrt* ((*Re B1*)^2 − (*Re A1*)^2)) − *cor* (*Re B1*)) / *A1*)
using ‹*is-real A1*› ‹*A1* $\neq$ *0*›
by (*simp add*: *Re-divide-real*)
ultimately
show *Re* (*to-complex-cvec* (*calc-x-axis-intersection-cmat-cvec H2*))
$\leq$ *Re* (*to-complex-cvec* (*calc-x-axis-intersection-cmat-cvec H1*))
using *2 3* ‹*A1* $\neq$ *0*› ‹*A2* $\neq$ *0*› ∗ ‹*is-real A1*› ‹*is-real A2*›
by (*simp del*: *is-poincare-line-cmat-def intersects-x-axis-cmat-def*)
qed
qed

8.8 Ideal points and x-axis intersection

lemma *ideal-points-intersects-x-axis*:
assumes *is-poincare-line H* **and** *ideal-points H* = {*i1*, *i2*} **and** *H* $\neq$ *x-axis*
shows *intersects-x-axis H* $\longleftrightarrow$ *Im* (*to-complex i1*) ∗ *Im* (*to-complex i2*) < *0*
using *assms*
proof−
have *i1* $\neq$ *i2*
using *assms*(*1*) *assms*(*2*) *ex-poincare-line-points ideal-points-different*(*1*)
by *blast*

have *calc-ideal-points H* = {*i1*, *i2*}
using *assms*
using *ideal-points-unique*
by *auto*

have ∀ *i1* ∈ *calc-ideal-points H*.
∀ *i2* ∈ *calc-ideal-points H*.
is-poincare-line H ∧ *H* ≠ *x-axis* ∧ *i1* ≠ *i2* ⟶ (*Im* (*to-complex i1*) ∗ *Im* (*to-complex i2*) < *0* ⟷ *intersects-x-axis H*)
proof (*transfer*, *transfer*, (*rule ballI*)+, *rule impI*, (*erule conjE*)+, *case-tac H*, *case-tac i1*, *case-tac i2*)
fix *i11 i12 i21 i22 A B C D H i1 i2*
assume *H*: *H* = (*A*, *B*, *C*, *D*) *hermitean H H* ≠ *mat-zero*
assume *line*: *is-poincare-line-cmat H*
assume *i1*: *i1* = (*i11*, *i12*) *i1* ∈ *calc-ideal-points-cmat-cvec H*
assume *i2*: *i2* = (*i21*, *i22*) *i2* ∈ *calc-ideal-points-cmat-cvec H*
assume *different*: ¬ *i1* $\approx_v$ *i2*
assume *not-x-axis*: ¬ *circline-eq-cmat H x-axis-cmat*

have *is-real A is-real D C* = *cnj B*
using *H hermitean-elems*
by *auto*
have (*cmod A*)^2 < (*cmod B*)^2 *A* = *D*
using *line H*
by *auto*

let *?discr* = *sqrt* ((*cmod B*)^2 − (*Re D*)^2)
let *?den* = (*cmod B*)^2
let *?i1* = *B* ∗ (− *D* − i ∗ *?discr*)
let *?i2* = *B* ∗ (− *D* + i ∗ *?discr*)

have *i11* = *?i1* ∨ *i11* = *?i2 i12* = *?den*
i21 = *?i1* ∨ *i21* = *?i2 i22* = *?den*
using *i1 i2 H line*
by (*auto split*: *if-split-asm*)
hence *i*: *i11* = *?i1* ∧ *i21* = *?i2* ∨ *i11* = *?i2* ∧ *i21* = *?i1*
using ‹¬ *i1* $\approx_v$ *i2*› *i1 i2*
by *auto*

have *Im* (*i11* / *i12*) ∗ *Im* (*i21* / *i22*) = *Im* (*?i1* / *?den*) ∗ *Im* (*?i2* / *?den*)
using *i* ‹*i12* = *?den*› ‹*i22* = *?den*›
by *auto*
also have ... = *Im* (*?i1*) ∗ *Im* (*?i2*) / *?den*^{2}
by *simp*
also have ... = (*Im B* ∗ (*Im B* ∗ (*Re D* ∗ *Re D*)) − *Re B* ∗ (*Re B* ∗ ((*cmod B*)^2 − (*Re D*)^2))) / *cmod B* ^ *4*
using ‹(*cmod B*)^2 > (*cmod A*)^2› ‹*A* = *D*›
using ‹*is-real D*› *cmod-eq-Re*[*of A*]
by (*auto simp add*: *field-simps*)
also have ... = ((*Im B*)^2 ∗ (*Re D*)^2 − (*Re B*)^2 ∗ ((*Re B*)^2 + (*Im B*)^2 − (*Re D*)^2)) / *cmod B* ^ *4*
proof−
have *cmod B* ∗ *cmod B* = *Re B* ∗ *Re B* + *Im B* ∗ *Im B*
by (*metis cmod-power2 power2-eq-square*)
thus *?thesis*
by (*simp add*: *power2-eq-square*)
qed
also have ... = (((*Re D*)^2 − (*Re B*)^2) ∗ ((*Re B*)^2 + (*Im B*)^2)) / *cmod B* ^ *4*
by (*simp add*: *power2-eq-square field-simps*)
finally have *Im-product*: *Im* (*i11* / *i12*) ∗ *Im* (*i21* / *i22*) = ((*Re D*)^2 − (*Re B*)^2) ∗ ((*Re B*)^2 + (*Im B*)^2) / *cmod B* ^ *4*
.

show *Im* (*to-complex-cvec i1*) ∗ *Im* (*to-complex-cvec i2*) < *0* ⟷ *intersects-x-axis-cmat H*
proof *safe*
assume *opposite*: *Im* (*to-complex-cvec i1*) ∗ *Im* (*to-complex-cvec i2*) < *0*
show *intersects-x-axis-cmat H*
proof−

have $((Re\ D)^2 - (Re\ B)^2) * ((Re\ B)^2 + (Im\ B)^2) / cmod\ B\ \hat{}\ 4 < 0$
using *Im-product opposite i1 i2*
by *simp*
hence $((Re\ D)^2 - (Re\ B)^2) * ((Re\ B)^2 + (Im\ B)^2) < 0$
by (*simp add*: *divide-less-0-iff*)
hence $(Re\ D)^2 < (Re\ B)^2$
by (*simp add*: *mult-less-0-iff not-sum-power2-lt-zero*)
thus *?thesis*
using *H* ‹*A* = *D*› ‹*is-real D*›
by *auto*
qed
next
have ∗: (∀ *k*. *k* ∗ *Im B* = *1* ⟶ *k* = *0*) ⟶ *Im B* = *0*
apply (*safe*, *erule-tac x*=*1* / *Im B* **in** *allE*)
using *divide-cancel-left* **by** *fastforce*
assume *intersects-x-axis-cmat H*
hence $Re\ D = 0 \lor (Re\ D)^2 < (Re\ B)^2$
using *H* ‹*A* = *D*›
by *auto*
hence $(Re\ D)^2 < (Re\ B)^2$
using ‹*is-real D*› *line H* ‹*C* = *cnj B*›
using *not-x-axis* ∗
by (*auto simp add*: *complex-eq-iff*)
hence $((Re\ D)^2 - (Re\ B)^2) * ((Re\ B)^2 + (Im\ B)^2) < 0$
by (*metis add-cancel-left-left diff-less-eq mult-eq-0-iff mult-less-0-iff power2-eq-square power2-less-0 sum-squares-gt-zero-iff*)
thus *Im* (*to-complex-cvec i1*) ∗ *Im* (*to-complex-cvec i2*) < *0*
using *Im-product i1 i2*
using *divide-eq-0-iff divide-less-0-iff prod.simps*(*2*) *to-complex-cvec-def zero-complex.simps*(*1*) *zero-less-norm-iff*
by *fastforce*
qed
qed
thus *?thesis*
using *assms* ‹*calc-ideal-points H* = {*i1*, *i2*}› ‹*i1* ≠ *i2*›
by *auto*
qed

end
theory *Poincare-Perpendicular*
imports *Poincare-Lines-Axis-Intersections*
begin

9 H-perpendicular h-lines in the Poincaré model

definition *perpendicular-to-x-axis-cmat* :: *complex-mat* ⇒ *bool* **where**
[*simp*]: *perpendicular-to-x-axis-cmat H* ⟷ (*let* (*A*, *B*, *C*, *D*) = *H in is-real B*)

lift-definition *perpendicular-to-x-axis-clmat* :: *circline-mat* ⇒ *bool* **is** *perpendicular-to-x-axis-cmat*
done

lift-definition *perpendicular-to-x-axis* :: *circline* ⇒ *bool* **is** *perpendicular-to-x-axis-clmat*
by *transfer auto*

lemma *perpendicular-to-x-axis*:
assumes *is-poincare-line H*
shows *perpendicular-to-x-axis H* ⟷ *perpendicular x-axis H*
using *assms*
unfolding *perpendicular-def*
proof (*transfer*, *transfer*)
fix *H*
assume *hh*: *hermitean H* ∧ *H* ≠ *mat-zero is-poincare-line-cmat H*
obtain *A B C D* **where** ∗: *H* = (*A*, *B*, *C*, *D*)
by (*cases H*, *auto*)
hence *is-real A* $(cmod\ B)^2 > (cmod\ A)^2$ *H* = (*A*, *B*, *cnj B*, *A*)
using *hermitean-elems*[*of A B C D*] *hh*
by *auto*

thus *perpendicular-to-x-axis-cmat H* =
(*cos-angle-cmat* (*of-circline-cmat x-axis-cmat*) (*of-circline-cmat H*) = *0*)
using *cmod-square*[*of B*] *cmod-square*[*of A*]
by *simp*
qed

lemma *perpendicular-to-x-axis-y-axis*:
assumes *perpendicular-to-x-axis* (*poincare-line* 0_h (*of-complex z*)) $z \neq 0$
shows *is-imag z*
using *assms*
by (*transfer*, *transfer*, *simp*)

lemma *wlog-perpendicular-axes*:
assumes *in-disc*: $u \in$ *unit-disc* $v \in$ *unit-disc* $z \in$ *unit-disc*
assumes *perpendicular*: *is-poincare-line H1 is-poincare-line H2 perpendicular H1 H2*
assumes $z \in$ *circline-set H1* $\cap$ *circline-set H2* $u \in$ *circline-set H1* $v \in$ *circline-set H2*
assumes *axes*: $\bigwedge$ *x y*. ⟦*is-real x*; $0 \leq Re\ x$; $Re\ x < 1$; *is-imag y*; $0 \leq Im\ y$; $Im\ y < 1$⟧ $\Longrightarrow$ *P* 0_h (*of-complex x*) (*of-complex y*)
assumes *moebius*: $\bigwedge$ *M u v w*. ⟦*unit-disc-fix M*; $u \in$ *unit-disc*; $v \in$ *unit-disc*; $w \in$ *unit-disc*; *P* (*moebius-pt M u*) (*moebius-pt M v*) (*moebius-pt M w*) ⟧ $\Longrightarrow$ *P u v w*
assumes *conjugate*: $\bigwedge$ *u v w*. ⟦$u \in$ *unit-disc*; $v \in$ *unit-disc*; $w \in$ *unit-disc*; *P* (*conjugate u*) (*conjugate v*) (*conjugate w*) ⟧ $\Longrightarrow$ *P u v w*
shows *P z u v*
proof−
have $\forall$ *v H1 H2*. *is-poincare-line H1* $\wedge$ *is-poincare-line H2* $\wedge$ *perpendicular H1 H2* $\wedge$
$z \in$ *circline-set H1* $\cap$ *circline-set H2* $\wedge$ $u \in$ *circline-set H1* $\wedge$ $v \in$ *circline-set H2* $\wedge$ $v \in$ *unit-disc* $\longrightarrow$ *P z u v* (**is** *?P z u*)
proof (*rule wlog-x-axis*[**where** *P*=*?P*])
fix *x*
assume *x*: *is-real x* $Re\ x \geq 0$ $Re\ x < 1$
have *of-complex x* $\in$ *unit-disc*
using *x*
by (*simp add*: *cmod-eq-Re*)

show *?P* 0_h (*of-complex x*)
proof *safe*
fix *v H1 H2*
assume $v \in$ *unit-disc*
then obtain *y* **where** *y*: *v* = *of-complex y*
using *inf-or-of-complex*[*of v*]
by *auto*

assume *1*: *is-poincare-line H1 is-poincare-line H2 perpendicular H1 H2*
assume *2*: $0_h \in$ *circline-set H1* $0_h \in$ *circline-set H2* *of-complex x* $\in$ *circline-set H1* $v \in$ *circline-set H2*

show *P* 0_h (*of-complex x*) *v*
proof (*cases of-complex x* = 0_h)
case *True*
show *P* 0_h (*of-complex x*) *v*
proof (*cases v* = 0_h)
case *True*
thus *?thesis*
using ‹*of-complex x* = 0_h›
using *axes*[*of 0 0*]
by *simp*
next
case *False*
show *?thesis*
proof (*rule wlog-rotation-to-positive-y-axis*)
show $v \in$ *unit-disc* $v \neq 0_h$
by *fact*+
next
fix *y*
assume *is-imag y* $0 < Im\ y$ $Im\ y < 1$
thus *P* 0_h (*of-complex x*) (*of-complex y*)

```
          using x axes[of x y]
          by simp
        next
          fix φ u
          assume u ∈ unit-disc u ≠ 0h
                 P 0h (of-complex x) (moebius-pt (moebius-rotation φ) u)
          thus P 0h (of-complex x) u
            using ‹of-complex x = 0h›
            using moebius[of moebius-rotation φ  0h 0h u]
            by simp
        qed
      qed
    next
      case False
      hence *: poincare-line 0h (of-complex x) = x-axis
        using x poincare-line-0-real-is-x-axis[of of-complex x]
        unfolding circline-set-x-axis
        by auto
      hence H1 = x-axis
        using unique-poincare-line[of 0h of-complex x H1] 1 2
        using ‹of-complex x ∈ unit-disc› False
        by simp
      have is-imag y
      proof (cases y = 0)
        case True
        thus ?thesis
          by simp
      next
        case False
        hence 0h ≠ of-complex y
          using of-complex-zero-iff [of y]
          by metis
        hence H2 = poincare-line 0h (of-complex y)
          using 1 2 ‹v ∈ unit-disc›
          using unique-poincare-line[of 0h of-complex y H2] y
          by simp
        thus ?thesis
          using 1 ‹H1 = x-axis›
          using perpendicular-to-x-axis-y-axis[of y] False
          using perpendicular-to-x-axis[of H2]
          by simp
      qed
      show P 0h (of-complex x) v
      proof (cases Im y ≥ 0)
        case True
        thus ?thesis
          using axes[of x y] x y ‹is-imag y› ‹v ∈ unit-disc›
          by (simp add: cmod-eq-Im)
      next
        case False
        show ?thesis
        proof (rule conjugate)
          have Im (cnj y) < 1
            using ‹v ∈ unit-disc› y ‹is-imag y› eq-minus-cnj-iff-imag[of y]
            by (simp add: cmod-eq-Im)
          thus P (conjugate 0h) (conjugate (of-complex x)) (conjugate v)
            using ‹is-real x› eq-cnj-iff-real[of x] y ‹is-imag y›
            using axes[OF x, of cnj y] False
            by simp
          show 0h ∈ unit-disc of-complex x ∈ unit-disc v ∈ unit-disc
            by (simp, fact+)
        qed
      qed
    qed
  qed
next
```

```
    show z ∈ unit-disc u ∈ unit-disc
      by fact+
  next
    fix M u v
    assume ∗: unit-disc-fix M u ∈ unit-disc v ∈ unit-disc
    assume ∗∗: ?P (moebius-pt M u) (moebius-pt M v)
    show ?P u v
    proof safe
      fix w H1 H2
      assume ∗∗∗: is-poincare-line H1 is-poincare-line H2 perpendicular H1 H2
                  u ∈ circline-set H1 u ∈ circline-set H2
                  v ∈ circline-set H1 w ∈ circline-set H2 w ∈ unit-disc
      thus P u v w
        using moebius[of M u v w] ∗
        using ∗∗[rule-format, of moebius-circline M H1 moebius-circline M H2 moebius-pt M w]
        by simp
    qed
  qed
  thus ?thesis
    using assms
    by blast
qed

lemma wlog-perpendicular-foot:
  assumes in-disc: u ∈ unit-disc v ∈ unit-disc w ∈ unit-disc z ∈ unit-disc
  assumes perpendicular: u ≠ v is-poincare-line H perpendicular (poincare-line u v) H
  assumes z ∈ circline-set (poincare-line u v) ∩ circline-set H w ∈ circline-set H
  assumes axes: ⋀ u v w. ⟦is-real u; 0 < Re u; Re u < 1; is-real v; −1 < Re v; Re v < 1; Re u ≠ Re v; is-imag w; 0
≤ Im w; Im w < 1⟧ ⟹ P 0ₕ (of-complex u) (of-complex v) (of-complex w)
   assumes moebius: ⋀ M z u v w. ⟦unit-disc-fix M; u ∈ unit-disc; v ∈ unit-disc; w ∈ unit-disc; z ∈ unit-disc; P
(moebius-pt M z) (moebius-pt M u) (moebius-pt M v) (moebius-pt M w) ⟧ ⟹ P z u v w
  assumes conjugate: ⋀ z u v w. ⟦u ∈ unit-disc; v ∈ unit-disc; w ∈ unit-disc; P (conjugate z) (conjugate u) (conjugate
v) (conjugate w) ⟧ ⟹ P z u v w
  assumes perm: P z v u w ⟹ P z u v w
  shows P z u v w
proof−
  obtain m n where mn: m = u ∨ m = v n = u ∨ n = v m ≠ n m ≠ z
    using ‹u ≠ v›
    by auto

  have n ∈ circline-set (poincare-line z m)
    using ‹z ∈ circline-set (poincare-line u v) ∩ circline-set H›
    using mn
    using unique-poincare-line[of z m poincare-line u v, symmetric] in-disc
    by auto

  have ∀ n. n ∈ unit-disc ∧ m ≠ n ∧ n ∈ circline-set (poincare-line z m) ∧ m ≠ z ⟶ P z m n w (is ?Q z m w)
  proof (rule wlog-perpendicular-axes[where P=?Q])
    show is-poincare-line (poincare-line u v)
      using ‹u ≠ v›
      by auto
  next
    show is-poincare-line H
      by fact
  next
    show m ∈ unit-disc m ∈ circline-set (poincare-line u v)
      using mn in-disc
      by auto
  next
    show w ∈ unit-disc z ∈ unit-disc
      by fact+
  next
    show z ∈ circline-set (poincare-line u v) ∩ circline-set H
      by fact
  next
    show perpendicular (poincare-line u v) H
```

```
      by fact
  next
    show w ∈ circline-set H
      by fact
  next
    fix x y
    assume xy: is-real x 0 ≤ Re x Re x < 1 is-imag y 0 ≤ Im y Im y < 1
    show ?Q 0_h (of-complex x) (of-complex y)
    proof safe
      fix n
      assume n ∈ unit-disc of-complex x ≠ n
      assume n ∈ circline-set (poincare-line 0_h (of-complex x)) of-complex x ≠ 0_h
      hence n ∈ circline-set x-axis
        using poincare-line-0-real-is-x-axis[of of-complex x] xy
        by (auto simp add: circline-set-x-axis)
      then obtain n' where n': n = of-complex n'
        using inf-or-of-complex[of n] ‹n ∈ unit-disc›
        by auto
      hence is-real n'
        using ‹n ∈ circline-set x-axis›
        using of-complex-inj
        unfolding circline-set-x-axis
        by auto
      hence −1 < Re n' Re n' < 1
        using ‹n ∈ unit-disc› n'
        by (auto simp add: cmod-eq-Re)

      have Re n' ≠ Re x
        using complex.expand[of n' x] ‹is-real n'› ‹is-real x› ‹of-complex x ≠ n› n'
        by auto

      have Re x > 0
        using xy ‹of-complex x ≠ 0_h›
        by (cases Re x = 0, auto simp add: complex.expand)

      show P 0_h (of-complex x) n (of-complex y)
        using axes[of x n' y] xy n' ‹Re x > 0› ‹is-real n'› ‹−1 < Re n'› ‹Re n' < 1› ‹Re n' ≠ Re x›
        by simp
    qed
  next
    fix M u v w
    assume 1: unit-disc-fix M u ∈ unit-disc v ∈ unit-disc w ∈ unit-disc
    assume 2: ?Q (moebius-pt M u) (moebius-pt M v) (moebius-pt M w)
    show ?Q u v w
    proof safe
      fix n
      assume n ∈ unit-disc v ≠ n n ∈ circline-set (poincare-line u v) v ≠ u
      thus P u v n w
        using moebius[of M v n w u] 1 2[rule-format, of moebius-pt M n]
        by fastforce
    qed
  next
    fix u v w
    assume 1: u ∈ unit-disc v ∈ unit-disc w ∈ unit-disc
    assume 2: ?Q (conjugate u) (conjugate v) (conjugate w)
    show ?Q u v w
    proof safe
      fix n
      assume n ∈ unit-disc v ≠ n n ∈ circline-set (poincare-line u v) v ≠ u
      thus P u v n w
        using conjugate[of v n w u] 1 2[rule-format, of conjugate n]
        using conjugate-inj
        by auto
    qed
  qed
  thus ?thesis
```

```
    using mn in-disc ‹n ∈ circline-set (poincare-line z m)› perm
    by auto
qed

lemma perpendicular-to-x-axis-intersects-x-axis:
  assumes is-poincare-line H perpendicular-to-x-axis H
  shows intersects-x-axis H
  using assms hermitean-elems
  by (transfer, transfer, auto simp add: cmod-eq-Re)


lemma perpendicular-intersects:
  assumes is-poincare-line H1 is-poincare-line H2
  assumes perpendicular H1 H2
  shows ∃ z. z ∈ unit-disc ∧ z ∈ circline-set H1 ∩ circline-set H2 (is ?P' H1 H2)
proof−
  have ∀ H2. is-poincare-line H2 ∧ perpendicular H1 H2 ⟶ ?P' H1 H2 (is ?P H1)
  proof (rule wlog-line-x-axis)
    show ?P x-axis
    proof safe
      fix H2
      assume is-poincare-line H2 perpendicular x-axis H2
      thus ∃z. z ∈ unit-disc ∧ z ∈ circline-set x-axis ∩ circline-set H2
        using perpendicular-to-x-axis[of H2]
        using perpendicular-to-x-axis-intersects-x-axis[of H2]
        using intersects-x-axis-iff[of H2]
        by auto
    qed
  next
    fix M
    assume unit-disc-fix M
    assume ∗: ?P (moebius-circline M H1)
    show ?P H1
    proof safe
      fix H2
      assume is-poincare-line H2 perpendicular H1 H2
      then obtain z where z ∈ unit-disc z ∈ circline-set (moebius-circline M H1) ∧ z ∈ circline-set (moebius-circline
M H2)
        using ∗[rule-format, of moebius-circline M H2] ‹unit-disc-fix M›
        by auto
      thus ∃z. z ∈ unit-disc ∧ z ∈ circline-set H1 ∩ circline-set H2
        using ‹unit-disc-fix M›
        by (rule-tac x=moebius-pt (−M) z in exI)
           (metis IntI add.inverse-inverse circline-set-moebius-circline-iff moebius-pt-comp-inv-left uminus-moebius-def
unit-disc-fix-discI unit-disc-fix-moebius-uminus)
    qed
  next
    show is-poincare-line H1
      by fact
  qed
  thus ?thesis
    using assms
    by auto
qed


definition calc-perpendicular-to-x-axis-cmat :: complex-vec ⇒ complex-mat where
 [simp]: calc-perpendicular-to-x-axis-cmat z =
    (let (z1, z2) = z
      in if z1∗cnj z2 + z2∗cnj z1 = 0 then
         (0, 1, 1, 0)
      else
         let A = z1∗cnj z2 + z2∗cnj z1;
             B = −(z1∗cnj z1 + z2∗cnj z2)
          in (A, B, B, A)
    )
```

lift-definition *calc-perpendicular-to-x-axis-clmat* :: *complex-homo-coords* ⇒ *circline-mat* **is** *calc-perpendicular-to-x-axis-cmat*
  by (*auto simp add*: *hermitean-def mat-adj-def mat-cnj-def Let-def split*: *if-split-asm*)

lift-definition *calc-perpendicular-to-x-axis* :: *complex-homo* ⇒ *circline* **is** *calc-perpendicular-to-x-axis-clmat*
proof (*transfer*)
  fix *z w*
  assume $z \neq$ *vec-zero* $w \neq$ *vec-zero*
  obtain *z1 z2 w1 w2* **where** *zw*: $z = (z1,\ z2)$ $w = (w1,\ w2)$
    by (*cases z*, *cases w*, *auto*)
  assume $z \approx_v w$
  then obtain *k* **where** ∗: $k \neq 0$ $w1 = k*z1$ $w2 = k*z2$
    using *zw*
    by *auto*
  have $w1 * cnj\ w2 + w2 * cnj\ w1 = (k * cnj\ k) * (z1 * cnj\ z2 + z2 * cnj\ z1)$
    using ∗
    by (*auto simp add*: *field-simps*)
  moreover
  have $w1 * cnj\ w1 + w2 * cnj\ w2 = (k * cnj\ k) * (z1 * cnj\ z1 + z2 * cnj\ z2)$
    using ∗
    by (*auto simp add*: *field-simps*)
  ultimately
  show *circline-eq-cmat* (*calc-perpendicular-to-x-axis-cmat z*) (*calc-perpendicular-to-x-axis-cmat w*)
    using *zw* ∗
    apply (*auto simp add*: *Let-def*)
    apply (*rule-tac x=Re* (*k* ∗ *cnj k*) **in** *exI*, *auto simp add*: *complex.expand field-simps*)
    done
qed

lemma *calc-perpendicular-to-x-axis*:
  assumes $z \neq$ *of-complex 1* $z \neq$ *of-complex* (-1)
  shows $z \in$ *circline-set* (*calc-perpendicular-to-x-axis z*) ∧
        is-poincare-line (*calc-perpendicular-to-x-axis z*) ∧
        perpendicular-to-x-axis (*calc-perpendicular-to-x-axis z*)
  using *assms*
  unfolding *circline-set-def perpendicular-def*
proof (*simp*, *transfer*, *transfer*)
  fix *z* :: *complex-vec*
  obtain *z1 z2* **where** *z*: $z = (z1,\ z2)$
    by (*cases z*, *auto*)
  assume ∗∗: $\neg\ z \approx_v$ *of-complex-cvec 1* $\neg\ z \approx_v$ *of-complex-cvec* $(-\ 1)$
  show *on-circline-cmat-cvec* (*calc-perpendicular-to-x-axis-cmat z*) *z* ∧
       is-poincare-line-cmat (*calc-perpendicular-to-x-axis-cmat z*) ∧
       perpendicular-to-x-axis-cmat (*calc-perpendicular-to-x-axis-cmat z*)
  proof (*cases* $z1*cnj\ z2 + z2*cnj\ z1 = 0$)
    case *True*
    thus *?thesis*
      using *z*
      by (*simp add*: *vec-cnj-def hermitean-def mat-adj-def mat-cnj-def mult.commute*)
  next
    case *False*
    hence $z2 \neq 0$
      using *z*
      by *auto*
    hence $Re\ (z2 * cnj\ z2) \neq 0$
      using ‹$z2 \neq 0$›
      by (*auto simp add*: *complex.expand*)

    have $z1 \neq -z2 \wedge z1 \neq z2$
    proof (*rule ccontr*)
      assume ¬ *?thesis*
      hence $z \approx_v$ *of-complex-cvec 1* ∨ $z \approx_v$ *of-complex-cvec* (-1)
        using *z* ‹$z2 \neq 0$›
        by *auto*
      thus *False*
        using ∗∗

```
      by auto
    qed

    let ?A = z1*cnj z2 + z2*cnj z1 and ?B = −(z1*cnj z1 + z2*cnj z2)
    have Re(z1*cnj z1 + z2*cnj z2) ≥ 0
      by auto
    hence Re ?B ≤ 0
      by (smt uminus-complex.simps(1))
    hence abs (Re ?B) = − Re ?B
      by auto
    also have ... = (Re z1)^2 + (Im z1)^2 + (Re z2)^2 + (Im z2)^2
      by (simp add: power2-eq-square[symmetric])
    also have ... > abs (Re ?A)
    proof (cases Re ?A ≥ 0)
      case False
      have (Re z1 + Re z2)^2 + (Im z1 + Im z2)^2 > 0
        using ‹z1 ≠ −z2 ∧ z1 ≠ z2›
        by (metis add.commute add.inverse-unique complex-neq-0 plus-complex.code plus-complex.simps)
      thus ?thesis
        using False
        by (simp add: power2-sum power2-eq-square field-simps)
    next
      case True
      have (Re z1 − Re z2)^2 + (Im z1 − Im z2)^2 > 0
        using ‹z1 ≠ −z2 ∧ z1 ≠ z2›
        by (meson complex-eq-iff right-minus-eq sum-power2-gt-zero-iff)
      thus ?thesis
        using True
        by (simp add: power2-sum power2-eq-square field-simps)
    qed
    finally
    have abs (Re ?B) > abs (Re ?A)
      .
    moreover
    have cmod ?B = abs (Re ?B) cmod ?A = abs (Re ?A)
      by (simp-all add: cmod-eq-Re)
    ultimately
    have (cmod ?B)^2 > (cmod ?A)^2
      by (smt power2-le-imp-le)
    thus ?thesis
      using z False
      by (simp-all add: Let-def hermitean-def mat-adj-def mat-cnj-def cmod-eq-Re vec-cnj-def field-simps)
  qed
qed

lemma ex-perpendicular:
  assumes is-poincare-line H z ∈ unit-disc
  shows ∃ H′. is-poincare-line H′ ∧ perpendicular H H′ ∧ z ∈ circline-set H′ (is ?P′ H z)
proof−
  have ∀ z. z ∈ unit-disc ⟶ ?P′ H z (is ?P H)
  proof (rule wlog-line-x-axis)
    show ?P x-axis
    proof safe
      fix z
      assume z ∈ unit-disc
      then have z ≠ of-complex 1 z ≠ of-complex (−1)
        by auto
      thus ?P′ x-axis z
        using ‹z ∈ unit-disc›
        using calc-perpendicular-to-x-axis[of z] perpendicular-to-x-axis
        by (rule-tac x = calc-perpendicular-to-x-axis z in exI, auto)
    qed
  next
    fix M
    assume unit-disc-fix M
    assume *: ?P (moebius-circline M H)
```

```
    show ?P H
    proof safe
      fix z
      assume z ∈ unit-disc
      hence moebius-pt M z ∈ unit-disc
        using ‹unit-disc-fix M›
        by auto
      then obtain H' where *: is-poincare-line H' perpendicular (moebius-circline M H) H' moebius-pt M z ∈ circline-set
H'
        using *
        by auto
      have h: H = moebius-circline (−M) (moebius-circline M H)
        by auto
      show ?P' H z
        using * ‹unit-disc-fix M›
        apply (subst h)
        apply (rule-tac x=moebius-circline (−M) H' in exI)
        apply (simp del: moebius-circline-comp-inv-left)
        done
    qed
  qed fact
  thus ?thesis
    using assms
    by simp
qed

lemma ex-perpendicular-foot:
  assumes is-poincare-line H z ∈ unit-disc
  shows ∃ H'. is-poincare-line H' ∧ z ∈ circline-set H' ∧ perpendicular H H' ∧
            (∃ z' ∈ unit-disc. z' ∈ circline-set H' ∩ circline-set H)
  using assms
  using ex-perpendicular[OF assms]
  using perpendicular-intersects[of H]
  by blast

lemma Pythagoras:
  assumes in-disc: u ∈ unit-disc v ∈ unit-disc w ∈ unit-disc v ≠ w
  assumes distinct[u, v, w] ⟶ perpendicular (poincare-line u v) (poincare-line u w)
  shows cosh (poincare-distance v w) = cosh (poincare-distance u v) * cosh (poincare-distance u w) (is ?P' u v w)
proof (cases distinct [u, v, w])
  case False
  thus ?thesis
    using in-disc
    by (auto simp add: poincare-distance-sym)
next
  case True
  have distinct [u, v, w] ⟶ ?P' u v w (is ?P u v w)
  proof (rule wlog-perpendicular-axes[where P=?P])
    show is-poincare-line (poincare-line u v) is-poincare-line (poincare-line u w)
      using ‹distinct [u, v, w]›
      by simp-all
  next
    show perpendicular (poincare-line u v) (poincare-line u w)
      using True assms
      by simp
  next
    show u ∈ unit-disc v ∈ unit-disc w ∈ unit-disc
      by fact+
  next
    show v ∈ circline-set (poincare-line u v) w ∈ circline-set (poincare-line u w)
         u ∈ circline-set (poincare-line u v) ∩ circline-set (poincare-line u w)
      using ‹distinct [u, v, w]›
      by auto
  next
    fix x y
    assume x: is-real x 0 ≤ Re x Re x < 1
```

```
assume y: is-imag y 0 ≤ Im y Im y < 1

have of-complex x ∈ unit-disc of-complex y ∈ unit-disc
  using x y
  by (simp-all add: cmod-eq-Re cmod-eq-Im)

show ?P 0ₕ (of-complex x) (of-complex y)
proof
  assume distinct [0ₕ, of-complex x, of-complex y]
  hence x ≠ 0 y ≠ 0
    by auto

  let ?den1 = 1 − (cmod x)^2 and ?den2 = 1 − (cmod y)^2
  have ?den1 > 0 ?den2 > 0
    using x y
    by (simp-all add: cmod-eq-Re cmod-eq-Im abs-square-less-1)

  let ?d1 = 1 + 2 ∗ (cmod x)^2 / ?den1
  have cosh (poincare-distance 0ₕ (of-complex x)) = ?d1
    using ‹?den1 > 0›
    using poincare-distance-formula[of 0ₕ of-complex x] ‹of-complex x ∈ unit-disc›
    by simp

  moreover

  let ?d2 = 1 + 2 ∗ (cmod y)^2 / ?den2
  have cosh (poincare-distance 0ₕ (of-complex y)) = ?d2
    using ‹?den2 > 0› ‹of-complex y ∈ unit-disc›
    using poincare-distance-formula[of 0ₕ of-complex y]
    by simp

  moreover
  let ?den = ?den1 ∗ ?den2
  let ?d3 = 1 + 2 ∗ (cmod (x − y))^2 / ?den
  have cosh (poincare-distance (of-complex x) (of-complex y)) = ?d3
    using ‹of-complex x ∈ unit-disc› ‹of-complex y ∈ unit-disc›
    using ‹?den1 > 0› ‹?den2 > 0›
    using poincare-distance-formula[of of-complex x of-complex y]
    by simp
  moreover
  have ?d1 ∗ ?d2 = ?d3
  proof−
    have ?d3 = ((1 − (cmod x)^2) ∗ (1 − (cmod y)^2) + 2 ∗ (cmod (x − y))^2) / ?den
      using ‹?den1 > 0› ‹?den2 > 0›
      by (subst add-num-frac, simp, simp)
    also have ... = (Re ((1 − x ∗ cnj x) ∗ (1 − y ∗ cnj y) + 2 ∗ (x − y)∗cnj (x − y)) / ?den)
      using ‹is-real x› ‹is-imag y›
      by ((subst cmod-square)+, simp)
    also have ... = Re (1 + x ∗ cnj x ∗ y ∗ cnj y
                        + x ∗ cnj x − 2 ∗ y ∗ cnj x − 2 ∗ x ∗ cnj y + y ∗ cnj y) / ?den
      by (simp add: field-simps)
    also have ... = Re ((1 + y ∗ cnj y) ∗ (1 + x ∗ cnj x)) / ?den
      using ‹is-real x› ‹is-imag y›
      by (simp add: field-simps)
    finally
    show ?thesis
      using ‹?den1 > 0› ‹?den2 > 0›
      apply (subst add-num-frac, simp)
      apply (subst add-num-frac, simp)
      apply simp
      apply (subst cmod-square)+
      apply (simp add: field-simps)
      done
  qed
  ultimately
  show ?P' 0ₕ (of-complex x) (of-complex y)
```

```
      by simp
    qed
  next
    fix M u v w
    assume 1: unit-disc-fix M u ∈ unit-disc v ∈ unit-disc w ∈ unit-disc
    assume 2: ?P (moebius-pt M u) (moebius-pt M v) (moebius-pt M w)
    show ?P u v w
      using 1 2
      by auto
  next
    fix u v w
    assume 1:  u ∈ unit-disc v ∈ unit-disc w ∈ unit-disc
    assume 2: ?P (conjugate u) (conjugate v) (conjugate w)
    show ?P u v w
      using 1 2
      by (auto simp add: conjugate-inj)
  qed
  thus ?thesis
    using True
    by simp
qed

end
```

10 Poincaré disc model types

In this section we introduce datatypes that represent objects in the Poincaré disc model. The types are defined as subtypes (e.g., the h-points are defined as elements of $\mathbb{C}P^1$ that lie within the unit disc). The functions on those types are defined by lifting the functions defined on the carrier type (e.g., h-distance is defined by lifting the distance function defined for elements of $\mathbb{C}P^1$).

```
theory Poincare
imports Poincare-Lines Poincare-Between Poincare-Distance Poincare-Circles
begin
```

10.1 H-points

```
typedef p-point = {z. z ∈ unit-disc}
  using zero-in-unit-disc
  by (rule-tac x=0h in exI, simp)

setup-lifting type-definition-p-point
```

Point zero

```
lift-definition p-zero :: p-point is 0h
  by (rule zero-in-unit-disc)
```

Constructing h-points from complex numbers

```
lift-definition p-of-complex :: complex ⇒ p-point is λ z. if cmod z < 1 then of-complex z else 0h
  by auto
```

10.2 H-lines

```
typedef p-line = {H. is-poincare-line H}
  by (rule-tac x=x-axis in exI, simp)

setup-lifting type-definition-p-line

lift-definition p-incident :: p-line ⇒ p-point ⇒ bool is on-circline
  done
```

Set of h-points on an h-line

```
definition p-points :: p-line ⇒ p-point set where
  p-points l = {p. p-incident l p}
```

x-axis is an example of an h-line

lift-definition *p-x-axis* :: *p-line* **is** *x-axis*
  by *simp*

Constructing the unique h-line from two h-points

lift-definition *p-line* :: *p-point* ⇒ *p-point* ⇒ *p-line* **is** *poincare-line*
proof−
  fix *u v*
  show *is-poincare-line* (*poincare-line u v*)
  proof (*cases u* ≠ *v*)
    case *True*
    thus *?thesis*
      by *simp*
  next

This branch must work only for formal reasons.

    case *False*
    thus *?thesis*
      by (*transfer*, *transfer*, *auto simp add*: *hermitean-def mat-adj-def mat-cnj-def split*: *if-split-asm*)
  qed
qed

Next we show how to lift some lemmas. This could be done for all the lemmas that we have proved earlier, but we do not do that.

If points are different then the constructed line contains the starting points

lemma *p-on-line*:
  assumes *z* ≠ *w*
  shows *p-incident* (*p-line z w*) *z*
        p-incident (*p-line z w*) *w*
  using *assms*
  by (*transfer*, *simp*)+

There is a unique h-line passing trough the two different given h-points

lemma
  assumes *u* ≠ *v*
  shows ∃! *l*. {*u*, *v*} ⊆ *p-points l*
  using *assms*
  apply (*rule-tac a*=*p-line u v* **in** *ex1I*, *auto simp add*: *p-points-def p-on-line*)
  apply (*transfer*, *simp add*: *unique-poincare-line*)
  done

The unique h-line trough zero and a non-zero h-point on the x-axis is the x-axis

lemma
  assumes *p-zero* ∈ *p-points l u* ∈ *p-points l u* ≠ *p-zero u* ∈ *p-points p-x-axis*
  shows *l* = *p-x-axis*
  using *assms*
  unfolding *p-points-def*
  apply *simp*
  apply *transfer*
  using *is-poincare-line-0-real-is-x-axis inf-notin-unit-disc*
  unfolding *circline-set-def*
  by *blast*

10.3 H-collinearity

lift-definition *p-collinear* :: *p-point set* ⇒ *bool* **is** *poincare-collinear*
  done

10.4 H-isometries

H-isometries are functions that map the unit disc onto itself

typedef *p-isometry* = {*f*. *unit-disc-fix-f f*}

by (*rule-tac x=id* **in** *exI*, *simp add*: *unit-disc-fix-f-def*, *rule-tac x=id-moebius* **in** *exI*, *simp*)

setup-lifting *type-definition-p-isometry*

Action of an h-isometry on an h-point

lift-definition *p-isometry-pt* :: *p-isometry* ⇒ *p-point* ⇒ *p-point* **is** λ *f p*. *f p*
using *unit-disc-fix-f-unit-disc*
by *auto*

Action of an h-isometry on an h-line

lift-definition *p-isometry-line* :: *p-isometry* ⇒ *p-line* ⇒ *p-line* **is** λ *f l*. *unit-disc-fix-f-circline f l*
proof−
fix *f H*
assume *unit-disc-fix-f f is-poincare-line H*
then obtain *M* **where** *unit-disc-fix M* **and** ∗: *f* = *moebius-pt M* ∨ *f* = *moebius-pt M* ∘ *conjugate*
unfolding *unit-disc-fix-f-def*
by *auto*
show *is-poincare-line* (*unit-disc-fix-f-circline f H*)
using ∗
proof
assume *f* = *moebius-pt M*
thus *?thesis*
using ‹*unit-disc-fix M*› ‹*is-poincare-line H*›
using *unit-disc-fix-f-circline-direct*[*of M f H*]
by *auto*
next
assume *f* = *moebius-pt M* ∘ *conjugate*
thus *?thesis*
using ‹*unit-disc-fix M*› ‹*is-poincare-line H*›
using *unit-disc-fix-f-circline-indirect*[*of M f H*]
by *auto*
qed
qed

An example lemma about h-isometries.

H-isometries preserve h-collinearity

lemma *p-collinear-p-isometry-pt* [*simp*]:
shows *p-collinear* (*p-isometry-pt M* ‘ *A*) ⟷ *p-collinear A*
proof−
have ∗: ∀ *M A*. ((λ*x*. *moebius-pt M* (*conjugate x*)) ‘ *A* = *moebius-pt M* ‘ (*conjugate* ‘ *A*))
by *auto*
show *?thesis*
by *transfer* (*auto simp add*: *unit-disc-fix-f-def* ∗)
qed

10.5 H-distance and h-congruence

lift-definition *p-dist* :: *p-point* ⇒ *p-point* ⇒ *real* **is** *poincare-distance*
done

definition *p-congruent* :: *p-point* ⇒ *p-point* ⇒ *p-point* ⇒ *p-point* ⇒ *bool* **where**
[*simp*]: *p-congruent u v u′ v′* ⟷ *p-dist u v* = *p-dist u′ v′*

lemma
assumes *p-dist u v* = *p-dist u′ v′*
assumes *p-dist v w* = *p-dist v′ w′*
assumes *p-dist u w* = *p-dist u′ w′*
shows ∃ *f*. *p-isometry-pt f u* = *u′* ∧ *p-isometry-pt f v* = *v′* ∧ *p-isometry-pt f w* = *w′*
using *assms*
apply *transfer*
using *unit-disc-fix-f-congruent-triangles*
by *auto*

We prove that unit disc equipped with Poincaré distance is a metric space, i.e. an instantiation of *metric-space* locale.

instantiation *p-point* :: *metric-space*
begin
definition *dist-p-point* = *p-dist*
definition (*uniformity-p-point* :: (*p-point* × *p-point*) *filter*) = (*INF* *e*∈{*0*<..}. *principal* {(*x*, *y*). *dist-class.dist x y* < *e*})
definition *open-p-point* (*U* :: *p-point set*) = (∀ *x* ∈ *U*. *eventually* (λ(*x*′, *y*). *x*′ = *x* ⟶ *y* ∈ *U*) *uniformity*)
instance
proof
  fix *x y* :: *p-point*
  show (*dist-class.dist x y* = *0*) = (*x* = *y*)
    unfolding *dist-p-point-def*
    by (*transfer*, *simp add*: *poincare-distance-eq-0-iff*)
next
  fix *x y z* :: *p-point*
  show *dist-class.dist x y* ≤ *dist-class.dist x z* + *dist-class.dist y z*
    unfolding *dist-p-point-def*
    apply *transfer*
    using *poincare-distance-triangle-inequality poincare-distance-sym*
    by *metis*
qed (*simp-all add*: *open-p-point-def uniformity-p-point-def*)
end

10.6 H-betweennes

lift-definition *p-between* :: *p-point* ⇒ *p-point* ⇒ *p-point* ⇒ *bool* **is** *poincare-between*
  done

end

11 Poincaré model satisfies Tarski axioms

theory *Poincare-Tarski*
  imports *Poincare Poincare-Lines-Axis-Intersections Tarski*
begin

11.1 Pasch axiom

lemma *Pasch-fun-mono*:
  fixes *r1 r2* :: *real*
  assumes *0* < *r1* **and** *r1* ≤ *r2* **and** *r2* < *1*
  shows *r1* + *1*/*r1* ≥ *r2* + *1*/*r2*
proof (*cases r1* = *r2*)
  case *True*
  thus *?thesis*
    by *simp*
next
  case *False*
  hence *r2* − *r1* > *0*
    using *assms*
    by *simp*

  have *r1* ∗ *r2* < *1*
    using *assms*
    by (*smt mult-le-cancel-left1*)
  hence *1* / (*r1* ∗ *r2*) > *1*
    using *assms*
    by *simp*
  hence (*r2* − *r1*) / (*r1* ∗ *r2*) > (*r2* − *r1*)
    using ‹*r2* − *r1* > *0*›
    using *mult-less-cancel-left-pos*[*of r2* − *r1 1 1* / (*r1* ∗ *r2*)]
    by *simp*
  hence *1* / *r1* − *1* / *r2* > *r2* − *r1*
    using *assms*
    by (*simp add*: *field-simps*)
  thus *?thesis*
    by *simp*

qed

Pasch axiom, non-degenerative case.

lemma *Pasch-nondeg*:
  assumes $x \in$ *unit-disc* **and** $y \in$ *unit-disc* **and** $z \in$ *unit-disc* **and** $u \in$ *unit-disc* **and** $v \in$ *unit-disc*
  assumes *distinct* $[x, y, z, u, v]$
  assumes $\neg$ *poincare-collinear* $\{x, y, z\}$
  assumes *poincare-between x u z* **and** *poincare-between y v z*
  shows $\exists$ *a. a* $\in$ *unit-disc* $\wedge$ *poincare-between u a y* $\wedge$ *poincare-between x a v*
proof−
  have $\forall$ *y z u. distinct* $[x, y, z, u, v]$ $\wedge$ $\neg$ *poincare-collinear* $\{x, y, z\}$ $\wedge$ $y \in$ *unit-disc* $\wedge$ $z \in$ *unit-disc* $\wedge$ $u \in$ *unit-disc* $\wedge$
    poincare-between x u z $\wedge$ *poincare-between y v z* $\longrightarrow$ ($\exists$ *a. a* $\in$ *unit-disc* $\wedge$ *poincare-between u a y* $\wedge$ *poincare-between x a v*) (**is** *?P x v*)
  proof (*rule wlog-positive-x-axis*[**where** *P=?P*])
    fix *v*
    assume *v*: *is-real v* $0 <$ *Re v Re v* < 1
    hence *of-complex v* $\in$ *unit-disc*
      by (*auto simp add*: *cmod-eq-Re*)
    show *?P* 0_h (*of-complex v*)
    proof *safe*
      fix *y z u*
      assume *distinct*: *distinct* $[0_h$, *y*, *z*, *u*, *of-complex v*$]$
      assume *in-disc*: $y \in$ *unit-disc* $z \in$ *unit-disc* $u \in$ *unit-disc*
      then obtain y' z' u'
        where ∗: $y =$ *of-complex* y' $z =$ *of-complex* z' $u =$ *of-complex* u'
        using *inf-or-of-complex inf-notin-unit-disc*
        by *metis*

      have $y' \neq 0$ $z' \neq 0$ $u' \neq 0$ $v \neq 0$ $y' \neq z'$ $y' \neq u'$ $z' \neq u'$ $y \neq z$ $y \neq u$ $z \neq u$
        using *of-complex-inj distinct* ∗
        by *auto*

      note *distinct* = *distinct this*

      assume $\neg$ *poincare-collinear* $\{0_h, y, z\}$

      hence *nondeg-yz*: y'∗*cnj* $z' \neq$ *cnj* y' ∗ z'
        using ∗ *poincare-collinear-zero-iff*[*of* y' z'] *in-disc distinct*
        by *auto*

      assume *poincare-between* 0_h *u z*

      hence *Arg* $u' =$ *Arg* z' *cmod* $u' \leq$ *cmod* z'
        using ∗ *poincare-between-0uv*[*of u z*] *distinct in-disc*
        by *auto*

      then obtain φ *ru rz* **where**
        uz-polar: $u' =$ *cor ru* ∗ *cis* φ $z' =$ *cor rz* ∗ *cis* φ $0 <$ *ru ru* $\leq$ *rz* $0 <$ *rz* **and**
          $\varphi =$ *Arg* u' $\varphi =$ *Arg* z'
        using ∗ ‹$u' \neq 0$› ‹$z' \neq 0$›
        by (*smt cmod-cis norm-le-zero-iff*)

      obtain ϑ *ry* **where**
        y-polar: $y' =$ *cor ry* ∗ *cis* ϑ *ry* > 0 **and** $\vartheta =$ *Arg* y'
        using ‹$y' \neq 0$›
        by (*smt cmod-cis norm-le-zero-iff*)

      from *in-disc* ∗ ‹$u' =$ *cor ru* ∗ *cis* φ› ‹$z' =$ *cor rz* ∗ *cis* φ› ‹$y' =$ *cor ry* ∗ *cis* ϑ›
      have *ru* < 1 *rz* < 1 *ry* < 1
        by (*auto simp*: *norm-mult*)

      note *polar* = *this y-polar uz-polar*

      have *nondeg*: *cis* ϑ ∗ *cis* $(-\ \varphi) \neq$ *cis* $(-\ \vartheta)$ ∗ *cis* φ
        using *nondeg-yz polar*
        by *simp*

let *?yz* = *poincare-line y z*
let *?v* = *calc-x-axis-intersection ?yz*

assume *poincare-between y* (*of-complex v*) *z*

hence *of-complex v* ∈ *circline-set ?yz*
  using *in-disc* ‹*of-complex v* ∈ *unit-disc*›
  using *distinct poincare-between-poincare-collinear*[*of y of-complex v z*]
  using *unique-poincare-line*[*of y z*]
  by (*auto simp add*: *poincare-collinear-def*)
moreover
have *of-complex v* ∈ *circline-set x-axis*
  using ‹*is-real v*›
  unfolding *circline-set-x-axis*
  by *auto*
moreover
have *?yz* ≠ *x-axis*
proof (*rule ccontr*)
  assume ¬ *?thesis*
  hence {0_h, *y*, *z*} ⊆ *circline-set* (*poincare-line y z*)
    unfolding *circline-set-def*
    using *distinct poincare-line*[*of y z*]
    by *auto*
  hence *poincare-collinear* {0_h, *y*, *z*}
    unfolding *poincare-collinear-def*
    using *distinct*
    by *force*
  thus *False*
    using ‹¬ *poincare-collinear* {0_h, *y*, *z*}›
    by *simp*
qed
ultimately
have *?v* = *of-complex v intersects-x-axis ?yz*
  using *unique-calc-x-axis-intersection*[*of poincare-line y z of-complex v*]
  using *intersects-x-axis-iff*[*of ?yz*]
  using *distinct* ‹*of-complex v* ∈ *unit-disc*›
  by (*metis IntI is-poincare-line-poincare-line*)+

have *intersects-x-axis-positive ?yz*
  using ‹*Re v* > *0*› ‹*of-complex v* ∈ *unit-disc*›
  using ‹*of-complex v* ∈ *circline-set ?yz*› ‹*of-complex v* ∈ *circline-set x-axis*›
  using *intersects-x-axis-positive-iff*[*of ?yz*] ‹*y* ≠ *z*› ‹*?yz* ≠ *x-axis*›
  unfolding *positive-x-axis-def*
  by *force*

have *y* ∉ *circline-set x-axis*
proof (*rule ccontr*)
  assume ¬ *?thesis*
  moreover
  hence *poincare-line y* (*of-complex v*) = *x-axis*
    using *distinct* ‹*of-complex v* ∈ *circline-set x-axis*›
    using *in-disc* ‹*of-complex v* ∈ *unit-disc*›
    using *unique-poincare-line*[*of y of-complex v x-axis*]
    by *simp*
  moreover
  have *z* ∈ *circline-set* (*poincare-line y* (*of-complex v*))
    using ‹*of-complex v* ∈ *circline-set ?yz*›
    using *unique-poincare-line*[*of y of-complex v poincare-line y z*]
    using *in-disc* ‹*of-complex v* ∈ *unit-disc*› *distinct*
    using *poincare-line*[*of y z*]
    unfolding *circline-set-def*
    by (*metis distinct-length-2-or-more is-poincare-line-poincare-line mem-Collect-eq*)
  ultimately
  have *y* ∈ *circline-set x-axis z* ∈ *circline-set x-axis*
    by *auto*

hence *poincare-collinear* $\{0_h,\ y,\ z\}$
unfolding *poincare-collinear-def*
by *force*
thus *False*
using ‹¬ *poincare-collinear* $\{0_h,\ y,\ z\}$›
by *simp*
qed

moreover

have $z \notin$ *circline-set x-axis*
proof (*rule ccontr*)
assume ¬ *?thesis*
moreover
hence *poincare-line z* (*of-complex v*) = *x-axis*
using *distinct* ‹*of-complex v* ∈ *circline-set x-axis*›
using *in-disc* ‹*of-complex v* ∈ *unit-disc*›
using *unique-poincare-line*[*of z of-complex v x-axis*]
by *simp*
moreover
have $y \in$ *circline-set* (*poincare-line z* (*of-complex v*))
using ‹*of-complex v* ∈ *circline-set ?yz*›
using *unique-poincare-line*[*of z of-complex v poincare-line y z*]
using *in-disc* ‹*of-complex v* ∈ *unit-disc*› *distinct*
using *poincare-line*[*of y z*]
unfolding *circline-set-def*
by (*metis distinct-length-2-or-more is-poincare-line-poincare-line mem-Collect-eq*)
ultimately
have $y \in$ *circline-set x-axis* $z \in$ *circline-set x-axis*
by *auto*
hence *poincare-collinear* $\{0_h,\ y,\ z\}$
unfolding *poincare-collinear-def*
by *force*
thus *False*
using ‹¬ *poincare-collinear* $\{0_h,\ y,\ z\}$›
by *simp*
qed

ultimately

have $\varphi * \vartheta < 0$
using ‹*poincare-between y* (*of-complex v*) *z*›
using *poincare-between-x-axis-intersection*[*of y z of-complex v*]
using *in-disc* ‹*of-complex v* ∈ *unit-disc*› *distinct*
using ‹*of-complex v* ∈ *circline-set ?yz*› ‹*of-complex v* ∈ *circline-set x-axis*›
using ‹φ = *Arg* z'› ‹ϑ = *Arg* y'› ∗
by (*simp add*: *field-simps*)

have $\varphi \neq pi$ $\varphi \neq 0$
using ‹$z \notin$ *circline-set x-axis*› ∗ *polar cis-pi*
unfolding *circline-set-x-axis*
by *auto*

have $\vartheta \neq pi$ $\vartheta \neq 0$
using ‹$y \notin$ *circline-set x-axis*› ∗ *polar cis-pi*
unfolding *circline-set-x-axis*
by *auto*

have *phi-sin*: $\varphi > 0 \longleftrightarrow sin\ \varphi > 0$ $\varphi < 0 \longleftrightarrow sin\ \varphi < 0$
using ‹φ = *Arg* z'› ‹$\varphi \neq 0$› ‹$\varphi \neq pi$›
using *Arg-bounded*[*of* z']
by (*smt sin-gt-zero sin-le-zero sin-pi-minus sin-0-iff-canon sin-ge-zero*)+

have *theta-sin*: $\vartheta > 0 \longleftrightarrow sin\ \vartheta > 0$ $\vartheta < 0 \longleftrightarrow sin\ \vartheta < 0$
using ‹ϑ = *Arg* y'› ‹$\vartheta \neq 0$› ‹$\vartheta \neq pi$›
using *Arg-bounded*[*of* y']

  by (*smt sin-gt-zero sin-le-zero sin-pi-minus sin-0-iff-canon sin-ge-zero*)+

have *sin* $\varphi * $ *sin* $\vartheta < 0$
  using ‹$\varphi * \vartheta < 0$› *phi-sin theta-sin*
  by (*simp add*: *mult-less-0-iff*)

have *sin* $(\varphi - \vartheta) \neq 0$
proof (*rule ccontr*)
  assume ¬ *?thesis*
  hence *sin* $(\varphi - \vartheta) = 0$
    by *simp*
  have $- 2 * pi < \varphi - \vartheta$ $\varphi - \vartheta < 2 * pi$
    using ‹$\varphi = Arg\ z'$› ‹$\vartheta = Arg\ y'$› *Arg-bounded*[*of z'*] *Arg-bounded*[*of y'*] ‹$\varphi \neq pi$› ‹$\vartheta \neq pi$›
    by *auto*
  hence $\varphi - \vartheta = -pi \vee \varphi - \vartheta = 0 \vee \varphi - \vartheta = pi$
    using ‹*sin* $(\varphi - \vartheta) = 0$›
    by (*smt sin-0-iff-canon sin-periodic-pi2*)
  moreover
  {
    assume $\varphi - \vartheta = - pi$
    hence $\varphi = \vartheta - pi$
      by *simp*
    hence *False*
      using *nondeg-yz*
      using ‹$y' = cor\ ry * cis\ \vartheta$› ‹$z' = cor\ rz * cis\ \varphi$› ‹$rz > 0$› ‹$ry > 0$›
      by *auto*
  }
  moreover
  {
    assume $\varphi - \vartheta = 0$
    hence $\varphi = \vartheta$
      by *simp*
    hence *False*
      using ‹$y' = cor\ ry * cis\ \vartheta$› ‹$z' = cor\ rz * cis\ \varphi$› ‹$rz > 0$› ‹$ry > 0$›
      using *nondeg-yz*
      by *auto*
  }
  moreover
  {
    assume $\varphi - \vartheta = pi$
    hence $\varphi = \vartheta + pi$
      by *simp*
    hence *False*
      using ‹$y' = cor\ ry * cis\ \vartheta$› ‹$z' = cor\ rz * cis\ \varphi$› ‹$rz > 0$› ‹$ry > 0$›
      using *nondeg-yz*
      by *auto*
  }
  ultimately
  show *False*
    by *auto*
qed

have $u \notin$ *circline-set x-axis*
proof−
  have ¬ *is-real* u'
    using ∗ *polar in-disc*
    using ‹$\varphi \neq 0$› ‹$\varphi = Arg\ u'$› ‹$\varphi \neq pi$› *phi-sin*(*1*) *phi-sin*(*2*)
    by (*metis is-real-arg2*)
  moreover
  have $u \neq \infty_h$
    using *in-disc*
    by *auto*
  ultimately
  show *?thesis*
    using ∗ *of-complex-inj*[*of u'*]
    unfolding *circline-set-x-axis*

by *auto*
qed

let *?yu = poincare-line y u*
have *nondeg-yu*: $y' * cnj\ u' \neq cnj\ u' * u'$
using *nondeg-yz polar* ‹$ru > 0$› ‹$rz > 0$› *distinct*
by *auto*

{

fix *r* :: *real*
assume $r > 0$

have *den*: $cor\ ry * cis\ \vartheta * cnj\ 1 * cnj\ (cor\ r * cis\ \varphi) * 1 - cor\ r * cis\ \varphi * cnj\ 1 * cnj\ (cor\ ry * cis\ \vartheta) * 1 \neq 0$
using ‹$0 < r$› ‹$0 < ry$› *nondeg*
by *auto*

let $?A = 2 * r * ry * sin(\varphi - \vartheta)$
let $?B = \mathrm{i} * (r * cis\ \varphi * (1 + ry^2) - ry * cis\ \vartheta * (1 + r^2))$
let $?ReB = ry * (1 + r^2) * sin\ \vartheta - r * (1 + ry^2) * sin\ \varphi$

have $Re\ (\mathrm{i} * (r * cis\ (-\varphi) * ry * cis\ (\vartheta) - ry * cis\ (-\vartheta) * r * cis\ (\varphi))) = ?A$
by (*simp add*: *sin-diff field-simps*)
moreover
have $cor\ ry * cis\ (-\ \vartheta) * (cor\ ry * cis\ \vartheta) = ry^2$ $\ cor\ r * cis\ (-\ \varphi) * (cor\ r * cis\ \varphi) = r^2$
by (*metis cis-inverse cis-neq-zero divide-complex-def cor-squared nonzero-mult-div-cancel-right power2-eq-square semiring-normalization-rules*(*15*))+
ultimately
have *1*: *poincare-line-cvec-cmat* (*of-complex-cvec* ($cor\ ry * cis\ \vartheta$)) (*of-complex-cvec* ($cor\ r * cis\ \varphi$)) = (*?A*, *?B*, *cnj ?B*, *?A*)
using *den*
unfolding *poincare-line-cvec-cmat-def of-complex-cvec-def Let-def prod.case*
by (*simp add*: *field-simps*)

have *2*: *is-real ?A*
by *simp*
let $?mix = cis\ \vartheta * cis\ (-\ \varphi) - cis\ (-\ \vartheta) * cis\ \varphi$
have *is-imag ?mix*
using *eq-minus-cnj-iff-imag*[*of ?mix*]
by *simp*
hence $Im\ ?mix \neq 0$
using *nondeg*
using *complex.expand*[*of ?mix 0*]
by *auto*
hence *3*: $Re\ ?A \neq 0$
using ‹$r > 0$› ‹$ry > 0$›
by (*simp add*: *sin-diff field-simps*)

have $?A \neq 0$
using *2 3*
by *auto*
hence *4*: $cor\ ?A \neq 0$
using *2 3*
by (*metis zero-complex.simps*(*1*))

have *5*: $?ReB\ /\ ?A = (sin\ \vartheta)\ /\ (2 * sin(\varphi - \vartheta)) * (1/r + r) - (sin\ \varphi)\ /\ (2 * sin\ (\varphi - \vartheta)) * (1/ry + ry)$
using ‹$ry > 0$› ‹$r > 0$›
apply (*subst diff-divide-distrib*)
apply (*subst add-frac-num*, *simp*)
apply (*subst add-frac-num*, *simp*)
apply (*simp add*: *power2-eq-square mult.commute*)
apply (*simp add*: *field-simps*)
done

have *poincare-line-cvec-cmat* (*of-complex-cvec* ($cor\ ry * cis\ \vartheta$)) (*of-complex-cvec* ($cor\ r * cis\ \varphi$)) = (*?A*, *?B*, *cnj ?B*, *?A*) $\wedge$

is-real ?A ∧ *Re ?A* ≠ *0* ∧ *?A* ≠ *0* ∧ *cor ?A* ≠ *0* ∧
Re ?B = *?ReB* ∧
?ReB / *?A* = (*sin* ϑ) / (*2* ∗ *sin*(φ − ϑ)) ∗ (*1*/*r* + *r*) − (*sin* φ) / (*2* ∗ *sin* (φ − ϑ)) ∗ (*1*/*ry* + *ry*)
using *1 2 3 4 5*
by *auto*
}
note ∗∗ = *this*

let *?Ayz* = *2* ∗ *rz* ∗ *ry* ∗ *sin* (φ − ϑ)
let *?Byz* = i ∗ (*rz* ∗ *cis* φ ∗ (*1* + ry^2) − *ry* ∗ *cis* ϑ ∗ (*1* + rz^2))
let *?ReByz* = *ry* ∗ (*1* + rz^2) ∗ *sin* ϑ − *rz* ∗ (*1* + ry^2) ∗ *sin* φ
let *?Kz* = (*sin* ϑ) / (*2* ∗ *sin*(φ − ϑ)) ∗ (*1*/*rz* + *rz*) − (*sin* φ) / (*2* ∗ *sin* (φ − ϑ)) ∗ (*1*/*ry* + *ry*)
have *yz*: *poincare-line-cvec-cmat* (*of-complex-cvec* (*cor ry* ∗ *cis* ϑ)) (*of-complex-cvec* (*cor rz* ∗ *cis* φ)) = (*?Ayz*, *?Byz*, *cnj ?Byz*, *?Ayz*)
is-real ?Ayz Re ?Ayz ≠ *0 ?Ayz* ≠ *0 cor ?Ayz* ≠ *0 Re ?Byz* = *?ReByz* **and** *Kz*: *?ReByz* / *?Ayz* = *?Kz*
using ∗∗[*OF* ‹*0* < *rz*›]
by *auto*

let *?Ayu* = *2* ∗ *ru* ∗ *ry* ∗ *sin* (φ − ϑ)
let *?Byu* = i ∗ (*ru* ∗ *cis* φ ∗ (*1* + ry^2) − *ry* ∗ *cis* ϑ ∗ (*1* + ru^2))
let *?ReByu* = *ry* ∗ (*1* + ru^2) ∗ *sin* ϑ − *ru* ∗ (*1* + ry^2) ∗ *sin* φ
let *?Ku* = (*sin* ϑ) / (*2* ∗ *sin*(φ − ϑ)) ∗ (*1*/*ru* + *ru*) − (*sin* φ) / (*2* ∗ *sin* (φ − ϑ)) ∗ (*1*/*ry* + *ry*)
have *yu*: *poincare-line-cvec-cmat* (*of-complex-cvec* (*cor ry* ∗ *cis* ϑ)) (*of-complex-cvec* (*cor ru* ∗ *cis* φ)) = (*?Ayu*, *?Byu*, *cnj ?Byu*, *?Ayu*)
is-real ?Ayu Re ?Ayu ≠ *0 ?Ayu* ≠ *0 cor ?Ayu* ≠ *0 Re ?Byu* = *?ReByu* **and** *Ku*: *?ReByu* / *?Ayu* = *?Ku*
using ∗∗[*OF* ‹*0* < *ru*›]
by *auto*

have *?Ayz* ≠ *0*
using ‹*sin* (φ − ϑ) ≠ *0*› ‹*ry* > *0*› ‹*rz* > *0*›
by *auto*

have *Re ?Byz* / *?Ayz* < −*1*
using ‹*intersects-x-axis-positive ?yz*›
∗ ‹*y*′ = *cor ry* ∗ *cis* ϑ› ‹*z*′ = *cor rz* ∗ *cis* φ› ‹*u*′ = *cor ru* ∗ *cis* φ›
apply *simp*
apply (*transfer fixing*: *ry rz ru* ϑ φ)
apply (*transfer fixing*: *ry rz ru* ϑ φ)
proof−
assume *intersects-x-axis-positive-cmat* (*poincare-line-cvec-cmat* (*of-complex-cvec* (*cor ry* ∗ *cis* ϑ)) (*of-complex-cvec* (*cor rz* ∗ *cis* φ)))
thus (*ry* ∗ *sin* ϑ ∗ (*1* + rz^2) − *rz* ∗ *sin* φ ∗ (*1* + ry^2)) / (*2* ∗ *rz* ∗ *ry* ∗ *sin* (φ − ϑ)) < − *1*
using *yz*
by *simp*
qed

have *?ReByz* / *?Ayz* ≥ *?ReByu* / *?Ayu*
proof (*cases sin* φ > *0*)
case *True*
hence *sin* ϑ < *0*
using ‹*sin* φ ∗ *sin* ϑ < *0*›
by (*smt mult-nonneg-nonneg*)

have *?ReByz* < *0*
proof−
have *ry* ∗ (*1* + rz^2) ∗ *sin* ϑ < *0*
using ‹*ry* > *0*› ‹*rz* > *0*›
using ‹*sin* ϑ < *0*›
by (*smt mult-pos-neg mult-pos-pos zero-less-power*)
moreover
have *rz* ∗ (*1* + ry^2) ∗ *sin* φ > *0*
using ‹*ry* > *0*› ‹*rz* > *0*›
using ‹*sin* φ > *0*›
by (*smt mult-pos-neg mult-pos-pos zero-less-power*)
ultimately
show *?thesis*

by *simp*
qed
have *?Ayz* > *0*
using ‹*Re ?Byz* / *?Ayz* < −*1*› ‹*Re ?Byz* = *?ReByz*› ‹*?ReByz* < *0*›
by (*smt divide-less-0-iff*)
hence *sin* (φ − ϑ) > *0*
using ‹*ry* > *0*› ‹*rz* > *0*›
by (*smt mult-pos-pos zero-less-mult-pos*)

have *1* / *ru* + *ru* ≥ *1* / *rz* + *rz*
using *Pasch-fun-mono*[*of ru rz*] ‹*0* < *ru*› ‹*ru* ≤ *rz*› ‹*rz* < *1*›
by *simp*
hence *sin* ϑ ∗ (*1* / *ru* + *ru*) ≤ *sin* ϑ ∗ (*1* / *rz* + *rz*)
using ‹*sin* ϑ < *0*›
by *auto*
thus *?thesis*
using ‹*ru* > *0*› ‹*rz* > *0*› ‹*ru* ≤ *rz*› ‹*rz* < *1*› ‹*?Ayz* > *0*› ‹*sin* (φ − ϑ) > *0*›
using *divide-right-mono*[*of sin* ϑ ∗ (*1* / *ru* + *ru*) *sin* ϑ ∗ (*1* / *rz* + *rz*) *2* ∗ *sin* (φ − ϑ)]
by (*subst Kz*, *subst Ku*) *simp*
next
assume ¬ *sin* φ > *0*
hence *sin* φ < *0*
using ‹*sin* φ ∗ *sin* ϑ < *0*›
by (*cases sin* φ = *0*, *simp-all*)
hence *sin* ϑ > *0*
using ‹*sin* φ ∗ *sin* ϑ < *0*›
by (*smt mult-nonpos-nonpos*)
have *?ReByz* > *0*
proof−
have *ry* ∗ (*1* + *rz*^{2}) ∗ *sin* ϑ > *0*
using ‹*ry* > *0*› ‹*rz* > *0*›
using ‹*sin* ϑ > *0*›
by (*smt mult-pos-neg mult-pos-pos zero-less-power*)
moreover
have *rz* ∗ (*1* + *ry*^{2}) ∗ *sin* φ < *0*
using ‹*ry* > *0*› ‹*rz* > *0*›
using ‹*sin* φ < *0*›
by (*smt mult-pos-neg mult-pos-pos zero-less-power*)
ultimately
show *?thesis*
by *simp*
qed
have *?Ayz* < *0*
using ‹*Re ?Byz* / *?Ayz* < −*1*› ‹*?Ayz* ≠ *0*› ‹*Re ?Byz* = *?ReByz*› ‹*?ReByz* > *0*›
by (*smt divide-less-0-iff*)
hence *sin* (φ − ϑ) < *0*
using ‹*ry* > *0*› ‹*rz* > *0*›
by (*smt mult-nonneg-nonneg*)

have *1* / *ru* + *ru* ≥ *1* / *rz* + *rz*
using *Pasch-fun-mono*[*of ru rz*] ‹*0* < *ru*› ‹*ru* ≤ *rz*› ‹*rz* < *1*›
by *simp*
hence *sin* ϑ ∗ (*1* / *ru* + *ru*) ≥ *sin* ϑ ∗ (*1* / *rz* + *rz*)
using ‹*sin* ϑ > *0*›
by *auto*
thus *?thesis*
using ‹*ru* > *0*› ‹*rz* > *0*› ‹*ru* ≤ *rz*› ‹*rz* < *1*› ‹*?Ayz* < *0*› ‹*sin* (φ − ϑ) < *0*›
using *divide-right-mono-neg*[*of sin* ϑ ∗ (*1* / *rz* + *rz*) *sin* ϑ ∗ (*1* / *ru* + *ru*) *2* ∗ *sin* (φ − ϑ)]
by (*subst Kz*, *subst Ku*) *simp*
qed

have *intersects-x-axis-positive ?yu*
using ∗ ‹*y'* = *cor ry* ∗ *cis* ϑ› ‹*z'* = *cor rz* ∗ *cis* φ› ‹*u'* = *cor ru* ∗ *cis* φ›
apply *simp*
apply (*transfer fixing*: *ry rz ru* ϑ φ)
apply (*transfer fixing*: *ry rz ru* ϑ φ)

proof−
have *Re ?Byu / ?Ayu* < −*1*
using ‹*Re ?Byz / ?Ayz* < −*1*› ‹*?ReByz / ?Ayz* ≥ *?ReByu / ?Ayu*›
by (*subst* (*asm*) ‹*Re ?Byz* = *?ReByz*›, *subst* ‹*Re ?Byu* = *?ReByu*›) *simp*
thus *intersects-x-axis-positive-cmat* (*poincare-line-cvec-cmat* (*of-complex-cvec* (*cor ry* ∗ *cis* ϑ)) (*of-complex-cvec* (*cor ru* ∗ *cis* φ)))
using *yu*
by *simp*
qed

let *?a* = *calc-x-axis-intersection ?yu*
have *?a* ∈ *positive-x-axis ?a* ∈ *circline-set ?yu ?a* ∈ *unit-disc*
using ‹*intersects-x-axis-positive ?yu*›
using *intersects-x-axis-positive-iff*′[*of ?yu*] ‹*y* ≠ *u*›
by *auto*

then obtain *a*′ **where** *a*′: *?a* = *of-complex a*′ *is-real a*′ *Re a*′ > *0 Re a*′ < *1*
unfolding *positive-x-axis-def circline-set-x-axis*
by (*auto simp add*: *cmod-eq-Re*)

have *intersects-x-axis ?yz intersects-x-axis ?yu*
using ‹*intersects-x-axis-positive ?yz*› ‹*intersects-x-axis-positive ?yu*›
by *auto*

show ∃ *a*. *a* ∈ *unit-disc* ∧ *poincare-between u a y* ∧ *poincare-between* 0_h *a* (*of-complex v*)
proof (*rule-tac x*=*?a* **in** *exI*, *safe*)
show *poincare-between u ?a y*
using *poincare-between-x-axis-intersection*[*of y u ?a*]
using *calc-x-axis-intersection*[*OF is-poincare-line-poincare-line*[*OF* ‹*y* ≠ *u*›] ‹*intersects-x-axis ?yu*›]
using *calc-x-axis-intersection-in-unit-disc*[*OF is-poincare-line-poincare-line*[*OF* ‹*y* ≠ *u*›] ‹*intersects-x-axis ?yu*›]
using *in-disc* ‹*y* ≠ *u*› ‹*y* ∉ *circline-set x-axis*› ‹*u* ∉ *circline-set x-axis*›
using ∗ ‹φ = *Arg u*′› ‹ϑ = *Arg y*′› ‹φ ∗ ϑ < *0*›
by (*subst poincare-between-rev*, *auto simp add*: *mult.commute*)
next
show *poincare-between* 0_h *?a* (*of-complex v*)
proof−
have −*?ReByz / ?Ayz* ≤ −*?ReByu / ?Ayu*
using ‹*?ReByz / ?Ayz* ≥ *?ReByu / ?Ayu*›
by *linarith*
have *outward ?yz ?yu*
using ∗ ‹*y*′ = *cor ry* ∗ *cis* ϑ› ‹*z*′ = *cor rz* ∗ *cis* φ› ‹*u*′ = *cor ru* ∗ *cis* φ›
apply *simp*
apply (*transfer fixing*: *ry rz ru* ϑ φ)
apply (*transfer fixing*: *ry rz ru* ϑ φ)
apply (*subst yz yu*)+
unfolding *outward-cmat-def*
apply (*simp only*: *Let-def prod.case*)
apply (*subst yz yu*)+
using ‹−*?ReByz / ?Ayz* ≤ −*?ReByu / ?Ayu*›
by *simp*
hence *Re a*′ ≤ *Re v*
using ‹*?v* = *of-complex v*›
using ‹*?a* = *of-complex a*′›
using ‹*intersects-x-axis-positive ?yz*› ‹*intersects-x-axis-positive ?yu*›
using *outward*[*OF is-poincare-line-poincare-line*[*OF* ‹*y* ≠ *z*›] *is-poincare-line-poincare-line*[*OF* ‹*y* ≠ *u*›]]
by *simp*
thus *?thesis*
using ‹*?v* = *of-complex v*›
using *poincare-between-x-axis-0uv*[*of Re a*′ *Re v*] *a*′ *v*
by *simp*
qed
next
show *?a* ∈ *unit-disc*
by *fact*
qed
qed

next
  show *x* ∈ *unit-disc v* ∈ *unit-disc x* ≠ *v*
    using *assms*
    by *auto*
next
  fix *M x v*
  let *?Mx* = *moebius-pt M x* **and** *?Mv* = *moebius-pt M v*
  assume *1*: *unit-disc-fix M x* ∈ *unit-disc v* ∈ *unit-disc x* ≠ *v*
  assume *2*: *?P ?Mx ?Mv*
  show *?P x v*
  proof *safe*
    fix *y z u*
    let *?My* = *moebius-pt M y* **and** *?Mz* = *moebius-pt M z* **and** *?Mu* = *moebius-pt M u*
    assume *distinct* [*x*, *y*, *z*, *u*, *v*] ¬ *poincare-collinear* {*x*, *y*, *z*} *y* ∈ *unit-disc z* ∈ *unit-disc u* ∈ *unit-disc*
        poincare-between x u z poincare-between y v z
    hence ∃ *Ma*. *Ma* ∈ *unit-disc* ∧ *poincare-between ?Mu Ma ?My* ∧ *poincare-between ?Mx Ma ?Mv*
      using *1 2*[*rule-format*, *of ?My ?Mz ?Mu*]
      by *simp*
    then obtain *Ma* **where** *Ma*: *Ma* ∈ *unit-disc poincare-between ?Mu Ma ?My* ∧ *poincare-between ?Mx Ma ?Mv*
      by *blast*
    let *?a* = *moebius-pt* (−*M*) *Ma*
    let *?Ma* = *moebius-pt M ?a*
    have *?Ma* = *Ma*
      by (*metis moebius-pt-invert uminus-moebius-def*)
    hence *?Ma* ∈ *unit-disc poincare-between ?Mu ?Ma ?My* ∧ *poincare-between ?Mx ?Ma ?Mv*
      using *Ma*
      by *auto*
    thus ∃ *a*. *a* ∈ *unit-disc* ∧ *poincare-between u a y* ∧ *poincare-between x a v*
      using *unit-disc-fix-moebius-inv*[*OF* ‹*unit-disc-fix M*›] ‹*unit-disc-fix M*› ‹*Ma* ∈ *unit-disc*›
      using ‹*u* ∈ *unit-disc*› ‹*v* ∈ *unit-disc*› ‹*x* ∈ *unit-disc*› ‹*y* ∈ *unit-disc*›
      by (*rule-tac x*=*?a* **in** *exI*, *simp del*: *moebius-pt-comp-inv-right*)
  qed
qed
thus *?thesis*
  using *assms*
  by *auto*
qed

Pasch axiom, only degenerative cases.

lemma *Pasch-deg*:
  assumes *x* ∈ *unit-disc* **and** *y* ∈ *unit-disc* **and** *z* ∈ *unit-disc* **and** *u* ∈ *unit-disc* **and** *v* ∈ *unit-disc*
  assumes ¬ *distinct* [*x*, *y*, *z*, *u*, *v*] ∨ *poincare-collinear* {*x*, *y*, *z*}
  assumes *poincare-between x u z* **and** *poincare-between y v z*
  shows ∃ *a*. *a* ∈ *unit-disc* ∧ *poincare-between u a y* ∧ *poincare-between x a v*
proof(*cases poincare-collinear* {*x*, *y*, *z*})
  case *True*
  hence *poincare-between x y z* ∨ *poincare-between y x z* ∨ *poincare-between y z x*
    using *assms*(*1*, *2*, *3*) *poincare-collinear3-between poincare-between-rev* **by** *blast*
  show *?thesis*
  proof(*cases poincare-between x y z*)
    case *True*
    have *poincare-between x y v*
      using *True assms poincare-between-transitivity*
      by (*meson poincare-between-rev*)
    thus *?thesis*
      using *assms*(*2*)
      by (*rule-tac x*=*y* **in** *exI*, *simp*)
  next
    case *False*
    hence *poincare-between y x z* ∨ *poincare-between y z x*
      using ‹*poincare-between x y z* ∨ *poincare-between y x z* ∨ *poincare-between y z x*›
      by *simp*
    show *?thesis*
    proof(*cases poincare-between y x z*)
      case *True*
      hence *poincare-between u x y*

using *assms*
by (*meson poincare-between-rev poincare-between-transitivity*)
thus *?thesis*
using *assms*
by (*rule-tac x=x* **in** *exI*, *simp*)
next
case *False*
hence *poincare-between y z x*
using ‹*poincare-between y x z* ∨ *poincare-between y z x*›
by *auto*
hence *poincare-between x z v*
using *assms*
by (*meson poincare-between-rev poincare-between-transitivity*)
hence *poincare-between x u v*
using *assms poincare-between-transitivity poincare-between-rev*
by (*smt poincare-between-sum-distances*)
thus *?thesis*
using *assms*
by (*rule-tac x=u* **in** *exI*, *simp*)
qed
qed
next
case *False*
hence ¬ *distinct* [*x*, *y*, *z*, *u*, *v*]
using *assms*(*6*) **by** *auto*
show *?thesis*
proof(*cases u=z*)
case *True*
thus *?thesis*
using *assms*
apply(*rule-tac x=v* **in** *exI*)
by(*simp add*:*poincare-between-rev*)
next
case *False*
hence $x \neq z$
using *assms poincare-between-sandwich* **by** *blast*
show *?thesis*
proof(*cases v=z*)
case *True*
thus *?thesis*
using *assms*
by (*rule-tac x=u* **in** *exI*, *simp*)
next
case *False*
hence $y \neq z$
using *assms poincare-between-sandwich* **by** *blast*
show *?thesis*
proof(*cases u = x*)
case *True*
thus *?thesis*
using *assms*
by (*rule-tac x=x* **in** *exI*, *simp*)
next
case *False*
have $x \neq y$
using *assms* ‹¬ *poincare-collinear* {*x*, *y*, *z*}›
by *fastforce*
have $x \neq v$
using *assms* ‹¬ *poincare-collinear* {*x*, *y*, *z*}›
by (*metis insert-commute poincare-between-poincare-collinear*)
have $u \neq y$
using *assms* ‹¬ *poincare-collinear* {*x*, *y*, *z*}›
using *poincare-between-poincare-collinear* **by** *blast*
have $u \neq v$
proof(*rule ccontr*)
assume ¬ $u \neq v$

```
        hence poincare-between x v z
          using assms by auto
        hence x ∈ circline-set (poincare-line z v)
          using poincare-between-rev[of x v z]
          using poincare-between-poincare-line-uvz[of z v x]
          using assms ‹v ≠ z›
          by auto
        have y ∈ circline-set (poincare-line z v)
          using assms ‹¬ u ≠ v›
          using poincare-between-rev[of y v z]
          using poincare-between-poincare-line-uvz[of z v y]
          using assms ‹v ≠ z›
          by auto
        have z ∈ circline-set (poincare-line z v)
          using ex-poincare-line-two-points[of z v] ‹v ≠ z›
          by auto
        have is-poincare-line (poincare-line z v)
          using ‹v ≠ z›
          by auto
        hence poincare-collinear {x, y, z}
          using ‹x ∈ circline-set (poincare-line z v)›
          using ‹y ∈ circline-set (poincare-line z v)›
          using ‹z ∈ circline-set (poincare-line z v)›
          unfolding poincare-collinear-def
          by (rule-tac x=poincare-line z v in exI, simp)
        thus False
          using ‹¬ poincare-collinear {x, y, z}› by simp
      qed
      have v = y
        using ‹u ≠ v› ‹u ≠ y› ‹x ≠ v› ‹x ≠ y› ‹u ≠ x› ‹y ≠ z› ‹v ≠ z› ‹x ≠ z› ‹u ≠ z›
        using ‹¬ distinct [x, y, z, u, v]›
        by auto
      thus ?thesis
        using assms
        by (rule-tac x=y in exI, simp)
    qed
  qed
 qed
qed
```

Axiom of Pasch

```
lemma Pasch:
  assumes x ∈ unit-disc and y ∈ unit-disc and z ∈ unit-disc and u ∈ unit-disc and v ∈ unit-disc
  assumes poincare-between x u z and poincare-between y v z
  shows ∃ a. a ∈ unit-disc ∧ poincare-between u a y ∧ poincare-between x a v
proof(cases distinct [x, y, z, u, v] ∧ ¬ poincare-collinear {x, y, z})
  case True
  thus ?thesis
    using assms Pasch-nondeg by auto
next
  case False
  thus ?thesis
    using assms Pasch-deg by auto
qed
```

11.2 Segment construction axiom

```
lemma segment-construction:
  assumes x ∈ unit-disc and y ∈ unit-disc
  assumes a ∈ unit-disc and b ∈ unit-disc
  shows ∃ z. z ∈ unit-disc ∧ poincare-between x y z ∧ poincare-distance y z = poincare-distance a b
proof−
  obtain d where d: d = poincare-distance a b
    by auto
  have d ≥ 0
    using assms
```

by (*simp add*: *d poincare-distance-ge0*)

have ∃ *z*. *z* ∈ *unit-disc* ∧ *poincare-between x y z* ∧ *poincare-distance y z* = *d* (**is** *?P x y*)
proof (*cases x* = *y*)
case *True*
have ∃ *z*. *z* ∈ *unit-disc* ∧ *poincare-distance x z* = *d*
proof (*rule wlog-zero*)
show ∃ *z*. *z* ∈ *unit-disc* ∧ *poincare-distance* 0_h *z* = *d*
using *ex-x-axis-poincare-distance-negative*[*of d*] ‹*d* ≥ *0*›
by *blast*
next
show *x* ∈ *unit-disc*
by *fact*
next
fix *a u*
assume *u* ∈ *unit-disc cmod a* < *1*
assume ∃*z*. *z* ∈ *unit-disc* ∧ *poincare-distance* (*moebius-pt* (*blaschke a*) *u*) *z* = *d*
then obtain *z* **where** ∗: *z* ∈ *unit-disc poincare-distance* (*moebius-pt* (*blaschke a*) *u*) *z* = *d*
by *auto*
obtain *z'* **where** *z'*: *z* = *moebius-pt* (*blaschke a*) *z' z'* ∈ *unit-disc*
using ‹*z* ∈ *unit-disc*›
using *unit-disc-fix-iff*[*of blaschke a*] ‹*cmod a* < *1*›
using *blaschke-unit-disc-fix*[*of a*]
by *blast*

show ∃*z*. *z* ∈ *unit-disc* ∧ *poincare-distance u z* = *d*
using ∗ *z'* ‹*u* : *unit-disc*›
using *blaschke-unit-disc-fix*[*of a*] ‹*cmod a* < *1*›
by (*rule-tac x*=*z'* **in** *exI*, *simp*)
qed
thus *?thesis*
using ‹*x* = *y*›
unfolding *poincare-between-def*
by *auto*
next
case *False*
show *?thesis*
proof (*rule wlog-positive-x-axis*[**where** *P*=λ *y x*. *?P x y*])
fix *x*
assume *is-real x 0* < *Re x Re x* < *1*

then obtain *z* **where** *z*: *is-real z Re z* ≤ *0* − *1* < *Re z of-complex z* ∈ *unit-disc*
of-complex z ∈ *unit-disc of-complex z* ∈ *circline-set x-axis poincare-distance* 0_h (*of-complex z*) = *d*
using *ex-x-axis-poincare-distance-negative*[*of d*] ‹*d* ≥ *0*›
by *auto*

have *poincare-between* (*of-complex x*) 0_h (*of-complex z*)
proof (*cases z* = *0*)
case *True*
thus *?thesis*
unfolding *poincare-between-def*
by *auto*
next
case *False*
have *x* ≠ *0*
using ‹*is-real x*› ‹*Re x* > *0*›
by *auto*
thus *?thesis*
using *poincare-between-x-axis-u0v*[*of x z*]
using *z* ‹*is-real x*› ‹*x* ≠ *0*› ‹*Re x* > *0*› *False*
using *complex-eq-if-Re-eq mult-pos-neg*
by *fastforce*
qed
thus *?P* (*of-complex x*) 0_h
using ‹*poincare-distance* 0_h (*of-complex z*) = *d*› ‹*of-complex z* ∈ *unit-disc*›
by *blast*

```
  next
    show x ∈ unit-disc y ∈ unit-disc
      by fact+
  next
    show y ≠ x using ‹x ≠ y› by simp
  next
    fix M u v
    assume unit-disc-fix M u ∈ unit-disc v ∈ unit-disc u ≠ v
    assume ?P (moebius-pt M v) (moebius-pt M u)
     then obtain z where *: z ∈ unit-disc poincare-between (moebius-pt M v) (moebius-pt M u) z poincare-distance
(moebius-pt M u) z = d
      by auto
    obtain z' where z': z = moebius-pt M z' z' ∈ unit-disc
        using ‹z ∈ unit-disc›
        using unit-disc-fix-iff[of M] ‹unit-disc-fix M›
        by blast
      thus ?P v u
        using * ‹u ∈ unit-disc› ‹v ∈ unit-disc› ‹unit-disc-fix M›
        by auto
    qed
  qed
  thus ?thesis
    using assms d
    by auto
qed
```

11.3 Five segment axiom

```
lemma five-segment-axiom:
  assumes
     in-disc: x ∈ unit-disc  y ∈ unit-disc z ∈ unit-disc u ∈ unit-disc and
     in-disc': x' ∈ unit-disc y' ∈ unit-disc z' ∈ unit-disc u' ∈ unit-disc and
      x ≠ y and
      betw: poincare-between x y z poincare-between x' y' z' and
      xy: poincare-distance x y = poincare-distance x' y' and
      xu: poincare-distance x u = poincare-distance x' u' and
      yu: poincare-distance y u = poincare-distance y' u' and
      yz: poincare-distance y z = poincare-distance y' z'
    shows
     poincare-distance z u = poincare-distance z' u'
proof−
  from assms obtain M where
  M: unit-disc-fix-f M M x = x' M u = u' M y = y'
    using unit-disc-fix-f-congruent-triangles[of x y u]
    by blast
  have M z = z'
  proof (rule unique-poincare-distance-on-ray[where u=x' and v=y' and y=M z and z=z' and d=poincare-distance
x z])
    show 0 ≤ poincare-distance x z
      using poincare-distance-ge0 in-disc
      by simp
  next
    show x' ≠ y'
      using M ‹x ≠ y›
      using in-disc in-disc' poincare-distance-eq-0-iff xy
      by auto
  next
    show poincare-distance x' (M z) = poincare-distance x z
      using M in-disc
      unfolding unit-disc-fix-f-def
      by auto
  next
    show M z ∈ unit-disc
      using M in-disc
      unfolding unit-disc-fix-f-def
      by auto
```

next
  show *poincare-distance* x' z' = *poincare-distance* x z
    using *xy yz betw*
    using *poincare-between-sum-distances*[*of* x y z]
    using *poincare-between-sum-distances*[*of* x' y' z']
    using *in-disc in-disc'*
    by *auto*
next
  show *poincare-between* x' y' (M z)
    using *M*
    using *in-disc betw*
    unfolding *unit-disc-fix-f-def*
    by *auto*
qed *fact+*
thus *?thesis*
  using ‹*unit-disc-fix-f M*›
  using *in-disc in-disc'*
  ‹M u = u'›
  unfolding *unit-disc-fix-f-def*
  by *auto*
qed

11.4 Upper dimension axiom

lemma *upper-dimension-axiom*:
  assumes *in-disc*: $x \in$ *unit-disc* $y \in$ *unit-disc* $z \in$ *unit-disc* $u \in$ *unit-disc* $v \in$ *unit-disc*
  assumes *poincare-distance* x u = *poincare-distance* x v
      poincare-distance y u = *poincare-distance* y v
      poincare-distance z u = *poincare-distance* z v
      $u \neq v$
  shows *poincare-between* x y z $\vee$ *poincare-between* y z x $\vee$ *poincare-between* z x y
proof (*cases* $x = y \vee y = z \vee x = z$)
  case *True*
  thus *?thesis*
    using *in-disc*
    by *auto*
next
  case *False*
  hence $x \neq y$ $x \neq z$ $y \neq z$
    by *auto*
  let *?cong* = λ a b a' b'. *poincare-distance* a b = *poincare-distance* a' b'
  have $\forall$ z u v. $z \in$ *unit-disc* $\wedge$ $u \in$ *unit-disc* $\wedge$ $v \in$ *unit-disc* $\wedge$
        ?cong x u x v $\wedge$ *?cong* y u y v $\wedge$ *?cong* z u z v $\wedge$ $u \neq v$ $\longrightarrow$
        poincare-collinear $\{x, y, z\}$ (**is** *?P* x y)
  proof (*rule wlog-positive-x-axis*[**where** P=*?P*])
    fix x
    assume x: *is-real* x $0 <$ *Re* x *Re* $x < 1$
    hence $x \neq 0$
      by *auto*
    have $0_h \in$ *circline-set x-axis*
      by *simp*
    show *?P* 0_h (*of-complex* x)
    proof *safe*
      fix z u v
      assume *in-disc*: $z \in$ *unit-disc* $u \in$ *unit-disc* $v \in$ *unit-disc*
      then obtain z' u' v' **where** *zuv*: z = *of-complex* z' u = *of-complex* u' v = *of-complex* v'
        using *inf-or-of-complex*[*of* z] *inf-or-of-complex*[*of* u] *inf-or-of-complex*[*of* v]
        by *auto*

      assume *cong*: *?cong* 0_h u 0_h v *?cong* (*of-complex* x) u (*of-complex* x) v *?cong* z u z v $u \neq v$

      let *?r0* = *poincare-distance* 0_h u **and**
        ?rx = *poincare-distance* (*of-complex* x) u

      have *?r0* > 0 *?rx* > 0
        using *in-disc cong*

```
      using poincare-distance-eq-0-iff[of 0h u] poincare-distance-ge0[of 0h u]
      using poincare-distance-eq-0-iff[of 0h v] poincare-distance-ge0[of 0h v]
      using poincare-distance-eq-0-iff[of of-complex x u] poincare-distance-ge0[of of-complex x u]
      using poincare-distance-eq-0-iff[of of-complex x v] poincare-distance-ge0[of of-complex x v]
      using x
      by (auto simp add: cmod-eq-Re)

    let ?pc0 = poincare-circle 0h ?r0 and
        ?pcx = poincare-circle (of-complex x) ?rx
    have u ∈ ?pc0 ∩ ?pcx v ∈ ?pc0 ∩ ?pcx
      using in-disc cong
      by (auto simp add: poincare-circle-def)
    hence u = conjugate v
      using intersect-poincare-circles-x-axis[of 0 x ?r0 ?rx u v]
      using x ‹x ≠ 0› ‹u ≠ v› ‹?r0 > 0› ‹?rx > 0›
      by simp

    let ?ru = poincare-distance u z
    have ?ru > 0
      using poincare-distance-ge0[of u z] in-disc
      using cong
      using poincare-distance-eq-0-iff[of z u] poincare-distance-eq-0-iff[of z v]
      using poincare-distance-eq-0-iff
      by force

    have z ∈ poincare-circle u ?ru ∩ poincare-circle v ?ru
      using cong in-disc
      unfolding poincare-circle-def
      by (simp add: poincare-distance-sym)

    hence is-real z'
      using intersect-poincare-circles-conjugate-centers[of u v ?ru z] ‹u = conjugate v› zuv
      using in-disc ‹u ≠ v› ‹?ru > 0›
      by simp

    thus poincare-collinear {0h, of-complex x, z}
      using poincare-line-0-real-is-x-axis[of of-complex x] x ‹x ≠ 0› zuv ‹0h ∈ circline-set x-axis›
      unfolding poincare-collinear-def
      by (rule-tac x=x-axis in exI, auto simp add: circline-set-x-axis)
  qed
next
  fix M x y
  assume 1: unit-disc-fix M x ∈ unit-disc y ∈ unit-disc x ≠ y
  assume 2: ?P (moebius-pt M x) (moebius-pt M y)
  show ?P x y
  proof safe
    fix z u v
    assume z ∈ unit-disc u ∈ unit-disc v ∈ unit-disc
           ?cong x u x v ?cong y u y v ?cong z u z v u ≠ v
    hence poincare-collinear {moebius-pt M x, moebius-pt M y, moebius-pt M z}
      using 1 2[rule-format, of moebius-pt M z moebius-pt M u moebius-pt M v]
      by simp
    then obtain p where is-poincare-line p {moebius-pt M x, moebius-pt M y, moebius-pt M z} ⊆ circline-set p
      unfolding poincare-collinear-def
      by auto
    thus poincare-collinear {x, y, z}
      using ‹unit-disc-fix M›
      unfolding poincare-collinear-def
      by (rule-tac x=moebius-circline (−M) p in exI, auto)
  qed
qed fact+

thus ?thesis
  using assms
  using poincare-collinear3-between[of x y z]
  using poincare-between-rev
```

by *auto*
qed

11.5 Lower dimension axiom

lemma *lower-dimension-axiom*:
  shows $\exists$ *a* $\in$ *unit-disc*. $\exists$ *b* $\in$ *unit-disc*. $\exists$ *c* $\in$ *unit-disc*.
    $\neg$ *poincare-between a b c* $\wedge$ $\neg$ *poincare-between b c a* $\wedge$ $\neg$ *poincare-between c a b*
proof−
  let *?u* = *of-complex* (*1/2*) **and** *?v* = *of-complex* (i/*2*)
  have *1*: 0_h $\in$ *unit-disc* **and** *2*: *?u* $\in$ *unit-disc* **and** *3*: *?v* $\in$ *unit-disc*
    by *simp-all*
  have $*$: $\neg$ *poincare-collinear* {0_h, *?u*, *?v*}
  proof (*rule ccontr*)
    assume $\neg$ *?thesis*
    then obtain *p* **where** *is-poincare-line p* {0_h, *?u*, *?v*} $\subseteq$ *circline-set p*
      unfolding *poincare-collinear-def*
      by *auto*
    moreover
    have *of-complex* (*1* / *2*) $\neq$ *of-complex* (i / *2*)
      using *of-complex-inj*
      by *fastforce*
    ultimately
    have 0_h $\in$ *circline-set* (*poincare-line ?u ?v*)
      using *unique-poincare-line*[*of ?u ?v p*]
      by *auto*
    thus *False*
      unfolding *circline-set-def*
      by *simp* (*transfer*, *transfer*, *simp add*: *vec-cnj-def*)
  qed
  show *?thesis*
    apply (*rule-tac x*=0_h **in** *bexI*, *rule-tac x*=*?u* **in** *bexI*, *rule-tac x*=*?v* **in** *bexI*)
    apply (*rule ccontr*, *auto*)
    using $*$
    using *poincare-between-poincare-collinear*[*OF 1 2 3*]
    using *poincare-between-poincare-collinear*[*OF 2 3 1*]
    using *poincare-between-poincare-collinear*[*OF 3 1 2*]
    by (*metis insert-commute*)+
qed

11.6 Negated Euclidean axiom

lemma *negated-euclidean-axiom-aux*:
  assumes *on-circline H* (*of-complex* (*1/2* + i/*2*)) **and** *is-poincare-line H*
  assumes *intersects-x-axis-positive H*
  shows $\neg$ *intersects-y-axis-positive H*
  using *assms*
proof (*transfer*, *transfer*)
  fix *H*
  assume *hh*: *hermitean H* $\wedge$ *H* $\neq$ *mat-zero is-poincare-line-cmat H*
  obtain *A B C D* **where** *H* = (*A*, *B*, *C*, *D*)
    by (*cases H*, *auto*)
  hence $*$: *is-real A H* = (*A*, *B*, *cnj B*, *A*) (*cmod B*)^2 > (*cmod A*)^2
    using *hermitean-elems*[*of A B C D*] *hh*
    by *auto*

  assume *intersects-x-axis-positive-cmat H*
  hence *Re A* $\neq$ *0 Re B* / *Re A* < − *1*
    using $*$
    by *auto*

  assume *on-circline-cmat-cvec H* (*of-complex-cvec* (*1* / *2* + i / *2*))
  hence *6*$*$*A* + *4*$*$*Re B* + *4*$*$*Im B* = *0*
    using $*$
    unfolding *of-real-mult*
    apply (*subst Re-express-cnj*[*of B*])

```
    apply (subst Im-express-cnj[of B])
    apply (simp add: vec-cnj-def)
    apply (simp add: field-simps)
    done
  hence Re (6*A + 4*Re B + 4*Im B) = 0
    by simp
  hence 3*Re A + 2*Re B + 2*Im B = 0
    using ‹is-real A›
    by simp

  hence 3/2 + Re B/Re A + Im B/Re A = 0
    using ‹Re A ≠ 0›
    by (simp add: field-simps)

  hence −Im B/Re A − 3/2 < −1
    using ‹Re B / Re A < −1›
    by simp
  hence Im B/Re A > −1/2
    by (simp add: field-simps)
  thus ¬ intersects-y-axis-positive-cmat H
    using *
    by simp
qed

lemma negated-euclidean-axiom:
  shows ∃ a b c d t.
          a ∈ unit-disc ∧ b ∈ unit-disc ∧ c ∈ unit-disc ∧ d ∈ unit-disc ∧ t ∈ unit-disc ∧
          poincare-between a d t ∧ poincare-between b d c ∧ a ≠ d ∧
               (∀ x y. x ∈ unit-disc ∧ y ∈ unit-disc ∧
                      poincare-between a b x ∧ poincare-between x t y ⟶ ¬ poincare-between a c y)
proof−
  let ?a = 0h
  let ?b = of-complex (1/2)
  let ?c = of-complex (i/2)
  let ?dl = (5 − sqrt 17) / 4
  let ?d = of-complex (?dl + i*?dl)
  let ?t = of-complex (1/2 + i/2)

  have ?dl ≠ 0
  proof−
    have (sqrt 17)^2 ≠ 5^2
      by simp
    hence sqrt 17 ≠ 5
      by force
    thus ?thesis
      by simp
  qed

  have ?d ≠ ?a
  proof (rule ccontr)
    assume ¬ ?thesis
    hence ?dl + i*?dl = 0
      by simp
    hence Re (?dl + i*?dl) = 0
      by simp
    thus False
      using ‹?dl ≠ 0›
      by simp
  qed

  have ?dl > 0
  proof−
    have (sqrt 17)^2 < 5^2
      by (simp add: power2-eq-square)
    hence sqrt 17 < 5
      by (rule power2-less-imp-less, simp)
```

```
    thus ?thesis
      by simp
  qed

  have ?a ≠ ?b
    by (metis divide-eq-0-iff of-complex-zero-iff zero-neq-numeral zero-neq-one)

  have ?a ≠ ?c
    by (metis complex-i-not-zero divide-eq-0-iff of-complex-zero-iff zero-neq-numeral)

  show ?thesis
  proof (rule-tac x=?a in exI, rule-tac x=?b in exI, rule-tac x=?c in exI, rule-tac x=?d in exI, rule-tac x=?t in exI,
safe)


    show ?a ∈ unit-disc ?b ∈ unit-disc ?c ∈ unit-disc ?t ∈ unit-disc
      by (auto simp add: cmod-def power2-eq-square)

    have cmod-d: cmod (?dl + i*?dl) = ?dl * sqrt 2
      using ‹?dl > 0›
      unfolding cmod-def
      by (simp add: real-sqrt-mult)

    show ?d ∈ unit-disc
    proof-
      have ?dl < 1 / sqrt 2
      proof-
        have 17^2 < (5 * sqrt 17)^2
          by (simp add: field-simps)
        hence 17 < 5 * sqrt 17
          by (rule power2-less-imp-less, simp)
        hence ?dl^2 < (1 / sqrt 2)^2
          by (simp add: power2-eq-square field-simps)
        thus ?dl < 1 / sqrt 2
          by (rule power2-less-imp-less, simp)
      qed
      thus ?thesis
        using cmod-d
        by (simp add: field-simps)
    qed

    have cmod-d: 1 − (cmod (to-complex ?d))^2 = (−17 + 5*sqrt 17) / 4 (is - = ?cmod-d)
      apply (simp only: to-complex-of-complex)
      apply (subst cmod-d)
      apply (simp add: power-mult-distrib)
      apply (simp add: power2-eq-square field-simps)
      done

    have cmod-d-c: (cmod (to-complex ?d − to-complex ?c))^2 = (17 − 4*sqrt 17) / 4 (is - = ?cmod-dc)
      unfolding cmod-square
      by (simp add: field-simps)

    have cmod-c: 1 − (cmod (to-complex ?c))^2 = 3/4 (is - = ?cmod-c)
      by (simp add: power2-eq-square)

    have xx: ⋀ x::real. x + x = 2*x
      by simp

    have cmod ((to-complex ?b) − (to-complex ?d)) = cmod ((to-complex ?d) − (to-complex ?c))
      by (simp add: cmod-def power2-eq-square field-simps)
    moreover
    have cmod (to-complex ?b) = cmod (to-complex ?c)
      by simp
    ultimately
    have *: poincare-distance-formula' (to-complex ?b) (to-complex ?d) =
            poincare-distance-formula' (to-complex ?d) (to-complex ?c)
```

```
      unfolding poincare-distance-formula'-def
      by simp

    have **: poincare-distance-formula' (to-complex ?d) (to-complex ?c) = (sqrt 17) / 3
      unfolding poincare-distance-formula'-def
    proof (subst cmod-d, subst cmod-c, subst cmod-d-c)
      have (sqrt 17 * 15)^2 ≠ 51^2
        by simp
      hence sqrt 17 * 15 ≠ 51
        by force
      hence sqrt 17 * 15 − 51 ≠ 0
        by simp

      have (5 * sqrt 17)^2 ≠ 17^2
        by simp
      hence 5 * sqrt 17 ≠ 17
        by force
      hence ?cmod-d * ?cmod-c ≠ 0
        by simp
     hence 1 + 2 * (?cmod-dc / (?cmod-d * ?cmod-c)) = (?cmod-d * ?cmod-c + 2 * ?cmod-dc) / (?cmod-d * ?cmod-c)
        using add-frac-num[of ?cmod-d * ?cmod-c 2 * ?cmod-dc 1]
        by (simp add: field-simps)
      also have ... = (64 * (85 − sqrt 17 * 17)) / (64 * (sqrt 17 * 15 − 51))
        by (simp add: field-simps)
      also have ... = (85 − sqrt 17 * 17) / (sqrt 17 * 15 − 51)
        by (rule mult-divide-mult-cancel-left, simp)
      also have ... = sqrt 17 / 3
        by (subst frac-eq-eq, fact, simp, simp add: field-simps)
      finally
      show 1 + 2 * (?cmod-dc / (?cmod-d * ?cmod-c)) = sqrt 17 / 3
        .
    qed

    have sqrt 17 ≥ 3
    proof−
      have (sqrt 17)^2 ≥ 3^2
        by simp
      thus ?thesis
        by (rule power2-le-imp-le, simp)
    qed
    thus poincare-between ?b ?d ?c
      unfolding poincare-between-sum-distances[OF ‹?b ∈ unit-disc› ‹?d ∈ unit-disc› ‹?c ∈ unit-disc›]
      unfolding poincare-distance-formula[OF ‹?b ∈ unit-disc› ‹?d ∈ unit-disc›]
      unfolding poincare-distance-formula[OF ‹?d ∈ unit-disc› ‹?c ∈ unit-disc›]
      unfolding poincare-distance-formula[OF ‹?b ∈ unit-disc› ‹?c ∈ unit-disc›]
      unfolding poincare-distance-formula-def
      apply (subst *, subst xx, subst **, subst arcosh-double)
      apply (simp-all add: cmod-def power2-eq-square)
      done

    show poincare-between ?a ?d ?t
    proof (subst poincare-between-0uv[OF ‹?d ∈ unit-disc› ‹?t ∈ unit-disc› ‹?d ≠ ?a›])
      show ?t ≠ 0_h
      proof (rule ccontr)
        assume ¬ ?thesis
        hence 1/2 + i/2 = 0
          by simp
        hence Re (1/2 + i/2) = 0
          by simp
        thus False
          by simp
      qed
    next
      have 19^2 ≤ (5 * sqrt 17)^2
        by simp
      hence 19 ≤ 5 * sqrt 17
```

by (*rule power2-le-imp-le*, *simp*)
hence *cmod* (*to-complex ?d*) $\leq$ *cmod* (*to-complex ?t*)
by (*simp add*: *Let-def cmod-def power2-eq-square field-simps*)
moreover
have *Arg* (*to-complex ?d*) = *Arg* (*to-complex ?t*)
proof−
have *1*: *to-complex ?d* = ((*5* − *sqrt 17*) / *4*) ∗ (*1* + i)
by (*simp add*: *field-simps*)

have *2*: *to-complex ?t* = (*cor* (*1/2*)) ∗ (*1* + i)
by (*simp add*: *field-simps*)

have $(sqrt\ 17)^2 < 5^2$
by *simp*
hence *sqrt 17* < *5*
by (*rule power2-less-imp-less*, *simp*)
hence *3*: (*5* − *sqrt 17*) / *4* > *0*
by *simp*

have *4*: (*1*::*real*) / *2* > *0*
by *simp*

show *?thesis*
apply (*subst 1*, *subst 2*)
apply (*subst arg-mult-real-positive*[*OF 3*])
apply (*subst arg-mult-real-positive*[*OF 4*])
by *simp*
qed
ultimately
show *let* d' = *to-complex ?d*; t' = *to-complex ?t in Arg* d' = *Arg* $t' \wedge$ *cmod* $d' \leq$ *cmod* t'
by *simp*
qed

show *?a* = *?d* $\Longrightarrow$ *False*
using ‹*?d* ≠ *?a*›
by *simp*

fix *x y*
assume *x* ∈ *unit-disc y* ∈ *unit-disc*

assume *abx*: *poincare-between ?a ?b x*
hence *x* ∈ *circline-set x-axis*
using *poincare-between-poincare-line-uvz*[*of ?a ?b x*] ‹*x* ∈ *unit-disc*› ‹*?a* ≠ *?b*›
using *poincare-line-0-real-is-x-axis*[*of ?b*]
by (*auto simp add*: *circline-set-x-axis*)

have $x \neq 0_h$
using *abx poincare-between-sandwich*[*of ?a ?b*] ‹*?a* ≠ *?b*›
by *auto*

have *x* ∈ *positive-x-axis*
using ‹*x* ∈ *circline-set x-axis*› ‹$x \neq 0_h$› ‹*x* ∈ *unit-disc*›
using *abx poincare-between-x-axis-0uv*[*of 1/2 Re* (*to-complex x*)]
unfolding *circline-set-x-axis positive-x-axis-def*
by (*auto simp add*: *cmod-eq-Re abs-less-iff complex-eq-if-Re-eq*)

assume *acy*: *poincare-between ?a ?c y*
hence *y* ∈ *circline-set y-axis*
using *poincare-between-poincare-line-uvz*[*of ?a ?c y*] ‹*y* ∈ *unit-disc*› ‹*?a* ≠ *?c*›
using *poincare-line-0-imag-is-y-axis*[*of ?c*]
by (*auto simp add*: *circline-set-y-axis*)

have $y \neq 0_h$
using *acy poincare-between-sandwich*[*of ?a ?c*] ‹*?a* ≠ *?c*›
by *auto*

```
    have y ∈ positive-y-axis
    proof−
      have ⋀x. ⟦x ≠ 0; poincare-between 0h (of-complex (i / 2)) (of-complex x); is-imag x; − 1 < Im x⟧ ⟹ 0 < Im x
        by (smt add.left-neutral complex.expand divide-complex-def complex-eq divide-less-0-1-iff divide-less-eq-1-pos imaginary-unit.simps(1) mult.left-neutral of-real-1 of-real-add of-real-divide of-real-eq-0-iff one-add-one poincare-between-y-axis-0uv zero-complex.simps(1) zero-complex.simps(2) zero-less-divide-1-iff)
      thus ?thesis
        using ‹y ∈ circline-set y-axis› ‹y ≠ 0h› ‹y ∈ unit-disc›
        using acy
        unfolding circline-set-y-axis positive-y-axis-def
        by (auto simp add: cmod-eq-Im abs-less-iff)
    qed

    have x ≠ y
      using ‹x ∈ positive-x-axis› ‹y ∈ positive-y-axis›
      unfolding positive-x-axis-def positive-y-axis-def circline-set-x-axis circline-set-y-axis
      by auto

    assume xty: poincare-between x ?t y

    let ?xy = poincare-line x y

    have ?t ∈ circline-set ?xy
      using xty poincare-between-poincare-line-uzv[OF ‹x ≠ y› ‹x ∈ unit-disc› ‹y ∈ unit-disc› ‹?t ∈ unit-disc›]
      by simp

    moreover

    have ?xy ≠ x-axis
      using poincare-line-circline-set[OF ‹x ≠ y›] ‹y ∈ positive-y-axis›
      by (auto simp add: circline-set-x-axis positive-y-axis-def)
    hence intersects-x-axis-positive ?xy
      using intersects-x-axis-positive-iff[of ?xy] ‹x ≠ y› ‹x ∈ unit-disc› ‹x ∈ positive-x-axis›
      by auto

    moreover

    have ?xy ≠ y-axis
      using poincare-line-circline-set[OF ‹x ≠ y›] ‹x ∈ positive-x-axis›
      by (auto simp add: circline-set-y-axis positive-x-axis-def)
    hence intersects-y-axis-positive ?xy
      using intersects-y-axis-positive-iff[of ?xy] ‹x ≠ y› ‹y ∈ unit-disc› ‹y ∈ positive-y-axis›
      by auto

    ultimately

    show False
      using negated-euclidean-axiom-aux[of ?xy] ‹x ≠ y›
      unfolding circline-set-def
      by auto
  qed
qed
```

Alternate form of the Euclidean axiom – this one is much easier to prove

```
lemma negated-euclidean-axiom':
  shows ∃ a b c.
          a ∈ unit-disc ∧ b ∈ unit-disc ∧ c ∈ unit-disc ∧ ¬(poincare-collinear {a, b, c}) ∧
              ¬(∃ x. x ∈ unit-disc ∧
                poincare-distance a x = poincare-distance b x ∧
                poincare-distance a x = poincare-distance c x)
proof−
  let ?a = of-complex (i/2)
  let ?b = of-complex (−i/2)
  let ?c = of-complex (1/5)

  have (i/2) ≠ (−i/2)
```

  by *simp*
hence *?a* ≠ *?b*
  by (*metis to-complex-of-complex*)
have (i/*2*) ≠ (*1*/*5*)
  by *simp*
hence *?a* ≠ *?c*
  by (*metis to-complex-of-complex*)
have (−i/*2*) ≠ (*1*/*5*)
  by (*simp add*: *minus-equation-iff*)
hence *?b* ≠ *?c*
  by (*metis to-complex-of-complex*)

have *?a* ∈ *unit-disc* *?b* ∈ *unit-disc* *?c* ∈ *unit-disc*
  by *auto*

moreover
have ¬(*poincare-collinear* {*?a*, *?b*, *?c*})
  unfolding *poincare-collinear-def*
proof(*rule ccontr*)
  assume ¬ (∄*p*. *is-poincare-line p* ∧ {*?a*, *?b*, *?c*} ⊆ *circline-set p*)
  then obtain *p* **where** *is-poincare-line p* ∧ {*?a*, *?b*, *?c*} ⊆ *circline-set p*
    by *auto*
  let *?ab* = *poincare-line ?a ?b*
  have *p* = *?ab*
    using ‹*is-poincare-line p* ∧ {*?a*, *?b*, *?c*} ⊆ *circline-set p*›
    using *unique-poincare-line*[*of ?a ?b*] ‹*?a* ≠ *?b*› ‹*?a* ∈ *unit-disc*› ‹*?b* ∈ *unit-disc*›
    by *auto*
  have *?c* ∉ *circline-set ?ab*
  proof(*rule ccontr*)
    assume ¬ *?c* ∉ *circline-set ?ab*
    have *poincare-between ?a* 0_h *?b*
      unfolding *poincare-between-def*
      using *cross-ratio-0inf* **by** *auto*
    hence 0_h ∈ *circline-set ?ab*
      using ‹*?a* ≠ *?b*› ‹*?a* ∈ *unit-disc*› ‹*?b* ∈ *unit-disc*›
      using *poincare-between-poincare-line-uzv zero-in-unit-disc*
      by *blast*
    hence *?ab* = *poincare-line* 0_h *?a*
      using *unique-poincare-line*[*of ?a ?b*] ‹*?a* ≠ *?b*› ‹*?a* ∈ *unit-disc*› ‹*?b* ∈ *unit-disc*›
      using ‹*is-poincare-line p* ∧ {*?a*, *?b*, *?c*} ⊆ *circline-set p*›
      using ‹*p* = *?ab*› *poincare-line-circline-set*(*1*) *unique-poincare-line*
      by (*metis add.inverse-neutral divide-minus-left of-complex-zero-iff zero-in-unit-disc*)
    hence (i/*2*) ∗ *cnj*(*1*/*5*) = *cnj*(i/*2*) ∗ (*1*/*5*)
      using *poincare-collinear-zero-iff*[*of* (i/*2*) (*1*/*5*)]
      using ‹*?a* ≠ *?c*› ‹¬ *?c* ∉ *circline-set ?ab*› ‹*?a* ∈ *unit-disc*› ‹*?c* ∈ *unit-disc*› ‹*p* = *?ab*›
      using ‹0_h ∈ *circline-set ?ab*› ‹*is-poincare-line p* ∧ {*?a*, *?b*, *?c*} ⊆ *circline-set p*›
      using *poincare-collinear-def* **by** *auto*
    thus *False*
      by *simp*
  qed
  thus *False*
    using ‹*p* = *?ab*› ‹*is-poincare-line p* ∧ {*?a*, *?b*, *?c*} ⊆ *circline-set p*›
    by *auto*
qed

moreover

have ¬(∃ *x*. *x* ∈ *unit-disc* ∧
        poincare-distance ?a x = *poincare-distance ?b x* ∧
        poincare-distance ?a x = *poincare-distance ?c x*)
proof(*rule ccontr*)
  assume ¬ *?thesis*
  then obtain *x* **where** *x* ∈ *unit-disc* *poincare-distance ?a x* = *poincare-distance ?b x*
        poincare-distance ?a x = *poincare-distance ?c x*
    by *blast*
  let *?x* = *to-complex x*

have *poincare-distance-formula'* (i/*2*) *?x* = *poincare-distance-formula'* (−i/*2*) *?x*
using ‹*poincare-distance ?a x* = *poincare-distance ?b x*›
using ‹*x* ∈ *unit-disc*› ‹*?a* ∈ *unit-disc*› ‹*?b* ∈ *unit-disc*›
by (*metis cosh-dist to-complex-of-complex*)
hence (*cmod* (i / *2* − *?x*))^2 = (*cmod* (− i / *2* − *?x*))^2
unfolding *poincare-distance-formula'-def*
apply (*simp add:field-simps*)
using ‹*x* ∈ *unit-disc*› *unit-disc-cmod-square-lt-1* **by** *fastforce*
hence *Im ?x* = *0*
unfolding *cmod-def*
by (*simp add*: *power2-eq-iff*)

have *1* − (*Re ?x*)^2 ≠ *0*
using ‹*x* ∈ *unit-disc*› *unit-disc-cmod-square-lt-1*
using *cmod-power2* **by** *force*
hence *24* − *24* ∗ (*Re ?x*)^2 ≠ *0*
by *simp*
have *poincare-distance-formula'* (i/*2*) *?x* = *poincare-distance-formula'* (*1*/*5*) *?x*
using ‹*poincare-distance ?a x* = *poincare-distance ?c x*›
using ‹*x* ∈ *unit-disc*› ‹*?a* ∈ *unit-disc*› ‹*?c* ∈ *unit-disc*›
by (*metis cosh-dist to-complex-of-complex*)
hence (*2* + *8* ∗ (*Re ?x*)^2) /(*3* − *3* ∗ (*Re ?x*)^2) = *2* ∗ (*1* − *Re ?x* ∗ *5*)^2 / (*24* − *24* ∗ (*Re ?x*)^2) (**is** *?lhs* = *?rhs*)
unfolding *poincare-distance-formula'-def*
apply (*simp add:field-simps*)
unfolding *cmod-def*
using ‹*Im ?x* = *0*›
by (*simp add:field-simps*)
hence ∗: *?lhs* ∗ (*24* − *24* ∗ (*Re ?x*)^2) = *?rhs* ∗ (*24* − *24* ∗ (*Re ?x*)^2)
using ‹(*24* − *24* ∗ (*Re ?x*)^2) ≠ *0*›
by *simp*
have *?lhs* ∗ (*24* − *24* ∗ (*Re ?x*)^2) = (*2* + *8* ∗ (*Re ?x*)^2) ∗ *8*
using ‹(*24* − *24* ∗ (*Re ?x*)^2) ≠ *0*› ‹*1* − (*Re ?x*)^2 ≠ *0*›
by (*simp add:field-simps*)
have *?rhs* ∗ (*24* − *24* ∗ (*Re ?x*)^2) = *2* ∗ (*1* − *Re ?x* ∗ *5*)^2
using ‹(*24* − *24* ∗ (*Re ?x*)^2) ≠ *0*› ‹*1* − (*Re ?x*)^2 ≠ *0*›
by (*simp add:field-simps*)
hence (*2* + *8* ∗ (*Re ?x*)^2) ∗ *8* = *2* ∗ (*1* − *Re ?x* ∗ *5*)^2
using ∗ ‹*?lhs* ∗ (*24* − *24* ∗ (*Re ?x*)^2) = (*2* + *8* ∗ (*Re ?x*)^2) ∗ *8*›
by *simp*
hence *7* ∗ (*Re ?x*)^2 + *10* ∗ (*Re ?x*) + *7* = *0*
by (*simp add:field-simps comm-ring-1-class.power2-diff*)
thus *False*
using *discriminant-iff*[*of 7 Re* (*to-complex x*) *10 7*] *discrim-def*[*of 7 10 7*]
by *auto*
qed

ultimately show *?thesis*
apply (*rule-tac x=?a* **in** *exI*)
apply (*rule-tac x=?b* **in** *exI*)
apply (*rule-tac x=?c* **in** *exI*)
by *auto*
qed

11.7 Continuity axiom

The set ϕ is on the left of the set ψ

abbreviation *set-order* **where**
set-order A φ ψ ≡ ∀ *x*∈ *unit-disc*. ∀ *y*∈ *unit-disc*. φ *x* ∧ ψ *y* ⟶ *poincare-between A x y*

The point B is between the sets ϕ and ψ

abbreviation *point-between-sets* **where**
point-between-sets φ *B* ψ ≡ ∀ *x*∈ *unit-disc*. ∀ *y*∈ *unit-disc*. φ *x* ∧ ψ *y* ⟶ *poincare-between x B y*

lemma *continuity*:
assumes ∃ *A* ∈ *unit-disc*. *set-order A* φ ψ

```
  shows  ∃ B ∈ unit-disc. point-between-sets φ B ψ
proof (cases (∃ x0 ∈ unit-disc. φ x0) ∧ (∃ y0 ∈ unit-disc. ψ y0))
  case False
  thus ?thesis
    using assms by blast
next
  case True
  then obtain Y0 where ψ Y0 Y0 ∈ unit-disc
    by auto
  obtain A where *: A ∈ unit-disc set-order A φ ψ
    using assms
    by auto
  show ?thesis
  proof(cases ∀ x ∈ unit-disc. φ x ⟶ x = A)
    case True
    thus ?thesis
      using ‹A ∈ unit-disc›
      using poincare-between-nonstrict(1) by blast
  next
    case False
    then obtain X0 where φ X0 X0 ≠ A X0 ∈ unit-disc
      by auto
    have Y0 ≠ A
    proof(rule ccontr)
      assume ¬ Y0 ≠ A
      hence ∀ x ∈ unit-disc. φ x ⟶ poincare-between A x A
        using * ‹ψ Y0›
        by (cases A) force
      hence ∀ x ∈ unit-disc. φ x ⟶ x = A
        using * poincare-between-sandwich by blast
      thus False
        using False by auto
    qed

    show ?thesis
    proof (cases ∃ B ∈ unit-disc. φ B ∧ ψ B)
      case True
      then obtain B where B ∈ unit-disc φ B ψ B
        by auto
      hence ∀ x ∈ unit-disc. φ x ⟶ poincare-between A x B
        using * by auto
      have ∀ y ∈ unit-disc. ψ y ⟶ poincare-between A B y
        using * ‹B ∈ unit-disc› ‹φ B›
        by auto

      show ?thesis
      proof(rule+)
        show B ∈ unit-disc
          by fact
      next
        fix x y
        assume x ∈ unit-disc y ∈ unit-disc φ x ∧ ψ y
        hence poincare-between A x B poincare-between A B y
          using ‹∀ x ∈ unit-disc. φ x ⟶ poincare-between A x B›
          using ‹∀ y ∈ unit-disc. ψ y ⟶ poincare-between A B y›
          by simp+
        thus poincare-between x B y
          using ‹x ∈ unit-disc› ‹y ∈ unit-disc› ‹B ∈ unit-disc› ‹A ∈ unit-disc›
          using poincare-between-transitivity[of A x B y]
          by simp
      qed
    next
      case False
      have poincare-between A X0 Y0
        using ‹φ X0› ‹ψ Y0› * ‹Y0 ∈ unit-disc› ‹X0 ∈ unit-disc›
        by auto
```

```
have ∀ φ. ∀ ψ. set-order A φ ψ ∧ ¬ (∃ B ∈ unit-disc. φ B ∧ ψ B) ∧ φ X0 ∧
          (∃ y ∈ unit-disc. ψ y) ∧ (∃ x ∈ unit-disc. φ x)
        ⟶ (∃ B ∈ unit-disc. point-between-sets φ B ψ)
    (is ?P A X0)
proof (rule wlog-positive-x-axis[where P=?P])
  show A ∈ unit-disc
    by fact
next
  show X0 ∈ unit-disc
    by fact
next
  show A ≠ X0
    using ‹X0 ≠ A› by simp
next
  fix M u v
  let ?M = λ x. moebius-pt M x
  let ?Mu = ?M u and ?Mv = ?M v
  assume hip: unit-disc-fix M u ∈ unit-disc v ∈ unit-disc u ≠ v
        ?P ?Mu ?Mv
  show ?P u v
  proof safe
    fix φ ψ x y
    assume set-order u φ ψ ¬ (∃B∈unit-disc. φ B ∧ ψ B) φ v
          y ∈ unit-disc ψ y x ∈ unit-disc φ x

    let ?Mφ = λ X'. ∃ X. φ X ∧ ?M X = X'
    let ?Mψ = λ X'. ∃ X. ψ X ∧ ?M X = X'

    obtain Mφ where Mφ = ?Mφ by simp
    obtain Mψ where Mψ = ?Mψ by simp

    have Mφ ?Mv
      using ‹φ v› using ‹Mφ = ?Mφ›
      by blast
    moreover
    have ¬ (∃ B ∈unit-disc. Mφ B ∧ Mψ B)
      using ‹¬ (∃B∈unit-disc. φ B ∧ ψ B)›
      using ‹Mφ = ?Mφ› ‹Mψ = ?Mψ›
      by (metis hip(1) moebius-pt-invert unit-disc-fix-discI unit-disc-fix-moebius-inv)
    moreover
    have ∃ y ∈ unit-disc. Mψ y
      using ‹y ∈ unit-disc› ‹ψ y›  ‹Mψ = ?Mψ› ‹unit-disc-fix M›
      by auto
    moreover
    have set-order ?Mu ?Mφ ?Mψ
    proof ((rule ballI)+, rule impI)
      fix Mx My
      assume Mx ∈ unit-disc My ∈ unit-disc ?Mφ Mx ∧ ?Mψ My
      then obtain x y where φ x ∧ ?M x = Mx ψ y ∧ ?M y = My
        by blast

      hence x ∈ unit-disc y ∈ unit-disc
        using ‹Mx ∈ unit-disc› ‹My ∈ unit-disc› ‹unit-disc-fix M›
        by (metis moebius-pt-comp-inv-left unit-disc-fix-discI unit-disc-fix-moebius-inv)+

      hence poincare-between u x y
        using ‹set-order u φ ψ›
        using ‹Mx ∈ unit-disc› ‹My ∈ unit-disc› ‹φ x ∧ ?M x = Mx› ‹ψ y ∧ ?M y = My›
        by blast
      then show poincare-between ?Mu Mx My
        using ‹φ x ∧ ?M x = Mx› ‹ψ y ∧ ?M y = My›
        using ‹x ∈ unit-disc› ‹y ∈ unit-disc› ‹u ∈ unit-disc› ‹unit-disc-fix M›
        using unit-disc-fix-moebius-preserve-poincare-between by blast
    qed

    hence  set-order ?Mu Mφ Mψ
```

```
      using ‹Mφ = ?Mφ› ‹Mψ = ?Mψ›
      by simp
    ultimately
    have ∃ Mb ∈ unit-disc. point-between-sets Mφ Mb Mψ
      using hip(5)
      by blast
    then obtain Mb where bbb:
      Mb ∈ unit-disc point-between-sets ?Mφ Mb ?Mψ
      using ‹Mφ = ?Mφ› ‹Mψ = ?Mψ›
      by auto

    let ?b = moebius-pt (moebius-inv M) Mb
    show ∃ b ∈ unit-disc. point-between-sets φ b ψ
    proof (rule-tac x=?b in bexI, (rule ballI)+, rule impI)
      fix x y
      assume x ∈ unit-disc y ∈ unit-disc φ x ∧ ψ y
      hence poincare-between u x y
        using ‹set-order u φ ψ›
        by blast

      let ?Mx = ?M x and ?My = ?M y

      have ?Mφ ?Mx ?Mψ ?My
        using ‹φ x ∧ ψ y›
        by blast+
      have ?Mx ∈ unit-disc ?My ∈ unit-disc
        using ‹x ∈ unit-disc› ‹unit-disc-fix M› ‹y ∈ unit-disc›
        by auto

      hence poincare-between ?Mx Mb ?My
        using ‹?Mφ ?Mx› ‹?Mψ ?My› ‹?Mx ∈ unit-disc› ‹?My ∈ unit-disc› bbb
        by auto

      then show poincare-between x ?b y
        using ‹unit-disc-fix M›
        using ‹x ∈ unit-disc› ‹y ∈ unit-disc› ‹Mb ∈ unit-disc› ‹?Mx ∈ unit-disc› ‹?My ∈ unit-disc›
        using unit-disc-fix-moebius-preserve-poincare-between[of M x ?b y]
        by auto
    next
      show ?b ∈ unit-disc
        using bbb ‹unit-disc-fix M›
        by auto
    qed
  qed
next
  fix X
  assume xx: is-real X 0 < Re X Re X < 1
  let ?X = of-complex X
  show ?P 0ₕ ?X
  proof ((rule allI)+, rule impI, (erule conjE)+)
    fix φ ψ
    assume set-order 0ₕ φ ψ ¬ (∃B∈unit-disc. φ B ∧ ψ B) φ ?X
           ∃y∈unit-disc. ψ y ∃x∈unit-disc. φ x
    have ?X ∈ unit-disc
      using xx
      by (simp add: cmod-eq-Re)

    have ψpos: ∀ y ∈ unit-disc. ψ y ⟶ (is-real (to-complex y) ∧ Re (to-complex y) > 0)
    proof(rule ballI, rule impI)
      fix y
      let ?y = to-complex y
      assume y ∈ unit-disc ψ y

      hence poincare-between 0ₕ ?X y
        using ‹set-order 0ₕ φ ψ›
        using ‹?X ∈ unit-disc› ‹φ ?X›
```

```
        by auto

      thus is-real ?y ∧ 0 < Re ?y
        using xx ‹?X ∈ unit-disc› ‹y ∈ unit-disc›
    by (metis (mono-tags, opaque-lifting) arg-0-iff of-complex-zero-iff poincare-between-0uv poincare-between-sandwich
to-complex-of-complex unit-disc-to-complex-inj zero-in-unit-disc)
    qed

    have φnoneg: ∀ x ∈ unit-disc. φ x ⟶ (is-real (to-complex x) ∧ Re (to-complex x) ≥ 0)
    proof(rule ballI, rule impI)
      fix x
      assume x ∈ unit-disc φ x

      obtain y where y ∈ unit-disc ψ y
        using ‹∃ y ∈ unit-disc. ψ y› by blast

      let ?x = to-complex x and ?y = to-complex y

      have is-real ?y Re ?y > 0
        using ψpos ‹ψ y› ‹y ∈ unit-disc›
        by auto

      have poincare-between 0h x y
        using ‹set-order 0h φ ψ›
        using ‹x ∈ unit-disc› ‹φ x› ‹y∈unit-disc› ‹ψ y›
        by auto

      thus is-real ?x ∧ 0 ≤ Re ?x
        using ‹x ∈ unit-disc› ‹y ∈ unit-disc› ‹is-real (to-complex y)› ‹ψ y›
        using ‹set-order 0h φ ψ›
        using ‹φ ?X› ‹?X ∈ unit-disc› ‹Re ?y > 0›
       by (metis arg-0-iff le-less of-complex-zero poincare-between-0uv to-complex-of-complex zero-complex.simps(1)
zero-complex.simps(2))
    qed

    have φlessψ: ∀x∈unit-disc. ∀y∈unit-disc. φ x ∧ ψ y ⟶ Re (to-complex x) < Re (to-complex y)
    proof((rule ballI)+, rule impI)
      fix x y
      let ?x = to-complex x and ?y = to-complex y
      assume x ∈ unit-disc y ∈ unit-disc φ x ∧ ψ y

      hence poincare-between 0h x y
        using ‹set-order 0h φ ψ›
        by auto
      moreover
      have is-real ?x Re ?x ≥ 0
        using φnoneg
        using ‹x ∈ unit-disc› ‹φ x ∧ ψ y› by auto
      moreover
      have is-real ?y Re ?y > 0
        using ψpos
        using ‹y ∈ unit-disc› ‹φ x ∧ ψ y› by auto
      ultimately
      have Re ?x ≤ Re ?y
        using ‹x ∈ unit-disc› ‹y ∈ unit-disc›
       by (metis Re-complex-of-real arg-0-iff le-less of-complex-zero poincare-between-0uv rcis-cmod-Arg rcis-zero-arg
to-complex-of-complex)

      have Re ?x ≠ Re ?y
        using ‹φ x ∧ ψ y› ‹is-real ?x› ‹is-real ?y›
        using ‹¬ (∃B∈unit-disc. φ B ∧ ψ B)› ‹x ∈ unit-disc› ‹y ∈ unit-disc›
        by (metis complex.expand unit-disc-to-complex-inj)

      thus Re ?x < Re ?y
        using ‹Re ?x ≤ Re ?y› by auto
    qed
```

have ∃ *b* ∈ *unit-disc*. ∀ *x* ∈ *unit-disc*. ∀ *y* ∈ *unit-disc*.
is-real (*to-complex b*) ∧
(φ *x* ∧ ψ *y* ⟶ (*Re* (*to-complex x*) ≤ *Re* (*to-complex b*) ∧ *Re* (*to-complex b*) ≤ *Re* (*to-complex y*)))
proof−
let *?Phi* = {*x*. (*of-complex* (*cor x*)) ∈ *unit-disc* ∧ φ (*of-complex* (*cor x*))}

have ∀ *x* ∈ *unit-disc*. φ *x* ⟶ *Re* (*to-complex x*) ≤ *Sup ?Phi*
proof(*safe*)
fix *x*
let *?x* = *to-complex x*
assume *x* ∈ *unit-disc* φ *x*
hence *is-real ?x Re ?x* ≥ *0*
using φ*noneg*
by *auto*
hence *cor* (*Re ?x*) = *?x*
using *complex-of-real-Re* **by** *blast*
hence *of-complex* (*cor* (*Re ?x*)) ∈ *unit-disc*
using ‹*x* ∈ *unit-disc*›
by (*metis inf-notin-unit-disc of-complex-to-complex*)
moreover
have φ (*of-complex* (*cor* (*Re ?x*)))
using ‹*cor* (*Re ?x*) = *?x*› ‹φ *x*› ‹*x* ∈ *unit-disc*›
by (*metis inf-notin-unit-disc of-complex-to-complex*)
ultimately
have *Re ?x* ∈ *?Phi*
by *auto*

have ∃ *M*. ∀ *x* ∈ *?Phi*. *x* ≤ *M*
using φ*less*ψ
using ‹∃ *y* ∈ *unit-disc*. ψ *y*›
by (*metis* (*mono-tags*, *lifting*) *Re-complex-of-real le-less mem-Collect-eq to-complex-of-complex*)

thus *Re ?x* ≤ *Sup ?Phi*
using *cSup-upper*[*of Re ?x ?Phi*]
unfolding *bdd-above-def*
using ‹*Re ?x* ∈ *?Phi*›
by *auto*
qed

have ∀ *y* ∈ *unit-disc*. ψ *y* ⟶ *Sup ?Phi* ≤ *Re* (*to-complex y*)
proof (*safe*)
fix *y*
let *?y* = *to-complex y*
assume ψ *y y* ∈ *unit-disc*
show *Sup ?Phi* ≤ *Re ?y*
proof (*rule ccontr*)
assume ¬ *?thesis*
hence *Re ?y* < *Sup ?Phi*
by *auto*

have ∃ *x*. φ (*of-complex* (*cor x*)) ∧ (*of-complex* (*cor x*)) ∈ *unit-disc*
proof −
obtain *x'* **where** *x'* ∈ *unit-disc* φ *x'*
using ‹∃ *x* ∈ *unit-disc*. φ *x*› **by** *blast*
let *?x'* = *to-complex x'*
have *is-real ?x'*
using ‹*x'* ∈ *unit-disc*› ‹φ *x'*›
using φ*noneg*
by *auto*
hence *cor* (*Re ?x'*) = *?x'*
using *complex-of-real-Re* **by** *blast*
hence *x'* = *of-complex* (*cor* (*Re ?x'*))
using ‹*x'* ∈ *unit-disc*›
by (*metis inf-notin-unit-disc of-complex-to-complex*)
show *?thesis*

```
          apply (rule-tac x=Re ?x' in exI)
          using ‹x' ∈ unit-disc›
          apply (subst (asm) ‹x' = of-complex (cor (Re ?x'))›, simp)
          using ‹φ x'›
          by (subst (asm) (2) ‹x' = of-complex (cor (Re ?x'))›, simp)
      qed

      hence ?Phi ≠ {}
        by auto

      then obtain x where φ (of-complex (cor x)) Re ?y < x
                          (of-complex (cor x)) ∈ unit-disc
        using ‹Re ?y < Sup ?Phi›
        using less-cSupE[of Re ?y ?Phi]
        by auto
      moreover
      have Re ?y < Re (to-complex (of-complex (cor x)))
        using ‹Re ?y < x›
        by simp
      ultimately
      show False
        using φlessψ
        using ‹ψ y› ‹y ∈ unit-disc›
        by (metis less-not-sym)
    qed
  qed

  thus ?thesis
    using ‹∀ x ∈ unit-disc. φ x ⟶ Re (to-complex x) ≤ Sup ?Phi›
    apply (rule-tac x=(of-complex (cor (Sup ?Phi))) in bexI, simp)
    using ‹∃ y∈unit-disc. ψ y› ‹φ ?X› ‹?X ∈ unit-disc›
    using ‹∀ y∈unit-disc. ψ y ⟶ is-real (to-complex y) ∧ 0 < Re (to-complex y)›
      by (smt complex-of-real-Re inf-notin-unit-disc norm-of-real of-complex-to-complex to-complex-of-complex
unit-disc-iff-cmod-lt-1 xx(2))
qed

then obtain B where B ∈ unit-disc is-real (to-complex B)
  ∀ x∈unit-disc. ∀ y∈unit-disc. φ x ∧ ψ y ⟶ Re (to-complex x) ≤ Re (to-complex B) ∧
   Re (to-complex B) ≤ Re (to-complex y)
  by blast

show ∃ b ∈ unit-disc. point-between-sets φ b ψ
proof (rule-tac x=B in bexI)
  show B ∈ unit-disc
    by fact
next
  show point-between-sets φ B ψ
  proof ((rule ballI)+, rule impI)
    fix x y
    let ?x = to-complex x and ?y = to-complex y and ?B = to-complex B
    assume x ∈ unit-disc y ∈ unit-disc φ x ∧ ψ y

    hence Re ?x ≤ Re ?B ∧ Re ?B ≤ Re ?y
      using ‹∀ x∈unit-disc. ∀ y∈unit-disc. φ x ∧ ψ y ⟶ Re (to-complex x) ≤ Re ?B ∧
             Re (to-complex B) ≤ Re (to-complex y)›
      by auto
    moreover
    have is-real ?x Re ?x ≥ 0
      using φnoneg
      using ‹x ∈ unit-disc› ‹φ x ∧ ψ y›
      by auto
    moreover
    have is-real ?y Re ?y > 0
      using ψpos
      using ‹y ∈ unit-disc› ‹φ x ∧ ψ y›
      by auto
```

moreover
have *cor* (*Re ?x*) = *?x*
using *complex-of-real-Re* ‹*is-real ?x*› **by** *blast*
hence *x* = *of-complex* (*cor* (*Re ?x*))
using ‹*x* ∈ *unit-disc*›
by (*metis inf-notin-unit-disc of-complex-to-complex*)
moreover
have *cor* (*Re ?y*) = *?y*
using *complex-of-real-Re* ‹*is-real ?y*› **by** *blast*
hence *y* = *of-complex* (*cor* (*Re ?y*))
using ‹*y* ∈ *unit-disc*›
by (*metis inf-notin-unit-disc of-complex-to-complex*)
moreover
have *cor* (*Re ?B*) = *?B*
using *complex-of-real-Re* ‹*is-real* (*to-complex B*)› **by** *blast*
hence *B* = *of-complex* (*cor* (*Re ?B*))
using ‹*B* ∈ *unit-disc*›
by (*metis inf-notin-unit-disc of-complex-to-complex*)
ultimately
show *poincare-between x B y*
using ‹*is-real* (*to-complex B*)› ‹*x* ∈ *unit-disc*› ‹*y* ∈ *unit-disc*› ‹*B* ∈ *unit-disc*›
using *poincare-between-x-axis-uvw*[*of Re* (*to-complex x*) *Re* (*to-complex B*) *Re* (*to-complex y*)]
by (*smt Re-complex-of-real arg-0-iff poincare-between-nonstrict*(*1*) *rcis-cmod-Arg rcis-zero-arg unit-disc-iff-cmod-lt-1*)
qed
qed
qed
qed
thus *?thesis*
using *False* ‹*φ X0*› ‹*ψ Y0*› * ‹*Y0* ∈ *unit-disc*› ‹*X0* ∈ *unit-disc*›
by *auto*
qed
qed
qed

11.8 Limiting parallels axiom

Auxiliary definitions

definition *poincare-on-line* **where**
poincare-on-line p a b ⟷ *poincare-collinear* {*p*, *a*, *b*}

definition *poincare-on-ray* **where**
poincare-on-ray p a b ⟷ *poincare-between a p b* ∨ *poincare-between a b p*

definition *poincare-in-angle* **where**
poincare-in-angle p a b c ⟷
b ≠ *a* ∧ *b* ≠ *c* ∧ *p* ≠ *b* ∧ (∃ *x* ∈ *unit-disc*. *poincare-between a x c* ∧ *x* ≠ *a* ∧ *x* ≠ *c* ∧ *poincare-on-ray p b x*)

definition *poincare-ray-meets-line* **where**
poincare-ray-meets-line a b c d ⟷ (∃ *x* ∈ *unit-disc*. *poincare-on-ray x a b* ∧ *poincare-on-line x c d*)

All points on ray are collinear

lemma *poincare-on-ray-poincare-collinear*:
assumes *p* ∈ *unit-disc* **and** *a* ∈ *unit-disc* **and** *b* ∈ *unit-disc* **and** *poincare-on-ray p a b*
shows *poincare-collinear* {*p*, *a*, *b*}
using *assms poincare-between-poincare-collinear*
unfolding *poincare-on-ray-def*
by (*metis insert-commute*)

H-isometries preserve all defined auxiliary relations

lemma *unit-disc-fix-preserves-poincare-on-line* [*simp*]:
assumes *unit-disc-fix M* **and** *p* ∈ *unit-disc a* ∈ *unit-disc b* ∈ *unit-disc*
shows *poincare-on-line* (*moebius-pt M p*) (*moebius-pt M a*) (*moebius-pt M b*) ⟷ *poincare-on-line p a b*
using *assms*
unfolding *poincare-on-line-def*
by *auto*

lemma *unit-disc-fix-preserves-poincare-on-ray* [*simp*]:
  assumes *unit-disc-fix M p* ∈ *unit-disc a* ∈ *unit-disc b* ∈ *unit-disc*
  shows *poincare-on-ray* (*moebius-pt M p*) (*moebius-pt M a*) (*moebius-pt M b*) ⟷ *poincare-on-ray p a b*
  using *assms*
  unfolding *poincare-on-ray-def*
  by *auto*

lemma *unit-disc-fix-preserves-poincare-in-angle* [*simp*]:
  assumes *unit-disc-fix M p* ∈ *unit-disc a* ∈ *unit-disc b* ∈ *unit-disc c* ∈ *unit-disc*
  shows *poincare-in-angle* (*moebius-pt M p*) (*moebius-pt M a*) (*moebius-pt M b*) (*moebius-pt M c*) ⟷ *poincare-in-angle p a b c* (**is** *?lhs* ⟷ *?rhs*)
proof
  assume *?lhs*
  then obtain *Mx* **where** ∗: *Mx* ∈ *unit-disc*
    poincare-between (*moebius-pt M a*) *Mx* (*moebius-pt M c*)
    Mx ≠ *moebius-pt M a Mx* ≠ *moebius-pt M c* *poincare-on-ray* (*moebius-pt M p*) (*moebius-pt M b*) *Mx*
    moebius-pt M b ≠ *moebius-pt M a moebius-pt M b* ≠ *moebius-pt M c moebius-pt M p* ≠ *moebius-pt M b*
   unfolding *poincare-in-angle-def*
   by *auto*
  obtain *x* **where** *Mx* = *moebius-pt M x x* ∈ *unit-disc*
   by (*metis* ∗(*1*) *assms*(*1*) *image-iff unit-disc-fix-iff*)
  thus *?rhs*
   using ∗ *assms*
   unfolding *poincare-in-angle-def*
   by *auto*
next
  assume *?rhs*
  then obtain *x* **where** ∗: *x* ∈ *unit-disc*
    poincare-between a x c
    x ≠ *a x* ≠ *c* *poincare-on-ray p b x*
    b ≠ *a b* ≠ *c p* ≠ *b*
   unfolding *poincare-in-angle-def*
   by *auto*
  thus *?lhs*
   using *assms*
   unfolding *poincare-in-angle-def*
   by *auto* (*rule-tac x*=*moebius-pt M x* **in** *bexI*, *auto*)
qed

lemma *unit-disc-fix-preserves-poincare-ray-meets-line* [*simp*]:
  assumes *unit-disc-fix M a* ∈ *unit-disc b* ∈ *unit-disc c* ∈ *unit-disc d* ∈ *unit-disc*
  shows *poincare-ray-meets-line* (*moebius-pt M a*) (*moebius-pt M b*) (*moebius-pt M c*) (*moebius-pt M d*) ⟷ *poincare-ray-meets-line a b c d* (**is** *?lhs* ⟷ *?rhs*)
proof
  assume *?lhs*
  then obtain *Mx* **where** ∗: *Mx* ∈ *unit-disc poincare-on-ray Mx* (*moebius-pt M a*) (*moebius-pt M b*)
   poincare-on-line Mx (*moebius-pt M c*) (*moebius-pt M d*)
   unfolding *poincare-ray-meets-line-def*
   by *auto*
  obtain *x* **where** *Mx* = *moebius-pt M x x* ∈ *unit-disc*
   by (*metis* ∗(*1*) *assms*(*1*) *image-iff unit-disc-fix-iff*)
  thus *?rhs*
   using *assms* ∗
   unfolding *poincare-ray-meets-line-def poincare-on-line-def*
   by *auto*
next
  assume *?rhs*
  then obtain *x* **where** ∗: *x* ∈ *unit-disc poincare-on-ray x a b*
   poincare-on-line x c d
   unfolding *poincare-ray-meets-line-def*
   by *auto*
  thus *?lhs*
   using *assms* ∗
   unfolding *poincare-ray-meets-line-def poincare-on-line-def*
   by *auto* (*rule-tac x*=*moebius-pt M x* **in** *bexI*, *auto*)

qed

H-lines that intersect on the absolute do not meet (they do not share a common h-point)

lemma *tangent-not-meet*:
  assumes $x1 \in$ *unit-disc* **and** $x2 \in$ *unit-disc* **and** $x1 \neq x2$ **and** $\neg$ *poincare-collinear* $\{0_h, x1, x2\}$
  assumes $i \in$ *ideal-points* (*poincare-line x1 x2*) $a \in$ *unit-disc* $a \neq 0_h$ *poincare-collinear* $\{0_h, a, i\}$
  shows $\neg$ *poincare-ray-meets-line* 0_h *a x1 x2*
proof (*rule ccontr*)
  assume $\neg$ *?thesis*
  then obtain *x* **where** $x \in$ *unit-disc poincare-on-ray x* 0_h *a poincare-collinear* $\{x, x1, x2\}$
    unfolding *poincare-ray-meets-line-def poincare-on-line-def*
    by *auto*

  have *poincare-collinear* $\{0_h, a, x\}$
    using ‹*poincare-on-ray x* 0_h *a*› ‹$x \in$ *unit-disc*› ‹$a \in$ *unit-disc*›
   by (*meson poincare-between-poincare-collinear poincare-between-rev poincare-on-ray-def poincare-on-ray-poincare-collinear zero-in-unit-disc*)

  have $x \neq 0_h$
    using ‹$\neg$ *poincare-collinear* $\{0_h, x1, x2\}$› ‹*poincare-collinear* $\{x, x1, x2\}$›
    unfolding *poincare-collinear-def*
    by (*auto simp add*: *assms*(*2*) *assms*(*3*) *poincare-between-rev*)

  let *?l1* = *poincare-line* 0_h *a*
  let *?l2* = *poincare-line x1 x2*

  have $i \in$ *circline-set unit-circle*
    using ‹$i \in$ *ideal-points* (*poincare-line x1 x2*)›
    using *assms*(*3*) *ideal-points-on-unit-circle is-poincare-line-poincare-line* **by** *blast*

  have $i \in$ *circline-set ?l1*
    using ‹*poincare-collinear* $\{0_h, a, i\}$›
    unfolding *poincare-collinear-def*
    using ‹$a \in$ *unit-disc*› ‹$a \neq 0_h$›
    by (*metis insert-subset unique-poincare-line zero-in-unit-disc*)

  moreover

  have $x \in$ *circline-set ?l1*
    using ‹$a \in$ *unit-disc*› ‹$a \neq 0_h$› ‹*poincare-collinear* $\{0_h, a, x\}$› ‹$x \in$ *unit-disc*›
   by (*metis poincare-collinear3-between poincare-between-poincare-line-uvz poincare-between-poincare-line-uzv poincare-line-sym zero-in-unit-disc*)

  moreover

  have *inversion* $x \in$ *circline-set ?l1*
    using ‹*poincare-collinear* $\{0_h, a, x\}$›
    using *poincare-line-inversion-full*[*of* 0_h *a x*] ‹$a \in$ *unit-disc*› ‹$a \neq 0_h$› ‹$x \in$ *unit-disc*›
   by (*metis poincare-collinear3-between is-poincare-line-inverse-point is-poincare-line-poincare-line poincare-between-poincare-line-uvz poincare-between-poincare-line-uzv poincare-line-sym zero-in-unit-disc*)

  moreover

  have $x \in$ *circline-set ?l2*
    using ‹*poincare-collinear* $\{x, x1, x2\}$› ‹$x1 \neq x2$› ‹$x1 \in$ *unit-disc*› ‹$x2 \in$ *unit-disc*› ‹$x \in$ *unit-disc*›
   by (*metis insert-commute inversion-noteq-unit-disc poincare-between-poincare-line-uvz poincare-between-poincare-line-uzv poincare-collinear3-iff poincare-line-sym-general*)

  moreover

  hence *inversion* $x \in$ *circline-set ?l2*
    using ‹$x1 \neq x2$› ‹$x1 \in$ *unit-disc*› ‹$x2 \in$ *unit-disc*› ‹$x \in$ *unit-disc*›
    using *poincare-line-inversion-full*[*of x1 x2 x*]
    unfolding *circline-set-def*
    by *auto*

moreover

have $i \in$ *circline-set ?l2*
  using ‹$x1 \neq x2$› ‹$x1 \in$ *unit-disc*› ‹$x2 \in$ *unit-disc*›
  using ‹$i \in$ *ideal-points ?l2*›
  by (*simp add*: *ideal-points-on-circline*)

moreover

have $x \neq$ *inversion x*
  using ‹$x \in$ *unit-disc*›
  using *inversion-noteq-unit-disc* **by** *fastforce*

moreover

have $x \neq i$
  using ‹$x \in$ *unit-disc*›
  using ‹$i \in$ *circline-set unit-circle*› *circline-set-def inversion-noteq-unit-disc*
  by *fastforce+*

moreover

have *inversion* $x \neq i$
  using ‹$i \in$ *circline-set unit-circle*› ‹$x \neq i$› *circline-set-def inversion-unit-circle*
  by *fastforce*

ultimately

have *?l1* = *?l2*
  using *unique-circline-set*[*of x inversion x i*]
  by *blast*

hence $0_h \in$ *circline-set ?l2*
  by (*metis* ‹$a \neq 0_h$› *poincare-line-circline-set*(*1*))

thus *False*
  using ‹¬ *poincare-collinear* $\{0_h, x1, x2\}$›
  unfolding *poincare-collinear-def*
  using ‹*poincare-collinear* $\{x, x1, x2\}$› ‹$x1 \neq x2$› ‹$x1 \in$ *unit-disc*› ‹$x2 \in$ *unit-disc*› *poincare-collinear-def unique-poincare-line*
  by *auto*
qed

lemma *limiting-parallels*:
  assumes $a \in$ *unit-disc* **and** $x1 \in$ *unit-disc* **and** $x2 \in$ *unit-disc* **and** ¬ *poincare-on-line a x1 x2*
  shows ∃ *a1*∈*unit-disc*. ∃ *a2*∈*unit-disc*.
      ¬ *poincare-on-line a a1 a2* ∧
      ¬ *poincare-ray-meets-line a a1 x1 x2* ∧ ¬ *poincare-ray-meets-line a a2 x1 x2* ∧
      (∀ *a'*∈*unit-disc*. *poincare-in-angle a' a1 a a2* ⟶ *poincare-ray-meets-line a a' x1 x2*) (**is** *?P a x1 x2*)
proof−
  have ¬ *poincare-collinear* $\{a, x1, x2\}$
    using ‹¬ *poincare-on-line a x1 x2*›
    unfolding *poincare-on-line-def*
    by *simp*

  have ∀ *x1 x2*. $x1 \in$ *unit-disc* ∧ $x2 \in$ *unit-disc* ∧ ¬ *poincare-collinear* $\{a, x1, x2\}$ ⟶ *?P a x1 x2* (**is** *?Q a*)
  proof (*rule wlog-zero*[*OF* ‹$a \in$ *unit-disc*›])
    fix *a u*
    assume ∗: $u \in$ *unit-disc cmod* $a < 1$
    hence *uf*: *unit-disc-fix* (*blaschke a*)
      by *simp*
    assume ∗∗: *?Q* (*moebius-pt* (*blaschke a*) *u*)
    show *?Q u*
    proof *safe*
      fix *x1 x2*
      let *?M* = *moebius-pt* (*blaschke a*)
      assume *xx*: $x1 \in$ *unit-disc* $x2 \in$ *unit-disc* ¬ *poincare-collinear* $\{u, x1, x2\}$

hence *MM*: *?M x1* ∈ *unit-disc* ∧ *?M x2*∈ *unit-disc* ∧ ¬ *poincare-collinear* {*?M u*, *?M x1*, *?M x2*}
  using ∗
  by *auto*
show *?P u x1 x2* (**is** ∃ *a1*∈*unit-disc*. ∃ *a2*∈*unit-disc*. *?P′ a1 a2 u x1 x2*)
proof−
  obtain *Ma1 Ma2* **where** *MM*: *Ma1* ∈ *unit-disc Ma2* ∈ *unit-disc ?P′ Ma1 Ma2* (*?M u*) (*?M x1*) (*?M x2*)
    using ∗∗[*rule-format*, *OF MM*]
    by *blast*
  hence *MM′*: ∀ *a′*∈*unit-disc*. *poincare-in-angle a′ Ma1* (*?M u*) *Ma2* ⟶ *poincare-ray-meets-line* (*?M u*) *a′* (*?M x1*) (*?M x2*)
    by *auto*
  obtain *a1 a2* **where** *a*: *a1* ∈ *unit-disc a2* ∈ *unit-disc ?M a1* = *Ma1 ?M a2* = *Ma2*
    using *uf*
    by (*metis* ‹*Ma1* ∈ *unit-disc*› ‹*Ma2* ∈ *unit-disc*› *image-iff unit-disc-fix-iff*)

  have ∀ *a′*∈*unit-disc*. *poincare-in-angle a′ a1 u a2* ⟶ *poincare-ray-meets-line u a′ x1 x2*
  proof *safe*
    fix *a′*
    assume *a′* ∈ *unit-disc poincare-in-angle a′ a1 u a2*
    thus *poincare-ray-meets-line u a′ x1 x2*
      using *MM*(*1*−*2*) *MM′*[*rule-format*, *of ?M a′*] ∗ *uf a xx*
    by (*meson unit-disc-fix-discI unit-disc-fix-preserves-poincare-in-angle unit-disc-fix-preserves-poincare-ray-meets-line*)
  qed

  hence *?P′ a1 a2 u x1 x2*
    using *MM* ∗ *uf xx a*
    by *auto*

  thus *?thesis*
    using ‹*a1* ∈ *unit-disc*› ‹*a2* ∈ *unit-disc*›
    by *blast*
qed
qed
next
show *?Q* 0_h
proof *safe*
  fix *x1 x2*
  assume *x1* ∈ *unit-disc x2* ∈ *unit-disc*
  assume ¬ *poincare-collinear* {0_h, *x1*, *x2*}
  show *?P* 0_h *x1 x2*
  proof−
    let *?lx* = *poincare-line x1 x2*

    have *x1* ≠ *x2*
      using ‹*x1* ∈ *unit-disc*› ‹*x2* ∈ *unit-disc*›‹¬ *poincare-collinear* {0_h, *x1*, *x2*}›
      using *poincare-collinear3-between*
      by *auto*

    have *lx*: *is-poincare-line ?lx*
      using *is-poincare-line-poincare-line*[*OF* ‹*x1* ≠ *x2*›]
      by *simp*

    obtain *i1 i2* **where** *ideal-points ?lx* = {*i1*, *i2*}
      by (*meson* ‹*x1* ≠ *x2*› *is-poincare-line-poincare-line obtain-ideal-points*)

    let *?li* = *poincare-line i1 i2*
    let *?i1* = *to-complex i1*
    let *?i2* = *to-complex i2*

    have *i1* ∈ *unit-circle-set i2* ∈ *unit-circle-set*
      using *lx* ‹*ideal-points ?lx* = {*i1*, *i2*}›
      unfolding *unit-circle-set-def*
      by (*metis ideal-points-on-unit-circle insertI1*, *metis ideal-points-on-unit-circle insertI1 insertI2*)

    have *i1* ≠ *i2*
      using ‹*ideal-points ?lx* = {*i1*, *i2*}› ‹*x1* ∈ *unit-disc*› ‹*x1* ≠ *x2*› ‹*x2* ∈ *unit-disc*› *ideal-points-different*(*1*)

by *blast*

let *?a1* = *of-complex* (*?i1* / *2*)
let *?a2* = *of-complex* (*?i2* / *2*)
let *?la* = *poincare-line ?a1 ?a2*

have *?a1* ∈ *unit-disc ?a2* ∈ *unit-disc*
using ‹*i1* ∈ *unit-circle-set*› ‹*i2* ∈ *unit-circle-set*›
unfolding *unit-circle-set-def unit-disc-def disc-def circline-set-def*
by *auto* (*transfer*, *transfer*, *case-tac i1*, *case-tac i2*, *simp add*: *vec-cnj-def*)+

have *?a1* ≠ 0_h *?a2* ≠ 0_h
using ‹*i1* ∈ *unit-circle-set*› ‹*i2* ∈ *unit-circle-set*›
unfolding *unit-circle-set-def*
by *auto*

have *?a1* ≠ *?a2*
using ‹*i1* ≠ *i2*›
by (*metis* ‹*i1* ∈ *unit-circle-set*› ‹*i2* ∈ *unit-circle-set*› *circline-set-def divide-cancel-right inversion-infty inversion-unit-circle mem-Collect-eq of-complex-to-complex of-complex-zero to-complex-of-complex unit-circle-set-def zero-neq-numeral*)

have *poincare-collinear* {0_h, *?a1*, *i1*}
unfolding *poincare-collinear-def*
using ‹*?a1* ≠ 0_h›[*symmetric*] *is-poincare-line-poincare-line*[*of* 0_h *?a1*]
unfolding *circline-set-def*
apply (*rule-tac x*=*poincare-line* 0_h *?a1* **in** *exI*, *auto*)
apply (*transfer*, *transfer*, *auto simp add*: *vec-cnj-def*)
done

have *poincare-collinear* {0_h, *?a2*, *i2*}
unfolding *poincare-collinear-def*
using ‹*?a2* ≠ 0_h›[*symmetric*] *is-poincare-line-poincare-line*[*of* 0_h *?a2*]
unfolding *circline-set-def*
apply (*rule-tac x*=*poincare-line* 0_h *?a2* **in** *exI*, *auto*)
apply (*transfer*, *transfer*, *auto simp add*: *vec-cnj-def*)
done

have ¬ *poincare-ray-meets-line* 0_h *?a1 x1 x2*
using *tangent-not-meet*[*of x1 x2 i1 ?a1*]
using ‹*x1* ∈ *unit-disc*› ‹*x2* ∈ *unit-disc*› ‹*?a1* ∈ *unit-disc*› ‹*x1* ≠ *x2*› ‹¬ *poincare-collinear* {0_h, *x1*, *x2*}›
using ‹*ideal-points ?lx* = {*i1*, *i2*}› ‹*?a1* ≠ 0_h› ‹*poincare-collinear* {0_h, *?a1*, *i1*}›
by *simp*

moreover

have ¬ *poincare-ray-meets-line* 0_h *?a2 x1 x2*
using *tangent-not-meet*[*of x1 x2 i2 ?a2*]
using ‹*x1* ∈ *unit-disc*› ‹*x2* ∈ *unit-disc*› ‹*?a2* ∈ *unit-disc*› ‹*x1* ≠ *x2*› ‹¬ *poincare-collinear* {0_h, *x1*, *x2*}›
using ‹*ideal-points ?lx* = {*i1*, *i2*}› ‹*?a2* ≠ 0_h› ‹*poincare-collinear* {0_h, *?a2*, *i2*}›
by *simp*

moreover

have ∀ *a'* ∈ *unit-disc*. *poincare-in-angle a' ?a1* 0_h *?a2* ⟶ *poincare-ray-meets-line* 0_h *a' x1 x2*
unfolding *poincare-in-angle-def*
proof *safe*
fix *a' a*
assume ∗: *a'* ∈ *unit-disc a* ∈ *unit-disc poincare-on-ray a'* 0_h *a a'* ≠ 0_h
poincare-between ?a1 a ?a2 a ≠ *?a1 a* ≠ *?a2*
show *poincare-ray-meets-line* 0_h *a' x1 x2*
proof−
have ∀ *a' a1 a2 x1 x2 i1 i2*.
a' ∈ *unit-disc* ∧ *x1* ∈ *unit-disc* ∧ *x2* ∈ *unit-disc* ∧ *x1* ≠ *x2* ∧
¬ *poincare-collinear* {0_h, *x1*, *x2*} ∧ *ideal-points* (*poincare-line x1 x2*) = {*i1*, *i2*} ∧
a1 = *of-complex* (*to-complex i1* / *2*) ∧ *a2* = *of-complex* (*to-complex i2* / *2*) ∧
i1 ≠ *i2* ∧ *a1* ≠ *a2* ∧ *poincare-collinear* {0_h, *a1*, *i1*} ∧ *poincare-collinear* {0_h, *a2*, *i2*} ∧

$a1 \in unit\text{-}disc \wedge a2 \in unit\text{-}disc \wedge i1 \in unit\text{-}circle\text{-}set \wedge i2 \in unit\text{-}circle\text{-}set \wedge$
$poincare\text{-}on\text{-}ray\ a'\ 0_h\ a \wedge a' \neq 0_h \wedge poincare\text{-}between\ a1\ a\ a2 \wedge a \neq a1 \wedge a \neq a2 \longrightarrow$
$poincare\text{-}ray\text{-}meets\text{-}line\ 0_h\ a'\ x1\ x2$ (**is** $\forall\ a'\ a1\ a2\ x1\ x2\ i1\ i2.\ ?R\ 0_h\ a'\ a1\ a2\ x1\ x2\ i1\ i2\ a$)
proof (*rule wlog-rotation-to-positive-x-axis*[*OF* ‹$a \in unit\text{-}disc$›])
let $?R' = \lambda\ a\ zero.\ \forall\ a'\ a1\ a2\ x1\ x2\ i1\ i2.\ ?R\ zero\ a'\ a1\ a2\ x1\ x2\ i1\ i2\ a$
fix *xa*
assume *xa*: $is\text{-}real\ xa\ 0 < Re\ xa\ Re\ xa < 1$
let $?a = of\text{-}complex\ xa$
show $?R'\ ?a\ 0_h$
proof *safe*
fix $a'\ a1\ a2\ x1\ x2\ i1\ i2$
let $?i1 = to\text{-}complex\ i1$ **and** $?i2 = to\text{-}complex\ i2$
let $?a1 = of\text{-}complex\ (?i1\ /\ 2)$ **and** $?a2 = of\text{-}complex\ (?i2\ /\ 2)$
let $?la = poincare\text{-}line\ ?a1\ ?a2$ **and** $?lx = poincare\text{-}line\ x1\ x2$ **and** $?li = poincare\text{-}line\ i1\ i2$
assume $a' \in unit\text{-}disc\ x1 \in unit\text{-}disc\ x2 \in unit\text{-}disc\ x1 \neq x2$
assume $\neg\ poincare\text{-}collinear\ \{0_h,\ x1,\ x2\}\ ideal\text{-}points\ ?lx = \{i1,\ i2\}$
assume $poincare\text{-}on\text{-}ray\ a'\ 0_h\ ?a\ a' \neq 0_h$
assume $poincare\text{-}between\ ?a1\ ?a\ ?a2\ \ ?a \neq ?a1\ ?a \neq ?a2$
assume $i1 \neq i2\ ?a1 \neq ?a2\ poincare\text{-}collinear\ \{0_h,\ ?a1,\ i1\}\ \ poincare\text{-}collinear\ \{0_h,\ ?a2,\ i2\}$
assume $?a1 \in unit\text{-}disc\ \ ?a2 \in unit\text{-}disc$
assume $i1 \in unit\text{-}circle\text{-}set\ i2 \in unit\text{-}circle\text{-}set$
show $poincare\text{-}ray\text{-}meets\text{-}line\ 0_h\ a'\ x1\ x2$
proof−
have $?lx = ?li$
using ‹$ideal\text{-}points\ ?lx = \{i1,\ i2\}$› ‹$x1 \neq x2$› *ideal-points-line-unique*
by *auto*

have *lx*: *is-poincare-line ?lx*
using *is-poincare-line-poincare-line*[*OF* ‹$x1 \neq x2$›]
by *simp*

have $x1 \in circline\text{-}set\ ?lx\ x2 \in circline\text{-}set\ ?lx$
using *lx* ‹$x1 \neq x2$›
by *auto*

have $?lx \neq x\text{-}axis$
using ‹$\neg\ poincare\text{-}collinear\ \{0_h,\ x1,\ x2\}$› ‹$x1 \in circline\text{-}set\ ?lx$› ‹$x2 \in circline\text{-}set\ ?lx$› *lx*
unfolding *poincare-collinear-def*
by *auto*

have $0_h \notin circline\text{-}set\ ?lx$
using ‹$\neg\ poincare\text{-}collinear\ \{0_h,\ x1,\ x2\}$› *lx* ‹$x1 \in circline\text{-}set\ ?lx$› ‹$x2 \in circline\text{-}set\ ?lx$›
unfolding *poincare-collinear-def*
by *auto*

have $xa \neq 0\ ?a \neq 0_h$
using *xa*
by *auto*
hence $0_h \neq ?a$
by *metis*

have $?a \in positive\text{-}x\text{-}axis$
using *xa*
unfolding *positive-x-axis-def*
by *simp*

have $?a \in unit\text{-}disc$
using *xa*
by (*auto simp add*: *cmod-eq-Re*)

have $?a \in circline\text{-}set\ ?la$
using ‹$poincare\text{-}between\ ?a1\ ?a\ ?a2$›
using ‹$?a1 \neq ?a2$› ‹$?a \in unit\text{-}disc$› ‹$?a1 \in unit\text{-}disc$› ‹$?a2 \in unit\text{-}disc$› *poincare-between-poincare-line-uzv*

by *blast*

```
have ?a1 ∈ circline-set ?la ?a2 ∈ circline-set ?la
  by (auto simp add: ‹?a1 ≠ ?a2›)

have la: is-poincare-line ?la
  using is-poincare-line-poincare-line[OF ‹?a1 ≠ ?a2›]
  by  simp

have inv: inversion i1 = i1 inversion i2 = i2
  using ‹i1 ∈ unit-circle-set› ‹i2 ∈ unit-circle-set›
  by (auto simp add: circline-set-def unit-circle-set-def)

have  i1 ≠ ∞h i2 ≠ ∞h
  using inv
  by auto

have ?a1 ∉ circline-set x-axis ∧ ?a2 ∉ circline-set x-axis
proof (rule ccontr)
  assume ¬ ?thesis
  hence ?a1 ∈ circline-set x-axis ∨ ?a2 ∈ circline-set x-axis
    by auto
  hence ?la = x-axis
  proof
    assume ?a1 ∈ circline-set x-axis
    hence {?a, ?a1} ⊆ circline-set ?la ∩ circline-set x-axis
      using ‹?a ∈ circline-set ?la› ‹?a1 ∈ circline-set ?la›‹?a ∈ positive-x-axis›
      using circline-set-x-axis-I xa(1)
      by blast
    thus ?la = x-axis
      using unique-is-poincare-line[of ?a ?a1 ?la x-axis]
      using ‹?a1 ∈ unit-disc› ‹?a ∈ unit-disc› la ‹?a ≠ ?a1›
      by auto
  next
    assume ?a2 ∈ circline-set x-axis
    hence {?a, ?a2} ⊆ circline-set ?la ∩ circline-set x-axis
      using ‹?a ∈ circline-set ?la› ‹?a2 ∈ circline-set ?la› ‹?a ∈ positive-x-axis›
      using circline-set-x-axis-I xa(1)
      by blast
    thus ?la = x-axis
      using unique-is-poincare-line[of ?a ?a2 ?la x-axis]
      using ‹?a2 ∈ unit-disc› ‹?a ∈ unit-disc› la ‹?a ≠ ?a2›
      by auto
  qed

  hence i1 ∈ circline-set x-axis ∧ i2 ∈ circline-set x-axis
    using ‹?a1 ∈ circline-set ?la› ‹?a2 ∈ circline-set ?la›
   by (metis ‹i1 ≠ ∞h› ‹i2 ≠ ∞h› ‹of-complex (to-complex i1 / 2) ∈ unit-disc› ‹of-complex (to-complex
i2 / 2) ∈ unit-disc› ‹poincare-collinear {0h, of-complex (to-complex i1 / 2), i1}› ‹poincare-collinear {0h, of-complex
(to-complex i2 / 2), i2}› divide-eq-0-iff inf-not-of-complex inv(1) inv(2) inversion-noteq-unit-disc of-complex-to-complex
of-complex-zero-iff poincare-collinear3-poincare-lines-equal-general poincare-line-0-real-is-x-axis poincare-line-circline-set(2)
zero-in-unit-disc zero-neq-numeral)

  thus False
    using ‹?lx ≠ x-axis› unique-is-poincare-line-general[of i1 i2 ?li x-axis] ‹i1 ≠ i2› inv ‹?lx = ?li›
    by auto
qed

hence ?la ≠ x-axis
  using ‹?a1 ≠ ?a2› poincare-line-circline-set(1)
  by fastforce

have intersects-x-axis-positive ?la
  using intersects-x-axis-positive-iff[of ?la] ‹?la ≠ x-axis› ‹?a ∈ circline-set ?la› la
  using ‹?a ∈ unit-disc› ‹?a ∈ positive-x-axis›
  by auto

have intersects-x-axis ?lx
```

proof−
have *Arg* (*to-complex ?a1*) ∗ *Arg* (*to-complex ?a2*) < *0*
using ‹*poincare-between ?a1 ?a ?a2*› ‹*?a1* ∈ *unit-disc*› ‹*?a2* ∈ *unit-disc*›
using *poincare-between-x-axis-intersection*[*of ?a1 ?a2 of-complex xa*]
using ‹*?a1* ≠ *?a2*› ‹*?a* ∈ *unit-disc*› ‹*?a1* ∉ *circline-set x-axis* ∧ *?a2* ∉ *circline-set x-axis*› ‹*?a* ∈ *positive-x-axis*›
using ‹*?a* ∈ *circline-set ?la*›
unfolding *positive-x-axis-def*
by *simp*

moreover

have ⋀ *x y x′ y′* :: *real*. ⟦*sgn x′* = *sgn x*; *sgn y′* = *sgn y*⟧ ⟹ *x*∗*y* < *0* ⟷ *x′*∗*y′* < *0*
by (*metis sgn-less sgn-mult*)

ultimately

have *Im* (*to-complex ?a1*) ∗ *Im* (*to-complex ?a2*) < *0*
using *arg-Im-sgn*[*of to-complex ?a1*] *arg-Im-sgn*[*of to-complex ?a2*]
using ‹*?a1* ∈ *unit-disc*› ‹*?a2* ∈ *unit-disc*› ‹*?a1* ∉ *circline-set x-axis* ∧ *?a2* ∉ *circline-set x-axis*›
using *inf-or-of-complex*[*of ?a1*] *inf-or-of-complex*[*of ?a2*] *circline-set-x-axis*
by (*metis circline-set-x-axis-I to-complex-of-complex*)

thus *?thesis*
using *ideal-points-intersects-x-axis*[*of ?lx i1 i2*]
using ‹*ideal-points ?lx* = {*i1*, *i2*}› *lx* ‹*?lx* ≠ *x-axis*›
by *simp*
qed

have *intersects-x-axis-positive ?lx*
proof−
have *cmod ?i1* = *1 cmod ?i2* = *1*
using ‹*i1* ∈ *unit-circle-set*› ‹*i2* ∈ *unit-circle-set*›
unfolding *unit-circle-set-def*
by *auto*

let *?a1′* = *?i1* / *2* **and** *?a2′* = *?i2* / *2*
let *?Aa1* = i ∗ (*?a1′* ∗ *cnj ?a2′* − *?a2′* ∗ *cnj ?a1′*) **and**
?Ba1 = i ∗ (*?a2′* ∗ *cor* ((*cmod ?a1′*)^2 + *1*) − *?a1′* ∗ *cor* ((*cmod ?a2′*)^2 + *1*))

have *?Aa1* ≠ *0* ∨ *?Ba1* ≠ *0*
using ‹*cmod* (*to-complex i1*) = *1*› ‹*cmod* (*to-complex i2*) = *1*› ‹*?a1* ≠ *?a2*›
by (*auto simp add*: *power-divide complex-mult-cnj-cmod*)

have *is-real ?Aa1*
by *simp*

have *?a1* ≠ *inversion ?a2*
using ‹*?a1* ∈ *unit-disc*› ‹*?a2* ∈ *unit-disc*› *inversion-noteq-unit-disc* **by** *fastforce*

hence *Re ?Ba1* / *Re ?Aa1* < −*1*
using ‹*intersects-x-axis-positive ?la*› ‹*?a1* ≠ *?a2*›
using *intersects-x-axis-positive-mk-circline*[*of ?Aa1 ?Ba1*] ‹*?Aa1* ≠ *0* ∨ *?Ba1* ≠ *0*› ‹*is-real ?Aa1*›
using *poincare-line-non-homogenous*[*of ?a1 ?a2*]
by (*simp add*: *Let-def*)

moreover

let *?i1′* = *to-complex i1* **and** *?i2′* = *to-complex i2*
let *?Ai1* = i ∗ (*?i1′* ∗ *cnj ?i2′* − *?i2′* ∗ *cnj ?i1′*) **and**
?Bi1 = i ∗ (*?i2′* ∗ *cor* ((*cmod ?i1′*)^2 + *1*) − *?i1′* ∗ *cor* ((*cmod ?i2′*)^2 + *1*))

have *?Ai1* ≠ *0* ∨ *?Bi1* ≠ *0*
using ‹*cmod* (*to-complex i1*) = *1*› ‹*cmod* (*to-complex i2*) = *1*› ‹*?a1* ≠ *?a2*›
by (*auto simp add*: *power-divide complex-mult-cnj-cmod*)

have *is-real ?Ai1*
  by *simp*

have *sgn* (*Re ?Bi1* / *Re ?Ai1*) = *sgn* (*Re ?Ba1* / *Re ?Aa1*)
proof−
  have *Re ?Bi1* / *Re ?Ai1* = (*Im ?i1* ∗ *2* − *Im ?i2* ∗ *2*) /
                (*Im ?i2* ∗ (*Re ?i1* ∗ *2*) − *Im ?i1* ∗ (*Re ?i2* ∗ *2*))
    using ‹*cmod ?i1* = *1*› ‹*cmod ?i2* = *1*›
    by (*auto simp add*: *complex-mult-cnj-cmod field-simps*)
  also have ... = (*Im ?i1* − *Im ?i2*) /
              (*Im ?i2* ∗ (*Re ?i1*) − *Im ?i1* ∗ (*Re ?i2*)) (**is** ... = *?expr*)
    apply (*subst left-diff-distrib*[*symmetric*])
    apply (*subst semiring-normalization-rules*(*18*))+
    apply (*subst left-diff-distrib*[*symmetric*])
    by (*metis mult.commute mult-divide-mult-cancel-left-if zero-neq-numeral*)
  finally have *1*: *Re ?Bi1* / *Re ?Ai1* = (*Im ?i1* − *Im ?i2*) / (*Im ?i2* ∗ (*Re ?i1*) − *Im ?i1* ∗ (*Re ?i2*))
    .

  have *Re ?Ba1* / *Re ?Aa1* = (*Im ?i1* ∗ *20* − *Im ?i2* ∗ *20*) /
                (*Im ?i2* ∗ (*Re ?i1* ∗ *16*) − *Im ?i1* ∗ (*Re ?i2* ∗ *16*))
    using ‹*cmod* (*to-complex i1*) = *1*› ‹*cmod* (*to-complex i2*) = *1*›
    by (*auto simp add*: *complex-mult-cnj-cmod field-simps*)
  also have ... = (*20* / *16*) ∗ ((*Im ?i1* − *Im ?i2*) /
              (*Im ?i2* ∗ (*Re ?i1*) − *Im ?i1* ∗ (*Re ?i2*)))
    apply (*subst left-diff-distrib*[*symmetric*])+
    apply (*subst semiring-normalization-rules*(*18*))+
    apply (*subst left-diff-distrib*[*symmetric*])+
  by (*metis* (*no-types*, *opaque-lifting*) *field-class.field-divide-inverse mult.commute times-divide-times-eq*)
  finally have *2*: *Re ?Ba1* / *Re ?Aa1* = (*5* / *4*) ∗ ((*Im ?i1* − *Im ?i2*) / (*Im ?i2* ∗ (*Re ?i1*) − *Im ?i1* ∗ (*Re ?i2*)))
    by *simp*

  have *?expr* ≠ *0*
    using ‹*Re ?Ba1* / *Re ?Aa1* < −*1*›
    apply (*subst* (*asm*) *2*)
    by *linarith*
  thus *?thesis*
    apply (*subst 1*, *subst 2*)
    apply (*simp only*: *sgn-mult*)
    by *simp*
qed

moreover

have *i1* ≠ *inversion i2*
  by (*simp add*: ‹*i1* ≠ *i2*› *inv*(*2*))

have (*Re ?Bi1* / *Re ?Ai1*)^2 > *1*
proof−
  have *?Ai1* = *0* ∨ (*Re ?Bi1*)^2 > (*Re ?Ai1*)^2
    using ‹*intersects-x-axis ?lx*›
    using ‹*i1* ≠ *i2*› ‹*i1* ≠ ∞_h› ‹*i2* ≠ ∞_h› ‹*i1* ≠ *inversion i2*›
    using *intersects-x-axis-mk-circline*[*of ?Ai1 ?Bi1*] ‹*?Ai1* ≠ *0* ∨ *?Bi1* ≠ *0*› ‹*is-real ?Ai1*›
    using *poincare-line-non-homogenous*[*of i1 i2*] ‹*?lx* = *?li*›
    by *metis*

  moreover
  have *?Ai1* ≠ *0*
  proof (*rule ccontr*)
    assume ¬ *?thesis*
    hence 0_h ∈ *circline-set ?li*
      unfolding *circline-set-def*
      apply *simp*
      apply (*transfer*, *transfer*, *case-tac i1*, *case-tac i2*)

```
        by (auto simp add: vec-cnj-def field-simps)
      thus False
        using ‹0h ∉ circline-set ?lx› ‹?lx = ?li›
        by simp
    qed

    ultimately

    have (Re ?Bi1)^2 > (Re ?Ai1)^2
      by auto

    moreover

    have Re ?Ai1 ≠ 0
      using ‹is-real ?Ai1› ‹?Ai1 ≠ 0›
      by (simp add: complex-eq-iff)

    ultimately

    show ?thesis
      by (simp add: power-divide)
  qed

  moreover

  {
    fix x1 x2 :: real
    assume sgn x1 = sgn x2 x1 < −1 x2^2 > 1
    hence x2 < −1
      by (smt one-power2 real-sqrt-abs real-sqrt-less-iff sgn-neg sgn-pos)
  }

  ultimately

  have Re ?Bi1 / Re ?Ai1 < −1
    by metis

  thus ?thesis
    using ‹i1 ≠ i2› ‹i1 ≠ ∞h› ‹i2 ≠ ∞h› ‹i1 ≠ inversion i2›
    using intersects-x-axis-positive-mk-circline[of ?Ai1 ?Bi1] ‹?Ai1 ≠ 0 ∨ ?Bi1 ≠ 0› ‹is-real ?Ai1›
    using poincare-line-non-homogenous[of i1 i2] ‹?lx = ?li›
    by (simp add: Let-def)
qed

then obtain x where x: x ∈ unit-disc x ∈ circline-set ?lx ∩ positive-x-axis
  using intersects-x-axis-positive-iff[OF lx ‹?lx ≠ x-axis›]
  by auto

have poincare-on-ray x 0h a' ∧ poincare-collinear {x1, x2, x}
proof
  show poincare-collinear {x1, x2, x}
    using x lx ‹x1 ∈ circline-set ?lx› ‹x2 ∈ circline-set ?lx›
    unfolding poincare-collinear-def
    by auto
next
  show poincare-on-ray x 0h a'
    unfolding poincare-on-ray-def
  proof−
    have a' ∈ circline-set x-axis
      using ‹poincare-on-ray a' 0h ?a› xa ‹0h ≠ ?a› ‹xa ≠ 0› ‹a' ∈ unit-disc›
      unfolding poincare-on-ray-def
      using poincare-line-0-real-is-x-axis[of of-complex xa]
      using poincare-between-poincare-line-uvz[of 0h of-complex xa a']
      using poincare-between-poincare-line-uzv[of 0h of-complex xa a']
      by (auto simp add: cmod-eq-Re)
```

then obtain xa' **where** xa': $a' = of\text{-}complex\ xa'\ is\text{-}real\ xa'$
using ‹$a' \in unit\text{-}disc$›
using *circline-set-def on-circline-x-axis*
by *auto*

hence $-1 < Re\ xa'\ Re\ xa' < 1\ xa' \neq 0$
using ‹$a' \in unit\text{-}disc$› ‹$a' \neq 0_h$›
by (*auto simp add*: *cmod-eq-Re*)

hence $Re\ xa' > 0\ Re\ xa' < 1\ is\text{-}real\ xa'$
using ‹*poincare-on-ray* $a'\ 0_h$ (*of-complex xa*)›
using *poincare-between-x-axis-0uv*[*of Re xa' Re xa*]
using *poincare-between-x-axis-0uv*[*of Re xa Re xa'*]
using *circline-set-positive-x-axis-I*[*of Re xa'*]
using *xa xa' complex-of-real-Re*
unfolding *poincare-on-ray-def*
by (*smt of-real-0*, *linarith*, *blast*)

moreover

obtain *xx* **where** $is\text{-}real\ xx\ Re\ xx > 0\ Re\ xx < 1\ x = of\text{-}complex\ xx$
using *x*
unfolding *positive-x-axis-def*
using *circline-set-def cmod-eq-Re on-circline-x-axis*
by *auto*

ultimately

show *poincare-between* $0_h\ x\ a' \vee$ *poincare-between* $0_h\ a'\ x$
using ‹$a' = of\text{-}complex\ xa'$›
by (*smt* ‹$a' \in unit\text{-}disc$› *arg-0-iff poincare-between-0uv poincare-between-def to-complex-of-complex x*(*1*))
qed

qed

thus *?thesis*
using ‹$x \in unit\text{-}disc$›
unfolding *poincare-ray-meets-line-def poincare-on-line-def*
by (*metis insert-commute*)
qed
qed
next
show $a \neq 0_h$
proof (*rule ccontr*)
assume ¬ *?thesis*
then obtain *k* **where** $k<0$ *to-complex ?a1* = *cor k* ∗ *to-complex ?a2*
using *poincare-between-u0v*[*OF* ‹$?a1 \in unit\text{-}disc$› ‹$?a2 \in unit\text{-}disc$› ‹$?a1 \neq 0_h$› ‹$?a2 \neq 0_h$›]
using ‹*poincare-between ?a1 a ?a2*›
by *auto*
hence *to-complex i1* = *cor k* ∗ *to-complex i2* $k < 0$
by *auto*
hence $0_h \in$ *circline-set* (*poincare-line x1 x2*)
using *ideal-points-proportional*[*of poincare-line x1 x2 i1 i2 k*] ‹*ideal-points* (*poincare-line x1 x2*) = {*i1*, *i2*}›
using *is-poincare-line-poincare-line*[*OF* ‹$x1 \neq x2$›]
by *simp*
thus *False*
using ‹¬ *poincare-collinear* {0_h, *x1*, *x2*}›
using *is-poincare-line-poincare-line*[*OF* ‹$x1 \neq x2$›]
unfolding *poincare-collinear-def*
by (*meson* ‹$x1 \neq x2$› *empty-subsetI insert-subset poincare-line-circline-set*(*1*) *poincare-line-circline-set*(*2*))
qed
next
fix φ *u*
let $?R' = \lambda$ *a zero*. ∀ a' *a1 a2 x1 x2 i1 i2*. *?R zero* a' *a1 a2 x1 x2 i1 i2 a*

let *?M* = *moebius-pt* (*moebius-rotation* φ)
assume $*$: $u \in$ *unit-disc* $u \neq 0_h$ **and** $**$: *?R′* (*?M u*) 0_h
have *uf*: *unit-disc-fix* (*moebius-rotation* φ)
by *simp*
have *?M* 0_h = 0_h
by *auto*
hence $**$: *?R′* (*?M u*) (*?M* 0_h)
using $**$
by *simp*
show *?R′ u* 0_h
proof (*rule allI*)+
fix *a′ a1 a2 x1 x2 i1 i2*
have *i1*: *i1* $\in$ *unit-circle-set* $\longrightarrow$ *moebius-pt* (*moebius-rotation* φ) (*of-complex* (*to-complex i1* / *2*)) = *of-complex* (*to-complex* (*moebius-pt* (*moebius-rotation* φ) *i1*) / *2*)
using *unit-circle-set-def* **by** *force*

have *i2*: *i2* $\in$ *unit-circle-set* $\longrightarrow$ *moebius-pt* (*moebius-rotation* φ) (*of-complex* (*to-complex i2* / *2*)) = *of-complex* (*to-complex* (*moebius-pt* (*moebius-rotation* φ) *i2*) / *2*)
using *unit-circle-set-def* **by** *force*

show *?R* 0_h *a′ a1 a2 x1 x2 i1 i2 u*
using $**$[*rule-format*, *of ?M a′ ?M x1 ?M x2 ?M i1 ?M i2 ?M a1 ?M a2*] *uf* $*$
apply (*auto simp del*: *moebius-pt-moebius-rotation-zero moebius-pt-moebius-rotation*)
using *i1 i2*
by *simp*
qed
qed
thus *?thesis*
using ‹*a′* $\in$ *unit-disc*› ‹*x1* $\in$ *unit-disc*› ‹*x2* $\in$ *unit-disc*› ‹*x1* $\neq$ *x2*›
using ‹$\neg$ *poincare-collinear* {0_h, *x1*, *x2*}› ‹*ideal-points ?lx* = {*i1*, *i2*}› ‹*i1* $\neq$ *i2*›
using ‹*?a1* $\neq$ *?a2*› ‹*poincare-collinear* {0_h, *?a1*, *i1*}› ‹*poincare-collinear* {0_h, *?a2*, *i2*}›
using ‹*?a1* $\in$ *unit-disc*› ‹*?a2* $\in$ *unit-disc*› ‹*i1* $\in$ *unit-circle-set*› ‹*i2* $\in$ *unit-circle-set*›
using ‹*poincare-on-ray a′* 0_h *a*› ‹*a′* $\neq$ 0_h› ‹*poincare-between ?a1 a ?a2*› ‹*a* $\neq$ *?a1*› ‹*a* $\neq$ *?a2*›
by *blast*
qed
qed

moreover

have $\neg$ *poincare-on-line* 0_h *?a1 ?a2*
proof
assume $*$: *poincare-on-line* 0_h *?a1 ?a2*
hence *poincare-collinear* {0_h, *?a1*, *?a2*}
unfolding *poincare-on-line-def*
by *simp*
hence *poincare-line* 0_h *?a1* = *poincare-line* 0_h *?a2*
using *poincare-collinear3-poincare-lines-equal-general*[*of* 0_h *?a1 ?a2*]
using ‹*?a1* $\in$ *unit-disc*› ‹*?a1* $\neq$ 0_h› ‹*?a2* $\in$ *unit-disc*› ‹*?a2* $\neq$ 0_h›
by (*metis inversion-noteq-unit-disc zero-in-unit-disc*)

have *i1* $\in$ *circline-set* (*poincare-line* 0_h *?a1*)
using ‹*poincare-collinear* {0_h, *?a1*, *i1*}›
using *poincare-collinear3-poincare-line-general*[*of i1* 0_h *?a1*]
using ‹*?a1* $\in$ *unit-disc*› ‹*?a1* $\neq$ 0_h›
by (*metis insert-commute inversion-noteq-unit-disc zero-in-unit-disc*)
moreover
have *i2* $\in$ *circline-set* (*poincare-line* 0_h *?a1*)
using ‹*poincare-collinear* {0_h, *?a2*, *i2*}›
using *poincare-collinear3-poincare-line-general*[*of i2* 0_h *?a2*]
using ‹*?a2* $\in$ *unit-disc*› ‹*?a2* $\neq$ 0_h› ‹*poincare-line* 0_h *?a1* = *poincare-line* 0_h *?a2*›
by (*metis insert-commute inversion-noteq-unit-disc zero-in-unit-disc*)

ultimately

have *poincare-collinear* {0_h, *i1*, *i2*}
using ‹*?a1* $\in$ *unit-disc*› ‹*?a1* $\neq$ 0_h› ‹*poincare-collinear* {0_h, *?a1*, *i1*}›

by (*smt insert-subset poincare-collinear-def unique-poincare-line zero-in-unit-disc*)
hence 0_h ∈ *circline-set* (*poincare-line i1 i2*)
using *poincare-collinear3-poincare-line-general*[*of* 0_h *i1 i2*]
using ‹*i1* ≠ *i2*› ‹*i2* ∈ *unit-circle-set*› *unit-circle-set-def*
by *force*

moreover

have *?lx* = *?li*
using ‹*ideal-points ?lx* = {*i1*, *i2*}› ‹*x1* ≠ *x2*› *ideal-points-line-unique*
by *auto*

ultimately

show *False*
using ‹¬ *poincare-collinear* {0_h, *x1*, *x2*}›
using ‹*x1* ≠ *x2*› *poincare-line-poincare-collinear3-general*
by *auto*
qed

ultimately

show *?thesis*
using ‹*?a1* ∈ *unit-disc*› ‹*?a2* ∈ *unit-disc*›
by *blast*
qed
qed
qed
thus *?thesis*
using ‹*x1* ∈ *unit-disc*› ‹*x2* ∈ *unit-disc*› ‹¬ *poincare-collinear* {*a*, *x1*, *x2*}›
by *blast*
qed

11.9 Interpretation of locales

global-interpretation *PoincareTarskiAbsolute*: *TarskiAbsolute* **where** *cong* = *p-congruent* **and** *betw* = *p-between*
defines *p-on-line* = *PoincareTarskiAbsolute.on-line* **and**
p-on-ray = *PoincareTarskiAbsolute.on-ray* **and**
p-in-angle = *PoincareTarskiAbsolute.in-angle* **and**
p-ray-meets-line = *PoincareTarskiAbsolute.ray-meets-line*
proof−
show *TarskiAbsolute p-congruent p-between*
proof

1. Reflexivity of congruence

fix *x y*
show *p-congruent x y y x*
unfolding *p-congruent-def*
by *transfer* (*simp add*: *poincare-distance-sym*)
next

2. Transitivity of congruence

fix *x y z u v w*
show *p-congruent x y z u* ∧ *p-congruent x y v w* ⟶ *p-congruent z u v w*
by (*transfer*, *simp*)
next

3. Identity of congruence

fix *x y z*
show *p-congruent x y z z* ⟶ *x* = *y*
unfolding *p-congruent-def*
by *transfer* (*simp add*: *poincare-distance-eq-0-iff*)
next

4. Segment construction

fix *x y a b*
show ∃ *z*. *p-between x y z* ∧ *p-congruent y z a b*
using *segment-construction*
unfolding *p-congruent-def*
by *transfer* (*simp*, *blast*)
next

5. Five segment

fix $x\ y\ z\ x'\ y'\ z'\ u\ u'$
show $x \neq y$ ∧ *p-between* $x\ y\ z$ ∧ *p-between* $x'\ y'\ z'$ ∧
p-congruent $x\ y\ x'\ y'$ ∧ *p-congruent* $y\ z\ y'\ z'$ ∧
p-congruent $x\ u\ x'\ u'$ ∧ *p-congruent* $y\ u\ y'\ u'$ ⟶
p-congruent $z\ u\ z'\ u'$
unfolding *p-congruent-def*
apply *transfer*
using *five-segment-axiom*
by *meson*
next

6. Identity of betweeness

fix *x y*
show *p-between x y x* ⟶ $x = y$
by *transfer* (*simp add*: *poincare-between-sum-distances poincare-distance-eq-0-iff poincare-distance-sym*)
next

7. Pasch

fix *x y z u v*
show *p-between x u z* ∧ *p-between y v z* ⟶ (∃ *a*. *p-between u a y* ∧ *p-between x a v*)
apply *transfer*
using *Pasch*
by *blast*
next

8. Lower dimension

show ∃ *a*. ∃ *b*. ∃ *c*. ¬ *p-between a b c* ∧ ¬ *p-between b c a* ∧ ¬ *p-between c a b*
apply (*transfer*)
using *lower-dimension-axiom*
by *simp*
next

9. Upper dimension

fix *x y z u v*
show *p-congruent x u x v* ∧ *p-congruent y u y v* ∧ *p-congruent z u z v* ∧ $u \neq v$ ⟶
p-between x y z ∨ *p-between y z x* ∨ *p-between z x y*
unfolding *p-congruent-def*
by (*transfer*, *simp add*: *upper-dimension-axiom*)
qed
qed

interpretation *PoincareTarskiHyperbolic*: *TarskiHyperbolic*
where *cong* = *p-congruent* **and** *betw* = *p-between*
proof

10. Euclid negation

show ∃ *a b c d t*. *p-between a d t* ∧ *p-between b d c* ∧ $a \neq d$ ∧
(∀ *x y*. *p-between a b x* ∧ *p-between a c y* ⟶ ¬ *p-between x t y*)
using *negated-euclidean-axiom*
by *transfer* (*auto*, *blast*)
next
fix *a x1 x2*
assume ¬ *TarskiAbsolute.on-line p-between a x1 x2*
hence ¬ *p-on-line a x1 x2*
using *TarskiAbsolute.on-line-def*[*OF PoincareTarskiAbsolute.TarskiAbsolute-axioms*]

```
    using PoincareTarskiAbsolute.on-line-def
    by simp
```

11. Limiting parallels

```
  thus ∃ a1 a2.
         ¬ TarskiAbsolute.on-line p-between a a1 a2 ∧
         ¬ TarskiAbsolute.ray-meets-line p-between a a1 x1 x2 ∧
         ¬ TarskiAbsolute.ray-meets-line p-between a a2 x1 x2 ∧
         (∀ a'. TarskiAbsolute.in-angle p-between a' a1 a a2 ⟶ TarskiAbsolute.ray-meets-line p-between a a' x1 x2)
    unfolding TarskiAbsolute.in-angle-def[OF PoincareTarskiAbsolute.TarskiAbsolute-axioms]
    unfolding TarskiAbsolute.on-ray-def[OF PoincareTarskiAbsolute.TarskiAbsolute-axioms]
    unfolding TarskiAbsolute.ray-meets-line-def[OF PoincareTarskiAbsolute.TarskiAbsolute-axioms]
    unfolding TarskiAbsolute.on-ray-def[OF PoincareTarskiAbsolute.TarskiAbsolute-axioms]
    unfolding TarskiAbsolute.on-line-def[OF PoincareTarskiAbsolute.TarskiAbsolute-axioms]
    unfolding PoincareTarskiAbsolute.on-line-def
    apply transfer
  proof−
    fix a x1 x2
    assume ∗: a ∈ unit-disc x1 ∈ unit-disc x2 ∈ unit-disc
              ¬ (poincare-between a x1 x2 ∨ poincare-between x1 a x2 ∨ poincare-between x1 x2 a)
    hence ¬ poincare-on-line a x1 x2
      using poincare-collinear3-iff[of a x1 x2]
      using poincare-between-rev poincare-on-line-def by blast
    hence ∃ a1∈unit-disc.
            ∃ a2∈unit-disc.
            ¬ poincare-on-line a a1 a2 ∧
            ¬ poincare-ray-meets-line a a1 x1 x2 ∧
            ¬ poincare-ray-meets-line a a2 x1 x2 ∧
            (∀ a'∈unit-disc.
                poincare-in-angle a' a1 a a2 ⟶
                poincare-ray-meets-line a a' x1 x2)
      using limiting-parallels[of a x1 x2] ∗
      by blast
    then obtain a1 a2 where ∗∗: a1∈unit-disc a2∈unit-disc ¬ poincare-on-line a a1 a2
                             ¬ poincare-ray-meets-line a a2 x1 x2
                             ¬ poincare-ray-meets-line a a1 x1 x2
                             ∀ a'∈unit-disc.
                                    poincare-in-angle a' a1 a a2 ⟶
                                    poincare-ray-meets-line a a' x1 x2
      by blast
    have ¬ (∃ x∈{z. z ∈ unit-disc}.
                  (poincare-between a x a1 ∨
                   poincare-between a a1 x) ∧
                  (poincare-between x x1 x2 ∨
                   poincare-between x1 x x2 ∨
                   poincare-between x1 x2 x))
      using ‹¬ poincare-ray-meets-line a a1 x1 x2›
      unfolding poincare-on-line-def poincare-ray-meets-line-def poincare-on-ray-def
      using poincare-collinear3-iff[of - x1 x2] poincare-between-rev ∗(2, 3)
      by auto
    moreover
    have ¬ (∃ x∈{z. z ∈ unit-disc}.
                  (poincare-between a x a2 ∨
                   poincare-between a a2 x) ∧
                  (poincare-between x x1 x2 ∨
                   poincare-between x1 x x2 ∨
                   poincare-between x1 x2 x))
      using ‹¬ poincare-ray-meets-line a a2 x1 x2›
      unfolding poincare-on-line-def poincare-ray-meets-line-def poincare-on-ray-def
      using poincare-collinear3-iff[of - x1 x2] poincare-between-rev ∗(2, 3)
      by auto
    moreover
    have ¬ (poincare-between a a1 a2 ∨ poincare-between a1 a a2 ∨ poincare-between a1 a2 a)
      using ‹¬ poincare-on-line a a1 a2› poincare-collinear3-iff[of a a1 a2]
      using ∗(1) ∗∗(1−2)
      unfolding poincare-on-line-def
```

by *simp*
moreover
have ($\forall$ *a′*∈{*z*. *z* ∈ *unit-disc*}.
a ≠ *a1* ∧
a ≠ *a2* ∧
a′ ≠ *a* ∧
(∃ *x*∈{*z*. *z* ∈ *unit-disc*}.
poincare-between a1 x a2 ∧
x ≠ *a1* ∧
x ≠ *a2* ∧
(*poincare-between a a′ x* ∨
poincare-between a x a′)) ⟶
(∃ *x*∈{*z*. *z* ∈ *unit-disc*}.
(*poincare-between a x a′* ∨
poincare-between a a′ x) ∧
(*poincare-between x x1 x2* ∨
poincare-between x1 x x2 ∨
poincare-between x1 x2 x)))
using ∗∗(*6*)
unfolding *poincare-on-line-def poincare-in-angle-def poincare-ray-meets-line-def poincare-on-ray-def*
using *poincare-collinear3-iff*[*of - x1 x2*] *poincare-between-rev* ∗(*2*, *3*)
by *auto*
ultimately
show ∃ *a1*∈{*z*. *z* ∈ *unit-disc*}.
∃ *a2*∈{*z*. *z* ∈ *unit-disc*}.
¬ (*poincare-between a a1 a2* ∨ *poincare-between a1 a a2* ∨ *poincare-between a1 a2 a*) ∧
¬ (∃ *x*∈{*z*. *z* ∈ *unit-disc*}.
(*poincare-between a x a1* ∨
poincare-between a a1 x) ∧
(*poincare-between x x1 x2* ∨
poincare-between x1 x x2 ∨
poincare-between x1 x2 x)) ∧
¬ (∃ *x*∈{*z*. *z* ∈ *unit-disc*}.
(*poincare-between a x a2* ∨
poincare-between a a2 x) ∧
(*poincare-between x x1 x2* ∨
poincare-between x1 x x2 ∨
poincare-between x1 x2 x)) ∧
(∀ *a′*∈{*z*. *z* ∈ *unit-disc*}.
a ≠ *a1* ∧
a ≠ *a2* ∧
a′ ≠ *a* ∧
(∃ *x*∈{*z*. *z* ∈ *unit-disc*}.
poincare-between a1 x a2 ∧
x ≠ *a1* ∧
x ≠ *a2* ∧
(*poincare-between a a′ x* ∨
poincare-between a x a′)) ⟶
(∃ *x*∈{*z*. *z* ∈ *unit-disc*}.
(*poincare-between a x a′* ∨
poincare-between a a′ x) ∧
(*poincare-between x x1 x2* ∨
poincare-between x1 x x2 ∨
poincare-between x1 x2 x)))
using ∗∗(*1*, *2*)
by *auto*
qed
qed

interpretation *PoincareElementaryTarskiHyperbolic*: *ElementaryTarskiHyperbolic p-congruent p-between*
proof

12. Continuity

fix *φ ψ*
assume ∃ *a*. ∀ *x*. ∀ *y*. *φ x* ∧ *ψ y* ⟶ *p-between a x y*
thus ∃ *b*. ∀ *x*. ∀ *y*. *φ x* ∧ *ψ y* ⟶ *p-between x b y*

apply *transfer*
using *continuity*
by *auto*
qed

end

References

[1] R. Coghetto. Klein-Beltrami Model. Part I. *Formalized Mathematics*, 26(1):21–32, 2018.

[2] R. Coghetto. Klein-Beltrami Model. Part II. *Formalized Mathematics*, 26(1):33–48, 2018.

[3] N. Lobatschewsky. *Geometrische Untersuchungen zur Theorie der Parallellinien*, pages 159–223. Springer Vienna, Vienna, 1985.

[4] T. J. M. Makarios. A Mechanical Verification of the Independence of Tarski's Euclidean Axiom. Master's thesis, Victoria University of Wellington, 2012. Master Thesis.

[5] F. Marić and D. Simić. Formalizing Complex Plane Geometry. *Annals of Mathematics and Artificial Intelligence*, 74(3-4):271–308, 2015.

[6] F. Mari and D. Simi. Complex geometry. *Archive of Formal Proofs*, Dec. 2019. http://isa-afp.org/entries/Complex_Geometry.html, Formal proof development.

[7] W. Schwabhäuser, W. Szmielew, and A. Tarski. *Metamathematische Methoden in der Geometrie*. Springer-Verlag, Berlin, 1983.

[8] H. Schwerdtfeger. *Geometry of Complex Numbers: Circle Geometry, Moebius Transformation, Non-euclidean Geometry*. Courier Corporation, 1979.