Complex Geometry

Filip Marić Danijela Simić

April 9, 2025

Abstract

A formalization of geometry of complex numbers is presented. Fundamental objects that are investigated are the complex plane extended by a single infinite point, its objects (points, lines and circles), and groups of transformations that act on them (e.g., inversions and Möbius transformations). Most objects are defined algebraically, but correspondence with classical geometric definitions is shown.

Contents

1 Introduction

The complex plane or some of its parts (e.g., the unit disc or the upper half plane) are often taken as the domain in which models of various geometries (both Euclidean and non-Euclidean ones) are formalized. The complex plane gives simpler and more compact formulas than the Cartesian plane. Within complex plane is easier to describe geometric objects and perform the calculations (usually shedding some new light on the subject). We give a formalization of the extended complex plane (given both as a complex projective space and as the Riemann sphere), its objects (points, circles and lines), and its transformations (Möbius transformations).

2 Related work

During the last decade, there have been many results in formalizing geometry in proof-assistants. Parts of Hilbert's seminal book „Foundations of Geometry" [6] have been formalized both in Coq and Isabelle/Isar. Formalization of first two groups of axioms in Coq, in an intuitionistic setting was done by Dehlinger et al. [1]. First formalization in Isabelle/HOL was done by Fleuriot and Meikele [8], and some further developments were made in master thesis of Scott [14]. Large fragments of Tarski's geometry [12] have been formalized in Coq by Narboux et al. [9]. Within Coq, there are also formalizations of von Plato's constructive geometry by Kahn [15, 7], French high school geometry by Guilhot [3] and ruler and compass geometry by Duprat [2], etc.
In our previous work [11], we have already formally investigated a Cartesian model of Euclidean geometry.

3 Background theories

In this section we introduce some basic mathematical notions and prove some lemmas needed in the rest of our formalization. We describe:

- trigonometric functions,
- complex numbers,
- systems of two and three linear equations with two unknowns (over arbitrary fields),
- quadratic equations (over real and complex numbers), systems of quadratic and real equations, and systems of two quadratic equations,
- two-dimensional vectors and matrices over complex numbers.

3.1 Library Additions for Trigonometric Functions

theory *More-Transcendental*
  imports *Complex-Main HOL−Library.Periodic-Fun*
begin

Additional properties of *sin* and *cos* functions that are later used in proving conjectures for argument of complex number.

Sign of trigonometric functions on some characteristic intervals.

lemma *cos-lt-zero-on-pi2-pi* [*simp*]:
  assumes $x > pi/2$ **and** $x \leq pi$
  shows $cos\ x < 0$
  using *cos-gt-zero-pi*[*of pi − x*] *assms*
  by *simp*

Value of trigonometric functions in points $k\pi$ and $\frac{\pi}{2} + k\pi$.

lemma *sin-kpi* [*simp*]:
  fixes *k*::*int*
  shows $sin\ (k * pi) = 0$
  by (*simp add*: *sin-zero-iff-int2*)

lemma *cos-odd-kpi* [*simp*]:
  fixes *k*::*int*
  assumes *odd k*
  shows $cos\ (k * pi) = -1$
  by (*simp add*: *assms mult.commute*)

lemma *cos-even-kpi* [*simp*]:
  fixes *k*::*int*
  assumes *even k*
  shows $cos\ (k * pi) = 1$
  by (*simp add*: *assms mult.commute*)

lemma *sin-pi2-plus-odd-kpi* [*simp*]:

fixes *k*::*int*
assumes *odd k*
shows *sin* (*pi* / *2* + *k* ∗ *pi*) = −*1*
using *assms*
by (*simp add*: *sin-add*)

lemma *sin-pi2-plus-even-kpi* [*simp*]:
fixes *k*::*int*
assumes *even k*
shows *sin* (*pi* / *2* + *k* ∗ *pi*) = *1*
using *assms*
by (*simp add*: *sin-add*)

Solving trigonometric equations and systems with special values (0, 1, or -1) of sine and cosine functions

lemma *cos-0-iff-canon*:
assumes *cos* φ = *0* **and** −*pi* < φ **and** φ ≤ *pi*
shows φ = *pi*/*2* ∨ φ = −*pi*/*2*
by (*smt* (*verit*, *best*) *arccos-0 arccos-cos assms cos-minus divide-minus-left*)

lemma *sin-0-iff-canon*:
assumes *sin* φ = *0* **and** −*pi* < φ **and** φ ≤ *pi*
shows φ = *0* ∨ φ = *pi*
using *assms sin-eq-0-pi* **by** *force*

lemma *cos0-sin1*:
assumes *sin* φ = *1*
shows ∃ *k*::*int*. φ = *pi*/*2* + *2*∗*k*∗*pi*
by (*smt* (*verit*, *ccfv-threshold*) *assms cos-diff cos-one-2pi-int cos-pi-half mult-cancel-right1 sin-pi-half sin-plus-pi*)

Sine is injective on $[-\frac{\pi}{2}, \frac{\pi}{2}]$

lemma *sin-inj*:
assumes −*pi*/*2* ≤ α ∧ α ≤ *pi*/*2* **and** −*pi*/*2* ≤ α′ ∧ α′ ≤ *pi*/*2*
assumes α ≠ α′
shows *sin* α ≠ *sin* α′
by (*metis assms divide-minus-left sin-inj-pi*)

Periodicity of trigonometric functions

The following are available in HOL-Decision_Procs.Approximation_Bounds, but we want to avoid that dependency

lemma *sin-periodic-nat* [*simp*]:
fixes *n* :: *nat*
shows *sin* (*x* + *n* ∗ (*2* ∗ *pi*)) = *sin x*
by (*metis* (*no-types*, *opaque-lifting*) *add.commute add.left-neutral cos-2npi cos-one-2pi-int mult.assoc mult.commute mult.left-neutral mult-zero-left sin-add sin-int-2pin*)

lemma *sin-periodic-int* [*simp*]:
fixes *i* :: *int*
shows *sin* (*x* + *i* ∗ (*2* ∗ *pi*)) = *sin x*
by (*metis add.right-neutral cos-int-2pin mult.commute mult.right-neutral mult-zero-right sin-add sin-int-2pin*)

lemma *cos-periodic-nat* [*simp*]:
fixes *n* :: *nat*
shows *cos* (*x* + *n* ∗ (*2* ∗ *pi*)) = *cos x*
by (*metis add.left-neutral cos-2npi cos-add cos-periodic mult.assoc mult-2 mult-2-right of-nat-numeral sin-periodic sin-periodic-nat*)

lemma *cos-periodic-int* [*simp*]:
fixes *i* :: *int*
shows *cos* (*x* + *i* ∗ (*2* ∗ *pi*)) = *cos x*
by (*metis cos-add cos-int-2pin diff-zero mult.commute mult.right-neutral mult-zero-right sin-int-2pin*)

Values of both sine and cosine are repeated only after multiples of $2 \cdot \pi$

lemma *sin-cos-eq*:
fixes *a b* :: *real*

assumes *cos a = cos b* **and** *sin a = sin b*
shows ∃ *k*::*int*. *a − b = 2∗k∗pi*
by (*metis assms cos-diff cos-one-2pi-int mult.commute sin-cos-squared-add3*)

The following two lemmas are consequences of surjectivity of cosine for the range $[-1, 1]$.

lemma *ex-cos-eq*:
assumes *−pi/2 ≤ α ∧ α ≤ pi/2*
assumes *a ≥ 0* **and** *a < 1*
shows ∃ *α′*. *−pi/2 ≤ α′ ∧ α′ ≤ pi/2 ∧ α′ ≠ α ∧ cos (α − α′) = a*
proof−
have *arccos a > 0 arccos a ≤ pi/2*
using ‹*a ≥ 0*› ‹*a < 1*›
using *arccos-lt-bounded arccos-le-pi2*
by *auto*

show *?thesis*
proof (*cases α − arccos a ≥ − pi/2*)
case *True*
thus *?thesis*
using *assms* ‹*arccos a > 0*› ‹*arccos a ≤ pi/2*›
by (*rule-tac x = α − arccos a* **in** *exI*) *auto*
next
case *False*
thus *?thesis*
using *assms* ‹*arccos a > 0*› ‹*arccos a ≤ pi/2*›
by (*rule-tac x = α + arccos a* **in** *exI*) *auto*
qed
qed

lemma *ex-cos-gt*:
assumes *−pi/2 ≤ α ∧ α ≤ pi/2*
assumes *a < 1*
shows ∃ *α′*. *−pi/2 ≤ α′ ∧ α′ ≤ pi/2 ∧ α′ ≠ α ∧ cos (α − α′) > a*
proof−
obtain *a′* **where** *a′ ≥ 0 a′ > a a′ < 1*
by (*metis assms*(*2*) *dense-le-bounded linear not-one-le-zero*)
thus *?thesis*
using *ex-cos-eq*[*of α a′*] *assms*
by *auto*
qed

The function *atan2* is a generalization of *arctan* that takes a pair of coordinates of non-zero points returns its angle in the range $[-\pi, \pi)$.

definition *atan2* **where**
atan2 y x =
(*if x > 0 then arctan* (*y*/*x*)
else if x < 0 then
if y > 0 then arctan (*y*/*x*) + *pi else arctan* (*y*/*x*) − *pi*
else
if y > 0 then pi/*2 else if y < 0 then −pi*/*2 else 0*)

lemma *atan2-bounded*:
shows *−pi ≤ atan2 y x ∧ atan2 y x < pi*
using *arctan-bounded*[*of y*/*x*] *zero-le-arctan-iff*[*of y*/*x*] *arctan-le-zero-iff*[*of y*/*x*] *zero-less-arctan-iff*[*of y*/*x*] *arctan-less-zero-iff*[*of y*/*x*]
using *divide-neg-neg*[*of y x*] *divide-neg-pos*[*of y x*] *divide-pos-pos*[*of y x*] *divide-pos-neg*[*of y x*]
unfolding *atan2-def*
by (*simp* (*no-asm-simp*)) *auto*

end

3.2 Canonical angle

Canonize any angle to $(-\pi, \pi]$ (taking account of 2π periodicity of *sin* and *cos*). With this function, for example, multiplicative properties of *arg* for complex numbers can easily be expressed and proved.

theory *Canonical-Angle*
imports *More-Transcendental*
begin

abbreviation *canon-ang-P* **where**
canon-ang-P $\alpha\ \alpha' \equiv (-pi < \alpha' \wedge \alpha' \leq pi) \wedge (\exists\ k{::}int.\ \alpha - \alpha' = 2{*}k{*}pi)$

definition *canon-ang* :: *real* ⇒ *real* (‹⌊-⌋›) **where**
$\lfloor\alpha\rfloor = (THE\ \alpha'.\ canon\text{-}ang\text{-}P\ \alpha\ \alpha')$

There is a canonical angle for every angle.

lemma *canon-ang-ex*:
shows $\exists\ \alpha'.\ canon\text{-}ang\text{-}P\ \alpha\ \alpha'$
proof−
have ∗∗∗: $\forall\ \alpha{::}real.\ \exists\ \alpha'.\ 0 < \alpha' \wedge \alpha' \leq 1 \wedge (\exists\ k{::}int.\ \alpha' = \alpha - k)$
proof
fix $\alpha{::}real$
show $\exists \alpha'{>}0.\ \alpha' \leq 1 \wedge (\exists k{::}int.\ \alpha' = \alpha - k)$
proof (*cases* $\alpha = floor\ \alpha$)
case *True*
thus *?thesis*
by (*rule-tac* $x{=}\alpha - floor\ \alpha + 1$ **in** *exI*, *auto*) (*rule-tac* $x{=}floor\ \alpha - 1$ **in** *exI*, *auto*)
next
case *False*
thus *?thesis*
using *real-of-int-floor-ge-diff-one*[*of* α]
using *of-int-floor-le*[*of* α]
by (*rule-tac* $x{=}\alpha - floor\ \alpha$ **in** *exI*) (*smt* (*verit*))
qed
qed

have ∗∗: $\forall\ \alpha{::}real.\ \exists\ \alpha'.\ 0 < \alpha' \wedge \alpha' \leq 2 \wedge (\exists\ k{::}int.\ \alpha - \alpha' = 2{*}k - 1)$
proof
fix $\alpha{::}real$
from ∗∗∗[*rule-format*, *of* $(\alpha + 1)\ /2$]
obtain α' **and** $k{::}int$ **where** $0 < \alpha'\ \alpha' \leq 1\ \alpha' = (\alpha + 1)/2 - k$
by *force*
hence $0 < \alpha'\ \alpha' \leq 1\ \alpha' = \alpha/2 - k + 1/2$
by *auto*
thus $\exists \alpha'{>}0.\ \alpha' \leq 2 \wedge (\exists k{::}int.\ \alpha - \alpha' = real\text{-}of\text{-}int\ (2 * k - 1))$
by (*rule-tac* $x{=}2{*}\alpha'$ **in** *exI*) *auto*
qed
have ∗: $\forall\ \alpha{::}real.\ \exists\ \alpha'.\ -1 < \alpha' \wedge \alpha' \leq 1 \wedge (\exists\ k{::}int.\ \alpha - \alpha' = 2{*}k)$
proof
fix $\alpha{::}real$
from ∗∗ **obtain** α' **and** $k :: int$ **where**
$0 < \alpha' \wedge \alpha' \leq 2 \wedge \alpha - \alpha' = 2{*}k - 1$
by *force*
thus $\exists \alpha'{>}{-}1.\ \alpha' \leq 1 \wedge (\exists k.\ \alpha - \alpha' = real\text{-}of\text{-}int\ (2 * (k{::}int)))$
by (*rule-tac* $x{=}\alpha' - 1$ **in** *exI*) (*auto simp add*: *field-simps*)
qed
obtain $\alpha'\ k$ **where** *1*: $\alpha' >{-} 1 \wedge \alpha' \leq 1$ **and** *2*: $\alpha\ /\ pi - \alpha' = real\text{-}of\text{-}int\ (2 * k)$
using ∗[*rule-format*, *of* $\alpha\ /\ pi$]
by *auto*
have $\alpha'{*}pi > -pi \wedge \alpha'{*}pi \leq pi$
using *1*
by (*smt* (*verit*) *mult.commute mult-le-cancel-left1 mult-minus-right pi-gt-zero*)
moreover
have $\alpha - \alpha'{*}pi = 2 * real\text{-}of\text{-}int\ k * pi$
using *2*
by (*auto simp add*: *field-simps*)
ultimately
show *?thesis*
by *auto*
qed

Canonical angle of any angle is unique.

lemma *canon-ang-unique*:
assumes *canon-ang-P* α α_1 **and** *canon-ang-P* α α_2
shows $\alpha_1 = \alpha_2$
proof−
obtain *k1*::*int* **where** $\alpha - \alpha_1 = 2{*}k1{*}pi$
using *assms*(*1*)
by *auto*
obtain *k2*::*int* **where** $\alpha - \alpha_2 = 2{*}k2{*}pi$
using *assms*(*2*)
by *auto*
hence ∗: $-\alpha_1 + \alpha_2 = 2{*}(k1 - k2){*}pi$
using ‹$\alpha - \alpha_1 = 2{*}k1{*}pi$›
by (*simp add*:*field-simps*)
moreover
have $-\alpha_1 + \alpha_2 < 2 * pi$ $-\alpha_1 + \alpha_2 > -2{*}pi$
using *assms*
by *auto*
ultimately
have $-\alpha_1 + \alpha_2 = 0$
using *mult-less-cancel-right*[*of* -2 *pi real-of-int*$(2 * (k1 - k2))$]
by *auto*
thus *?thesis*
by *auto*
qed

Canonical angle is always in $(-\pi, \pi]$ and differs from the starting angle by $2k\pi$.

lemma *canon-ang*:
shows $-pi < \lfloor\alpha\rfloor$ **and** $\lfloor\alpha\rfloor \leq pi$ **and** ∃ *k*::*int*. $\alpha - \lfloor\alpha\rfloor = 2{*}k{*}pi$
proof−
obtain α' **where** *canon-ang-P* α α'
using *canon-ang-ex*[*of* α]
by *auto*
have *canon-ang-P* α $\lfloor\alpha\rfloor$
unfolding *canon-ang-def*
proof (*rule theI*[**where** *a*=α'])
show *canon-ang-P* α α'
by *fact*
next
fix α''
assume *canon-ang-P* α α''
thus $\alpha'' = \alpha'$
using ‹*canon-ang-P* α α'›
using *canon-ang-unique*[*of* α' α α'']
by *simp*
qed
thus $-pi < \lfloor\alpha\rfloor$ $\lfloor\alpha\rfloor \leq pi$ ∃ *k*::*int*. $\alpha - \lfloor\alpha\rfloor = 2{*}k{*}pi$
by *auto*
qed

Angles in $(-\pi, \pi]$ are already canonical.

lemma *canon-ang-id*:
assumes $-pi < \alpha \wedge \alpha \leq pi$
shows $\lfloor\alpha\rfloor = \alpha$
using *assms*
using *canon-ang-unique*[*of canon-ang* α α α] *canon-ang*[*of* α]
by *auto*

Angles that differ by $2k\pi$ have equal canonical angles.

lemma *canon-ang-eq*:
assumes ∃ *k*::*int*. $\alpha_1 - \alpha_2 = 2{*}k{*}pi$
shows $\lfloor\alpha_1\rfloor = \lfloor\alpha_2\rfloor$
proof−
obtain k'::*int* **where** ∗: $- pi < \lfloor\alpha_1\rfloor$ $\lfloor\alpha_1\rfloor \leq pi$ $\alpha_1 - \lfloor\alpha_1\rfloor = 2 * k' * pi$
using *canon-ang*[*of* α_1]

by *auto*

obtain k''::*int* **where** $**$: $-\ pi < \lfloor\alpha_2\rfloor\ \lfloor\alpha_2\rfloor \leq pi\ \alpha_2 - \lfloor\alpha_2\rfloor = 2 * k'' * pi$
using *canon-ang*[*of* α_2]
by *auto*

obtain *k*::*int* **where** $***$: $\alpha_1 - \alpha_2 = 2*k*pi$
using *assms*
by *auto*

have $\exists\, m$::*int*. $\alpha_1 - \lfloor\alpha_2\rfloor = 2 * m * pi$
using $**(3)$ $***$
by (*rule-tac x*=*k*+k'' **in** *exI*) (*auto simp add*: *field-simps*)

thus *?thesis*
using *canon-ang-unique*[*of* $\lfloor\alpha_1\rfloor\ \alpha_1\ \lfloor\alpha_2\rfloor$] $* **$
by *auto*
qed

Introduction and elimination rules

lemma *canon-ang-eqI*:
assumes $\exists\, k$::*int*. $\alpha' - \alpha = 2 * k * pi$ **and** $-\ pi < \alpha' \wedge \alpha' \leq pi$
shows $\lfloor\alpha\rfloor = \alpha'$
using *assms*
using *canon-ang-eq*[*of* $\alpha'\ \alpha$]
using *canon-ang-id*[*of* α']
by *auto*

lemma *canon-ang-eqE*:
assumes $\lfloor\alpha_1\rfloor = \lfloor\alpha_2\rfloor$
shows $\exists\ (k$::*int*). $\alpha_1 - \alpha_2 = 2 *k * pi$
proof−
obtain *k1 k2* :: *int* **where**
$\alpha_1 - \lfloor\alpha_1\rfloor = 2 * k1 * pi$
$\alpha_2 - \lfloor\alpha_2\rfloor = 2 * k2 * pi$
using *canon-ang*[*of* α_1] *canon-ang*[*of* α_2]
by *auto*
thus *?thesis*
using *assms*
by (*rule-tac x*=*k1 − k2* **in** *exI*) (*auto simp add*: *field-simps*)
qed

Canonical angle of opposite angle

lemma *canon-ang-uminus*:
assumes $\lfloor\alpha\rfloor \neq pi$
shows $\lfloor-\alpha\rfloor = -\lfloor\alpha\rfloor$
proof (*rule canon-ang-eqI*)
show $\exists\, x$::*int*. $-\ \lfloor\alpha\rfloor - - \alpha = 2 * x * pi$
using *canon-ang*(*3*)[*of* α]
by (*metis minus-diff-eq minus-diff-minus*)
next
show $-\ pi < -\ \lfloor\alpha\rfloor \wedge -\ \lfloor\alpha\rfloor \leq pi$
using *canon-ang*(*1*)[*of* α] *canon-ang*(*2*)[*of* α] *assms*
by *auto*
qed

lemma *canon-ang-uminus-pi*:
assumes $\lfloor\alpha\rfloor = pi$
shows $\lfloor-\alpha\rfloor = \lfloor\alpha\rfloor$
proof (*rule canon-ang-eqI*)
obtain *k*::*int* **where** $\alpha - \lfloor\alpha\rfloor = 2 * k * pi$
using *canon-ang*(*3*)[*of* α]
by *auto*
thus $\exists\, x$::*int*. $\lfloor\alpha\rfloor - - \alpha = 2 * x * pi$
using *assms*
by (*rule-tac x*=*k*+(*1*::*int*) **in** *exI*) (*auto simp add*: *field-simps*)

next
show $-pi < \lfloor\alpha\rfloor \wedge \lfloor\alpha\rfloor \leq pi$
using *assms*
by *auto*
qed

Canonical angle of difference of two angles

lemma *canon-ang-diff*:
shows $\lfloor\alpha - \beta\rfloor = \lfloor\lfloor\alpha\rfloor - \lfloor\beta\rfloor\rfloor$
proof (*rule canon-ang-eq*)
show $\exists\, x{::}int.\ \alpha - \beta - (\lfloor\alpha\rfloor - \lfloor\beta\rfloor) = 2 * x * pi$
proof−
obtain *k1*::*int* **where** $\alpha - \lfloor\alpha\rfloor = 2{*}k1{*}pi$
using *canon-ang*(*3*)
by *auto*
moreover
obtain *k2*::*int* **where** $\beta - \lfloor\beta\rfloor = 2{*}k2{*}pi$
using *canon-ang*(*3*)
by *auto*
ultimately
show *?thesis*
by (*rule-tac x=k1 − k2* **in** *exI*) (*auto simp add*: *field-simps*)
qed
qed

Canonical angle of sum of two angles

lemma *canon-ang-sum*:
shows $\lfloor\alpha + \beta\rfloor = \lfloor\lfloor\alpha\rfloor + \lfloor\beta\rfloor\rfloor$
proof (*rule canon-ang-eq*)
show $\exists\, x{::}int.\ \alpha + \beta - (\lfloor\alpha\rfloor + \lfloor\beta\rfloor) = 2 * x * pi$
proof−
obtain *k1*::*int* **where** $\alpha - \lfloor\alpha\rfloor = 2{*}k1{*}pi$
using *canon-ang*(*3*)
by *auto*
moreover
obtain *k2*::*int* **where** $\beta - \lfloor\beta\rfloor = 2{*}k2{*}pi$
using *canon-ang*(*3*)
by *auto*
ultimately
show *?thesis*
by (*rule-tac x=k1 + k2* **in** *exI*) (*auto simp add*: *field-simps*)
qed
qed

Canonical angle of angle from $(0, 2\pi]$ shifted by π

lemma *canon-ang-plus-pi1*:
assumes $0 < \alpha$ **and** $\alpha \leq 2{*}pi$
shows $\lfloor\alpha + pi\rfloor = \alpha - pi$
proof (*rule canon-ang-eqI*)
show $\exists\ x{::}int.\ \alpha - pi - (\alpha + pi) = 2 * x * pi$
by (*rule-tac x=−1* **in** *exI*) *auto*
next
show $-pi < \alpha - pi \wedge \alpha - pi \leq pi$
using *assms*
by *auto*
qed

lemma *canon-ang-minus-pi1*:
assumes $0 < \alpha$ **and** $\alpha \leq 2{*}pi$
shows $\lfloor\alpha - pi\rfloor = \alpha - pi$
proof (*rule canon-ang-id*)
show $-pi < \alpha - pi \wedge \alpha - pi \leq pi$
using *assms*
by *auto*
qed

Canonical angle of angles from $(-2\pi, 0]$ shifted by π

lemma *canon-ang-plus-pi2*:
  assumes $-2*pi < \alpha$ **and** $\alpha \leq 0$
  shows $\lfloor \alpha + pi \rfloor = \alpha + pi$
proof (*rule canon-ang-id*)
  show $-pi < \alpha + pi \wedge \alpha + pi \leq pi$
    using *assms*
    by *auto*
qed

lemma *canon-ang-minus-pi2*:
  assumes $-2*pi < \alpha$ **and** $\alpha \leq 0$
  shows $\lfloor \alpha - pi \rfloor = \alpha + pi$
proof (*rule canon-ang-eqI*)
  show $\exists\ x{::}int.\ \alpha + pi - (\alpha - pi) = 2 * x * pi$
    by (*rule-tac x=1* **in** *exI*) *auto*
next
  show $-pi < \alpha + pi \wedge \alpha + pi \leq pi$
    using *assms*
    by *auto*
qed

Canonical angle of angle in $(\pi, 3\pi]$.

lemma *canon-ang-pi-3pi*:
  assumes $pi < \alpha$ **and** $\alpha \leq 3 * pi$
  shows $\lfloor \alpha \rfloor = \alpha - 2*pi$
proof−
  have $\exists x.\ -pi = pi * real\text{-}of\text{-}int\ x$
    by (*rule-tac x=−1* **in** *exI*, *simp*)
  thus *?thesis*
    using *assms canon-ang-eqI*[*of* $\alpha - 2*pi\ \alpha$]
    by *auto*
qed

Canonical angle of angle in $(-3\pi, -\pi]$.

lemma *canon-ang-minus-3pi-minus-pi*:
  assumes $-3*pi < \alpha$ **and** $\alpha \leq -pi$
  shows $\lfloor \alpha \rfloor = \alpha + 2*pi$
proof−
  have $\exists x.\ pi = pi * real\text{-}of\text{-}int\ x$
    by (*rule-tac x=1* **in** *exI*, *simp*)
  thus *?thesis*
    using *assms canon-ang-eqI*[*of* $\alpha + 2*pi\ \alpha$]
    by *auto*
qed

Canonical angles for some special angles

lemma *zero-canonical* [*simp*]:
  shows $\lfloor 0 \rfloor = 0$
  using *canon-ang-eqI*[*of 0 0*]
  by *simp*

lemma *pi-canonical* [*simp*]:
  shows $\lfloor pi \rfloor = pi$
  by (*simp add*: *canon-ang-id*)

lemma *two-pi-canonical* [*simp*]:
  shows $\lfloor 2 * pi \rfloor = 0$
  using *canon-ang-plus-pi1*[*of pi*]
  by *simp*

Canonization preserves sine and cosine

lemma *canon-ang-sin* [*simp*]:
  shows $sin\ \lfloor \alpha \rfloor = sin\ \alpha$
proof−

obtain x::*int* **where** $\alpha = \lfloor\alpha\rfloor + pi * (x * 2)$
using *canon-ang*(*3*)[*of* α]
by (*auto simp add*: *field-simps*)
thus *?thesis*
using *sin-periodic-int*[*of* $\lfloor\alpha\rfloor$ x]
by (*simp add*: *field-simps*)
qed

lemma *canon-ang-cos* [*simp*]:
shows *cos* $\lfloor\alpha\rfloor$ = *cos* α
proof−
obtain x::*int* **where** $\alpha = \lfloor\alpha\rfloor + pi * (x * 2)$
using *canon-ang*(*3*)[*of* α]
by (*auto simp add*: *field-simps*)
thus *?thesis*
using *cos-periodic-int*[*of* $\lfloor\alpha\rfloor$ x]
by (*simp add*: *field-simps*)
qed

end

3.3 Library Additions for Complex Numbers

Some additional lemmas about complex numbers.

theory *More-Complex*
imports *Complex-Main More-Transcendental Canonical-Angle*
begin

Conjugation and *cis*

declare *cis-cnj*[*simp*]

lemmas *complex-cnj* = *complex-cnj-diff complex-cnj-mult complex-cnj-add complex-cnj-divide complex-cnj-minus*

Some properties for *complex-of-real*. Also, since it is often used in our formalization we abbreviate it to *cor*.

abbreviation *cor* :: *real* ⇒ *complex* **where**
cor ≡ *complex-of-real*

lemma *cmod-cis* [*simp*]:
assumes $a \neq 0$
shows *of-real* (*cmod a*) ∗ *cis* (*Arg a*) = *a*
using *assms*
by (*metis rcis-cmod-Arg rcis-def*)

lemma *cis-cmod* [*simp*]:
assumes $a \neq 0$
shows *cis* (*Arg a*) ∗ *of-real* (*cmod a*) = *a*
by (*metis assms cmod-cis mult.commute*)

lemma *cor-squared*:
shows $(cor\ x)^2 = cor\ (x^2)$
by (*simp add*: *power2-eq-square*)

lemma *cor-sqrt-mult-cor-sqrt* [*simp*]:
shows *cor* (*sqrt A*) ∗ *cor* (*sqrt A*) = *cor* |*A*|
by (*metis of-real-mult real-sqrt-mult-self*)

lemma *cor-eq-0*: *cor x* + i ∗ *cor y* = *0* ⟷ *x* = *0* ∧ *y* = *0*
by (*metis Complex-eq Im-complex-of-real Im-i-times Re-complex-of-real add-cancel-left-left of-real-eq-0-iff plus-complex.sel*(*2*) *zero-complex.code*)

lemma *one-plus-square-neq-zero* [*simp*]:
shows $1 + (cor\ x)^2 \neq 0$
by (*metis* (*opaque-lifting*, *no-types*) *of-real-1 of-real-add of-real-eq-0-iff of-real-power power-one sum-power2-eq-zero-iff zero-neq-one*)

Additional lemmas about *Complex* constructor. Following newer versions of Isabelle, these should be deprecated.

lemma *complex-real-two* [*simp*]:
shows *Complex 2 0 = 2*
by (*simp add*: *Complex-eq*)

lemma *complex-double* [*simp*]:
shows (*Complex a b*) * *2* = *Complex* (*2***a*) (*2***b*)
by (*simp add*: *Complex-eq*)

lemma *complex-half* [*simp*]:
shows (*Complex a b*) / *2* = *Complex* (*a*/*2*) (*b*/*2*)
by (*subst complex-eq-iff*) *auto*

lemma *Complex-scale1*:
shows *Complex* (*a* * *b*) (*a* * *c*) = *cor a* * *Complex b c*
unfolding *complex-of-real-def*
unfolding *Complex-eq*
by (*auto simp add*: *field-simps*)

lemma *Complex-scale2*:
shows *Complex* (*a* * *c*) (*b* * *c*) = *Complex a b* * *cor c*
unfolding *complex-of-real-def*
unfolding *Complex-eq*
by (*auto simp add*: *field-simps*)

lemma *Complex-scale3*:
shows *Complex* (*a* / *b*) (*a* / *c*) = *cor a* * *Complex* (*1* / *b*) (*1* / *c*)
unfolding *complex-of-real-def*
unfolding *Complex-eq*
by (*auto simp add*: *field-simps*)

lemma *Complex-scale4*:
shows *c* ≠ *0* ⟹ *Complex* (*a* / *c*) (*b* / *c*) = *Complex a b* / *cor c*
unfolding *complex-of-real-def*
unfolding *Complex-eq*
by (*auto simp add*: *field-simps power2-eq-square*)

lemma *Complex-Re-express-cnj*:
shows *Complex* (*Re z*) *0* = (*z* + *cnj z*) / *2*
by (*cases z*) (*simp add*: *Complex-eq*)

lemma *Complex-Im-express-cnj*:
shows *Complex 0* (*Im z*) = (*z* − *cnj z*)/*2*
by (*cases z*) (*simp add*: *Complex-eq*)

Additional properties of *cmod*.

lemma *complex-mult-cnj-cmod*:
shows *z* * *cnj z* = *cor* ((*cmod z*)^2)
using *complex-norm-square*
by *auto*

lemma *cmod-square*:
shows (*cmod z*)^2 = *Re* (*z* * *cnj z*)
using *complex-mult-cnj-cmod*[*of z*]
by (*simp add*: *power2-eq-square*)

lemma *cor-cmod-power-4* [*simp*]:
shows *cor* (*cmod z*) ^ *4* = (*z* * *cnj z*)^2
by (*simp add*: *complex-mult-cnj-cmod*)

lemma *cnjE*:
assumes *x* ≠ *0*
shows *cnj x* = *cor* ((*cmod x*)^2) / *x*
using *complex-mult-cnj-cmod*[*of x*] *assms*
by (*auto simp add*: *field-simps*)

lemma *cmod-cor-divide* [*simp*]:

shows *cmod* (*z* / *cor k*) = *cmod z* / |*k*|
by (*simp add*: *norm-divide*)

lemma *cmod-mult-minus-left-distrib* [*simp*]:
shows *cmod* (*z*∗*z1* − *z*∗*z2*) = *cmod z* ∗ *cmod*(*z1* − *z2*)
by (*metis norm-mult right-diff-distrib*)

lemma *cmod-eqI*:
assumes *z1* ∗ *cnj z1* = *z2* ∗ *cnj z2*
shows *cmod z1* = *cmod z2*
using *assms*
by (*subst complex-mod-sqrt-Re-mult-cnj*)+ *auto*

lemma *cmod-eqE*:
assumes *cmod z1* = *cmod z2*
shows *z1* ∗ *cnj z1* = *z2* ∗ *cnj z2*
by (*simp add*: *assms complex-mult-cnj-cmod*)

lemma *cmod-eq-one* [*simp*]:
shows *cmod a* = *1* ⟷ *a*∗*cnj a* = *1*
by (*metis cmod-eqE cmod-eqI complex-cnj-one monoid-mult-class.mult.left-neutral norm-one*)

We introduce *is-real* (the imaginary part of complex number is zero) and *is-imag* (real part of complex number is zero) operators and prove some of their properties.

abbreviation *is-real* **where**
is-real z ≡ *Im z* = *0*

abbreviation *is-imag* **where**
is-imag z ≡ *Re z* = *0*

lemma *real-imag-0*:
assumes *is-real a is-imag a*
shows *a* = *0*
using *assms*
by (*simp add*: *complex.expand*)

lemma *complex-eq-if-Re-eq*:
assumes *is-real z1* **and** *is-real z2*
shows *z1* = *z2* ⟷ *Re z1* = *Re z2*
using *assms*
by (*cases z1*, *cases z2*) *auto*

lemma *mult-reals* [*simp*]:
assumes *is-real a* **and** *is-real b*
shows *is-real* (*a* ∗ *b*)
using *assms*
by *auto*

lemma *div-reals* [*simp*]:
assumes *is-real a* **and** *is-real b*
shows *is-real* (*a* / *b*)
using *assms*
by (*simp add*: *complex-is-Real-iff*)

lemma *complex-of-real-Re* [*simp*]:
assumes *is-real k*
shows *cor* (*Re k*) = *k*
using *assms*
by (*cases k*) (*auto simp add*: *complex-of-real-def*)

lemma *cor-cmod-real*:
assumes *is-real a*
shows *cor* (*cmod a*) = *a* ∨ *cor* (*cmod a*) = −*a*
using *assms*
unfolding *cmod-def*
by (*cases Re a* > *0*) *auto*

lemma *eq-cnj-iff-real*:
shows *cnj z = z ⟷ is-real z*
by (*cases z*) (*simp add*: *Complex-eq*)

lemma *eq-minus-cnj-iff-imag*:
shows *cnj z = −z ⟷ is-imag z*
by (*cases z*) (*simp add*: *Complex-eq*)

lemma *Re-divide-real*:
assumes *is-real b* **and** *b ≠ 0*
shows *Re (a / b) = (Re a) / (Re b)*
using *assms*
by (*simp add*: *complex-is-Real-iff*)

lemma *Re-mult-real*:
assumes *is-real a*
shows *Re (a * b) = (Re a) * (Re b)*
using *assms*
by *simp*

lemma *Im-mult-real*:
assumes *is-real a*
shows *Im (a * b) = (Re a) * (Im b)*
using *assms*
by *simp*

lemma *Im-divide-real*:
assumes *is-real b* **and** *b ≠ 0*
shows *Im (a / b) = (Im a) / (Re b)*
using *assms*
by (*simp add*: *complex-is-Real-iff*)

lemma *Re-sgn*:
assumes *is-real R*
shows *Re (sgn R) = sgn (Re R)*
using *assms*
by (*metis Re-sgn complex-of-real-Re norm-of-real real-sgn-eq*)

lemma *is-real-div*:
assumes *b ≠ 0*
shows *is-real (a / b) ⟷ a∗cnj b = b∗cnj a*
using *assms*
by (*metis complex-cnj-divide complex-cnj-zero-iff eq-cnj-iff-real frac-eq-eq mult.commute*)

lemma *is-real-mult-real*:
assumes *is-real a* **and** *a ≠ 0*
shows *is-real b ⟷ is-real (a * b)*
using *assms*
by (*cases a*, *auto simp add*: *Complex-eq*)

lemma *Im-express-cnj*:
shows *Im z = (z − cnj z) / (2 * ı)*
by (*simp add*: *complex-diff-cnj field-simps*)

lemma *Re-express-cnj*:
shows *Re z = (z + cnj z) / 2*
by (*simp add*: *complex-add-cnj*)

Rotation of complex number for 90 degrees in the positive direction.

abbreviation *rot90* **where**
rot90 z ≡ Complex (−Im z) (Re z)

lemma *rot90-ii*:
shows *rot90 z = z * ı*
by (*metis Complex-mult-i complex-surj*)

With *cnj-mix* we introduce scalar product between complex vectors. This operation shows to be useful to succinctly express some conditions.

abbreviation *cnj-mix* **where**
cnj-mix z1 z2 ≡ *cnj z1* ∗ *z2* + *z1* ∗ *cnj z2*

abbreviation *scalprod* **where**
scalprod z1 z2 ≡ *cnj-mix z1 z2* / *2*

lemma *cnj-mix-minus*:
shows *cnj z1*∗*z2* − *z1*∗*cnj z2* = i ∗ *cnj-mix* (*rot90 z1*) *z2*
by (*cases z1*, *cases z2*) (*simp add*: *Complex-eq field-simps*)

lemma *cnj-mix-minus′*:
shows *cnj z1*∗*z2* − *z1*∗*cnj z2* = *rot90* (*cnj-mix* (*rot90 z1*) *z2*)
by (*cases z1*, *cases z2*) (*simp add*: *Complex-eq field-simps*)

lemma *cnj-mix-real* [*simp*]:
shows *is-real* (*cnj-mix z1 z2*)
by (*cases z1*, *cases z2*) *simp*

lemma *scalprod-real* [*simp*]:
shows *is-real* (*scalprod z1 z2*)
using *cnj-mix-real*
by *simp*

Additional properties of *cis* function.

lemma *cis-minus-pi2* [*simp*]:
shows *cis* (−*pi*/*2*) = −i
by (*simp add*: *cis-inverse*[*symmetric*])

lemma *cis-pi2-minus-x* [*simp*]:
shows *cis* (*pi*/*2* − *x*) = i ∗ *cis*(−*x*)
using *cis-divide*[*of pi*/*2 x*, *symmetric*]
using *cis-divide*[*of 0 x*, *symmetric*]
by *simp*

lemma *cis-pm-pi* [*simp*]:
shows *cis* (*x* − *pi*) = − *cis x* **and** *cis* (*x* + *pi*) = − *cis x*
by (*simp add*: *cis.ctr complex-minus*)+

lemma *cis-times-cis-opposite* [*simp*]:
shows *cis* *φ* ∗ *cis* (− *φ*) = *1*
by (*simp add*: *cis-mult*)

cis repeats only after $2k\pi$

lemma *cis-eq*:
assumes *cis a* = *cis b*
shows ∃ *k*::*int*. *a* − *b* = *2* ∗ *k* ∗ *pi*
using *assms sin-cos-eq*[*of a b*]
using *cis.sel*[*of a*] *cis.sel*[*of b*]
by (*cases cis a*, *cases cis b*) *auto*

cis is injective on $(-\pi, \pi]$.

lemma *cis-inj*:
assumes −*pi* < *α* **and** *α* ≤ *pi* **and** −*pi* < *α′* **and** *α′* ≤ *pi*
assumes *cis α* = *cis α′*
shows *α* = *α′*
using *assms*
by (*metis cis-Arg-unique sgn-cis*)

cis of an angle combined with *cis* of the opposite angle

lemma *cis-diff-cis-opposite* [*simp*]:
shows *cis* *φ* − *cis* (− *φ*) = *2* ∗ i ∗ *sin* *φ*

using *Im-express-cnj*[*of cis* φ]
by *simp*

lemma *cis-opposite-diff-cis* [*simp*]:
shows *cis* $(-\varphi)$ − *cis* $(\varphi) = -$ *2* $*$ i $*$ *sin* φ
using *cis-diff-cis-opposite*[*of* $-\varphi$]
by *simp*

lemma *cis-add-cis-opposite* [*simp*]:
shows *cis* φ + *cis* $(-\varphi) = 2 *$ *cos* φ
by (*metis cis.sel*(*1*) *cis-cnj complex-add-cnj*)

cis equal to 1 or -1

lemma *cis-one* [*simp*]:
assumes *sin* $\varphi = 0$ **and** *cos* $\varphi = 1$
shows *cis* $\varphi = 1$
using *assms*
by (*auto simp add*: *cis.ctr one-complex.code*)

lemma *cis-minus-one* [*simp*]:
assumes *sin* $\varphi = 0$ **and** *cos* $\varphi = -1$
shows *cis* $\varphi = -1$
using *assms*
by (*auto simp add*: *cis.ctr Complex-eq-neg-1*)

3.3.1 Additional properties of complex number argument

Arg of real numbers

lemma *is-real-arg1*:
assumes *Arg z* $= 0 \vee$ *Arg z* $=$ *pi*
shows *is-real z*
using *assms*
using *rcis-cmod-Arg*[*of z*] *Im-rcis*[*of cmod z Arg z*]
by *auto*

lemma *is-real-arg2*:
assumes *is-real z*
shows *Arg z* $= 0 \vee$ *Arg z* $=$ *pi*
proof (*cases z* $= 0$)
case *False*
thus *?thesis*
using *Arg-bounded*[*of z*]
by (*smt* (*verit*, *best*) *Im-sgn assms cis.simps*(*2*) *cis-Arg div-0 sin-zero-pi-iff*)
qed (*auto simp add*: *Arg-zero*)

lemma *arg-complex-of-real-positive* [*simp*]:
assumes *k* > 0
shows *Arg* (*cor k*) $= 0$
proof−
have *cos* (*Arg* (*Complex k 0*)) > 0
using *assms*
using *rcis-cmod-Arg*[*of Complex k 0*] *Re-rcis*[*of cmod* (*Complex k 0*) *Arg* (*Complex k 0*)]
using *cmod-eq-Re* **by** *force*
thus *?thesis*
using *assms is-real-arg2*[*of cor k*]
unfolding *complex-of-real-def*
by *auto*
qed

lemma *arg-complex-of-real-negative* [*simp*]:
assumes *k* < 0
shows *Arg* (*cor k*) $=$ *pi*
proof−
have *cos* (*Arg* (*Complex k 0*)) < 0
using ‹*k* < 0› *rcis-cmod-Arg*[*of Complex k 0*] *Re-rcis*[*of cmod* (*Complex k 0*) *Arg* (*Complex k 0*)]

by (*metis complex.sel*(*1*) *mult-less-0-iff norm-not-less-zero*)
thus *?thesis*
using *assms is-real-arg2*[*of cor k*]
unfolding *complex-of-real-def*
by *auto*
qed

lemma *arg-0-iff*:
shows $z \neq 0 \wedge Arg\ z = 0 \longleftrightarrow$ *is-real z* $\wedge$ *Re z* > 0
by (*smt* (*verit*, *best*) *arg-complex-of-real-negative arg-complex-of-real-positive Arg-zero complex-of-real-Re is-real-arg1 pi-gt-zero zero-complex.simps*)

lemma *arg-pi-iff*:
shows $Arg\ z = pi \longleftrightarrow$ *is-real z* $\wedge$ *Re z* < 0
by (*smt* (*verit*, *best*) *arg-complex-of-real-negative arg-complex-of-real-positive Arg-zero complex-of-real-Re is-real-arg1 pi-gt-zero zero-complex.simps*)

Arg of imaginary numbers

lemma *is-imag-arg1*:
assumes $Arg\ z = pi/2 \vee Arg\ z = -pi/2$
shows *is-imag z*
using *assms*
using *rcis-cmod-Arg*[*of z*] *Re-rcis*[*of cmod z Arg z*]
by (*metis cos-minus cos-pi-half minus-divide-left mult-eq-0-iff*)

lemma *is-imag-arg2*:
assumes *is-imag z* **and** $z \neq 0$
shows $Arg\ z = pi/2 \vee Arg\ z = -pi/2$
using *Arg-bounded assms cos-0-iff-canon cos-Arg-i-mult-zero* **by** *presburger*

lemma *arg-complex-of-real-times-i-positive* [*simp*]:
assumes $k > 0$
shows $Arg\ (cor\ k * \mathrm{i}) = pi\ /\ 2$
proof−
have $sin\ (Arg\ (Complex\ 0\ k)) > 0$
using ‹$k > 0$› *rcis-cmod-Arg*[*of Complex 0 k*] *Im-rcis*[*of cmod* (*Complex 0 k*) *Arg* (*Complex 0 k*)]
by (*smt* (*verit*, *best*) *complex.sel*(*2*) *mult-nonneg-nonpos norm-ge-zero*)
thus *?thesis*
using *assms is-imag-arg2*[*of cor k* ∗ i]
using *Arg-zero complex-of-real-i*
by *force*
qed

lemma *arg-complex-of-real-times-i-negative* [*simp*]:
assumes $k < 0$
shows $Arg\ (cor\ k * \mathrm{i}) = -\ pi\ /\ 2$
proof−
have $sin\ (Arg\ (Complex\ 0\ k)) < 0$
using ‹$k < 0$› *rcis-cmod-Arg*[*of Complex 0 k*] *Im-rcis*[*of cmod* (*Complex 0 k*) *Arg* (*Complex 0 k*)]
by (*metis complex.sel*(*2*) *mult-less-0-iff norm-not-less-zero*)
thus *?thesis*
using *assms is-imag-arg2*[*of cor k* ∗ i]
using *Arg-zero complex-of-real-i*[*of k*]
by (*smt* (*verit*, *best*) *complex.sel*(*1*) *sin-pi-half sin-zero*)
qed

lemma *arg-pi2-iff*:
shows $z \neq 0 \wedge Arg\ z = pi\ /\ 2 \longleftrightarrow$ *is-imag z* $\wedge$ *Im z* > 0
by (*smt* (*verit*, *best*) *Im-rcis Re-i-times Re-rcis arcsin-minus-1 cos-pi-half divide-minus-left mult.commute mult-cancel-right1 rcis-cmod-Arg is-imag-arg2 sin-arcsin sin-pi-half zero-less-mult-pos zero-less-norm-iff*)

lemma *arg-minus-pi2-iff*:
shows $z \neq 0 \wedge Arg\ z = -\ pi\ /\ 2 \longleftrightarrow$ *is-imag z* $\wedge$ *Im z* < 0
by (*smt* (*verit*, *best*) *arg-pi2-iff complex.expand divide-cancel-right pi-neq-zero is-imag-arg1 is-imag-arg2 zero-complex.simps*(*1*) *zero-complex.simps*(*2*))

Argument is a canonical angle

lemma *canon-ang-arg*:
shows $\lfloor Arg\ z \rfloor = Arg\ z$
using *canon-ang-id*[*of Arg z*] *Arg-bounded*
by *simp*

lemma *arg-cis*:
shows $Arg\ (cis\ \varphi) = \lfloor \varphi \rfloor$
using *cis-Arg-unique canon-ang canon-ang-cos canon-ang-sin cis.ctr sgn-cis* **by** *presburger*

Cosine and sine of *Arg*

lemma *cos-arg*:
assumes $z \neq 0$
shows $cos\ (Arg\ z) = Re\ z\ /\ cmod\ z$
by (*metis Complex.Re-sgn cis.simps*(*1*) *assms cis-Arg*)

lemma *sin-arg*:
assumes $z \neq 0$
shows $sin\ (Arg\ z) = Im\ z\ /\ cmod\ z$
by (*metis Complex.Im-sgn cis.simps*(*2*) *assms cis-Arg*)

Argument of product

lemma *cis-arg-mult*:
assumes $z1 * z2 \neq 0$
shows $cis\ (Arg\ (z1 * z2)) = cis\ (Arg\ z1 + Arg\ z2)$
by (*metis assms cis-Arg cis-mult mult-eq-0-iff sgn-mult*)

lemma *arg-mult-2kpi*:
assumes $z1 * z2 \neq 0$
shows $\exists\ k::int.\ Arg\ (z1 * z2) = Arg\ z1 + Arg\ z2 + 2*k*pi$
proof−
have $cis\ (Arg\ (z1*z2)) = cis\ (Arg\ z1 + Arg\ z2)$
by (*rule cis-arg-mult*[*OF assms*])
thus *?thesis*
using *cis-eq*[*of Arg* (*z1*∗*z2*) *Arg z1 + Arg z2*]
by (*auto simp add*: *field-simps*)
qed

lemma *arg-mult*:
assumes $z1 * z2 \neq 0$
shows $Arg(z1 * z2) = \lfloor Arg\ z1 + Arg\ z2 \rfloor$
proof−
obtain $k::int$ **where** $Arg(z1 * z2) = Arg\ z1 + Arg\ z2 + 2*k*pi$
using *arg-mult-2kpi*[*of z1 z2*]
using *assms*
by *auto*
hence $\lfloor Arg(z1 * z2) \rfloor = \lfloor Arg\ z1 + Arg\ z2 \rfloor$
using *canon-ang-eq*
by(*simp add*:*field-simps*)
thus *?thesis*
using *canon-ang-arg*[*of z1*∗*z2*]
by *auto*
qed

lemma *arg-mult-real-positive* [*simp*]:
assumes $k > 0$
shows $Arg\ (cor\ k * z) = Arg\ z$
proof (*cases z* = *0*)
case *False*
thus *?thesis*
using *arg-mult assms canon-ang-arg* **by** *force*
qed (*auto simp*: *Arg-zero*)

lemma *arg-mult-real-negative* [*simp*]:
assumes $k < 0$

shows *Arg* (*cor k* ∗ *z*) = *Arg* (−*z*)
proof (*cases z* = *0*)
case *False*
thus *?thesis*
using *assms*
by (*metis arg-mult-real-positive minus-mult-commute neg-0-less-iff-less of-real-minus minus-minus*)
qed (*auto simp*: *Arg-zero*)

lemma *arg-div-real-positive* [*simp*]:
assumes *k* > *0*
shows *Arg* (*z* / *cor k*) = *Arg z*
proof(*cases z* = *0*)
case *True*
thus *?thesis*
by *auto*
next
case *False*
thus *?thesis*
using *assms*
using *arg-mult-real-positive*[*of* 1 / *k z*]
by *auto*
qed

lemma *arg-div-real-negative* [*simp*]:
assumes *k* < *0*
shows *Arg* (*z* / *cor k*) = *Arg* (−*z*)
proof(*cases z* = *0*)
case *True*
thus *?thesis*
by *auto*
next
case *False*
thus *?thesis*
using *assms*
using *arg-mult-real-negative*[*of* 1 / *k z*]
by *auto*
qed

lemma *arg-mult-eq*:
assumes *z* ∗ *z1* ≠ *0* **and** *z* ∗ *z2* ≠ *0*
assumes *Arg* (*z* ∗ *z1*) = *Arg* (*z* ∗ *z2*)
shows *Arg z1* = *Arg z2*
by (*metis* (*no-types*, *lifting*) *arg-cis assms canon-ang-arg cis-Arg mult-eq-0-iff nonzero-mult-div-cancel-left sgn-divide*)

Argument of conjugate

lemma *arg-cnj-pi*:
assumes *Arg z* = *pi*
shows *Arg* (*cnj z*) = *pi*
using *arg-pi-iff assms* **by** *auto*

lemma *arg-cnj-not-pi*:
assumes *Arg z* ≠ *pi*
shows *Arg* (*cnj z*) = −*Arg z*
proof(*cases Arg z* = *0*)
case *True*
thus *?thesis*
using *eq-cnj-iff-real*[*of z*] *is-real-arg1*[*of z*] **by** *force*
next
case *False*
have *Arg* (*cnj z*) = *Arg z* ∨ *Arg*(*cnj z*) = −*Arg z*
using *Arg-bounded*[*of z*] *Arg-bounded*[*of cnj z*]
by (*smt* (*verit*, *best*) *arccos-cos arccos-cos2 cnj.sel*(*1*) *complex-cnj-zero-iff complex-mod-cnj cos-arg*)
moreover
have *Arg* (*cnj z*) ≠ *Arg z*
using *sin-0-iff-canon*[*of Arg* (*cnj z*)] *Arg-bounded False assms*
by (*metis complex-mod-cnj eq-cnj-iff-real is-real-arg2 rcis-cmod-Arg*)

ultimately
show *?thesis*
by *auto*
qed

Argument of reciprocal

lemma *arg-inv-not-pi*:
assumes $z \neq 0$ **and** $Arg\ z \neq pi$
shows $Arg\ (1\ /\ z) = -\ Arg\ z$
proof−
have $1/z = cnj\ z\ /\ cor\ ((cmod\ z)^2\ )$
using ‹$z \neq 0$› *complex-mult-cnj-cmod*[*of z*]
by (*auto simp add*:*field-simps*)
thus *?thesis*
using *arg-div-real-positive*[*of* $(cmod\ z)^2$ *cnj z*] ‹$z \neq 0$›
using *arg-cnj-not-pi*[*of z*] ‹$Arg\ z \neq pi$›
by *auto*
qed

lemma *arg-inv-pi*:
assumes $z \neq 0$ **and** $Arg\ z = pi$
shows $Arg\ (1\ /\ z) = pi$
proof−
have $1/z = cnj\ z\ /\ cor\ ((cmod\ z)^2\ )$
using ‹$z \neq 0$› *complex-mult-cnj-cmod*[*of z*]
by (*auto simp add*:*field-simps*)
thus *?thesis*
using *arg-div-real-positive*[*of* $(cmod\ z)^2$ *cnj z*] ‹$z \neq 0$›
using *arg-cnj-pi*[*of z*] ‹$Arg\ z = pi$›
by *auto*
qed

lemma *arg-inv-2kpi*:
assumes $z \neq 0$
shows $\exists\ k{::}int.\ Arg\ (1\ /\ z) = -\ Arg\ z + 2{*}k{*}pi$
using *arg-inv-pi*[*OF assms*]
using *arg-inv-not-pi*[*OF assms*]
by (*cases* $Arg\ z = pi$) (*rule-tac x=1* **in** *exI*, *simp*, *rule-tac x=0* **in** *exI*, *simp*)

lemma *arg-inv*:
assumes $z \neq 0$
shows $Arg\ (1\ /\ z) = \lfloor -\ Arg\ z \rfloor$
by (*metis arg-inv-not-pi arg-inv-pi assms canon-ang-arg canon-ang-uminus-pi*)

Argument of quotient

lemma *arg-div-2kpi*:
assumes $z1 \neq 0$ **and** $z2 \neq 0$
shows $\exists\ k{::}int.\ Arg\ (z1\ /\ z2) = Arg\ z1\ -\ Arg\ z2\ +\ 2{*}k{*}pi$
proof−
obtain *x1* **where** $Arg\ (z1 * (1\ /\ z2)) = Arg\ z1 + Arg\ (1\ /\ z2) + 2 * real\text{-}of\text{-}int\ x1 * pi$
using *assms arg-mult-2kpi*[*of z1* $1/z2$]
by *auto*
moreover
obtain *x2* **where** $Arg\ (1\ /\ z2) = -\ Arg\ z2 + 2 * real\text{-}of\text{-}int\ x2 * pi$
using *assms arg-inv-2kpi*[*of z2*]
by *auto*
ultimately
show *?thesis*
by (*rule-tac x=x1 + x2* **in** *exI*, *simp add*: *field-simps*)
qed

lemma *arg-div*:
assumes $z1 \neq 0$ **and** $z2 \neq 0$
shows $Arg(z1\ /\ z2) = \lfloor Arg\ z1\ -\ Arg\ z2 \rfloor$
proof−
obtain $k{::}int$ **where** $Arg(z1\ /\ z2) = Arg\ z1\ -\ Arg\ z2\ +\ 2{*}k{*}pi$

```
    using arg-div-2kpi[of z1 z2]
    using assms
    by auto
  hence canon-ang(Arg(z1 / z2)) = canon-ang(Arg z1 − Arg z2)
    using canon-ang-eq
    by(simp add:field-simps)
  thus ?thesis
    using canon-ang-arg[of z1/z2]
    by auto
qed
```

Argument of opposite

```
lemma arg-uminus:
  assumes z ≠ 0
  shows Arg (−z) = ⌊Arg z + pi⌋
  using assms
  using arg-mult[of −1 z]
  using arg-complex-of-real-negative[of −1]
  by (auto simp add: field-simps)

lemma arg-uminus-opposite-sign:
  assumes z ≠ 0
  shows Arg z > 0 ⟷ ¬ Arg (−z) > 0
proof (cases Arg z = 0)
  case True
  thus ?thesis
    using assms
    by (simp add: arg-uminus)
next
  case False
  show ?thesis
  proof (cases Arg z > 0)
    case True
    thus ?thesis
      using assms
      using Arg-bounded[of z]
      using canon-ang-plus-pi1[of Arg z]
      by (simp add: arg-uminus)
  next
    case False
    thus ?thesis
      using ‹Arg z ≠ 0›
      using assms
      using Arg-bounded[of z]
      using canon-ang-plus-pi2[of Arg z]
      by (simp add: arg-uminus)
  qed
qed
```

Sign of argument is the same as the sign of the Imaginary part

```
lemma arg-Im-sgn:
  assumes ¬ is-real z
  shows sgn (Arg z) = sgn (Im z)
proof−
  have z ≠ 0
    using assms
    by auto
  then obtain r φ where polar: z = cor r ∗ cis φ φ = Arg z r > 0
    by (smt (verit, best) cmod-cis mult-eq-0-iff norm-ge-zero of-real-0)
  hence Im z = r ∗ sin φ
    by (metis Im-mult-real Re-complex-of-real cis.simps(2) Im-complex-of-real)
  hence  Im z > 0 ⟷ sin φ > 0 Im z < 0 ⟷ sin φ < 0
    using ‹r > 0›
    using mult-pos-pos mult-nonneg-nonneg zero-less-mult-pos mult-less-cancel-left
    by (smt (verit, best))+
  moreover
```

have $\varphi \neq pi$ $\varphi \neq 0$
using ‹$\neg$ *is-real z*› *polar cis-pi*
by *force+*
hence *sin* $\varphi > 0 \longleftrightarrow \varphi > 0$ $\varphi < 0 \longleftrightarrow$ *sin* $\varphi < 0$
using ‹φ = *Arg z*› ‹$\varphi \neq 0$› ‹$\varphi \neq pi$›
using *Arg-bounded*[*of z*]
by (*smt* (*verit*, *best*) *sin-gt-zero sin-le-zero sin-pi-minus sin-0-iff-canon sin-ge-zero*)+
ultimately
show *?thesis*
using ‹φ = *Arg z*›
by *auto*
qed

3.3.2 Complex square root

definition
ccsqrt z = *rcis* (*sqrt* (*cmod z*)) (*Arg z* / *2*)

lemma *square-ccsqrt* [*simp*]:
shows $(ccsqrt\ x)^2 = x$
unfolding *ccsqrt-def*
by (*subst DeMoivre2*) (*simp add*: *rcis-cmod-Arg*)

lemma *ex-complex-sqrt*:
shows $\exists$ *s*::*complex*. *s*∗*s* = *z*
unfolding *power2-eq-square*[*symmetric*]
by (*rule-tac x*=*csqrt z* **in** *exI*) *simp*

lemma *ccsqrt*:
assumes *s* ∗ *s* = *z*
shows *s* = *ccsqrt z* ∨ *s* = −*ccsqrt z*
proof (*cases s* = *0*)
case *True*
thus *?thesis*
using *assms*
unfolding *ccsqrt-def*
by *simp*
next
case *False*
then obtain *k*::*int* **where** *cmod s* ∗ *cmod s* = *cmod z 2* ∗ *Arg s* − *Arg z* = *2*∗*k*∗*pi*
using *assms*
using *rcis-cmod-Arg*[*of z*] *rcis-cmod-Arg*[*of s*]
using *arg-mult*[*of s s*]
using *canon-ang*(*3*)[*of 2*∗*Arg s*]
by (*auto simp add*: *norm-mult arg-mult*)
have ∗: *sqrt* (*cmod z*) = *cmod s*
using ‹*cmod s* ∗ *cmod s* = *cmod z*›
by (*smt* (*verit*, *best*) *norm-not-less-zero real-sqrt-abs2*)

have ∗∗: *Arg z* / *2* = *Arg s* − *k*∗*pi*
using ‹*2* ∗ *Arg s* − *Arg z* = *2*∗*k*∗*pi*›
by *simp*

have *cis* (*Arg s* − *k*∗*pi*) = *cis* (*Arg s*) ∨ *cis* (*Arg s* − *k*∗*pi*) = −*cis* (*Arg s*)
proof (*cases even k*)
case *True*
hence *cis* (*Arg s* − *k*∗*pi*) = *cis* (*Arg s*)
by (*simp add*: *cis-def complex.corec cos-diff sin-diff*)
thus *?thesis*
by *simp*
next
case *False*
hence *cis* (*Arg s* − *k*∗*pi*) = −*cis* (*Arg s*)
by (*simp add*: *cis-def complex.corec Complex-eq cos-diff sin-diff*)
thus *?thesis*
by *simp*

```
  qed
  thus ?thesis
  proof
    assume ***: cis (Arg s − k * pi) = cis (Arg s)
    hence s = ccsqrt z
      using rcis-cmod-Arg[of s]
      unfolding ccsqrt-def rcis-def
      by (subst *, subst **, subst ***, simp)
    thus ?thesis
      by simp
  next
    assume ***: cis (Arg s − k * pi) = −cis (Arg s)
    hence s = − ccsqrt z
      using rcis-cmod-Arg[of s]
      unfolding ccsqrt-def rcis-def
      by (subst *, subst **, subst ***, simp)
    thus ?thesis
      by simp
  qed
qed

lemma null-ccsqrt [simp]:
  shows ccsqrt x = 0 ⟷ x = 0
  unfolding ccsqrt-def
  by auto

lemma ccsqrt-mult:
  shows ccsqrt (a * b) = ccsqrt a * ccsqrt b ∨
        ccsqrt (a * b) = − ccsqrt a * ccsqrt b
proof (cases a = 0 ∨ b = 0)
  case True
  thus ?thesis
    by auto
next
  case False
  obtain k::int where Arg a + Arg b − ⌊Arg a + Arg b⌋ = 2 * real-of-int k * pi
    using canon-ang(3)[of Arg a + Arg b]
    by auto
  hence *: ⌊Arg a + Arg b⌋ = Arg a + Arg b − 2 * (real-of-int k) * pi
    by (auto simp add: field-simps)

  have cis (⌊Arg a + Arg b⌋ / 2) = cis (Arg a / 2 + Arg b / 2) ∨ cis (⌊Arg a + Arg b⌋ / 2) = − cis (Arg a / 2 + Arg
b / 2)
    using cos-even-kpi[of k] cos-odd-kpi[of k]
    by ((subst *)+, (subst diff-divide-distrib)+, (subst add-divide-distrib)+)
      (cases even k, auto simp add: cis-def complex.corec Complex-eq cos-diff sin-diff)
  thus ?thesis
    using False
    unfolding ccsqrt-def
   by (smt (verit, best) arg-mult mult-minus-left mult-minus-right no-zero-divisors norm-mult rcis-def rcis-mult real-sqrt-mult)
qed

lemma csqrt-real:
  assumes is-real x
  shows (Re x ≥ 0 ∧ ccsqrt x = cor (sqrt (Re x))) ∨
        (Re x < 0 ∧ ccsqrt x = i * cor (sqrt (− (Re x))))
proof (cases x = 0)
  case True
  thus ?thesis
    by auto
next
  case False
  show ?thesis
  proof (cases Re x > 0)
    case True
    hence Arg x = 0
```

```
    using ‹is-real x›
    by (metis arg-complex-of-real-positive complex-of-real-Re)
  thus ?thesis
    using ‹Re x > 0› ‹is-real x›
    unfolding ccsqrt-def
    by (simp add: cmod-eq-Re)
 next
  case False
  hence Re x < 0
    using ‹x ≠ 0› ‹is-real x›
    using complex-eq-if-Re-eq by auto
  hence Arg x = pi
    using ‹is-real x›
    by (metis arg-complex-of-real-negative complex-of-real-Re)
  thus ?thesis
    using ‹Re x < 0› ‹is-real x›
    unfolding ccsqrt-def rcis-def
    by (simp add: cis-def complex.corec Complex-eq cmod-eq-Re)
 qed
qed
```

Rotation of complex vector to x-axis.

```
lemma is-real-rot-to-x-axis:
 assumes z ≠ 0
 shows is-real (cis (−Arg z) ∗ z)
proof (cases Arg z = pi)
 case True
 thus ?thesis
  using is-real-arg1[of z]
  by auto
next
 case False
 hence ⌊− Arg z⌋ = − Arg z
  using canon-ang-eq1[of − Arg z −Arg z]
  using Arg-bounded[of z]
  by (auto simp add: field-simps)
 hence Arg (cis (− (Arg z)) ∗ z) = 0
  using arg-mult[of cis (− (Arg z)) z] ‹z ≠ 0›
  using arg-cis[of − Arg z]
  by simp
 thus ?thesis
  using is-real-arg1[of cis (− Arg z) ∗ z]
  by auto
qed

lemma positive-rot-to-x-axis:
 assumes z ≠ 0
 shows Re (cis (−Arg z) ∗ z) > 0
 using assms
 by (smt (verit) ab-group-add-class.ab-left-minus add-eq-0-iff arg-cis arg-inv arg-inv-not-pi arg-mult arg-pi-iff cis-neq-zero
cis-pm-pi(2)
    divisors-zero is-real-rot-to-x-axis pi-canonical pi-neq-zero real-imag-0 zero-canonical)
```

Inequalities involving *cmod*.

```
lemma cmod-1-plus-mult-le:
 shows cmod (1 + z∗w) ≤ sqrt((1 + (cmod z)^2) ∗ (1 + (cmod w)^2))
proof−
 have Re ((1+z∗w)∗(1+cnj z∗cnj w)) ≤ Re (1+z∗cnj z)∗ Re (1+w∗cnj w)
 proof−
  have Re ((w − cnj z)∗cnj(w − cnj z)) ≥ 0
   by (subst complex-mult-cnj-cmod) (simp add: power2-eq-square)
  hence Re (z∗w + cnj z ∗ cnj w) ≤ Re (w∗cnj w) + Re(z∗cnj z)
   by (simp add: field-simps)
  thus ?thesis
   by (simp add: field-simps)
 qed
```

hence $(cmod\ (1 + z * w))^2 \leq (1 + (cmod\ z)^2) * (1 + (cmod\ w)^2)$
by (*subst cmod-square*)+ *simp*
thus *?thesis*
by (*metis abs-norm-cancel real-sqrt-abs real-sqrt-le-iff*)
qed

lemma *cmod-diff-ge*:
shows $cmod\ (b - c) \geq sqrt\ (1 + (cmod\ b)^2) - sqrt\ (1 + (cmod\ c)^2)$
proof−
have $(cmod\ (b - c))^2 + (1/2*Im(b*cnj\ c - c*cnj\ b))^2 \geq 0$
by *simp*
hence $(cmod\ (b - c))^2 \geq -(1/2*Im(b*cnj\ c - c*cnj\ b))^2$
by *simp*
hence $(cmod\ (b - c))^2 \geq (1/2*Re(b*cnj\ c + c*cnj\ b))^2 - Re(b*cnj\ b*c*cnj\ c)$
by (*auto simp add*: *power2-eq-square field-simps*)
hence $Re\ ((b - c)*(cnj\ b - cnj\ c)) \geq (1/2*Re(b*cnj\ c + c*cnj\ b))^2 - Re(b*cnj\ b*c*cnj\ c)$
by (*subst* (*asm*) *cmod-square*) *simp*
moreover
have $(1 + (cmod\ b)^2) * (1 + (cmod\ c)^2) = 1 + Re(b*cnj\ b) + Re(c*cnj\ c) + Re(b*cnj\ b*c*cnj\ c)$
by (*subst cmod-square*)+ (*simp add*: *field-simps power2-eq-square*)
moreover
have $(1 + Re\ (scalprod\ b\ c))^2 = 1 + 2*Re(scalprod\ b\ c) + ((Re\ (scalprod\ b\ c))^2)$
by (*subst power2-sum*) *simp*
hence $(1 + Re\ (scalprod\ b\ c))^2 = 1 + Re(b*cnj\ c + c*cnj\ b) + (1/2 * Re\ (b*cnj\ c + c*cnj\ b))^2$
by *simp*
ultimately
have $(1 + (cmod\ b)^2) * (1 + (cmod\ c)^2) \geq (1 + Re\ (scalprod\ b\ c))^2$
by (*simp add*: *field-simps*)
moreover
have $sqrt((1 + (cmod\ b)^2) * (1 + (cmod\ c)^2)) \geq 0$
by (*metis one-power2 real-sqrt-sum-squares-mult-ge-zero*)
ultimately
have $sqrt((1 + (cmod\ b)^2) * (1 + (cmod\ c)^2)) \geq 1 + Re\ (scalprod\ b\ c)$
by (*metis power2-le-imp-le real-sqrt-ge-0-iff real-sqrt-pow2-iff*)
hence $Re\ ((b - c) * (cnj\ b - cnj\ c)) \geq 1 + Re\ (c*cnj\ c) + 1 + Re\ (b*cnj\ b) - 2*sqrt((1 + (cmod\ b)^2) * (1 + (cmod\ c)^2))$
by (*simp add*: *field-simps*)
hence $*: (cmod\ (b - c))^2 \geq (sqrt\ (1 + (cmod\ b)^2) - sqrt\ (1 + (cmod\ c)^2))^2$
apply (*subst cmod-square*)+
apply (*subst* (*asm*) *cmod-square*)+
apply (*subst power2-diff*)
apply (*subst real-sqrt-pow2*, *simp*)
apply (*subst real-sqrt-pow2*, *simp*)
apply (*simp add*: *real-sqrt-mult*)
done
thus *?thesis*
proof (*cases* $sqrt\ (1 + (cmod\ b)^2) - sqrt\ (1 + (cmod\ c)^2) > 0$)
case *True*
thus *?thesis*
using *power2-le-imp-le*[*OF* $*$]
by *simp*
next
case *False*
hence $0 \geq sqrt\ (1 + (cmod\ b)^2) - sqrt\ (1 + (cmod\ c)^2)$
by (*metis less-eq-real-def linorder-neqE-linordered-idom*)
moreover
have $cmod\ (b - c) \geq 0$
by *simp*
ultimately
show *?thesis*
by (*metis add-increasing monoid-add-class.add.right-neutral*)
qed
qed

lemma *cmod-diff-le*:
shows $cmod\ (b - c) \leq sqrt\ (1 + (cmod\ b)^2) + sqrt\ (1 + (cmod\ c)^2)$

proof −
have $(cmod\ (b + c))^2 + (1/2*Im(b*cnj\ c - c*cnj\ b))^2 \geq 0$
by *simp*
hence $(cmod\ (b + c))^2 \geq -(1/2*Im(b*cnj\ c - c*cnj\ b))^2$
by *simp*
hence $(cmod\ (b + c))^2 \geq (1/2*Re(b*cnj\ c + c*cnj\ b))^2 - Re(b*cnj\ b*c*cnj\ c)$
by (*auto simp add*: *power2-eq-square field-simps*)
hence $Re\ ((b + c)*(cnj\ b + cnj\ c)) \geq (1/2*Re(b*cnj\ c + c*cnj\ b))^2 - Re(b*cnj\ b*c*cnj\ c)$
by (*subst* (*asm*) *cmod-square*) *simp*
moreover
have $(1 + (cmod\ b)^2) * (1 + (cmod\ c)^2) = 1 + Re(b*cnj\ b) + Re(c*cnj\ c) + Re(b*cnj\ b*c*cnj\ c)$
by (*subst cmod-square*)+ (*simp add*: *field-simps power2-eq-square*)
moreover
have ++: $2*Re(scalprod\ b\ c) = Re(b*cnj\ c + c*cnj\ b)$
by *simp*
have $(1 - Re\ (scalprod\ b\ c))^2 = 1 - 2*Re(scalprod\ b\ c) + ((Re\ (scalprod\ b\ c))^2)$
by (*subst power2-diff*) *simp*
hence $(1 - Re\ (scalprod\ b\ c))^2 = 1 - Re(b*cnj\ c + c*cnj\ b) + (1/2 * Re\ (b*cnj\ c + c*cnj\ b))^2$
by (*subst* ++[*symmetric*]) *simp*
ultimately
have $(1 + (cmod\ b)^2) * (1 + (cmod\ c)^2) \geq (1 - Re\ (scalprod\ b\ c))^2$
by (*simp add*: *field-simps*)
moreover
have $sqrt((1 + (cmod\ b)^2) * (1 + (cmod\ c)^2)) \geq 0$
by (*metis one-power2 real-sqrt-sum-squares-mult-ge-zero*)
ultimately
have $sqrt((1 + (cmod\ b)^2) * (1 + (cmod\ c)^2)) \geq 1 - Re\ (scalprod\ b\ c)$
by (*metis power2-le-imp-le real-sqrt-ge-0-iff real-sqrt-pow2-iff*)
hence $Re\ ((b - c) * (cnj\ b - cnj\ c)) \leq 1 + Re\ (c*cnj\ c) + 1 + Re\ (b*cnj\ b) + 2*sqrt((1 + (cmod\ b)^2) * (1 + (cmod\ c)^2))$
by (*simp add*: *field-simps*)
hence $*: (cmod\ (b - c))^2 \leq (sqrt\ (1 + (cmod\ b)^2) + sqrt\ (1 + (cmod\ c)^2))^2$
apply (*subst cmod-square*)+
apply (*subst* (*asm*) *cmod-square*)+
apply (*subst power2-sum*)
apply (*subst real-sqrt-pow2*, *simp*)
apply (*subst real-sqrt-pow2*, *simp*)
apply (*simp add*: *real-sqrt-mult*)
done
thus *?thesis*
using *power2-le-imp-le*[*OF* *]
by *simp*
qed

Definition of Euclidean distance between two complex numbers.

definition *cdist* **where**
[*simp*]: *cdist z1 z2* ≡ *cmod* (*z2* − *z1*)

Misc. properties of complex numbers.

lemma *ex-complex-to-complex* [*simp*]:
fixes *z1 z2* :: *complex*
assumes $z1 \neq 0$ **and** $z2 \neq 0$
shows $\exists k.\ k \neq 0 \wedge z2 = k * z1$
using *assms*
by (*rule-tac x*=*z2*/*z1* **in** *exI*) *simp*

lemma *ex-complex-to-one* [*simp*]:
fixes *z*::*complex*
assumes $z \neq 0$
shows $\exists k.\ k \neq 0 \wedge k * z = 1$
using *assms*
by (*rule-tac x*=1/*z* **in** *exI*) *simp*

lemma *ex-complex-to-complex2* [*simp*]:
fixes *z*::*complex*
shows $\exists k.\ k \neq 0 \wedge k * z = z$

by (*rule-tac x=1* **in** *exI*) *simp*

lemma *complex-sqrt-1*:
fixes *z*::*complex*
assumes $z \neq 0$
shows $z = 1 \ / \ z \longleftrightarrow z = 1 \vee z = -1$
using *assms*
using *nonzero-eq-divide-eq square-eq-iff*
by *fastforce*

end

3.4 Angle between two vectors

In this section we introduce different measures of angle between two vectors (represented by complex numbers).

theory *Angles*
imports *More-Transcendental Canonical-Angle More-Complex*
begin

3.4.1 Oriented angle

Oriented angle between two vectors (it is always in the interval $(-\pi, \pi]$).

definition *ang-vec* (‹∠›) **where**
[*simp*]: $\angle\ z1\ z2 \equiv \lfloor Arg\ z2 - Arg\ z1 \rfloor$

lemma *ang-vec-bounded*:
shows $-pi < \angle\ z1\ z2 \wedge \angle\ z1\ z2 \leq pi$
by (*simp add*: *canon-ang*(*1*) *canon-ang*(*2*))

lemma *ang-vec-sym*:
assumes $\angle\ z1\ z2 \neq pi$
shows $\angle\ z1\ z2 = -\angle\ z2\ z1$
using *assms*
unfolding *ang-vec-def*
using *canon-ang-uminus*[*of Arg z2 − Arg z1*]
by *simp*

lemma *ang-vec-sym-pi*:
assumes $\angle\ z1\ z2 = pi$
shows $\angle\ z1\ z2 = \angle\ z2\ z1$
using *assms*
unfolding *ang-vec-def*
using *canon-ang-uminus-pi*[*of Arg z2 − Arg z1*]
by *simp*

lemma *ang-vec-plus-pi1*:
assumes $\angle\ z1\ z2 > 0$
shows $\lfloor \angle\ z1\ z2 + pi \rfloor = \angle\ z1\ z2 - pi$
proof (*rule canon-ang-eqI*)
show $\exists\ x{::}int.\ \angle\ z1\ z2 - pi - (\angle\ z1\ z2 + pi) = 2 * real\text{-}of\text{-}int\ x * pi$
by (*rule-tac x=−1* **in** *exI*) *auto*
next
show $-pi < \angle\ z1\ z2 - pi \wedge \angle\ z1\ z2 - pi \leq pi$
using *assms*
unfolding *ang-vec-def*
using *canon-ang*(*1*)[*of Arg z2 − Arg z1*] *canon-ang*(*2*)[*of Arg z2 − Arg z1*]
by *auto*
qed

lemma *ang-vec-plus-pi2*:
assumes $\angle\ z1\ z2 \leq 0$
shows $\lfloor \angle\ z1\ z2 + pi \rfloor = \angle\ z1\ z2 + pi$
proof (*rule canon-ang-id*)
show $-pi < \angle\ z1\ z2 + pi \wedge \angle\ z1\ z2 + pi \leq pi$
using *assms*

unfolding *ang-vec-def*
using *canon-ang*(*1*)[*of Arg z2 − Arg z1*] *canon-ang*(*2*)[*of Arg z2 − Arg z1*]
by *auto*
qed

lemma *ang-vec-opposite1*:
assumes *z1 ≠ 0*
shows *∠ (−z1) z2 = ⌊∠ z1 z2 − pi⌋*
proof−
have *∠ (−z1) z2 = ⌊Arg z2 − (Arg z1 + pi)⌋*
unfolding *ang-vec-def*
using *arg-uminus*[*OF assms*]
using *canon-ang-arg*[*of z2*, *symmetric*]
using *canon-ang-diff*[*of Arg z2 Arg z1 + pi*, *symmetric*]
by *simp*
moreover
have *⌊∠ z1 z2 − pi⌋ = ⌊Arg z2 − Arg z1 − pi⌋*
using *canon-ang-id*[*of pi*, *symmetric*]
using *canon-ang-diff*[*of Arg z2 − Arg z1 pi*, *symmetric*]
by *simp-all*
ultimately
show *?thesis*
by (*simp add*: *field-simps*)
qed

lemma *ang-vec-opposite2*:
assumes *z2 ≠ 0*
shows *∠ z1 (−z2) = ⌊∠ z1 z2 + pi⌋*
unfolding *ang-vec-def*
using *arg-mult*[*of −1 z2*] *assms*
using *arg-complex-of-real-negative*[*of −1*]
using *canon-ang-diff*[*of Arg (−1) + Arg z2 Arg z1*, *symmetric*]
using *canon-ang-sum*[*of Arg z2 − Arg z1 pi*, *symmetric*]
using *canon-ang-id*[*of pi*] *canon-ang-arg*[*of z1*]
by (*auto simp*: *algebra-simps*)

lemma *ang-vec-opposite-opposite*:
assumes *z1 ≠ 0* **and** *z2 ≠ 0*
shows *∠ (−z1) (−z2) = ∠ z1 z2*
proof−
have *∠ (−z1) (−z2) = ⌊⌊∠ z1 z2 + pi⌋ − ⌊pi⌋⌋*
using *ang-vec-opposite1*[*OF assms*(*1*)]
using *ang-vec-opposite2*[*OF assms*(*2*)]
using *canon-ang-id*[*of pi*, *symmetric*]
by *simp-all*
also have *... = ⌊∠ z1 z2⌋*
by (*subst canon-ang-diff*[*symmetric*], *simp*)
finally
show *?thesis*
by (*metis ang-vec-def canon-ang*(*1*) *canon-ang*(*2*) *canon-ang-id*)
qed

lemma *ang-vec-opposite-opposite′*:
assumes *z1 ≠ z* **and** *z2 ≠ z*
shows *∠ (z − z1) (z − z2) = ∠ (z1 − z) (z2 − z)*
using *ang-vec-opposite-opposite*[*of z − z1 z − z2*] *assms*
by (*simp add*: *field-simps del*: *ang-vec-def*)

Cosine, scalar product and the law of cosines

lemma *cos-cmod-scalprod*:
shows *cmod z1 ∗ cmod z2 ∗ (cos (∠ z1 z2)) = Re (scalprod z1 z2)*
proof (*cases z1 = 0 ∨ z2 = 0*)
case *True*
thus *?thesis*
by *auto*

next
case *False*
thus *?thesis*
by (*simp add*: *cos-diff cos-arg sin-arg field-simps*)
qed

lemma *cos0-scalprod0*:
assumes *z1* ≠ *0* **and** *z2* ≠ *0*
shows *cos* (∠ *z1 z2*) = *0* ⟷ *scalprod z1 z2* = *0*
using *assms*
using *cnj-mix-real*[*of z1 z2*]
using *cos-cmod-scalprod*[*of z1 z2*]
by (*auto simp add*: *complex-eq-if-Re-eq*)

lemma *ortho-scalprod0*:
assumes *z1* ≠ *0* **and** *z2* ≠ *0*
shows ∠ *z1 z2* = *pi*/*2* ∨ ∠ *z1 z2* = −*pi*/*2* ⟷ *scalprod z1 z2* = *0*
using *cos0-scalprod0*[*OF assms*]
using *ang-vec-bounded*[*of z1 z2*]
using *cos-0-iff-canon*[*of* ∠ *z1 z2*]
by (*metis cos-minus cos-pi-half divide-minus-left*)

lemma *law-of-cosines*:
shows (*cdist B C*)^2 = (*cdist A C*)^2 + (*cdist A B*)^2 − *2*∗(*cdist A C*)∗(*cdist A B*)∗(*cos* (∠ (*C*−*A*) (*B*−*A*)))
proof−
let *?a* = *C*−*B* **and** *?b* = *C*−*A* **and** *?c* = *B*−*A*
have *?a* = *?b* − *?c*
by *simp*
hence (*cmod ?a*)^2 = (*cmod* (*?b* − *?c*))^2
by *metis*
also have ... = *Re* (*scalprod* (*?b*−*?c*) (*?b*−*?c*))
by (*simp add*: *cmod-square*)
also have ... = (*cmod ?b*)^2 + (*cmod ?c*)^2 − *2*∗*Re* (*scalprod ?b ?c*)
by (*simp add*: *cmod-square field-simps*)
finally
show *?thesis*
using *cos-cmod-scalprod*[*of ?b ?c*]
by *simp*
qed

3.4.2 Unoriented angle

Convex unoriented angle between two vectors (it is always in the interval [0, *pi*]).

definition *ang-vec-c* (‹∠*c*›) **where**
[*simp*]:∠*c z1 z2* ≡ *abs* (∠ *z1 z2*)

lemma *ang-vec-c-sym*:
shows ∠*c z1 z2* = ∠*c z2 z1*
unfolding *ang-vec-c-def*
using *ang-vec-sym-pi*[*of z1 z2*] *ang-vec-sym*[*of z1 z2*]
by (*cases* ∠ *z1 z2* = *pi*) *auto*

lemma *ang-vec-c-bounded*: *0* ≤ ∠*c z1 z2* ∧ ∠*c z1 z2* ≤ *pi*
using *canon-ang*(*1*)[*of Arg z2* − *Arg z1*] *canon-ang*(*2*)[*of Arg z2* − *Arg z1*]
by *auto*

Cosine and scalar product

lemma *cos-c-*: *cos* (∠*c z1 z2*) = *cos* (∠ *z1 z2*)
unfolding *ang-vec-c-def*
by (*smt* (*verit*) *cos-minus*)

lemma *ortho-c-scalprod0*:
assumes *z1* ≠ *0* **and** *z2* ≠ *0*
shows ∠*c z1 z2* = *pi*/*2* ⟷ *scalprod z1 z2* = *0*
proof−

```
  have ∠ z1 z2 = pi / 2 ∨ ∠ z1 z2 = − pi / 2 ⟷ ∠c z1 z2 = pi/2
    unfolding ang-vec-c-def
    using arctan
    by force
  thus ?thesis
    using ortho-scalprod0[OF assms]
    by simp
qed
```

3.4.3 Acute angle

Acute or right angle (non-obtuse) between two vectors (it is always in the interval $[0, \frac{\pi}{2}]$). We will use this to measure angle between two circles, since it can always be acute (or right).

```
definition acute-ang where
  [simp]: acute-ang α = (if α > pi / 2 then pi − α else α)

definition ang-vec-a (‹∠a›) where
  [simp]: ∠a z1 z2 ≡ acute-ang (∠c z1 z2)

lemma ang-vec-a-sym:
  ∠a z1 z2 = ∠a z2 z1
  unfolding ang-vec-a-def
  using ang-vec-c-sym
  by auto

lemma ang-vec-a-opposite2:
  ∠a z1 z2 = ∠a z1 (−z2)
proof(cases z2 = 0)
  case True
  thus ?thesis
    by (metis minus-zero)
next
  case False
  thus ?thesis
  proof(cases ∠ z1 z2 < −pi / 2)
    case True
    hence ∠ z1 z2 < 0
      using pi-not-less-zero
      by linarith
    have ∠a z1 z2 = pi + ∠ z1 z2
      using True ‹∠ z1 z2 < 0›
      unfolding ang-vec-a-def ang-vec-c-def ang-vec-a-def abs-real-def
      by auto
    moreover
    have ∠a z1 (−z2) = pi + ∠ z1 z2
      unfolding ang-vec-a-def ang-vec-c-def abs-real-def
      using canon-ang(1)[of Arg z2 − Arg z1] canon-ang(2)[of Arg z2 − Arg z1]
      using ang-vec-plus-pi2[of z1 z2] True ‹∠ z1 z2 < 0› ‹z2 ≠ 0›
      using ang-vec-opposite2[of z2 z1]
      by auto
    ultimately
    show ?thesis
      by auto
  next
    case False
    show ?thesis
    proof (cases ∠ z1 z2 ≤ 0)
      case True
      have ∠a z1 z2 = − ∠ z1 z2
        using ‹¬ ∠ z1 z2 < − pi / 2› True
        unfolding ang-vec-a-def ang-vec-c-def ang-vec-a-def abs-real-def
        by auto
      moreover
      have ∠a z1 (−z2) = − ∠ z1 z2
        using ‹¬ ∠ z1 z2 < − pi / 2› True
        unfolding ang-vec-a-def ang-vec-c-def abs-real-def
```

```
      using ang-vec-plus-pi2[of z1 z2]
      using canon-ang(1)[of Arg z2 − Arg z1] canon-ang(2)[of Arg z2 − Arg z1]
      using ‹z2 ≠ 0› ang-vec-opposite2[of z2 z1]
      by auto
    ultimately
    show ?thesis
      by simp
  next
    case False
    show ?thesis
    proof (cases ∠ z1 z2 < pi / 2)
      case True
      have ∠a z1 z2 = ∠ z1 z2
        using ‹¬ ∠ z1 z2 ≤ 0› True
        unfolding ang-vec-a-def ang-vec-c-def ang-vec-a-def abs-real-def
        by auto
      moreover
      have ∠a z1 (−z2) = ∠ z1 z2
        using ‹¬ ∠ z1 z2 ≤ 0› True
        unfolding ang-vec-a-def ang-vec-c-def abs-real-def
        using ang-vec-plus-pi1[of z1 z2]
        using canon-ang(1)[of Arg z2 − Arg z1] canon-ang(2)[of Arg z2 − Arg z1]
        using ‹z2 ≠ 0› ang-vec-opposite2[of z2 z1]
        by auto
      ultimately
      show ?thesis
        by simp
    next
      case False
      have ∠ z1 z2 > 0
        using False
        by (metis less-linear less-trans pi-half-gt-zero)
      have ∠a z1 z2 = pi − ∠ z1 z2
        using False ‹∠ z1 z2 > 0›
        unfolding ang-vec-a-def ang-vec-c-def ang-vec-a-def abs-real-def
        by auto
      moreover
      have ∠a z1 (−z2) = pi − ∠ z1 z2
        unfolding ang-vec-a-def ang-vec-c-def abs-real-def
        using False ‹∠ z1 z2 > 0›
        using ang-vec-plus-pi1[of z1 z2]
        using canon-ang(1)[of Arg z2 − Arg z1] canon-ang(2)[of Arg z2 − Arg z1]
        using ‹z2 ≠ 0› ang-vec-opposite2[of z2 z1]
        by auto
      ultimately
      show ?thesis
        by auto
    qed
  qed
qed

lemma ang-vec-a-opposite1:
  shows ∠a z1 z2 = ∠a (−z1) z2
  using ang-vec-a-sym[of −z1 z2] ang-vec-a-opposite2[of z2 z1] ang-vec-a-sym[of z2 z1]
  by auto

lemma ang-vec-a-scale1:
  assumes k ≠ 0
  shows ∠a (cor k ∗ z1) z2 = ∠a z1 z2
proof (cases k > 0)
  case True
  thus ?thesis
    unfolding ang-vec-a-def ang-vec-c-def ang-vec-def
    using arg-mult-real-positive[of k z1]
    by auto
```

```
next
  case False
  hence k < 0
    using assms
    by auto
  thus ?thesis
    using arg-mult-real-negative[of k z1]
    using ang-vec-a-opposite1[of z1 z2]
    unfolding ang-vec-a-def ang-vec-c-def ang-vec-def
    by simp
qed

lemma ang-vec-a-scale2:
  assumes k ≠ 0
  shows ∠a z1 (cor k * z2) = ∠a z1 z2
  using ang-vec-a-sym[of z1 complex-of-real k * z2]
  using ang-vec-a-scale1[OF assms, of z2 z1]
  using ang-vec-a-sym[of z1 z2]
  by auto

lemma ang-vec-a-scale:
  assumes k1 ≠ 0 and k2 ≠ 0
  shows ∠a (cor k1 * z1) (cor k2 * z2) = ∠a z1 z2
  using ang-vec-a-scale1[OF assms(1)] ang-vec-a-scale2[OF assms(2)]
  by auto

lemma ang-a-cnj-cnj:
  shows ∠a z1 z2 = ∠a (cnj z1) (cnj z2)
unfolding ang-vec-a-def ang-vec-c-def ang-vec-def
proof(cases Arg z1 ≠ pi ∧ Arg z2 ≠ pi)
  case True
  thus acute-ang |⌊Arg z2 − Arg z1⌋| = acute-ang |⌊Arg (cnj z2) − Arg (cnj z1)⌋|
    using arg-cnj-not-pi[of z1] arg-cnj-not-pi[of z2]
    apply (auto simp del:acute-ang-def)
    proof(cases ⌊Arg z2 − Arg z1⌋ = pi)
      case True
      thus acute-ang |⌊Arg z2 − Arg z1⌋| = acute-ang |⌊Arg z1 − Arg z2⌋|
        using  canon-ang-uminus-pi[of Arg z2 − Arg z1]
        by (auto simp add:field-simps)
    next
      case False
      thus acute-ang |⌊Arg z2 − Arg z1⌋| = acute-ang |⌊Arg z1 − Arg z2⌋|
        using  canon-ang-uminus[of Arg z2 − Arg z1]
        by (auto simp add:field-simps)
    qed
  next
    case False
    thus acute-ang |⌊Arg z2 − Arg z1⌋| = acute-ang |⌊Arg (cnj z2) − Arg (cnj z1)⌋|
    proof(cases Arg z1 = pi)
      case False
      hence Arg z2 = pi
        using ‹ ¬ (Arg z1 ≠ pi ∧ Arg z2 ≠ pi)›
        by auto
      thus ?thesis
        using False
        using arg-cnj-not-pi[of z1] arg-cnj-pi[of z2]
        apply (auto simp del:acute-ang-def)
      proof(cases Arg z1 > 0)
        case True
        hence −Arg z1 ≤ 0
          by auto
        thus acute-ang |⌊pi − Arg z1⌋| = acute-ang |⌊pi + Arg z1⌋|
          using True canon-ang-plus-pi1[of Arg z1]
          using Arg-bounded[of z1] canon-ang-plus-pi2[of −Arg z1]
          by (auto simp add:field-simps)
      next
```

```
        case False
        hence −Arg z1 ≥ 0
          by simp
        thus acute-ang ||⌊pi − Arg z1⌋|| = acute-ang ||⌊pi + Arg z1⌋||
        proof(cases Arg z1 = 0)
          case True
          thus ?thesis
            by (auto simp del:acute-ang-def)
        next
          case False
          hence −Arg z1 > 0
            using ‹−Arg z1 ≥ 0›
            by auto
          thus ?thesis
          using False canon-ang-plus-pi1[of −Arg z1]
          using Arg-bounded[of z1] canon-ang-plus-pi2[of Arg z1]
          by (auto simp add:field-simps)
      qed
    qed
  next
    case True
    thus ?thesis
      using arg-cnj-pi[of z1]
      apply (auto simp del:acute-ang-def)
    proof(cases Arg z2 = pi)
      case True
      thus acute-ang ||⌊Arg z2 − pi⌋|| = acute-ang ||⌊Arg (cnj z2) − pi⌋||
        using arg-cnj-pi[of z2]
        by auto
    next
      case False
      thus acute-ang ||⌊Arg z2 − pi⌋|| = acute-ang ||⌊Arg (cnj z2) − pi⌋||
        using arg-cnj-not-pi[of z2]
        apply (auto simp del:acute-ang-def)
      proof(cases Arg z2 > 0)
        case True
        hence −Arg z2 ≤ 0
          by auto
        thus acute-ang ||⌊Arg z2 − pi⌋|| = acute-ang ||⌊− Arg z2 − pi⌋||
          using True canon-ang-minus-pi1[of Arg z2]
          using Arg-bounded[of z2] canon-ang-minus-pi2[of −Arg z2]
          by (auto simp add: field-simps)
      next
        case False
        hence −Arg z2 ≥ 0
          by simp
        thus acute-ang ||⌊Arg z2 − pi⌋|| = acute-ang ||⌊− Arg z2 − pi⌋||
        proof(cases Arg z2 = 0)
          case True
          thus ?thesis
            by (auto simp del:acute-ang-def)
        next
          case False
          hence −Arg z2 > 0
            using ‹−Arg z2 ≥ 0›
            by auto
          thus ?thesis
          using False canon-ang-minus-pi1[of −Arg z2]
          using Arg-bounded[of z2] canon-ang-minus-pi2[of Arg z2]
          by (auto simp add:field-simps)
      qed
    qed
  qed
 qed
qed
```

Cosine and scalar product

lemma *ortho-a-scalprod0*:
  assumes *z1* ≠ *0* **and** *z2* ≠ *0*
  shows ∠*a z1 z2* = *pi*/*2* ⟷ *scalprod z1 z2* = *0*
  unfolding *ang-vec-a-def*
  using *assms ortho-c-scalprod0*[*of z1 z2*]
  by *auto*

declare *ang-vec-c-def*[*simp del*]

lemma *cos-a-c*: *cos* (∠*a z1 z2*) = *abs* (*cos* (∠*c z1 z2*))
proof−
  have *0* ≤ ∠*c z1 z2* ∠*c z1 z2* ≤ *pi*
    using *ang-vec-c-bounded*[*of z1 z2*]
    by *auto*
  show *?thesis*
  proof (*cases* ∠*c z1 z2* = *pi*/*2*)
    case *True*
    thus *?thesis*
      unfolding *ang-vec-a-def acute-ang-def*
      by (*smt* (*verit*) *cos-pi-half pi-def pi-half*)
  next
    case *False*
    show *?thesis*
    proof (*cases* ∠*c z1 z2* < *pi* / *2*)
      case *True*
      thus *?thesis*
        using ‹*0* ≤ ∠*c z1 z2*›
        using *cos-gt-zero-pi*[*of* ∠*c z1 z2*]
        unfolding *ang-vec-a-def*
        by *simp*
    next
      case *False*
      hence ∠*c z1 z2* > *pi*/*2*
        using ‹∠*c z1 z2* ≠ *pi*/*2*›
        by *simp*
      hence *cos* (∠*c z1 z2*) < *0*
        using ‹∠*c z1 z2* ≤ *pi*›
        using *cos-lt-zero-on-pi2-pi*[*of* ∠*c z1 z2*]
        by *simp*
      thus *?thesis*
        using ‹∠*c z1 z2* > *pi*/*2*›
        unfolding *ang-vec-a-def*
        by *simp*
    qed
  qed
qed

end

3.5 Library Aditions for Set Cardinality

In this section some additional simple lemmas about set cardinality are proved.

theory *More-Set*
imports *Main*
begin

Every infinite set has at least two different elements

lemma *infinite-contains-2-elems*:
  assumes *infinite A*
  shows ∃ *x y*. *x* ≠ *y* ∧ *x* ∈ *A* ∧ *y* ∈ *A*
  by (*metis assms finite.simps is-singletonI' is-singleton-def*)

Every infinite set has at least three different elements

lemma *infinite-contains-3-elems*:

assumes *infinite A*
shows $\exists\ x\ y\ z.\ x \neq y \wedge x \neq z \wedge y \neq z \wedge x \in A \wedge y \in A \wedge z \in A$
by (*metis Diff-iff assms infinite-contains-2-elems infinite-remove insertI1*)

Every set with cardinality greater than 1 has at least two different elements

lemma *card-geq-2-iff-contains-2-elems*:
shows $card\ A \geq 2 \longleftrightarrow finite\ A \wedge (\exists\ x\ y.\ x \neq y \wedge x \in A \wedge y \in A)$
proof (*intro iffI conjI*)
assume $*$: $finite\ A \wedge (\exists\ x\ y.\ x \neq y \wedge x \in A \wedge y \in A)$
thus $card\ A \geq 2$
by (*metis card-0-eq card-Suc-eq empty-iff leI less-2-cases singletonD*)
next
assume $*$: $2 \leq card\ A$
then show *finite A*
using *card.infinite* **by** *force*
show $\exists\ x\ y.\ x \neq y \wedge x \in A \wedge y \in A$
by (*meson* $*$ *card-2-iff' in-mono obtain-subset-with-card-n*)
qed

Set cardinality is at least 3 if and only if it contains three different elements

lemma *card-geq-3-iff-contains-3-elems*:
shows $card\ A \geq 3 \longleftrightarrow finite\ A \wedge (\exists\ x\ y\ z.\ x \neq y \wedge x \neq z \wedge y \neq z \wedge x \in A \wedge y \in A \wedge z \in A)$
proof (*intro iffI conjI*)
assume $*$: $card\ A \geq 3$
then show *finite A*
using *card.infinite* **by** *force*
show $\exists\ x\ y\ z.\ x \neq y \wedge x \neq z \wedge y \neq z \wedge x \in A \wedge y \in A \wedge z \in A$
by (*smt* (*verit*, *best*) $*$ *card-2-iff' card-geq-2-iff-contains-2-elems le-cases3 not-less-eq-eq numeral-2-eq-2 numeral-3-eq-3*)
next
assume $*$: $finite\ A \wedge (\exists\ x\ y\ z.\ x \neq y \wedge x \neq z \wedge y \neq z \wedge x \in A \wedge y \in A \wedge z \in A)$
thus $card\ A \geq 3$
by (*metis One-nat-def Suc-le-eq card-2-iff' card-le-Suc0-iff-eq leI numeral-3-eq-3 one-add-one order-class.order.eq-iff plus-1-eq-Suc*)
qed

Set cardinality of A is equal to 2 if and only if A=x, y for two different elements x and y

lemma *card-eq-2-iff-doubleton*: $card\ A = 2 \longleftrightarrow (\exists\ x\ y.\ x \neq y \wedge A = \{x, y\})$
using *card-geq-2-iff-contains-2-elems*[*of A*]
using *card-geq-3-iff-contains-3-elems*[*of A*]
by *auto* (*rule-tac x=x* **in** *exI*, *rule-tac x=y* **in** *exI*, *auto*)

lemma *card-eq-2-doubleton*:
assumes $card\ A = 2$ **and** $x \neq y$ **and** $x \in A$ **and** $y \in A$
shows $A = \{x, y\}$
using *assms card-eq-2-iff-doubleton*[*of A*]
by *auto*

Bijections map singleton to singleton sets

lemma *bij-image-singleton*:
shows $[\![f\ `\ A = \{b\};\ f\ a = b;\ bij\ f]\!] \Longrightarrow A = \{a\}$
by (*metis bij-betw-def image-empty image-insert inj-image-eq-iff*)

end

3.6 Systems of linear equations

In this section some simple properties of systems of linear equations with two or three unknowns are derived. Existence and uniqueness of solutions of regular and singular homogenous and non-homogenous systems is characterized.

theory *Linear-Systems*
imports *Main*
begin

Determinant of 2x2 matrix

definition *det2* :: ('*a*::*field*) ⇒ '*a* ⇒ '*a* ⇒ '*a* ⇒ '*a* **where**
[*simp*]: *det2 a11 a12 a21 a22* ≡ *a11*∗*a22* − *a12*∗*a21*

Regular homogenous system has only trivial solution

lemma *regular-homogenous-system*:
fixes *a11 a12 a21 a22 x1 x2* :: '*a*::*field*
assumes *det2 a11 a12 a21 a22* ≠ *0*
assumes *a11*∗*x1* + *a12*∗*x2* = *0* **and**
a21∗*x1* + *a22*∗*x2* = *0*
shows *x1* = *0* ∧ *x2* = *0*
proof (*cases a11* = *0*)
case *True*
with *assms*(*1*) **have** *a12* ≠ *0 a21* ≠ *0*
by *auto*
thus *?thesis*
using ‹*a11* = *0*› *assms*(*2*) *assms*(*3*)
by *auto*
next
case *False*
hence *x1* = − *a12*∗*x2* / *a11*
using *assms*(*2*)
by (*metis eq-neg-iff-add-eq-0 mult-minus-left nonzero-mult-div-cancel-left*)
hence *a21* ∗ (− *a12* ∗ *x2* / *a11*) + *a22* ∗ *x2* = *0*
using *assms*(*3*)
by *simp*
hence *a21* ∗ (− *a12* ∗ *x2*) + *a22* ∗ *x2* ∗ *a11* = *0*
using ‹*a11* ≠ *0*›
by *auto*
hence (*a11*∗*a22* − *a12*∗*a21*)∗*x2* = *0*
by (*simp add*: *field-simps*)
thus *?thesis*
using *assms*(*1*) *assms*(*2*) ‹*a11* ≠ *0*›
by *auto*
qed

Regular system has a unique solution

lemma *regular-system*:
fixes *a11 a12 a21 a22 b1 b2* :: '*a*::*field*
assumes *det2 a11 a12 a21 a22* ≠ *0*
shows ∃! *x*. *a11*∗(*fst x*) + *a12*∗(*snd x*) = *b1* ∧
a21∗(*fst x*) + *a22*∗(*snd x*) = *b2*
proof
let *?d* = *a11*∗*a22* − *a12*∗*a21* **and** *?d1* = *b1*∗*a22* − *b2*∗*a12* **and** *?d2* = *b2*∗*a11* − *b1*∗*a21*
let *?x* = (*?d1* / *?d*, *?d2* / *?d*)
have *a11* ∗ *?d1* + *a12* ∗ *?d2* = *b1*∗*?d a21* ∗ *?d1* + *a22* ∗ *?d2* = *b2*∗*?d*
by (*auto simp add*: *field-simps*)
thus *a11* ∗ *fst ?x* + *a12* ∗ *snd ?x* = *b1* ∧ *a21* ∗ *fst ?x* + *a22* ∗ *snd ?x* = *b2*
using *assms*
by (*metis* (*opaque-lifting*, *no-types*) *det2-def add-divide-distrib eq-divide-imp fst-eqD snd-eqD times-divide-eq-right*)

fix *x*′
assume *a11* ∗ *fst x*′ + *a12* ∗ *snd x*′ = *b1* ∧ *a21* ∗ *fst x*′ + *a22* ∗ *snd x*′ = *b2*
with ‹*a11* ∗ *fst ?x* + *a12* ∗ *snd ?x* = *b1* ∧ *a21* ∗ *fst ?x* + *a22* ∗ *snd ?x* = *b2*›
have *a11* ∗ (*fst x*′ − *fst ?x*) + *a12* ∗ (*snd x*′ − *snd ?x*) = *0* ∧ *a21* ∗ (*fst x*′ − *fst ?x*) + *a22* ∗ (*snd x*′ − *snd ?x*) = *0*
by (*auto simp add*: *field-simps*)
thus *x*′ = *?x*
using *regular-homogenous-system*[*OF assms*, *of fst x*′ − *fst ?x snd x*′ − *snd ?x*]
by (*cases x*′) *auto*
qed

Singular system does not have a unique solution

lemma *singular-system*:
fixes *a11 a12 a21 a22* ::'*a*::*field*
assumes *det2 a11 a12 a21 a22* = *0* **and** *a11* ≠ *0* ∨ *a12* ≠ *0*
assumes *x0*: *a11*∗*fst x0* + *a12*∗*snd x0* = *b1*

```
        a21*fst x0 + a22*snd x0 = b2
  assumes x: a11*fst x + a12*snd x = b1
  shows a21*fst x + a22*snd x = b2
proof (cases a11 = 0)
  case True
  with assms have a21 = 0 a12 ≠ 0
    by auto
  let ?k = a22 / a12
  have b2 = ?k * b1
    using x0 ‹a11 = 0› ‹a21 = 0› ‹a12 ≠ 0›
    by auto
  thus ?thesis
    using ‹a11 = 0› ‹a21 = 0› ‹a12 ≠ 0› x
    by auto
next
  case False
  let ?k = a21 / a11
  from x
  have ?k * a11 * fst x + ?k * a12 * snd x = ?k * b1
    using ‹a11 ≠ 0›
    by (auto simp add: field-simps)
  moreover
  have a21 = ?k * a11 a22 = ?k * a12 b2 = ?k * b1
    using assms(1) x0 ‹a11 ≠ 0›
    by (auto simp add: field-simps)
  ultimately
  show ?thesis
    by auto
qed
```

All solutions of a homogenous system of 2 equations with 3 unknows are proportional

```
lemma linear-system-homogenous-3-2:
  fixes a11 a12 a13 a21 a22 a23 x1 y1 z1 x2 y2 z2 :: 'a::field
  assumes f1 = (λ x y z. a11 * x + a12 * y + a13 * z)
  assumes f2 = (λ x y z. a21 * x + a22 * y + a23 * z)
  assumes f1 x1 y1 z1 = 0 and f2 x1 y1 z1 = 0
  assumes f1 x2 y2 z2 = 0 and f2 x2 y2 z2 = 0
  assumes x2 ≠ 0 ∨ y2 ≠ 0 ∨ z2 ≠ 0
  assumes det2 a11 a12 a21 a22 ≠ 0 ∨ det2 a11 a13 a21 a23 ≠ 0 ∨ det2 a12 a13 a22 a23 ≠ 0
  shows ∃ k. x1 = k * x2 ∧ y1 = k * y2 ∧ z1 = k * z2
proof−
  let ?Dz = det2 a11 a12 a21 a22
  let ?Dy = det2 a11 a13 a21 a23
  let ?Dx = det2 a12 a13 a22 a23

  have a21 * (f1 x1 y1 z1) − a11 * (f2 x1 y1 z1) = 0
    using assms
    by simp
  hence yz1: ?Dz*y1 + ?Dy*z1 = 0
    using assms
    by (simp add: field-simps)

  have a21 * (f1 x2 y2 z2) − a11 * (f2 x2 y2 z2) = 0
    using assms
    by simp
  hence yz2: ?Dz*y2 + ?Dy*z2 = 0
    using assms
    by (simp add: field-simps)

  have a22 * (f1 x1 y1 z1) − a12 * (f2 x1 y1 z1) = 0
    using assms
    by simp
  hence xz1: −?Dz*x1 + ?Dx*z1 = 0
    using assms
    by (simp add: field-simps)
```

```
have a22 * (f1 x2 y2 z2) - a12 * (f2 x2 y2 z2) = 0
  using assms
  by simp
hence xz2: -?Dz*x2 + ?Dx*z2 = 0
  using assms
  by (simp add: field-simps)

have a23 * (f1 x1 y1 z1) - a13 * (f2 x1 y1 z1) = 0
  using assms
  by simp
hence xy1: ?Dy*x1 + ?Dx*y1 = 0
  using assms
  by (simp add: field-simps)

have a23 * (f1 x2 y2 z2) - a13 * (f2 x2 y2 z2) = 0
  using assms
  by simp
hence xy2: ?Dy*x2 + ?Dx*y2 = 0
  using assms
  by (simp add: field-simps)

show ?thesis
  using ‹?Dz ≠ 0 ∨ ?Dy ≠ 0 ∨ ?Dx ≠ 0›
proof safe
  assume ?Dz ≠ 0

  hence *:
    x2 = (?Dx / ?Dz) * z2 y2 = - (?Dy / ?Dz) * z2
    x1 = (?Dx / ?Dz) * z1 y1 = - (?Dy / ?Dz) * z1
    using xy2 xz2 xy1 xz1 yz1 yz2
    by (simp-all add: field-simps)

  hence z2 ≠ 0
    using ‹x2 ≠ 0 ∨ y2 ≠ 0 ∨ z2 ≠ 0›
    by auto

  thus ?thesis
    using * ‹?Dz ≠ 0›
    by (rule-tac x=z1/z2 in exI) auto
next
  assume ?Dy ≠ 0
  hence *:
    x2 = - (?Dx / ?Dy) * y2 z2 = - (?Dz / ?Dy) * y2
    x1 = - (?Dx / ?Dy) * y1 z1 = - (?Dz / ?Dy) * y1
    using xy2 xz2 xy1 xz1 yz1 yz2
    by (simp-all add: field-simps)

  hence y2 ≠ 0
    using ‹x2 ≠ 0 ∨ y2 ≠ 0 ∨ z2 ≠ 0›
    by auto

  thus ?thesis
    using * ‹?Dy ≠ 0›
    by (rule-tac x=y1/y2 in exI) auto
next
  assume ?Dx ≠ 0
  hence *:
    y2 = - (?Dy / ?Dx) * x2 z2 = (?Dz / ?Dx) * x2
    y1 = - (?Dy / ?Dx) * x1 z1 = (?Dz / ?Dx) * x1
    using xy2 xz2 xy1 xz1 yz1 yz2
    by (simp-all add: field-simps)

  hence x2 ≠ 0
    using ‹x2 ≠ 0 ∨ y2 ≠ 0 ∨ z2 ≠ 0›
    by auto
```

```
    thus ?thesis
      using * ‹?Dx ≠ 0›
      by (rule-tac x=x1/x2 in exI) auto
  qed
qed

end
```

3.7 Quadratic equations

In this section some simple properties of quadratic equations and their roots are derived. Quadratic equations over reals and over complex numbers, but also systems of quadratic equations and systems of quadratic and linear equations are analysed.

```
theory Quadratic
  imports More-Complex HOL−Library.Quadratic-Discriminant
begin
```

3.7.1 Real quadratic equations, Viette rules

```
lemma viette2-monic:
  fixes b c ξ1 ξ2 :: real
  assumes b^2 − 4*c ≥ 0 and ξ1^2 + b*ξ1 + c = 0 and ξ2^2 + b*ξ2 + c = 0 and ξ1 ≠ ξ2
  shows ξ1*ξ2 = c
  using assms
  by algebra

lemma viette2:
  fixes a b c ξ1 ξ2 :: real
  assumes a ≠ 0 and b^2 − 4*a*c ≥ 0 and a*ξ1^2 + b*ξ1 + c = 0 and a*ξ2^2 + b*ξ2 + c = 0 and ξ1 ≠ ξ2
  shows ξ1*ξ2 = c/a
proof (rule viette2-monic[of b/a c/a ξ1 ξ2])
  have (b / a)^2 − 4 * (c / a) = (b^2 − 4*a*c) / a^2
    using ‹a ≠ 0›
    by (auto simp add: power2-eq-square field-simps)
  thus 0 ≤ (b / a)^2 − 4 * (c / a)
    using ‹b^2 − 4*a*c ≥ 0›
    by simp
next
  show ξ1^2 + b / a * ξ1 + c / a = 0 ξ2^2 + b / a * ξ2 + c / a = 0
    using assms
    by (auto simp add: power2-eq-square field-simps)
next
  show ξ1 ≠ ξ2
    by fact
qed

lemma viette2'-monic:
  fixes b c ξ :: real
  assumes b^2 − 4*c = 0 and ξ^2 + b*ξ + c = 0
  shows ξ*ξ = c
  using assms
  by algebra

lemma viette2':
  fixes a b c ξ :: real
  assumes a ≠ 0 and b^2 − 4*a*c = 0 and a*ξ^2 + b*ξ + c = 0
  shows ξ*ξ = c/a
proof (rule viette2'-monic)
  have (b / a)^2 − 4 * (c / a) = (b^2 − 4*a*c) / a^2
    using ‹a ≠ 0›
    by (auto simp add: power2-eq-square field-simps)
  thus (b / a)^2 − 4 * (c / a) = 0
    using ‹b^2 − 4*a*c = 0›
    by simp
next
  show ξ^2 + b / a * ξ + c / a = 0
```

using *assms*
by (*auto simp add*: *power2-eq-square field-simps*)
qed

3.7.2 Complex quadratic equations

lemma *complex-quadratic-equation-monic-only-two-roots*:
fixes ξ :: *complex*
assumes $\xi^2 + b * \xi + c = 0$
shows $\xi = (-b + ccsqrt(b^2 - 4*c)) / 2 \vee \xi = (-b - ccsqrt(b^2 - 4*c)) / 2$
using *assms*
proof−
from *assms* **have** $(2 * (\xi + b/2))^2 = b^2 - 4*c$
by (*simp add*: *power2-eq-square field-simps*)
(*metis* (*no-types*, *lifting*) *distrib-right-numeral mult.assoc mult-zero-left*)
hence $2 * (\xi + b/2) = ccsqrt\ (b^2 - 4*c) \vee 2 * (\xi + b/2) = -\ ccsqrt\ (b^2 - 4*c)$
using *ccsqrt*[*of* $(2 * (\xi + b\ /\ 2))\ b^2 - 4 * c$]
by (*simp add*: *power2-eq-square*)
thus *?thesis*
using *mult-cancel-right*[*of* $b + \xi * 2\ 2$ *ccsqrt* $(b^2 - 4*c)$]
using *mult-cancel-right*[*of* $b + \xi * 2\ 2\ -ccsqrt\ (b^2 - 4*c)$]
by (*auto simp add*: *field-simps*) (*metis add-diff-cancel diff-minus-eq-add minus-diff-eq*)
qed

lemma *complex-quadratic-equation-monic-roots*:
fixes ξ :: *complex*
assumes $\xi = (-b + ccsqrt(b^2 - 4*c)) / 2 \vee$
$\xi = (-b - ccsqrt(b^2 - 4*c)) / 2$
shows $\xi^2 + b * \xi + c = 0$
using *assms*
proof
assume $*$: $\xi = (-\ b + ccsqrt\ (b^2 - 4\ *\ c))\ /\ 2$
show *?thesis*
by ((*subst* $*$)+) (*subst power-divide*, *subst power2-sum*, *simp add*: *field-simps*, *simp add*: *power2-eq-square*)
next
assume $*$: $\xi = (-\ b - ccsqrt\ (b^2 - 4\ *\ c))\ /\ 2$
show *?thesis*
by ((*subst* $*$)+, *subst power-divide*, *subst power2-diff*, *simp add*: *field-simps*, *simp add*: *power2-eq-square*)
qed

lemma *complex-quadratic-equation-monic-distinct-roots*:
fixes $b\ c$ :: *complex*
assumes $b^2 - 4*c \neq 0$
shows $\exists\ k_1\ k_2.\ k_1 \neq k_2 \wedge {k_1}^2 + b*k_1 + c = 0 \wedge {k_2}^2 + b*k_2 + c = 0$
proof−
let *?ξ1* $= (-b + ccsqrt(b^2 - 4*c)) / 2$
let *?ξ2* $= (-b - ccsqrt(b^2 - 4*c)) / 2$
show *?thesis*
apply (*rule-tac x*=*?ξ1* **in** *exI*)
apply (*rule-tac x*=*?ξ2* **in** *exI*)
using *assms*
using *complex-quadratic-equation-monic-roots*[*of ?ξ1 b c*]
using *complex-quadratic-equation-monic-roots*[*of ?ξ2 b c*]
by *simp*
qed

lemma *complex-quadratic-equation-two-roots*:
fixes ξ :: *complex*
assumes $a \neq 0$ **and** $a*\xi^2 + b * \xi + c = 0$
shows $\xi = (-b + ccsqrt(b^2 - 4*a*c)) / (2*a) \vee$
$\xi = (-b - ccsqrt(b^2 - 4*a*c)) / (2*a)$
proof−
from *assms* **have** $\xi^2 + (b/a) * \xi + (c/a) = 0$
by (*simp add*: *field-simps*)
hence $\xi = (-(b/a) + ccsqrt((b/a)^2 - 4*(c/a))) / 2 \vee \xi = (-(b/a) - ccsqrt((b/a)^2 - 4*(c/a))) / 2$
using *complex-quadratic-equation-monic-only-two-roots*[*of* ξ b/a c/a]

by *simp*
hence $\exists\ k.\ \xi = (-(b/a) + (-1)\hat{}k * ccsqrt((b/a)^2 - 4*(c/a))) / 2$
by *safe* (*rule-tac x=2* **in** *exI*, *simp*, *rule-tac x=1* **in** *exI*, *simp*)
then obtain *k1* **where** $\xi = (-(b/a) + (-1)\hat{}k1 * ccsqrt((b/a)^2 - 4*(c/a))) / 2$
by *auto*
moreover
have $(b\ /\ a)^2 - 4 * (c\ /\ a) = (b^2 - 4 * a * c) * (1\ /\ a^2)$
using ‹$a \neq 0$›
by (*simp add*: *field-simps power2-eq-square*)
hence $ccsqrt\ ((b\ /\ a)^2 - 4 * (c\ /\ a)) = ccsqrt\ (b^2 - 4 * a * c) * ccsqrt\ (1/a^2) \vee$
$ccsqrt\ ((b\ /\ a)^2 - 4 * (c\ /\ a)) = -\ ccsqrt\ (b^2 - 4 * a * c) * ccsqrt\ (1/a^2)$
using *ccsqrt-mult*[*of* $b^2 - 4 * a * c\ 1/a^2$]
by *auto*
hence $\exists\ k.\ \ ccsqrt\ ((b\ /\ a)^2 - 4 * (c\ /\ a)) = (-1)\hat{}k * ccsqrt\ (b^2 - 4 * a * c) * ccsqrt\ (1\ /\ a^2)$
by *safe* (*rule-tac x=2* **in** *exI*, *simp*, *rule-tac x=1* **in** *exI*, *simp*)
then obtain *k2* **where** $ccsqrt\ ((b\ /\ a)^2 - 4 * (c\ /\ a)) = (-1)\hat{}k2 * ccsqrt\ (b^2 - 4 * a * c) * ccsqrt\ (1\ /\ a^2)$
by *auto*
moreover
have $ccsqrt\ (1\ /\ a^2) = 1/a \vee ccsqrt\ (1\ /\ a^2) = -1/a$
using *ccsqrt*[*of* $1/a\ 1\ /\ a^2$]
by (*auto simp add*: *power2-eq-square*)
hence $\exists\ k.\ ccsqrt\ (1\ /\ a^2) = (-1)\hat{}k * 1/a$
by *safe* (*rule-tac x=2* **in** *exI*, *simp*, *rule-tac x=1* **in** *exI*, *simp*)
then obtain *k3* **where** $ccsqrt\ (1\ /\ a^2) = (-1)\hat{}k3 * 1/a$
by *auto*
ultimately
have $\xi = (-\ (b\ /\ a) + ((-1)\ \hat{}\ k1 * (-1)\ \hat{}\ k2 * (-1)\ \hat{}\ k3) * ccsqrt\ (b^2 - 4 * a * c) * 1/a) / 2$
by *simp*
moreover
have $(-(1::complex))\ \hat{}\ k1 * (-1)\ \hat{}\ k2 * (-1)\ \hat{}\ k3 = 1 \vee (-(1::complex))\ \hat{}\ k1 * (-1)\ \hat{}\ k2 * (-1)\ \hat{}\ k3 = -1$
using *neg-one-even-power*[*of k1 + k2 + k3*]
using *neg-one-odd-power*[*of k1 + k2 + k3*]
by (*smt* (*verit*) *power-add*)
ultimately
have $\xi = (-\ (b\ /\ a) + ccsqrt\ (b^2 - 4 * a * c) * 1\ /\ a) / 2 \vee \xi = (-\ (b\ /\ a) - ccsqrt\ (b^2 - 4 * a * c) * 1\ /\ a) / 2$
by *auto*
thus *?thesis*
using ‹$a \neq 0$›
by (*simp add*: *field-simps*)
qed

lemma *complex-quadratic-equation-only-two-roots*:
fixes $x :: complex$
assumes $a \neq 0$
assumes $qf = (\lambda\ x.\ a*x^2 + b*x + c)$
$qf\ x1 = 0$ **and** $qf\ x2 = 0$ **and** $x1 \neq x2$
$qf\ x = 0$
shows $x = x1 \vee x = x2$
using *assms*
using *complex-quadratic-equation-two-roots*
by *blast*

3.7.3 Intersections of linear and quadratic forms

lemma *quadratic-linear-at-most-2-intersections-help*:
fixes $x\ y :: complex$
assumes $(a11,\ a12,\ a22) \neq (0,\ 0,\ 0)$ **and** $k2 \neq 0$
$qf = (\lambda\ x\ y.\ a11*x^2 + 2*a12*x*y + a22*y^2 + b1*x + b2*y + c)$ **and** $lf = (\lambda\ x\ y.\ k1*x + k2*y + n)$
$qf\ x\ y = 0$ **and** $lf\ x\ y = 0$
$pf = (\lambda\ x.\ (a11 - 2*a12*k1/k2 + a22*k1^2/k2^2)*x^2 + (-2*a12*n/k2 + b1 + a22*2*n*k1/k2^2 - b2*k1/k2)*x$
$+ a22*n^2/k2^2 - b2*n/k2 + c)$
$yf = (\lambda\ x.\ (-n - k1*x)\ /\ k2)$
shows $pf\ x = 0$ **and** $y = yf\ x$
proof −
show $y = yf\ x$
using *assms*

```
    by (simp add:field-simps eq-neg-iff-add-eq-0)
next
  have 2*a12*x*(-n - k1*x)/k2 = (-2*a12*n/k2)*x - (2*a12*k1/k2)*x^2
    by algebra
  have a22*((-n - k1*x)/k2)^2 = a22*n^2/k2^2 + (a22*2*n*k1/k2^2)*x + (a22*k1^2/k2^2)*x^2
    by (simp add: power-divide) algebra
  have 2*a12*x*(-n - k1*x)/k2 = (-2*a12*n/k2)*x - (2*a12*k1/k2)*x^2
    by algebra
  have b2*(-n - k1*x)/k2 = -b2*n/k2 - (b2*k1/k2)*x
    by algebra

  have *: y = (-n - k1*x)/k2
    using assms(2, 4, 6)
    by (simp add:field-simps eq-neg-iff-add-eq-0)

  have 0 = a11*x^2 + 2*a12*x*y + a22*y^2 + b1*x + b2*y + c
    using assms
    by simp
  hence 0 = a11*x^2 + 2*a12*x*(-n - k1*x)/k2 + a22*((-n - k1*x)/k2)^2 + b1*x + b2*(-n - k1*x)/k2 + c
    by (subst (asm) *, subst (asm) *, subst (asm) *) auto
 also have ... = (a11 - 2*a12*k1/k2 + a22*k1^2/k2^2)*x^2 + (-2*a12*n/k2 + b1 + a22*2*n*k1/k2^2 - b2*k1/k2)*x
+ a22*n^2/k2^2 -b2*n/k2 + c
    using ‹2*a12*x*(-n - k1*x)/k2 = (-2*a12*n/k2)*x - (2*a12*k1/k2)*x^2›
    using ‹a22*((-n - k1*x)/k2)^2 = a22*n^2/k2^2 + (a22*2*n*k1/k2^2)*x + (a22*k1^2/k2^2)*x^2›
    using ‹b2*(-n - k1*x)/k2 = -b2*n/k2 - (b2*k1/k2)*x›
    by (simp add:field-simps)
  finally show pf x = 0
    using assms(7)
    by auto
qed

lemma quadratic-linear-at-most-2-intersections-help':
  fixes x y :: complex
  assumes qf = (λ x y. a11*x^2 + 2*a12*x*y + a22*y^2 + b1*x + b2*y + c)
        x = -n/k1 and k1 ≠ 0 and qf x y = 0
        yf = (λ y. k1^2*a22*y^2 + (-2*a12*n*k1 + b2*k1^2)*y + a11*n^2 - b1*n*k1 + c*k1^2)
  shows yf y = 0
proof-
  have 0 = a11*n^2/k1^2 - 2*a12*n*y/k1 + a22*y^2 - b1*n/k1 + b2*y + c
    using assms(1, 2, 4)
    by (simp add: power-divide)
  hence 0 = a11*n^2 - 2*a12*n*k1*y + a22*y^2*k1^2 - b1*n*k1 + b2*y*k1^2 + c*k1^2
    using assms(3)
    apply (simp add:field-simps power2-eq-square)
    by algebra
  thus ?thesis
    using assms(1, 4, 5)
    by (simp add:field-simps)
qed

lemma quadratic-linear-at-most-2-intersections:
  fixes x y x1 y1 x2 y2 :: complex
  assumes (a11, a12, a22) ≠ (0, 0, 0) and (k1, k2) ≠ (0, 0)
  assumes a11*k2^2 - 2*a12*k1*k2 + a22*k1^2 ≠ 0
  assumes qf = (λ x y. a11*x^2 + 2*a12*x*y + a22*y^2 + b1*x + b2*y + c) and lf = (λ x y. k1*x + k2*y + n)
        qf x1 y1 = 0 and lf x1 y1 = 0
        qf x2 y2 = 0 and lf x2 y2 = 0
        (x1, y1) ≠ (x2, y2)
        qf x y = 0 and lf x y = 0
  shows (x, y) = (x1, y1) ∨ (x, y) = (x2, y2)
proof(cases k2 = 0)
  case True
  hence k1 ≠ 0
    using assms(2)
    by simp
```

```
  have a22*k1^2 ≠ 0
    using assms(3) True
    by auto

  have x1 = −n/k1
    using ‹k1 ≠ 0› assms(5, 7) True
    by (metis add.right-neutral add-eq-0-iff2 mult-zero-left nonzero-mult-div-cancel-left)
  have x2 = −n/k1
    using ‹k1 ≠ 0› assms(5, 9) True
    by (metis add.right-neutral add-eq-0-iff2 mult-zero-left nonzero-mult-div-cancel-left)
  have x = −n/k1
    using ‹k1 ≠ 0› assms(5, 12) True
    by (metis add.right-neutral add-eq-0-iff2 mult-zero-left nonzero-mult-div-cancel-left)

  let ?yf = (λ y. k1^2*a22*y^2 + (−2*a12*n*k1 + b2*k1^2)*y + a11*n^2 − b1*n*k1 + c*k1^2)

  have ?yf y = 0
    using quadratic-linear-at-most-2-intersections-help'[of qf a11 a12 a22 b1 b2 c x n k1 y ?yf]
    using assms(4, 11) ‹k1 ≠ 0› ‹x = −n/k1›
    by auto
  have ?yf y1 = 0
    using quadratic-linear-at-most-2-intersections-help'[of qf a11 a12 a22 b1 b2 c x1 n k1 y1 ?yf]
    using assms(4, 6) ‹k1 ≠ 0› ‹x1 = −n/k1›
    by auto
  have ?yf y2 = 0
    using quadratic-linear-at-most-2-intersections-help'[of qf a11 a12 a22 b1 b2 c x2 n k1 y2 ?yf]
    using assms(4, 8) ‹k1 ≠ 0› ‹x2 = −n/k1›
    by auto

  have y1 ≠ y2
    using assms(10) ‹x1 = −n/k1› ‹x2 = −n/k1›
    by blast

  have y = y1 ∨ y = y2
     using complex-quadratic-equation-only-two-roots[of a22*k1^2 ?yf −2*a12*n*k1 + b2*k1^2 a11*n^2 − b1*n*k1 +
c*k1^2
                                                      y1 y2 y]
    using ‹a22*k1^2 ≠ 0› ‹?yf y1 = 0› ‹y1 ≠ y2› ‹?yf y2 = 0› ‹?yf y = 0›
    by fastforce

  thus ?thesis
    using ‹x1 = −n/k1› ‹x2 = −n/k1› ‹x = −n/k1›
    by auto
next
  case False

  let ?py = (λ x. (−n − k1*x)/k2)
  let ?pf = (λ x. (a11 − 2*a12*k1/k2 + a22*k1^2/k2^2)*x^2 + (−2*a12*n/k2 + b1 + a22*2*n*k1/k2^2 − b2*k1/k2)*x
+ a22*n^2/k2^2 −b2*n/k2 + c)
  have ?pf x1 = 0 y1 = ?py x1
    using quadratic-linear-at-most-2-intersections-help[of a11 a12 a22 k2 qf b1 b2 c lf k1 n x1 y1]
    using assms(1, 4, 5, 6, 7) False
    by auto
  have ?pf x2 = 0 y2 = ?py x2
    using quadratic-linear-at-most-2-intersections-help[of a11 a12 a22 k2 qf b1 b2 c lf k1 n x2 y2]
    using assms(1, 4, 5, 8, 9) False
    by auto
  have ?pf x = 0 y = ?py x
    using quadratic-linear-at-most-2-intersections-help[of a11 a12 a22 k2 qf b1 b2 c lf k1 n x y]
    using assms(1, 4, 5, 11, 12) False
    by auto

  have x1 ≠ x2
    using assms(10) ‹y1 = ?py x1› ‹y2 = ?py x2›
    by auto
```

```
  have a11 − 2∗a12∗k1/k2 + a22∗k1²/k2² = (a11 ∗ k2² − 2 ∗ a12 ∗ k1 ∗ k2 + a22 ∗ k1²)/k2²
    by (simp add: False power2-eq-square add-divide-distrib diff-divide-distrib)
  also have ... ≠ 0
    using False assms(3)
    by simp
  finally have a11 − 2∗a12∗k1/k2 + a22∗k1²/k2² ≠ 0
    .

  have x = x1 ∨ x = x2
    using complex-quadratic-equation-only-two-roots[of a11 − 2∗a12∗k1/k2 + a22∗k1²/k2² ?pf
                                                       (− 2 ∗ a12 ∗ n / k2 + b1 + a22 ∗ 2 ∗ n ∗ k1 / k2² − b2 ∗ k1 / k2)
                                                       a22 ∗ n² / k2² − b2 ∗ n / k2 + c x1 x2 x]
    using ‹?pf x2 = 0› ‹?pf x1 = 0› ‹?pf x = 0›
    using ‹a11 − 2 ∗ a12 ∗ k1 / k2 + a22 ∗ k1² / k2² ≠ 0›
    using ‹x1 ≠ x2›
    by fastforce

  thus ?thesis
    using ‹y = ?py x› ‹y1 = ?py x1› ‹y2 = ?py x2›
    by (cases x = x1, auto)
qed

lemma quadratic-quadratic-at-most-2-intersections':
  fixes x y x1 y1 x2 y2 :: complex
  assumes b2 ≠ B2 ∨ b1 ≠ B1
        (b2 − B2)² + (b1 − B1)² ≠ 0
  assumes qf1 = (λ x y. x² + y² + b1∗x + b2∗y + c)
        qf2 = (λ x y. x² + y² + B1∗x + B2∗y + C)
        qf1 x1 y1 = 0 qf2 x1 y1 = 0
        qf1 x2 y2 = 0 qf2 x2 y2 = 0
        (x1, y1) ≠ (x2, y2)
        qf1 x y = 0 qf2 x y = 0
  shows (x, y) = (x1, y1) ∨ (x, y) = (x2, y2)
proof−
  have x² + y² + b1∗x + b2∗y + c = 0
    using assms by auto
  have x² + y² + B1∗x + B2∗y + C = 0
    using assms by auto
  hence 0 = x² + y² + b1∗x + b2∗y + c − (x² + y² + B1∗x + B2∗y + C)
    using ‹x² + y² + b1∗x + b2∗y + c = 0›
    by auto
  hence 0 = (b1 − B1)∗x + (b2 − B2)∗y + c − C
    by (simp add:field-simps)

  have x1² + y1² + b1∗x1 + b2∗y1 + c = 0
    using assms by auto
  have x1² + y1² + B1∗x1 + B2∗y1 + C = 0
    using assms by auto
  hence 0 = x1² + y1² + b1∗x1 + b2∗y1 + c − (x1² + y1² + B1∗x1 + B2∗y1 + C)
    using ‹x1² + y1² + b1∗x1 + b2∗y1 + c = 0›
    by auto
  hence 0 = (b1 − B1)∗x1 + (b2 − B2)∗y1 + c − C
    by (simp add:field-simps)

  have x2² + y2² + b1∗x2 + b2∗y2 + c = 0
    using assms by auto
  have x2² + y2² + B1∗x2 + B2∗y2 + C = 0
    using assms by auto
  hence 0 = x2² + y2² + b1∗x2 + b2∗y2 + c − (x2² + y2² + B1∗x2 + B2∗y2 + C)
    using ‹x2² + y2² + b1∗x2 + b2∗y2 + c = 0›
    by auto
  hence 0 = (b1 − B1)∗x2 + (b2 − B2)∗y2 + c − C
    by (simp add:field-simps)

  have (b1 − B1, b2 − B2) ≠ (0, 0)
    using assms(1) by auto
```

let *?lf* = $(\lambda\ x\ y.\ (b1 - B1)*x + (b2 - B2)*y + c - C)$

have *?lf x y* = *0 ?lf x1 y1* = *0 ?lf x2 y2* = *0*
using ‹*0* = $(b1 - B1)*x2 + (b2 - B2)*y2 + c - C$›
‹*0* = $(b1 - B1)*x1 + (b2 - B2)*y1 + c - C$›
‹*0* = $(b1 - B1)*x + (b2 - B2)*y + c - C$›
by *auto*

thus *?thesis*
using *quadratic-linear-at-most-2-intersections*[*of 1 0 1 b1 − B1 b2 − B2 qf1 b1 b2 c ?lf c − C x1 y1 x2 y2 x y*]
using ‹$(b1 - B1,\ b2 - B2) \neq (0,\ 0)$›
using *assms* ‹$(b1 - B1,\ b2 - B2) \neq (0,\ 0)$›
using ‹$(b1 - B1) * x + (b2 - B2) * y + c - C = 0$› ‹$(b1 - B1) * x1 + (b2 - B2) * y1 + c - C = 0$›
by (*simp add*: *add-diff-eq*)
qed

lemma *quadratic-change-coefficients*:
fixes *x y* :: *complex*
assumes *A1* ≠ *0*
assumes *qf* = $(\lambda\ x\ y.\ A1*x^2 + A1*y^2 + b1*x + b2*y + c)$
qf x y = *0*
qf-1 = $(\lambda\ x\ y.\ x^2 + y^2 + (b1/A1)*x + (b2/A1)*y + c/A1)$
shows *qf-1 x y* = *0*
proof−
have *0* = $A1*x^2 + A1*y^2 + b1*x + b2*y + c$
using *assms* **by** *auto*
hence *0*/*A1* = $(A1*x^2 + A1*y^2 + b1*x + b2*y + c)/A1$
using *assms*(*1*) **by** *auto*
also have ... = $A1*x^2/A1 + A1*y^2/A1 + b1*x/A1 + b2*y/A1 + c/A1$
by (*simp add*: *add-divide-distrib*)
also have ... = $x^2 + y^2 + (b1/A1)*x + (b2/A1)*y + c/A1$
using *assms*(*1*)
by (*simp add*:*field-simps*)
finally have *0* = $x^2 + y^2 + (b1/A1)*x + (b2/A1)*y + c/A1$
by *simp*
thus *?thesis*
using *assms*
by *simp*
qed

lemma *quadratic-quadratic-at-most-2-intersections*:
fixes *x y x1 y1 x2 y2* :: *complex*
assumes *A1* ≠ *0* **and** *A2* ≠ *0*
assumes *qf1* = $(\lambda\ x\ y.\ A1*x^2 + A1*y^2 + b1*x + b2*y + c)$ **and**
qf2 = $(\lambda\ x\ y.\ A2*x^2 + A2*y^2 + B1*x + B2*y + C)$ **and**
qf1 x1 y1 = *0* **and** *qf2 x1 y1* = *0* **and**
qf1 x2 y2 = *0* **and** *qf2 x2 y2* = *0* **and**
$(x1,\ y1) \neq (x2,\ y2)$ **and**
qf1 x y = *0* **and** *qf2 x y* = *0*
assumes $(b2*A2 - B2*A1)^2 + (b1*A2 - B1*A1)^2 \neq 0$ **and**
*b2*A2* ≠ *B2*A1* ∨ *b1*A2* ≠ *B1*A1*
shows $(x,\ y) = (x1,\ y1) \vee (x,\ y) = (x2,\ y2)$
proof−
have ∗: *b2 / A1* ≠ *B2 / A2* ∨ *b1 / A1* ≠ *B1 / A2*
using *assms*(*1*, *2*) *assms*(*13*)
by (*simp add*:*field-simps*)
have ∗∗: $(b2 / A1 - B2 / A2)^2 + (b1 / A1 - B1 / A2)^2 \neq 0$
using *assms*(*1*, *2*) *assms*(*12*)
by (*simp add*:*field-simps*)

let *?qf-1* = $(\lambda\ x\ y.\ x^2 + y^2 + (b1/A1)*x + (b2/A1)*y + c/A1)$
let *?qf-2* = $(\lambda\ x\ y.\ x^2 + y^2 + (B1/A2)*x + (B2/A2)*y + C/A2)$

have *?qf-1 x1 y1* = *0 ?qf-1 x2 y2* = *0 ?qf-1 x y* = *0*
?qf-2 x1 y1 = *0 ?qf-2 x2 y2* = *0 ?qf-2 x y* = *0*

using *assms quadratic-change-coefficients*[*of A1 qf1 b1 b2 c x2 y2 ?qf-1*]
quadratic-change-coefficients[*of A1 qf1 b1 b2 c x1 y1 ?qf-1*]
quadratic-change-coefficients[*of A2 qf2 B1 B2 C x1 y1 ?qf-2*]
quadratic-change-coefficients[*of A2 qf2 B1 B2 C x2 y2 ?qf-2*]
quadratic-change-coefficients[*of A1 qf1 b1 b2 c x y ?qf-1*]
quadratic-change-coefficients[*of A2 qf2 B1 B2 C x y ?qf-2*]
by *auto*

thus *?thesis*
using *quadratic-quadratic-at-most-2-intersections′*
[*of b2 / A1 B2 / A2 b1 / A1 B1 / A2 ?qf-1 c / A1 ?qf-2 C / A2 x1 y1 x2 y2 x y*]
using ∗ ∗∗ ‹(*x1*, *y1*) ≠ (*x2*, *y2*)›
by *fastforce*
qed

end

3.8 Vectors and Matrices in $\mathbb{C}^2$

Representing vectors and matrices of arbitrary dimensions pose a challenge in formal theorem proving [4], but we only need to consider finite dimension spaces $\mathbb{C}^2$ and $\mathbb{R}^3$.

theory *Matrices*
imports *More-Complex Linear-Systems Quadratic*
begin

3.8.1 Vectors in $\mathbb{C}^2$

Type of complex vector

type-synonym *complex-vec* = *complex* × *complex*

definition *vec-zero* :: *complex-vec* **where**
[*simp*]: *vec-zero* = (*0*, *0*)

Vector scalar multiplication

fun *mult-sv* :: *complex* ⇒ *complex-vec* ⇒ *complex-vec* (**infixl** ‹$*_{sv}$› *100*) **where**
k $*_{sv}$ (*x*, *y*) = (*k*∗*x*, *k*∗*y*)

lemma *fst-mult-sv* [*simp*]:
shows *fst* (*k* $*_{sv}$ *v*) = *k* ∗ *fst v*
by (*cases v*) *simp*

lemma *snd-mult-sv* [*simp*]:
shows *snd* (*k* $*_{sv}$ *v*) = *k* ∗ *snd v*
by (*cases v*) *simp*

lemma *mult-sv-mult-sv* [*simp*]:
shows *k1* $*_{sv}$ (*k2* $*_{sv}$ *v*) = (*k1*∗*k2*) $*_{sv}$ *v*
by (*cases v*) *simp*

lemma *one-mult-sv* [*simp*]:
shows *1* $*_{sv}$ *v* = *v*
by (*cases v*) *simp*

lemma *mult-sv-ex-id1* [*simp*]:
shows ∃ *k*::*complex*. *k* ≠ *0* ∧ *k* $*_{sv}$ *v* = *v*
by (*rule-tac x=1* **in** *exI*, *simp*)

lemma *mult-sv-ex-id2* [*simp*]:
shows ∃ *k*::*complex*. *k* ≠ *0* ∧ *v* = *k* $*_{sv}$ *v*
by (*rule-tac x=1* **in** *exI*, *simp*)

Scalar product of two vectors

fun *mult-vv* :: *complex* × *complex* ⇒ *complex* × *complex* ⇒ *complex* (**infixl** ‹$*_{vv}$› *100*) **where**
(*x*, *y*) $*_{vv}$ (*a*, *b*) = *x*∗*a* + *y*∗*b*

lemma *mult-vv-commute*:
shows *v1* $*_{vv}$ *v2* = *v2* $*_{vv}$ *v1*
by (*cases v1*, *cases v2*) *auto*

lemma *mult-vv-scale-sv1*:
shows (*k* $*_{sv}$ *v1*) $*_{vv}$ *v2* = *k* * (*v1* $*_{vv}$ *v2*)
by (*cases v1*, *cases v2*) (*auto simp add*: *field-simps*)

lemma *mult-vv-scale-sv2*:
shows *v1* $*_{vv}$ (*k* $*_{sv}$ *v2*) = *k* * (*v1* $*_{vv}$ *v2*)
by (*cases v1*, *cases v2*) (*auto simp add*: *field-simps*)

Conjugate vector

fun *vec-map* **where**
vec-map f (*x*, *y*) = (*f x*, *f y*)

definition *vec-cnj* **where**
vec-cnj = *vec-map cnj*

lemma *vec-cnj-vec-cnj* [*simp*]:
shows *vec-cnj* (*vec-cnj v*) = *v*
by (*cases v*) (*simp add*: *vec-cnj-def*)

lemma *cnj-mult-vv*:
shows *cnj* (*v1* $*_{vv}$ *v2*) = (*vec-cnj v1*) $*_{vv}$ (*vec-cnj v2*)
by (*cases v1*, *cases v2*) (*simp add*: *vec-cnj-def*)

lemma *vec-cnj-sv* [*simp*]:
shows *vec-cnj* (*k* $*_{sv}$ *A*) = *cnj k* $*_{sv}$ *vec-cnj A*
by (*cases A*) (*auto simp add*: *vec-cnj-def*)

lemma *scalsquare-vv-zero*:
shows (*vec-cnj v*) $*_{vv}$ *v* = *0* ⟷ *v* = *vec-zero*
apply (*cases v*)
apply (*auto simp add*: *vec-cnj-def field-simps complex-mult-cnj-cmod power2-eq-square*)
apply (*metis* (*no-types*) *norm-eq-zero of-real-0 of-real-add of-real-eq-iff of-real-mult sum-squares-eq-zero-iff*)+
done

3.8.2 Matrices in $\mathbb{C}^2$

Type of complex matrices

type-synonym *complex-mat* = *complex* × *complex* × *complex* × *complex*

Matrix scalar multiplication

fun *mult-sm* :: *complex* ⇒ *complex-mat* ⇒ *complex-mat* (**infixl** ‹$*_{sm}$› *100*) **where**
k $*_{sm}$ (*a*, *b*, *c*, *d*) = (*k*∗*a*, *k*∗*b*, *k*∗*c*, *k*∗*d*)

lemma *mult-sm-distribution* [*simp*]:
shows *k1* $*_{sm}$ (*k2* $*_{sm}$ *A*) = (*k1*∗*k2*) $*_{sm}$ *A*
by (*cases A*) *auto*

lemma *mult-sm-neutral* [*simp*]:
shows *1* $*_{sm}$ *A* = *A*
by (*cases A*) *auto*

lemma *mult-sm-inv-l*:
assumes *k* ≠ *0* **and** *k* $*_{sm}$ *A* = *B*
shows *A* = (*1*/*k*) $*_{sm}$ *B*
using *assms*
by *auto*

lemma *mult-sm-ex-id1* [*simp*]:
shows ∃ *k*::*complex*. *k* ≠ *0* ∧ *k* $*_{sm}$ *M* = *M*
by (*rule-tac x*=*1* **in** *exI*, *simp*)

lemma *mult-sm-ex-id2* [*simp*]:
shows $\exists\ k{::}complex.\ k \neq 0 \wedge M = k *_{sm} M$
by (*rule-tac x=1* **in** *exI*, *simp*)

Matrix addition and subtraction

definition *mat-zero* :: *complex-mat* **where** [*simp*]: *mat-zero* = (*0*, *0*, *0*, *0*)

fun *mat-plus* :: *complex-mat* ⇒ *complex-mat* ⇒ *complex-mat* (**infixl** ‹$+_{mm}$› *100*) **where**
mat-plus (*a1*, *b1*, *c1*, *d1*) (*a2*, *b2*, *c2*, *d2*) = (*a1*+*a2*, *b1*+*b2*, *c1*+*c2*, *d1*+*d2*)

fun *mat-minus* :: *complex-mat* ⇒ *complex-mat* ⇒ *complex-mat* (**infixl** ‹$-_{mm}$› *100*) **where**
mat-minus (*a1*, *b1*, *c1*, *d1*) (*a2*, *b2*, *c2*, *d2*) = (*a1*−*a2*, *b1*−*b2*, *c1*−*c2*, *d1*−*d2*)

fun *mat-uminus* :: *complex-mat* ⇒ *complex-mat* **where**
mat-uminus (*a*, *b*, *c*, *d*) = (−*a*, −*b*, −*c*, −*d*)

lemma *nonzero-mult-real*:
assumes *A* ≠ *mat-zero* **and** *k* ≠ *0*
shows *k* $*_{sm}$ *A* ≠ *mat-zero*
using *assms*
by (*cases A*) *simp*

Matrix multiplication.

fun *mult-mm* :: *complex-mat* ⇒ *complex-mat* ⇒ *complex-mat* (**infixl** ‹$*_{mm}$› *100*) **where**
(*a1*, *b1*, *c1*, *d1*) $*_{mm}$ (*a2*, *b2*, *c2*, *d2*) =
(*a1*∗*a2* + *b1*∗*c2*, *a1*∗*b2* + *b1*∗*d2*, *c1*∗*a2*+*d1*∗*c2*, *c1*∗*b2*+*d1*∗*d2*)

lemma *mult-mm-assoc*:
shows *A* $*_{mm}$ (*B* $*_{mm}$ *C*) = (*A* $*_{mm}$ *B*) $*_{mm}$ *C*
by (*cases A*, *cases B*, *cases C*) (*auto simp add*: *field-simps*)

lemma *mult-assoc-5*:
shows *A* $*_{mm}$ (*B* $*_{mm}$ *C* $*_{mm}$ *D*) $*_{mm}$ *E* = (*A* $*_{mm}$ *B*) $*_{mm}$ *C* $*_{mm}$ (*D* $*_{mm}$ *E*)
by (*simp only*: *mult-mm-assoc*)

lemma *mat-zero-r* [*simp*]:
shows *A* $*_{mm}$ *mat-zero* = *mat-zero*
by (*cases A*) *simp*

lemma *mat-zero-l* [*simp*]:
shows *mat-zero* $*_{mm}$ *A* = *mat-zero*
by (*cases A*) *simp*

definition *eye* :: *complex-mat* **where**
[*simp*]: *eye* = (*1*, *0*, *0*, *1*)

lemma *mat-eye-l*:
shows *eye* $*_{mm}$ *A* = *A*
by (*cases A*) *auto*

lemma *mat-eye-r*:
shows *A* $*_{mm}$ *eye* = *A*
by (*cases A*) *auto*

lemma *mult-mm-sm* [*simp*]:
shows *A* $*_{mm}$ (*k* $*_{sm}$ *B*) = *k* $*_{sm}$ (*A* $*_{mm}$ *B*)
by (*cases A*, *cases B*) (*simp add*: *field-simps*)

lemma *mult-sm-mm* [*simp*]:
shows (*k* $*_{sm}$ *A*) $*_{mm}$ *B* = *k* $*_{sm}$ (*A* $*_{mm}$ *B*)
by (*cases A*, *cases B*) (*simp add*: *field-simps*)

lemma *mult-sm-eye-mm* [*simp*]:
shows *k* $*_{sm}$ *eye* $*_{mm}$ *A* = *k* $*_{sm}$ *A*
by (*cases A*) *simp*

Matrix determinant

fun *mat-det* **where** *mat-det* (*a*, *b*, *c*, *d*) = *a*∗*d* − *b*∗*c*

lemma *mat-det-mult* [*simp*]:
shows *mat-det* (*A* $*_{mm}$ *B*) = *mat-det A* ∗ *mat-det B*
by (*cases A*, *cases B*) (*auto simp add*: *field-simps*)

lemma *mat-det-mult-sm* [*simp*]:
shows *mat-det* (*k* $*_{sm}$ *A*) = (*k*∗*k*) ∗ *mat-det A*
by (*cases A*) (*auto simp add*: *field-simps*)

Matrix inverse

fun *mat-inv* :: *complex-mat* ⇒ *complex-mat* **where**
mat-inv (*a*, *b*, *c*, *d*) = (*1*/(*a*∗*d* − *b*∗*c*)) $*_{sm}$ (*d*, −*b*, −*c*, *a*)

lemma *mat-inv-r*:
assumes *mat-det A* ≠ *0*
shows *A* $*_{mm}$ (*mat-inv A*) = *eye*
using *assms*
proof (*cases A*, *auto simp add*: *field-simps*)
fix *a b c d* :: *complex*
assume *a* ∗ (*a* ∗ (*d* ∗ *d*)) + *b* ∗ (*b* ∗ (*c* ∗ *c*)) = *a* ∗ (*b* ∗ (*c* ∗ (*d* ∗ *2*)))
hence (*a*∗*d* − *b*∗*c*)∗(*a*∗*d* − *b*∗*c*) = *0*
by (*auto simp add*: *field-simps*)
hence ∗: *a*∗*d* − *b*∗*c* = *0*
by *auto*
assume *a*∗*d* ≠ *b*∗*c*
with ∗ **show** *False*
by *auto*
qed

lemma *mat-inv-l*:
assumes *mat-det A* ≠ *0*
shows (*mat-inv A*) $*_{mm}$ *A* = *eye*
using *assms*
proof (*cases A*, *auto simp add*: *field-simps*)
fix *a b c d* :: *complex*
assume *a* ∗ (*a* ∗ (*d* ∗ *d*)) + *b* ∗ (*b* ∗ (*c* ∗ *c*)) = *a* ∗ (*b* ∗ (*c* ∗ (*d* ∗ *2*)))
hence (*a*∗*d* − *b*∗*c*)∗(*a*∗*d* − *b*∗*c*) = *0*
by (*auto simp add*: *field-simps*)
hence ∗: *a*∗*d* − *b*∗*c* = *0*
by *auto*
assume *a*∗*d* ≠ *b*∗*c*
with ∗ **show** *False*
by *auto*
qed

lemma *mat-det-inv*:
assumes *mat-det A* ≠ *0*
shows *mat-det* (*mat-inv A*) = *1* / *mat-det A*
proof−
have *mat-det eye* = *mat-det A* ∗ *mat-det* (*mat-inv A*)
using *mat-inv-l*[*OF assms*, *symmetric*]
by *simp*
thus *?thesis*
using *assms*
by (*simp add*: *field-simps*)
qed

lemma *mult-mm-inv-l*:
assumes *mat-det A* ≠ *0* **and** *A* $*_{mm}$ *B* = *C*
shows *B* = *mat-inv A* $*_{mm}$ *C*
using *assms mat-eye-l*[*of B*]
by (*auto simp add*: *mult-mm-assoc mat-inv-l*)

lemma *mult-mm-inv-r*:

assumes *mat-det B* $\neq$ *0* **and** *A* $*_{mm}$ *B* = *C*
shows *A* = *C* $*_{mm}$ *mat-inv B*
using *assms mat-eye-r*[*of A*]
by (*auto simp add*: *mult-mm-assoc*[*symmetric*] *mat-inv-r*)

lemma *mult-mm-non-zero-l*:
assumes *mat-det A* $\neq$ *0* **and** *B* $\neq$ *mat-zero*
shows *A* $*_{mm}$ *B* $\neq$ *mat-zero*
using *assms mat-zero-r*
using *mult-mm-inv-l*[*OF assms*(*1*), *of B mat-zero*]
by *auto*

lemma *mat-inv-mult-mm*:
assumes *mat-det A* $\neq$ *0* **and** *mat-det B* $\neq$ *0*
shows *mat-inv* (*A* $*_{mm}$ *B*) = *mat-inv B* $*_{mm}$ *mat-inv A*
using *assms*
proof−
have (*A* $*_{mm}$ *B*) $*_{mm}$ (*mat-inv B* $*_{mm}$ *mat-inv A*) = *eye*
using *assms*
by (*metis mat-inv-r mult-mm-assoc mult-mm-inv-r*)
thus *?thesis*
using *mult-mm-inv-l*[*of A* $*_{mm}$ *B mat-inv B* $*_{mm}$ *mat-inv A eye*] *assms mat-eye-r*
by *simp*
qed

lemma *mult-mm-cancel-l*:
assumes *mat-det M* $\neq$ *0* *M* $*_{mm}$ *A* = *M* $*_{mm}$ *B*
shows *A* = *B*
using *assms*
by (*metis mult-mm-inv-l*)

lemma *mult-mm-cancel-r*:
assumes *mat-det M* $\neq$ *0* *A* $*_{mm}$ *M* = *B* $*_{mm}$ *M*
shows *A* = *B*
using *assms*
by (*metis mult-mm-inv-r*)

lemma *mult-mm-non-zero-r*:
assumes *A* $\neq$ *mat-zero* **and** *mat-det B* $\neq$ *0*
shows *A* $*_{mm}$ *B* $\neq$ *mat-zero*
using *assms mat-zero-l*
using *mult-mm-inv-r*[*OF assms*(*2*), *of A mat-zero*]
by *auto*

lemma *mat-inv-mult-sm*:
assumes *k* $\neq$ *0*
shows *mat-inv* (*k* $*_{sm}$ *A*) = (*1* / *k*) $*_{sm}$ *mat-inv A*
proof−
obtain *a b c d* **where** *A* = (*a*, *b*, *c*, *d*)
by (*cases A*) *auto*
thus *?thesis*
using *assms*
by *auto* (*subst mult.assoc*[*of k a k*∗*d*], *subst mult.assoc*[*of k b k*∗*c*], *subst right-diff-distrib*[*of k a*∗(*k*∗*d*) *b*∗(*k*∗*c*), *symmetric*], *simp*, *simp add*: *field-simps*)+
qed

lemma *mat-inv-inv* [*simp*]:
assumes *mat-det M* $\neq$ *0*
shows *mat-inv* (*mat-inv M*) = *M*
proof−
have *mat-inv M* $*_{mm}$ *M* = *eye*
using *mat-inv-l*[*OF assms*]
by *simp*
thus *?thesis*
using *assms mat-det-inv*[*of M*]
using *mult-mm-inv-l*[*of mat-inv M M eye*] *mat-eye-r*

by (*auto simp del*: *eye-def*)
qed

Matrix transpose

fun *mat-transpose* **where**
mat-transpose (*a*, *b*, *c*, *d*) = (*a*, *c*, *b*, *d*)

lemma *mat-t-mat-t* [*simp*]:
shows *mat-transpose* (*mat-transpose A*) = *A*
by (*cases A*) *auto*

lemma *mat-t-mult-sm* [*simp*]:
shows *mat-transpose* (*k* $*_{sm}$ *A*) = *k* $*_{sm}$ (*mat-transpose A*)
by (*cases A*) *simp*

lemma *mat-t-mult-mm* [*simp*]:
shows *mat-transpose* (*A* $*_{mm}$ *B*) = *mat-transpose B* $*_{mm}$ *mat-transpose A*
by (*cases A*, *cases B*) *auto*

lemma *mat-inv-transpose*:
shows *mat-transpose* (*mat-inv M*) = *mat-inv* (*mat-transpose M*)
by (*cases M*) *auto*

lemma *mat-det-transpose* [*simp*]:
fixes *M* :: *complex-mat*
shows *mat-det* (*mat-transpose M*) = *mat-det M*
by (*cases M*) *auto*

Diagonal matrices definition

fun *mat-diagonal* **where**
mat-diagonal (*A*, *B*, *C*, *D*) = (*B* = *0* ∧ *C* = *0*)

Matrix conjugate

fun *mat-map* **where**
mat-map f (*a*, *b*, *c*, *d*) = (*f a*, *f b*, *f c*, *f d*)

definition *mat-cnj* **where**
mat-cnj = *mat-map cnj*

lemma *mat-cnj-cnj* [*simp*]:
shows *mat-cnj* (*mat-cnj A*) = *A*
unfolding *mat-cnj-def*
by (*cases A*) *auto*

lemma *mat-cnj-sm* [*simp*]:
shows *mat-cnj* (*k* $*_{sm}$ *A*) = *cnj k* $*_{sm}$ (*mat-cnj A*)
by (*cases A*) (*simp add*: *mat-cnj-def*)

lemma *mat-det-cnj* [*simp*]:
shows *mat-det* (*mat-cnj A*) = *cnj* (*mat-det A*)
by (*cases A*) (*simp add*: *mat-cnj-def*)

lemma *nonzero-mat-cnj*:
shows *mat-cnj A* = *mat-zero* ⟷ *A* = *mat-zero*
by (*cases A*) (*auto simp add*: *mat-cnj-def*)

lemma *mat-inv-cnj*:
shows *mat-cnj* (*mat-inv M*) = *mat-inv* (*mat-cnj M*)
unfolding *mat-cnj-def*
by (*cases M*) *auto*

Matrix adjoint - the conjugate traspose matrix ($A^* = \overline{A^t}$)

definition *mat-adj* **where**
mat-adj A = *mat-cnj* (*mat-transpose A*)

lemma *mat-adj-mult-mm* [*simp*]:
shows *mat-adj* (*A* $*_{mm}$ *B*) = *mat-adj B* $*_{mm}$ *mat-adj A*
by (*cases A*, *cases B*) (*auto simp add*: *mat-adj-def mat-cnj-def*)

lemma *mat-adj-mult-sm* [*simp*]:
shows *mat-adj* (*k* $*_{sm}$ *A*) = *cnj k* $*_{sm}$ *mat-adj A*
by (*cases A*) (*auto simp add*: *mat-adj-def mat-cnj-def*)

lemma *mat-det-adj*:
shows *mat-det* (*mat-adj A*) = *cnj* (*mat-det A*)
by (*cases A*) (*auto simp add*: *mat-adj-def mat-cnj-def*)

lemma *mat-adj-inv*:
assumes *mat-det M* ≠ *0*
shows *mat-adj* (*mat-inv M*) = *mat-inv* (*mat-adj M*)
by (*cases M*) (*auto simp add*: *mat-adj-def mat-cnj-def*)

lemma *mat-transpose-mat-cnj*:
shows *mat-transpose* (*mat-cnj A*) = *mat-adj A*
by (*cases A*) (*auto simp add*: *mat-adj-def mat-cnj-def*)

lemma *mat-adj-adj* [*simp*]:
shows *mat-adj* (*mat-adj A*) = *A*
unfolding *mat-adj-def*
by (*subst mat-transpose-mat-cnj*) (*simp add*: *mat-adj-def*)

lemma *mat-adj-eye* [*simp*]:
shows *mat-adj eye* = *eye*
by (*auto simp add*: *mat-adj-def mat-cnj-def*)

Matrix trace

fun *mat-trace* **where**
mat-trace (*a*, *b*, *c*, *d*) = *a* + *d*

Multiplication of matrix and a vector

fun *mult-mv* :: *complex-mat* ⇒ *complex-vec* ⇒ *complex-vec* (**infixl** ‹$*_{mv}$› *100*) **where**
(*a*, *b*, *c*, *d*) $*_{mv}$ (*x*, *y*) = (*x*∗*a* + *y*∗*b*, *x*∗*c* + *y*∗*d*)

fun *mult-vm* :: *complex-vec* ⇒ *complex-mat* ⇒ *complex-vec* (**infixl** ‹$*_{vm}$› *100*) **where**
(*x*, *y*) $*_{vm}$ (*a*, *b*, *c*, *d*) = (*x*∗*a* + *y*∗*c*, *x*∗*b* + *y*∗*d*)

lemma *eye-mv-l* [*simp*]:
shows *eye* $*_{mv}$ *v* = *v*
by (*cases v*) *simp*

lemma *mult-mv-mv* [*simp*]:
shows *B* $*_{mv}$ (*A* $*_{mv}$ *v*) = (*B* $*_{mm}$ *A*) $*_{mv}$ *v*
by (*cases v*, *cases A*, *cases B*) (*auto simp add*: *field-simps*)

lemma *mult-vm-vm* [*simp*]:
shows (*v* $*_{vm}$ *A*) $*_{vm}$ *B* = *v* $*_{vm}$ (*A* $*_{mm}$ *B*)
by (*cases v*, *cases A*, *cases B*) (*auto simp add*: *field-simps*)

lemma *mult-mv-inv*:
assumes *x* = *A* $*_{mv}$ *y* **and** *mat-det A* ≠ *0*
shows *y* = (*mat-inv A*) $*_{mv}$ *x*
using *assms*
by (*cases y*) (*simp add*: *mat-inv-l*)

lemma *mult-vm-inv*:
assumes *x* = *y* $*_{vm}$ *A* **and** *mat-det A* ≠ *0*
shows *y* = *x* $*_{vm}$ (*mat-inv A*)
using *assms*
by (*cases y*) (*simp add*: *mat-inv-r*)

lemma *mult-mv-cancel-l*:

assumes *mat-det A* $\neq$ *0* **and** *A* $*_{mv}$ *v* = *A* $*_{mv}$ *v′*
shows *v* = *v′*
using *assms*
using *mult-mv-inv*
by *blast*

lemma *mult-vm-cancel-r*:
assumes *mat-det A* $\neq$ *0* **and** *v* $*_{vm}$ *A* = *v′* $*_{vm}$ *A*
shows *v* = *v′*
using *assms*
using *mult-vm-inv*
by *blast*

lemma *vec-zero-l* [*simp*]:
shows *A* $*_{mv}$ *vec-zero* = *vec-zero*
by (*cases A*) *simp*

lemma *vec-zero-r* [*simp*]:
shows *vec-zero* $*_{vm}$ *A* = *vec-zero*
by (*cases A*) *simp*

lemma *mult-mv-nonzero*:
assumes *v* $\neq$ *vec-zero* **and** *mat-det A* $\neq$ *0*
shows *A* $*_{mv}$ *v* $\neq$ *vec-zero*
apply (*rule ccontr*)
using *assms mult-mv-inv*[*of vec-zero A v*] *mat-inv-l vec-zero-l*
by *auto*

lemma *mult-vm-nonzero*:
assumes *v* $\neq$ *vec-zero* **and** *mat-det A* $\neq$ *0*
shows *v* $*_{vm}$ *A* $\neq$ *vec-zero*
apply (*rule ccontr*)
using *assms mult-vm-inv*[*of vec-zero v A*] *mat-inv-r vec-zero-r*
by *auto*

lemma *mult-sv-mv*:
shows *k* $*_{sv}$ (*A* $*_{mv}$ *v*) = (*A* $*_{mv}$ (*k* $*_{sv}$ *v*))
by (*cases A*, *cases v*) (*simp add*: *field-simps*)

lemma *mult-mv-mult-vm*:
shows *A* $*_{mv}$ *x* = *x* $*_{vm}$ (*mat-transpose A*)
by (*cases A*, *cases x*) *auto*

lemma *mult-mv-vv*:
shows *A* $*_{mv}$ *v1* $*_{vv}$ *v2* = *v1* $*_{vv}$ (*mat-transpose A* $*_{mv}$ *v2*)
by (*cases v1*, *cases v2*, *cases A*) (*auto simp add*: *field-simps*)

lemma *mult-vv-mv*:
shows *x* $*_{vv}$ (*A* $*_{mv}$ *y*) = (*x* $*_{vm}$ *A*) $*_{vv}$ *y*
by (*cases x*, *cases y*, *cases A*) (*auto simp add*: *field-simps*)

lemma *vec-cnj-mult-mv*:
shows *vec-cnj* (*A* $*_{mv}$ *x*) = (*mat-cnj A*) $*_{mv}$ (*vec-cnj x*)
by (*cases A*, *cases x*) (*auto simp add*: *vec-cnj-def mat-cnj-def*)

lemma *vec-cnj-mult-vm*:
shows *vec-cnj* (*v* $*_{vm}$ *A*) = *vec-cnj v* $*_{vm}$ *mat-cnj A*
unfolding *vec-cnj-def mat-cnj-def*
by (*cases A*, *cases v*, *auto*)

3.8.3 Eigenvalues and eigenvectors

definition *eigenpair* **where**
[*simp*]: *eigenpair k v H* $\longleftrightarrow$ *v* $\neq$ *vec-zero* $\wedge$ *H* $*_{mv}$ *v* = *k* $*_{sv}$ *v*

definition *eigenval* **where**

```
  [simp]: eigenval k H ⟷ (∃ v. v ≠ vec-zero ∧ H *mv v = k *sv v)

lemma eigen-equation:
  shows eigenval k H ⟷ k^2 − mat-trace H * k + mat-det H = 0 (is ?lhs ⟷ ?rhs)
proof−
  obtain A B C D where HH: H = (A, B, C, D)
    by (cases H) auto
  show ?thesis
  proof
    assume ?lhs
    then obtain v where v ≠ vec-zero H *mv v = k *sv v
      unfolding eigenval-def
      by blast
    obtain v1 v2 where vv: v = (v1, v2)
      by (cases v) auto
    from ‹H *mv v = k *sv v› have (H −mm (k *sm eye)) *mv v = vec-zero
      using HH vv
      by (auto simp add: field-simps)
    hence mat-det (H −mm (k *sm eye)) = 0
      using ‹v ≠ vec-zero› vv HH
      using regular-homogenous-system[of A − k B C D − k v1 v2]
      unfolding det2-def
      by (auto simp add: field-simps)
    thus ?rhs
      using HH
      by (auto simp add: power2-eq-square field-simps)
  next
    assume ?rhs
    hence *: mat-det (H −mm (k *sm eye)) = 0
      using HH
      by (auto simp add: field-simps power2-eq-square)
    show ?lhs
    proof (cases H −mm (k *sm eye) = mat-zero)
      case True
      thus ?thesis
        using HH
        by (auto) (rule-tac x=1 in exI, simp)
    next
      case False
      hence (A − k ≠ 0 ∨ B ≠ 0) ∨ (D − k ≠ 0 ∨ C ≠ 0)
        using HH
        by auto
      thus ?thesis
      proof
        assume A − k ≠ 0 ∨ B ≠ 0
        hence C * B + (D − k) * (k − A) = 0
          using * singular-system[of A−k D−k B C (0, 0) 0 0  (B, k−A)] HH
          by (auto simp add: field-simps)
        hence  (B, k−A) ≠ vec-zero (H −mm (k *sm eye)) *mv (B, k−A) = vec-zero
          using HH ‹A − k ≠ 0 ∨ B ≠ 0›
          by (auto simp add: field-simps)
        then obtain v where v ≠ vec-zero ∧ (H −mm (k *sm eye)) *mv v = vec-zero
          by blast
        thus ?thesis
          using HH
          unfolding eigenval-def
          by (rule-tac x=v in exI) (case-tac v, simp add: field-simps)
      next
        assume D − k ≠ 0 ∨ C ≠ 0
        hence C * B + (D − k) * (k − A) = 0
          using * singular-system[of D−k A−k C B (0, 0) 0 0  (C, k−D)] HH
          by (auto simp add: field-simps)
        hence  (k−D, C) ≠ vec-zero (H −mm (k *sm eye)) *mv (k−D, C) = vec-zero
          using HH ‹D − k ≠ 0 ∨ C ≠ 0›
          by (auto simp add: field-simps)
        then obtain v where v ≠ vec-zero ∧ (H −mm (k *sm eye)) *mv v = vec-zero
```

```
        by blast
      thus ?thesis
        using HH
        unfolding eigenval-def
        by (rule-tac x=v in exI) (case-tac v, simp add: field-simps)
    qed
  qed
 qed
qed
```

3.8.4 Bilinear and Quadratic forms, Congruence, and Similarity

Bilinear forms

```
definition bilinear-form where
 [simp]: bilinear-form v1 v2 H = (vec-cnj v1) *_{vm} H *_{vv} v2

lemma bilinear-form-scale-m:
 shows bilinear-form v1 v2 (k *_{sm} H) = k * bilinear-form v1 v2 H
 by (cases v1, cases v2, cases H) (simp add: vec-cnj-def field-simps)

lemma bilinear-form-scale-v1:
 shows bilinear-form (k *_{sv} v1) v2 H = cnj k * bilinear-form v1 v2 H
 by (cases v1, cases v2, cases H) (simp add: vec-cnj-def field-simps)

lemma bilinear-form-scale-v2:
 shows bilinear-form  v1 (k *_{sv} v2) H = k * bilinear-form v1 v2 H
 by (cases v1, cases v2, cases H) (simp add: vec-cnj-def field-simps)
```

Quadratic forms

```
definition quad-form where
 [simp]: quad-form v H = (vec-cnj v) *_{vm} H *_{vv} v

lemma quad-form-bilinear-form:
 shows quad-form v H = bilinear-form v v H
 by simp

lemma quad-form-scale-v:
 shows quad-form (k *_{sv} v) H = cor ((cmod k)^2) * quad-form v H
 using bilinear-form-scale-v1 bilinear-form-scale-v2
 by (simp add: complex-mult-cnj-cmod field-simps)

lemma quad-form-scale-m:
 shows quad-form v (k *_{sm} H) = k * quad-form v H
 using bilinear-form-scale-m
 by simp

lemma cnj-quad-form [simp]:
 shows cnj (quad-form z H) = quad-form z (mat-adj H)
 by (cases H, cases z) (auto simp add: mat-adj-def mat-cnj-def vec-cnj-def field-simps)
```

Matrix congruence

Two matrices are congruent iff they represent the same quadratic form with respect to different bases (for example if one circline can be transformed to another by a Möbius trasformation).

```
definition congruence where
 [simp]: congruence M H ≡ mat-adj M *_{mm} H *_{mm} M

lemma congruence-nonzero:
 assumes H ≠ mat-zero and mat-det M ≠ 0
 shows congruence M H ≠ mat-zero
 using assms
 unfolding congruence-def
 by (subst mult-mm-non-zero-r, subst mult-mm-non-zero-l) (auto simp add: mat-det-adj)

lemma congruence-congruence:
```

shows *congruence M1 (congruence M2 H) = congruence (M2 $*_{mm}$ M1) H*
unfolding *congruence-def*
apply (*subst mult-mm-assoc*)
apply (*subst mult-mm-assoc*)
apply (*subst mat-adj-mult-mm*)
apply (*subst mult-mm-assoc*)
by *simp*

lemma *congruence-eye* [*simp*]:
shows *congruence eye H = H*
by (*cases H*) (*simp add*: *mat-adj-def mat-cnj-def*)

lemma *congruence-congruence-inv* [*simp*]:
assumes *mat-det M ≠ 0*
shows *congruence M (congruence (mat-inv M) H) = H*
using *assms congruence-congruence*[*of M mat-inv M H*]
using *mat-inv-l*[*of M*] *mat-eye-l mat-eye-r*
unfolding *congruence-def*
by (*simp del*: *eye-def*)

lemma *congruence-inv*:
assumes *mat-det M ≠ 0* **and** *congruence M H = H′*
shows *congruence (mat-inv M) H′ = H*
using *assms*
using ‹*mat-det M ≠ 0*› *mult-mm-inv-l*[*of mat-adj M H $*_{mm}$ M H′*]
using *mult-mm-inv-r*[*of M H mat-inv (mat-adj M) $*_{mm}$ H′*]
by (*simp add*: *mat-det-adj mult-mm-assoc mat-adj-inv*)

lemma *congruence-scale-m* [*simp*]:
shows *congruence M (k $*_{sm}$ H) = k $*_{sm}$ (congruence M H)*
by (*cases M*, *cases H*) (*auto simp add*: *mat-adj-def mat-cnj-def field-simps*)

lemma *inj-congruence*:
assumes *mat-det M ≠ 0* **and** *congruence M H = congruence M H′*
shows *H = H′*
proof−
have *H $*_{mm}$ M = H′ $*_{mm}$ M*
using *assms*
using *mult-mm-cancel-l*[*of mat-adj M H $*_{mm}$ M H′ $*_{mm}$ M*]
by (*simp add*: *mat-det-adj mult-mm-assoc*)
thus *?thesis*
using *assms*
using *mult-mm-cancel-r*[*of M H H′*]
by *simp*
qed

lemma *mat-det-congruence* [*simp*]:
mat-det (congruence M H) = (cor ((cmod (mat-det M))^2)) ∗ mat-det H
using *complex-mult-cnj-cmod*[*of mat-det M*]
by (*auto simp add*: *mat-det-adj field-simps*)

lemma *det-sgn-congruence* [*simp*]:
assumes *mat-det M ≠ 0*
shows *sgn (mat-det (congruence M H)) = sgn (mat-det H)*
using *assms*
by (*subst mat-det-congruence*, *auto simp add*: *sgn-mult power2-eq-square*) (*simp add*: *sgn-of-real*)

lemma *Re-det-sgn-congruence* [*simp*]:
assumes *mat-det M ≠ 0*
shows *sgn (Re (mat-det (congruence M H))) = sgn (Re (mat-det H))*
proof−
have ∗: *Re (mat-det (congruence M H)) = (cmod (mat-det M))^2 ∗ Re (mat-det H)*
by (*subst mat-det-congruence*, *subst Re-mult-real*, *rule Im-complex-of-real*) (*subst Re-complex-of-real*, *simp*)
show *?thesis*
using *assms*
by (*subst* ∗) (*auto simp add*: *sgn-mult*)

qed

Transforming a matrix H by a regular matrix M preserves its bilinear and quadratic forms.

lemma *bilinear-form-congruence* [*simp*]:
assumes *mat-det M* $\neq$ *0*
shows *bilinear-form* (*M* $*_{mv}$ *v1*) (*M* $*_{mv}$ *v2*) (*congruence* (*mat-inv M*) *H*) =
bilinear-form v1 v2 H
proof−
have *mat-det* (*mat-adj M*) $\neq$ *0*
using *assms*
by (*simp add*: *mat-det-adj*)
show *?thesis*
unfolding *bilinear-form-def congruence-def*
apply (*subst mult-mv-mult-vm*)
apply (*subst vec-cnj-mult-vm*)
apply (*subst mat-adj-def*[*symmetric*])
apply (*subst mult-vm-vm*)
apply (*subst mult-vv-mv*)
apply (*subst mult-vm-vm*)
apply (*subst mat-adj-inv*[*OF* ‹*mat-det M* $\neq$ *0*›])
apply (*subst mult-assoc-5*)
apply (*subst mat-inv-r*[*OF* ‹*mat-det* (*mat-adj M*) $\neq$ *0*›])
apply (*subst mat-inv-l*[*OF* ‹*mat-det M* $\neq$ *0*›])
apply (*subst mat-eye-l*, *subst mat-eye-r*)
by *simp*
qed

lemma *quad-form-congruence* [*simp*]:
assumes *mat-det M* $\neq$ *0*
shows *quad-form* (*M* $*_{mv}$ *z*) (*congruence* (*mat-inv M*) *H*) = *quad-form z H*
using *bilinear-form-congruence*[*OF assms*]
by *simp*

Similar matrices

Two matrices are similar iff they represent the same linear operator with respect to (possibly) different bases (e.g., if they represent the same Möbius transformation after changing the coordinate system)

definition *similarity* **where**
similarity A M = *mat-inv A* $*_{mm}$ *M* $*_{mm}$ *A*

lemma *mat-det-similarity* [*simp*]:
assumes *mat-det A* $\neq$ *0*
shows *mat-det* (*similarity A M*) = *mat-det M*
using *assms*
unfolding *similarity-def*
by (*simp add*: *mat-det-inv*)

lemma *mat-trace-similarity* [*simp*]:
assumes *mat-det A* $\neq$ *0*
shows *mat-trace* (*similarity A M*) = *mat-trace M*
proof−
obtain *a b c d* **where** *AA*: *A* = (*a*, *b*, *c*, *d*)
by (*cases A*) *auto*
obtain *mA mB mC mD* **where** *MM*: *M* = (*mA*, *mB*, *mC*, *mD*)
by (*cases M*) *auto*
have *mA* $*$ (*a* $*$ *d*) / (*a* $*$ *d* − *b* $*$ *c*) + *mD* $*$ (*a* $*$ *d*) / (*a* $*$ *d* − *b* $*$ *c*) =
mA + *mD* + *mA* $*$ (*b* $*$ *c*) / (*a* $*$ *d* − *b* $*$ *c*) + *mD* $*$ (*b* $*$ *c*) / (*a* $*$ *d* − *b* $*$ *c*)
using *assms AA*
by (*simp add*: *field-simps*)
thus *?thesis*
using *AA MM*
by (*simp add*: *field-simps similarity-def*)
qed

lemma *similarity-eye* [*simp*]:
shows *similarity eye M* = *M*

unfolding *similarity-def*
using *mat-eye-l mat-eye-r*
by *auto*

lemma *similarity-eye′* [*simp*]:
shows *similarity* (*1*, *0*, *0*, *1*) *M* = *M*
unfolding *eye-def*[*symmetric*]
by (*simp del*: *eye-def*)

lemma *similarity-comp* [*simp*]:
assumes *mat-det A1* ≠ *0* **and** *mat-det A2* ≠ *0*
shows *similarity A1* (*similarity A2 M*) = *similarity* (*A2*∗$_{mm}$*A1*) *M*
using *assms*
unfolding *similarity-def*
by (*simp add*: *mult-mm-assoc mat-inv-mult-mm*)

lemma *similarity-inv*:
assumes *similarity A M1* = *M2* **and** *mat-det A* ≠ *0*
shows *similarity* (*mat-inv A*) *M2* = *M1*
using *assms*
unfolding *similarity-def*
by (*metis mat-det-mult mult-mm-assoc mult-mm-inv-l mult-mm-inv-r mult-zero-left*)

end

3.9 Generalized Unitary Matrices

theory *Unitary-Matrices*
imports *Matrices More-Complex*
begin

In this section (generalized) 2 × 2 unitary matrices are introduced.

Unitary matrices

definition *unitary* **where**
unitary M ⟷ *mat-adj M* ∗$_{mm}$ *M* = *eye*

Generalized unitary matrices

definition *unitary-gen* **where**
unitary-gen M ⟷
(∃ *k*::*complex*. *k* ≠ *0* ∧ *mat-adj M* ∗$_{mm}$ *M* = *k* ∗$_{sm}$ *eye*)

Scalar can be always be a positive real

lemma *unitary-gen-real*:
assumes *unitary-gen M*
shows (∃ *k*::*real*. *k* > *0* ∧ *mat-adj M* ∗$_{mm}$ *M* = *cor k* ∗$_{sm}$ *eye*)
proof−
obtain *k* **where** ∗: *mat-adj M* ∗$_{mm}$ *M* = *k* ∗$_{sm}$ *eye k* ≠ *0*
using *assms*
by (*auto simp add*: *unitary-gen-def*)
obtain *a b c d* **where** *M* = (*a*, *b*, *c*, *d*)
by (*cases M*) *auto*
hence *k* = *cor* ((*cmod a*)^2) + *cor* ((*cmod c*)^2)
using ∗
by (*subst complex-mult-cnj-cmod*[*symmetric*])+ (*auto simp add*: *mat-adj-def mat-cnj-def*)
hence *is-real k* ∧ *Re k* > *0*
using ‹*k* ≠ *0*›
by (*smt* (*verit*) *add-cancel-left-left arg-0-iff arg-complex-of-real-positive not-sum-power2-lt-zero of-real-0 plus-complex.simps*(*1*) *plus-complex.simps*(*2*))
thus *?thesis*
using ∗
by (*rule-tac x*=*Re k* **in** *exI*) *simp*
qed

Generalized unitary matrices can be factored into a product of a unitary matrix and a real positive scalar multiple of the identity matrix

lemma *unitary-gen-unitary*:
shows *unitary-gen M* $\longleftrightarrow$
(∃ *k M′*. *k* > *0* ∧ *unitary M′* ∧ *M* = (*cor k* $*_{sm}$ *eye*) $*_{mm}$ *M′*) (**is** *?lhs* = *?rhs*)
proof
assume *?lhs*
then obtain *k* **where** ∗: *k*>*0 mat-adj M* $*_{mm}$ *M* = *cor k* $*_{sm}$ *eye*
using *unitary-gen-real*[*of M*]
by *auto*

let *?k′* = *cor* (*sqrt k*)
have *?k′* ∗ *cnj ?k′* = *cor k*
using ‹*k* > *0*›
by *simp*
moreover
have *Re ?k′* > *0 is-real ?k′ ?k′* ≠ *0*
using ‹*k* > *0*›
by *auto*
ultimately
show *?rhs*
using ∗ *mat-eye-l*
unfolding *unitary-gen-def unitary-def*
by (*rule-tac x*=*Re ?k′* **in** *exI*) (*rule-tac x*=(*1* / *?k′*)$*_{sm}$*M* **in** *exI*, *simp add*: *mult-sm-mm*[*symmetric*])
next
assume *?rhs*
then obtain *k M′* **where** *k* > *0 unitary M′ M* = (*cor k* $*_{sm}$ *eye*) $*_{mm}$ *M′*
by *blast*
hence *M* = *cor k* $*_{sm}$ *M′*
using *mult-sm-mm*[*of cor k eye M′*] *mat-eye-l*
by *simp*
thus *?lhs*
using ‹*unitary M′*› ‹*k* > *0*›
by (*simp add*: *unitary-gen-def unitary-def*)
qed

When they represent Möbius transformations, eneralized unitary matrices fix the imaginary unit circle. Therefore, they fix a Hermitean form with (2, 0) signature (two positive and no negative diagonal elements).

lemma *unitary-gen-iff′*:
shows *unitary-gen M* $\longleftrightarrow$
(∃ *k*::*complex*. *k* ≠ *0* ∧ *congruence M* (*1*, *0*, *0*, *1*) = *k* $*_{sm}$ (*1*, *0*, *0*, *1*))
unfolding *unitary-gen-def*
using *mat-eye-r*
by (*auto simp add*: *mult.assoc*)

Unitary matrices are special cases of general unitary matrices

lemma *unitary-unitary-gen* [*simp*]:
assumes *unitary M*
shows *unitary-gen M*
using *assms*
unfolding *unitary-gen-def unitary-def*
by *auto*

Generalized unitary matrices are regular

lemma *unitary-gen-regular*:
assumes *unitary-gen M*
shows *mat-det M* ≠ *0*
proof−
from *assms* **obtain** *k* **where**
k ≠ *0 mat-adj M* $*_{mm}$ *M* = *k* $*_{sm}$ *eye*
unfolding *unitary-gen-def*
by *auto*
hence *mat-det* (*mat-adj M* $*_{mm}$ *M*) ≠ *0*
by *simp*
thus *?thesis*
by (*simp add*: *mat-det-adj*)
qed

lemmas *unitary-regular* = *unitary-gen-regular*[*OF unitary-unitary-gen*]

3.9.1 Group properties

Generalized 2×2 unitary matrices form a group under multiplication (usually denoted by $GU(2, \mathbb{C})$). The group is closed under non-zero complex scalar multiplication. Since these matrices are always regular, they form a subgroup of general linear group (usually denoted by $GL(2, \mathbb{C})$) of all regular matrices.

lemma *unitary-gen-scale* [*simp*]:
assumes *unitary-gen M* **and** $k \neq 0$
shows *unitary-gen* ($k *_{sm} M$)
using *assms*
unfolding *unitary-gen-def*
by *auto*

lemma *unitary-comp*:
assumes *unitary M1* **and** *unitary M2*
shows *unitary* (*M1* $*_{mm}$ *M2*)
using *assms*
unfolding *unitary-def*
by (*metis mat-adj-mult-mm mat-eye-l mult-mm-assoc*)

lemma *unitary-gen-comp*:
assumes *unitary-gen M1* **and** *unitary-gen M2*
shows *unitary-gen* (*M1* $*_{mm}$ *M2*)
proof −
obtain *k1 k2* **where** $*$: *k1* $*$ *k2* $\neq$ *0 mat-adj M1* $*_{mm}$ *M1* = *k1* $*_{sm}$ *eye mat-adj M2* $*_{mm}$ *M2* = *k2* $*_{sm}$ *eye*
using *assms*
unfolding *unitary-gen-def*
by *auto*
have *mat-adj M2* $*_{mm}$ *mat-adj M1* $*_{mm}$ (*M1* $*_{mm}$ *M2*) = *mat-adj M2* $*_{mm}$ (*mat-adj M1* $*_{mm}$ *M1*) $*_{mm}$ *M2*
by (*auto simp add*: *mult-mm-assoc*)
also have ... = *mat-adj M2* $*_{mm}$ ((*k1* $*_{sm}$ *eye*) $*_{mm}$ *M2*)
using $*$
by (*auto simp add*: *mult-mm-assoc*)
also have ... = *mat-adj M2* $*_{mm}$ (*k1* $*_{sm}$ *M2*)
using *mult-sm-eye-mm*[*of k1 M2*]
by (*simp del*: *eye-def*)
also have ... = *k1* $*_{sm}$ (*k2* $*_{sm}$ *eye*)
using $*$
by *auto*
finally
show *?thesis*
using $*$
unfolding *unitary-gen-def*
by (*rule-tac x*=*k1*$*$*k2* **in** *exI*, *simp del*: *eye-def*)
qed

lemma *unitary-adj-eq-inv*:
shows *unitary M* $\longleftrightarrow$ *mat-det M* $\neq$ *0* $\wedge$ *mat-adj M* = *mat-inv M*
using *unitary-regular*[*of M*] *mult-mm-inv-r*[*of M mat-adj M eye*] *mat-eye-l*[*of mat-inv M*] *mat-inv-l*[*of M*]
unfolding *unitary-def*
by − (*rule*, *simp-all*)

lemma *unitary-inv*:
assumes *unitary M*
shows *unitary* (*mat-inv M*)
using *assms*
unfolding *unitary-adj-eq-inv*
using *mat-adj-inv*[*of M*] *mat-det-inv*[*of M*]
by *simp*

lemma *unitary-gen-inv*:
assumes *unitary-gen M*
shows *unitary-gen* (*mat-inv M*)
proof −

obtain $k\ M'$ **where** $0 < k$ *unitary* M' $M = cor\ k *_{sm} eye *_{mm} M'$
using *unitary-gen-unitary*[*of* M] *assms*
by *blast*
hence *mat-inv* $M = cor\ (1/k) *_{sm}$ *mat-inv* M'
by (*metis mat-inv-mult-sm mult-sm-eye-mm norm-not-less-zero of-real-1 of-real-divide of-real-eq-0-iff sgn-1-neg sgn-greater sgn-if sgn-pos sgn-sgn*)
thus *?thesis*
using ‹$k > 0$› ‹*unitary* M'›
by (*subst unitary-gen-unitary*[*of mat-inv* M]) (*rule-tac* $x=1/k$ **in** *exI*, *rule-tac* x=*mat-inv* M' **in** *exI*, *metis divide-pos-pos mult-sm-eye-mm unitary-inv zero-less-one*)
qed

3.9.2 The characterization in terms of matrix elements

Special matrices are those having the determinant equal to 1. We first give their characterization.

lemma *unitary-special*:
assumes *unitary* M **and** *mat-det* $M = 1$
shows $\exists\ a\ b.\ M = (a,\ b,\ -cnj\ b,\ cnj\ a)$
proof−
have *mat-adj* M = *mat-inv* M
using *assms mult-mm-inv-r*[*of* M *mat-adj* M *eye*] *mat-eye-r mat-eye-l*
by (*simp add*: *unitary-def*)
thus *?thesis*
using ‹*mat-det* $M = 1$›
by (*cases* M) (*auto simp add*: *mat-adj-def mat-cnj-def*)
qed

lemma *unitary-gen-special*:
assumes *unitary-gen* M **and** *mat-det* $M = 1$
shows $\exists\ a\ b.\ M = (a,\ b,\ -cnj\ b,\ cnj\ a)$
proof−
from *assms*
obtain k **where** $*$: $k \neq 0$ *mat-adj* $M *_{mm} M = k *_{sm} eye$
unfolding *unitary-gen-def*
by *auto*
hence *mat-det* (*mat-adj* $M *_{mm} M$) = $k*k$
by *simp*
hence $k*k = 1$
using *assms*(*2*)
by (*simp add*: *mat-det-adj*)
hence $k = 1 \vee k = -1$
using *square-eq-1-iff*[*of* k]
by *simp*
moreover
have *mat-adj* $M = k *_{sm}$ *mat-inv* M
using $*$
using *assms mult-mm-inv-r*[*of* M *mat-adj* M $k *_{sm} eye$] *mat-eye-r mat-eye-l*
by *simp* (*metis mult-sm-eye-mm* $*$(*2*))
moreover
obtain $a\ b\ c\ d$ **where** $M = (a,\ b,\ c,\ d)$
by (*cases* M) *auto*
ultimately
have $M = (a,\ b,\ -cnj\ b,\ cnj\ a) \vee M = (a,\ b,\ cnj\ b,\ -cnj\ a)$
using *assms*(*2*)
by (*auto simp add*: *mat-adj-def mat-cnj-def*)
moreover
have *Re* $(-\ (cor\ (cmod\ a))^2 - (cor\ (cmod\ b))^2) < 1$
by (*smt* (*verit*) *cmod-square complex-norm-square minus-complex.simps*(*1*) *of-real-power realpow-square-minus-le uminus-complex.simps*(*1*))
hence $-\ (cor\ (cmod\ a))^2 - (cor\ (cmod\ b))^2 \neq 1$
by *force*
hence $M \neq (a,\ b,\ cnj\ b,\ -cnj\ a)$
using ‹*mat-det* $M = 1$› *complex-mult-cnj-cmod*[*of* a] *complex-mult-cnj-cmod*[*of* b]
by *auto*
ultimately
show *?thesis*

by *auto*
qed

A characterization of all generalized unitary matrices

lemma *unitary-gen-iff*:
shows *unitary-gen M* $\longleftrightarrow$
(∃ *a b k*. *k* ≠ *0* ∧ *mat-det* (*a*, *b*, −*cnj b*, *cnj a*) ≠ *0* ∧
M = *k* $*_{sm}$ (*a*, *b*, −*cnj b*, *cnj a*)) (**is** *?lhs* = *?rhs*)
proof
assume *?lhs*
obtain *d* **where** ∗: *d*∗*d* = *mat-det M*
using *ex-complex-sqrt*
by *auto*
hence *d* ≠ *0*
using *unitary-gen-regular*[*OF* ‹*unitary-gen M*›]
by *auto*
from ‹*unitary-gen M*›
obtain *k* **where** *k* ≠ *0 mat-adj M* $*_{mm}$ *M* = *k* $*_{sm}$ *eye*
unfolding *unitary-gen-def*
by *auto*
hence *mat-adj* ((*1*/*d*)$*_{sm}$*M*) $*_{mm}$ ((*1*/*d*)$*_{sm}$*M*) = (*k* / (*d*∗*cnj d*)) $*_{sm}$ *eye*
by *simp*
obtain *a b* **where** (*a*, *b*, − *cnj b*, *cnj a*) = (*1* / *d*) $*_{sm}$ *M*
using *unitary-gen-special*[*of* (*1* / *d*) $*_{sm}$ *M*] ‹*unitary-gen M*› ∗ *unitary-gen-regular*[*of M*] ‹*d* ≠ *0*›
by *force*
moreover
hence *mat-det* (*a*, *b*, − *cnj b*, *cnj a*) ≠ *0*
using *unitary-gen-regular*[*OF* ‹*unitary-gen M*›] ‹*d* ≠ *0*›
by *auto*
ultimately
show *?rhs*
apply (*rule-tac x*=*a* **in** *exI*, *rule-tac x*=*b* **in** *exI*, *rule-tac x*=*d* **in** *exI*)
using *mult-sm-inv-l*[*of 1*/*d M*]
by (*auto simp add*: *field-simps*)
next
assume *?rhs*
then obtain *a b k* **where** *k* ≠ *0* ∧ *mat-det* (*a*, *b*, − *cnj b*, *cnj a*) ≠ *0* ∧ *M* = *k* $*_{sm}$ (*a*, *b*, − *cnj b*, *cnj a*)
by *auto*
thus *?lhs*
unfolding *unitary-gen-def*
apply (*auto simp add*: *mat-adj-def mat-cnj-def*)
using *mult-eq-0-iff*[*of cnj k* ∗ *k cnj a* ∗ *a* + *cnj b* ∗ *b*]
by (*auto simp add*: *field-simps*)
qed

A characterization of unitary matrices

lemma *unitary-iff*:
shows *unitary M* $\longleftrightarrow$
(∃ *a b k*. (*cmod a*)^2 + (*cmod b*)^2 ≠ *0* ∧
(*cmod k*)^2 = *1* / ((*cmod a*)^2 + (*cmod b*)^2) ∧
M = *k* $*_{sm}$ (*a*, *b*, −*cnj b*, *cnj a*)) (**is** *?lhs* = *?rhs*)
proof
assume *?lhs*
obtain *k a b* **where** ∗: *M* = *k* $*_{sm}$ (*a*, *b*, −*cnj b*, *cnj a*) *k* ≠ *0 mat-det* (*a*, *b*, −*cnj b*, *cnj a*) ≠ *0*
using *unitary-gen-iff unitary-unitary-gen*[*OF* ‹*unitary M*›]
by *auto*

have *md*: *mat-det* (*a*, *b*, −*cnj b*, *cnj a*) = *cor* ((*cmod a*)^2 + (*cmod b*)^2)
by (*auto simp add*: *complex-mult-cnj-cmod*)

have *k* ∗ *cnj k* ∗ *mat-det* (*a*, *b*, −*cnj b*, *cnj a*) = *1*
using ‹*unitary M*› ∗
unfolding *unitary-def*
by (*auto simp add*: *mat-adj-def mat-cnj-def field-simps*)
hence (*cmod k*)^2 ∗ ((*cmod a*)^2 + (*cmod b*)^2) = *1*
by (*metis* (*mono-tags*, *lifting*) *complex-norm-square md of-real-1 of-real-eq-iff of-real-mult*)

thus *?rhs*
using ∗ *mat-eye-l*
apply (*rule-tac x=a* **in** *exI*, *rule-tac x=b* **in** *exI*, *rule-tac x=k* **in** *exI*)
apply (*auto simp add*: *complex-mult-cnj-cmod*)
by (*metis* ‹(*cmod k*)^2 ∗ ((*cmod a*)^2 + (*cmod b*)^2) = *1*› *mult-eq-0-iff nonzero-eq-divide-eq zero-neq-one*)
next
assume *?rhs*
then obtain *a b k* **where** ∗: (*cmod a*)^2 + (*cmod b*)^2 ≠ *0* (*cmod k*)^2 = *1* / ((*cmod a*)^2 + (*cmod b*)^2) *M* = *k* ∗$_{sm}$ (*a*, *b*, −*cnj b*, *cnj a*)
by *auto*
have (*k* ∗ *cnj k*) ∗ (*a* ∗ *cnj a*) + (*k* ∗ *cnj k*) ∗ (*b* ∗ *cnj b*) = *1*
apply (*subst complex-mult-cnj-cmod*)+
using ∗(*1*−*2*)
by (*metis* (*no-types*, *lifting*) *distrib-left nonzero-eq-divide-eq of-real-1 of-real-add of-real-divide of-real-eq-0-iff*)
thus *?lhs*
using ∗
unfolding *unitary-def*
by (*simp add*: *mat-adj-def mat-cnj-def field-simps*)
qed

end

3.10 Generalized unitary matrices with signature $(1, 1)$

theory *Unitary11-Matrices*
imports *Matrices More-Complex*
begin

When acting as Möbius transformations in the extended complex plane, generalized complex 2×2 unitary matrices fix the imaginary unit circle (a Hermitean form with (2, 0) signature). We now describe matrices that fix the ordinary unit circle (a Hermitean form with (1, 1) signature, i.e., one positive and one negative element on the diagonal). These are extremely important for further formalization, since they will represent disc automorphisims and isometries of the Poincaré disc. The development of this theory follows the development of the theory of generalized unitary matrices.

Unitary11 matrices

definition *unitary11* **where**
unitary11 M ⟷ *congruence M* (*1*, *0*, *0*, −*1*) = (*1*, *0*, *0*, −*1*)

Generalized unitary11 matrices

definition *unitary11-gen* **where**
unitary11-gen M ⟷ (∃ *k*. *k* ≠ *0* ∧ *congruence M* (*1*, *0*, *0*, −*1*) = *k* ∗$_{sm}$ (*1*, *0*, *0*, −*1*))

Scalar can always be a non-zero real number

lemma *unitary11-gen-real*:
shows *unitary11-gen M* ⟷ (∃ *k*. *k* ≠ *0* ∧ *congruence M* (*1*, *0*, *0*, −*1*) = *cor k* ∗$_{sm}$ (*1*, *0*, *0*, −*1*))
unfolding *unitary11-gen-def*
proof (*auto simp del*: *congruence-def*)
fix *k*
assume *k* ≠ *0 congruence M* (*1*, *0*, *0*, −*1*) = (*k*, *0*, *0*, − *k*)
hence *mat-det* (*congruence M* (*1*, *0*, *0*, −*1*)) = −*k*∗*k*
by *simp*
moreover
have *is-real* (*mat-det* (*congruence M* (*1*, *0*, *0*, −*1*))) *Re* (*mat-det* (*congruence M* (*1*, *0*, *0*, −*1*))) ≤ *0*
by (*auto simp add*: *mat-det-adj*)
ultimately
have *is-real* (*k*∗*k*) *Re* (−*k*∗*k*) ≤ *0*
by *auto*
hence *is-real* (*k*∗*k*) ∧ *Re* (*k* ∗ *k*) > *0*
using ‹*k* ≠ *0*›
by (*smt* (*verit*) *complex-eq-if-Re-eq mult-eq-0-iff mult-minus-left uminus-complex.simps*(*1*) *zero-complex.simps*(*1*) *zero-complex.simps*(*2*))
hence *is-real k*
by *auto*
thus ∃*ka*. *ka* ≠ *0* ∧ *k* = *cor ka*

using ‹$k \neq 0$›
by (*rule-tac x=Re k* **in** *exI*) (*cases k*, *auto simp add*: *Complex-eq*)
qed

Unitary11 matrices are special cases of generalized unitary 11 matrices

lemma *unitary11-unitary11-gen* [*simp*]:
assumes *unitary11 M*
shows *unitary11-gen M*
using *assms*
unfolding *unitary11-gen-def unitary11-def*
by (*rule-tac x=1* **in** *exI*, *auto*)

All generalized unitary11 matrices are regular

lemma *unitary11-gen-regular*:
assumes *unitary11-gen M*
shows *mat-det M* $\neq$ *0*
proof−
from *assms* **obtain** *k* **where**
$k \neq 0$ *mat-adj M* $*_{mm}$ (*1*, *0*, *0*, -1) $*_{mm}$ *M* = *cor k* $*_{sm}$ (*1*, *0*, *0*, -1)
unfolding *unitary11-gen-real*
by *auto*
hence *mat-det* (*mat-adj M* $*_{mm}$ (*1*, *0*, *0*, -1) $*_{mm}$ *M*) $\neq$ *0*
by *simp*
thus *?thesis*
by (*simp add*: *mat-det-adj*)
qed

lemmas *unitary11-regular* = *unitary11-gen-regular*[*OF unitary11-unitary11-gen*]

3.10.1 The characterization in terms of matrix elements

Special matrices are those having the determinant equal to 1. We first give their characterization.

lemma *unitary11-special*:
assumes *unitary11 M* **and** *mat-det M* = *1*
shows ∃ *a b*. *M* = (*a*, *b*, *cnj b*, *cnj a*)
proof−
have *mat-adj M* $*_{mm}$ (*1*, *0*, *0*, -1) = (*1*, *0*, *0*, -1) $*_{mm}$ *mat-inv M*
using *assms mult-mm-inv-r*
by (*simp add*: *unitary11-def*)
thus *?thesis*
using *assms*(*2*)
by (*cases M*) (*simp add*: *mat-adj-def mat-cnj-def*)
qed

lemma *unitary11-gen-special*:
assumes *unitary11-gen M* **and** *mat-det M* = *1*
shows ∃ *a b*. *M* = (*a*, *b*, *cnj b*, *cnj a*) ∨ *M* = (*a*, *b*, $-$*cnj b*, $-$*cnj a*)
proof−
from *assms*
obtain *k* **where** ∗: $k \neq 0$ *mat-adj M* $*_{mm}$ (*1*, *0*, *0*, -1) $*_{mm}$ *M* = *cor k* $*_{sm}$ (*1*, *0*, *0*, -1)
unfolding *unitary11-gen-real*
by *auto*
hence *mat-det* (*mat-adj M* $*_{mm}$ (*1*, *0*, *0*, -1) $*_{mm}$ *M*) = $-$ *cor k*∗ *cor k*
by *simp*
hence *mat-det* (*mat-adj M* $*_{mm}$ *M*) = *cor k*∗ *cor k*
by *simp*
hence *cor k*∗ *cor k* = *1*
using *assms*(*2*)
by (*simp add*: *mat-det-adj*)
hence *cor k* = *1* ∨ *cor k* = -1
using *square-eq-1-iff*[*of cor k*]
by *simp*
moreover
have *mat-adj M* $*_{mm}$ (*1*, *0*, *0*, -1) = (*cor k* $*_{sm}$ (*1*, *0*, *0*, -1)) $*_{mm}$ *mat-inv M*
using ∗

```
    using assms mult-mm-inv-r mat-eye-r mat-eye-l
    by auto
  moreover
  obtain a b c d where M = (a, b, c, d)
    by (cases M) auto
  ultimately
  have M = (a, b, cnj b, cnj a) ∨ M = (a, b, −cnj b, −cnj a)
    using assms(2)
    by (auto simp add: mat-adj-def mat-cnj-def)
  thus ?thesis
    by auto
qed
```

A characterization of all generalized unitary11 matrices

```
lemma unitary11-gen-iff':
  shows unitary11-gen M ⟷
        (∃ a b k. k ≠ 0 ∧ mat-det (a, b, cnj b, cnj a) ≠ 0 ∧
                  (M = k *sm (a, b, cnj b, cnj a) ∨
                   M = k *sm (−1, 0, 0, 1) *mm (a, b, cnj b, cnj a))) (is ?lhs = ?rhs)
proof
  assume ?lhs
  obtain d where *: d*d = mat-det M
    using ex-complex-sqrt
    by auto
  hence d ≠ 0
    using unitary11-gen-regular[OF ‹unitary11-gen M›]
    by auto
  from ‹unitary11-gen M›
  obtain k where k ≠ 0 mat-adj M *mm (1, 0, 0, −1) *mm M = cor k *sm (1, 0, 0, −1)
    unfolding unitary11-gen-real
    by auto
  hence mat-adj ((1/d)*sm M)*mm (1, 0, 0, −1) *mm ((1/d)*sm M) = (cor k / (d*cnj d)) *sm (1, 0, 0, −1)
    by simp
  moreover
  have is-real (cor k / (d * cnj d))
    by (metis complex-In-mult-cnj-zero div-reals Im-complex-of-real)
  hence cor (Re (cor k / (d * cnj d))) = cor k / (d * cnj d)
    by simp
  ultimately
  have unitary11-gen ((1/d)*sm M)
    unfolding unitary11-gen-real
    using ‹d ≠ 0› ‹k ≠ 0›
    using ‹cor (Re (cor k / (d * cnj d))) = cor k / (d * cnj d)›
    by (rule-tac x=Re (cor k / (d * cnj d)) in exI, auto, simp add: *)
  moreover
  have mat-det ((1 / d) *sm M) = 1
    using * unitary11-gen-regular[of M] ‹unitary11-gen M›
    by auto
  ultimately
  obtain a b where (a, b, cnj b, cnj a) = (1 / d) *sm M ∨ (a, b, −cnj b, −cnj a) = (1 / d) *sm M
    using unitary11-gen-special[of (1 / d) *sm M]
    by force
  thus ?rhs
  proof
    assume (a, b, cnj b, cnj a) = (1 / d) *sm M
    moreover
    hence mat-det (a, b, cnj b, cnj a) ≠ 0
      using unitary11-gen-regular[OF ‹unitary11-gen M›] ‹d ≠ 0›
      by auto
    ultimately
    show ?rhs
      using ‹d ≠ 0›
      by (rule-tac x=a in exI, rule-tac x=b in exI, rule-tac x=d in exI, simp)
  next
    assume *: (a, b, −cnj b, −cnj a) = (1 / d) *sm M
    hence (1 / d) *sm M = (a, b, −cnj b, −cnj a)
```

```
    by simp
  hence M = (a * d, b * d, - (d * cnj b), - (d * cnj a))
    using ‹d ≠ 0›
    using mult-sm-inv-l[of 1/d M (a, b, -cnj b, -cnj a), symmetric]
    by (simp add: field-simps)
  moreover
  have mat-det (a, b, -cnj b, -cnj a) ≠ 0
    using * unitary11-gen-regular[OF ‹unitary11-gen M›] ‹d ≠ 0›
    by auto
  ultimately
  show ?thesis
    using ‹d ≠ 0›
    by (rule-tac x=a in exI, rule-tac x=b in exI, rule-tac x=-d in exI) (simp add: field-simps)
 qed
next
 assume ?rhs
 then obtain a b k where k ≠ 0 mat-det (a, b, cnj b, cnj a) ≠ 0
  M = k *sm (a, b, cnj b, cnj a) ∨ M = k *sm (-1, 0, 0, 1) *mm (a, b, cnj b, cnj a)
  by auto
 moreover
 let ?x = cnj k * cnj a * (k * a) + - (cnj k * b * (k * cnj b))
 have ?x = (k*cnj k)*(a*cnj a - b*cnj b)
  by (auto simp add: field-simps)
 hence is-real ?x
  by simp
 hence cor (Re ?x) = ?x
  by (rule complex-of-real-Re)
 moreover
 have ?x ≠ 0
  using mult-eq-0-iff[of cnj k * k (cnj a * a + - cnj b * b)]
  using ‹mat-det (a, b, cnj b, cnj a) ≠ 0› ‹k ≠ 0›
  by (auto simp add: field-simps)
 hence Re ?x ≠ 0
  using ‹is-real ?x›
  by (metis calculation(4) of-real-0)
 ultimately
 show ?lhs
  unfolding unitary11-gen-real
  by (rule-tac x=Re ?x in exI) (auto simp add: mat-adj-def mat-cnj-def)
qed
```

Another characterization of all generalized unitary11 matrices. They are products of rotation and Blaschke factor matrices.

```
lemma unitary11-gen-cis-blaschke:
 assumes k ≠ 0 and M = k *sm (a, b, cnj b, cnj a) and
      a ≠ 0 and mat-det (a, b, cnj b, cnj a) ≠ 0
 shows ∃ k' φ a'. k' ≠ 0 ∧ a' * cnj a' ≠ 1 ∧
                          M = k' *sm (cis φ, 0, 0, 1) *mm (1, -a', -cnj a', 1)
proof-
 have a = cnj a * cis (2 * Arg a)
  using rcis-cmod-Arg[of a] rcis-cnj[of a]
  using cis-rcis-eq rcis-mult
  by simp
 thus ?thesis
  using assms
  by (rule-tac x=k*cnj a in exI, rule-tac x=2*Arg a in exI, rule-tac x=- b / a in exI) (auto simp add: field-simps)
qed

lemma unitary11-gen-cis-blaschke':
 assumes k ≠ 0 and M = k *sm (-1, 0, 0, 1) *mm (a, b, cnj b, cnj a) and
      a ≠ 0 and mat-det (a, b, cnj b, cnj a) ≠ 0
 shows ∃ k' φ a'. k' ≠ 0 ∧ a' * cnj a' ≠ 1 ∧
                          M = k' *sm (cis φ, 0, 0, 1) *mm (1, -a', -cnj a', 1)
proof-
 obtain k' φ a' where *: k' ≠ 0 k *sm (a, b, cnj b, cnj a) = k' *sm (cis φ, 0, 0, 1) *mm (1, -a', -cnj a', 1) a' *
cnj a' ≠ 1
```

```
    using unitary11-gen-cis-blaschke[OF ‹k ≠ 0› - ‹a ≠ 0›] ‹mat-det (a, b, cnj b, cnj a) ≠ 0›
    by blast
  have (cis φ, 0, 0, 1) *mm (−1, 0, 0, 1) = (cis (φ + pi), 0, 0, 1)
   by (simp add: cis-def complex.corec Complex-eq)
  thus ?thesis
    using * ‹M = k *sm (−1, 0, 0, 1) *mm (a, b, cnj b, cnj a)›
    by (rule-tac x=k' in exI, rule-tac x=φ + pi in exI, rule-tac x=a' in exI, simp)
qed

lemma unitary11-gen-cis-blaschke-rev:
  assumes k' ≠ 0 and M = k' *sm (cis φ, 0, 0, 1) *mm (1, −a', −cnj a', 1) and
        a' * cnj a' ≠ 1
  shows ∃ k a b. k ≠ 0 ∧ mat-det (a, b, cnj b, cnj a) ≠ 0 ∧
                       M = k *sm (a, b, cnj b, cnj a)
  using assms
  apply (rule-tac x=k'*cis(φ/2) in exI, rule-tac x=cis(φ/2) in exI, rule-tac x=−a'*cis(φ/2) in exI)
  apply (simp add: cis-mult mult.commute mult.left-commute)
  done

lemma unitary11-gen-cis-inversion:
  assumes k ≠ 0 and M = k *sm (0, b, cnj b, 0) and b ≠ 0
  shows ∃ k' φ. k' ≠ 0 ∧
                          M = k' *sm (cis φ, 0, 0, 1) *mm (0, 1, 1, 0)
using assms
using rcis-cmod-Arg[of b, symmetric] rcis-cnj[of b] cis-rcis-eq
by simp (rule-tac x=2*Arg b in exI, simp add: rcis-mult)

lemma unitary11-gen-cis-inversion':
  assumes k ≠ 0 and M = k *sm (−1, 0, 0, 1) *mm (0, b, cnj b, 0) and b ≠ 0
  shows ∃ k' φ. k' ≠ 0 ∧
                M = k' *sm (cis φ, 0, 0, 1) *mm (0, 1, 1, 0)
proof−
  obtain k' φ where *: k' ≠ 0 k *sm (0, b, cnj b, 0) = k' *sm (cis φ, 0, 0, 1) *mm (0, 1, 1, 0)
    using unitary11-gen-cis-inversion[OF ‹k ≠ 0› - ‹b ≠ 0›]
    by metis
  have (cis φ, 0, 0, 1) *mm (−1, 0, 0, 1) = (cis (φ + pi), 0, 0, 1)
    by (simp add: cis-def complex.corec Complex-eq)
  thus ?thesis
    using * ‹M = k *sm (−1, 0, 0, 1) *mm (0, b, cnj b, 0)›
    by (rule-tac x=k' in exI, rule-tac x=φ + pi in exI, simp)
qed

lemma unitary11-gen-cis-inversion-rev:
  assumes k' ≠ 0 and M = k' *sm (cis φ, 0, 0, 1) *mm (0, 1, 1, 0)
  shows ∃ k a b. k ≠ 0 ∧ mat-det (a, b, cnj b, cnj a) ≠ 0 ∧
                       M = k *sm (a, b, cnj b, cnj a)
  using assms
  by (rule-tac x=k'*cis(φ/2) in exI, rule-tac x=0 in exI, rule-tac x=cis(φ/2) in exI) (simp add: cis-mult)
```

Another characterization of generalized unitary11 matrices

```
lemma unitary11-gen-iff:
  shows unitary11-gen M ⟷
        (∃ k a b. k ≠ 0 ∧ mat-det (a, b, cnj b, cnj a) ≠ 0 ∧
                       M = k *sm (a, b, cnj b, cnj a)) (is ?lhs = ?rhs)
proof
  assume ?lhs
  then obtain a b k where *: k ≠ 0 mat-det (a, b, cnj b, cnj a) ≠ 0 M = k *sm (a, b, cnj b, cnj a) ∨ M = k *sm
(−1, 0, 0, 1) *mm (a, b, cnj b, cnj a)
    using unitary11-gen-iff'
    by auto
  show ?rhs
  proof (cases M = k *sm (a, b, cnj b, cnj a))
    case True
    thus ?thesis
      using *
      by auto
```

```
  next
    case False
    hence **: M = k *sm (-1, 0, 0, 1) *mm (a, b, cnj b, cnj a)
      using *
      by simp
    show ?thesis
    proof (cases a = 0)
      case True
      hence b ≠ 0
        using *
        by auto
      show ?thesis
        using unitary11-gen-cis-inversion-rev[of - M]
        using ** ‹a = 0›
        using unitary11-gen-cis-inversion'[OF ‹k ≠ 0› - ‹b ≠ 0›, of M]
        by auto
    next
      case False
      show ?thesis
        using unitary11-gen-cis-blaschke-rev[of - M]
        using **
        using unitary11-gen-cis-blaschke'[OF ‹k ≠ 0› - ‹a ≠ 0›, of M b] ‹mat-det (a, b, cnj b, cnj a) ≠ 0›
        by blast
    qed
  qed
next
  assume ?rhs
  thus ?lhs
    using unitary11-gen-iff'
    by auto
qed

lemma unitary11-iff:
  shows unitary11 M ⟷
        (∃ a b k. (cmod a)^2 > (cmod b)^2 ∧
                  (cmod k)^2 = 1 / ((cmod a)^2 - (cmod b)^2) ∧
                  M = k *sm (a, b, cnj b, cnj a)) (is ?lhs = ?rhs)
proof
  assume ?lhs
  obtain k a b where *:
    M = k *sm (a, b, cnj b, cnj a)mat-det (a, b, cnj b, cnj a) ≠ 0 k ≠ 0
    using unitary11-gen-iff unitary11-unitary11-gen[OF ‹unitary11 M›]
    by auto

  have md: mat-det (a, b, cnj b, cnj a) = cor ((cmod a)^2 - (cmod b)^2)
    by (auto simp add: complex-mult-cnj-cmod)
  hence **: (cmod a)^2 ≠ (cmod b)^2
    using ‹mat-det (a, b, cnj b, cnj a) ≠ 0›
    by auto

  have k * cnj k * mat-det (a, b, cnj b, cnj a) = 1
    using ‹M = k *sm (a, b, cnj b, cnj a)›
    using ‹unitary11 M›
    unfolding unitary11-def
    by (auto simp add: mat-adj-def mat-cnj-def) (simp add: field-simps)
  hence ***: (cmod k)^2 * ((cmod a)^2 - (cmod b)^2) = 1
    by (metis complex-mult-cnj-cmod md of-real-1 of-real-eq-iff of-real-mult)
  hence ((cmod a)^2 - (cmod b)^2) = 1 / (cmod k)^2
    by (cases k=0) (auto simp add: field-simps)
  hence cmod a ^ 2 = cmod b ^ 2 + 1 / cmod k ^ 2
    by simp
  thus ?rhs
    using ‹M = k *sm (a, b, cnj b, cnj a)› ** mat-eye-l
    by (rule-tac x=a in exI, rule-tac x=b in exI, rule-tac x=k in exI)
       (auto simp add: complex-mult-cnj-cmod intro!: )
next
```

assume *?rhs*
then obtain *a b k* **where** $(cmod\ b)^2 < (cmod\ a)^2 \wedge (cmod\ k)^2 = 1\ /\ ((cmod\ a)^2 - (cmod\ b)^2) \wedge M = k *_{sm} (a, b, cnj\ b, cnj\ a)$
by *auto*
moreover
have $cnj\ k * cnj\ a * (k * a) + - (cnj\ k * b * (k * cnj\ b)) = (cor\ ((cmod\ k)^2 * ((cmod\ a)^2 - (cmod\ b)^2)))$
proof−
have $cnj\ k * cnj\ a * (k * a) = cor\ ((cmod\ k)^2 * (cmod\ a)^2)$
using *complex-mult-cnj-cmod*[*of a*] *complex-mult-cnj-cmod*[*of k*]
by (*auto simp add*: *field-simps*)
moreover
have $cnj\ k * b * (k * cnj\ b) = cor\ ((cmod\ k)^2 * (cmod\ b)^2)$
using *complex-mult-cnj-cmod*[*of b*, *symmetric*] *complex-mult-cnj-cmod*[*of k*]
by (*auto simp add*: *field-simps*)
ultimately
show *?thesis*
by (*auto simp add*: *field-simps*)
qed
ultimately
show *?lhs*
unfolding *unitary11-def*
by (*auto simp add*: *mat-adj-def mat-cnj-def field-simps*)
qed

3.10.2 Group properties

Generalized unitary11 matrices form a group under multiplication (it is sometimes denoted by $GU_{1,1}(2,\mathbb{C})$). The group is also closed under non-zero complex scalar multiplication. Since these matrices are always regular, they form a subgroup of general linear group (usually denoted by $GL(2,\mathbb{C})$) of all regular matrices.

lemma *unitary11-gen-mult-sm*:
assumes $k \neq 0$ **and** *unitary11-gen M*
shows *unitary11-gen* $(k *_{sm} M)$
proof−
have $k * cnj\ k = cor\ (Re\ (k * cnj\ k))$
by (*subst complex-of-real-Re*) *auto*
thus *?thesis*
using *assms*
unfolding *unitary11-gen-real*
by *auto* (*rule-tac x=Re* (*k*∗*cnj k*) ∗ *ka* **in** *exI*, *auto*)
qed

lemma *unitary11-gen-div-sm*:
assumes $k \neq 0$ **and** *unitary11-gen* $(k *_{sm} M)$
shows *unitary11-gen M*
using *assms unitary11-gen-mult-sm*[*of* $1/k\ k *_{sm} M$]
by *simp*

lemma *unitary11-inv*:
assumes $k \neq 0$ **and** $M = k *_{sm} (a, b, cnj\ b, cnj\ a)$ **and** *mat-det* $(a, b, cnj\ b, cnj\ a) \neq 0$
shows $\exists\ k'\ a'\ b'.\ k' \neq 0 \wedge mat\text{-}inv\ M = k' *_{sm} (a', b', cnj\ b', cnj\ a') \wedge mat\text{-}det\ (a', b', cnj\ b', cnj\ a') \neq 0$
using *assms*
by (*subst assms*, *subst mat-inv-mult-sm*[*OF assms*(*1*)])
(*rule-tac x=1/*(*k* ∗ *mat-det* (*a*, *b*, *cnj b*, *cnj a*)) **in** *exI*, *rule-tac x=cnj a* **in** *exI*, *rule-tac x=*−*b* **in** *exI*, *simp add*: *field-simps*)

lemma *unitary11-comp*:
assumes $k1 \neq 0$ **and** $M1 = k1 *_{sm} (a1, b1, cnj\ b1, cnj\ a1)$ **and** *mat-det* $(a1, b1, cnj\ b1, cnj\ a1) \neq 0$
$k2 \neq 0\ M2 = k2 *_{sm} (a2, b2, cnj\ b2, cnj\ a2)$ *mat-det* $(a2, b2, cnj\ b2, cnj\ a2) \neq 0$
shows $\exists\ k\ a\ b.\ k \neq 0 \wedge M1 *_{mm} M2 = k *_{sm} (a, b, cnj\ b, cnj\ a) \wedge mat\text{-}det\ (a, b, cnj\ b, cnj\ a) \neq 0$
using *assms*
apply (*rule-tac x=k1*∗*k2* **in** *exI*)
apply (*rule-tac x=a1*∗*a2* + *b1*∗*cnj b2* **in** *exI*)
apply (*rule-tac x=a1*∗*b2* + *b1*∗*cnj a2* **in** *exI*)
proof (*auto simp add*: *algebra-simps*)
assume ∗: $a1 * (a2 * (cnj\ a1 * cnj\ a2)) + b1 * (b2 * (cnj\ b1 * cnj\ b2)) =$

```
        a1 * (b2 * (cnj a1 * cnj b2)) + a2 * (b1 * (cnj a2 * cnj b1)) and
      **: a1*cnj a1 ≠ b1 * cnj b1 a2*cnj a2 ≠ b2*cnj b2
  hence (a1*cnj a1)*(a2*cnj a2 − b2*cnj b2) = (b1*cnj b1)*(a2*cnj a2 − b2*cnj b2)
    by (simp add: field-simps)
  hence a1*cnj a1 = b1*cnj b1
    using **(2)
    by simp
  thus False
    using **(1)
    by simp
qed

lemma unitary11-gen-mat-inv:
  assumes unitary11-gen M and mat-det M ≠ 0
  shows unitary11-gen (mat-inv M)
proof−
  obtain k a b where k ≠ 0 ∧ mat-det (a, b, cnj b, cnj a) ≠ 0 ∧ M = k *sm (a, b, cnj b, cnj a)
    using assms unitary11-gen-iff[of M]
    by auto
  then obtain k' a' b' where k' ≠ 0 ∧ mat-inv M = k' *sm (a', b', cnj b', cnj a') ∧ mat-det (a', b', cnj b', cnj a') ≠ 0
    using unitary11-inv [of k M a b]
    by auto
  thus ?thesis
    using unitary11-gen-iff[of mat-inv M]
    by auto
qed

lemma unitary11-gen-comp:
  assumes unitary11-gen M1 and mat-det M1 ≠ 0 and unitary11-gen M2  and mat-det M2 ≠ 0
  shows unitary11-gen (M1 *mm M2)
proof−
  from assms obtain k1 k2 a1 a2 b1 b2 where
    k1 ≠ 0 ∧ mat-det (a1, b1, cnj b1, cnj a1) ≠ 0 ∧ M1 = k1 *sm (a1, b1, cnj b1, cnj a1)
    k2 ≠ 0 ∧ mat-det (a2, b2, cnj b2, cnj a2) ≠ 0 ∧ M2 = k2 *sm (a2, b2, cnj b2, cnj a2)
    using unitary11-gen-iff[of M1]  unitary11-gen-iff[of M2]
    by blast
  then obtain k a b where k ≠ 0 ∧ M1 *mm M2 = k *sm (a, b, cnj b, cnj a) ∧ mat-det (a, b, cnj b, cnj a) ≠ 0
    using unitary11-comp[of k1 M1 a1 b1 k2 M2 a2 b2]
    by blast
  thus ?thesis
    using unitary11-gen-iff[of M1 *mm M2]
    by blast
qed
```

Classification into orientation-preserving and orientation-reversing matrices

```
lemma unitary11-sgn-det-orientation:
  assumes k ≠ 0 and mat-det (a, b, cnj b, cnj a) ≠ 0 and M = k *sm (a, b, cnj b, cnj a)
  shows ∃ k'. sgn k' = sgn (Re (mat-det (a, b, cnj b, cnj a))) ∧ congruence M (1, 0, 0, −1) = cor k' *sm (1, 0, 0,
−1)
proof−
  let ?x = cnj k * cnj a * (k * a) − (cnj k * b * (k * cnj b))
  have *: ?x = k * cnj k * (a * cnj a − b * cnj b)
    by (auto simp add: field-simps)
  hence is-real ?x
    by auto
  hence cor (Re ?x) = ?x
    by (rule complex-of-real-Re)
  moreover
  have sgn (Re ?x) = sgn (Re (a * cnj a − b * cnj b))
  proof−
    have *: Re ?x = (cmod k)^2 * Re (a * cnj a − b * cnj b)
      by (subst *, subst complex-mult-cnj-cmod, subst Re-mult-real) (metis Im-complex-of-real, metis Re-complex-of-real)
    show ?thesis
      using ‹k ≠ 0›
      by (subst *) (simp add: sgn-mult)
  qed
```

ultimately
show *?thesis*
using *assms*(3)
by (*rule-tac x=Re ?x* **in** *exI*) (*auto simp add*: *mat-adj-def mat-cnj-def*)
qed

lemma *unitary11-sgn-det*:
assumes $k \neq 0$ **and** *mat-det* (a, b, *cnj b*, *cnj a*) $\neq 0$ **and** $M = k *_{sm} (a, b, cnj\ b, cnj\ a)$ **and** $M = (A, B, C, D)$
shows *sgn* (*Re* (*mat-det* (a, b, *cnj b*, *cnj a*))) = (*if* $b = 0$ *then* 1 *else sgn* (*Re* (($A*D$)/($B*C$)) − 1))
proof (*cases* $b = 0$)
case *True*
thus *?thesis*
using *assms*
by (*simp only*: *mat-det.simps*, *subst complex-mult-cnj-cmod*, *subst minus-complex.sel*, *subst Re-complex-of-real*, *simp*)
next
case *False*
from *assms* **have** ∗: $A = k * a\ B = k * b\ C = k * cnj\ b\ D = k * cnj\ a$
by *auto*
hence ∗: ($A*D$)/($B*C$) = ($a*cnj\ a$)/($b*cnj\ b$)
using ‹$k \neq 0$›
by *simp*
show *?thesis*
using ‹$b \neq 0$›
apply (*subst* ∗, *subst Re-divide-real*, *simp*, *simp*)
apply (*simp only*: *mat-det.simps*)
apply (*subst complex-mult-cnj-cmod*)+
apply ((*subst Re-complex-of-real*)+, *subst minus-complex.sel*, (*subst Re-complex-of-real*)+, *simp add*: *field-simps sgn-if*)
done
qed

lemma *unitary11-orientation*:
assumes *unitary11-gen M* **and** $M = (A, B, C, D)$
shows ∃ k'. *sgn* k' = *sgn* (*if* $B = 0$ *then* 1 *else sgn* (*Re* (($A*D$)/($B*C$)) − 1)) ∧ *congruence* M (1, 0, 0, −1) = *cor* $k' *_{sm}$ (1, 0, 0, −1)
proof−
from ‹*unitary11-gen M*›
obtain $k\ a\ b$ **where** ∗: $k \neq 0$ *mat-det* (a, b, *cnj b*, *cnj a*) $\neq 0$ $M = k *_{sm} (a, b, cnj\ b, cnj\ a)$
using *unitary11-gen-iff*[*of M*]
by *auto*
moreover
have $b = 0 \longleftrightarrow B = 0$
using ‹$M = (A, B, C, D)$› ∗
by *auto*
ultimately
show *?thesis*
using *unitary11-sgn-det-orientation*[*OF* ∗] *unitary11-sgn-det*[*OF* ∗ ‹$M = (A, B, C, D)$›]
by *auto*
qed

lemma *unitary11-sgn-det-orientation′*:
assumes *congruence* M (1, 0, 0, −1) = *cor* $k' *_{sm}$ (1, 0, 0, −1) **and** $k' \neq 0$
shows ∃ $a\ b\ k$. $k \neq 0 \wedge M = k *_{sm} (a, b, cnj\ b, cnj\ a) \wedge sgn\ k' = sgn\ (Re\ (mat\text{-}det\ (a, b, cnj\ b, cnj\ a)))$
proof−
obtain $a\ b\ k$ **where**
$k \neq 0$ *mat-det* (a, b, *cnj b*, *cnj a*) $\neq 0$ $M = k *_{sm} (a, b, cnj\ b, cnj\ a)$
using *assms*
using *unitary11-gen-iff*[*of M*]
unfolding *unitary11-gen-def*
by *auto*
moreover
have *sgn* k' = *sgn* (*Re* (*mat-det* (a, b, *cnj b*, *cnj a*)))
proof−
let *?x* = *cnj* k ∗ *cnj* a ∗ (k ∗ a) − (*cnj* k ∗ b ∗ (k ∗ *cnj b*))
have ∗: *?x* = k ∗ *cnj* k ∗ (a ∗ *cnj* a − b ∗ *cnj b*)
by (*auto simp add*: *field-simps*)

hence *is-real ?x*
by *auto*
hence *cor* (*Re ?x*) = *?x*
by (*rule complex-of-real-Re*)

have ∗∗: *sgn* (*Re ?x*) = *sgn* (*Re* (*a* ∗ *cnj a* − *b* ∗ *cnj b*))
proof−
have ∗: *Re ?x* = (*cmod k*)^2 ∗ *Re* (*a* ∗ *cnj a* − *b* ∗ *cnj b*)
by (*subst* ∗, *subst complex-mult-cnj-cmod*, *subst Re-mult-real*) (*metis Im-complex-of-real*, *metis Re-complex-of-real*)
show *?thesis*
using ‹*k* ≠ *0*›
by (*subst* ∗) (*simp add*: *sgn-mult*)
qed
moreover
have *?x* = *cor k′*
using ‹*M* = *k* ∗$_{sm}$ (*a*, *b*, *cnj b*, *cnj a*)› *assms*
by (*simp add*: *mat-adj-def mat-cnj-def*)
hence *sgn* (*Re ?x*) = *sgn k′*
using ‹*cor* (*Re ?x*) = *?x*›
unfolding *complex-of-real-def*
by *simp*
ultimately
show *?thesis*
by *simp*
qed
ultimately
show *?thesis*
by (*rule-tac x*=*a* **in** *exI*, *rule-tac x*=*b* **in** *exI*, *rule-tac x*=*k* **in** *exI*) *simp*
qed

end

3.11 Hermitean matrices

Hermitean matrices over $\mathbb{C}$ generalize symmetric matrices over $\mathbb{R}$. Quadratic forms with Hermitean matrices represent circles and lines in the extended complex plane (when applied to homogenous coordinates).

theory *Hermitean-Matrices*
imports *Unitary-Matrices*
begin

definition *hermitean* :: *complex-mat* ⇒ *bool* **where**
hermitean A ⟷ *mat-adj A* = *A*

lemma *hermitean-transpose*:
shows *hermitean A* ⟷ *mat-transpose A* = *mat-cnj A*
unfolding *hermitean-def*
by (*cases A*) (*auto simp add*: *mat-adj-def mat-cnj-def*)

Characterization of 2x2 Hermitean matrices elements. All 2x2 Hermitean matrices are of the form

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

for real A and D and complex B.

lemma *hermitean-mk-circline* [*simp*]:
shows *hermitean* (*cor A*, *B*, *cnj B*, *cor D*)
unfolding *hermitean-def mat-adj-def mat-cnj-def*
by *simp*

lemma *hermitean-mk-circline′* [*simp*]:
assumes *is-real A* **and** *is-real D*
shows *hermitean* (*A*, *B*, *cnj B*, *D*)
using *assms eq-cnj-iff-real*
unfolding *hermitean-def mat-adj-def mat-cnj-def*
by *force*

lemma *hermitean-elems*:
assumes *hermitean* (*A*, *B*, *C*, *D*)
shows *is-real A* **and** *is-real D* **and** *B* = *cnj C* **and** *cnj B* = *C*
using *assms eq-cnj-iff-real*[*of A*] *eq-cnj-iff-real*[*of D*]
by (*auto simp add*: *hermitean-def mat-adj-def mat-cnj-def*)

Operations that preserve the Hermitean property

lemma *hermitean-mat-cnj*:
shows *hermitean H* ⟷ *hermitean* (*mat-cnj H*)
by (*cases H*) (*auto simp add*: *hermitean-def mat-adj-def mat-cnj-def*)

lemma *hermitean-mult-real*:
assumes *hermitean H*
shows *hermitean* ((*cor k*) ∗$_{sm}$ *H*)
using *assms*
unfolding *hermitean-def*
by *simp*

lemma *hermitean-congruence*:
assumes *hermitean H*
shows *hermitean* (*congruence M H*)
using *assms*
unfolding *hermitean-def*
by (*auto simp add*: *mult-mm-assoc*)

Identity matrix is Hermitean

lemma *hermitean-eye* [*simp*]:
shows *hermitean eye*
by (*auto simp add*: *hermitean-def mat-adj-def mat-cnj-def*)

lemma *hermitean-eye′* [*simp*]:
shows *hermitean* (*1*, *0*, *0*, *1*)
by (*auto simp add*: *hermitean-def mat-adj-def mat-cnj-def*)

Unit circle matrix is Hermitean

lemma *hermitean-unit-circle* [*simp*]:
shows *hermitean* (*1*, *0*, *0*, −*1*)
by (*auto simp add*: *hermitean-def mat-adj-def mat-cnj-def*)

Hermitean matrices have real determinant

lemma *mat-det-hermitean-real*:
assumes *hermitean A*
shows *is-real* (*mat-det A*)
using *assms*
unfolding *hermitean-def*
by (*metis eq-cnj-iff-real mat-det-adj*)

Zero matrix is the only Hermitean matrix with both determinant and trace equal to zero

lemma *hermitean-det-zero-trace-zero*:
assumes *mat-det A* = *0* **and** *mat-trace A* = (*0*::*complex*) **and** *hermitean A*
shows *A* = *mat-zero*
using *assms*
proof−
{
fix *a d c*
assume *a* ∗ *d* = *cnj c* ∗ *c a* + *d* = *0 cnj a* = *a*
from ‹*a* + *d* = *0*› **have** *d* = −*a*
by (*metis add-eq-0-iff*)
hence − (*cor* (*Re a*))^2 = (*cor* (*cmod c*))^2
using ‹*cnj a* = *a*› *eq-cnj-iff-real*[*of a*]
using ‹*a*∗*d* = *cnj c* ∗ *c*›
using *complex-mult-cnj-cmod*[*of cnj c*]
by (*simp add*: *power2-eq-square*)
hence − (*Re a*)^2 ≥ *0*
using *zero-le-power2*[*of cmod c*]

by (*metis Re-complex-of-real of-real-minus of-real-power*)
hence $a = 0$
using *zero-le-power2*[*of Re a*]
using ‹*cnj a = a*› *eq-cnj-iff-real*[*of a*]
by (*simp add*: *complex-eq-if-Re-eq*)
} note $* = this$
obtain $a\ b\ c\ d$ **where** $A = (a, b, c, d)$
by (*cases A*) *auto*
thus *?thesis*
using $*$[*of a d c*] $*$[*of d a c*]
using *assms* ‹$A = (a, b, c, d)$›
by (*auto simp add*: *hermitean-def mat-adj-def mat-cnj-def*)
qed

3.11.1 Bilinear and quadratic forms with Hermitean matrices

A Hermitean matrix $(A, B, \overline{B}, D)$, for real A and D, gives rise to bilinear form $A \cdot \overline{v_{11}} \cdot v_{21} + \overline{B} \cdot \overline{v_{12}} \cdot v_{21} + B \cdot \overline{v_{11}} \cdot v_{22} + D \cdot \overline{v_{12}} \cdot v_{22}$ (acting on vectors (v_{11}, v_{12}) and (v_{21}, v_{22})) and to the quadratic form $A \cdot \overline{v_1} \cdot v_1 + \overline{B} \cdot \overline{v_2} \cdot v_1 + B \cdot \overline{v_1} \cdot v_2 + D \cdot \overline{v_2} \cdot v_2$ (acting on the vector (v_1, v_2)).

lemma *bilinear-form-hermitean-commute*:
assumes *hermitean H*
shows *bilinear-form v1 v2 H = cnj* (*bilinear-form v2 v1 H*)
proof−
have *v2* $*_{vm}$ *mat-cnj H* $*_{vv}$ *vec-cnj v1 = vec-cnj v1* $*_{vv}$ (*mat-adj H* $*_{mv}$ *v2*)
by (*subst mult-vv-commute*, *subst mult-mv-mult-vm*, *simp add*: *mat-adj-def mat-transpose-mat-cnj*)
also
have ... = *bilinear-form v1 v2 H*
using *assms*
by (*simp add*: *mult-vv-mv hermitean-def*)
finally
show *?thesis*
by (*simp add*: *cnj-mult-vv vec-cnj-mult-vm*)
qed

lemma *quad-form-hermitean-real*:
assumes *hermitean H*
shows *is-real* (*quad-form z H*)
using *assms*
by (*subst eq-cnj-iff-real*[*symmetric*]) (*simp del*: *quad-form-def add*: *hermitean-def*)

lemma *quad-form-vec-cnj-mat-cnj*:
assumes *hermitean H*
shows *quad-form* (*vec-cnj z*) (*mat-cnj H*) = *quad-form z H*
using *assms*
using *cnj-mult-vv cnj-quad-form hermitean-def vec-cnj-mult-vm* **by** *auto*

3.11.2 Eigenvalues, eigenvectors and diagonalization of Hermitean matrices

Hermitean matrices have real eigenvalues

lemma *hermitean-eigenval-real*:
assumes *hermitean H* **and** *eigenval k H*
shows *is-real k*
proof−
from *assms* **obtain** *v* **where** $v \neq$ *vec-zero H* $*_{mv}$ *v = k* $*_{sv}$ *v*
unfolding *eigenval-def*
by *blast*
have $k * (v *_{vv}$ *vec-cnj v*) = ($k *_{sv}$ *v*) $*_{vv}$ (*vec-cnj v*)
by (*simp add*: *mult-vv-scale-sv1*)
also have ... = (*H* $*_{mv}$ *v*) $*_{vv}$ (*vec-cnj v*)
using ‹*H* $*_{mv}$ *v = k* $*_{sv}$ *v*›
by *simp*
also have ... = *v* $*_{vv}$ (*mat-transpose H* $*_{mv}$ (*vec-cnj v*))
by (*simp add*: *mult-mv-vv*)
also have ... = *v* $*_{vv}$ (*vec-cnj* (*mat-cnj* (*mat-transpose H*) $*_{mv}$ *v*))
by (*simp add*: *vec-cnj-mult-mv*)

also have ... = $v *_{vv} (vec\text{-}cnj\ (H *_{mv} v))$
using ‹*hermitean H*›
by (*simp add*: *hermitean-def mat-adj-def*)
also have ... = $v *_{vv} (vec\text{-}cnj\ (k *_{sv} v))$
using ‹$H *_{mv} v = k *_{sv} v$›
by *simp*
finally have $k * (v *_{vv} vec\text{-}cnj\ v) = cnj\ k * (v *_{vv} vec\text{-}cnj\ v)$
by (*simp add*: *mult-vv-scale-sv2*)
hence $k = cnj\ k$
using ‹$v \neq vec\text{-}zero$›
using *scalsquare-vv-zero*[*of v*]
by (*simp add*: *mult-vv-commute*)
thus *?thesis*
by (*metis eq-cnj-iff-real*)
qed

Non-diagonal Hermitean matrices have distinct eigenvalues

lemma *hermitean-distinct-eigenvals*:
assumes *hermitean H*
shows $(\exists\ k_1\ k_2.\ k_1 \neq k_2 \wedge eigenval\ k_1\ H \wedge eigenval\ k_2\ H) \vee mat\text{-}diagonal\ H$
proof−
obtain *A B C D* **where** *HH*: $H = (A,\ B,\ C,\ D)$
by (*cases H*) *auto*
show *?thesis*
proof (*cases B = 0*)
case *True*
thus *?thesis*
using ‹*hermitean H*› *hermitean-elems*[*of A B C D*] *HH*
by *auto*
next
case *False*
have $(mat\text{-}trace\ H)^2 \neq 4 * mat\text{-}det\ H$
proof (*rule ccontr*)
have *C = cnj B is-real A is-real D*
using *hermitean-elems HH* ‹*hermitean H*›
by *auto*
assume ¬ *?thesis*
hence $(A + D)^2 = 4*(A*D - B*C)$
using *HH*
by *auto*
hence $(A - D)^2 = -4*B*cnj\ B$
using ‹$C = cnj\ B$›
by (*auto simp add*: *power2-eq-square field-simps*)
hence $(A - D)^2\ /\ cor\ ((cmod\ B)^2) = -4$
using ‹$B \neq 0$› *complex-mult-cnj-cmod*[*of B*]
by (*auto simp add*: *field-simps*)
hence $(Re\ A - Re\ D)^2\ /\ (cmod\ B)^2 = -4$
using ‹*is-real A*› ‹*is-real D*› ‹$B \neq 0$›
using *Re-divide-real*[*of cor* $((cmod\ B)^2)$ $(A - D)^2$]
by (*auto simp add*: *power2-eq-square*)
thus *False*
by (*metis abs-neg-numeral abs-power2 neg-numeral-neq-numeral power-divide*)
qed
show *?thesis*
apply (*rule disjI1*)
apply (*subst eigen-equation*)+
using *complex-quadratic-equation-monic-distinct-roots*[*of* $-mat\text{-}trace\ H\ mat\text{-}det\ H$] ‹$(mat\text{-}trace\ H)^2 \neq 4 * mat\text{-}det\ H$›
by *auto*
qed
qed

Eigenvectors corresponding to different eigenvalues of Hermitean matrices are orthogonal

lemma *hermitean-ortho-eigenvecs*:
assumes *hermitean H*
assumes *eigenpair k1 v1 H* **and** *eigenpair k2 v2 H* **and** *k1* ≠ *k2*

shows *vec-cnj v2* $*_{vv}$ *v1 = 0* **and** *vec-cnj v1* $*_{vv}$ *v2 = 0*
proof−
from *assms*
have *v1* ≠ *vec-zero H* $*_{mv}$ *v1 = k1* $*_{sv}$ *v1*
v2 ≠ *vec-zero H* $*_{mv}$ *v2 = k2* $*_{sv}$ *v2*
unfolding *eigenpair-def*
by *auto*
have *real-k*: *is-real k1 is-real k2*
using *assms*
using *hermitean-eigenval-real*[*of H k1*]
using *hermitean-eigenval-real*[*of H k2*]
unfolding *eigenpair-def eigenval-def*
by *blast*+

have *vec-cnj* (*H* $*_{mv}$ *v2*) = *vec-cnj* (*k2* $*_{sv}$ *v2*)
using ‹*H* $*_{mv}$ *v2 = k2* $*_{sv}$ *v2*›
by *auto*
hence *vec-cnj v2* $*_{vm}$ *H = k2* $*_{sv}$ *vec-cnj v2*
using ‹*hermitean H*› *real-k eq-cnj-iff-real*[*of k1*] *eq-cnj-iff-real*[*of k2*]
unfolding *hermitean-def*
by (*cases H*, *cases v2*) (*auto simp add*: *mat-adj-def mat-cnj-def vec-cnj-def*)
have *k2* ∗ (*vec-cnj v2* $*_{vv}$ *v1*) = *k1* ∗ (*vec-cnj v2* $*_{vv}$ *v1*)
using ‹*H* $*_{mv}$ *v1 = k1* $*_{sv}$ *v1*›
using ‹*vec-cnj v2* $*_{vm}$ *H = k2* $*_{sv}$ *vec-cnj v2*›
by (*cases v1*, *cases v2*, *cases H*)
(*metis mult-vv-mv mult-vv-scale-sv1 mult-vv-scale-sv2*)
thus *vec-cnj v2* $*_{vv}$ *v1 = 0*
using ‹*k1* ≠ *k2*›
by *simp*
hence *cnj* (*vec-cnj v2* $*_{vv}$ *v1*) = *0*
by *simp*
thus *vec-cnj v1* $*_{vv}$ *v2 = 0*
by (*simp add*: *cnj-mult-vv mult-vv-commute*)
qed

Hermitean matrices are diagonizable by unitary matrices. Diagonal entries are real and the sign of the determinant is preserved.

lemma *hermitean-diagonizable*:
assumes *hermitean H*
shows ∃ *k1 k2 M*. *mat-det M* ≠ *0* ∧ *unitary M* ∧ *congruence M H* = (*k1*, *0*, *0*, *k2*) ∧
is-real k1 ∧ *is-real k2* ∧ *sgn* (*Re k1* ∗ *Re k2*) = *sgn* (*Re* (*mat-det H*))
proof−
from *assms*
have (∃ k_1 k_2. k_1 ≠ k_2 ∧ *eigenval* k_1 *H* ∧ *eigenval* k_2 *H*) ∨ *mat-diagonal H*
using *hermitean-distinct-eigenvals*[*of H*]
by *simp*
thus *?thesis*
proof
assume ∃ k_1 k_2. k_1 ≠ k_2 ∧ *eigenval* k_1 *H* ∧ *eigenval* k_2 *H*
then obtain *k1 k2* **where** *k1* ≠ *k2 eigenval k1 H eigenval k2 H*
using *hermitean-distinct-eigenvals*
by *blast*
then obtain *v1 v2* **where** *eigenpair k1 v1 H eigenpair k2 v2 H*
v1 ≠ *vec-zero v2* ≠ *vec-zero*
unfolding *eigenval-def eigenpair-def*
by *blast*
hence ∗: *vec-cnj v2* $*_{vv}$ *v1 = 0 vec-cnj v1* $*_{vv}$ *v2 = 0*
using ‹*k1* ≠ *k2*› *hermitean-ortho-eigenvecs* ‹*hermitean H*›
by *auto*
obtain *v11 v12 v21 v22* **where** *vv*: *v1* = (*v11*, *v12*) *v2* = (*v21*, *v22*)
by (*cases v1*, *cases v2*) *auto*
let *?nv1′* = *vec-cnj v1* $*_{vv}$ *v1* **and** *?nv2′* = *vec-cnj v2* $*_{vv}$ *v2*
let *?nv1* = *cor* (*sqrt* (*Re ?nv1′*))
let *?nv2* = *cor* (*sqrt* (*Re ?nv2′*))
have *?nv1′* ≠ *0* *?nv2′* ≠ *0*
using ‹*v1* ≠ *vec-zero*› ‹*v2* ≠ *vec-zero*› *vv*

```
  by (simp add: scalsquare-vv-zero)+
moreover
have is-real ?nv1' is-real ?nv2'
  using vv
  by (auto simp add: vec-cnj-def)
ultimately
have ?nv1 ≠ 0  ?nv2 ≠ 0
  using complex-eq-if-Re-eq
  by auto
have Re (?nv1') ≥ 0  Re (?nv2') ≥ 0
  using vv
  by (auto simp add: vec-cnj-def)
obtain nv1 nv2 where nv1 = ?nv1 nv1 ≠ 0  nv2 = ?nv2 nv2 ≠ 0
  using ‹?nv1 ≠ 0› ‹?nv2 ≠ 0›
  by auto
let ?M = (1/nv1 * v11, 1/nv2 * v21, 1/nv1 * v12, 1/nv2 * v22)

have is-real k1 is-real k2
  using ‹eigenval k1 H› ‹eigenval k2 H› ‹hermitean H›
  by (auto simp add: hermitean-eigenval-real)
moreover
have mat-det ?M ≠ 0
proof (rule ccontr)
  assume ¬ ?thesis
  hence v11 * v22 = v12 * v21
    using ‹nv1 ≠ 0› ‹nv2 ≠ 0›
    by (auto simp add: field-simps)
  hence ∃ k. k ≠ 0 ∧ v2 = k *_sv v1
    using vv ‹v1 ≠ vec-zero› ‹v2 ≠ vec-zero›
    apply auto
    apply (rule-tac x=v21/v11 in exI, force simp add: field-simps)
    apply (rule-tac x=v21/v11 in exI, force simp add: field-simps)
    apply (rule-tac x=v22/v12 in exI, force simp add: field-simps)
    apply (rule-tac x=v22/v12 in exI, force simp add: field-simps)
    done
  thus False
    using * ‹vec-cnj v1 *_vv v2 = 0› ‹vec-cnj v2 *_vv v2 ≠ 0› vv ‹?nv1' ≠ 0›
    by (metis mult-vv-scale-sv2 mult-zero-right)
qed
moreover
have unitary ?M
proof−
  have **: cnj nv1 * nv1 = ?nv1'  cnj nv2 * nv2 = ?nv2'
    using ‹nv1 = ?nv1› ‹nv1 ≠ 0›  ‹nv2 = ?nv2› ‹nv2 ≠ 0› ‹is-real ?nv1'› ‹is-real ?nv2'›
    using ‹Re (?nv1') ≥ 0›  ‹Re (?nv2') ≥ 0›
    by auto
  have ***: cnj nv1 * nv2 ≠ 0  cnj nv2 * nv1 ≠ 0
    using vv ‹nv1 = ?nv1› ‹nv1 ≠ 0›  ‹nv2 = ?nv2› ‹nv2 ≠ 0› ‹is-real ?nv1'› ‹is-real ?nv2'›
    by auto

  show ?thesis
    unfolding unitary-def
    using vv ** ‹?nv1' ≠ 0› ‹?nv2' ≠ 0› * ***
    unfolding mat-adj-def mat-cnj-def vec-cnj-def
    by simp (metis (no-types, lifting) add-divide-distrib divide-eq-0-iff divide-eq-1-iff)
qed
moreover
have congruence ?M H = (k1, 0, 0, k2)
proof−
  have mat-inv ?M *_mm H *_mm ?M = (k1, 0, 0, k2)
  proof−
    have *: H *_mm ?M = ?M *_mm (k1, 0, 0, k2)
      using ‹eigenpair k1 v1 H› ‹eigenpair k2 v2 H› vv ‹?nv1 ≠ 0› ‹?nv2 ≠ 0›
      unfolding eigenpair-def vec-cnj-def
      apply (cases H)
      apply simp
```

apply (*metis add-divide-distrib mult.commute*)
done
show *?thesis*
using *mult-mm-inv-l*[*of ?M* (*k1*, *0*, *0*, *k2*) *H* $*_{mm}$ *?M*, *OF* ‹*mat-det ?M* ≠ *0*› *∗*[*symmetric*], *symmetric*]
by (*simp add*: *mult-mm-assoc*)
qed
moreover
have *mat-inv ?M* = *mat-adj ?M*
using ‹*mat-det ?M* ≠ *0*› ‹*unitary ?M*› *mult-mm-inv-r*[*of ?M mat-adj ?M eye*]
by (*simp add*: *unitary-def*)
ultimately
show *?thesis*
by *simp*
qed
moreover
have *sgn* (*Re k1* ∗ *Re k2*) = *sgn* (*Re* (*mat-det H*))
using ‹*congruence ?M H* = (*k1*, *0*, *0*, *k2*)› ‹*is-real k1*› ‹*is-real k2*›
using *Re-det-sgn-congruence*[*of ?M H*] ‹*mat-det ?M* ≠ *0*› ‹*hermitean H*›
by *simp*
ultimately
show *?thesis*
by (*rule-tac x*=*k1* **in** *exI*, *rule-tac x*=*k2* **in** *exI*, *rule-tac x*=*?M* **in** *exI*) *simp*
next
assume *mat-diagonal H*
then obtain *A D* **where** *H* = (*A*, *0*, *0*, *D*)
by (*cases H*) *auto*
moreover
hence *is-real A is-real D*
using ‹*hermitean H*› *hermitean-elems*[*of A 0 0 D*]
by *auto*
ultimately
show *?thesis*
by (*rule-tac x*=*A* **in** *exI*, *rule-tac x*=*D* **in** *exI*, *rule-tac x*=*eye* **in** *exI*) (*simp add*: *unitary-def mat-adj-def mat-cnj-def*)
qed
qed

end

4 Elementary complex geometry

In this section equations and basic properties of the most fundamental objects and relations in geometry – collinearity, lines, circles and circlines. These are defined by equations in ℂ (not extended by an infinite point). Later these equations will be generalized to equations in the extended complex plane, over homogenous coordinates.

theory *Elementary-Complex-Geometry*
imports *More-Complex Linear-Systems Angles*
begin

4.1 Collinear points

definition *collinear* :: *complex* ⇒ *complex* ⇒ *complex* ⇒ *bool* **where**
collinear z1 z2 z3 ⟷ *z1* = *z2* ∨ *Im* ((*z3* − *z1*) / (*z2* − *z1*)) = *0*

lemma *collinear-ex-real*:
shows *collinear z1 z2 z3* ⟷
(∃ *k*::*real*. *z1* = *z2* ∨ *z3* − *z1* = *complex-of-real k* ∗ (*z2* − *z1*))
unfolding *collinear-def*
by (*metis Im-complex-of-real add-diff-cancel-right' complex-eq diff-zero legacy-Complex-simps*(*15*) *nonzero-mult-div-cancel-right right-minus-eq times-divide-eq-left zero-complex.code*)

Collinearity characterization using determinants

lemma *collinear-det*:
assumes ¬ *collinear z1 z2 z3*
shows *det2* (*z3* − *z1*) (*cnj* (*z3* − *z1*)) (*z1* − *z2*) (*cnj* (*z1* − *z2*)) ≠ *0*

proof −
from *assms* **have** $((z3 - z1) / (z2 - z1)) - cnj ((z3 - z1) / (z2 - z1)) \neq 0$ $z2 \neq z1$
unfolding *collinear-def*
using *Complex-Im-express-cnj*[*of* $(z3 - z1) / (z2 - z1)$]
by (*auto simp add*: *Complex-eq*)
thus *?thesis*
by (*auto simp add*: *field-simps*)
qed

Properties of three collinear points

lemma *collinear-sym1*:
shows *collinear z1 z2 z3* $\longleftrightarrow$ *collinear z1 z3 z2*
unfolding *collinear-def*
using *div-reals*[*of 1* $(z3 - z1)/(z2 - z1)$] *div-reals*[*of 1* $(z2 - z1)/(z3 - z1)$]
by *auto*

lemma *collinear-sym2'*:
assumes *collinear z1 z2 z3*
shows *collinear z2 z1 z3*
proof −
obtain *k* **where** $z1 = z2 \vee z3 - z1 = complex\text{-}of\text{-}real\ k * (z2 - z1)$
using *assms*
unfolding *collinear-ex-real*
by *auto*
thus *?thesis*
proof
assume $z3 - z1 = complex\text{-}of\text{-}real\ k * (z2 - z1)$
thus *?thesis*
unfolding *collinear-ex-real*
by (*rule-tac* $x=1-k$ **in** *exI*) (*auto simp add*: *field-simps*)
qed (*simp add*: *collinear-def*)
qed

lemma *collinear-sym2*:
shows *collinear z1 z2 z3* $\longleftrightarrow$ *collinear z2 z1 z3*
using *collinear-sym2'*[*of z1 z2 z3*] *collinear-sym2'*[*of z2 z1 z3*]
by *auto*

Properties of four collinear points

lemma *collinear-trans1*:
assumes *collinear z0 z2 z1* **and** *collinear z0 z3 z1* **and** $z0 \neq z1$
shows *collinear z0 z2 z3*
using *assms*
unfolding *collinear-ex-real*
by (*cases* $z0 = z2$, *auto*) (*rule-tac* $x=k/ka$ **in** *exI*, *case-tac* $ka = 0$, *auto simp add*: *field-simps*)

4.2 Euclidean line

Line is defined by using collinearity

definition *line* :: *complex* ⇒ *complex* ⇒ *complex set* **where**
line z1 z2 = {*z*. *collinear z1 z2 z*}

lemma *line-points-collinear*:
assumes $z1 \in line\ z\ z'$ **and** $z2 \in line\ z\ z'$ **and** $z3 \in line\ z\ z'$ **and** $z \neq z'$
shows *collinear z1 z2 z3*
using *assms*
unfolding *line-def*
by (*smt* (*verit*) *collinear-sym1 collinear-sym2' collinear-trans1 mem-Collect-eq*)

Parametric equation of a line

lemma *line-param*:
shows $z1 + cor\ k * (z2 - z1) \in line\ z1\ z2$
unfolding *line-def*
by (*auto simp add*: *collinear-def*)

Equation of the line containing two different given points

lemma *line-equation*:
  assumes $z1 \neq z2$ **and** $\mu = rot90\ (z2 - z1)$
  shows $line\ z1\ z2 = \{z.\ cnj\ \mu*z + \mu*cnj\ z - (cnj\ \mu * z1 + \mu * cnj\ z1)\ = 0\}$
proof−
  {
    fix z
    have $z \in line\ z1\ z2 \longleftrightarrow Im\ ((z - z1)/(z2 - z1)) = 0$
      using *assms*
      by (*simp add*: *line-def collinear-def*)
    also have $... \longleftrightarrow (z - z1)/(z2 - z1) = cnj\ ((z - z1)/(z2 - z1))$
      using *complex-diff-cnj*[*of* $(z - z1)/(z2 - z1)$]
      by *auto*
    also have $... \longleftrightarrow (z - z1)*(cnj\ z2 - cnj\ z1) = (cnj\ z - cnj\ z1)*(z2 - z1)$
      using *assms*(*1*)
      using ‹$(z \in line\ z1\ z2) = is\text{-}real\ ((z - z1)\ /\ (z2 - z1))$› *calculation is-real-div*
      by *auto*
    also have $... \longleftrightarrow cnj(z2 - z1)*z - (z2 - z1)*cnj\ z - (cnj(z2 - z1)*z1 - (z2 - z1)*cnj\ z1) = 0$
      by (*simp add*: *field-simps*)
    also have $... \longleftrightarrow cnj\ \mu * z + \mu * cnj\ z\ - (cnj\ \mu * z1 + \mu * cnj\ z1) = 0$
      apply (*subst assms*)+
      apply (*subst cnj-mix-minus*)+
      by *simp*
    finally have $z \in line\ z1\ z2 \longleftrightarrow cnj\ \mu * z + \mu * cnj\ z\ - (cnj\ \mu * z1 + \mu * cnj\ z1) = 0$
      .
  }
  thus *?thesis*
    by *auto*
qed

4.3 Euclidean circle

Definition of the circle with given center and radius. It consists of all points on the distance r from the center μ.

definition *circle* :: *complex* ⇒ *real* ⇒ *complex set* **where**
  $circle\ \mu\ r = \{z.\ cmod\ (z - \mu) = r\}$

Equation of the circle centered at μ with the radius r.

lemma *circle-equation*:
  assumes $r \geq 0$
  shows $circle\ \mu\ r = \{z.\ z*cnj\ z - z*cnj\ \mu - cnj\ z*\mu + \mu*cnj\ \mu - cor\ (r*r) = 0\}$
proof (*safe*)
  fix z
  assume $z \in circle\ \mu\ r$
  hence $(z - \mu)*cnj\ (z - \mu) = complex\text{-}of\text{-}real\ (r*r)$
    unfolding *circle-def*
    using *complex-mult-cnj-cmod*[*of* $z - \mu$]
    by (*auto simp add*: *power2-eq-square*)
  thus $z * cnj\ z - z * cnj\ \mu - cnj\ z * \mu + \mu * cnj\ \mu - cor\ (r * r) = 0$
    by (*auto simp add*: *field-simps*)
next
  fix z
  assume $z * cnj\ z - z * cnj\ \mu - cnj\ z * \mu + \mu * cnj\ \mu - cor\ (r * r) = 0$
  hence $(z - \mu)*cnj\ (z - \mu) = cor\ (r*r)$
    by (*auto simp add*: *field-simps*)
  thus $z \in circle\ \mu\ r$
    using *assms*
    using *complex-mult-cnj-cmod*[*of* $z - \mu$]
    using *power2-eq-imp-eq*[*of cmod* $(z - \mu)\ r$]
    unfolding *circle-def power2-eq-square*[*symmetric*] *complex-of-real-def*
    by *auto*
qed

4.4 Circline

A very important property of the extended complex plane is that it is possible to treat circles and lines in a uniform way. The basic object is *generalized circle*, or *circline* for short. We introduce circline equation given in $\mathbb{C}$, and it will later be generalized to an equation in the extended complex plane $\overline{\mathbb{C}}$ given in matrix form using a Hermitean matrix and a quadratic form over homogenous coordinates.

definition *circline* **where**
circline A BC D = {*z*. *cor A*∗*z*∗*cnj z* + *cnj BC*∗*z* + *BC*∗*cnj z* + *cor D* = *0*}

Connection between circline and Euclidean circle

Every circline with positive determinant and $A \neq 0$ represents an Euclidean circle

lemma *circline-circle*:
assumes $A \neq 0$ **and** $A * D \leq (cmod\ BC)^2$
$cl = circline\ A\ BC\ D$ **and**
$\mu = -BC/cor\ A$ **and**
$r2 = ((cmod\ BC)^2 - A*D)\ /\ A^2$ **and** $r = sqrt\ r2$
shows $cl = circle\ \mu\ r$
proof−
have ∗: $cl = \{z.\ z * cnj\ z + cnj\ (BC\ /\ cor\ A) * z + (BC\ /\ cor\ A) * cnj\ z + cor\ (D\ /\ A) = 0\}$
using ‹$cl = circline\ A\ BC\ D$› ‹$A \neq 0$›
by (*auto simp add*: *circline-def field-simps*)

have $r2 \geq 0$
proof−
have $(cmod\ BC)^2 - A * D \geq 0$
using ‹$A * D \leq (cmod\ BC)^2$›
by *auto*
thus *?thesis*
using ‹$A \neq 0$› ‹$r2 = ((cmod\ BC)^2 - A*D)\ /\ A^2$›
by (*metis zero-le-divide-iff zero-le-power2*)
qed
hence ∗∗: $r * r = r2\ r \geq 0$
using ‹$r = sqrt\ r2$›
by (*auto simp add*: *real-sqrt-mult*[*symmetric*])

have ∗∗∗: $-\mu * - cnj\ \mu - cor\ r2 = cor\ (D\ /\ A)$
using ‹$\mu = -\ BC\ /\ complex\text{-}of\text{-}real\ A$› ‹$r2 = ((cmod\ BC)^2 - A * D)\ /\ A^2$›
by (*auto simp add*: *power2-eq-square complex-mult-cnj-cmod field-simps*)
(*simp add*: *add-divide-eq-iff assms*(*1*))
thus *?thesis*
using ‹$r2 = ((cmod\ BC)^2 - A*D)\ /\ A^2$› ‹$\mu = -\ BC\ /\ cor\ A$›
by (*subst* ∗, *subst circle-equation*[*of r* μ, *OF* ‹$r \geq 0$›], *subst* ∗∗) (*auto simp add*: *field-simps power2-eq-square*)
qed

lemma *circline-ex-circle*:
assumes $A \neq 0$ **and** $A * D \leq (cmod\ BC)^2$ **and** $cl = circline\ A\ BC\ D$
shows ∃ $\mu\ r.\ cl = circle\ \mu\ r$
using *circline-circle*[*OF assms*]
by *auto*

Every Euclidean circle can be represented by a circline

lemma *circle-circline*:
assumes $cl = circle\ \mu\ r$ **and** $r \geq 0$
shows $cl = circline\ 1\ (-\mu)\ ((cmod\ \mu)^2 - r^2)$
proof−
have $complex\text{-}of\text{-}real\ ((cmod\ \mu)^2 - r^2) = \mu * cnj\ \mu - complex\text{-}of\text{-}real\ (r^2)$
by (*auto simp add*: *complex-mult-cnj-cmod*)
thus $cl = circline\ 1\ (-\mu)\ ((cmod\ \mu)^2 - r^2)$
using *assms*
using *circle-equation*[*of r* μ]
unfolding *circline-def power2-eq-square*
by (*simp add*: *field-simps*)
qed

lemma *circle-ex-circline*:
assumes $cl = circle\ \mu\ r$ **and** $r \geq 0$
shows $\exists\ A\ BC\ D.\ A \neq 0 \wedge A*D \leq (cmod\ BC)^2 \wedge cl = circline\ A\ BC\ D$
using *circle-circline*[*OF assms*]
using ‹$r \geq 0$›
by (*rule-tac x=1* **in** *exI*, *rule-tac x=*$-\mu$ **in** *exI*, *rule-tac x=Re* $(\mu * cnj\ \mu) - (r * r)$ **in** *exI*) (*simp add*: *complex-mult-cnj-cmod power2-eq-square*)

Connection between circline and Euclidean line

Every circline with a positive determinant and $A = 0$ represents an Euclidean line

lemma *circline-line*:
assumes
$A = 0$ **and** $BC \neq 0$ **and**
$cl = circline\ A\ BC\ D$ **and**
$z1 = -\ cor\ D * BC\ /\ (2 * BC * cnj\ BC)$ **and**
$z2 = z1 + \mathrm{i} * sgn\ (if\ Arg\ BC > 0\ then\ -BC\ else\ BC)$
shows
$cl = line\ z1\ z2$
proof−
have $cl = \{z.\ cnj\ BC*z + BC*cnj\ z + complex\text{-}of\text{-}real\ D = 0\}$
using *assms*
by (*simp add*: *circline-def*)
have $\{z.\ cnj\ BC*z + BC*cnj\ z + complex\text{-}of\text{-}real\ D = 0\} =$
$\{z.\ cnj\ BC*z + BC*cnj\ z - (cnj\ BC*z1 + BC*cnj\ z1) = 0\}$
using ‹$BC \neq 0$› *assms*
by *simp*
moreover
have $z1 \neq z2$
using ‹$BC \neq 0$› *assms*
by (*auto simp add*: *sgn-eq*)
moreover
have $\exists\ k.\ k \neq 0 \wedge BC = cor\ k*rot90\ (z2 - z1)$
proof (*cases Arg BC* > 0)
case *True*
thus *?thesis*
using *assms*
by (*rule-tac x=*$(cmod\ BC)$ **in** *exI*, *auto simp add*: *Complex-scale4*)
next
case *False*
thus *?thesis*
using *assms*
by (*rule-tac x=*$-(cmod\ BC)$ **in** *exI*, *simp*)
(*smt* (*verit*) *Complex.Re-sgn Im-sgn cis-Arg complex-minus complex-surj mult-minus-right rcis-cmod-Arg rcis-def*)
qed
then obtain k **where** $cor\ k \neq 0\ BC = cor\ k*rot90\ (z2 - z1)$
by *auto*
moreover
have $*$: $\bigwedge z.\ cnj\text{-}mix\ (BC\ /\ cor\ k)\ z - cnj\text{-}mix\ (BC\ /\ cor\ k)\ z1 = (1/cor\ k) * (cnj\text{-}mix\ BC\ z - cnj\text{-}mix\ BC\ z1)$
using ‹$cor\ k \neq 0$›
by (*simp add*: *field-simps*)
hence $\{z.\ cnj\text{-}mix\ BC\ z - cnj\text{-}mix\ BC\ z1 = 0\} = \{z.\ cnj\text{-}mix\ (BC\ /\ cor\ k)\ z - cnj\text{-}mix\ (BC\ /\ cor\ k)\ z1 = 0\}$
using ‹$cor\ k \neq 0$›
by *auto*
ultimately
have $cl = line\ z1\ z2$
using *line-equation*[*of z1 z2 BC*/*cor k*] ‹$cl = \{z.\ cnj\ BC*z + BC*cnj\ z + complex\text{-}of\text{-}real\ D = 0\}$›
by *auto*
thus *?thesis*
using ‹$z1 \neq z2$›
by *blast*
qed

lemma *circline-ex-line*:
assumes $A = 0$ **and** $BC \neq 0$ **and** $cl = circline\ A\ BC\ D$
shows $\exists\ z1\ z2.\ z1 \neq z2 \wedge cl = line\ z1\ z2$

```
proof−
  let ?z1 = − cor D ∗ BC / (2 ∗ BC ∗ cnj BC)
  let ?z2 = ?z1 + i ∗ sgn (if 0 < Arg BC then − BC else BC)
  have ?z1 ≠ ?z2
    using ‹BC ≠ 0›
    by (simp add: sgn-eq)
  thus ?thesis
    using circline-line[OF assms, of ?z1 ?z2] ‹BC ≠ 0›
    by (rule-tac x=?z1 in exI, rule-tac x=?z2 in exI, simp)
qed
```

Every Euclidean line can be represented by a circline

```
lemma line-ex-circline:
  assumes cl = line z1 z2 and z1 ≠ z2
  shows ∃ BC D. BC ≠ 0 ∧ cl = circline 0 BC D
proof−
  let ?BC = rot90 (z2 − z1)
  let ?D = Re (− 2 ∗ scalprod z1 ?BC)
  show ?thesis
  proof (rule-tac x=?BC in exI, rule-tac x=?D in exI, rule conjI)
    show ?BC ≠ 0
      using ‹z1 ≠ z2› rot90-ii[of z2 − z1]
      by auto
  next
    have ∗: complex-of-real (Re (− 2 ∗ scalprod z1 (rot90 (z2 − z1)))) = − (cnj-mix z1 (rot90 (z2 − z1)))
      using rot90-ii[of z2 − z1]
      by (cases z1, cases z2, simp add: Complex-eq field-simps)
    show cl = circline 0 ?BC ?D
      apply (subst assms, subst line-equation[of z1 z2 ?BC])
      unfolding circline-def
      by (fact, simp, subst ∗, simp add: field-simps)
  qed
qed

lemma circline-line′:
  assumes z1 ≠ z2
  shows circline 0 (i ∗ (z2 − z1)) (Re (− cnj-mix (i ∗ (z2 − z1)) z1)) = line z1 z2
proof−
  let ?B = i ∗ (z2 − z1)
  let ?D = Re (− cnj-mix ?B z1)
  have circline 0 ?B ?D = {z. cnj ?B∗z + ?B∗cnj z + complex-of-real ?D = 0}
    using assms
    by (simp add: circline-def)
  moreover
  have is-real (− cnj-mix (i ∗ (z2 − z1)) z1)
    using cnj-mix-real[of ?B z1]
    by auto
  hence {z. cnj ?B∗z + ?B∗cnj z + complex-of-real ?D = 0} =
        {z. cnj ?B∗z + ?B∗cnj z − (cnj ?B∗z1 + ?B∗cnj z1) = 0}
    apply (subst complex-of-real-Re, simp)
    unfolding diff-conv-add-uminus
    by simp
  moreover
  have line z1 z2 = {z. cnj-mix (i ∗ (z2 − z1)) z − cnj-mix (i ∗ (z2 − z1)) z1 = 0}
    using line-equation[of z1 z2 ?B] assms
    unfolding rot90-ii
    by simp
  ultimately
  show ?thesis
    by simp
qed
```

4.5 Angle between two circles

Given a center μ of an Euclidean circle and a point E on it, we define the tangent vector in E as the radius vector $\overrightarrow{\mu E}$, rotated by $\pi/2$, clockwise or counterclockwise, depending on the circle orientation. The Boolean p

encodes the orientation of the circle, and the function *sgn-bool p* returns 1 when p is true, and -1 when p is false.

abbreviation *sgn-bool* **where**
sgn-bool p ≡ *if p then 1 else −1*

definition *circ-tang-vec* :: *complex* ⇒ *complex* ⇒ *bool* ⇒ *complex* **where**
circ-tang-vec μ E p = *sgn-bool p* ∗ i ∗ (*E* − *μ*)

Tangent vector is orthogonal to the radius.

lemma *circ-tang-vec-ortho*:
shows *scalprod* (*E* − *μ*) (*circ-tang-vec μ E p*) = *0*
unfolding *circ-tang-vec-def Let-def*
by *auto*

Changing the circle orientation gives the opposite tangent vector.

lemma *circ-tang-vec-opposite-orient*:
shows *circ-tang-vec μ E p* = − *circ-tang-vec μ E* (¬ *p*)
unfolding *circ-tang-vec-def*
by *auto*

Angle between two oriented circles at their common point E is defined as the angle between tangent vectors at E. Again we define three different angle measures.

The oriented angle between two circles at the point E. The first circle is centered at μ_1 and its orientation is given by the Boolean p_1, while the second circle is centered at μ_2 and its orientation is given by the Boolea p_2.

definition *ang-circ* **where**
ang-circ E μ1 μ2 p1 p2 = ∠ (*circ-tang-vec μ1 E p1*) (*circ-tang-vec μ2 E p2*)

The unoriented angle between the two circles

definition *ang-circ-c* **where**
ang-circ-c E μ1 μ2 p1 p2 = ∠*c* (*circ-tang-vec μ1 E p1*) (*circ-tang-vec μ2 E p2*)

The acute angle between the two circles

definition *ang-circ-a* **where**
ang-circ-a E μ1 μ2 p1 p2 = ∠*a* (*circ-tang-vec μ1 E p1*) (*circ-tang-vec μ2 E p2*)

Explicit expression for oriented angle between two circles

lemma *ang-circ-simp*:
assumes *E* ≠ *μ1* **and** *E* ≠ *μ2*
shows *ang-circ E μ1 μ2 p1 p2* =
⌊*Arg* (*E* − *μ2*) − *Arg* (*E* − *μ1*) + *sgn-bool p1* ∗ *pi* / *2* − *sgn-bool p2* ∗ *pi* / *2*⌋
unfolding *ang-circ-def ang-vec-def circ-tang-vec-def*
apply (*rule canon-ang-eq*)
using *assms*
using *arg-mult-2kpi*[*of sgn-bool p2*∗i *E* − *μ2*]
using *arg-mult-2kpi*[*of sgn-bool p1*∗i *E* − *μ1*]
apply *auto*
apply (*rule-tac x*=*x*−*xa* **in** *exI*, *auto simp add*: *field-simps*)
apply (*rule-tac x*=−*1*+*x*−*xa* **in** *exI*, *auto simp add*: *field-simps*)
apply (*rule-tac x*=*1*+*x*−*xa* **in** *exI*, *auto simp add*: *field-simps*)
apply (*rule-tac x*=*x*−*xa* **in** *exI*, *auto simp add*: *field-simps*)
done

Explicit expression for the cosine of angle between two circles

lemma *cos-ang-circ-simp*:
assumes *E* ≠ *μ1* **and** *E* ≠ *μ2*
shows *cos* (*ang-circ E μ1 μ2 p1 p2*) =
sgn-bool (*p1* = *p2*) ∗ *cos* (*Arg* (*E* − *μ2*) − *Arg* (*E* − *μ1*))
using *assms*
using *cos-periodic-pi2*[*of Arg* (*E* − *μ2*) − *Arg* (*E* − *μ1*)]
using *cos-minus-pi*[*of Arg* (*E* − *μ2*) − *Arg* (*E* − *μ1*)]
using *ang-circ-simp*[*OF assms*, *of p1 p2*]
by *auto*

Explicit expression for the unoriented angle between two circles

lemma *ang-circ-c-simp*:
assumes $E \neq \mu 1$ **and** $E \neq \mu 2$
shows *ang-circ-c E μ1 μ2 p1 p2* =
$||\downarrow Arg\ (E - \mu 2) - Arg\ (E - \mu 1) + sgn\text{-}bool\ p1 * pi\ /\ 2 - sgn\text{-}bool\ p2 * pi\ /\ 2\downarrow||$
unfolding *ang-circ-c-def ang-vec-c-def*
using *ang-circ-simp*[*OF assms*]
unfolding *ang-circ-def*
by *auto*

Explicit expression for the acute angle between two circles

lemma *ang-circ-a-simp*:
assumes $E \neq \mu 1$ **and** $E \neq \mu 2$
shows *ang-circ-a E μ1 μ2 p1 p2* =
$acute\text{-}ang\ (abs\ (canon\text{-}ang\ (Arg(E - \mu 2) - Arg(E - \mu 1) + (sgn\text{-}bool\ p1) * pi/2 - (sgn\text{-}bool\ p2) * pi/2)))$
unfolding *ang-circ-a-def ang-vec-a-def*
using *ang-circ-c-simp*[*OF assms*]
unfolding *ang-circ-c-def*
by *auto*

Acute angle between two circles does not depend on the circle orientation.

lemma *ang-circ-a-pTrue*:
assumes $E \neq \mu 1$ **and** $E \neq \mu 2$
shows *ang-circ-a E μ1 μ2 p1 p2* = *ang-circ-a E μ1 μ2 True True*
proof (*cases p1*)
case *True*
show *?thesis*
proof (*cases p2*)
case *True*
show *?thesis*
using ‹*p1*› ‹*p2*›
by *simp*
next
case *False*
show *?thesis*
using ‹*p1*› ‹¬ *p2*›
unfolding *ang-circ-a-def*
using *circ-tang-vec-opposite-orient*[*of μ2 E p2*]
using *ang-vec-a-opposite2*
by *simp*
qed
next
case *False*
show *?thesis*
proof (*cases p2*)
case *True*
show *?thesis*
using ‹¬ *p1*› ‹*p2*›
unfolding *ang-circ-a-def*
using *circ-tang-vec-opposite-orient*[*of μ1 E p1*]
using *ang-vec-a-opposite1*
by *simp*
next
case *False*
show *?thesis*
using ‹¬ *p1*› ‹¬ *p2*›
unfolding *ang-circ-a-def*
using *circ-tang-vec-opposite-orient*[*of μ1 E p1*] *circ-tang-vec-opposite-orient*[*of μ2 E p2*]
using *ang-vec-a-opposite1 ang-vec-a-opposite2*
by *simp*
qed
qed

Definition of the acute angle between the two unoriented circles

abbreviation *ang-circ-a′* **where**

ang-circ-a′ E μ1 μ2 ≡ ang-circ-a E μ1 μ2 True True

A very simple expression for the acute angle between the two circles

lemma *ang-circ-a-simp1*:
assumes $E \neq \mu 1$ **and** $E \neq \mu 2$
shows *ang-circ-a E μ1 μ2 p1 p2* $= \angle a\ (E - \mu 1)\ (E - \mu 2)$
unfolding *ang-vec-a-def ang-vec-c-def ang-vec-def*
by (*subst ang-circ-a-pTrue*[*OF assms*, *of p1 p2*], *subst ang-circ-a-simp*[*OF assms*, *of True True*]) (*metis add-diff-cancel*)

lemma *ang-circ-a′-simp*:
assumes $E \neq \mu 1$ **and** $E \neq \mu 2$
shows *ang-circ-a′ E μ1 μ2* $= \angle a\ (E - \mu 1)\ (E - \mu 2)$
by (*rule ang-circ-a-simp1*[*OF assms*])

end

5 Homogeneous coordinates in extended complex plane

Extended complex plane $\overline{\mathbb{C}}$ is complex plane with an additional element (treated as the infinite point). The extended complex plane $\overline{\mathbb{C}}$ is identified with a complex projective line (the one-dimensional projective space over the complex field, sometimes denoted by $\mathbb{C}P^1$). Each point of $\overline{\mathbb{C}}$ is represented by a pair of complex homogeneous coordinates (not both equal to zero), and two pairs of homogeneous coordinates represent the same point in $\overline{\mathbb{C}}$ iff they are proportional by a non-zero complex factor.

theory *Homogeneous-Coordinates*
imports *More-Complex Matrices*
begin

5.1 Definition of homogeneous coordinates

Two complex vectors are equivalent iff they are proportional.

definition *complex-cvec-eq* :: *complex-vec* ⇒ *complex-vec* ⇒ *bool* (**infix** ‹≈$_v$› *50*) **where**
[*simp*]: *z1* $\approx_v$ *z2* ⟷ (∃ *k*. *k* ≠ (*0*::*complex*) ∧ *z2* = *k* $*_{sv}$ *z1*)

lemma *complex-cvec-eq-mix*:
assumes (*z1*, *z2*) ≠ *vec-zero* **and** (*w1*, *w2*) ≠ *vec-zero*
shows (*z1*, *z2*) $\approx_v$ (*w1*, *w2*) ⟷ *z1*∗*w2* = *z2*∗*w1*
proof *safe*
assume (*z1*, *z2*) $\approx_v$ (*w1*, *w2*)
thus *z1* ∗ *w2* = *z2* ∗ *w1*
by *auto*
next
assume ∗: *z1* ∗ *w2* = *z2* ∗ *w1*
show (*z1*, *z2*) $\approx_v$ (*w1*, *w2*)
proof (*cases z2* = *0*)
case *True*
thus *?thesis*
using ∗ *assms*
by *auto*
next
case *False*
hence *w1* = (*w2*/*z2*)∗*z1* ∧ *w2* = (*w2*/*z2*)∗*z2* *w2*/*z2* ≠ *0*
using ∗ *assms*
by (*auto simp add*: *field-simps*)
thus (*z1*, *z2*) $\approx_v$ (*w1*, *w2*)
by (*metis complex-cvec-eq-def mult-sv.simps*)
qed
qed

lemma *complex-eq-cvec-reflp* [*simp*]:
shows *reflp* (≈$_v$)
unfolding *reflp-def complex-cvec-eq-def*
by *safe* (*rule-tac x*=*1* **in** *exI*, *simp*)

lemma *complex-eq-cvec-symp* [*simp*]:
shows *symp* ($\approx_v$)
unfolding *symp-def complex-cvec-eq-def*
by *safe* (*rule-tac x=1/k* **in** *exI*, *simp*)

lemma *complex-eq-cvec-transp* [*simp*]:
shows *transp* ($\approx_v$)
unfolding *transp-def complex-cvec-eq-def*
by *safe* (*rule-tac x=k∗ka* **in** *exI*, *simp*)

lemma *complex-eq-cvec-equivp* [*simp*]:
shows *equivp* ($\approx_v$)
by (*auto intro*: *equivpI*)

Non-zero pairs of complex numbers (also treated as non-zero complex vectors)

typedef *complex-homo-coords* = {*v*::*complex-vec*. $v \neq$ *vec-zero*}
by (*rule-tac x=*(*1*, *0*) **in** *exI*, *simp*)

setup-lifting *type-definition-complex-homo-coords*

lift-definition *complex-homo-coords-eq* :: *complex-homo-coords* ⇒ *complex-homo-coords* ⇒ *bool* (**infix** ‹≈› *50*) **is** *complex-cvec-eq*
done

lemma *complex-homo-coords-eq-reflp* [*simp*]:
shows *reflp* (≈)
using *complex-eq-cvec-reflp*
unfolding *reflp-def*
by *transfer blast*

lemma *complex-homo-coords-eq-symp* [*simp*]:
shows *symp* (≈)
using *complex-eq-cvec-symp*
unfolding *symp-def*
by *transfer blast*

lemma *complex-homo-coords-eq-transp* [*simp*]:
shows *transp* (≈)
using *complex-eq-cvec-transp*
unfolding *transp-def*
by *transfer blast*

lemma *complex-homo-coords-eq-equivp*:
shows *equivp* (≈)
by (*auto intro*: *equivpI*)

lemma *complex-homo-coords-eq-refl* [*simp*]:
shows *z* ≈ *z*
using *complex-homo-coords-eq-reflp*
unfolding *reflp-def refl-on-def*
by *blast*

lemma *complex-homo-coords-eq-sym*:
assumes *z1* ≈ *z2*
shows *z2* ≈ *z1*
using *assms complex-homo-coords-eq-symp*
unfolding *symp-def*
by *blast*

lemma *complex-homo-coords-eq-trans*:
assumes *z1* ≈ *z2* **and** *z2* ≈ *z3*
shows *z1* ≈ *z3*
using *assms complex-homo-coords-eq-transp*
unfolding *transp-def*
by *blast*

Quotient type of homogeneous coordinates

quotient-type
complex-homo = *complex-homo-coords* / *complex-homo-coords-eq*
by (*rule complex-homo-coords-eq-equivp*)

5.2 Some characteristic points in $\mathbb{C}P^1$

Infinite point

definition *inf-cvec* :: *complex-vec* ($\langle\infty_v\rangle$) **where**
[*simp*]: *inf-cvec* = (*1*, *0*)
lift-definition *inf-hcoords* :: *complex-homo-coords* ($\langle\infty_{hc}\rangle$) **is** *inf-cvec*
by *simp*
lift-definition *inf* :: *complex-homo* ($\langle\infty_h\rangle$) **is** *inf-hcoords*
done

lemma *inf-cvec-z2-zero-iff*:
assumes (*z1*, *z2*) ≠ *vec-zero*
shows (*z1*, *z2*) ≈$_v$ ∞$_v$ ⟷ *z2* = *0*
using *assms*
by *auto*

Zero

definition *zero-cvec* :: *complex-vec* ($\langle 0_v\rangle$) **where**
[*simp*]: *zero-cvec* = (*0*, *1*)
lift-definition *zero-hcoords* :: *complex-homo-coords* ($\langle 0_{hc}\rangle$) **is** *zero-cvec*
by *simp*
lift-definition *zero* :: *complex-homo* ($\langle 0_h\rangle$) **is** *zero-hcoords*
done

lemma *zero-cvec-z1-zero-iff*:
assumes (*z1*, *z2*) ≠ *vec-zero*
shows (*z1*, *z2*) ≈$_v$ *0*$_v$ ⟷ *z1* = *0*
using *assms*
by *auto*

One

definition *one-cvec* :: *complex-vec* ($\langle 1_v\rangle$)**where**
[*simp*]: *one-cvec* = (*1*, *1*)
lift-definition *one-hcoords* :: *complex-homo-coords* ($\langle 1_{hc}\rangle$) **is** *one-cvec*
by *simp*
lift-definition *one* :: *complex-homo* ($\langle 1_h\rangle$) **is** *one-hcoords*
done

lemma *zero-one-infty-not-equal* [*simp*]:
shows *1*$_h$ ≠ ∞$_h$ **and** *0*$_h$ ≠ ∞$_h$ **and** *0*$_h$ ≠ *1*$_h$ **and** *1*$_h$ ≠ *0*$_h$ **and** ∞$_h$ ≠ *0*$_h$ **and** ∞$_h$ ≠ *1*$_h$
by (*transfer*, *transfer*, *simp*)+

Imaginary unit

definition *ii-cvec* :: *complex-vec* ($\langle ii_v\rangle$) **where**
[*simp*]: *ii-cvec* = (i, *1*)
lift-definition *ii-hcoords* :: *complex-homo-coords* ($\langle ii_{hc}\rangle$) **is** *ii-cvec*
by *simp*
lift-definition *ii* :: *complex-homo* ($\langle ii_h\rangle$) **is** *ii-hcoords*
done

lemma *ex-3-different-points*:
fixes *z*::*complex-homo*
shows ∃ *z1 z2*. *z* ≠ *z1* ∧ *z1* ≠ *z2* ∧ *z* ≠ *z2*
proof (*cases z* ≠ *0*$_h$ ∧ *z* ≠ *1*$_h$)
case *True*
thus *?thesis*
by (*rule-tac x*=*0*$_h$ **in** *exI*, *rule-tac x*=*1*$_h$ **in** *exI*, *auto*)
next
case *False*

hence $z = 0_h \vee z = 1_h$
by *simp*
thus *?thesis*
proof
assume $z = 0_h$
thus *?thesis*
by (*rule-tac* $x{=}\infty_h$ **in** *exI*, *rule-tac* $x{=}1_h$ **in** *exI*, *auto*)
next
assume $z = 1_h$
thus *?thesis*
by (*rule-tac* $x{=}\infty_h$ **in** *exI*, *rule-tac* $x{=}0_h$ **in** *exI*, *auto*)
qed
qed

5.3 Connection to ordinary complex plane $\mathbb{C}$

Conversion from complex

definition *of-complex-cvec* :: *complex* ⇒ *complex-vec* **where**
[*simp*]: *of-complex-cvec z* = (*z*, *1*)
lift-definition *of-complex-hcoords* :: *complex* ⇒ *complex-homo-coords* **is** *of-complex-cvec*
by *simp*
lift-definition *of-complex* :: *complex* ⇒ *complex-homo* **is** *of-complex-hcoords*
done

lemma *of-complex-inj*:
assumes *of-complex x* = *of-complex y*
shows *x* = *y*
using *assms*
by (*transfer*, *transfer*, *simp*)

lemma *of-complex-image-inj*:
assumes *of-complex* ‘ *A* = *of-complex* ‘ *B*
shows *A* = *B*
using *assms*
using *of-complex-inj*
by *auto*

lemma *of-complex-not-inf* [*simp*]:
shows *of-complex x* ≠ ∞_h
by (*transfer*, *transfer*, *simp*)

lemma *inf-not-of-complex* [*simp*]:
shows ∞_h ≠ *of-complex x*
by (*transfer*, *transfer*, *simp*)

lemma *inf-or-of-complex*:
shows *z* = ∞_h ∨ (∃ *x*. *z* = *of-complex x*)
proof (*transfer*, *transfer*)
fix *z* :: *complex-vec*
obtain *z1 z2* **where** ∗: *z* = (*z1*, *z2*)
by (*cases z*) *auto*
assume *z* ≠ *vec-zero*
thus *z* ≈$_v$ ∞_v ∨ (∃ *x*. *z* ≈$_v$ *of-complex-cvec x*)
using ∗
by (*cases z2* = *0*, *auto*)
qed

lemma *of-complex-zero* [*simp*]:
shows *of-complex 0* = 0_h
by (*transfer*, *transfer*, *simp*)

lemma *of-complex-one* [*simp*]:
shows *of-complex 1* = 1_h
by (*transfer*, *transfer*, *simp*)

lemma *of-complex-ii* [*simp*]:

shows *of-complex* i = ii_h
by (*transfer*, *transfer*, *simp*)

lemma *of-complex-zero-iff* [*simp*]:
shows *of-complex x* = $0_h \longleftrightarrow x = 0$
by (*subst of-complex-zero*[*symmetric*]) (*auto simp add*: *of-complex-inj*)

lemma *of-complex-one-iff* [*simp*]:
shows *of-complex x* = $1_h \longleftrightarrow x = 1$
by (*subst of-complex-one*[*symmetric*]) (*auto simp add*: *of-complex-inj*)

lemma *of-complex-ii-iff* [*simp*]:
shows *of-complex x* = $ii_h \longleftrightarrow x$ = i
by (*subst of-complex-ii*[*symmetric*]) (*auto simp add*: *of-complex-inj*)

Conversion to complex

definition *to-complex-cvec* :: *complex-vec* ⇒ *complex* **where**
[*simp*]: *to-complex-cvec z* = (*let* (*z1*, *z2*) = *z in z1*/*z2*)
lift-definition *to-complex-homo-coords* :: *complex-homo-coords* ⇒ *complex* **is** *to-complex-cvec*
done
lift-definition *to-complex* :: *complex-homo* ⇒ *complex* **is** *to-complex-homo-coords*
proof−
fix *z w*
assume *z* ≈ *w*
thus *to-complex-homo-coords z* = *to-complex-homo-coords w*
by *transfer auto*
qed

lemma *to-complex-of-complex* [*simp*]:
shows *to-complex* (*of-complex z*) = *z*
by (*transfer*, *transfer*, *simp*)

lemma *of-complex-to-complex* [*simp*]:
assumes *z* ≠ ∞_h
shows (*of-complex* (*to-complex z*)) = *z*
using *assms*
proof (*transfer*, *transfer*)
fix *z* :: *complex-vec*
obtain *z1 z2* **where** ∗: *z* = (*z1*, *z2*)
by (*cases z*, *auto*)
assume *z* ≠ *vec-zero* ¬ *z* ≈$_v$ ∞_v
hence *z2* ≠ *0*
using ∗
by (*simp*, *erule-tac x*=*1*/*z1* **in** *allE*, *auto*)
thus (*of-complex-cvec* (*to-complex-cvec z*)) ≈$_v$ *z*
using ∗
by *simp*
qed

lemma *to-complex-zero-zero* [*simp*]:
shows *to-complex* 0_h = *0*
by (*metis of-complex-zero to-complex-of-complex*)

lemma *to-complex-one-one* [*simp*]:
shows *to-complex* 1_h = *1*
by (*metis of-complex-one to-complex-of-complex*)

lemma *to-complex-img-one* [*simp*]:
shows *to-complex* ii_h = i
by (*metis of-complex-ii to-complex-of-complex*)

5.4 Arithmetic operations

Due to the requirement of HOL that all functions are total, we could not define the function only for the well-defined cases, and in the lifting proofs we must also handle the ill-defined cases. For example, $\infty_h +_h \infty_h$ is ill-defined, but we must define it, so we define it arbitrarily to be ∞_h.

5.4.1 Addition

$\infty_h \ +_h \ \infty_h$ is ill-defined. Since functions must be total, for formal reasons we define it arbitrarily to be ∞_h.

definition *add-cvec* :: *complex-vec* ⇒ *complex-vec* ⇒ *complex-vec* (**infixl** ‹$+_v$› *60*) **where**
  [*simp*]: *add-cvec z w* = (*let* (*z1*, *z2*) = *z*; (*w1*, *w2*) = *w*
    in if z2 ≠ *0* ∨ *w2* ≠ *0 then*
      (*z1*∗*w2* + *w1*∗*z2*, *z2*∗*w2*)
    else
      (*1*, *0*))
lift-definition *add-hcoords* :: *complex-homo-coords* ⇒ *complex-homo-coords* ⇒ *complex-homo-coords* (**infixl** ‹$+_{hc}$› *60*)
is *add-cvec*
  by (*auto split*: *if-split-asm*)

lift-definition *add* :: *complex-homo* ⇒ *complex-homo* ⇒ *complex-homo* (**infixl** ‹$+_h$› *60*) **is** *add-hcoords*
proof *transfer*
  fix *z w z′ w′* :: *complex-vec*
  obtain *z1 z2 w1 w2 z′1 z′2 w′1 w′2* **where**
    ∗: *z* = (*z1*, *z2*) *w* = (*w1*, *w2*) *z′* = (*z′1*, *z′2*) *w′* = (*w′1*, *w′2*)
    by (*cases z*, *auto*, *cases w*, *auto*, *cases z′*, *auto*, *cases w′*, *auto*)
  assume ∗∗:
      z ≠ *vec-zero w* ≠ *vec-zero z* ≈$_v$ *z′*
      z′ ≠ *vec-zero w′* ≠ *vec-zero w* ≈$_v$ *w′*
  show *z* +$_v$ *w* ≈$_v$ *z′* +$_v$ *w′*
  proof (*cases z2* ≠ *0* ∨ *w2* ≠ *0*)
    case *True*
    hence *z′2* ≠ *0* ∨ *w′2* ≠ *0*
      using ∗ ∗∗
      by *auto*
    show *?thesis*
      using ‹*z2* ≠ *0* ∨ *w2* ≠ *0*› ‹*z′2* ≠ *0* ∨ *w′2* ≠ *0*›
      using ∗ ∗∗
      by *simp* ((*erule exE*)+, *rule-tac x*=*k*∗*ka* **in** *exI*, *simp add*: *field-simps*)
  next
    case *False*
    hence *z′2* = *0* ∨ *w′2* = *0*
      using ∗ ∗∗
      by *auto*
    show *?thesis*
      using ‹¬ (*z2* ≠ *0* ∨ *w2* ≠ *0*)› ‹*z′2* = *0* ∨ *w′2* = *0*›
      using ∗ ∗∗
      by *auto*
  qed
qed

lemma *add-commute*:
  shows *z* +$_h$ *w* = *w* +$_h$ *z*
  apply (*transfer*, *transfer*)
  unfolding *complex-cvec-eq-def*
  by (*rule-tac x*=*1* **in** *exI*, *auto split*: *if-split-asm*)

lemma *add-zero-right* [*simp*]:
  shows *z* +$_h$ *0*$_h$ = *z*
  by (*transfer*, *transfer*, *force*)

lemma *add-zero-left* [*simp*]:
  shows *0*$_h$ +$_h$ *z* = *z*
  by (*subst add-commute*) *simp*

lemma *of-complex-add-of-complex* [*simp*]:
  shows (*of-complex x*) +$_h$ (*of-complex y*) = *of-complex* (*x* + *y*)
  by (*transfer*, *transfer*, *simp*)

lemma *of-complex-add-inf* [*simp*]:
  shows (*of-complex x*) +$_h$ ∞$_h$ = ∞$_h$
  by (*transfer*, *transfer*, *simp*)

lemma *inf-add-of-complex* [*simp*]:
shows $\infty_h +_h (\textit{of-complex}\ x) = \infty_h$
by (*subst add-commute*) *simp*

lemma *inf-add-right*:
assumes $z \neq \infty_h$
shows $z +_h \infty_h = \infty_h$
using *assms*
using *inf-or-of-complex*[*of z*]
by *auto*

lemma *inf-add-left*:
assumes $z \neq \infty_h$
shows $\infty_h +_h z = \infty_h$
using *assms*
by (*subst add-commute*) (*rule inf-add-right*, *simp*)

This is ill-defined, but holds by our definition

lemma *inf-add-inf*:
shows $\infty_h +_h \infty_h = \infty_h$
by (*transfer*, *transfer*, *simp*)

5.4.2 Unary minus

definition *uminus-cvec* :: *complex-vec* ⇒ *complex-vec* ($\langle{}^\sim{}_v\rangle$) **where**
[*simp*]: ${}^\sim{}_v\ z = (\textit{let}\ (z1, z2) = z\ \textit{in}\ (-z1, z2))$
lift-definition *uminus-hcoords* :: *complex-homo-coords* ⇒ *complex-homo-coords* ($\langle{}^\sim{}_{hc}\rangle$) **is** *uminus-cvec*
by *auto*
lift-definition *uminus* :: *complex-homo* ⇒ *complex-homo* ($\langle{}^\sim{}_h\rangle$) **is** *uminus-hcoords*
by *transfer auto*

lemma *uminus-of-complex* [*simp*]:
shows ${}^\sim{}_h\ (\textit{of-complex}\ z) = \textit{of-complex}\ (-z)$
by (*transfer*, *transfer*, *simp*)

lemma *uminus-zero* [*simp*]:
shows ${}^\sim{}_h\ 0_h = 0_h$
by (*transfer*, *transfer*, *simp*)

lemma *uminus-inf* [*simp*]:
shows ${}^\sim{}_h\ \infty_h = \infty_h$
apply (*transfer*, *transfer*)
unfolding *complex-cvec-eq-def*
by (*rule-tac x=−1* **in** *exI*, *simp*)

lemma *uminus-inf-iff*:
shows ${}^\sim{}_h\ z = \infty_h \longleftrightarrow z = \infty_h$
apply (*transfer*, *transfer*)
by *auto* (*rule-tac x=−1/a* **in** *exI*, *auto*)

lemma *uminus-id-iff*:
shows ${}^\sim{}_h\ z = z \longleftrightarrow z = 0_h \vee z = \infty_h$
apply (*transfer*, *transfer*)
apply *auto*
apply (*erule-tac x=1/a* **in** *allE*, *simp*)
apply (*rule-tac x=−1* **in** *exI*, *simp*)
done

5.4.3 Subtraction

Operation $\infty_h\ -_h\ \infty_h$ is ill-defined, but we define it arbitrarily to 0_h. It breaks the connection between subtraction with addition and unary minus, but seems more intuitive.

definition *sub* :: *complex-homo* ⇒ *complex-homo* ⇒ *complex-homo* (**infixl** $\langle -_h\rangle$ *60*) **where**
$z -_h w = (\textit{if}\ z = \infty_h \wedge w = \infty_h\ \textit{then}\ 0_h\ \textit{else}\ z +_h ({}^\sim{}_h\ w))$

lemma *of-complex-sub-of-complex* [*simp*]:

shows (*of-complex x*) $-_h$ (*of-complex y*) = *of-complex* (*x* − *y*)
unfolding *sub-def*
by *simp*

lemma *zero-sub-right*[*simp*]:
shows *z* $-_h$ 0_h = *z*
unfolding *sub-def*
by *simp*

lemma *zero-sub-left*[*simp*]:
shows 0_h $-_h$ *of-complex x* = *of-complex* (−*x*)
by (*subst of-complex-zero*[*symmetric*], *simp del*: *of-complex-zero*)

lemma *zero-sub-one*[*simp*]:
shows 0_h $-_h$ 1_h = *of-complex* (−*1*)
by (*metis of-complex-one zero-sub-left*)

lemma *of-complex-sub-one* [*simp*]:
shows *of-complex x* $-_h$ 1_h = *of-complex* (*x* − *1*)
by (*metis of-complex-one of-complex-sub-of-complex*)

lemma *sub-eq-zero* [*simp*]:
assumes *z* ≠ ∞_h
shows *z* $-_h$ *z* = 0_h
using *assms*
using *inf-or-of-complex*[*of z*]
by *auto*

lemma *sub-eq-zero-iff*:
assumes *z* ≠ ∞_h ∨ *w* ≠ ∞_h
shows *z* $-_h$ *w* = 0_h ⟷ *z* = *w*
proof
assume *z* $-_h$ *w* = 0_h
thus *z* = *w*
using *assms*
unfolding *sub-def*
proof (*transfer*, *transfer*)
fix *z w* :: *complex-vec*
obtain *z1 z2 w1 w2* **where** ∗: *z* = (*z1*, *z2*) *w* = (*w1*, *w2*)
by (*cases z*, *auto*, *cases w*, *auto*)
assume *z* ≠ *vec-zero w* ≠ *vec-zero* ¬ *z* ≈$_v$ ∞_v ∨ ¬ *w* ≈$_v$ ∞_v **and**
∗∗: (*if z* ≈$_v$ ∞_v ∧ *w* ≈$_v$ ∞_v *then* 0_v *else z* +$_v$ ~$_v$ *w*) ≈$_v$ 0_v
have *z2* ≠ *0* ∨ *w2* ≠ *0*
using ∗ ‹¬ *z* ≈$_v$ ∞_v ∨ ¬ *w* ≈$_v$ ∞_v› ‹*z* ≠ *vec-zero*› ‹*w* ≠ *vec-zero*›
apply *auto*
apply (*erule-tac x=1/z1* **in** *allE*, *simp*)
apply (*erule-tac x=1/w1* **in** *allE*, *simp*)
done

thus *z* ≈$_v$ *w*
using ∗ ∗∗
by *simp* (*rule-tac x=w2/z2* **in** *exI*, *auto simp add*: *field-simps*)
qed
next
assume *z* = *w*
thus *z* $-_h$ *w* = 0_h
using *sub-eq-zero*[*of z*] *assms*
by *auto*
qed

lemma *inf-sub-left* [*simp*]:
assumes *z* ≠ ∞_h
shows ∞_h $-_h$ *z* = ∞_h
using *assms*
using *uminus-inf-iff*
using *inf-or-of-complex*

unfolding *sub-def*
by *force*

lemma *inf-sub-right* [*simp*]:
assumes $z \neq \infty_h$
shows $z -_h \infty_h = \infty_h$
using *assms*
using *inf-or-of-complex*
unfolding *sub-def*
by *force*

This is ill-defined, but holds by our definition

lemma *inf-sub-inf*:
shows $\infty_h -_h \infty_h = 0_h$
unfolding *sub-def*
by *simp*

lemma *sub-noteq-inf*:
assumes $z \neq \infty_h$ **and** $w \neq \infty_h$
shows $z -_h w \neq \infty_h$
using *assms*
using *inf-or-of-complex*[*of z*]
using *inf-or-of-complex*[*of w*]
using *inf-or-of-complex*[*of* $z -_h w$]
using *of-complex-sub-of-complex*
by *auto*

lemma *sub-eq-inf*:
assumes $z -_h w = \infty_h$
shows $z = \infty_h \vee w = \infty_h$
using *assms sub-noteq-inf*
by *blast*

5.4.4 Multiplication

Operations $0_h \cdot_h \infty_h$ and $\infty_h \cdot_h 0_h$ are ill defined. Since all functions must be total, for formal reasons we define it arbitrarily to be 1_h.

definition *mult-cvec* :: *complex-vec* ⇒ *complex-vec* ⇒ *complex-vec* (**infixl** ‹$*_v$› *70*) **where**
[*simp*]: $z *_v w$ = (*let* (*z1*, *z2*) = *z*; (*w1*, *w2*) = *w*
in if (*z1* = 0 ∧ *w2* = 0) ∨ (*w1* = 0 ∧ *z2* = 0) *then*
(*1*, *1*)
else
(*z1*∗*w1*, *z2*∗*w2*))
lift-definition *mult-hcoords* :: *complex-homo-coords* ⇒ *complex-homo-coords* ⇒ *complex-homo-coords* (**infixl** ‹$*_{hc}$› *70*)
is *mult-cvec*
by (*auto split*: *if-split-asm*)

lift-definition *mult* :: *complex-homo* ⇒ *complex-homo* ⇒ *complex-homo* (**infixl** ‹$*_h$› *70*) **is** *mult-hcoords*
proof *transfer*
fix *z w z′ w′* :: *complex-vec*
obtain *z1 z2 w1 w2 z′1 z′2 w′1 w′2* **where**
∗: *z* = (*z1*, *z2*) *w* = (*w1*, *w2*) *z′* = (*z′1*, *z′2*) *w′* = (*w′1*, *w′2*)
by (*cases z*, *auto*, *cases w*, *auto*, *cases z′*, *auto*, *cases w′*, *auto*)
assume ∗∗:
z ≠ *vec-zero w* ≠ *vec-zero z* ≈$_v$ *z′*
z′ ≠ *vec-zero w′* ≠ *vec-zero w* ≈$_v$ *w′*
show *z* ∗$_v$ *w* ≈$_v$ *z′* ∗$_v$ *w′*
proof (*cases* (*z1* = 0 ∧ *w2* = 0) ∨ (*w1* = 0 ∧ *z2* = 0))
case *True*
hence (*z′1* = 0 ∧ *w′2* = 0) ∨ (*w′1* = 0 ∧ *z′2* = 0)
using ∗ ∗∗
by *auto*
show *?thesis*
using ‹(*z1* = 0 ∧ *w2* = 0) ∨ (*w1* = 0 ∧ *z2* = 0)› ‹(*z′1* = 0 ∧ *w′2* = 0) ∨ (*w′1* = 0 ∧ *z′2* = 0)›
using ∗ ∗∗

```
    by simp
  next
    case False
    hence ¬((z'1 = 0 ∧ w'2 = 0) ∨ (w'1 = 0 ∧ z'2 = 0))
      using * **
      by auto
    hence ***: z *v w = (z1*w1, z2*w2) z' *v w' = (z'1*w'1, z'2*w'2)
      using ‹¬((z1 = 0 ∧ w2 = 0) ∨ (w1 = 0 ∧ z2 = 0))› ‹¬((z'1 = 0 ∧ w'2 = 0) ∨ (w'1 = 0 ∧ z'2 = 0))›
      using *
      by auto
    show ?thesis
      apply (subst ***)+
      using * **
      by simp ((erule exE)+, rule-tac x=k*ka in exI, simp)
  qed
qed

lemma of-complex-mult-of-complex [simp]:
  shows (of-complex z1) *h (of-complex z2) = of-complex (z1 * z2)
  by (transfer, transfer, simp)

lemma mult-commute:
  shows z1 *h z2 = z2 *h z1
  apply (transfer, transfer)
  unfolding complex-cvec-eq-def
  by (rule-tac x=1 in exI, auto split: if-split-asm)

lemma mult-zero-left [simp]:
  assumes z ≠ ∞h
  shows 0h *h z = 0h
  using assms
proof (transfer, transfer)
  fix z :: complex-vec
  obtain z1 z2 where *: z = (z1, z2)
    by (cases z, auto)
  assume z ≠ vec-zero ¬ (z ≈v ∞v)
  hence z2 ≠ 0
    using *
    by force
  thus 0v *v z ≈v 0v
    using *
    by simp
qed

lemma mult-zero-right [simp]:
  assumes z ≠ ∞h
  shows z *h 0h = 0h
  using mult-zero-left[OF assms]
  by (simp add: mult-commute)

lemma mult-inf-right [simp]:
  assumes z ≠ 0h
  shows z *h ∞h = ∞h
using assms
proof (transfer, transfer)
  fix z :: complex-vec
  obtain z1 z2 where *: z = (z1, z2)
    by (cases z, auto)
  assume z ≠ vec-zero ¬ (z ≈v 0v)
  hence z1 ≠ 0
    using *
    by force
  thus z *v ∞v ≈v ∞v
    using *
    by simp
qed
```

lemma *mult-inf-left* [*simp*]:
assumes $z \neq 0_h$
shows $\infty_h *_h z = \infty_h$
using *mult-inf-right*[*OF assms*]
by (*simp add*: *mult-commute*)

lemma *mult-one-left* [*simp*]:
shows $1_h *_h z = z$
by (*transfer*, *transfer*, *force*)

lemma *mult-one-right* [*simp*]:
shows $z *_h 1_h = z$
using *mult-one-left*[*of z*]
by (*simp add*: *mult-commute*)

This is ill-defined, but holds by our definition

lemma *inf-mult-zero*:
shows $\infty_h *_h 0_h = 1_h$
by (*transfer*, *transfer*, *simp*)
lemma *zero-mult-inf*:
shows $0_h *_h \infty_h = 1_h$
by (*transfer*, *transfer*, *simp*)

lemma *mult-eq-inf*:
assumes $z *_h w = \infty_h$
shows $z = \infty_h \vee w = \infty_h$
using *assms*
using *inf-or-of-complex*[*of z*]
using *inf-or-of-complex*[*of w*]
using *inf-or-of-complex*[*of* $z *_h w$]
using *of-complex-mult-of-complex*
by *auto*

lemma *mult-noteq-inf*:
assumes $z \neq \infty_h$ **and** $w \neq \infty_h$
shows $z *_h w \neq \infty_h$
using *assms mult-eq-inf*
by *blast*

5.4.5 Reciprocal

definition *reciprocal-cvec* :: *complex-vec* ⇒ *complex-vec* **where**
[*simp*]: *reciprocal-cvec z* = (*let* (*z1*, *z2*) = *z in* (*z2*, *z1*))
lift-definition *reciprocal-hcoords* :: *complex-homo-coords* ⇒ *complex-homo-coords* **is** *reciprocal-cvec*
by *auto*

lift-definition *reciprocal* :: *complex-homo* ⇒ *complex-homo* **is** *reciprocal-hcoords*
by *transfer auto*

lemma *reciprocal-involution* [*simp*]: *reciprocal* (*reciprocal z*) = *z*
by (*transfer*, *transfer*, *auto*)

lemma *reciprocal-zero* [*simp*]: *reciprocal* $0_h = \infty_h$
by (*transfer*, *transfer*, *simp*)

lemma *reciprocal-inf* [*simp*]: *reciprocal* $\infty_h = 0_h$
by (*transfer*, *transfer*, *simp*)

lemma *reciprocal-one* [*simp*]: *reciprocal* $1_h = 1_h$
by (*transfer*, *transfer*, *simp*)

lemma *reciprocal-inf-iff* [*iff*]: *reciprocal* $z = \infty_h \longleftrightarrow z = 0_h$
by (*transfer*, *transfer*, *auto*)

lemma *reciprocal-zero-iff* [*iff*]: *reciprocal* $z = 0_h \longleftrightarrow z = \infty_h$

by (*transfer*, *transfer*, *auto*)

lemma *reciprocal-of-complex* [*simp*]:
assumes $z \neq 0$
shows *reciprocal* (*of-complex z*) = *of-complex* (*1 / z*)
using *assms*
by (*transfer*, *transfer*, *simp*)

lemma *reciprocal-real*:
assumes *is-real* (*to-complex z*) **and** $z \neq 0_h$ **and** $z \neq \infty_h$
shows *Re* (*to-complex* (*reciprocal z*)) = *1 / Re* (*to-complex z*)
proof −
obtain *c* **where** *z* = *of-complex c c* ≠ *0 is-real c*
using *assms inf-or-of-complex*[*of z*]
by *auto*
thus *?thesis*
by (*simp add*: *Re-divide-real*)
qed

lemma *reciprocal-id-iff*:
shows *reciprocal z* = *z* ⟷ *z* = *of-complex 1* ∨ *z* = *of-complex* (−*1*)
proof (*cases z* = 0_h)
case *True*
thus *?thesis*
by (*metis inf-not-of-complex of-complex-zero-iff reciprocal-inf-iff zero-neq-neg-one zero-neq-one*)
next
case *False*
thus *?thesis*
using *inf-or-of-complex*[*of z*]
by (*smt* (*verit*) *complex-sqrt-1 of-complex-zero-iff reciprocal-inf-iff reciprocal-of-complex to-complex-of-complex*)
qed

5.4.6 Division

Operations $0_h :_h 0_h$ and $\infty_h :_h \infty_h$ are ill-defined. For formal reasons they are defined to be 1_h (by the definition of multiplication).

definition *divide* :: *complex-homo* ⇒ *complex-homo* ⇒ *complex-homo* (**infixl** ‹:$_h$› *70*) **where**
x :$_h$ *y* = *x* ∗$_h$ (*reciprocal y*)

lemma *divide-zero-right* [*simp*]:
assumes $z \neq 0_h$
shows $z :_h 0_h = \infty_h$
using *assms*
unfolding *divide-def*
by *simp*

lemma *divide-zero-left* [*simp*]:
assumes $z \neq 0_h$
shows $0_h :_h z = 0_h$
using *assms*
unfolding *divide-def*
by *simp*

lemma *divide-inf-right* [*simp*]:
assumes $z \neq \infty_h$
shows $z :_h \infty_h = 0_h$
using *assms*
unfolding *divide-def*
by *simp*

lemma *divide-inf-left* [*simp*]:
assumes $z \neq \infty_h$
shows $\infty_h :_h z = \infty_h$
using *assms reciprocal-zero-iff*[*of z*] *mult-inf-left*
unfolding *divide-def*
by *simp*

lemma *divide-eq-inf*:
assumes $z :_h w = \infty_h$
shows $z = \infty_h \vee w = 0_h$
using *assms*
using *reciprocal-inf-iff*[*of w*] *mult-eq-inf*
unfolding *divide-def*
by *auto*

lemma *inf-divide-zero* [*simp*]:
shows $\infty_h :_h 0_h = \infty_h$
unfolding *divide-def*
by (*transfer*, *simp*)

lemma *zero-divide-inf* [*simp*]:
shows $0_h :_h \infty_h = 0_h$
unfolding *divide-def*
by (*transfer*, *simp*)

lemma *divide-one-right* [*simp*]:
shows $z :_h 1_h = z$
unfolding *divide-def*
by *simp*

lemma *of-complex-divide-of-complex* [*simp*]:
assumes $z2 \neq 0$
shows (*of-complex z1*) $:_h$ (*of-complex z2*) = *of-complex* (*z1* / *z2*)
using *assms*
unfolding *divide-def*
apply *transfer*
apply *transfer*
by (*simp*, *rule-tac x=1/z2* **in** *exI*, *simp*)

lemma *one-div-of-complex* [*simp*]:
assumes $x \neq 0$
shows $1_h :_h$ *of-complex x* = *of-complex* (*1* / *x*)
using *assms*
unfolding *divide-def*
by *simp*

This is ill-defined, but holds by our definition

lemma *inf-divide-inf*:
shows $\infty_h :_h \infty_h = 1_h$
unfolding *divide-def*
by (*simp add*: *inf-mult-zero*)

This is ill-defined, but holds by our definition

lemma *zero-divide-zero*:
shows $0_h :_h 0_h = 1_h$
unfolding *divide-def*
by (*simp add*: *zero-mult-inf*)

5.4.7 Conjugate

definition *conjugate-cvec* :: *complex-vec* ⇒ *complex-vec* **where**
[*simp*]: *conjugate-cvec z* = *vec-cnj z*
lift-definition *conjugate-hcoords* :: *complex-homo-coords* ⇒ *complex-homo-coords* **is** *conjugate-cvec*
by (*auto simp add*: *vec-cnj-def*)
lift-definition *conjugate* :: *complex-homo* ⇒ *complex-homo* **is** *conjugate-hcoords*
by *transfer* (*auto simp add*: *vec-cnj-def*)

lemma *conjugate-involution* [*simp*]:
shows *conjugate* (*conjugate z*) = *z*
by (*transfer*, *transfer*, *auto*)

lemma *conjugate-conjugate-comp* [*simp*]:

shows *conjugate* ∘ *conjugate* = *id*
by (*rule ext*, *simp*)

lemma *inv-conjugate* [*simp*]:
shows *inv conjugate* = *conjugate*
using *inv-unique-comp*[*of conjugate conjugate*]
by *simp*

lemma *conjugate-of-complex* [*simp*]:
shows *conjugate* (*of-complex z*) = *of-complex* (*cnj z*)
by (*transfer*, *transfer*, *simp add*: *vec-cnj-def*)

lemma *conjugate-inf* [*simp*]:
shows *conjugate* ∞_h = ∞_h
by (*transfer*, *transfer*, *simp add*: *vec-cnj-def*)

lemma *conjugate-zero* [*simp*]:
shows *conjugate* 0_h = 0_h
by (*transfer*, *transfer*, *simp add*: *vec-cnj-def*)

lemma *conjugate-one* [*simp*]:
shows *conjugate* 1_h = 1_h
by (*transfer*, *transfer*, *simp add*: *vec-cnj-def*)

lemma *conjugate-inj*:
assumes *conjugate x* = *conjugate y*
shows *x* = *y*
using *assms*
using *conjugate-involution*[*of x*] *conjugate-involution*[*of y*]
by *metis*

lemma *bij-conjugate* [*simp*]:
shows *bij conjugate*
unfolding *bij-def inj-on-def*
proof *auto*
fix *x y*
assume *conjugate x* = *conjugate y*
thus *x* = *y*
by (*simp add*: *conjugate-inj*)
next
fix *x*
show *x* ∈ *range conjugate*
by (*metis conjugate-involution range-eqI*)
qed

lemma *conjugate-id-iff*:
shows *conjugate a* = *a* ⟷ *is-real* (*to-complex a*) ∨ *a* = ∞_h
using *inf-or-of-complex*[*of a*]
by (*metis conjugate-inf conjugate-of-complex eq-cnj-iff-real to-complex-of-complex*)

5.4.8 Inversion

Geometric inversion wrt. the unit circle

definition *inversion* **where**
inversion = *conjugate* ∘ *reciprocal*

lemma *inversion-sym*:
shows *inversion* = *reciprocal* ∘ *conjugate*
unfolding *inversion-def*
apply (*rule ext*, *simp*)
apply *transfer*
apply *transfer*
apply (*auto simp add*: *vec-cnj-def*)
using *one-neq-zero*
by *blast+*

lemma *inversion-involution* [*simp*]:
shows *inversion* (*inversion z*) = *z*
proof−
have ∗: *conjugate* ∘ *reciprocal* = *reciprocal* ∘ *conjugate*
using *inversion-sym*
by (*simp add*: *inversion-def*)
show *?thesis*
unfolding *inversion-def*
by (*subst* ∗) *simp*
qed

lemma *inversion-inversion-id* [*simp*]:
shows *inversion* ∘ *inversion* = *id*
by (*rule ext*, *simp*)

lemma *inversion-zero* [*simp*]:
shows *inversion* 0_h = ∞_h
by (*simp add*: *inversion-def*)

lemma *inversion-infty* [*simp*]:
shows *inversion* ∞_h = 0_h
by (*simp add*: *inversion-def*)

lemma *inversion-of-complex* [*simp*]:
assumes *z* ≠ *0*
shows *inversion* (*of-complex z*) = *of-complex* (*1* / *cnj z*)
using *assms*
by (*simp add*: *inversion-def*)

lemma *is-real-inversion*:
assumes *is-real x* **and** *x* ≠ *0*
shows *is-real* (*to-complex* (*inversion* (*of-complex x*)))
using *assms eq-cnj-iff-real*[*of x*]
by *simp*

lemma *inversion-id-iff*:
shows *a* = *inversion a* ⟷ *a* ≠ ∞_h ∧ (*to-complex a*) ∗ *cnj* (*to-complex a*) = *1* (**is** *?lhs* = *?rhs*)
proof
assume *a* = *inversion a*
thus *?rhs*
unfolding *inversion-def*
using *inf-or-of-complex*[*of a*]
by (*metis* (*full-types*) *comp-apply complex-cnj-cancel-iff complex-cnj-zero inversion-def inversion-infty inversion-of-complex inversion-sym nonzero-eq-divide-eq of-complex-zero reciprocal-zero to-complex-of-complex zero-one-infty-not-equal*(*5*))
next
assume *?rhs*
thus *?lhs*
using *inf-or-of-complex*[*of a*]
by (*metis inversion-of-complex mult-not-zero nonzero-mult-div-cancel-right one-neq-zero to-complex-of-complex*)
qed

5.5 Ratio and cross-ratio

5.5.1 Ratio

Ratio of points z, v and w is usually defined as $\frac{z-v}{z-w}$. Our definition introduces it in homogeneous coordinates. It is well-defined if $z_1 \neq z_2 \vee z_1 \neq z_3$ and $z_1 \neq \infty_h$ and $z_2 \neq \infty_h \vee z_3 \neq \infty_h$

definition *ratio* :: *complex-homo* ⇒ *complex-homo* ⇒ *complex-homo* ⇒ *complex-homo* **where**
ratio za zb zc = (*za* −$_h$ *zb*) :$_h$ (*za* −$_h$ *zc*)

This is ill-defined, but holds by our definition

lemma
assumes *zb* ≠ ∞_h **and** *zc* ≠ ∞_h
shows *ratio* ∞_h *zb zc* = 1_h
using *assms*
using *inf-sub-left*[*OF assms*(*1*)]

using *inf-sub-left*[*OF assms*(*2*)]
unfolding *ratio-def*
by (*simp add*: *inf-divide-inf*)

lemma
assumes $za \neq \infty_h$ **and** $zc \neq \infty_h$
shows *ratio za* ∞_h *zc* $= \infty_h$
using *assms*
unfolding *ratio-def*
using *inf-sub-right*[*OF assms*(*1*)]
using *sub-noteq-inf*[*OF assms*]
using *divide-inf-left*
by *simp*

lemma
assumes $za \neq \infty_h$ **and** $zb \neq \infty_h$
shows *ratio za zb* $\infty_h = 0_h$
unfolding *ratio-def*
using *sub-noteq-inf*[*OF assms*]
using *inf-sub-right*[*OF assms*(*1*)]
using *divide-inf-right*
by *simp*

lemma
assumes *z1* $\neq$ *z2* **and** *z1* $\neq \infty_h$
shows *ratio z1 z2 z1* $= \infty_h$
using *assms*
unfolding *ratio-def*
using *divide-zero-right*[*of z1* $-_h$ *z2*]
using *sub-eq-zero-iff*[*of z1 z2*]
by *simp*

5.5.2 Cross-ratio

The cross-ratio is defined over 4 points (z, u, v, w), usually as $\frac{(z-u)(v-w)}{(z-w)(v-u)}$. We define it using homogeneous coordinates. Cross ratio is ill-defined when $z = u \vee v = w$ and $z = w$ and $v = u$ i.e. when 3 points are equal. Since function must be total, in that case we define it arbitrarily to 1.

definition *cross-ratio-cvec* :: *complex-vec* ⇒ *complex-vec* ⇒ *complex-vec* ⇒ *complex-vec* ⇒ *complex-vec* **where**
[*simp*]: *cross-ratio-cvec z u v w* =
(*let* (*z'*, *z''*) = *z*;
(*u'*, *u''*) = *u*;
(*v'*, *v''*) = *v*;
(*w'*, *w''*) = *w*;
n1 = *z'*∗*u''* − *u'*∗*z''*;
n2 = *v'*∗*w''* − *w'*∗*v''*;
d1 = *z'*∗*w''* − *w'*∗*z''*;
d2 = *v'*∗*u''* − *u'*∗*v''*
in
if n1 ∗ *n2* ≠ *0* ∨ *d1* ∗ *d2* ≠ *0 then*
(*n1* ∗ *n2*, *d1* ∗ *d2*)
else
(*1*, *1*))

lift-definition *cross-ratio-hcoords* :: *complex-homo-coords* ⇒ *complex-homo-coords* ⇒ *complex-homo-coords* ⇒ *complex-homo-coords* ⇒ *complex-homo-coords* **is** *cross-ratio-cvec*
by (*auto split*: *if-split-asm*)

lift-definition *cross-ratio* :: *complex-homo* ⇒ *complex-homo* ⇒ *complex-homo* ⇒ *complex-homo* ⇒ *complex-homo* **is** *cross-ratio-hcoords*
proof *transfer*
fix *z u v w z' u' v' w'* :: *complex-vec*
obtain *z1 z2 u1 u2 v1 v2 w1 w2 z'1 z'2 u'1 u'2 v'1 v'2 w'1 w'2*
where ∗: *z* = (*z1*, *z2*) *u* = (*u1*, *u2*) *v* = (*v1*, *v2*) *w* = (*w1*, *w2*)
z' = (*z'1*, *z'2*) *u'* = (*u'1*, *u'2*) *v'* = (*v'1*, *v'2*) *w'* = (*w'1*, *w'2*)
by (*cases z*, *auto*, *cases u*, *auto*, *cases v*, *auto*, *cases w*, *auto*,

```
      cases z', auto, cases u', auto, cases v', auto, cases w', auto)
  let ?n1 = z1*u2 - u1*z2
  let ?n2 = v1*w2 - w1*v2
  let ?d1 = z1*w2 - w1*z2
  let ?d2 = v1*u2 - u1*v2
  let ?n1' = z'1*u'2 - u'1*z'2
  let ?n2' = v'1*w'2 - w'1*v'2
  let ?d1' = z'1*w'2 - w'1*z'2
  let ?d2' = v'1*u'2 - u'1*v'2

  assume **:
      z ≠ vec-zero u ≠ vec-zero v ≠ vec-zero w ≠ vec-zero
      z' ≠ vec-zero u' ≠ vec-zero v' ≠ vec-zero w' ≠ vec-zero
      z ≈v z' v ≈v v' u ≈v u' w ≈v w'
  show cross-ratio-cvec z u v w ≈v cross-ratio-cvec z' u' v' w'
  proof (cases ?n1*?n2 ≠ 0 ∨ ?d1*?d2 ≠ 0)
    case True
    hence ?n1'*?n2' ≠ 0 ∨ ?d1'*?d2' ≠ 0
      using * **
      by simp ((erule exE)+, simp)
    show ?thesis
      using ‹?n1*?n2 ≠ 0 ∨ ?d1*?d2 ≠ 0›
      using ‹?n1'*?n2' ≠ 0 ∨ ?d1'*?d2' ≠ 0›
      using * **
      by simp ((erule exE)+, rule-tac x=k*ka*kb*kc in exI, simp add: field-simps)
  next
    case False
    hence ¬ (?n1'*?n2' ≠ 0 ∨ ?d1'*?d2' ≠ 0)
      using * **
      by simp ((erule exE)+, simp)
    show ?thesis
      using ‹¬ (?n1*?n2 ≠ 0 ∨ ?d1*?d2 ≠ 0)›
      using ‹¬ (?n1'*?n2' ≠ 0 ∨ ?d1'*?d2' ≠ 0)›
      using * **
      by simp blast
  qed
qed

lemma cross-ratio-01inf-id [simp]:
  shows cross-ratio z 0h 1h ∞h = z
proof (transfer, transfer)
  fix z :: complex-vec
  obtain z1 z2 where *: z = (z1, z2)
    by (cases z, auto)
  assume z ≠ vec-zero
  thus cross-ratio-cvec z 0v 1v ∞v ≈v z
    using *
    by simp (rule-tac x=−1 in exI, simp)
qed

lemma cross-ratio-0:
  assumes u ≠ v and u ≠ w
  shows cross-ratio u u v w = 0h
  using assms
proof (transfer, transfer)
  fix u v w :: complex-vec
  obtain u1 u2 v1 v2 w1 w2
    where *: u = (u1, u2) v = (v1, v2) w = (w1, w2)
    by (cases u, auto, cases v, auto, cases w, auto)
  assume u ≠ vec-zero v ≠ vec-zero w ≠ vec-zero ¬ u ≈v v ¬ u ≈v w
  thus cross-ratio-cvec u u v w ≈v 0v
    using * complex-cvec-eq-mix[of u1 u2 v1 v2] complex-cvec-eq-mix[of u1 u2 w1 w2]
    by (force simp add: mult.commute)
qed

lemma cross-ratio-1:
```

assumes $u \neq v$ **and** $v \neq w$
shows *cross-ratio v u v w* $= 1_h$
using *assms*
proof (*transfer*, *transfer*)
fix *u v w* :: *complex-vec*
obtain *u1 u2 v1 v2 w1 w2*
where $*$: $u = (u1,\ u2)$ $v = (v1,\ v2)$ $w = (w1,\ w2)$
by (*cases u*, *auto*, *cases v*, *auto*, *cases w*, *auto*)
let *?n1* $= v1*u2 - u1*v2$
let *?n2* $= v1*w2 - w1*v2$
assume $u \neq$ *vec-zero* $v \neq$ *vec-zero* $w \neq$ *vec-zero* $\neg\ u \approx_v v \neg\ v \approx_v w$
hence *?n1* $\neq 0 \wedge$ *?n2* $\neq 0$
using $*$ *complex-cvec-eq-mix*[*of u1 u2 v1 v2*] *complex-cvec-eq-mix*[*of v1 v2 w1 w2*]
by (*auto simp add*: *field-simps*)
thus *cross-ratio-cvec v u v w* $\approx_v 1_v$
using $*$
by *simp* (*rule-tac x=1 / (?n1 * ?n2)* **in** *exI*, *simp*)
qed

lemma *cross-ratio-inf*:
assumes $u \neq w$ **and** $v \neq w$
shows *cross-ratio w u v w* $= \infty_h$
using *assms*
proof (*transfer*, *transfer*)
fix *u v w* :: *complex-vec*
obtain *u1 u2 v1 v2 w1 w2*
where $*$: $u = (u1,\ u2)$ $v = (v1,\ v2)$ $w = (w1,\ w2)$
by (*cases u*, *auto*, *cases v*, *auto*, *cases w*, *auto*)
let *?n1* $= w1*u2 - u1*w2$
let *?n2* $= v1*w2 - w1*v2$
assume $u \neq$ *vec-zero* $v \neq$ *vec-zero* $w \neq$ *vec-zero* $\neg\ u \approx_v w \neg\ v \approx_v w$
hence *?n1* $\neq 0 \wedge$ *?n2* $\neq 0$
using $*$ *complex-cvec-eq-mix*[*of u1 u2 w1 w2*] *complex-cvec-eq-mix*[*of v1 v2 w1 w2*]
by (*auto simp add*: *field-simps*)
thus *cross-ratio-cvec w u v w* $\approx_v \infty_v$
using $*$
by *simp*
qed

lemma *cross-ratio-0inf*:
assumes $y \neq 0$
shows *cross-ratio* (*of-complex x*) 0_h (*of-complex y*) $\infty_h =$ (*of-complex* $(x\ /\ y)$)
using *assms*
by (*transfer*, *transfer*) (*simp*, *rule-tac x=*$-1/y$ **in** *exI*, *simp*)

lemma *cross-ratio-commute-13*:
shows *cross-ratio z u v w* $=$ *reciprocal* (*cross-ratio v u z w*)
by (*transfer*, *transfer*, *case-tac z*, *case-tac u*, *case-tac v*, *case-tac w*, *simp*)

lemma *cross-ratio-commute-24*:
shows *cross-ratio z u v w* $=$ *reciprocal* (*cross-ratio z w v u*)
by (*transfer*, *transfer*, *case-tac z*, *case-tac u*, *case-tac v*, *case-tac w*, *simp*)

lemma *cross-ratio-not-inf*:
assumes $z \neq w$ **and** $u \neq v$
shows *cross-ratio z u v w* $\neq \infty_h$
using *assms*
proof (*transfer*, *transfer*)
fix *z u v w*
assume *nz*: $z \neq$ *vec-zero* $u \neq$ *vec-zero* $v \neq$ *vec-zero* $w \neq$ *vec-zero*
obtain *z1 z2 u1 u2 v1 v2 w1 w2* **where** $*$: $z = (z1,\ z2)$ $u = (u1,\ u2)$ $v = (v1,\ v2)$ $w = (w1,\ w2)$
by (*cases z*, *cases u*, *cases v*, *cases w*, *auto*)
obtain *x1 x2* **where** $**$: *cross-ratio-cvec z u v w* $= (x1,\ x2)$
by (*cases cross-ratio-cvec z u v w*, *auto*)
assume $\neg\ z \approx_v w \neg\ u \approx_v v$
hence *z1*$*$*w2* $\neq$ *z2*$*$*w1 u1*$*$*v2* $\neq$ *u2*$*$*v1*

```
    using * nz complex-cvec-eq-mix
    by blast+
  hence x2 ≠ 0
    using * **
    by (auto split: if-split-asm) (simp add: field-simps)
  thus ¬ cross-ratio-cvec z u v w ≈v ∞v
    using inf-cvec-z2-zero-iff * **
    by simp
qed

lemma cross-ratio-not-zero:
  assumes z ≠ u and v ≠ w
  shows cross-ratio z u v w ≠ 0h
  using assms
proof (transfer, transfer)
  fix z u v w
  assume nz: z ≠ vec-zero u ≠ vec-zero v ≠ vec-zero w ≠ vec-zero
  obtain z1 z2 u1 u2 v1 v2 w1 w2 where *: z = (z1, z2) u = (u1, u2) v = (v1, v2) w = (w1, w2)
    by (cases z, cases u, cases v, cases w, auto)
  obtain x1 x2 where **: cross-ratio-cvec z u v w = (x1, x2)
    by (cases cross-ratio-cvec z u v w, auto)
  assume ¬ z ≈v u ¬ v ≈v w
  hence z1*u2 ≠ z2*u1 v1*w2 ≠ v2*w1
    using * nz complex-cvec-eq-mix
    by blast+
  hence x1 ≠ 0
    using * **
    by (auto split: if-split-asm)
  thus ¬ cross-ratio-cvec z u v w ≈v 0v
    using zero-cvec-z1-zero-iff * **
    by simp
qed

lemma cross-ratio-real:
  assumes is-real z and is-real u and is-real v and is-real w
  assumes z ≠ u ∧ v ≠ w ∨ z ≠ w ∧ u ≠ v
  shows is-real (to-complex (cross-ratio (of-complex z) (of-complex u) (of-complex v) (of-complex w)))
  using assms
  by (transfer, transfer, auto)

lemma cross-ratio:
  assumes (z ≠ u ∧ v ≠ w) ∨ (z ≠ w ∧ u ≠ v) and
        z ≠ ∞h and  u ≠ ∞h and v ≠ ∞h and w ≠ ∞h
  shows cross-ratio z u v w = ((z −h u) *h (v −h w)) :h ((z −h w) *h (v −h u))
  unfolding sub-def divide-def
  using assms
  apply transfer
  apply simp
  apply transfer
proof−
  fix z u v w :: complex-vec
  obtain z1 z2 u1 u2 v1 v2 w1 w2
    where *: z = (z1, z2) u = (u1, u2) v = (v1, v2) w = (w1, w2)
    by (cases z, auto, cases u, auto, cases v, auto, cases w, auto)

  let ?n1 = z1*u2 − u1*z2
  let ?n2 = v1*w2 − w1*v2
  let ?d1 = z1*w2 − w1*z2
  let ?d2 = v1*u2 − u1*v2
  assume **: z ≠ vec-zero u ≠ vec-zero v ≠ vec-zero w ≠ vec-zero
        ¬ z ≈v u ∧ ¬ v ≈v w ∨ ¬ z ≈v w ∧ ¬ u ≈v v
        ¬ z ≈v ∞v ¬ u ≈v ∞v ¬ v ≈v ∞v ¬ w ≈v ∞v

  hence ***: ?n1 * ?n2 ≠ 0 ∨ ?d1 * ?d2 ≠ 0
    using *
    using complex-cvec-eq-mix[of z1 z2 u1 u2] complex-cvec-eq-mix[of v1 v2 w1 w2]
```

using *complex-cvec-eq-mix*[*of z1 z2 w1 w2*] *complex-cvec-eq-mix*[*of u1 u2 v1 v2*]
by (*metis eq-iff-diff-eq-0 mult.commute mult-eq-0-iff*)

have ****: *z2* ≠ *0 w2* ≠ *0 u2* ≠ *0 v2* ≠ *0*
using * **(1−4) **(6−9)
using *inf-cvec-z2-zero-iff*[*of z1 z2*]
using *inf-cvec-z2-zero-iff*[*of u1 u2*]
using *inf-cvec-z2-zero-iff*[*of v1 v2*]
using *inf-cvec-z2-zero-iff*[*of w1 w2*]
by *blast+*

have *cross-ratio-cvec z u v w* = (*?n1*?n2*, *?d1*?d2*)
using * ***
by *simp*
moreover
let *?k* = *z2*u2*v2*w2*
have (*z* +$_v$ ~$_v$ *u*) *$_v$ (*v* +$_v$ ~$_v$ *w*) *$_v$ *reciprocal-cvec* ((*z* +$_v$ ~$_v$ *w*) *$_v$ (*v* +$_v$ ~$_v$ *u*)) = (*?k* * *?n1* * *?n2*, *?k* * *?d1* * *?d2*)
using * *** ****
by *auto*
ultimately
show *cross-ratio-cvec z u v w* ≈$_v$
(*z* +$_v$ ~$_v$ *u*) *$_v$ (*v* +$_v$ ~$_v$ *w*) *$_v$ *reciprocal-cvec* ((*z* +$_v$ ~$_v$ *w*) *$_v$ (*v* +$_v$ ~$_v$ *u*))
using ****
unfolding *complex-cvec-eq-def*
by (*rule-tac x*=*?k* **in** *exI*) *simp*
qed

end

6 Möbius transformations

Möbius transformations (also called homographic, linear fractional, or bilinear transformations) are the fundamental transformations of the extended complex plane. Here they are introduced algebraically. Each transformation is represented by a regular (non-singular, non-degenerate) 2×2 matrix that acts linearly on homogeneous coordinates. As proportional homogeneous coordinates represent same points of $\overline{\mathbb{C}}$, proportional matrices will represent the same Möbius transformation.

theory *Moebius*
imports *Homogeneous-Coordinates*
begin

6.1 Definition of Möbius transformations

typedef *moebius-mat* = {*M*::*complex-mat*. *mat-det M* ≠ *0*}
by (*rule-tac x*=*eye* **in** *exI*, *simp*)

setup-lifting *type-definition-moebius-mat*

definition *moebius-cmat-eq* :: *complex-mat* ⇒ *complex-mat* ⇒ *bool* **where**
[*simp*]: *moebius-cmat-eq A B* ⟷ (∃ *k*::*complex*. *k* ≠ *0* ∧ *B* = *k* *$_{sm}$ *A*)

lift-definition *moebius-mat-eq* :: *moebius-mat* ⇒ *moebius-mat* ⇒ *bool* **is** *moebius-cmat-eq*
done

lemma *moebius-mat-eq-refl* [*simp*]:
shows *moebius-mat-eq x x*
by *transfer simp*

quotient-type *moebius* = *moebius-mat* / *moebius-mat-eq*
proof (*rule equivpI*)
show *reflp moebius-mat-eq*
unfolding *reflp-def*

by *transfer auto*
next
show *symp moebius-mat-eq*
unfolding *symp-def*
by *transfer* (*auto simp add*: *symp-def*, *rule-tac x=1/k* **in** *exI*, *simp*)
next
show *transp moebius-mat-eq*
unfolding *transp-def*
by *transfer* (*auto simp add*: *transp-def*, *rule-tac x=ka∗k* **in** *exI*, *simp*)
qed

definition *mk-moebius-cmat* :: *complex* ⇒ *complex* ⇒ *complex* ⇒ *complex* ⇒ *complex-mat* **where**
[*simp*]: *mk-moebius-cmat a b c d* =
(*let M* = (*a*, *b*, *c*, *d*)
in if mat-det M ≠ *0 then*
M
else
eye)

lift-definition *mk-moebius-mat* :: *complex* ⇒ *complex* ⇒ *complex* ⇒ *complex* ⇒ *moebius-mat* **is** *mk-moebius-cmat*
by *simp*

lift-definition *mk-moebius* :: *complex* ⇒ *complex* ⇒ *complex* ⇒ *complex* ⇒ *moebius* **is** *mk-moebius-mat*
done

lemma *ex-mk-moebius*:
shows ∃ *a b c d*. *M* = *mk-moebius a b c d* ∧ *mat-det* (*a*, *b*, *c*, *d*) ≠ *0*
proof (*transfer*, *transfer*)
fix *M* :: *complex-mat*
assume *mat-det M* ≠ *0*
obtain *a b c d* **where** *M* = (*a*, *b*, *c*, *d*)
by (*cases M*, *auto*)
hence *moebius-cmat-eq M* (*mk-moebius-cmat a b c d*) ∧ *mat-det* (*a*, *b*, *c*, *d*) ≠ *0*
using ‹*mat-det M* ≠ *0*›
by *auto* (*rule-tac x=1* **in** *exI*, *simp*)
thus ∃*a b c d*. *moebius-cmat-eq M* (*mk-moebius-cmat a b c d*) ∧ *mat-det* (*a*, *b*, *c*, *d*) ≠ *0*
by *blast*
qed

6.2 Action on points

Möbius transformations are given as the action of Möbius group on the points of the extended complex plane (in homogeneous coordinates).

definition *moebius-pt-cmat-cvec* :: *complex-mat* ⇒ *complex-vec* ⇒ *complex-vec* **where**
[*simp*]: *moebius-pt-cmat-cvec M z* = *M* $*_{mv}$ *z*

lift-definition *moebius-pt-mmat-hcoords* :: *moebius-mat* ⇒ *complex-homo-coords* ⇒ *complex-homo-coords* **is** *moebius-pt-cmat-cvec*
by *auto algebra+*

lift-definition *moebius-pt* :: *moebius* ⇒ *complex-homo* ⇒ *complex-homo* **is** *moebius-pt-mmat-hcoords*
proof *transfer*
fix *M M′ x x′*
assume *moebius-cmat-eq M M′ x* ≈$_v$ *x′*
thus *moebius-pt-cmat-cvec M x* ≈$_v$ *moebius-pt-cmat-cvec M′ x′*
by (*cases M*, *cases x*, *auto simp add*: *field-simps*) (*rule-tac x=k∗ka* **in** *exI*, *simp*)
qed

lemma *bij-moebius-pt* [*simp*]:
shows *bij* (*moebius-pt M*)
unfolding *bij-def inj-on-def surj-def*
proof *safe*
fix *x y*
assume *moebius-pt M x* = *moebius-pt M y*
thus *x* = *y*
proof (*transfer*, *transfer*)

```
    fix M x y
    assume mat-det M ≠ 0 moebius-pt-cmat-cvec M x ≈v moebius-pt-cmat-cvec M y
    thus x ≈v y
      using mult-sv-mv[of - M x] mult-mv-inv[of - M]
      unfolding moebius-pt-cmat-cvec-def
      by (metis complex-cvec-eq-def)
  qed
next
  fix y
  show ∃ x. y = moebius-pt M x
  proof (transfer, transfer)
    fix y :: complex-vec and M :: complex-mat
    assume *: y ≠ vec-zero mat-det M ≠ 0
    let ?iM = mat-inv M
    let ?x = ?iM *mv y
    have ?x ≠ vec-zero
      using *
      by (metis mat-det-mult mat-eye-r mat-inv-r mult-cancel-right1 mult-mv-nonzero)
    moreover
    have y ≈v moebius-pt-cmat-cvec M ?x
      by (simp del: eye-def add: mat-inv-r[OF ‹mat-det M ≠ 0›])
    ultimately
    show ∃ x∈{v. v ≠ vec-zero}. y ≈v moebius-pt-cmat-cvec M x
      by (rule-tac x=?x in bexI, simp-all)
  qed
qed

lemma moebius-pt-eq-I:
  assumes moebius-pt M z1 = moebius-pt M z2
  shows z1 = z2
  using assms
  using bij-moebius-pt[of M]
  unfolding bij-def inj-on-def
  by blast

lemma moebius-pt-neq-I [simp]:
  assumes z1 ≠ z2
  shows moebius-pt M z1 ≠ moebius-pt M z2
  using assms
  by (auto simp add: moebius-pt-eq-I)

definition is-moebius :: (complex-homo ⇒ complex-homo) ⇒ bool where
  is-moebius f ⟷ (∃ M. f = moebius-pt M)
```

In the classic literature Möbius transformations are often expressed in the form $\frac{az+b}{cz+d}$. The following lemma shows that when restricted to finite points, the action of Möbius transformations is bilinear.

```
lemma moebius-pt-bilinear:
  assumes mat-det (a, b, c, d) ≠ 0
  shows moebius-pt (mk-moebius a b c d) z =
          (if z ≠ ∞h then
              ((of-complex a) *h z +h (of-complex b)) :h
              ((of-complex c) *h z +h (of-complex d))
           else
              (of-complex a) :h
              (of-complex c))
  unfolding divide-def
  using assms
proof (transfer, transfer)
  fix a b c d :: complex and z :: complex-vec
  obtain z1 z2 where zz: z = (z1, z2)
    by (cases z, auto)
  assume *: mat-det (a, b, c, d) ≠ 0 z ≠ vec-zero
  let ?oc = of-complex-cvec
  show moebius-pt-cmat-cvec (mk-moebius-cmat a b c d) z ≈v
      (if ¬ z ≈v ∞v
       then (?oc a *v z +v ?oc b) *v
```

```
          reciprocal-cvec (?oc c *v z +v ?oc d)
      else ?oc a *v
          reciprocal-cvec (?oc c))
  proof (cases z ≈v ∞v)
    case True
    thus ?thesis
      using zz *
      by auto
  next
    case False
    hence z2 ≠ 0
      using zz inf-cvec-z2-zero-iff ‹z ≠ vec-zero›
      by auto
    thus ?thesis
      using zz * False
      using regular-homogenous-system[of a b c d z1 z2]
      by auto
  qed
qed
```

6.3 Möbius group

Möbius elements form a group under composition. This group is called the *projective general linear group* and denoted by $PGL(2, \mathbb{C})$ (the group $SGL(2, \mathbb{C})$ containing elements with the determinant 1 can also be considered).

Identity Möbius transformation is represented by the identity matrix.

```
definition id-moebius-cmat :: complex-mat where
  [simp]: id-moebius-cmat = eye

lift-definition id-moebius-mmat :: moebius-mat is id-moebius-cmat
  by simp

lift-definition id-moebius :: moebius is id-moebius-mmat
  done

lemma moebius-pt-moebius-id [simp]:
  shows moebius-pt id-moebius = id
  unfolding id-def
  apply (rule ext, transfer, transfer)
  using eye-mv-l
  by simp

lemma mk-moeibus-id [simp]:
  shows mk-moebius a 0 0 a = id-moebius
  by (transfer, transfer, simp)
```

The inverse Möbius transformation is obtained by taking the inverse representative matrix.

```
definition moebius-inv-cmat :: complex-mat ⇒ complex-mat where
  [simp]: moebius-inv-cmat M = mat-inv M

lift-definition moebius-inv-mmat :: moebius-mat ⇒ moebius-mat is moebius-inv-cmat
  by (simp add: mat-det-inv)

lift-definition moebius-inv :: moebius ⇒ moebius is moebius-inv-mmat
proof (transfer)
  fix x y
  assume moebius-cmat-eq x y
  thus moebius-cmat-eq (moebius-inv-cmat x) (moebius-inv-cmat y)
    by (auto simp add: mat-inv-mult-sm) (rule-tac x=1/k in exI, simp)
qed

lemma moebius-inv:
  shows moebius-pt (moebius-inv M) = inv (moebius-pt M)
proof (rule inv-equality[symmetric])
```

fix x
show *moebius-pt* (*moebius-inv M*) (*moebius-pt M x*) = x
proof (*transfer*, *transfer*)
fix M::*complex-mat* **and** x::*complex-vec*
assume *mat-det M* ≠ *0 x* ≠ *vec-zero*
show *moebius-pt-cmat-cvec* (*moebius-inv-cmat M*) (*moebius-pt-cmat-cvec M x*) ≈$_v$ x
using *eye-mv-l*
by (*simp add*: *mat-inv-l*[*OF* ‹*mat-det M* ≠ *0*›])
qed
next
fix y
show *moebius-pt M* (*moebius-pt* (*moebius-inv M*) y) = y
proof (*transfer*, *transfer*)
fix M::*complex-mat* **and** y::*complex-vec*
assume *mat-det M* ≠ *0 y* ≠ *vec-zero*
show *moebius-pt-cmat-cvec M* (*moebius-pt-cmat-cvec* (*moebius-inv-cmat M*) y) ≈$_v$ y
using *eye-mv-l*
by (*simp add*: *mat-inv-r*[*OF* ‹*mat-det M* ≠ *0*›])
qed
qed

lemma *is-moebius-inv* [*simp*]:
assumes *is-moebius m*
shows *is-moebius* (*inv m*)
using *assms*
using *moebius-inv*
unfolding *is-moebius-def*
by *metis*

lemma *moebius-inv-mk-moebus* [*simp*]:
assumes *mat-det* (a, b, c, d) ≠ *0*
shows *moebius-inv* (*mk-moebius a b c d*) =
mk-moebius (d/(a∗d−b∗c)) (−b/(a∗d−b∗c)) (−c/(a∗d−b∗c)) (a/(a∗d−b∗c))
using *assms*
by (*transfer*, *transfer*) (*auto*, *rule-tac x=1* **in** *exI*, *simp-all add*: *field-simps*)

Composition of Möbius elements is obtained by multiplying their representing matrices.

definition *moebius-comp-cmat* :: *complex-mat* ⇒ *complex-mat* ⇒ *complex-mat* **where**
[*simp*]: *moebius-comp-cmat M1 M2* = *M1* ∗$_{mm}$ *M2*

lift-definition *moebius-comp-mmat* :: *moebius-mat* ⇒ *moebius-mat* ⇒ *moebius-mat* **is** *moebius-comp-cmat*
by *simp*

lift-definition *moebius-comp* :: *moebius* ⇒ *moebius* ⇒ *moebius* **is** *moebius-comp-mmat*
by *transfer* (*simp*, (*erule exE*)+, *rule-tac x=k∗ka* **in** *exI*, *simp add*: *field-simps*)

lemma *moebius-comp*:
shows *moebius-pt* (*moebius-comp M1 M2*) = *moebius-pt M1* ∘ *moebius-pt M2*
unfolding *comp-def*
by (*rule ext*, *transfer*, *transfer*, *simp*)

lemma *moebius-pt-comp* [*simp*]:
shows *moebius-pt* (*moebius-comp M1 M2*) z = *moebius-pt M1* (*moebius-pt M2 z*)
by (*auto simp add*: *moebius-comp*)

lemma *is-moebius-comp* [*simp*]:
assumes *is-moebius m1* **and** *is-moebius m2*
shows *is-moebius* (*m1* ∘ *m2*)
using *assms*
unfolding *is-moebius-def*
using *moebius-comp*
by *metis*

lemma *moebius-comp-mk-moebius* [*simp*]:
assumes *mat-det* (a, b, c, d) ≠ *0* **and** *mat-det* (a', b', c', d') ≠ *0*
shows *moebius-comp* (*mk-moebius a b c d*) (*mk-moebius a′ b′ c′ d′*) =

```
        mk-moebius (a * a' + b * c') (a * b' + b * d') (c * a' + d * c') (c * b' + d * d')
  using mat-det-mult[of (a, b, c, d) (a', b', c', d')]
  using assms
  by (transfer, transfer) (auto, rule-tac x=1 in exI, simp)

instantiation moebius :: group-add
begin
definition plus-moebius :: moebius ⇒ moebius ⇒ moebius where
  [simp]: plus-moebius = moebius-comp

definition uminus-moebius :: moebius ⇒ moebius where
  [simp]: uminus-moebius = moebius-inv

definition zero-moebius :: moebius where
  [simp]: zero-moebius = id-moebius

definition minus-moebius :: moebius ⇒ moebius ⇒ moebius where
  [simp]: minus-moebius A B = A + (−B)

instance proof
  fix a b c :: moebius
  show a + b + c = a + (b + c)
    unfolding plus-moebius-def
  proof (transfer, transfer)
    fix a b c :: complex-mat
    assume mat-det a ≠ 0 mat-det b ≠ 0 mat-det c ≠ 0
   show moebius-cmat-eq (moebius-comp-cmat (moebius-comp-cmat a b) c) (moebius-comp-cmat a (moebius-comp-cmat
b c))
      by simp (rule-tac x=1 in exI, simp add: mult-mm-assoc)
  qed
next
  fix a :: moebius
  show a + 0 = a
    unfolding plus-moebius-def zero-moebius-def
  proof (transfer, transfer)
    fix A :: complex-mat
    assume mat-det A ≠ 0
    thus moebius-cmat-eq (moebius-comp-cmat A id-moebius-cmat) A
      using mat-eye-r
      by simp
  qed
next
  fix a :: moebius
  show 0 + a = a
    unfolding plus-moebius-def zero-moebius-def
  proof (transfer, transfer)
    fix A :: complex-mat
    assume mat-det A ≠ 0
    thus moebius-cmat-eq (moebius-comp-cmat id-moebius-cmat A) A
      using mat-eye-l
      by simp
  qed
next
  fix a :: moebius
  show − a + a = 0
    unfolding plus-moebius-def uminus-moebius-def zero-moebius-def
  proof (transfer, transfer)
    fix a :: complex-mat
    assume mat-det a ≠ 0
    thus moebius-cmat-eq (moebius-comp-cmat (moebius-inv-cmat a) a) id-moebius-cmat
      by (simp add: mat-inv-l)
  qed
next
  fix a b :: moebius
  show a + − b = a − b
    unfolding minus-moebius-def
```

by *simp*
qed
end

Composition with inverse

lemma *moebius-comp-inv-left* [*simp*]:
shows *moebius-comp* (*moebius-inv M*) *M* = *id-moebius*
by (*metis left-minus plus-moebius-def uminus-moebius-def zero-moebius-def*)

lemma *moebius-comp-inv-right* [*simp*]:
shows *moebius-comp M* (*moebius-inv M*) = *id-moebius*
by (*metis right-minus plus-moebius-def uminus-moebius-def zero-moebius-def*)

lemma *moebius-pt-comp-inv-left* [*simp*]:
shows *moebius-pt* (*moebius-inv M*) (*moebius-pt M z*) = *z*
by (*subst moebius-pt-comp*[*symmetric*], *simp*)

lemma *moebius-pt-comp-inv-right* [*simp*]:
shows *moebius-pt M* (*moebius-pt* (*moebius-inv M*) *z*) = *z*
by (*subst moebius-pt-comp*[*symmetric*], *simp*)

lemma *moebius-pt-comp-inv-image-left* [*simp*]:
shows *moebius-pt* (*moebius-inv M*) ‘ *moebius-pt M ‘ A* = *A*
by *force*

lemma *moebius-pt-comp-inv-image-right* [*simp*]:
shows *moebius-pt M ‘ moebius-pt* (*moebius-inv M*) ‘ *A* = *A*
by *force*

lemma *moebius-pt-invert*:
assumes *moebius-pt M z1* = *z2*
shows *moebius-pt* (*moebius-inv M*) *z2* = *z1*
using *assms*[*symmetric*]
by *simp*

lemma *moebius-pt-moebius-inv-in-set* [*simp*]:
assumes *moebius-pt M z* ∈ *A*
shows *z* ∈ *moebius-pt* (*moebius-inv M*) ‘ *A*
using *assms*
using *image-iff*
by *fastforce*

6.4 Special kinds of Möbius transformations

6.4.1 Reciprocal (1/z) as a Möbius transformation

definition *moebius-reciprocal* :: *moebius* **where**
moebius-reciprocal = *mk-moebius 0 1 1 0*

lemma *moebius-reciprocal* [*simp*]:
shows *moebius-pt moebius-reciprocal* = *reciprocal*
unfolding *moebius-reciprocal-def*
by (*rule ext*, *transfer*, *transfer*) (*force simp add*: *split-def*)

lemma *moebius-reciprocal-inv* [*simp*]:
shows *moebius-inv moebius-reciprocal* = *moebius-reciprocal*
unfolding *moebius-reciprocal-def*
by (*transfer*, *transfer*) *simp*

6.4.2 Euclidean similarities as a Möbius transform

Euclidean similarities include Euclidean isometries (translations and rotations) and dilatations.

definition *moebius-similarity* :: *complex* ⇒ *complex* ⇒ *moebius* **where**
moebius-similarity a b = *mk-moebius a b 0 1*

lemma *moebius-pt-moebius-similarity* [*simp*]:

assumes $a \neq 0$
shows *moebius-pt* (*moebius-similarity a b*) z = (*of-complex a*) $*_h$ z $+_h$ (*of-complex b*)
unfolding *moebius-similarity-def*
using *assms*
using *mult-inf-right*[*of of-complex a*]
by (*subst moebius-pt-bilinear*, *auto*)

Their action is a linear transformation of $\mathbb{C}$.

lemma *moebius-pt-moebius-similarity'*:
assumes $a \neq 0$
shows *moebius-pt* (*moebius-similarity a b*) = (λ z. (*of-complex a*) $*_h$ z $+_h$ (*of-complex b*))
using *moebius-pt-moebius-similarity*[*OF assms*, *symmetric*]
by *simp*

lemma *is-moebius-similarity'*:
assumes $a \neq 0_h$ **and** $a \neq \infty_h$ **and** $b \neq \infty_h$
shows (λ z. a $*_h$ z $+_h$ b) = *moebius-pt* (*moebius-similarity* (*to-complex a*) (*to-complex b*))
proof−
obtain *ka kb* **where** $*$: a = *of-complex ka* $ka \neq 0$ b = *of-complex kb*
using *assms*
using *inf-or-of-complex*[*of a*] *inf-or-of-complex*[*of b*]
by *auto*
thus *?thesis*
unfolding *is-moebius-def*
using *moebius-pt-moebius-similarity'*[*of ka kb*]
by *simp*
qed

lemma *is-moebius-similarity*:
assumes $a \neq 0_h$ **and** $a \neq \infty_h$ **and** $b \neq \infty_h$
shows *is-moebius* (λ z. a $*_h$ z $+_h$ b)
using *is-moebius-similarity'*[*OF assms*]
unfolding *is-moebius-def*
by *auto*

Euclidean similarities form a group.

lemma *moebius-similarity-id* [*simp*]:
shows *moebius-similarity 1 0* = *id-moebius*
unfolding *moebius-similarity-def*
by *simp*

lemma *moebius-similarity-inv* [*simp*]:
assumes $a \neq 0$
shows *moebius-inv* (*moebius-similarity a b*) = *moebius-similarity* (*1*/a) ($-b/a$)
using *assms*
unfolding *moebius-similarity-def*
by *simp*

lemma *moebius-similarity-uminus* [*simp*]:
assumes $a \neq 0$
shows $-$ *moebius-similarity a b* = *moebius-similarity* (*1*/a) ($-b/a$)
using *assms*
by *simp*

lemma *moebius-similarity-comp* [*simp*]:
assumes $a \neq 0$ **and** $c \neq 0$
shows *moebius-comp* (*moebius-similarity a b*) (*moebius-similarity c d*) = *moebius-similarity* (a∗c) (a∗d+b)
using *assms*
unfolding *moebius-similarity-def*
by *simp*

lemma *moebius-similarity-plus* [*simp*]:
assumes $a \neq 0$ **and** $c \neq 0$
shows *moebius-similarity a b* + *moebius-similarity c d* = *moebius-similarity* (a∗c) (a∗d+b)
using *assms*
by *simp*

Euclidean similarities are the only Möbius group elements such that their action leaves the ∞_h fixed.

lemma *moebius-similarity-inf* [*simp*]:
assumes $a \neq 0$
shows *moebius-pt* (*moebius-similarity a b*) $\infty_h = \infty_h$
using *assms*
unfolding *moebius-similarity-def*
by (*transfer*, *transfer*, *simp*)

lemma *moebius-similarity-only-inf-to-inf*:
assumes $a \neq 0$ *moebius-pt* (*moebius-similarity a b*) $z = \infty_h$
shows $z = \infty_h$
using *assms*
using *inf-or-of-complex*[*of z*]
by *auto*

lemma *moebius-similarity-inf-iff* [*simp*]:
assumes $a \neq 0$
shows *moebius-pt* (*moebius-similarity a b*) $z = \infty_h \longleftrightarrow z = \infty_h$
using *assms*
using *moebius-similarity-only-inf-to-inf*[*of a b z*]
by *auto*

lemma *inf-fixed-only-moebius-similarity*:
assumes *moebius-pt M* $\infty_h = \infty_h$
shows $\exists$ *a b*. $a \neq 0 \wedge M =$ *moebius-similarity a b*
using *assms*
unfolding *moebius-similarity-def*
proof (*transfer*, *transfer*)
fix *M* :: *complex-mat*
obtain *a b c d* **where** *MM*: $M = (a, b, c, d)$
by (*cases M*, *auto*)
assume *mat-det M* $\neq 0$ *moebius-pt-cmat-cvec M* $\infty_v \approx_v \infty_v$
hence $*$: $c = 0$ $a \neq 0 \wedge d \neq 0$
using *MM*
by *auto*
show $\exists a\ b.\ a \neq 0 \wedge$ *moebius-cmat-eq M* (*mk-moebius-cmat a b 0 1*)
proof (*rule-tac x=a/d* **in** *exI*, *rule-tac x=b/d* **in** *exI*)
show $a/d \neq 0 \wedge$ *moebius-cmat-eq M* (*mk-moebius-cmat* (*a / d*) (*b / d*) *0 1*)
using *MM* $*$
by *simp* (*rule-tac x=1/d* **in** *exI*, *simp*)
qed
qed

Euclidean similarities include translations, rotations, and dilatations.

6.4.3 Translation

definition *moebius-translation* **where**
moebius-translation v = *moebius-similarity 1 v*

lemma *moebius-translation-comp* [*simp*]:
shows *moebius-comp* (*moebius-translation v1*) (*moebius-translation v2*) = *moebius-translation* (*v1* + *v2*)
unfolding *moebius-translation-def*
by (*simp add*: *field-simps*)

lemma *moebius-translation-plus* [*simp*]:
shows (*moebius-translation v1*) + (*moebius-translation v2*) = *moebius-translation* (*v1* + *v2*)
by *simp*

lemma *moebius-translation-zero* [*simp*]:
shows *moebius-translation 0* = *id-moebius*
unfolding *moebius-translation-def moebius-similarity-id*
by *simp*

lemma *moebius-translation-inv* [*simp*]:
shows *moebius-inv* (*moebius-translation v1*) = *moebius-translation* (−*v1*)

using *moebius-translation-comp*[*of v1 −v1*] *moebius-translation-zero*
using *minus-unique*[*of moebius-translation v1 moebius-translation* (*−v1*)]
by *simp*

lemma *moebius-translation-uminus* [*simp*]:
shows − (*moebius-translation v1*) = *moebius-translation* (*−v1*)
by *simp*

lemma *moebius-translation-inv-translation* [*simp*]:
shows *moebius-pt* (*moebius-translation v*) (*moebius-pt* (*moebius-translation* (*−v*)) *z*) = *z*
using *moebius-translation-inv*[*symmetric*, *of v*]
by (*simp del*: *moebius-translation-inv*)

lemma *moebius-inv-translation-translation* [*simp*]:
shows *moebius-pt* (*moebius-translation* (*−v*)) (*moebius-pt* (*moebius-translation v*) *z*) = *z*
using *moebius-translation-inv*[*symmetric*, *of v*]
by (*simp del*: *moebius-translation-inv*)

lemma *moebius-pt-moebius-translation* [*simp*]:
shows *moebius-pt* (*moebius-translation v*) (*of-complex z*) = *of-complex* (*z* + *v*)
unfolding *moebius-translation-def*
by (*simp add*: *field-simps*)

lemma *moebius-pt-moebius-translation-inf* [*simp*]:
shows *moebius-pt* (*moebius-translation v*) $\infty_h = \infty_h$
unfolding *moebius-translation-def*
by *simp*

6.4.4 Rotation

definition *moebius-rotation* **where**
moebius-rotation φ = *moebius-similarity* (*cis φ*) *0*

lemma *moebius-rotation-comp* [*simp*]:
shows *moebius-comp* (*moebius-rotation φ1*) (*moebius-rotation φ2*) = *moebius-rotation* (*φ1* + *φ2*)
unfolding *moebius-rotation-def*
using *moebius-similarity-comp*[*of cis φ1 cis φ2 0 0*]
by (*simp add*: *cis-mult*)

lemma *moebius-rotation-plus* [*simp*]:
shows (*moebius-rotation φ1*) + (*moebius-rotation φ2*) = *moebius-rotation* (*φ1* + *φ2*)
by *simp*

lemma *moebius-rotation-zero* [*simp*]:
shows *moebius-rotation 0* = *id-moebius*
unfolding *moebius-rotation-def*
using *moebius-similarity-id*
by *simp*

lemma *moebius-rotation-inv* [*simp*]:
shows *moebius-inv* (*moebius-rotation φ*) = *moebius-rotation* (*− φ*)
using *moebius-rotation-comp*[*of φ −φ*] *moebius-rotation-zero*
using *minus-unique*[*of moebius-rotation φ moebius-rotation* (*−φ*)]
by *simp*

lemma *moebius-rotation-uminus* [*simp*]:
shows − (*moebius-rotation φ*) = *moebius-rotation* (*− φ*)
by *simp*

lemma *moebius-rotation-inv-rotation* [*simp*]:
shows *moebius-pt* (*moebius-rotation φ*) (*moebius-pt* (*moebius-rotation* (*−φ*)) *z*) = *z*
using *moebius-rotation-inv*[*symmetric*, *of φ*]
by (*simp del*: *moebius-rotation-inv*)

lemma *moebius-inv-rotation-rotation* [*simp*]:
shows *moebius-pt* (*moebius-rotation* (*−φ*)) (*moebius-pt* (*moebius-rotation φ*) *z*) = *z*

using *moebius-rotation-inv*[*symmetric*, *of* φ]
by (*simp del*: *moebius-rotation-inv*)

lemma *moebius-pt-moebius-rotation* [*simp*]:
shows *moebius-pt* (*moebius-rotation* φ) (*of-complex z*) = *of-complex* (*cis* φ * *z*)
unfolding *moebius-rotation-def*
by *simp*

lemma *moebius-pt-moebius-rotation-inf* [*simp*]:
shows *moebius-pt* (*moebius-rotation v*) ∞_h = ∞_h
unfolding *moebius-rotation-def*
by *simp*

lemma *moebius-pt-rotation-inf-iff* [*simp*]:
shows *moebius-pt* (*moebius-rotation v*) *x* = ∞_h ⟷ *x* = ∞_h
unfolding *moebius-rotation-def*
using *cis-neq-zero moebius-similarity-only-inf-to-inf*
by (*simp del*: *moebius-pt-moebius-similarity*)

lemma *moebius-pt-moebius-rotation-zero* [*simp*]:
shows *moebius-pt* (*moebius-rotation* φ) 0_h = 0_h
unfolding *moebius-rotation-def*
by *simp*

lemma *moebius-pt-moebius-rotation-zero-iff* [*simp*]:
shows *moebius-pt* (*moebius-rotation* φ) *x* = 0_h ⟷ *x* = 0_h
using *moebius-pt-invert*[*of moebius-rotation* φ *x* 0_h]
by *auto*

lemma *moebius-rotation-preserve-cmod* [*simp*]:
assumes *u* ≠ ∞_h
shows *cmod* (*to-complex* (*moebius-pt* (*moebius-rotation* φ) *u*)) = *cmod* (*to-complex u*)
using *assms*
using *inf-or-of-complex*[*of u*]
by (*auto simp*: *norm-mult*)

6.4.5 Dilatation

definition *moebius-dilatation* **where**
moebius-dilatation a = *moebius-similarity* (*cor a*) *0*

lemma *moebius-dilatation-comp* [*simp*]:
assumes *a1* > *0* **and** *a2* > *0*
shows *moebius-comp* (*moebius-dilatation a1*) (*moebius-dilatation a2*) = *moebius-dilatation* (*a1* * *a2*)
using *assms*
unfolding *moebius-dilatation-def*
by *simp*

lemma *moebius-dilatation-plus* [*simp*]:
assumes *a1* > *0* **and** *a2* > *0*
shows (*moebius-dilatation a1*) + (*moebius-dilatation a2*) = *moebius-dilatation* (*a1* * *a2*)
using *assms*
by *simp*

lemma *moebius-dilatation-zero* [*simp*]:
shows *moebius-dilatation 1* = *id-moebius*
unfolding *moebius-dilatation-def*
using *moebius-similarity-id*
by *simp*

lemma *moebius-dilatation-inverse* [*simp*]:
assumes *a* > *0*
shows *moebius-inv* (*moebius-dilatation a*) = *moebius-dilatation* (*1*/*a*)
using *assms*
unfolding *moebius-dilatation-def*
by *simp*

lemma *moebius-dilatation-uminus* [*simp*]:
assumes $a > 0$
shows $-$ (*moebius-dilatation a*) = *moebius-dilatation* $(1/a)$
using *assms*
by *simp*

lemma *moebius-pt-dilatation* [*simp*]:
assumes $a \neq 0$
shows *moebius-pt* (*moebius-dilatation a*) (*of-complex z*) = *of-complex* (*cor a* $*$ *z*)
using *assms*
unfolding *moebius-dilatation-def*
by *simp*

6.4.6 Rotation-dilatation

definition *moebius-rotation-dilatation* **where**
moebius-rotation-dilatation a = *moebius-similarity a 0*

lemma *moebius-rotation-dilatation*:
assumes $a \neq 0$
shows *moebius-rotation-dilatation a* = *moebius-rotation* (*Arg a*) + *moebius-dilatation* (*cmod a*)
using *assms*
unfolding *moebius-rotation-dilatation-def moebius-rotation-def moebius-dilatation-def*
by *simp*

6.4.7 Conjugate Möbius

Conjugation is not a Möbius transformation, and conjugate Möbius transformations (obtained by conjugating each matrix element) do not represent conjugation function (although they are somewhat related).

lift-definition *conjugate-moebius-mmat* :: *moebius-mat* $\Rightarrow$ *moebius-mat* **is** *mat-cnj*
by *auto*
lift-definition *conjugate-moebius* :: *moebius* $\Rightarrow$ *moebius* **is** *conjugate-moebius-mmat*
by *transfer* (*auto simp add*: *mat-cnj-def*)

lemma *conjugate-moebius*:
shows *conjugate* $\circ$ *moebius-pt M* = *moebius-pt* (*conjugate-moebius M*) $\circ$ *conjugate*
apply (*rule ext*, *simp*)
apply (*transfer*, *transfer*)
using *vec-cnj-mult-mv* **by** *auto*

6.5 Decomposition of Möbius transformations

Every Euclidean similarity can be decomposed using translations, rotations, and dilatations.

lemma *similarity-decomposition*:
assumes $a \neq 0$
shows *moebius-similarity a b* = (*moebius-translation b*) + (*moebius-rotation* (*Arg a*)) + (*moebius-dilatation* (*cmod a*))
proof$-$
have *moebius-similarity a b* = (*moebius-translation b*) + (*moebius-rotation-dilatation a*)
using *assms*
unfolding *moebius-rotation-dilatation-def moebius-translation-def moebius-similarity-def*
by *auto*
thus *?thesis*
using *moebius-rotation-dilatation* [*OF assms*]
by (*auto simp add*: *add.assoc simp del*: *plus-moebius-def*)
qed

A very important fact is that every Möbius transformation can be composed of Euclidean similarities and a reciprocation.

lemma *moebius-decomposition*:
assumes $c \neq 0$ **and** $a*d - b*c \neq 0$
shows *mk-moebius a b c d* =
moebius-translation (a/c) +
moebius-rotation-dilatation $((b*c - a*d)/(c*c))$ +
moebius-reciprocal +

moebius-translation (d/c)
using *assms*
unfolding *moebius-rotation-dilatation-def moebius-translation-def moebius-similarity-def plus-moebius-def moebius-reciprocal-def*
by (*simp add*: *field-simps*) (*transfer*, *transfer*, *auto simp add*: *field-simps*, *rule-tac x=1/c* **in** *exI*, *simp*)

lemma *moebius-decomposition-similarity*:
assumes $a \neq 0$
shows *mk-moebius a b 0 d* = *moebius-similarity* (a/d) (b/d)
using *assms*
unfolding *moebius-similarity-def*
by (*transfer*, *transfer*, *auto*, *rule-tac x=1/d* **in** *exI*, *simp*)

Decomposition is used in many proofs. Namely, to show that every Möbius transformation has some property, it suffices to show that reciprocation and all Euclidean similarities have that property, and that the property is preserved under compositions.

lemma *wlog-moebius-decomposition*:
assumes
trans: $\bigwedge$ *v*. *P* (*moebius-translation v*) **and**
rot: $\bigwedge$ α. *P* (*moebius-rotation* α) **and**
dil: $\bigwedge$ *k*. *P* (*moebius-dilatation k*) **and**
recip: *P* (*moebius-reciprocal*) **and**
comp: $\bigwedge$ *M1 M2*. $[\![P\ M1;\ P\ M2]\!] \Longrightarrow P\ (M1 + M2)$
shows *P M*
proof−
obtain *a b c d* **where** *M* = *mk-moebius a b c d mat-det* $(a, b, c, d) \neq 0$
using *ex-mk-moebius*[*of M*]
by *auto*
show *?thesis*
proof (*cases c* = *0*)
case *False*
show *?thesis*
using *moebius-decomposition*[*of c a d b*] ‹*mat-det* $(a, b, c, d) \neq 0$› ‹$c \neq 0$› ‹*M* = *mk-moebius a b c d*›
using *moebius-rotation-dilatation*[*of* $(b*c - a*d) / (c*c)$]
using *trans*[*of a/c*] *rot*[*of Arg* $((b*c - a*d) / (c*c))$] *dil*[*of cmod* $((b*c - a*d) / (c*c))$] *recip*
using *comp*
by (*simp add*: *trans*)
next
case *True*
hence *M* = *moebius-similarity* (a/d) (b/d)
using ‹*M* = *mk-moebius a b c d*› ‹*mat-det* $(a, b, c, d) \neq 0$›
using *moebius-decomposition-similarity*
by *auto*
thus *?thesis*
using ‹$c = 0$› ‹*mat-det* $(a, b, c, d) \neq 0$›
using *similarity-decomposition*[*of a/d b/d*]
using *trans*[*of b/d*] *rot*[*of Arg* (a/d)] *dil*[*of cmod* (a/d)] *comp*
by *simp*
qed
qed

6.6 Cross ratio and Möbius existence

For any fixed three points $z1$, $z2$ and $z3$, *cross-ratio z z1 z2 z3* can be seen as a function of a single variable z.

lemma *is-moebius-cross-ratio*:
assumes *z1* $\neq$ *z2* **and** *z2* $\neq$ *z3* **and** *z1* $\neq$ *z3*
shows *is-moebius* $(\lambda\ z.$ *cross-ratio z z1 z2 z3*$)$
proof−
have $\exists\ M.\ \forall\ z.$ *cross-ratio z z1 z2 z3* = *moebius-pt M z*
using *assms*
proof (*transfer*, *transfer*)
fix *z1 z2 z3*
assume *vz*: *z1* $\neq$ *vec-zero z2* $\neq$ *vec-zero z3* $\neq$ *vec-zero*
obtain *z1′ z1″* **where** *zz1*: *z1* = (*z1′*, *z1″*)
by (*cases z1*, *auto*)
obtain *z2′ z2″* **where** *zz2*: *z2* = (*z2′*, *z2″*)

by (*cases z2*, *auto*)
obtain *z3′ z3″* **where** *zz3*: *z3* = (*z3′*, *z3″*)
by (*cases z3*, *auto*)

let *?m23* = *z2′*z3″*−*z3′*z2″*
let *?m21* = *z2′*z1″*−*z1′*z2″*
let *?m13* = *z1′*z3″*−*z3′*z1″*
let *?M* = (*z1″*?m23*, −*z1′*?m23*, *z3″*?m21*, −*z3′*?m21*)
assume ¬ *z1* ≈*v* *z2* ¬ *z2* ≈*v* *z3* ¬ *z1* ≈*v* *z3*
hence ∗: *?m23* ≠ *0* *?m21* ≠ *0* *?m13* ≠ *0*
using *vz zz1 zz2 zz3*
using *complex-cvec-eq-mix*[*of z1′ z1″ z2′ z2″*]
using *complex-cvec-eq-mix*[*of z1′ z1″ z3′ z3″*]
using *complex-cvec-eq-mix*[*of z2′ z2″ z3′ z3″*]
by (*auto simp del*: *complex-cvec-eq-def simp add*: *field-simps*)

have *mat-det ?M* = *?m21*?m23*?m13*
by (*simp add*: *field-simps*)
hence *mat-det ?M* ≠ *0*
using ∗
by *simp*
moreover
have ∀ *z*∈{*v*. *v* ≠ *vec-zero*}. *cross-ratio-cvec z z1 z2 z3* ≈*v* *moebius-pt-cmat-cvec ?M z*
proof
fix *z*
assume *z* ∈ {*v*. *v* ≠ *vec-zero*}
hence *z* ≠ *vec-zero*
by *simp*
obtain *z′ z″* **where** *zz*: *z* = (*z′*, *z″*)
by (*cases z*, *auto*)

let *?m01* = *z′*z1″*−*z1′*z″*
let *?m03* = *z′*z3″*−*z3′*z″*

have *?m01* ≠ *0* ∨ *?m03* ≠ *0*
proof (*cases z″* = *0* ∨ *z1″* = *0* ∨ *z3″* = *0*)
case *True*
thus *?thesis*
using ∗ ‹*z* ≠ *vec-zero*› *zz*
by *auto*
next
case *False*
hence *1*: *z″* ≠ *0* ∧ *z1″* ≠ *0* ∧ *z3″* ≠ *0*
by *simp*
show *?thesis*
proof (*rule ccontr*)
assume ¬ *?thesis*
hence *z′* ∗ *z1″* − *z1′* ∗ *z″* = *0* *z′* ∗ *z3″* − *z3′* ∗ *z″* = *0*
by *auto*
hence *z1′*/*z1″* = *z3′*/*z3″*
using *1 zz* ‹*z* ≠ *vec-zero*›
by (*metis frac-eq-eq right-minus-eq*)
thus *False*
using ∗ *1*
using *frac-eq-eq*
by *auto*
qed
qed
note ∗ = ∗ *this*
show *cross-ratio-cvec z z1 z2 z3* ≈*v* *moebius-pt-cmat-cvec ?M z*
using ∗ *zz zz1 zz2 zz3 mult-mv-nonzero*[*of z ?M*] ‹*mat-det ?M* ≠ *0*›
by *simp* (*rule-tac x=1* **in** *exI*, *simp add*: *field-simps*)
qed
ultimately
show ∃ *M*∈{*M*. *mat-det M* ≠ *0*}.
∀ *z*∈{*v*. *v* ≠ *vec-zero*}. *cross-ratio-cvec z z1 z2 z3* ≈*v* *moebius-pt-cmat-cvec M z*

```
    by blast
  qed
  thus ?thesis
    by (auto simp add: is-moebius-def)
qed
```

Using properties of the cross-ratio, it is shown that there is a Möbius transformation mapping any three different points to 0_{hc}, 1_{hc} and ∞_{hc}, respectively.

```
lemma ex-moebius-01inf:
  assumes z1 ≠ z2 and z1 ≠ z3 and z2 ≠ z3
  shows ∃ M. ((moebius-pt M z1 = 0h) ∧ (moebius-pt M z2 = 1h) ∧ (moebius-pt M z3 = ∞h))
  using assms
  using is-moebius-cross-ratio[OF ‹z1 ≠ z2› ‹z2 ≠ z3› ‹z1 ≠ z3›]
  using cross-ratio-0[OF ‹z1 ≠ z2› ‹z1 ≠ z3›] cross-ratio-1[OF ‹z1 ≠ z2› ‹z2 ≠ z3›] cross-ratio-inf[OF ‹z1 ≠ z3›
‹z2 ≠ z3›]
  by (metis is-moebius-def)
```

There is a Möbius transformation mapping any three different points to any three different points.

```
lemma ex-moebius:
  assumes z1 ≠ z2 and z1 ≠ z3 and z2 ≠ z3
          w1 ≠ w2 and w1 ≠ w3 and w2 ≠ w3
  shows ∃ M. ((moebius-pt M z1 = w1) ∧ (moebius-pt M z2 = w2) ∧ (moebius-pt M z3 = w3))
proof−
  obtain M1 where ∗: moebius-pt M1 z1 = 0h ∧ moebius-pt M1 z2 = 1h ∧ moebius-pt M1 z3 = ∞h
    using ex-moebius-01inf[OF assms(1−3)]
    by auto
  obtain M2 where ∗∗: moebius-pt M2 w1 = 0h ∧ moebius-pt M2 w2 = 1h ∧ moebius-pt M2 w3 = ∞h
    using ex-moebius-01inf[OF assms(4−6)]
    by auto
  let ?M = moebius-comp (moebius-inv M2) M1
  show ?thesis
    using ∗ ∗∗
    by (rule-tac x=?M in exI, auto simp add: moebius-pt-invert)
qed

lemma ex-moebius-1:
  shows ∃ M. moebius-pt M z1 = w1
proof−
  obtain z2 z3 where z1 ≠ z2 z1 ≠ z3 z2 ≠ z3
    using ex-3-different-points[of z1]
    by auto
  moreover
  obtain w2 w3 where w1 ≠ w2 w1 ≠ w3 w2 ≠ w3
    using ex-3-different-points[of w1]
    by auto
  ultimately
  show ?thesis
    using ex-moebius[of z1 z2 z3 w1 w2 w3]
    by auto
qed
```

The next lemma turns out to have very important applications in further proof development, as it enables so called „without-loss-of-generality (wlog)" reasoning [5]. Namely, if the property is preserved under Möbius transformations, then instead of three arbitrary different points one can consider only the case of points 0_{hc}, 1_{hc}, and ∞_{hc}.

```
lemma wlog-moebius-01inf:
  fixes M::moebius
  assumes P 0h 1h ∞h and z1 ≠ z2 and z2 ≠ z3 and z1 ≠ z3
   ⋀ M a b c. P a b c ⟹ P (moebius-pt M a) (moebius-pt M b) (moebius-pt M c)
  shows P z1 z2 z3
proof−
  from assms obtain M where ∗:
    moebius-pt M z1 = 0h  moebius-pt M z2 = 1h  moebius-pt M z3 = ∞h
    using ex-moebius-01inf[of z1 z2 z3]
    by auto
```

have $**$: *moebius-pt (moebius-inv M) 0$_h$ = z1 moebius-pt (moebius-inv M) 1$_h$ = z2 moebius-pt (moebius-inv M) ∞$_h$ = z3*
by (*subst ∗*[*symmetric*], *simp*)+
thus *?thesis*
using *assms*
by *auto*
qed

6.7 Fixed points and Möbius transformation uniqueness

lemma *three-fixed-points-01inf*:
assumes *moebius-pt M 0$_h$ = 0$_h$* **and** *moebius-pt M 1$_h$ = 1$_h$* **and** *moebius-pt M ∞$_h$ = ∞$_h$*
shows *M = id-moebius*
using *assms*
by (*transfer*, *transfer*, *auto*)

lemma *three-fixed-points*:
assumes *z1 ≠ z2* **and** *z1 ≠ z3* **and** *z2 ≠ z3*
assumes *moebius-pt M z1 = z1* **and** *moebius-pt M z2 = z2* **and** *moebius-pt M z3 = z3*
shows *M = id-moebius*
proof−
from *assms* **obtain** *M′* **where** *∗*: *moebius-pt M′ z1 = 0$_h$ moebius-pt M′ z2 = 1$_h$ moebius-pt M′ z3 = ∞$_h$*
using *ex-moebius-01inf*[*of z1 z2 z3*]
by *auto*
have $**$: *moebius-pt (moebius-inv M′) 0$_h$ = z1 moebius-pt (moebius-inv M′) 1$_h$ = z2 moebius-pt (moebius-inv M′) ∞$_h$ = z3*
by (*subst ∗*[*symmetric*], *simp*)+

have *M′ + M + (−M′) = 0*
unfolding *zero-moebius-def*
apply (*rule three-fixed-points-01inf*)
using *∗ ∗∗ assms*
by (*simp add*: *moebius-comp*[*symmetric*])+
thus *?thesis*
by (*metis eq-neg-iff-add-eq-0 minus-add-cancel zero-moebius-def*)
qed

lemma *unique-moebius-three-points*:
assumes *z1 ≠ z2* **and** *z1 ≠ z3* **and** *z2 ≠ z3*
assumes *moebius-pt M1 z1 = w1* **and** *moebius-pt M1 z2 = w2* **and** *moebius-pt M1 z3 = w3*
moebius-pt M2 z1 = w1 **and** *moebius-pt M2 z2 = w2* **and** *moebius-pt M2 z3 = w3*
shows *M1 = M2*
proof−
let *?M = moebius-comp (moebius-inv M2) M1*
have *moebius-pt ?M z1 = z1*
using ‹*moebius-pt M1 z1 = w1*› ‹*moebius-pt M2 z1 = w1*›
by (*auto simp add*: *moebius-pt-invert*)
moreover
have *moebius-pt ?M z2 = z2*
using ‹*moebius-pt M1 z2 = w2*› ‹*moebius-pt M2 z2 = w2*›
by (*auto simp add*: *moebius-pt-invert*)
moreover
have *moebius-pt ?M z3 = z3*
using ‹*moebius-pt M1 z3 = w3*› ‹*moebius-pt M2 z3 = w3*›
by (*auto simp add*: *moebius-pt-invert*)
ultimately
have *?M = id-moebius*
using *assms three-fixed-points*
by *auto*
thus *?thesis*
by (*metis add-minus-cancel left-minus plus-moebius-def uminus-moebius-def zero-moebius-def*)
qed

There is a unique Möbius transformation mapping three different points to other three different points.

lemma *ex-unique-moebius-three-points*:
assumes *z1 ≠ z2* **and** *z1 ≠ z3* **and** *z2 ≠ z3*

```
        w1 ≠ w2 and w1 ≠ w3 and w2 ≠ w3
  shows ∃! M. ((moebius-pt M z1 = w1) ∧ (moebius-pt M z2 = w2) ∧ (moebius-pt M z3 = w3))
proof−
  obtain M where ∗: moebius-pt M z1 = w1 ∧ moebius-pt M z2 = w2 ∧ moebius-pt M z3 = w3
    using ex-moebius[OF assms]
    by auto
  show ?thesis
    unfolding Ex1-def
  proof (rule-tac x=M in exI, rule)
    show ∀ y. moebius-pt y z1 = w1 ∧ moebius-pt y z2 = w2 ∧ moebius-pt y z3 = w3 ⟶ y = M
      using ∗
      using unique-moebius-three-points[OF assms(1−3)]
      by simp
  qed (simp add: ∗)
qed

lemma ex-unique-moebius-three-points-fun:
  assumes z1 ≠ z2 and z1 ≠ z3 and z2 ≠ z3
        w1 ≠ w2 and w1 ≠ w3 and w2 ≠ w3
  shows ∃! f. is-moebius f ∧ (f z1 = w1) ∧ (f z2 = w2) ∧ (f z3 = w3)
proof−
  obtain M where moebius-pt M z1 = w1 moebius-pt M z2 = w2 moebius-pt M z3 = w3
    using ex-unique-moebius-three-points[OF assms]
    by auto
  thus ?thesis
    using ex-unique-moebius-three-points[OF assms]
    unfolding Ex1-def
    by (rule-tac x=moebius-pt M in exI) (auto simp add: is-moebius-def)
qed
```

Different Möbius transformations produce different actions.

```
lemma unique-moebius-pt:
  assumes moebius-pt M1 = moebius-pt M2
  shows M1 = M2
  using assms unique-moebius-three-points[of 0_h 1_h ∞_h]
  by auto

lemma is-cross-ratio-01inf:
  assumes z1 ≠ z2 and z1 ≠ z3 and z2 ≠ z3 and is-moebius f
  assumes f z1 = 0_h and f z2 = 1_h and f z3 = ∞_h
  shows f = (λ z. cross-ratio z z1 z2 z3)
  using assms
  using cross-ratio-0[OF ‹z1 ≠ z2› ‹z1 ≠ z3›] cross-ratio-1[OF ‹z1 ≠ z2› ‹z2 ≠ z3›] cross-ratio-inf[OF ‹z1 ≠ z3›
‹z2 ≠ z3›]
  using is-moebius-cross-ratio[OF ‹z1 ≠ z2› ‹z2 ≠ z3› ‹z1 ≠ z3›]
  using ex-unique-moebius-three-points-fun[OF ‹z1 ≠ z2› ‹z1 ≠ z3› ‹z2 ≠ z3›, of 0_h 1_h ∞_h]
  by auto
```

Möbius transformations preserve cross-ratio.

```
lemma moebius-preserve-cross-ratio [simp]:
  assumes z1 ≠ z2 and z1 ≠ z3 and z2 ≠ z3
  shows cross-ratio (moebius-pt M z) (moebius-pt M z1) (moebius-pt M z2) (moebius-pt M z3) =
        cross-ratio z z1 z2 z3
proof−
  let ?f = λ z. cross-ratio z z1 z2 z3
  let ?M = moebius-pt M
  let ?iM = inv ?M
  have (?f ∘ ?iM) (?M z1) = 0_h
    using bij-moebius-pt[of M] cross-ratio-0[OF ‹z1 ≠ z2› ‹z1 ≠ z3›]
    by (simp add: bij-def)
  moreover
  have (?f ∘ ?iM) (?M z2) = 1_h
    using bij-moebius-pt[of M]  cross-ratio-1[OF ‹z1 ≠ z2› ‹z2 ≠ z3›]
    by (simp add: bij-def)
  moreover
  have (?f ∘ ?iM) (?M z3) = ∞_h
```

```
    using bij-moebius-pt[of M] cross-ratio-inf[OF ‹z1 ≠ z3› ‹z2 ≠ z3›]
    by (simp add: bij-def)
  moreover
  have is-moebius (?f ∘ ?iM)
    by (rule is-moebius-comp, rule is-moebius-cross-ratio[OF ‹z1 ≠ z2› ‹z2 ≠ z3› ‹z1 ≠ z3›], rule is-moebius-inv, auto
simp add: is-moebius-def)
  moreover
  have ?M z1 ≠ ?M z2 ?M z1 ≠ ?M z3  ?M z2 ≠ ?M z3
    using assms
    by simp-all
  ultimately
  have ?f ∘ ?iM = (λ z. cross-ratio z (?M z1) (?M z2) (?M z3))
    using assms
    using is-cross-ratio-01inf[of ?M z1 ?M z2 ?M z3 ?f ∘ ?iM]
    by simp
  moreover
  have (?f ∘ ?iM) (?M z) = cross-ratio z z1 z2 z3
    using bij-moebius-pt[of M]
    by (simp add: bij-def)
  moreover
  have (λ z. cross-ratio z (?M z1) (?M z2) (?M z3)) (?M z) = cross-ratio (?M z) (?M z1) (?M z2) (?M z3)
    by simp
  ultimately
  show ?thesis
    by simp
qed

lemma conjugate-cross-ratio [simp]:
  assumes z1 ≠ z2 and z1 ≠ z3 and z2 ≠ z3
  shows cross-ratio (conjugate z) (conjugate z1) (conjugate z2) (conjugate z3) =
        conjugate (cross-ratio z z1 z2 z3)
proof−
  let ?f = λ z. cross-ratio z z1 z2 z3
  let ?M = conjugate
  let ?iM = conjugate
  have (conjugate ∘ ?f ∘ ?iM) (?M z1) = 0h
    using cross-ratio-0[OF ‹z1 ≠ z2› ‹z1 ≠ z3›]
    by simp
  moreover
  have (conjugate ∘ ?f ∘ ?iM) (?M z2) = 1h
    using cross-ratio-1[OF ‹z1 ≠ z2› ‹z2 ≠ z3›]
    by simp
  moreover
  have (conjugate ∘ ?f ∘ ?iM) (?M z3) = ∞h
    using cross-ratio-inf[OF ‹z1 ≠ z3› ‹z2 ≠ z3›]
    by simp
  moreover
  have is-moebius (conjugate ∘ ?f ∘ ?iM)
  proof−
    obtain M where ?f = moebius-pt M
      using is-moebius-cross-ratio[OF ‹z1 ≠ z2› ‹z2 ≠ z3› ‹z1 ≠ z3›]
      by (auto simp add: is-moebius-def)
    thus ?thesis
      using conjugate-moebius[of M]
      by (auto simp add: comp-assoc is-moebius-def)
  qed
  moreover
  have ?M z1 ≠ ?M z2 ?M z1 ≠ ?M z3  ?M z2 ≠ ?M z3
    using assms
    by (auto simp add: conjugate-inj)
  ultimately
  have conjugate ∘ ?f ∘ ?iM = (λ z. cross-ratio z (?M z1) (?M z2) (?M z3))
    using assms
    using is-cross-ratio-01inf[of ?M z1 ?M z2 ?M z3 conjugate ∘ ?f ∘ ?iM]
    by simp
  moreover
```

```
  have (conjugate ∘ ?f ∘ ?iM) (?M z) = conjugate (cross-ratio z z1 z2 z3)
    by simp
  moreover
  have (λ z. cross-ratio z (?M z1) (?M z2) (?M z3)) (?M z) = cross-ratio (?M z) (?M z1) (?M z2) (?M z3)
    by simp
  ultimately
  show ?thesis
    by simp
qed

lemma cross-ratio-reciprocal [simp]:
  assumes u ≠ v and v ≠ w and u ≠ w
  shows cross-ratio (reciprocal z) (reciprocal u) (reciprocal v) (reciprocal w) =
        cross-ratio z u v w
  using assms
  by (subst moebius-reciprocal[symmetric])+ (simp del: moebius-reciprocal)

lemma cross-ratio-inversion [simp]:
  assumes u ≠ v and v ≠ w and u ≠ w
  shows cross-ratio (inversion z) (inversion u) (inversion v) (inversion w) =
        conjugate (cross-ratio z u v w)
proof−
  have reciprocal u ≠ reciprocal v reciprocal u ≠ reciprocal w reciprocal v ≠ reciprocal w
    using assms
    by ((subst moebius-reciprocal[symmetric])+, simp del: moebius-reciprocal)+
  thus ?thesis
    using assms
    unfolding inversion-def
    by simp
qed


lemma fixed-points-0inf':
  assumes moebius-pt M 0h = 0h and moebius-pt M ∞h = ∞h
  shows ∃ k::complex-homo. (k ≠ 0h ∧ k ≠ ∞h) ∧ (∀ z. moebius-pt M z = k *h z)
using assms
proof (transfer, transfer)
  fix M :: complex-mat
  assume mat-det M ≠ 0
  obtain a b c d where MM: M = (a, b, c, d)
    by (cases M) auto
  assume moebius-pt-cmat-cvec M 0v ≈v 0v moebius-pt-cmat-cvec M ∞v ≈v ∞v
  hence ∗: b = 0 c = 0 a ≠ 0 ∧ d ≠ 0
    using MM
    by auto
  let ?z = (a, d)
  have ?z ≠ vec-zero
    using ∗
    by simp
  moreover
  have ¬ ?z ≈v 0v ∧ ¬ ?z ≈v ∞v
    using ∗
    by simp
  moreover
  have ∀ z∈{v. v ≠ vec-zero}. moebius-pt-cmat-cvec M z ≈v ?z *v z
    using MM ‹mat-det M ≠ 0› ∗
    by force
  ultimately
  show ∃ k∈{v. v ≠ vec-zero}.
                (¬ k ≈v 0v ∧ ¬ k ≈v ∞v) ∧
                (∀ z∈{v. v ≠ vec-zero}. moebius-pt-cmat-cvec M z ≈v k *v z)
    by blast
qed

lemma fixed-points-0inf:
  assumes moebius-pt M 0h = 0h and moebius-pt M ∞h = ∞h
```

shows $\exists\ k$::*complex-homo*. $(k \neq 0_h \wedge k \neq \infty_h) \wedge$ *moebius-pt* $M = (\lambda\ z.\ k *_h z)$
using *fixed-points-0inf'*[*OF assms*]
by *auto*

lemma *ex-cross-ratio*:
assumes $u \neq v$ **and** $u \neq w$ **and** $v \neq w$
shows $\exists\ z.$ *cross-ratio* $z\ u\ v\ w = c$
proof−
obtain M **where** $(\lambda\ z.$ *cross-ratio* $z\ u\ v\ w) =$ *moebius-pt* M
using *assms is-moebius-cross-ratio*[*of u v w*]
unfolding *is-moebius-def*
by *auto*
hence $*$: $\forall\ z.$ *cross-ratio* $z\ u\ v\ w =$ *moebius-pt* $M\ z$
by *metis*
let *?z* = *moebius-pt* $(-M)\ c$
have *cross-ratio ?z u v w* = c
using $*$
by *auto*
thus *?thesis*
by *auto*
qed

lemma *unique-cross-ratio*:
assumes $u \neq v$ **and** $v \neq w$ **and** $u \neq w$
assumes *cross-ratio* $z\ u\ v\ w =$ *cross-ratio* $z'\ u\ v\ w$
shows $z = z'$
proof−
obtain M **where** $(\lambda\ z.$ *cross-ratio* $z\ u\ v\ w) =$ *moebius-pt* M
using *is-moebius-cross-ratio*[*OF assms*(1−3)]
unfolding *is-moebius-def*
by *auto*
hence *moebius-pt* $M\ z =$ *moebius-pt* $M\ z'$
using *assms*(4)
by *metis*
thus *?thesis*
using *moebius-pt-eq-I*
by *metis*
qed

lemma *ex1-cross-ratio*:
assumes $u \neq v$ **and** $u \neq w$ **and** $v \neq w$
shows $\exists!\ z.$ *cross-ratio* $z\ u\ v\ w = c$
using *assms ex-cross-ratio*[*OF assms*, *of c*] *unique-cross-ratio*[*of u v w*]
by *blast*

6.8 Pole

definition *is-pole* :: *moebius* ⇒ *complex-homo* ⇒ *bool* **where**
is-pole $M\ z \longleftrightarrow$ *moebius-pt* $M\ z = \infty_h$

lemma *ex1-pole*:
shows $\exists!\ z.$ *is-pole* $M\ z$
using *bij-moebius-pt*[*of M*]
unfolding *is-pole-def bij-def inj-on-def surj-def*
unfolding *Ex1-def*
by (*metis UNIV-I*)

definition *pole* :: *moebius* ⇒ *complex-homo* **where**
pole M = (*THE z*. *is-pole* $M\ z$)

lemma *pole-mk-moebius*:
assumes *is-pole* (*mk-moebius a b c d*) z **and** $c \neq 0$ **and** $a*d - b*c \neq 0$
shows $z =$ *of-complex* $(-d/c)$
proof−
let *?t1* = *moebius-translation* $(a\ /\ c)$
let *?rd* = *moebius-rotation-dilatation* $((b * c - a * d)\ /\ (c * c))$

let *?r* = *moebius-reciprocal*
let *?t2* = *moebius-translation* (*d* / *c*)
have *moebius-pt* (*?rd* + *?r* + *?t2*) *z* = ∞_h
using *assms*
unfolding *is-pole-def*
apply (*subst* (*asm*) *moebius-decomposition*)
apply (*auto simp add*: *moebius-comp*[*symmetric*] *moebius-translation-def*)
apply (*subst moebius-similarity-only-inf-to-inf*[*of 1 a*/*c*], *auto*)
done
hence *moebius-pt* (*?r* + *?t2*) *z* = ∞_h
using ‹*a*∗*d* − *b*∗*c* ≠ *0*› ‹*c* ≠ *0*›
unfolding *moebius-rotation-dilatation-def*
by (*simp del*: *moebius-pt-moebius-similarity*)
hence *moebius-pt ?t2 z* = 0_h
by *simp*
thus *?thesis*
using *moebius-pt-invert*[*of ?t2 z* 0_h]
by *simp* ((*subst* (*asm*) *of-complex-zero*[*symmetric*])+, *simp del*: *of-complex-zero*)
qed

lemma *pole-similarity*:
assumes *is-pole* (*moebius-similarity a b*) *z* **and** *a* ≠ *0*
shows *z* = ∞_h
using *assms*
unfolding *is-pole-def*
using *moebius-similarity-only-inf-to-inf*[*of a b z*]
by *simp*

6.9 Homographies and antihomographies

Inversion is not a Möbius transformation (it is a canonical example of so called anti-Möbius transformations, or antihomographies). All antihomographies are compositions of homographies and conjugation. The fundamental theorem of projective geometry (that we shall not prove) states that all automorphisms (bijective functions that preserve the cross-ratio) of $\mathbb{C}P^1$ are either homographies or antihomographies.

definition *is-homography* :: (*complex-homo* ⇒ *complex-homo*) ⇒ *bool* **where**
is-homography f ⟷ *is-moebius f*

definition *is-antihomography* :: (*complex-homo* ⇒ *complex-homo*) ⇒ *bool* **where**
is-antihomography f ⟷ (∃ *f'*. *is-moebius f'* ∧ *f* = *f'* ∘ *conjugate*)

Conjugation is not a Möbius transformation, but is antihomograhpy.

lemma *not-moebius-conjugate*:
shows ¬ *is-moebius conjugate*
proof
assume *is-moebius conjugate*
then obtain *M* **where** ∗: *moebius-pt M* = *conjugate*
unfolding *is-moebius-def*
by *metis*
hence *moebius-pt M* 0_h = 0_h *moebius-pt M* 1_h = 1_h *moebius-pt M* ∞_h = ∞_h
by *auto*
hence *M* = *id-moebius*
using *three-fixed-points-01inf*
by *auto*
hence *conjugate* = *id*
using ∗
by *simp*
moreover
have *conjugate* ii_h ≠ ii_h
using *of-complex-inj*[*of* i −i]
by (*subst of-complex-ii*[*symmetric*])+ (*auto simp del*: *of-complex-ii*)
ultimately
show *False*
by *simp*
qed

lemma *conjugation-is-antihomography*[*simp*]:
shows *is-antihomography conjugate*
unfolding *is-antihomography-def*
by (*rule-tac x=id* **in** *exI*, *metis fun.map-id0 id-apply is-moebius-def moebius-pt-moebius-id*)

lemma *inversion-is-antihomography* [*simp*]:
shows *is-antihomography inversion*
using *moebius-reciprocal*
unfolding *inversion-sym is-antihomography-def is-moebius-def*
by *metis*

Functions cannot simultaneously be homographies and antihomographies - the disjunction is exclusive.

lemma *homography-antihomography-exclusive*:
assumes *is-antihomography f*
shows ¬ *is-homography f*
proof
assume *is-homography f*
then obtain *M* **where** *f* = *moebius-pt M*
unfolding *is-homography-def is-moebius-def*
by *auto*
then obtain *M′* **where** *moebius-pt M* = *moebius-pt M′* ∘ *conjugate*
using *assms*
unfolding *is-antihomography-def is-moebius-def*
by *auto*
hence *conjugate* = *moebius-pt* (−*M′*) ∘ *moebius-pt M*
by *auto*
hence *conjugate* = *moebius-pt* (−*M′* + *M*)
by (*simp add*: *moebius-comp*)
thus *False*
using *not-moebius-conjugate*
unfolding *is-moebius-def*
by *metis*
qed

6.10 Classification of Möbius transformations

Möbius transformations can be classified to parabolic, elliptic and loxodromic. We do not develop this part of the theory in depth.

lemma *similarity-scale-1*:
assumes *k* ≠ *0*
shows *similarity* (*k* ∗*sm I*) *M* = *similarity I M*
using *assms*
unfolding *similarity-def*
using *mat-inv-mult-sm*[*of k I*]
by *simp*

lemma *similarity-scale-2*:
shows *similarity I* (*k* ∗*sm M*) = *k* ∗*sm* (*similarity I M*)
unfolding *similarity-def*
by *auto*

lemma *mat-trace-mult-sm* [*simp*]:
shows *mat-trace* (*k* ∗*sm M*) = *k* ∗ *mat-trace M*
by (*cases M*) (*simp add*: *field-simps*)

definition *moebius-mb-cmat* :: *complex-mat* ⇒ *complex-mat* ⇒ *complex-mat* **where**
[*simp*]: *moebius-mb-cmat I M* = *similarity I M*

lift-definition *moebius-mb-mmat* :: *moebius-mat* ⇒ *moebius-mat* ⇒ *moebius-mat* **is** *moebius-mb-cmat*
by (*simp add*: *similarity-def mat-det-inv*)

lift-definition *moebius-mb* :: *moebius* ⇒ *moebius* ⇒ *moebius* **is** *moebius-mb-mmat*
proof *transfer*
fix *M M′ I I′*
assume *moebius-cmat-eq M M′ moebius-cmat-eq I I′*

```
  thus moebius-cmat-eq (moebius-mb-cmat I M) (moebius-mb-cmat I' M')
    by (auto simp add: similarity-scale-1 similarity-scale-2)
qed

definition similarity-invar-cmat :: complex-mat ⇒ complex where
  [simp]: similarity-invar-cmat M = (mat-trace M)^2 / mat-det M - 4

lift-definition similarity-invar-mmat :: moebius-mat ⇒ complex is similarity-invar-cmat
  done

lift-definition similarity-invar :: moebius ⇒ complex is similarity-invar-mmat
  by transfer (auto simp add: power2-eq-square field-simps)

lemma similarity-invar-moeibus-mb:
  shows similarity-invar (moebius-mb I M) = similarity-invar M
  by (transfer, transfer, simp)

definition similar :: moebius ⇒ moebius ⇒ bool where
  similar M1 M2 ⟷ (∃ I. moebius-mb I M1 = M2)

lemma similar-refl [simp]:
  shows similar M M
  unfolding similar-def
  by (rule-tac x=id-moebius in exI) (transfer, transfer, simp)

lemma similar-sym:
  assumes similar M1 M2
  shows similar M2 M1
proof−
  from assms obtain I where M2 = moebius-mb I M1
    unfolding similar-def
    by auto
  hence M1 = moebius-mb (moebius-inv I) M2
  proof (transfer, transfer)
    fix M2 I M1
    assume moebius-cmat-eq M2 (moebius-mb-cmat I M1) mat-det I ≠ 0
    then obtain k where k ≠ 0 similarity I M1 = k *sm M2
      by auto
    thus moebius-cmat-eq M1 (moebius-mb-cmat (moebius-inv-cmat I) M2)
      using similarity-inv[of I M1 k *sm M2, OF - ‹mat-det I ≠ 0›]
      by (auto simp add: similarity-scale-2) (rule-tac x=1/k in exI, simp)
  qed
  thus ?thesis
    unfolding similar-def
    by auto
qed

lemma similar-trans:
  assumes similar M1 M2 and similar M2 M3
  shows similar M1 M3
proof−
  obtain I1 I2 where moebius-mb I1 M1 = M2 moebius-mb I2 M2 = M3
    using assms
    by (auto simp add: similar-def)
  thus ?thesis
    unfolding similar-def
  proof (rule-tac x=moebius-comp I1 I2 in exI, transfer, transfer)
    fix I1 I2 M1 M2 M3
    assume moebius-cmat-eq (moebius-mb-cmat I1 M1) M2
           moebius-cmat-eq (moebius-mb-cmat I2 M2) M3
           mat-det I1 ≠ 0 mat-det I2 ≠ 0
    thus moebius-cmat-eq (moebius-mb-cmat (moebius-comp-cmat I1 I2) M1) M3
      by (auto simp add: similarity-scale-2) (rule-tac x=ka*k in exI, simp)
  qed
qed
```

end

7 Circlines

theory *Circlines*
imports *More-Set Moebius Hermitean-Matrices Elementary-Complex-Geometry*
begin

7.1 Definition of circlines

In our formalization we follow the approach described by Schwerdtfeger [13] and represent circlines by Hermitean, non-zero 2×2 matrices. In the original formulation, a matrix $\begin{pmatrix} A & B \\ C & D \end{pmatrix}$ corresponds to the equation $A \cdot z \cdot \overline{z} + B \cdot \overline{z} + C \cdot z + D = 0$, where $C = \overline{B}$ and A and D are real (as the matrix is Hermitean).

abbreviation *hermitean-nonzero* **where**
hermitean-nonzero $\equiv \{H.\ hermitean\ H \wedge H \neq mat\text{-}zero\}$

typedef *circline-mat* = *hermitean-nonzero*
by (*rule-tac x=eye* **in** *exI*) (*auto simp add*: *hermitean-def mat-adj-def mat-cnj-def*)

setup-lifting *type-definition-circline-mat*

definition *circline-eq-cmat* :: *complex-mat* ⇒ *complex-mat* ⇒ *bool* **where**
[*simp*]: *circline-eq-cmat A B* ⟷ (∃ *k*::*real*. $k \neq 0 \wedge B = cor\ k *_{sm} A$)

lemma *symp-circline-eq-cmat*: *symp circline-eq-cmat*
unfolding *symp-def*
proof ((*rule allI*)+, *rule impI*)
fix *x y*
assume *circline-eq-cmat x y*
then obtain *k* **where** $k \neq 0 \wedge y = cor\ k *_{sm} x$
by *auto*
hence $1 / k \neq 0 \wedge x = cor\ (1 / k) *_{sm} y$
by *auto*
thus *circline-eq-cmat y x*
unfolding *circline-eq-cmat-def*
by *blast*
qed

Hermitean non-zero matrices are equivalent only to such matrices

lemma *circline-eq-cmat-hermitean-nonzero*:
assumes *hermitean H* ∧ $H \neq mat\text{-}zero$ *circline-eq-cmat H H'*
shows *hermitean H'* ∧ $H' \neq mat\text{-}zero$
using *assms*
by (*metis circline-eq-cmat-def hermitean-mult-real nonzero-mult-real of-real-eq-0-iff*)

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

lemma *circline-eq-clmat-refl* [*simp*]: *circline-eq-clmat H H*
by *transfer* (*simp*, *rule-tac x=1* **in** *exI*, *simp*)

quotient-type *circline* = *circline-mat* / *circline-eq-clmat*
proof (*rule equivpI*)
show *reflp circline-eq-clmat*
unfolding *reflp-def*
by *transfer* (*auto*, *rule-tac x=1* **in** *exI*, *simp*)
next
show *symp circline-eq-clmat*
unfolding *symp-def*
by *transfer* (*auto*, (*rule-tac x=1/k* **in** *exI*, *simp*)+)
next

show *transp circline-eq-clmat*
unfolding *transp-def*
by *transfer* (*simp*, *safe*, (*rule-tac x=ka∗k* **in** *exI*, *simp*)+)
qed

Circline with specified matrix

An auxiliary constructor *mk-circline* returns a circline (an equivalence class) for given four complex numbers A, B, C and D (provided that they form a Hermitean, non-zero matrix).

definition *mk-circline-cmat* :: *complex* ⇒ *complex* ⇒ *complex* ⇒ *complex* ⇒ *complex-mat* **where**
[*simp*]: *mk-circline-cmat A B C D* =
(*let M* = (*A*, *B*, *C*, *D*)
in if M ∈ *hermitean-nonzero then*
M
else
eye)

lift-definition *mk-circline-clmat* :: *complex* ⇒ *complex* ⇒ *complex* ⇒ *complex* ⇒ *circline-mat* **is** *mk-circline-cmat*
by (*auto simp add*: *Let-def hermitean-def mat-adj-def mat-cnj-def*)

lift-definition *mk-circline* :: *complex* ⇒ *complex* ⇒ *complex* ⇒ *complex* ⇒ *circline* **is** *mk-circline-clmat*
done

lemma *ex-mk-circline*:
shows ∃ *A B C D*. *H* = *mk-circline A B C D* ∧ *hermitean* (*A*, *B*, *C*, *D*) ∧ (*A*, *B*, *C*, *D*) ≠ *mat-zero*
proof (*transfer*, *transfer*)
fix *H*
assume ∗: *hermitean H* ∧ *H* ≠ *mat-zero*
obtain *A B C D* **where** *H* = (*A*, *B*, *C*, *D*)
by (*cases H*, *auto*)
hence *circline-eq-cmat H* (*mk-circline-cmat A B C D*) ∧ *hermitean* (*A*, *B*, *C*, *D*) ∧ (*A*, *B*, *C*, *D*) ≠ *mat-zero*
using ∗
by *auto*
thus ∃ *A B C D*. *circline-eq-cmat H* (*mk-circline-cmat A B C D*) ∧ *hermitean* (*A*, *B*, *C*, *D*) ∧ (*A*, *B*, *C*, *D*) ≠ *mat-zero*
by *blast*
qed

7.2 Circline type

definition *circline-type-cmat* :: *complex-mat* ⇒ *real* **where**
[*simp*]: *circline-type-cmat H* = *sgn* (*Re* (*mat-det H*))

lift-definition *circline-type-clmat* :: *circline-mat* ⇒ *real* **is** *circline-type-cmat*
done

lift-definition *circline-type* :: *circline* ⇒ *real* **is** *circline-type-clmat*
by *transfer* (*simp*, *erule exE*, *simp add*: *sgn-mult*)

lemma *circline-type*: *circline-type H* = −*1* ∨ *circline-type H* = *0* ∨ *circline-type H* = *1*
by (*transfer*, *transfer*, *simp add*: *sgn-if*)

lemma *circline-type-mk-circline* [*simp*]:
assumes (*A*, *B*, *C*, *D*) ∈ *hermitean-nonzero*
shows *circline-type* (*mk-circline A B C D*) = *sgn* (*Re* (*A∗D* − *B∗C*))
using *assms*
by (*transfer*, *transfer*, *simp*)

7.3 Points on the circline

Each circline determines a corresponding set of points. Again, a description given in homogeneous coordinates is a bit better than the original description defined only for ordinary complex numbers. The point with homogeneous coordinates (z_1, z_2) will belong to the set of circline points iff $A \cdot z_1 \cdot \overline{z_1} + B \cdot \overline{z_1} \cdot z_2 + C \cdot z_1 \cdot \overline{z_2} + D \cdot z_2 \cdot \overline{z_2} = 0$. Note that this is a quadratic form determined by a vector of homogeneous coordinates and the Hermitean matrix.

definition *on-circline-cmat-cvec* :: *complex-mat* ⇒ *complex-vec* ⇒ *bool* **where**

[*simp*]: *on-circline-cmat-cvec H z* ⟷ *quad-form z H = 0*

lift-definition *on-circline-clmat-hcoords* :: *circline-mat* ⇒ *complex-homo-coords* ⇒ *bool* **is** *on-circline-cmat-cvec*
done

lift-definition *on-circline* :: *circline* ⇒ *complex-homo* ⇒ *bool* **is** *on-circline-clmat-hcoords*
by *transfer* (*simp del*: *quad-form-def*, (*erule exE*)+, *simp del*: *quad-form-def add*: *quad-form-scale-m quad-form-scale-v*)

definition *circline-set* :: *circline* ⇒ *complex-homo set* **where**
circline-set H = {*z*. *on-circline H z*}

lemma *circline-set-I* [*simp*]:
assumes *on-circline H z*
shows *z* ∈ *circline-set H*
using *assms*
unfolding *circline-set-def*
by *auto*

abbreviation *circline-equation* **where**
circline-equation A B C D z1 z2 ≡ *A*z1*cnj z1 + B*z2*cnj z1 + C*cnj z2*z1 + D*z2*cnj z2 = 0*

lemma *on-circline-cmat-cvec-circline-equation*:
on-circline-cmat-cvec (*A*, *B*, *C*, *D*) (*z1*, *z2*) ⟷ *circline-equation A B C D z1 z2*
by (*simp add*: *vec-cnj-def field-simps*)

lemma *circline-equation*:
assumes *H = mk-circline A B C D* **and** (*A*, *B*, *C*, *D*) ∈ *hermitean-nonzero*
shows *of-complex z* ∈ *circline-set H* ⟷ *circline-equation A B C D z 1*
using *assms*
unfolding *circline-set-def*
by *simp* (*transfer*, *transfer*, *simp add*: *vec-cnj-def field-simps*)

Circlines trough 0 and inf.

The circline represents a line when $A = 0$ or a circle, otherwise.

definition *circline-A0-cmat* :: *complex-mat* ⇒ *bool* **where**
[*simp*]: *circline-A0-cmat H* ⟷ (*let* (*A*, *B*, *C*, *D*) = *H in A = 0*)
lift-definition *circline-A0-clmat* :: *circline-mat* ⇒ *bool* **is** *circline-A0-cmat*
done
lift-definition *circline-A0* :: *circline* ⇒ *bool* **is** *circline-A0-clmat*
by *transfer auto*

abbreviation *is-line* **where**
is-line H ≡ *circline-A0 H*

abbreviation *is-circle* **where**
is-circle H ≡ ¬ *circline-A0 H*

definition *circline-D0-cmat* :: *complex-mat* ⇒ *bool* **where**
[*simp*]: *circline-D0-cmat H* ⟷ (*let* (*A*, *B*, *C*, *D*) = *H in D = 0*)
lift-definition *circline-D0-clmat* :: *circline-mat* ⇒ *bool* **is** *circline-D0-cmat*
done
lift-definition *circline-D0* :: *circline* ⇒ *bool* **is** *circline-D0-clmat*
by *transfer auto*

lemma *inf-on-circline*: *on-circline H* ∞_h ⟷ *circline-A0 H*
by (*transfer*, *transfer*, *auto simp add*: *vec-cnj-def*)

lemma
inf-in-circline-set: ∞_h ∈ *circline-set H* ⟷ *is-line H*
using *inf-on-circline*
unfolding *circline-set-def*
by *simp*

lemma *zero-on-circline*: *on-circline H* 0_h ⟷ *circline-D0 H*
by (*transfer*, *transfer*, *auto simp add*: *vec-cnj-def*)

lemma
zero-in-circline-set: $0_h \in$ *circline-set* $H \longleftrightarrow$ *circline-D0* H
using *zero-on-circline*
unfolding *circline-set-def*
by *simp*

7.4 Connection with circles and lines in the classic complex plane

Every Euclidean circle and Euclidean line can be represented by a circline.

lemma *classic-circline*:
assumes $H =$ *mk-circline* A B C D **and** *hermitean* $(A, B, C, D) \wedge (A, B, C, D) \neq$ *mat-zero*
shows *circline-set* $H - \{\infty_h\} =$ *of-complex* ‘ *circline* (*Re* A) B (*Re* D)
using *assms*
unfolding *circline-set-def*
proof (*safe*)
fix z
assume *hermitean* (A, B, C, D) $(A, B, C, D) \neq$ *mat-zero* $z \in$ *circline* (*Re* A) B (*Re* D)
thus *on-circline* (*mk-circline* A B C D) (*of-complex* z)
using *hermitean-elems*[*of* A B C D]
by (*transfer*, *transfer*) (*auto simp add*: *circline-def vec-cnj-def field-simps*)
next
fix z
assume *of-complex* $z = \infty_h$
thus *False*
by *simp*
next
fix z
assume *hermitean* (A, B, C, D) $(A, B, C, D) \neq$ *mat-zero on-circline* (*mk-circline* A B C D) z $z \notin$ *of-complex* ‘ *circline* (*Re* A) B (*Re* D)
moreover
have $z \neq \infty_h \longrightarrow z \in$ *of-complex* ‘ *circline* (*Re* A) B (*Re* D)
proof
assume $z \neq \infty_h$
show $z \in$ *of-complex* ‘ *circline* (*Re* A) B (*Re* D)
proof
show $z =$ *of-complex* (*to-complex* z)
using ‹$z \neq \infty_h$›
by *simp*
next
show *to-complex* $z \in$ *circline* (*Re* A) B (*Re* D)
using ‹*on-circline* (*mk-circline* A B C D) z› ‹$z \neq \infty_h$›
using ‹*hermitean* (A, B, C, D)› ‹$(A, B, C, D) \neq$ *mat-zero*›
proof (*transfer*, *transfer*)
fix A B C D **and** $z ::$ *complex-vec*
obtain *z1 z2* **where** *zz*: $z = (z1, z2)$
by (*cases* z, *auto*)
assume $*$: $z \neq$ *vec-zero* $\neg\ z \approx_v \infty_v$
on-circline-cmat-cvec (*mk-circline-cmat* A B C D) z
hermitean (A, B, C, D) $(A, B, C, D) \neq$ *mat-zero*
have *z2* $\neq 0$
using ‹$z \neq$ *vec-zero*› ‹$\neg\ z \approx_v \infty_v$›
using *inf-cvec-z2-zero-iff zz*
by *blast*
thus *to-complex-cvec* $z \in$ *circline* (*Re* A) B (*Re* D)
using $*$ *zz*
using *hermitean-elems*[*of* A B C D]
by (*simp add*: *vec-cnj-def circline-def field-simps*)
qed
qed
qed
ultimately
show $z = \infty_h$
by *simp*
qed

The matrix of the circline representing circle determined with center and radius.

definition *mk-circle-cmat* :: *complex* ⇒ *real* ⇒ *complex-mat* **where**
[*simp*]: *mk-circle-cmat a r* = (*1*, −*a*, −*cnj a*, *a*∗*cnj a* − *cor r*∗*cor r*)

lift-definition *mk-circle-clmat* :: *complex* ⇒ *real* ⇒ *circline-mat* **is** *mk-circle-cmat*
by (*simp add*: *hermitean-def mat-adj-def mat-cnj-def*)

lift-definition *mk-circle* :: *complex* ⇒ *real* ⇒ *circline* **is** *mk-circle-clmat*
done

lemma *is-circle-mk-circle*: *is-circle* (*mk-circle a r*)
by (*transfer*, *transfer*, *simp*)

lemma *circline-set-mk-circle* [*simp*]:
assumes *r* ≥ *0*
shows *circline-set* (*mk-circle a r*) = *of-complex* ‘ *circle a r*
proof−
let *?A* = *1* **and** *?B* = −*a* **and** *?C* = −*cnj a* **and** *?D* = *a*∗*cnj a* − *cor r*∗*cor r*
have ∗: (*?A*, *?B*, *?C*, *?D*) ∈ {*H*. *hermitean H* ∧ *H* ≠ *mat-zero*}
by (*simp add*: *hermitean-def mat-adj-def mat-cnj-def*)
have *mk-circle a r* = *mk-circline ?A ?B ?C ?D*
using ∗
by (*transfer*, *transfer*, *simp*)
hence *circline-set* (*mk-circle a r*) − {∞$_h$} = *of-complex* ‘ *circline ?A ?B* (*Re ?D*)
using *classic-circline*[*of mk-circle a r ?A ?B ?C ?D*] ∗
by *simp*
moreover
have *circline ?A ?B* (*Re ?D*) = *circle a r*
by (*rule circline-circle*[*of ?A Re ?D ?B circline ?A ?B* (*Re ?D*) *a r*∗*r r*], *simp-all add*: *cmod-square* ‹*r* ≥ *0*›)
moreover
have ∞$_h$ ∉ *circline-set* (*mk-circle a r*)
using *inf-in-circline-set*[*of mk-circle a r*] *is-circle-mk-circle*[*of a r*]
by *auto*
ultimately
show *?thesis*
unfolding *circle-def*
by *simp*
qed

The matrix of the circline representing line determined with two (not equal) complex points.

definition *mk-line-cmat* :: *complex* ⇒ *complex* ⇒ *complex-mat* **where**
[*simp*]: *mk-line-cmat z1 z2* =
(*if z1* ≠ *z2 then*
let B = i ∗ (*z2* − *z1*) *in* (*0*, *B*, *cnj B*, −*cnj-mix B z1*)
else
eye)

lift-definition *mk-line-clmat* :: *complex* ⇒ *complex* ⇒ *circline-mat* **is** *mk-line-cmat*
by (*auto simp add*: *Let-def hermitean-def mat-adj-def mat-cnj-def split*: *if-split-asm*)

lift-definition *mk-line* :: *complex* ⇒ *complex* ⇒ *circline* **is** *mk-line-clmat*
done

lemma *circline-set-mk-line* [*simp*]:
assumes *z1* ≠ *z2*
shows *circline-set* (*mk-line z1 z2*) − {∞$_h$} = *of-complex* ‘ *line z1 z2*
proof−
let *?A* = *0* **and** *?B* = i∗(*z2* − *z1*)
let *?C* = *cnj ?B* **and** *?D* = −*cnj-mix ?B z1*
have ∗: (*?A*, *?B*, *?C*, *?D*) ∈ {*H*. *hermitean H* ∧ *H* ≠ *mat-zero*}
using *assms*
by (*simp add*: *hermitean-def mat-adj-def mat-cnj-def*)
have *mk-line z1 z2* = *mk-circline ?A ?B ?C ?D*
using ∗ *assms*
by (*transfer*, *transfer*, *auto simp add*: *Let-def*)
hence *circline-set* (*mk-line z1 z2*) − {∞$_h$} = *of-complex* ‘ *circline ?A ?B* (*Re ?D*)

using *classic-circline*[*of mk-line z1 z2 ?A ?B ?C ?D*] ∗
by *simp*
moreover
have *circline ?A ?B* (*Re ?D*) = *line z1 z2*
using ‹*z1* ≠ *z2*›
using *circline-line′*
by *simp*
ultimately
show *?thesis*
by *simp*
qed

The set of points determined by a circline is always either an Euclidean circle or an Euclidean line.

Euclidean circle is determined by its center and radius.

type-synonym *euclidean-circle* = *complex* × *real*

definition *euclidean-circle-cmat* :: *complex-mat* ⇒ *euclidean-circle* **where**
[*simp*]: *euclidean-circle-cmat H* = (*let* (*A*, *B*, *C*, *D*) = *H in* (−*B*/*A*, *sqrt*(*Re* ((*B*∗*C* − *A*∗*D*)/(*A*∗*A*)))))

lift-definition *euclidean-circle-clmat* :: *circline-mat* ⇒ *euclidean-circle* **is** *euclidean-circle-cmat*
done

lift-definition *euclidean-circle* :: *circline* ⇒ *euclidean-circle* **is** *euclidean-circle-clmat*
proof *transfer*
fix *H1 H2*
assume *hh*: *hermitean H1* ∧ *H1* ≠ *mat-zero hermitean H2* ∧ *H2* ≠ *mat-zero*
obtain *A1 B1 C1 D1* **where** *HH1*: *H1* = (*A1*, *B1*, *C1*, *D1*)
by (*cases H1*) *auto*
obtain *A2 B2 C2 D2* **where** *HH2*: *H2* = (*A2*, *B2*, *C2*, *D2*)
by (*cases H2*) *auto*
assume *circline-eq-cmat H1 H2*
then obtain *k* **where** *k* ≠ *0* **and** ∗: *A2* = *cor k* ∗ *A1 B2* = *cor k* ∗ *B1 C2* = *cor k* ∗ *C1 D2* = *cor k* ∗ *D1*
using *HH1 HH2*
by *auto*
have (*cor k* ∗ *B1* ∗ (*cor k* ∗ *C1*) − *cor k* ∗ *A1* ∗ (*cor k* ∗ *D1*)) = (*cor k*)^2 ∗ (*B1*∗*C1* − *A1*∗*D1*)
(*cor k* ∗ *A1* ∗ (*cor k* ∗ *A1*)) = (*cor k*)^2 ∗ (*A1*∗*A1*)
by (*auto simp add*: *field-simps power2-eq-square*)
hence (*cor k* ∗ *B1* ∗ (*cor k* ∗ *C1*) − *cor k* ∗ *A1* ∗ (*cor k* ∗ *D1*)) /
(*cor k* ∗ *A1* ∗ (*cor k* ∗ *A1*)) = (*B1*∗*C1* − *A1*∗*D1*) / (*A1*∗*A1*)
using ‹*k* ≠ *0*›
by (*simp add*: *power2-eq-square*)
thus *euclidean-circle-cmat H1* = *euclidean-circle-cmat H2*
using *HH1 HH2* ∗ *hh*
by *auto*
qed

lemma *classic-circle*:
assumes *is-circle H* **and** (*a*, *r*) = *euclidean-circle H* **and** *circline-type H* ≤ *0*
shows *circline-set H* = *of-complex* ‘ *circle a r*
proof−
obtain *A B C D* **where** ∗: *H* = *mk-circline A B C D hermitean* (*A*, *B*, *C*, *D*) (*A*, *B*, *C*, *D*) ≠ *mat-zero*
using *ex-mk-circline*[*of H*]
by *auto*
have *is-real A is-real D C* = *cnj B*
using ∗ *hermitean-elems*
by *auto*
have *Re* (*A*∗*D* − *B*∗*C*) ≤ *0*
using ‹*circline-type H* ≤ *0*› ∗
by *simp*

hence ∗∗: *Re A* ∗ *Re D* ≤ (*cmod B*)^2
using ‹*is-real A*› ‹*is-real D*› ‹*C* = *cnj B*›
by (*simp add*: *cmod-square*)

have *A* ≠ *0*

using ‹*is-circle H*› ∗ ‹*is-real A*›
by *simp* (*transfer*, *transfer*, *simp*)

hence *Re A* ≠ *0*
using ‹*is-real A*›
by (*metis complex-surj zero-complex.code*)

have ∗∗∗: $\infty_h \notin$ *circline-set H*
using ∗ *inf-in-circline-set*[*of H*] ‹*is-circle H*›
by *simp*

let *?a* = −*B*/*A*
let *?r2* = ((*cmod B*)^2 − *Re A* ∗ *Re D*) / (*Re A*)^2
let *?r* = *sqrt ?r2*

have *?a* = *a* ∧ *?r* = *r*
using ‹(*a*, *r*) = *euclidean-circle H*›
using ∗ ‹*is-real A*› ‹*is-real D*› ‹*C* = *cnj B*› ‹*A* ≠ *0*›
apply *simp*
apply *transfer*
apply *transfer*
apply *simp*
apply (*subst Re-divide-real*)
apply (*simp-all add*: *cmod-square*, *simp add*: *power2-eq-square*)
done

show *?thesis*
using ∗ ∗∗ ∗∗∗ ‹*Re A* ≠ *0*› ‹*is-real A*› ‹*C* = *cnj B*› ‹*?a* = *a* ∧ *?r* = *r*›
using *classic-circline*[*of H A B C D*] *assms circline-circle*[*of Re A Re D B circline* (*Re A*) *B* (*Re D*) *?a ?r2 ?r*]
by (*simp add*: *circle-def*)
qed

Euclidean line is represented by two points.

type-synonym *euclidean-line* = *complex* × *complex*

definition *euclidean-line-cmat* :: *complex-mat* ⇒ *euclidean-line* **where**
[*simp*]: *euclidean-line-cmat H* =
(*let* (*A*, *B*, *C*, *D*) = *H*;
z1 = −(*D*∗*B*)/(*2*∗*B*∗*C*);
z2 = *z1* + i ∗ *sgn* (*if Arg B* > *0 then* −*B else B*)
in (*z1*, *z2*))

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

lift-definition *euclidean-line* :: *circline* ⇒ *complex* × *complex* **is** *euclidean-line-clmat*
proof *transfer*
fix *H1 H2*
assume *hh*: *hermitean H1* ∧ *H1* ≠ *mat-zero hermitean H2* ∧ *H2* ≠ *mat-zero*
obtain *A1 B1 C1 D1* **where** *HH1*: *H1* = (*A1*, *B1*, *C1*, *D1*)
by (*cases H1*) *auto*
obtain *A2 B2 C2 D2* **where** *HH2*: *H2* = (*A2*, *B2*, *C2*, *D2*)
by (*cases H2*) *auto*
assume *circline-eq-cmat H1 H2*
then obtain *k* **where** *k* ≠ *0* **and** ∗: *A2* = *cor k* ∗ *A1 B2* = *cor k* ∗ *B1 C2* = *cor k* ∗ *C1 D2* = *cor k* ∗ *D1*
using *HH1 HH2*
by *auto*
have *1*: *B1* ≠ *0* ∧ *0* < *Arg B1* ⟶ ¬ *0* < *Arg* (− *B1*)
using *canon-ang-plus-pi1*[*of Arg B1*] *Arg-bounded*[*of B1*]
by (*auto simp add*: *arg-uminus*)
have *2*: *B1* ≠ *0* ∧ ¬ *0* < *Arg B1* ⟶ *0* < *Arg* (− *B1*)
using *canon-ang-plus-pi2*[*of Arg B1*] *Arg-bounded*[*of B1*]
by (*auto simp add*: *arg-uminus*)

show *euclidean-line-cmat H1* = *euclidean-line-cmat H2*
using *HH1 HH2* ∗ ‹*k* ≠ *0*›

by (*cases k > 0*) (*auto simp add*: *Let-def*, *simp-all add*: *norm-mult sgn-eq 1 2*)
qed

lemma *classic-line*:
assumes *is-line H* **and** *circline-type H < 0* **and** (*z1*, *z2*) = *euclidean-line H*
shows *circline-set H − {∞ₕ} = of-complex ‘ line z1 z2*
proof−
obtain *A B C D* **where** ∗: *H = mk-circline A B C D hermitean* (*A*, *B*, *C*, *D*) (*A*, *B*, *C*, *D*) ≠ *mat-zero*
using *ex-mk-circline*[*of H*]
by *auto*
have *is-real A is-real D C = cnj B*
using ∗ *hermitean-elems*
by *auto*
have *Re A = 0*
using ‹*is-line H*› ∗ ‹*is-real A*› ‹*is-real D*› ‹*C = cnj B*›
by *simp* (*transfer*, *transfer*, *simp*)
have *B ≠ 0*
using ‹*Re A = 0*› ‹*is-real A*› ‹*is-real D*› ‹*C = cnj B*› ∗ ‹*circline-type H < 0*›
using *circline-type-mk-circline*[*of A B C D*]
by *auto*

let *?z1 = − cor* (*Re D*) ∗ *B* / (*2* ∗ *B* ∗ *cnj B*)
let *?z2 = ?z1* + i ∗ *sgn* (*if 0 < Arg B then − B else B*)
have *z1 = ?z1* ∧ *z2 = ?z2*
using ‹(*z1*, *z2*) = *euclidean-line H*› ∗ ‹*is-real A*› ‹*is-real D*› ‹*C = cnj B*›
by *simp* (*transfer*, *transfer*, *simp add*: *Let-def*)
thus *?thesis*
using ∗
using *classic-circline*[*of H A B C D*] *circline-line*[*of Re A B circline* (*Re A*) *B* (*Re D*) *Re D ?z1 ?z2*] ‹*Re A = 0*› ‹*B ≠ 0*›
by *simp*
qed

7.5 Some special circlines

7.5.1 Unit circle

definition *unit-circle-cmat* :: *complex-mat* **where**
[*simp*]: *unit-circle-cmat* = (*1*, *0*, *0*, *−1*)
lift-definition *unit-circle-clmat* :: *circline-mat* **is** *unit-circle-cmat*
by (*simp add*: *hermitean-def mat-adj-def mat-cnj-def*)
lift-definition *unit-circle* :: *circline* **is** *unit-circle-clmat*
done

lemma *on-circline-cmat-cvec-unit*:
shows *on-circline-cmat-cvec unit-circle-cmat* (*z1*, *z2*) ⟷
z1 ∗ *cnj z1* = *z2* ∗ *cnj z2*
by (*simp add*: *vec-cnj-def field-simps*)

lemma
one-on-unit-circle [*simp*]: *on-circline unit-circle 1ₕ* **and**
ii-on-unit-circle [*simp*]: *on-circline unit-circle iiₕ* **and**
not-zero-on-unit-circle [*simp*]: ¬ *on-circline unit-circle 0ₕ*
by (*transfer*, *transfer*, *simp add*: *vec-cnj-def*)+

lemma
one-in-unit-circle-set [*simp*]: *1ₕ* ∈ *circline-set unit-circle* **and**
ii-in-unit-circle-set [*simp*]: *iiₕ* ∈ *circline-set unit-circle* **and**
zero-in-unit-circle-set [*simp*]: *0ₕ* ∉ *circline-set unit-circle*
unfolding *circline-set-def*
by *simp-all*

lemma *is-circle-unit-circle* [*simp*]:
shows *is-circle unit-circle*
by (*transfer*, *transfer*, *simp*)

lemma *not-inf-on-unit-circle′* [*simp*]:

shows ¬ *on-circline unit-circle* ∞_h
using *is-circle-unit-circle inf-on-circline*
by *blast*

lemma *not-inf-on-unit-circle''* [*simp*]:
shows $\infty_h \notin$ *circline-set unit-circle*
by (*simp add*: *inf-in-circline-set*)

lemma *euclidean-circle-unit-circle* [*simp*]:
shows *euclidean-circle unit-circle* = (*0*, *1*)
by (*transfer*, *transfer*, *simp*)

lemma *circline-type-unit-circle* [*simp*]:
shows *circline-type unit-circle* = *−1*
by (*transfer*, *transfer*, *simp*)

lemma *on-circline-unit-circle* [*simp*]:
shows *on-circline unit-circle* (*of-complex z*) ⟷ *cmod z* = *1*
by (*transfer*, *transfer*, *simp add*: *vec-cnj-def mult.commute*)

lemma *circline-set-unit-circle* [*simp*]:
shows *circline-set unit-circle* = *of-complex* ‘ {*z*. *cmod z* = *1*}
proof −
show *?thesis*
proof *safe*
fix *x*
assume *x* ∈ *circline-set unit-circle*
then obtain *x′* **where** *x* = *of-complex x′*
using *inf-or-of-complex*[*of x*]
by *auto*
thus *x* ∈ *of-complex* ‘ {*z*. *cmod z* = *1*}
using ‹*x* ∈ *circline-set unit-circle*›
unfolding *circline-set-def*
by *auto*
next
fix *x*
assume *cmod x* = *1*
thus *of-complex x* ∈ *circline-set unit-circle*
unfolding *circline-set-def*
by *auto*
qed
qed

lemma *circline-set-unit-circle-I* [*simp*]:
assumes *cmod z* = *1*
shows *of-complex z* ∈ *circline-set unit-circle*
using *assms*
unfolding *circline-set-unit-circle*
by *simp*

lemma *inversion-unit-circle* [*simp*]:
assumes *on-circline unit-circle x*
shows *inversion x* = *x*
proof −
obtain *x′* **where** *x* = *of-complex x′ x′* ≠ *0*
using *inf-or-of-complex*[*of x*]
using *assms*
by *force*
moreover
hence *x′* ∗ *cnj x′* = *1*
using *assms*
using *circline-set-unit-circle*
unfolding *circline-set-def*
by *auto*
hence *1* / *cnj x′* = *x′*
using ‹*x′* ≠ *0*›

by (*simp add*: *field-simps*)
ultimately
show *?thesis*
using *assms*
unfolding *inversion-def*
by *simp*
qed

lemma *inversion-id-iff-on-unit-circle*:
shows *inversion a* = *a* ⟷ *on-circline unit-circle a*
using *inversion-id-iff*[*of a*] *inf-or-of-complex*[*of a*]
by *auto*

lemma *on-unit-circle-conjugate* [*simp*]:
shows *on-circline unit-circle* (*conjugate z*) ⟷ *on-circline unit-circle z*
by (*transfer*, *transfer*, *auto simp add*: *vec-cnj-def field-simps*)

lemma *conjugate-unit-circle-set* [*simp*]:
shows *conjugate* ' (*circline-set unit-circle*) = *circline-set unit-circle*
unfolding *circline-set-def*
by (*auto simp add*: *image-iff*, *rule-tac x*=*conjugate x* **in** *exI*, *simp*)

7.5.2 x-axis

definition *x-axis-cmat* :: *complex-mat* **where**
[*simp*]: *x-axis-cmat* = (*0*, i, −i, *0*)
lift-definition *x-axis-clmat* :: *circline-mat* **is** *x-axis-cmat*
by (*simp add*: *hermitean-def mat-adj-def mat-cnj-def*)
lift-definition *x-axis* :: *circline* **is** *x-axis-clmat*
done

lemma *special-points-on-x-axis'* [*simp*]:
shows *on-circline x-axis* 0_h **and** *on-circline x-axis* 1_h **and** *on-circline x-axis* ∞_h
by (*transfer*, *transfer*, *simp add*: *vec-cnj-def*)+

lemma *special-points-on-x-axis''* [*simp*]:
shows 0_h ∈ *circline-set x-axis* **and** 1_h ∈ *circline-set x-axis* **and** ∞_h ∈ *circline-set x-axis*
unfolding *circline-set-def*
by *auto*

lemma *is-line-x-axis* [*simp*]:
shows *is-line x-axis*
by (*transfer*, *transfer*, *simp*)

lemma *circline-type-x-axis* [*simp*]:
shows *circline-type x-axis* = −*1*
by (*transfer*, *transfer*, *simp*)

lemma *on-circline-x-axis*:
shows *on-circline x-axis z* ⟷ (∃ *c*. *is-real c* ∧ *z* = *of-complex c*) ∨ *z* = ∞_h
proof *safe*
fix *z c*
assume *is-real c*
thus *on-circline x-axis* (*of-complex c*)
proof (*transfer*, *transfer*)
fix *c*
assume *is-real c*
thus *on-circline-cmat-cvec x-axis-cmat* (*of-complex-cvec c*)
using *eq-cnj-iff-real*[*of c*]
by (*simp add*: *vec-cnj-def*)
qed
next
fix *z*
assume *on-circline x-axis z z* ≠ ∞_h
thus ∃ *c*. *is-real c* ∧ *z* = *of-complex c*
proof (*transfer*, *transfer*, *safe*)

```
    fix a b
    assume (a, b) ≠ vec-zero
      on-circline-cmat-cvec x-axis-cmat (a, b)
      ¬ (a, b) ≈v ∞v
    hence b ≠ 0 cnj a * b = cnj b * a using inf-cvec-z2-zero-iff
      by (auto simp add: vec-cnj-def)
    thus ∃ c. is-real c ∧ (a, b) ≈v of-complex-cvec c
      apply (rule-tac x=a/b in exI)
      apply (auto simp add: is-real-div field-simps)
      apply (rule-tac x=1/b in exI, simp)
      done
  qed
next
  show on-circline x-axis ∞h
    by auto
qed

lemma on-circline-x-axis-I [simp]:
  assumes is-real z
  shows on-circline x-axis (of-complex z)
  using assms
  unfolding on-circline-x-axis
  by auto

lemma circline-set-x-axis:
  shows circline-set x-axis = of-complex ' {x. is-real x} ∪ {∞h}
  using on-circline-x-axis
  unfolding circline-set-def
  by auto

lemma circline-set-x-axis-I:
  assumes is-real z
  shows of-complex z ∈ circline-set x-axis
  using assms
  unfolding circline-set-x-axis
  by auto

lemma circline-equation-x-axis:
  shows of-complex z ∈ circline-set x-axis ⟷ z = cnj z
  unfolding circline-set-x-axis
proof auto
  fix x
  assume of-complex z = of-complex x is-real x
  hence z = x
    using of-complex-inj[of z x]
    by simp
  thus z = cnj z
    using eq-cnj-iff-real[of z] ‹is-real x›
    by auto
next
  assume z = cnj z
  thus of-complex z ∈ of-complex ' {x. is-real x}
    using eq-cnj-iff-real[of z]
    by auto
qed
```

Positive and negative part of x-axis

```
definition positive-x-axis where
  positive-x-axis = {z. z ∈ circline-set x-axis ∧ z ≠ ∞h ∧ Re (to-complex z) > 0}

definition negative-x-axis where
  negative-x-axis = {z. z ∈ circline-set x-axis ∧ z ≠ ∞h ∧ Re (to-complex z) < 0}

lemma circline-set-positive-x-axis-I [simp]:
  assumes is-real z and Re z > 0
  shows of-complex z ∈ positive-x-axis
```

using *assms*
unfolding *positive-x-axis-def*
by *simp*

lemma *circline-set-negative-x-axis-I* [*simp*]:
assumes *is-real z* **and** *Re z* < *0*
shows *of-complex z* ∈ *negative-x-axis*
using *assms*
unfolding *negative-x-axis-def*
by *simp*

7.5.3 y-axis

definition *y-axis-cmat* :: *complex-mat* **where**
[*simp*]: *y-axis-cmat* = (*0*, *1*, *1*, *0*)
lift-definition *y-axis-clmat* :: *circline-mat* **is** *y-axis-cmat*
by (*simp add*: *hermitean-def mat-adj-def mat-cnj-def*)
lift-definition *y-axis* :: *circline* **is** *y-axis-clmat*
done

lemma *special-points-on-y-axis'* [*simp*]:
shows *on-circline y-axis* 0_h **and** *on-circline y-axis* ii_h **and** *on-circline y-axis* ∞_h
by (*transfer*, *transfer*, *simp add*: *vec-cnj-def*)+

lemma *special-points-on-y-axis''* [*simp*]:
shows 0_h ∈ *circline-set y-axis* **and** ii_h ∈ *circline-set y-axis* **and** ∞_h ∈ *circline-set y-axis*
unfolding *circline-set-def*
by *auto*

lemma *on-circline-y-axis*:
shows *on-circline y-axis z* ⟷ (∃ *c*. *is-imag c* ∧ *z* = *of-complex c*) ∨ *z* = ∞_h
proof *safe*
fix *z c*
assume *is-imag c*
thus *on-circline y-axis* (*of-complex c*)
proof (*transfer*, *transfer*)
fix *c*
assume *is-imag c*
thus *on-circline-cmat-cvec y-axis-cmat* (*of-complex-cvec c*)
using *eq-minus-cnj-iff-imag*[*of c*]
by (*simp add*: *vec-cnj-def*)
qed
next
fix *z*
assume *on-circline y-axis z z* ≠ ∞_h
thus ∃ *c*. *is-imag c* ∧ *z* = *of-complex c*
proof (*transfer*, *transfer*, *safe*)
fix *a b*
assume (*a*, *b*) ≠ *vec-zero*
on-circline-cmat-cvec y-axis-cmat (*a*, *b*)
¬ (*a*, *b*) ≈$_v$ ∞$_v$
hence *b* ≠ *0 cnj a* ∗ *b* + *cnj b* ∗ *a* = *0*
using *inf-cvec-z2-zero-iff*
apply *blast*
apply (*smt* (*verit*, *ccfv-threshold*) ‹*on-circline-cmat-cvec y-axis-cmat* (*a*, *b*)› *add-0 add-cancel-left-right mult.commute*
mult-cancel-left2 on-circline-cmat-cvec-circline-equation y-axis-cmat-def)
done
thus ∃ *c*. *is-imag c* ∧ (*a*, *b*) ≈$_v$ *of-complex-cvec c*
using *eq-minus-cnj-iff-imag*[*of a / b*]
apply (*rule-tac x*=*a*/*b* **in** *exI*)
apply (*auto simp add*: *field-simps*)
apply (*rule-tac x*=*1*/*b* **in** *exI*, *simp*)
using *add-eq-0-iff* **apply** *blast*
apply (*rule-tac x*=*1*/*b* **in** *exI*, *simp*)
done
qed

next
show *on-circline y-axis* ∞_h
by *simp*
qed

lemma *on-circline-y-axis-I* [*simp*]:
assumes *is-imag z*
shows *on-circline y-axis* (*of-complex z*)
using *assms*
unfolding *on-circline-y-axis*
by *auto*

lemma *circline-set-y-axis*:
shows *circline-set y-axis* = *of-complex* ‘ {*x*. *is-imag x*} ∪ {∞_h}
using *on-circline-y-axis*
unfolding *circline-set-def*
by *auto*

lemma *circline-set-y-axis-I*:
assumes *is-imag z*
shows *of-complex z* ∈ *circline-set y-axis*
using *assms*
unfolding *circline-set-y-axis*
by *auto*

Positive and negative part of y-axis

definition *positive-y-axis* **where**
positive-y-axis = {*z*. *z* ∈ *circline-set y-axis* ∧ *z* ≠ ∞_h ∧ *Im* (*to-complex z*) > *0*}

definition *negative-y-axis* **where**
negative-y-axis = {*z*. *z* ∈ *circline-set y-axis* ∧ *z* ≠ ∞_h ∧ *Im* (*to-complex z*) < *0*}

lemma *circline-set-positive-y-axis-I* [*simp*]:
assumes *is-imag z* **and** *Im z* > *0*
shows *of-complex z* ∈ *positive-y-axis*
using *assms*
unfolding *positive-y-axis-def*
by *simp*

lemma *circline-set-negative-y-axis-I* [*simp*]:
assumes *is-imag z* **and** *Im z* < *0*
shows *of-complex z* ∈ *negative-y-axis*
using *assms*
unfolding *negative-y-axis-def*
by *simp*

7.5.4 Point zero as a circline

definition *circline-point-0-cmat* :: *complex-mat* **where**
[*simp*]: *circline-point-0-cmat* = *(1, 0, 0, 0)*
lift-definition *circline-point-0-clmat* :: *circline-mat* **is** *circline-point-0-cmat*
by (*simp add*: *hermitean-def mat-adj-def mat-cnj-def*)
lift-definition *circline-point-0* :: *circline* **is** *circline-point-0-clmat*
done

lemma *circline-type-circline-point-0* [*simp*]:
shows *circline-type circline-point-0* = *0*
by (*transfer*, *transfer*, *simp*)

lemma *zero-in-circline-point-0* [*simp*]:
shows 0_h ∈ *circline-set circline-point-0*
unfolding *circline-set-def*
by *auto* (*transfer*, *transfer*, *simp add*: *vec-cnj-def*)+

7.5.5 Imaginary unit circle

definition *imag-unit-circle-cmat* :: *complex-mat* **where**

[*simp*]: *imag-unit-circle-cmat* = (*1*, *0*, *0*, *1*)
lift-definition *imag-unit-circle-clmat* :: *circline-mat* **is** *imag-unit-circle-cmat*
by (*simp add*: *hermitean-def mat-adj-def mat-cnj-def*)
lift-definition *imag-unit-circle* :: *circline* **is** *imag-unit-circle-clmat*
done

lemma *circline-type-imag-unit-circle* [*simp*]:
shows *circline-type imag-unit-circle* = *1*
by (*transfer*, *transfer*, *simp*)

7.6 Intersection of circlines

definition *circline-intersection* :: *circline* ⇒ *circline* ⇒ *complex-homo set* **where**
circline-intersection H1 H2 = {*z*. *on-circline H1 z* ∧ *on-circline H2 z*}

lemma *circline-equation-cancel-z2*:
assumes *circline-equation A B C D z1 z2* **and** *z2* ≠ *0*
shows *circline-equation A B C D* (*z1*/*z2*) *1*
using *assms*
by (*simp add*: *field-simps*)

lemma *circline-equation-quadratic-equation*:
assumes *circline-equation A B* (*cnj B*) *D z 1* **and**
Re z = *x* **and** *Im z* = *y* **and** *Re B* = *bx* **and** *Im B* = *by*
shows *A*∗*x*^{2} + *A*∗*y*^{2} + *2*∗*bx*∗*x* + *2*∗*by*∗*y* + *D* = *0*
using *assms*
proof−
have *z* = *x* + i∗*y B* = *bx* + i∗*by*
using *assms complex-eq*
by *auto*
thus *?thesis*
using *assms*
by (*simp add*: *field-simps power2-eq-square*)
qed

lemma *circline-intersection-symetry*:
shows *circline-intersection H1 H2* = *circline-intersection H2 H1*
unfolding *circline-intersection-def*
by *auto*

7.7 Möbius action on circlines

definition *moebius-circline-cmat-cmat* :: *complex-mat* ⇒ *complex-mat* ⇒ *complex-mat* **where**
[*simp*]: *moebius-circline-cmat-cmat M H* = *congruence* (*mat-inv M*) *H*

lift-definition *moebius-circline-mmat-clmat* :: *moebius-mat* ⇒ *circline-mat* ⇒ *circline-mat* **is** *moebius-circline-cmat-cmat*
using *mat-det-inv congruence-nonzero hermitean-congruence*
by *simp*

lift-definition *moebius-circline* :: *moebius* ⇒ *circline* ⇒ *circline* **is** *moebius-circline-mmat-clmat*
proof *transfer*
fix *M M′ H H′*
assume *moebius-cmat-eq M M′ circline-eq-cmat H H′*
thus *circline-eq-cmat* (*moebius-circline-cmat-cmat M H*) (*moebius-circline-cmat-cmat M′ H′*)
by (*auto simp add*: *mat-inv-mult-sm*) (*rule-tac x*=*ka* / *Re* (*k* ∗ *cnj k*) **in** *exI*, *auto simp add*: *complex-mult-cnj-cmod power2-eq-square*)
qed

lemma *moebius-preserve-circline-type* [*simp*]:
shows *circline-type* (*moebius-circline M H*) = *circline-type H*
proof (*transfer*, *transfer*)
fix *M H* :: *complex-mat*
assume *mat-det M* ≠ *0 hermitean H* ∧ *H* ≠ *mat-zero*
thus *circline-type-cmat* (*moebius-circline-cmat-cmat M H*) = *circline-type-cmat H*
using *Re-det-sgn-congruence*[*of mat-inv M H*] *mat-det-inv*[*of M*]
by (*simp del*: *congruence-def*)

qed

The central lemma in this section connects the action of Möbius transformations on points and on circlines.

lemma *moebius-circline*:
shows $\{z.\ on\text{-}circline\ (moebius\text{-}circline\ M\ H)\ z\} =$
$moebius\text{-}pt\ M\ `\ \{z.\ on\text{-}circline\ H\ z\}$
proof *safe*
fix z
assume *on-circline H z*
thus *on-circline* (*moebius-circline M H*) (*moebius-pt M z*)
proof (*transfer*, *transfer*)
fix z :: *complex-vec* **and** $M\ H$ :: *complex-mat*
assume *hh*: *hermitean H* $\wedge$ $H \neq$ *mat-zero* $z \neq$ *vec-zero mat-det M* $\neq 0$
let *?z* = $M *_{mv} z$
let *?H* = *mat-adj* (*mat-inv M*) $*_{mm}$ H $*_{mm}$ (*mat-inv M*)
assume $*$: *on-circline-cmat-cvec H z*
hence *quad-form z H* = *0*
by *simp*
hence *quad-form ?z ?H* = *0*
using *quad-form-congruence*[*of M z H*] *hh*
by *simp*
thus *on-circline-cmat-cvec* (*moebius-circline-cmat-cmat M H*) (*moebius-pt-cmat-cvec M z*)
by *simp*
qed
next
fix z
assume *on-circline* (*moebius-circline M H*) z
hence $\exists\ z'.\ z = moebius\text{-}pt\ M\ z' \wedge on\text{-}circline\ H\ z'$
proof (*transfer*, *transfer*)
fix z :: *complex-vec* **and** $M\ H$ :: *complex-mat*
assume *hh*: *hermitean H* $\wedge$ $H \neq$ *mat-zero* $z \neq$ *vec-zero mat-det M* $\neq 0$
let *?iM* = *mat-inv M*
let *?z'* = *?iM* $*_{mv}$ z
assume $*$: *on-circline-cmat-cvec* (*moebius-circline-cmat-cmat M H*) z
have *?z'* $\neq$ *vec-zero*
using *hh*
using *mat-det-inv mult-mv-nonzero*
by *auto*
moreover
have $z \approx_v$ *moebius-pt-cmat-cvec M ?z'*
using *hh eye-mv-l mat-inv-r*
by *simp*
moreover
have $M *_{mv}$ (*?iM* $*_{mv}$ z) = z
using *hh eye-mv-l mat-inv-r*
by *auto*
hence *on-circline-cmat-cvec H ?z'*
using *hh* $*$
using *quad-form-congruence*[*of M ?iM* $*_{mv}$ *z H*, *symmetric*]
unfolding *moebius-circline-cmat-cmat-def*
unfolding *on-circline-cmat-cvec-def*
by *simp*
ultimately
show $\exists z' \in \{v.\ v \neq vec\text{-}zero\}.\ z \approx_v moebius\text{-}pt\text{-}cmat\text{-}cvec\ M\ z' \wedge on\text{-}circline\text{-}cmat\text{-}cvec\ H\ z'$
by *blast*
qed
thus $z \in moebius\text{-}pt\ M\ `\ \{z.\ on\text{-}circline\ H\ z\}$
by *auto*
qed

lemma *on-circline-moebius-circline-I* [*simp*]:
assumes *on-circline H z*
shows *on-circline* (*moebius-circline M H*) (*moebius-pt M z*)
using *assms moebius-circline*
by *fastforce*

lemma *circline-set-moebius-circline* [*simp*]:
shows *circline-set* (*moebius-circline M H*) = *moebius-pt M ' circline-set H*
using *moebius-circline*[*of M H*]
unfolding *circline-set-def*
by *auto*

lemma *circline-set-moebius-circline-I* [*simp*]:
assumes *z* ∈ *circline-set H*
shows *moebius-pt M z* ∈ *circline-set* (*moebius-circline M H*)
using *assms*
by *simp*

lemma *circline-set-moebius-circline-E*:
assumes *moebius-pt M z* ∈ *circline-set* (*moebius-circline M H*)
shows *z* ∈ *circline-set H*
using *assms*
using *moebius-pt-eq-I*[*of M z*]
by *auto*

lemma *circline-set-moebius-circline-iff* [*simp*]:
shows *moebius-pt M z* ∈ *circline-set* (*moebius-circline M H*) ⟷
z ∈ *circline-set H*
using *moebius-pt-eq-I*[*of M z*]
by *auto*

lemma *inj-moebius-circline*:
shows *inj* (*moebius-circline M*)
unfolding *inj-on-def*
proof (*safe*)
fix *H H'*
assume *moebius-circline M H* = *moebius-circline M H'*
thus *H* = *H'*
proof (*transfer*, *transfer*)
fix *M H H'* :: *complex-mat*
assume *hh*: *mat-det M* ≠ *0*
let *?iM* = *mat-inv M*
assume *circline-eq-cmat* (*moebius-circline-cmat-cmat M H*) (*moebius-circline-cmat-cmat M H'*)
then obtain *k* **where** *congruence ?iM H'* = *congruence ?iM* (*cor k* ∗$_{sm}$ *H*) *k* ≠ *0*
by *auto*
thus *circline-eq-cmat H H'*
using *hh inj-congruence*[*of ?iM H' cor k* ∗$_{sm}$ *H*] *mat-det-inv*[*of M*]
by *auto*
qed
qed

lemma *moebius-circline-eq-I*:
assumes *moebius-circline M H1* = *moebius-circline M H2*
shows *H1* = *H2*
using *assms inj-moebius-circline*[*of M*]
unfolding *inj-on-def*
by *blast*

lemma *moebius-circline-neq-I* [*simp*]:
assumes *H1* ≠ *H2*
shows *moebius-circline M H1* ≠ *moebius-circline M H2*
using *assms inj-moebius-circline*[*of M*]
unfolding *inj-on-def*
by *blast*

7.7.1 Group properties of Möbius action on ciclines

Möbius actions on circlines have similar properties as Möbius actions on points.

lemma *moebius-circline-id* [*simp*]:
shows *moebius-circline id-moebius H* = *H*
by (*transfer*, *transfer*) (*simp add*: *mat-adj-def mat-cnj-def*, *rule-tac x=1* **in** *exI*, *auto*)

lemma *moebius-circline-comp* [*simp*]:
shows *moebius-circline* (*moebius-comp M1 M2*) *H* = *moebius-circline M1* (*moebius-circline M2 H*)
by (*transfer*, *transfer*) (*simp add*: *mat-inv-mult-mm*, *rule-tac x=1* **in** *exI*, *simp add*: *mult-mm-assoc*)

lemma *moebius-circline-comp-inv-left* [*simp*]:
shows *moebius-circline* (*moebius-inv M*) (*moebius-circline M H*) = *H*
by (*subst moebius-circline-comp*[*symmetric*], *simp*)

lemma *moebius-circline-comp-inv-right* [*simp*]:
shows *moebius-circline M* (*moebius-circline* (*moebius-inv M*) *H*) = *H*
by (*subst moebius-circline-comp*[*symmetric*], *simp*)

7.8 Action of Euclidean similarities on circlines

lemma *moebius-similarity-lines-to-lines* [*simp*]:
assumes *a* ≠ *0*
shows ∞$_h$ ∈ *circline-set* (*moebius-circline* (*moebius-similarity a b*) *H*) ⟷
∞$_h$ ∈ *circline-set H*
using *assms*
by (*metis circline-set-moebius-circline-iff moebius-similarity-inf*)

lemma *moebius-similarity-lines-to-lines′*:
assumes *a* ≠ *0*
shows *on-circline* (*moebius-circline* (*moebius-similarity a b*) *H*) ∞$_h$ ⟷
∞$_h$ ∈ *circline-set H*
using *moebius-similarity-lines-to-lines assms*
unfolding *circline-set-def*
by *simp*

7.9 Conjugation, recpiprocation and inversion of circlines

Conjugation of circlines

definition *conjugate-circline-cmat* :: *complex-mat* ⇒ *complex-mat* **where**
[*simp*]: *conjugate-circline-cmat* = *mat-cnj*
lift-definition *conjugate-circline-clmat* :: *circline-mat* ⇒ *circline-mat* **is** *conjugate-circline-cmat*
by (*auto simp add*: *hermitean-def mat-adj-def mat-cnj-def*)
lift-definition *conjugate-circline* :: *circline* ⇒ *circline* **is** *conjugate-circline-clmat*
by *transfer* (*metis circline-eq-cmat-def conjugate-circline-cmat-def hermitean-transpose mat-t-mult-sm*)

lemma *conjugate-circline-set′*:
shows *conjugate* ‘ *circline-set H* ⊆ *circline-set* (*conjugate-circline H*)
proof (*safe*)
fix *z*
assume *z* ∈ *circline-set H*
thus *conjugate z* ∈ *circline-set* (*conjugate-circline H*)
unfolding *circline-set-def*
apply *simp*
apply (*transfer*, *transfer*)
unfolding *on-circline-cmat-cvec-def conjugate-cvec-def conjugate-circline-cmat-def*
apply (*subst quad-form-vec-cnj-mat-cnj*, *simp-all*)
done
qed

lemma *conjugate-conjugate-circline* [*simp*]:
shows *conjugate-circline* (*conjugate-circline H*) = *H*
by (*transfer*, *transfer*, *force*)

lemma *circline-set-conjugate-circline* [*simp*]:
shows *circline-set* (*conjugate-circline H*) = *conjugate* ‘ *circline-set H* (**is** *?lhs* = *?rhs*)
proof (*safe*)
fix *z*
assume *z* ∈ *?lhs*
show *z* ∈ *?rhs*
proof
show *z* = *conjugate* (*conjugate z*)
by *simp*

```
  next
    show conjugate z ∈ circline-set H
      using ‹z ∈ circline-set (conjugate-circline H)›
      using conjugate-circline-set'[of conjugate-circline H]
      by auto
  qed
next
  fix z
  assume z ∈ circline-set H
  thus conjugate z ∈ circline-set (conjugate-circline H)
    using conjugate-circline-set'[of H]
    by auto
qed

lemma on-circline-conjugate-circline [simp]:
  shows on-circline (conjugate-circline H) z ⟷ on-circline H (conjugate z)
  using circline-set-conjugate-circline[of H]
  unfolding circline-set-def
  by force
```

Inversion of circlines

```
definition circline-inversion-cmat :: complex-mat ⇒ complex-mat where
  [simp]: circline-inversion-cmat H = (let (A, B, C, D) = H in (D, B, C, A))
lift-definition circline-inversion-clmat :: circline-mat ⇒ circline-mat is circline-inversion-cmat
  by (auto simp add: hermitean-def mat-adj-def mat-cnj-def)
lift-definition circline-inversion :: circline ⇒ circline is circline-inversion-clmat
  by transfer auto

lemma on-circline-circline-inversion [simp]:
  shows on-circline (circline-inversion H) z ⟷ on-circline H (reciprocal (conjugate z))
  by (transfer, transfer, auto simp add: vec-cnj-def field-simps)

lemma circline-set-circline-inversion [simp]:
  shows circline-set (circline-inversion H) = inversion ‘ circline-set H
  unfolding circline-set-def inversion-def
  by (force simp add: comp-def image-iff)
```

Reciprocal of circlines

```
definition circline-reciprocal :: circline ⇒ circline where
  circline-reciprocal = conjugate-circline ∘ circline-inversion

lemma circline-set-circline-reciprocal:
  shows circline-set (circline-reciprocal H) = reciprocal ‘ circline-set H
  unfolding circline-reciprocal-def comp-def
  by (auto simp add: inversion-def image-iff)
```

Rotation of circlines

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

lemma rotation-minus-pi-2-y-axis [simp]:
  shows moebius-circline (moebius-rotation (−pi/2)) y-axis = x-axis
  unfolding moebius-rotation-def moebius-similarity-def
  by (transfer, transfer, simp add: mat-adj-def mat-cnj-def, rule-tac x=−1 in exI, simp)

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

lemma rotation-pi-2-x-axis [simp]:
  shows moebius-circline (moebius-rotation (pi/2)) x-axis = y-axis
  unfolding moebius-rotation-def moebius-similarity-def
```

by (*transfer*, *transfer*, *simp add*: *mat-adj-def mat-cnj-def*, *rule-tac x=−1* **in** *exI*, *simp*)

lemma *rotation-minus-pi-2-positive-y-axis* [*simp*]:
shows (*moebius-pt* (*moebius-rotation* (*−pi/2*))) ‘ *positive-y-axis* = *positive-x-axis*
proof *safe*
fix *y*
assume *y*: *y* ∈ *positive-y-axis*
have ∗: *Re* (*a* ∗ i / *b*) < *0* ⟷ *Im* (*a* / *b*) > *0* **for** *a b*
by (*subst times-divide-eq-left* [*symmetric*], *subst mult.commute*, *subst Re-i-times*) *auto*
from *y* ∗ **show** *moebius-pt* (*moebius-rotation* (*−pi/2*)) *y* ∈ *positive-x-axis*
unfolding *positive-y-axis-def positive-x-axis-def circline-set-def*
unfolding *moebius-rotation-def moebius-similarity-def*
apply *simp*
apply *transfer*
apply *transfer*
apply (*auto simp add*: *vec-cnj-def field-simps add-eq-0-iff*)
done
next
fix *x*
assume *x*: *x* ∈ *positive-x-axis*
let *?y* = *moebius-pt* (*moebius-rotation* (*pi/2*)) *x*
have ∗: *Im* (*a* ∗ i / *b*) > *0* ⟷ *Re* (*a* / *b*) > *0* **for** *a b*
by (*subst times-divide-eq-left* [*symmetric*], *subst mult.commute*, *subst Im-i-times*) *auto*
hence *?y* ∈ *positive-y-axis*
using ‹*x* ∈ *positive-x-axis*›
unfolding *positive-x-axis-def positive-y-axis-def*
unfolding *moebius-rotation-def moebius-similarity-def*
unfolding *circline-set-def*
apply *simp*
apply *transfer*
apply *transfer*
apply (*auto simp add*: *vec-cnj-def field-simps add-eq-0-iff*)
done
thus *x* ∈ *moebius-pt* (*moebius-rotation* (*−pi/2*)) ‘ *positive-y-axis*
by (*auto simp add*: *image-iff*) (*rule-tac x=?y* **in** *bexI*, *simp-all*)
qed

7.10 Circline uniqueness

7.10.1 Zero type circline uniqueness

lemma *unique-circline-type-zero-0′*:
shows (*circline-type circline-point-0* = *0* ∧ *0*$_h$ ∈ *circline-set circline-point-0*) ∧
(∀ *H*. *circline-type H* = *0* ∧ *0*$_h$ ∈ *circline-set H* ⟶ *H* = *circline-point-0*)
unfolding *circline-set-def*
proof (*safe*)
show *circline-type circline-point-0* = *0*
by (*transfer*, *transfer*, *simp*)
next
show *on-circline circline-point-0 0*$_h$
using *circline-set-def zero-in-circline-point-0*
by *auto*
next
fix *H*
assume *circline-type H* = *0 on-circline H 0*$_h$
thus *H* = *circline-point-0*
proof (*transfer*, *transfer*)
fix *H* :: *complex-mat*
assume *hh*: *hermitean H* ∧ *H* ≠ *mat-zero*
obtain *A B C D* **where** *HH*: *H* = (*A*, *B*, *C*, *D*)
by (*cases H*) *auto*
hence ∗: *C* = *cnj B is-real A*
using *hh hermitean-elems*[*of A B C D*]
by *auto*
assume *circline-type-cmat H* = *0 on-circline-cmat-cvec H 0*$_v$
thus *circline-eq-cmat H circline-point-0-cmat*
using *HH hh* ∗

```
    by (simp add: Let-def vec-cnj-def sgn-minus sgn-mult sgn-zero-iff)
      (rule-tac x=1/Re A in exI, cases A, cases B, simp add: Complex-eq sgn-zero-iff)
  qed
qed

lemma unique-circline-type-zero-0:
  shows ∃! H. circline-type H = 0 ∧ 0_h ∈ circline-set H
  using unique-circline-type-zero-0'
  by blast

lemma unique-circline-type-zero:
  shows ∃! H. circline-type H = 0 ∧ z ∈ circline-set H
proof−
  obtain M where ++: moebius-pt M z = 0_h
    using ex-moebius-1[of z]
    by auto
  have +++: z = moebius-pt (moebius-inv M) 0_h
    by (subst ++[symmetric]) simp
  then obtain H0 where ∗: circline-type H0 = 0 ∧ 0_h ∈ circline-set H0 and
    ∗∗: ∀ H'. circline-type H' = 0 ∧ 0_h ∈ circline-set H' ⟶ H' = H0
    using unique-circline-type-zero-0
    by auto
  let ?H' = moebius-circline (moebius-inv M) H0
  show ?thesis
    unfolding Ex1-def
    using ∗ +++
  proof (rule-tac x=?H' in exI, simp, safe)
    fix H'
    assume circline-type H' = 0 moebius-pt (moebius-inv M) 0_h ∈ circline-set H'
    hence 0_h ∈ circline-set (moebius-circline M H')
      using ++ +++
      by force
    hence moebius-circline M H' = H0
      using ∗∗[rule-format, of moebius-circline M H']
      using ‹circline-type H' = 0›
      by simp
    thus H' = moebius-circline (moebius-inv M) H0
      by auto
  qed
qed
```

7.10.2 Negative type circline uniqueness

```
lemma unique-circline-01inf':
  shows 0_h ∈ circline-set x-axis ∧ 1_h ∈ circline-set x-axis ∧ ∞_h ∈ circline-set x-axis ∧
      (∀ H. 0_h ∈ circline-set H ∧ 1_h ∈ circline-set H ∧ ∞_h ∈ circline-set H ⟶ H = x-axis)
proof safe
  fix H
  assume 0_h ∈ circline-set H  1_h ∈ circline-set H ∞_h ∈ circline-set H
  thus H = x-axis
    unfolding circline-set-def
    apply simp
  proof (transfer, transfer)
    fix H
    assume hh: hermitean H ∧ H ≠ mat-zero
    obtain A B C D where HH: H = (A, B, C, D)
      by (cases H) auto
    have ∗: C = cnj B A = 0 ∧ D = 0 ⟶ B ≠ 0
      using hermitean-elems[of A B C D] hh HH
      by auto
    obtain Bx By where B = Complex Bx By
      by (cases B) auto
    assume on-circline-cmat-cvec H 0_v on-circline-cmat-cvec H 1_v on-circline-cmat-cvec H ∞_v
    thus circline-eq-cmat H x-axis-cmat
      using ∗ HH ‹C = cnj B› ‹B = Complex Bx By›
      by (simp add: Let-def vec-cnj-def Complex-eq) (rule-tac x=1/By in exI, auto)
```

```
  qed
qed simp-all

lemma unique-circline-set:
  assumes A ≠ B and A ≠ C and B ≠ C
  shows ∃! H. A ∈ circline-set H ∧ B ∈ circline-set H ∧ C ∈ circline-set H
proof−
  let ?P = λ A B C. A ≠ B ∧ A ≠ C ∧ B ≠ C ⟶ (∃! H. A ∈ circline-set H ∧ B ∈ circline-set H ∧ C ∈ circline-set
H)
  have ?P A B C
  proof (rule wlog-moebius-01inf[of ?P])
    fix M a b c
    let ?M = moebius-pt M
    assume ?P a b c
    show ?P (?M a) (?M b) (?M c)
    proof
      assume ?M a ≠ ?M b ∧ ?M a ≠ ?M c ∧ ?M b ≠ ?M c
      hence a ≠ b b ≠ c a ≠ c
        by auto
      hence ∃!H. a ∈ circline-set H ∧ b ∈ circline-set H ∧ c ∈ circline-set H
        using ‹?P a b c›
        by simp
      then obtain H where
        ∗: a ∈ circline-set H ∧ b ∈ circline-set H ∧ c ∈ circline-set H and
        ∗∗: ∀ H'. a ∈ circline-set H' ∧ b ∈ circline-set H' ∧ c ∈ circline-set H' ⟶ H' = H
        unfolding Ex1-def
        by auto
      let ?H' = moebius-circline M H
      show ∃! H. ?M a ∈ circline-set H ∧ moebius-pt M b ∈ circline-set H ∧ moebius-pt M c ∈ circline-set H
        unfolding Ex1-def
      proof (rule-tac x=?H' in exI, rule)
        show ?M a ∈ circline-set ?H' ∧ ?M b ∈ circline-set ?H' ∧ ?M c ∈ circline-set ?H'
          using ∗
          by auto
      next
        show ∀ H'. ?M a ∈ circline-set H' ∧ ?M b ∈ circline-set H' ∧ ?M c ∈ circline-set H' ⟶ H' = ?H'
        proof (safe)
          fix H'
          let ?iH' = moebius-circline (moebius-inv M) H'
          assume ?M a ∈ circline-set H' ?M b ∈ circline-set H' ?M c ∈ circline-set H'
          hence a ∈ circline-set ?iH' ∧ b ∈ circline-set ?iH' ∧ c ∈ circline-set ?iH'
            by simp
          hence H = ?iH'
            using ∗∗
            by blast
          thus H' = moebius-circline M H
            by simp
        qed
      qed
    qed
  next
    show ?P 0h 1h ∞h
      using unique-circline-01inf'
      unfolding Ex1-def
      by (safe, rule-tac x=x-axis in exI) auto
  qed fact+
  thus ?thesis
    using assms
    by simp
qed

lemma zero-one-inf-x-axis [simp]:
  assumes 0h ∈ circline-set H and 1h ∈ circline-set H and ∞h ∈ circline-set H
  shows H = x-axis
  using assms unique-circline-set[of 0h 1h ∞h]
  by auto
```

7.11 Circline set cardinality

7.11.1 Diagonal circlines

definition *is-diag-circline-cmat* :: *complex-mat* ⇒ *bool* **where**
[*simp*]: *is-diag-circline-cmat H* = (*let* (*A*, *B*, *C*, *D*) = *H in B* = *0* ∧ *C* = *0*)
lift-definition *is-diag-circline-clmat* :: *circline-mat* ⇒ *bool* **is** *is-diag-circline-cmat*
done
lift-definition *circline-diag* :: *circline* ⇒ *bool* **is** *is-diag-circline-clmat*
by *transfer auto*

lemma *circline-diagonalize*:
shows ∃ *M H'*. *moebius-circline M H* = *H'* ∧ *circline-diag H'*
proof (*transfer*, *transfer*)
fix *H*
assume *hh*: *hermitean H* ∧ *H* ≠ *mat-zero*
obtain *A B C D* **where** *HH*: *H* = (*A*, *B*, *C*, *D*)
by (*cases H*) *auto*
hence *HH-elems*: *is-real A is-real D C* = *cnj B*
using *hermitean-elems*[*of A B C D*] *hh*
by *auto*
obtain *M k1 k2* **where** ∗: *mat-det M* ≠ *0 unitary M congruence M H* = (*k1*, *0*, *0*, *k2*) *is-real k1 is-real k2*
using *hermitean-diagonizable*[*of H*] *hh*
by *auto*
have *k1* ≠ *0* ∨ *k2* ≠ *0*
using ‹*congruence M H* = (*k1*, *0*, *0*, *k2*)› *hh congruence-nonzero*[*of H M*] ‹*mat-det M* ≠ *0*›
by *auto*
let *?M'* = *mat-inv M*
let *?H'* = (*k1*, *0*, *0*, *k2*)
have *circline-eq-cmat* (*moebius-circline-cmat-cmat ?M' H*) *?H'* ∧ *is-diag-circline-cmat ?H'*
using ∗
by *force*
moreover
have *?H'* ∈ *hermitean-nonzero*
using ∗ ‹*k1* ≠ *0* ∨ *k2* ≠ *0*› *eq-cnj-iff-real*[*of k1*] *eq-cnj-iff-real*[*of k2*]
by (*auto simp add*: *hermitean-def mat-adj-def mat-cnj-def*)
moreover
have *mat-det ?M'* ≠ *0*
using ∗ *mat-det-inv*[*of M*]
by *auto*
ultimately
show ∃ *M*∈{*M*. *mat-det M* ≠ *0*}.
∃ *H'*∈*hermitean-nonzero*.
circline-eq-cmat (*moebius-circline-cmat-cmat M H*) *H'* ∧ *is-diag-circline-cmat H'*
by *blast*
qed

lemma *wlog-circline-diag*:
assumes ⋀ *H*. *circline-diag H* ⟹ *P H*
⋀ *M H*. *P H* ⟹ *P* (*moebius-circline M H*)
shows *P H*
proof−
obtain *M H'* **where** *moebius-circline M H* = *H' circline-diag H'*
using *circline-diagonalize*[*of H*]
by *auto*
hence *P* (*moebius-circline M H*)
using *assms*(*1*)
by *simp*
thus *?thesis*
using *assms*(*2*)[*of moebius-circline M H moebius-inv M*]
by *simp*
qed

7.11.2 Zero type circline set cardinality

lemma *circline-type-zero-card-eq1-0*:
assumes *circline-type H* = *0* **and** *0*$_h$ ∈ *circline-set H*

```
  shows circline-set H = {0_h}
using assms
unfolding circline-set-def
proof(safe)
  fix z
  assume on-circline H z circline-type H = 0 on-circline H 0_h
  hence H = circline-point-0
    using unique-circline-type-zero-0'
    unfolding circline-set-def
    by simp
  thus z = 0_h
    using ‹on-circline H z›
    by (transfer, transfer) (case-tac z, case-tac H, force simp add: vec-cnj-def)
qed


lemma circline-type-zero-card-eq1:
  assumes circline-type H = 0
  shows ∃ z. circline-set H = {z}
proof−
  have ∃ z. on-circline H z
    using assms
  proof (transfer, transfer)
    fix H
    assume hh: hermitean H ∧ H ≠ mat-zero
    obtain A B C D where HH: H = (A, B, C, D)
      by (cases H) auto
    hence C = cnj B is-real A is-real D
      using hh hermitean-elems[of A B C D]
      by auto
    assume circline-type-cmat H = 0
    hence mat-det H = 0
      by (simp add: complex-eq-if-Re-eq hh mat-det-hermitean-real sgn-eq-0-iff)
    hence A∗D = B∗C
      using HH
      by simp
    show Bex {v. v ≠ vec-zero} (on-circline-cmat-cvec H)
    proof (cases A ≠ 0 ∨ B ≠ 0)
      case True
      thus ?thesis
        using HH ‹A∗D = B∗C›
        by (rule-tac x=(−B, A) in bexI) (auto simp add: Let-def vec-cnj-def field-simps)
    next
      case False
      thus ?thesis
        using HH ‹C = cnj B›
        by (rule-tac x=(1, 0) in bexI) (simp-all add: Let-def vec-cnj-def)
    qed
  qed
  then obtain z where on-circline H z
    by auto
  obtain M where moebius-pt M z = 0_h
    using ex-moebius-1[of z]
    by auto
  hence 0_h ∈ circline-set (moebius-circline M H)
    using on-circline-moebius-circline-I[OF ‹on-circline H z›, of M]
    unfolding circline-set-def
    by simp
  hence circline-set (moebius-circline M H) = {0_h}
    using circline-type-zero-card-eq1-0[of moebius-circline M H] ‹circline-type H = 0›
    by auto
  hence circline-set H = {z}
    using ‹moebius-pt M z = 0_h›
    using bij-moebius-pt[of M] bij-image-singleton[of moebius-pt M circline-set H - z]
    by simp
  thus ?thesis
```

by *auto*
qed

7.11.3 Negative type circline set cardinality

lemma *quad-form-diagonal-iff*:
assumes $k1 \neq 0$ **and** *is-real k1* **and** *is-real k2* **and** $Re\ k1 * Re\ k2 < 0$
shows *quad-form* $(z1,\ 1)\ (k1,\ 0,\ 0,\ k2) = 0 \longleftrightarrow (\exists\ \varphi.\ z1 = rcis\ (sqrt\ (Re\ (-k2\ /k1)))\ \varphi)$
proof−
have $Re\ (-k2/k1) \geq 0$
using ‹$Re\ k1 * Re\ k2 < 0$› ‹*is-real k1*› ‹*is-real k2*› ‹$k1 \neq 0$›
using *Re-divide-real*[*of k1* $-k2$]
by (*smt* (*verit*) *divide-less-0-iff mult-nonneg-nonneg mult-nonpos-nonpos uminus-complex.simps*(1))

have *quad-form* $(z1,\ 1)\ (k1,\ 0,\ 0,\ k2) = 0 \longleftrightarrow (cor\ (cmod\ z1))^2 = -k2\ /\ k1$
using *assms add-eq-0-iff*[*of k2 k1*∗$(cor\ (cmod\ z1))^2$]
using *eq-divide-imp*[*of k1* $(cor\ (cmod\ z1))^2\ -k2$]
by (*auto simp add*: *vec-cnj-def field-simps complex-mult-cnj-cmod*)
also have ... $\longleftrightarrow (cmod\ z1)^2 = Re\ (-k2\ /k1)$
using *assms*
apply (*subst complex-eq-if-Re-eq*)
using *Re-complex-of-real*[*of* $(cmod\ z1)^2$] *div-reals*
by *auto*
also have ... $\longleftrightarrow cmod\ z1 = sqrt\ (Re\ (-k2\ /k1))$
by (*metis norm-ge-zero real-sqrt-ge-0-iff real-sqrt-pow2 real-sqrt-power*)
also have ... $\longleftrightarrow (\exists\ \varphi.\ z1 = rcis\ (sqrt\ (Re\ (-k2\ /k1)))\ \varphi)$
using *rcis-cmod-Arg*[*of z1*, *symmetric*] *assms abs-of-nonneg*[*of sqrt* $(Re\ (-k2/k1))$]
using ‹$Re\ (-k2/k1) \geq 0$›
by *auto*
finally show *?thesis*
.
qed

lemma *circline-type-neg-card-gt3-diag*:
assumes *circline-type H* < 0 **and** *circline-diag H*
shows $\exists\ A\ B\ C.\ A \neq B \wedge A \neq C \wedge B \neq C \wedge \{A,\ B,\ C\} \subseteq$ *circline-set H*
using *assms*
unfolding *circline-set-def*
apply (*simp del*: *HOL.ex-simps*)
proof (*transfer*, *transfer*)
fix *H*
assume *hh*: *hermitean H* $\wedge$ $H \neq$ *mat-zero*
obtain *A B C D* **where** *HH*: $H = (A,\ B,\ C,\ D)$
by (*cases H*) *auto*
hence *HH-elems*: *is-real A is-real D C* $=$ *cnj B*
using *hermitean-elems*[*of A B C D*] *hh*
by *auto*
assume *circline-type-cmat H* < 0 *is-diag-circline-cmat H*
hence $B = 0\ C = 0\ Re\ A * Re\ D < 0\ A \neq 0$
using *HH* ‹*is-real A*› ‹*is-real D*›
by *auto*

let *?x* $=$ $sqrt\ (Re\ (-\ D\ /\ A))$
let *?A* $= (rcis\ ?x\ 0,\ 1)$
let *?B* $= (rcis\ ?x\ (pi/2),\ 1)$
let *?C* $= (rcis\ ?x\ pi,\ 1)$
from *quad-form-diagonal-iff*[*OF* ‹$A \neq 0$› ‹*is-real A*› ‹*is-real D*› ‹$Re\ A * Re\ D < 0$›]
have *quad-form ?A* $(A,\ 0,\ 0,\ D) = 0$ *quad-form ?B* $(A,\ 0,\ 0,\ D) = 0$ *quad-form ?C* $(A,\ 0,\ 0,\ D) = 0$
by (*auto simp del*: *rcis-zero-arg*)
hence *on-circline-cmat-cvec H ?A* $\wedge$ *on-circline-cmat-cvec H ?B* $\wedge$ *on-circline-cmat-cvec H ?C*
using *HH* ‹$B = 0$› ‹$C = 0$›
by *simp*
moreover
have $Re\ (D\ /\ A) < 0$
using ‹$Re\ A * Re\ D < 0$› ‹$A \neq 0$› ‹*is-real A*› ‹*is-real D*›
using *Re-divide-real*[*of A D*]

by (*metis Re-complex-div-lt-0 Re-mult-real div-reals eq-cnj-iff-real is-real-div*)
hence ¬ *?A* ≈$_v$ *?B* ∧ ¬ *?A* ≈$_v$ *?C* ∧ ¬ *?B* ≈$_v$ *?C*
unfolding *rcis-def*
by (*auto simp add*: *cis-def complex.corec*)
moreover
have *?A* ≠ *vec-zero ?B* ≠ *vec-zero ?C* ≠ *vec-zero*
by *auto*
ultimately
show ∃ *A*∈{*v*. *v* ≠ *vec-zero*}. ∃ *B*∈{*v*. *v* ≠ *vec-zero*}. ∃ *C*∈{*v*. *v* ≠ *vec-zero*}.
¬ *A* ≈$_v$ *B* ∧ ¬ *A* ≈$_v$ *C* ∧ ¬ *B* ≈$_v$ *C* ∧
on-circline-cmat-cvec H A ∧ *on-circline-cmat-cvec H B* ∧ *on-circline-cmat-cvec H C*
by *blast*
qed

lemma *circline-type-neg-card-gt3*:
assumes *circline-type H* < *0*
shows ∃ *A B C*. *A* ≠ *B* ∧ *A* ≠ *C* ∧ *B* ≠ *C* ∧ {*A*, *B*, *C*} ⊆ *circline-set H*
proof−
obtain *M H′* **where** *moebius-circline M H* = *H′ circline-diag H′*
using *circline-diagonalize*[*of H*] *assms*
by *auto*
moreover
hence *circline-type H′* < *0*
using *assms moebius-preserve-circline-type*
by *auto*
ultimately
obtain *A B C* **where** *A* ≠ *B A* ≠ *C B* ≠ *C* {*A*, *B*, *C*} ⊆ *circline-set H′*
using *circline-type-neg-card-gt3-diag*[*of H′*]
by *auto*
let *?iM* = *moebius-inv M*
have *moebius-circline ?iM H′* = *H*
using ‹*moebius-circline M H* = *H′*›[*symmetric*]
by *simp*
let *?A* = *moebius-pt ?iM A* **and** *?B*= *moebius-pt ?iM B* **and** *?C* = *moebius-pt ?iM C*
have *?A* ∈ *circline-set H* *?B* ∈ *circline-set H* *?C* ∈ *circline-set H*
using ‹*moebius-circline ?iM H′* = *H*›[*symmetric*] ‹{*A*, *B*, *C*} ⊆ *circline-set H′*›
by *simp-all*
moreover
have *?A* ≠ *?B ?A* ≠ *?C ?B* ≠ *?C*
using ‹*A* ≠ *B*› ‹*A* ≠ *C*› ‹*B* ≠ *C*›
by *auto*
ultimately
show *?thesis*
by *auto*
qed

7.11.4 Positive type circline set cardinality

lemma *circline-type-pos-card-eq0-diag*:
assumes *circline-diag H* **and** *circline-type H* > *0*
shows *circline-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** *HH*: *H* = (*A*, *B*, *C*, *D*)
by (*cases H*) *auto*
hence *HH-elems*: *is-real A is-real D C* = *cnj B*
using *hermitean-elems*[*of A B C D*] *hh*
by *auto*
assume *is-diag-circline-cmat H 0* < *circline-type-cmat H*
hence *B* = *0 C* = *0 Re A* ∗ *Re D* > *0 A* ≠ *0*
using *HH* ‹*is-real A*› ‹*is-real D*›
by *auto*

```
  show ∀ x∈{v. v ≠ vec-zero}. ¬ on-circline-cmat-cvec H x
  proof
    fix x
    assume x ∈ {v. v ≠ vec-zero}
    obtain x1 x2 where xx: x = (x1, x2)
      by (cases x, auto)
    have (Re A > 0 ∧ Re D > 0) ∨ (Re A < 0 ∧ Re D < 0)
      using ‹Re A * Re D > 0›
      by (metis linorder-neqE-linordered-idom mult-eq-0-iff zero-less-mult-pos zero-less-mult-pos2)
    moreover
    have (Re (x1 * cnj x1) ≥ 0 ∧ Re (x2 * cnj x2) > 0) ∨ (Re (x1 * cnj x1) > 0 ∧ Re (x2 * cnj x2) ≥ 0)
      using ‹x ∈ {v. v ≠ vec-zero}› xx
      apply auto
      apply (simp add: complex-neq-0 power2-eq-square)+
      done
    ultimately
    have Re A * Re (x1 * cnj x1) + Re D * Re (x2 * cnj x2) ≠ 0
      by (smt (verit) mult-neg-pos mult-nonneg-nonneg mult-nonpos-nonneg mult-pos-pos)
    hence A * (x1 * cnj x1) + D * (x2 * cnj x2) ≠ 0
      using ‹is-real A› ‹is-real D›
      by (metis Re-mult-real plus-complex.simps(1) zero-complex.simps(1))
    thus ¬ on-circline-cmat-cvec H x
      using HH ‹B = 0› ‹C = 0› xx
      by (simp add: vec-cnj-def field-simps)
  qed
qed

lemma circline-type-pos-card-eq0:
  assumes circline-type H > 0
  shows circline-set H = {}
proof−
  obtain M H' where moebius-circline M H = H' circline-diag H'
    using circline-diagonalize[of H] assms
    by auto
  moreover
  hence circline-type H' > 0
    using assms moebius-preserve-circline-type
    by auto
  ultimately
  have circline-set H' = {}
    using circline-type-pos-card-eq0-diag[of H']
    by auto
  let ?iM = moebius-inv M
  have moebius-circline ?iM H' = H
    using ‹moebius-circline M H = H'›[symmetric]
    by simp
  thus ?thesis
    using ‹circline-set H' = {}›
    by auto
qed
```

7.11.5 Cardinality determines type

```
lemma card-eq1-circline-type-zero:
  assumes ∃ z. circline-set H = {z}
  shows circline-type H = 0
proof (cases circline-type H < 0)
  case True
  thus ?thesis
    using circline-type-neg-card-gt3[of H] assms
    by auto
next
  case False
  show ?thesis
  proof (cases circline-type H > 0)
    case True
```

thus *?thesis*
using *circline-type-pos-card-eq0*[*of H*] *assms*
by *auto*
next
case *False*
thus *?thesis*
using ‹¬ (*circline-type H*) < *0*›
by *simp*
qed
qed

7.11.6 Circline set is injective

lemma *inj-circline-set*:
assumes *circline-set H* = *circline-set H′* **and** *circline-set H* ≠ {}
shows *H* = *H′*
proof (*cases circline-type H* < *0*)
case *True*
then obtain *A B C* **where** *A* ≠ *B A* ≠ *C B* ≠ *C* {*A*, *B*, *C*} ⊆ *circline-set H*
using *circline-type-neg-card-gt3*[*of H*]
by *auto*
hence ∃!*H*. *A* ∈ *circline-set H* ∧ *B* ∈ *circline-set H* ∧ *C* ∈ *circline-set H*
using *unique-circline-set*[*of A B C*]
by *simp*
thus *?thesis*
using ‹*circline-set H* = *circline-set H′*› ‹{*A*, *B*, *C*} ⊆ *circline-set H*›
by *auto*
next
case *False*
show *?thesis*
proof (*cases circline-type H* = *0*)
case *True*
moreover
then obtain *A* **where** {*A*} = *circline-set H*
using *circline-type-zero-card-eq1*[*of H*]
by *auto*
moreover
hence *circline-type H′* = *0*
using ‹*circline-set H* = *circline-set H′*› *card-eq1-circline-type-zero*[*of H′*]
by *auto*
ultimately
show *?thesis*
using *unique-circline-type-zero*[*of A*] ‹*circline-set H* = *circline-set H′*›
by *auto*
next
case *False*
hence *circline-type H* > *0*
using ‹¬ (*circline-type H* < *0*)›
by *auto*
thus *?thesis*
using ‹*circline-set H* ≠ {}› *circline-type-pos-card-eq0*[*of H*]
by *auto*
qed
qed

7.12 Circline points - cross ratio real

lemma *four-points-on-circline-iff-cross-ratio-real*:
assumes *distinct* [*z*, *u*, *v*, *w*]
shows *is-real* (*to-complex* (*cross-ratio z u v w*)) ⟷
(∃ *H*. {*z*, *u*, *v*, *w*} ⊆ *circline-set H*)
proof−
have ∀ *z*. *distinct* [*z*, *u*, *v*, *w*] ⟶ *is-real* (*to-complex* (*cross-ratio z u v w*)) ⟷ (∃ *H*. {*z*, *u*, *v*, *w*} ⊆ *circline-set H*)
(**is** *?P u v w*)
proof (*rule wlog-moebius-01inf*[*of ?P u v w*])
fix *M a b c*

```
    assume aa: ?P a b c
    let ?Ma = moebius-pt M a and ?Mb = moebius-pt M b and ?Mc = moebius-pt M c
    show ?P ?Ma ?Mb ?Mc
    proof (rule allI, rule impI)
      fix z
      obtain d where *: z = moebius-pt M d
        using bij-moebius-pt[of M]
        unfolding bij-def
        by auto
      let ?Md = moebius-pt M d
      assume distinct [z, moebius-pt M a, moebius-pt M b, moebius-pt M c]
      hence distinct [a, b, c, d]
        using *
        by auto
      moreover
      have (∃ H. {d, a, b, c} ⊆ circline-set H) ⟷ (∃ H. {z, ?Ma, ?Mb, ?Mc} ⊆ circline-set H)
        using *
        apply auto
        apply (rule-tac x=moebius-circline M H in exI, simp)
        apply (rule-tac x=moebius-circline (moebius-inv M) H in exI, simp)
        done
      ultimately
      show is-real (to-complex (cross-ratio z ?Ma ?Mb ?Mc)) = (∃ H. {z, ?Ma, ?Mb, ?Mc} ⊆ circline-set H)
        using aa[rule-format, of d] *
        by auto
    qed
  next
    show ?P 0h 1h ∞h
    proof safe
      fix z
      assume distinct [z, 0h, 1h, ∞h]
      hence z ≠ ∞h
        by auto
      assume is-real (to-complex (cross-ratio z 0h 1h ∞h))
      hence is-real (to-complex z)
        by simp
      hence z ∈ circline-set x-axis
        using of-complex-to-complex[symmetric, OF ‹z ≠ ∞h›]
        using circline-set-x-axis
        by auto
      thus ∃ H. {z, 0h, 1h, ∞h} ⊆ circline-set H
        by (rule-tac x=x-axis in exI, auto)
    next
      fix z H
      assume *: distinct [z, 0h, 1h, ∞h] {z, 0h, 1h, ∞h} ⊆ circline-set H
      hence H = x-axis
        by auto
      hence z ∈ circline-set x-axis
        using *
        by auto
      hence is-real (to-complex z)
        using * circline-set-x-axis
        by auto
      thus is-real (to-complex (cross-ratio z 0h 1h ∞h))
        by simp
    qed
  next
    show u ≠ v v ≠ w u ≠ w
      using assms
      by auto
  qed
  thus ?thesis
    using assms
    by auto
qed
```

7.13 Symmetric points wrt. circline

In the extended complex plane there are no substantial differences between circles and lines, so we will consider only one kind of relation and call two points *circline symmetric* if they are mapped to one another using either reflection or inversion over arbitrary line or circle. Points are symmetric iff the bilinear form of their representation vectors and matrix is zero.

definition *circline-symmetric-cvec-cmat* :: *complex-vec* ⇒ *complex-vec* ⇒ *complex-mat* ⇒ *bool* **where**
[*simp*]: *circline-symmetric-cvec-cmat z1 z2 H* ⟷ *bilinear-form z1 z2 H = 0*
lift-definition *circline-symmetric-hcoords-clmat* :: *complex-homo-coords* ⇒ *complex-homo-coords* ⇒ *circline-mat* ⇒ *bool* **is** *circline-symmetric-cvec-cmat*
done
lift-definition *circline-symmetric* :: *complex-homo* ⇒ *complex-homo* ⇒ *circline* ⇒ *bool* **is** *circline-symmetric-hcoords-clmat*
apply *transfer*
apply (*simp del*: *bilinear-form-def*)
apply (*erule exE*)+
apply (*simp add*: *bilinear-form-scale-m bilinear-form-scale-v1 bilinear-form-scale-v2 del*: *vec-cnj-sv quad-form-def bilinear-form-def*)
done

lemma *symmetry-principle* [*simp*]:
assumes *circline-symmetric z1 z2 H*
shows *circline-symmetric* (*moebius-pt M z1*) (*moebius-pt M z2*) (*moebius-circline M H*)
using *assms*
by (*transfer*, *transfer*, *simp del*: *bilinear-form-def congruence-def*)

Symmetry wrt. *unit-circle*

lemma *circline-symmetric-0inf-disc* [*simp*]:
shows *circline-symmetric* 0_h ∞_h *unit-circle*
by (*transfer*, *transfer*, *simp add*: *vec-cnj-def*)

lemma *circline-symmetric-inv-homo-disc* [*simp*]:
shows *circline-symmetric a* (*inversion a*) *unit-circle*
unfolding *inversion-def*
by (*transfer*, *transfer*) (*case-tac a*, *auto simp add*: *vec-cnj-def*)

lemma *circline-symmetric-inv-homo-disc′*:
assumes *circline-symmetric a a′ unit-circle*
shows *a′ = inversion a*
unfolding *inversion-def*
using *assms*
proof (*transfer*, *transfer*)
fix *a a′*
assume *vz*: *a* ≠ *vec-zero a′* ≠ *vec-zero*
obtain *a1 a2* **where** *aa*: *a* = (*a1*, *a2*)
by (*cases a*, *auto*)
obtain *a1′ a2′* **where** *aa′*: *a′* = (*a1′*, *a2′*)
by (*cases a′*, *auto*)
assume ∗: *circline-symmetric-cvec-cmat a a′ unit-circle-cmat*
show *a′* ≈$_v$ (*conjugate-cvec* ∘ *reciprocal-cvec*) *a*
proof (*cases a1′ = 0*)
case *True*
thus *?thesis*
using *aa aa′ vz* ∗
by (*auto simp add*: *vec-cnj-def field-simps*)
next
case *False*
show *?thesis*
proof (*cases a2 = 0*)
case *True*
thus *?thesis*
using ‹*a1′* ≠ *0*›
using *aa aa′* ∗ *vz*
by (*simp add*: *vec-cnj-def field-simps*)
next
case *False*
thus *?thesis*

```
      using ‹a1' ≠ 0› aa aa' ∗
      by (simp add: vec-cnj-def field-simps) (rule-tac x=cnj a2 / a1' in exI, simp add: field-simps)
    qed
  qed
qed

lemma ex-moebius-circline-x-axis:
  assumes circline-type H < 0
  shows ∃ M. moebius-circline M H = x-axis
proof−
  obtain A B C where ∗: A ≠ B A ≠ C B ≠ C on-circline H A on-circline H B on-circline H C
    using circline-type-neg-card-gt3[OF assms]
    unfolding circline-set-def
    by auto
  then obtain M where moebius-pt M A = 0_h moebius-pt M B = 1_h moebius-pt M C = ∞_h
    using ex-moebius-01inf by blast
  hence moebius-circline M H = x-axis
    using ∗
    by (metis circline-set-I circline-set-moebius-circline rev-image-eqI unique-circline-01inf')
  thus ?thesis
    by blast
qed

lemma wlog-circline-x-axis:
  assumes circline-type H < 0
  assumes ⋀ M H. P H ⟹ P (moebius-circline M H)
  assumes P x-axis
  shows P H
proof−
  obtain M where moebius-circline M H = x-axis
    using ex-moebius-circline-x-axis[OF assms(1)]
    by blast
  then obtain M' where moebius-circline M' x-axis = H
    by (metis moebius-circline-comp-inv-left)
  thus ?thesis
    using assms(2)[of x-axis M'] assms(3)
    by simp
qed

lemma circline-intersection-at-most-2-points:
  assumes H1 ≠ H2
  shows finite (circline-intersection H1 H2) ∧ card (circline-intersection H1 H2) ≤ 2
proof (rule ccontr)
  assume ¬ ?thesis
  hence infinite (circline-intersection H1 H2) ∨ card (circline-intersection H1 H2) > 2
    by auto
  hence ∃ A B C. A ≠ B ∧ B ≠ C ∧ A ≠ C ∧ {A, B, C} ⊆ circline-intersection H1 H2
  proof
    assume card (circline-intersection H1 H2) > 2
    thus ?thesis
      using card-geq-3-iff-contains-3-elems[of circline-intersection H1 H2]
      by auto
  next
    assume infinite (circline-intersection H1 H2)
    thus ?thesis
      using infinite-contains-3-elems
      by blast
  qed
  then obtain A B C where A ≠ B B ≠ C A ≠ C {A, B, C} ⊆ circline-intersection H1 H2
    by blast
  hence H2 = H1
    using circline-intersection-def mem-Collect-eq unique-circline-set by fastforce
  thus False
    using assms
    by simp
qed
```

end

8 Oriented circlines

theory *Oriented-Circlines*
imports *Circlines*
begin

8.1 Oriented circlines definition

In this section we describe how the orientation is introduced for the circlines. Similarly as the set of circline points, the set of disc points is introduced using the quadratic form induced by the circline matrix — the set of points of the circline disc is the set of points such that satisfy that $A \cdot z \cdot \overline{z} + B \cdot \overline{z} + C \cdot z + D < 0$, where (A, B, C, D) is a circline matrix representative Hermitean matrix. As the set of disc points must be invariant to the choice of representative, it is clear that oriented circlines matrices are equivalent only if they are proportional by a positive real factor (recall that unoriented circline allowed arbitrary non-zero real factors).

definition *ocircline-eq-cmat* :: *complex-mat* ⇒ *complex-mat* ⇒ *bool* **where**
  [*simp*]: *ocircline-eq-cmat A B* ⟷(∃ *k*::*real*. *k* > *0* ∧ *B* = *cor k* ∗$_{sm}$ *A*)
lift-definition *ocircline-eq-clmat* :: *circline-mat* ⇒ *circline-mat* ⇒ *bool* **is** *ocircline-eq-cmat*
  done

lemma *ocircline-eq-cmat-id* [*simp*]:
  shows *ocircline-eq-cmat H H*
  by (*simp*, *rule-tac x=1* **in** *exI*, *simp*)

quotient-type *ocircline* = *circline-mat* / *ocircline-eq-clmat*
proof (*rule equivpI*)
  show *reflp ocircline-eq-clmat*
    unfolding *reflp-def*
    by *transfer* (*auto*, *rule-tac x=1* **in** *exI*, *simp*)
next
  show *symp ocircline-eq-clmat*
    unfolding *symp-def*
    by *transfer* (*simp only*: *ocircline-eq-cmat-def*, *safe*, *rule-tac x=1/k* **in** *exI*, *simp*)
next
  show *transp ocircline-eq-clmat*
    unfolding *transp-def*
    by *transfer* (*simp only*: *ocircline-eq-cmat-def*, *safe*, *rule-tac x=k∗ka* **in** *exI*, *simp*)
qed

8.2 Points on oriented circlines

Boundary of the circline.

lift-definition *on-ocircline* :: *ocircline* ⇒ *complex-homo* ⇒ *bool* **is** *on-circline-clmat-hcoords*
  by *transfer* (*simp del*: *quad-form-def*, (*erule exE*)+, *simp add*: *quad-form-scale-m quad-form-scale-v del*: *quad-form-def*)

definition *ocircline-set* :: *ocircline* ⇒ *complex-homo set* **where**
  ocircline-set H = {*z*. *on-ocircline H z*}

lemma *ocircline-set-I* [*simp*]:
  assumes *on-ocircline H z*
  shows *z* ∈ *ocircline-set H*
  using *assms*
  unfolding *ocircline-set-def*
  by *simp*

8.3 Disc and disc complement - in and out points

Interior and the exterior of an oriented circline.

definition *in-ocircline-cmat-cvec* :: *complex-mat* ⇒ *complex-vec* ⇒ *bool* **where**
  [*simp*]: *in-ocircline-cmat-cvec H z* ⟷ *Re* (*quad-form z H*) < *0*
lift-definition *in-ocircline-clmat-hcoords* :: *circline-mat* ⇒ *complex-homo-coords* ⇒ *bool* **is** *in-ocircline-cmat-cvec*

done
lift-definition *in-ocircline* :: *ocircline* ⇒ *complex-homo* ⇒ *bool* **is** *in-ocircline-clmat-hcoords*
proof *transfer*
fix *H H' z z'*
assume *hh*: *hermitean H* ∧ *H* ≠ *mat-zero* **and** *hermitean H'* ∧ *H'* ≠ *mat-zero* **and**
z ≠ *vec-zero* **and** *z'* ≠ *vec-zero*
assume *ocircline-eq-cmat H H'* **and** *z* ≈$_v$ *z'*
then obtain *k k'* **where**
∗: *0* < *k H'* = *cor k* ∗$_{sm}$ *H k'* ≠ *0 z'* = *k'* ∗$_{sv}$ *z*
by *auto*
hence *quad-form z' H'* = *cor k* ∗ *cor* ((*cmod k'*)^2) ∗ *quad-form z H*
by (*simp add*: *quad-form-scale-v quad-form-scale-m del*: *vec-cnj-sv quad-form-def*)
hence *Re* (*quad-form z' H'*) = *k* ∗ (*cmod k'*)^2 ∗ *Re* (*quad-form z H*)
using *hh quad-form-hermitean-real*[*of H*]
by (*simp add*: *power2-eq-square*)
thus *in-ocircline-cmat-cvec H z* = *in-ocircline-cmat-cvec H' z'*
using ‹*k* > *0*› ‹*k'* ≠ *0*›
using *mult-less-0-iff*
by *fastforce*
qed

definition *disc* :: *ocircline* ⇒ *complex-homo set* **where**
disc H = {*z*. *in-ocircline H z*}

lemma *disc-I* [*simp*]:
assumes *in-ocircline H z*
shows *z* ∈ *disc H*
using *assms*
unfolding *disc-def*
by *simp*

definition *out-ocircline-cmat-cvec* :: *complex-mat* ⇒ *complex-vec* ⇒ *bool* **where**
[*simp*]: *out-ocircline-cmat-cvec H z* ⟷ *Re* (*quad-form z H*) > *0*
lift-definition *out-ocircline-clmat-hcoords* :: *circline-mat* ⇒ *complex-homo-coords* ⇒ *bool* **is** *out-ocircline-cmat-cvec*
done
lift-definition *out-ocircline* :: *ocircline* ⇒ *complex-homo* ⇒ *bool* **is** *out-ocircline-clmat-hcoords*
proof *transfer*
fix *H H' z z'*
assume *hh*: *hermitean H* ∧ *H* ≠ *mat-zero hermitean H'* ∧ *H'* ≠ *mat-zero*
z ≠ *vec-zero z'* ≠ *vec-zero*
assume *ocircline-eq-cmat H H' z* ≈$_v$ *z'*
then obtain *k k'* **where**
∗: *0* < *k H'* = *cor k* ∗$_{sm}$ *H k'* ≠ *0 z'* = *k'* ∗$_{sv}$ *z*
by *auto*
hence *quad-form z' H'* = *cor k* ∗ *cor* ((*cmod k'*)^2) ∗ *quad-form z H*
by (*simp add*: *quad-form-scale-v quad-form-scale-m del*: *vec-cnj-sv quad-form-def*)
hence *Re* (*quad-form z' H'*) = *k* ∗ (*cmod k'*)^2 ∗ *Re* (*quad-form z H*)
using *hh quad-form-hermitean-real*[*of H*]
by (*simp add*: *power2-eq-square*)
thus *out-ocircline-cmat-cvec H z* = *out-ocircline-cmat-cvec H' z'*
using ‹*k* > *0*› ‹*k'* ≠ *0*›
using *zero-less-mult-pos*
by *fastforce*
qed

definition *disc-compl* :: *ocircline* ⇒ *complex-homo set* **where**
disc-compl H = {*z*. *out-ocircline H z*}

These three sets are mutually disjoint and they fill up the entire plane.

lemma *disc-compl-I* [*simp*]:
assumes *out-ocircline H z*
shows *z* ∈ *disc-compl H*
using *assms*
unfolding *disc-compl-def*
by *simp*

lemma *in-on-out*:
shows *in-ocircline H z* ∨ *on-ocircline H z* ∨ *out-ocircline H z*
apply (*transfer*, *transfer*)
using *quad-form-hermitean-real*
using *complex-eq-if-Re-eq*
by *auto*

lemma *in-on-out-univ*:
shows *disc H* ∪ *disc-compl H* ∪ *ocircline-set H* = *UNIV*
unfolding *disc-def disc-compl-def ocircline-set-def*
using *in-on-out*[*of H*]
by *auto*

lemma *disc-inter-disc-compl* [*simp*]:
shows *disc H* ∩ *disc-compl H* = {}
unfolding *disc-def disc-compl-def*
by *auto* (*transfer*, *transfer*, *simp*)

lemma *disc-inter-ocircline-set* [*simp*]:
shows *disc H* ∩ *ocircline-set H* = {}
unfolding *disc-def ocircline-set-def*
by *auto* (*transfer*, *transfer*, *simp*)

lemma *disc-compl-inter-ocircline-set* [*simp*]:
shows *disc-compl H* ∩ *ocircline-set H* = {}
unfolding *disc-compl-def ocircline-set-def*
by *auto* (*transfer*, *transfer*, *simp*)

8.4 Opposite orientation

Finding opposite circline is idempotent, and opposite circlines share the same set of points, but exchange disc and its complement.

definition *opposite-ocircline-cmat* :: *complex-mat* ⇒ *complex-mat* **where**
[*simp*]: *opposite-ocircline-cmat H* = (−*1*) $*_{sm}$ *H*
lift-definition *opposite-ocircline-clmat* :: *circline-mat* ⇒ *circline-mat* **is** *opposite-ocircline-cmat*
by (*auto simp add*: *hermitean-def mat-adj-def mat-cnj-def*)
lift-definition *opposite-ocircline* :: *ocircline* ⇒ *ocircline* **is** *opposite-ocircline-clmat*
by *transfer auto*

lemma *opposite-ocircline-involution* [*simp*]:
shows *opposite-ocircline* (*opposite-ocircline H*) = *H*
by (*transfer*, *transfer*) (*auto*, *rule-tac x=1* **in** *exI*, *simp*)

lemma *on-circline-opposite-ocircline-cmat* [*simp*]:
assumes *hermitean H* ∧ *H* ≠ *mat-zero* **and** *z* ≠ *vec-zero*
shows *on-circline-cmat-cvec* (*opposite-ocircline-cmat H*) *z* = *on-circline-cmat-cvec H z*
using *assms*
by (*simp add*: *quad-form-scale-m del*: *quad-form-def*)

lemma *on-circline-opposite-ocircline* [*simp*]:
shows *on-ocircline* (*opposite-ocircline H*) *z* ⟷ *on-ocircline H z*
using *on-circline-opposite-ocircline-cmat*
by (*transfer*, *transfer*, *simp*)

lemma *ocircline-set-opposite-ocircline* [*simp*]:
shows *ocircline-set* (*opposite-ocircline H*) = *ocircline-set H*
unfolding *ocircline-set-def*
by *auto*

lemma *disc-compl-opposite-ocircline* [*simp*]:
shows *disc-compl* (*opposite-ocircline H*) = *disc H*
unfolding *disc-def disc-compl-def*
apply *auto*
apply (*transfer*, *transfer*)
apply (*auto simp add*: *quad-form-scale-m simp del*: *quad-form-def*)

apply (*transfer* ,*transfer*)
apply (*auto simp add*: *quad-form-scale-m simp del*: *quad-form-def*)
done

lemma *disc-opposite-ocircline* [*simp*]:
shows *disc* (*opposite-ocircline H*) = *disc-compl H*
using *disc-compl-opposite-ocircline*[*of opposite-ocircline H*]
by *simp*

8.5 Positive orientation. Conversion between unoriented and oriented circlines

Given an oriented circline, one can trivially obtain its unoriented counterpart, and these two share the same set of points.

lift-definition *of-ocircline* :: *ocircline* ⇒ *circline* **is** *id*::*circline-mat* ⇒ *circline-mat*
by *transfer* (*simp*, *erule exE*, *force*)

lemma *of-ocircline-opposite-ocircline* [*simp*]:
shows *of-ocircline* (*opposite-ocircline H*) = *of-ocircline H*
by (*transfer*, *transfer*) (*simp*, *erule exE*, *rule-tac x*=−*1* **in** *exI*, *simp*)

lemma *on-ocircline-of-circline* [*simp*]:
shows *on-circline* (*of-ocircline H*) *z* ⟷ *on-ocircline H z*
by (*transfer*, *transfer*, *simp*)

lemma *circline-set-of-ocircline* [*simp*]:
shows *circline-set* (*of-ocircline H*) = *ocircline-set H*
unfolding *ocircline-set-def circline-set-def*
by (*safe*) (*transfer*, *simp*)+

lemma *inj-of-ocircline*:
assumes *of-ocircline H* = *of-ocircline H′*
shows *H* = *H′* ∨ *H* = *opposite-ocircline H′*
using *assms*
by (*transfer*, *transfer*) (*simp*, *metis linorder-neqE-linordered-idom minus-of-real-eq-of-real-iff mult-minus1 mult-sm-distribution neg-0-equal-iff-equal neg-less-0-iff-less*)

lemma *inj-ocircline-set*:
assumes *ocircline-set H* = *ocircline-set H′* **and** *ocircline-set H* ≠ {}
shows *H* = *H′* ∨ *H* = *opposite-ocircline H′*
proof−
from *assms*
have *circline-set* (*of-ocircline H*) = *circline-set* (*of-ocircline H′*)
circline-set (*of-ocircline H′*) ≠ {}
by *auto*
hence *of-ocircline H* = *of-ocircline H′*
by (*simp add*: *inj-circline-set*)
thus *?thesis*
by (*rule inj-of-ocircline*)
qed

Positive orientation.

Given a representative Hermitean matrix of a circline, it represents exactly one of the two possible oriented circlines. The choice of what should be called a positive orientation is arbitrary. We follow Schwerdtfeger [13], use the leading coefficient A as the first criterion, and say that circline matrices with $A > 0$ are called positively oriented, and with $A < 0$ negatively oriented. However, Schwerdtfeger did not discuss the possible case of $A = 0$ (the case of lines), so we had to extend his definition to achieve a total characterization.

definition *pos-oriented-cmat* :: *complex-mat* ⇒ *bool* **where**
[*simp*]: *pos-oriented-cmat H* ⟷
(*let* (*A*, *B*, *C*, *D*) = *H*
in (*Re A* > *0* ∨ (*Re A* = *0* ∧ ((*B* ≠ *0* ∧ *Arg B* > *0*) ∨ (*B* = *0* ∧ *Re D* > *0*)))))
lift-definition *pos-oriented-clmat* :: *circline-mat* ⇒ *bool* **is** *pos-oriented-cmat*
done

lift-definition *pos-oriented* :: *ocircline* ⇒ *bool* **is** *pos-oriented-clmat*

by *transfer*
(*case-tac circline-mat1*, *case-tac circline-mat2*, *simp*, *erule exE*, *simp*,
metis mult-pos-pos zero-less-mult-pos)

lemma *pos-oriented*:
shows *pos-oriented H* ∨ *pos-oriented* (*opposite-ocircline H*)
proof (*transfer*, *transfer*)
fix *H*
assume *hh*: *hermitean H* ∧ *H* ≠ *mat-zero*
obtain *A B C D* **where** *HH*: *H* = (*A*, *B*, *C*, *D*)
by (*cases H*) *auto*
moreover
hence *Re A* = *0* ∧ *Re D* = *0* ⟶ *B* ≠ *0*
using *hh hermitean-elems*[*of A B C D*]
by (*cases A*, *cases D*) (*auto simp add*: *Complex-eq*)
moreover
have *B* ≠ *0* ∧ ¬ *0* < *Arg B* ⟶ *0* < *Arg* (− *B*)
using *canon-ang-plus-pi2*[*of Arg B*] *Arg-bounded*[*of B*]
by (*auto simp add*: *arg-uminus*)
ultimately
show *pos-oriented-cmat H* ∨ *pos-oriented-cmat* (*opposite-ocircline-cmat H*)
by *auto*
qed

lemma *pos-oriented-opposite-ocircline-cmat* [*simp*]:
assumes *hermitean H* ∧ *H* ≠ *mat-zero*
shows *pos-oriented-cmat* (*opposite-ocircline-cmat H*) ⟷ ¬ *pos-oriented-cmat H*
proof−
obtain *A B C D* **where** *HH*: *H* = (*A*, *B*, *C*, *D*)
by (*cases H*) *auto*
moreover
hence *Re A* = *0* ∧ *Re D* = *0* ⟶ *B* ≠ *0*
using *assms hermitean-elems*[*of A B C D*]
by (*cases A*, *cases D*) (*auto simp add*: *Complex-eq*)
moreover
have *B* ≠ *0* ∧ ¬ *0* < *Arg B* ⟶ *0* < *Arg* (− *B*)
using *canon-ang-plus-pi2*[*of Arg B*] *Arg-bounded*[*of B*]
by (*auto simp add*: *arg-uminus*)
moreover
have *B* ≠ *0* ∧ *0* < *Arg B* ⟶ ¬ *0* < *Arg* (− *B*)
using *canon-ang-plus-pi1*[*of Arg B*] *Arg-bounded*[*of B*]
by (*auto simp add*: *arg-uminus*)
ultimately
show *pos-oriented-cmat* (*opposite-ocircline-cmat H*) = (¬ *pos-oriented-cmat H*)
by *simp* (*metis not-less-iff-gr-or-eq*)
qed

lemma *pos-oriented-opposite-ocircline* [*simp*]:
shows *pos-oriented* (*opposite-ocircline H*) ⟷ ¬ *pos-oriented H*
using *pos-oriented-opposite-ocircline-cmat*
by (*transfer*, *transfer*, *simp*)

lemma *pos-oriented-circle-inf*:
assumes ∞$_h$ ∉ *ocircline-set H*
shows *pos-oriented H* ⟷ ∞$_h$ ∉ *disc H*
using *assms*
unfolding *ocircline-set-def disc-def*
apply *simp*
proof (*transfer*, *transfer*)
fix *H*
assume *hh*: *hermitean H* ∧ *H* ≠ *mat-zero*
obtain *A B C D* **where** *HH*: *H* = (*A*, *B*, *C*, *D*)
by (*cases H*) *auto*
hence *is-real A*
using *hh hermitean-elems*
by *auto*

assume ¬ *on-circline-cmat-cvec H* ∞_v
thus *pos-oriented-cmat H* = (¬ *in-ocircline-cmat-cvec H* ∞_v)
using *HH* ‹*is-real A*›
by (*cases A*) (*auto simp add*: *vec-cnj-def Complex-eq*)
qed

lemma *pos-oriented-euclidean-circle*:
assumes *is-circle* (*of-ocircline H*)
(*a*, *r*) = *euclidean-circle* (*of-ocircline H*)
circline-type (*of-ocircline H*) < *0*
shows *pos-oriented H* ⟷ *of-complex a* ∈ *disc H*
using *assms*
unfolding *disc-def*
apply *simp*
proof (*transfer*, *transfer*)
fix *H a r*
assume *hh*: *hermitean H* ∧ *H* ≠ *mat-zero*
obtain *A B C D* **where** *HH*: *H* = (*A*, *B*, *C*, *D*)
by (*cases H*) *auto*
hence *is-real A is-real D C* = *cnj B*
using *hh hermitean-elems*
by *auto*

assume ∗: ¬ *circline-A0-cmat* (*id H*) (*a*, *r*) = *euclidean-circle-cmat* (*id H*) *circline-type-cmat* (*id H*) < *0*
hence *A* ≠ *0 Re A* ≠ *0*
using *HH* ‹*is-real A*›
by (*case-tac*[!] *A*) (*auto simp add*: *Complex-eq*)

have *Re* (*A*∗*D* − *B*∗*C*) < *0*
using ‹*circline-type-cmat* (*id H*) < *0*› *HH*
by *simp*

have ∗∗: (*A* ∗ (*D* ∗ *cnj A*) − *B* ∗ (*C* ∗ *cnj A*)) / (*A* ∗ *cnj A*) = (*A*∗*D* − *B*∗*C*) / *A*
using ‹*A* ≠ *0*›
by (*simp add*: *field-simps*)
hence ∗∗∗: *0* < *Re A* ⟷ *Re* ((*A* ∗ (*D* ∗ *cnj A*) − *B* ∗ (*C* ∗ *cnj A*)) / (*A* ∗ *cnj A*)) < *0*
using ‹*is-real A*› ‹*A* ≠ *0*› ‹*Re* (*A*∗*D* − *B*∗*C*) < *0*›
by (*simp add*: *Re-divide-real divide-less-0-iff*)

have *Re D* − *Re* (*cnj B* ∗ *B* / *cnj A*) < *Re* ((*C* − *cnj B* ∗ *A* / *cnj A*) ∗ *B* / *A*) **if** *Re A* > *0*
using *HH* ∗ ‹*is-real A*› *that*
by *simp*
(*metis* ∗∗ ∗∗∗ ‹*C* = *cnj B*› *cancel-comm-monoid-add-class.diff-cancel diff-divide-distrib eq-cnj-iff-real*
minus-complex.simps(*1*) *mult.commute mult-eq-0-iff nonzero-mult-div-cancel-right*)
moreover have *Re A* > *0* **if** *Re D* − *Re* (*cnj B* ∗ *B* / *cnj A*) < *Re* ((*C* − *cnj B* ∗ *A* / *cnj A*) ∗ *B* / *A*)
using *HH* ∗ ‹*is-real A*› *that*
by *simp*
(*metis* ∗∗ ∗∗∗ ‹*C* = *cnj B*› *cancel-comm-monoid-add-class.diff-cancel diff-divide-distrib eq-cnj-iff-real*
minus-complex.simps(*1*) *mult.commute mult-eq-0-iff nonzero-mult-div-cancel-right*)
ultimately show *pos-oriented-cmat H* = *in-ocircline-cmat-cvec H* (*of-complex-cvec a*)
using *HH* ‹*Re A* ≠ *0*› ∗ ‹*is-real A*› **by** (*auto simp add*: *vec-cnj-def*)
qed

Introduce positive orientation

definition *of-circline-cmat* :: *complex-mat* ⇒ *complex-mat* **where**
[*simp*]: *of-circline-cmat H* = (*if pos-oriented-cmat H then H else opposite-ocircline-cmat H*)

lift-definition *of-circline-clmat* :: *circline-mat* ⇒ *circline-mat* **is** *of-circline-cmat*
by (*auto simp add*: *hermitean-def mat-adj-def mat-cnj-def*)

lemma *of-circline-clmat-def′*:
shows *of-circline-clmat H* = (*if pos-oriented-clmat H then H else opposite-ocircline-clmat H*)
by *transfer simp*

lemma *pos-oriented-cmat-mult-positive′*:
assumes

```
    hermitean H1 ∧ H1 ≠ mat-zero and
    hermitean H2 ∧ H2 ≠ mat-zero and
    ∃k. k > 0 ∧ H2 = cor k *sm H1 and
    pos-oriented-cmat H1
  shows pos-oriented-cmat H2
proof−
  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)
  thus ?thesis
    using assms
    by fastforce
qed

lemma pos-oriented-cmat-mult-positive:
  assumes
    hermitean H1 ∧ H1 ≠ mat-zero and
    hermitean H2 ∧ H2 ≠ mat-zero and
    ∃k. k > 0 ∧ H2 = cor k *sm H1
  shows
    pos-oriented-cmat H1 ⟷ pos-oriented-cmat H2
proof−
  from assms(3) obtain k where k > 0 ∧ H2 = cor k *sm H1
    by auto
  hence ∃k. k > 0 ∧ H1 = cor k *sm H2
    by (rule-tac x=1/k in exI, auto)
  thus ?thesis
    using assms pos-oriented-cmat-mult-positive'
    by blast
qed


lemma pos-oriented-cmat-mult-negative:
  assumes
    hermitean H1 ∧ H1 ≠ mat-zero and
    hermitean H2 ∧ H2 ≠ mat-zero and
    ∃k. k < 0 ∧ H2 = cor k *sm H1
  shows
    pos-oriented-cmat H1 ⟷ ¬ pos-oriented-cmat H2
  using assms
proof−
  obtain A B C D A1 B1 C1 D1
    where *: H1 = (A, B, C, D) H2 = (A1, B1, C1, D1)
    by (cases H1, cases H2) auto
  hence **: is-real A is-real D is-real A1 is-real D1 B = 0 ⟷ C = 0 B1 = 0 ⟷ C1 = 0
    using assms hermitean-elems[of A B C D] hermitean-elems[of A1 B1 C1 D1]
    by auto
  show ?thesis
  proof (rule iffI)
    assume H1: pos-oriented-cmat H1
    show ¬ pos-oriented-cmat H2
    proof (cases Re A > 0)
      case True
      thus ?thesis
        using assms * ** mult-neg-pos
        by fastforce
    next
      case False
      show ?thesis
      proof (cases B = 0)
        case True
        thus ?thesis
          using assms * ** H1 ‹¬ Re A > 0› mult-neg-pos
          by fastforce
      next
        case False
```

```
      thus ?thesis
        using arg-uminus-opposite-sign[of B] arg-mult-real-negative
        using assms * ** H1 ‹¬ Re A > 0› mult-neg-pos
        by fastforce
      qed
    qed
  next
    assume H2: ¬ pos-oriented-cmat H2
    show pos-oriented-cmat H1
    proof (cases Re A > 0)
      case True
      thus ?thesis
        using * ** mult-neg-pos
        by fastforce
    next
      case False
      show ?thesis
      proof (cases B = 0)
        case True
        thus ?thesis
          using assms * ** H2 ‹¬ Re A > 0›
          by simp
            (smt (verit, del-insts) Im-complex-of-real Re-complex-of-real Re-mult-real arg-0-iff arg-pi-iff is-real-arg2
              mult-less-0-iff mult-minus-right)
      next
        case False
        thus ?thesis
          using assms ‹¬ Re A > 0› H2 * **
          using arg-uminus-opposite-sign[of B]
          by (cases Re A = 0, auto simp add: mult-neg-neg)
      qed
    qed
  qed
qed

lift-definition of-circline :: circline ⇒ ocircline is of-circline-clmat
proof transfer
  fix H1 H2
  assume hh:
    hermitean H1 ∧ H1 ≠ mat-zero
    hermitean H2 ∧ H2 ≠ mat-zero
  assume circline-eq-cmat H1 H2
  then obtain k where *: k ≠ 0 ∧ H2 = cor k *sm H1
    by auto
  show ocircline-eq-cmat (of-circline-cmat H1) (of-circline-cmat H2)
  proof (cases k > 0)
    case True
    hence pos-oriented-cmat H1 = pos-oriented-cmat H2
      using * pos-oriented-cmat-mult-positive[OF hh]
      by blast
    thus ?thesis
      using hh * ‹k > 0›
      apply (simp del: pos-oriented-cmat-def)
      apply (rule conjI)
       apply (rule impI)
       apply (simp, rule-tac x=k in exI, simp)
      apply (rule impI)
      apply (simp, rule-tac x=k in exI, simp)
      done
  next
    case False
    hence k < 0
      using *
      by simp
    hence pos-oriented-cmat H1 ⟷ ¬ (pos-oriented-cmat H2)
      using * pos-oriented-cmat-mult-negative[OF hh]
```

```
    by blast
   thus ?thesis
    using hh ∗ ‹k < 0›
    apply (simp del: pos-oriented-cmat-def)
    apply (rule conjI)
     apply (rule impI)
     apply (simp, rule-tac x=−k in exI, simp)
    apply (rule impI)
    apply (simp, rule-tac x=−k in exI, simp)
    done
  qed
qed

lemma pos-oriented-of-circline [simp]:
  shows pos-oriented (of-circline H)
  using pos-oriented-opposite-ocircline-cmat
  by (transfer, transfer, simp)

lemma of-ocircline-of-circline [simp]:
  shows of-ocircline (of-circline H) = H
  apply (transfer, auto simp add: of-circline-clmat-def')
  apply (transfer, simp, rule-tac x=−1 in exI, simp)
  done

lemma of-circline-of-ocircline-pos-oriented [simp]:
  assumes pos-oriented H
  shows of-circline (of-ocircline H) = H
  using assms
  by (transfer, transfer, simp, rule-tac x=1 in exI, simp)

lemma inj-of-circline:
  assumes of-circline H = of-circline H'
  shows H = H'
  using assms
proof (transfer, transfer)
  fix H H'
  assume ocircline-eq-cmat (of-circline-cmat H) (of-circline-cmat H')
  then obtain k where k > 0 of-circline-cmat H' = cor k ∗sm of-circline-cmat H
    by auto
  thus circline-eq-cmat H H'
    using mult-sm-inv-l[of −1 H' cor k ∗sm H]
    using mult-sm-inv-l[of −1 H' (− (cor k)) ∗sm H]
    apply (simp split: if-split-asm)
    apply (rule-tac x=k in exI, simp)
    apply (rule-tac x=−k in exI, simp)
    apply (rule-tac x=−k in exI, simp)
    apply (rule-tac x=k in exI, simp)
    done
qed

lemma of-circline-of-ocircline:
  shows of-circline (of-ocircline H') = H' ∨
        of-circline (of-ocircline H') = opposite-ocircline H'
proof (cases pos-oriented H')
  case True
  thus ?thesis
    by auto
next
  case False
  hence pos-oriented (opposite-ocircline H')
    using pos-oriented
    by auto
  thus ?thesis
    using of-ocircline-opposite-ocircline[of H']
    using of-circline-of-ocircline-pos-oriented [of opposite-ocircline H']
    by auto
```

qed

8.5.1 Set of points on oriented and unoriented circlines

lemma *ocircline-set-of-circline* [*simp*]:
shows *ocircline-set* (*of-circline H*) = *circline-set H*
unfolding *ocircline-set-def circline-set-def*
proof (*safe*)
fix *z*
assume *on-ocircline* (*of-circline H*) *z*
thus *on-circline H z*
by (*transfer*, *transfer*, *simp del*: *on-circline-cmat-cvec-def opposite-ocircline-cmat-def split*: *if-split-asm*)
next
fix *z*
assume *on-circline H z*
thus *on-ocircline* (*of-circline H*) *z*
by (*transfer*, *transfer*, *simp del*: *on-circline-cmat-cvec-def opposite-ocircline-cmat-def split*: *if-split-asm*)
qed

8.6 Some special oriented circlines and discs

lift-definition *mk-ocircline* :: *complex* ⇒ *complex* ⇒ *complex* ⇒ *complex* ⇒ *ocircline* **is** *mk-circline-clmat*
done

oriented unit circle and unit disc

lift-definition *ounit-circle* :: *ocircline* **is** *unit-circle-clmat*
done

lemma *pos-oriented-ounit-circle* [*simp*]:
shows *pos-oriented ounit-circle*
by (*transfer*, *transfer*, *simp*)

lemma *of-ocircline-ounit-circle* [*simp*]:
shows *of-ocircline ounit-circle* = *unit-circle*
by (*transfer*, *transfer*, *simp*)

lemma *of-circline-unit-circle* [*simp*]:
shows *of-circline* (*unit-circle*) = *ounit-circle*
by (*transfer*, *transfer*, *simp*)

lemma *ocircline-set-ounit-circle* [*simp*]:
shows *ocircline-set ounit-circle* = *circline-set unit-circle*
apply (*subst of-circline-unit-circle*[*symmetric*])
apply (*subst ocircline-set-of-circline*)
apply *simp*
done

definition *unit-disc* :: *complex-homo set* **where**
unit-disc = *disc ounit-circle*

definition *unit-disc-compl* :: *complex-homo set* **where**
unit-disc-compl = *disc-compl ounit-circle*

definition *unit-circle-set* :: *complex-homo set* **where**
unit-circle-set = *circline-set unit-circle*

lemma *zero-in-unit-disc* [*simp*]:
shows $0_h \in$ *unit-disc*
unfolding *unit-disc-def disc-def*
by (*simp*, *transfer*, *transfer*) (*simp add*: *Let-def vec-cnj-def*)

lemma *one-notin-unit-dic* [*simp*]:
shows $1_h \notin$ *unit-disc*
unfolding *unit-disc-def disc-def*
by (*simp*, *transfer*, *transfer*) (*simp add*: *Let-def vec-cnj-def*)

lemma *inf-notin-unit-disc* [*simp*]:
shows $\infty_h \notin$ *unit-disc*
unfolding *unit-disc-def disc-def*
by (*simp*, *transfer*, *transfer*) (*simp add*: *Let-def vec-cnj-def*)

lemma *unit-disc-iff-cmod-lt-1* [*simp*]:
shows *of-complex c* $\in$ *unit-disc* $\longleftrightarrow$ *cmod c* $<$ *1*
unfolding *unit-disc-def disc-def*
by (*simp*, *transfer*, *transfer*, *simp add*: *vec-cnj-def cmod-def power2-eq-square*)

lemma *unit-disc-cmod-square-lt-1* [*simp*]:
assumes *z* $\in$ *unit-disc*
shows (*cmod* (*to-complex z*))^2 $<$ *1*
using *assms inf-or-of-complex*[*of z*]
by (*auto simp add*: *abs-square-less-1*)

lemma *unit-disc-to-complex-inj*:
assumes *u* $\in$ *unit-disc* **and** *v* $\in$ *unit-disc*
assumes *to-complex u* $=$ *to-complex v*
shows *u* $=$ *v*
using *assms*
using *inf-or-of-complex*[*of u*] *inf-or-of-complex*[*of v*]
by *auto*

lemma *inversion-unit-disc* [*simp*]:
shows *inversion* ‘ *unit-disc* $=$ *unit-disc-compl*
unfolding *unit-disc-def unit-disc-compl-def disc-def disc-compl-def*
proof *safe*
fix *x*
assume *in-ocircline ounit-circle x*
thus *out-ocircline ounit-circle* (*inversion x*)
unfolding *inversion-def*
by (*transfer*, *transfer*, *auto simp add*: *vec-cnj-def*)
next
fix *x*
assume $*$: *out-ocircline ounit-circle x*
show *x* $\in$ *inversion* ‘ *Collect* (*in-ocircline ounit-circle*)
proof (*rule image-eqI*)
show *x* $=$ *inversion* (*inversion x*)
by *auto*
next
show *inversion x* $\in$ *Collect* (*in-ocircline ounit-circle*)
using $*$
unfolding *inversion-def*
by (*simp*, *transfer*, *transfer*, *auto simp add*: *vec-cnj-def*)
qed
qed

lemma *inversion-unit-disc-compl* [*simp*]:
shows *inversion* ‘ *unit-disc-compl* $=$ *unit-disc*
proof−
have *inversion* ‘ (*inversion* ‘ *unit-disc*) $=$ *unit-disc*
by (*auto simp del*: *inversion-unit-disc simp add*: *image-iff*)
thus *?thesis*
by *simp*
qed

lemma *inversion-noteq-unit-disc*:
assumes *u* $\in$ *unit-disc* **and** *v* $\in$ *unit-disc*
shows *inversion u* $\neq$ *v*
proof−
from *assms*
have *inversion u* $\in$ *unit-disc-compl*
by (*metis image-eqI inversion-unit-disc*)
thus *?thesis*
using *assms*

unfolding *unit-disc-def unit-disc-compl-def*
using *disc-inter-disc-compl*
by *fastforce*
qed

lemma *in-ocircline-ounit-circle-conjugate* [*simp*]:
assumes *in-ocircline ounit-circle z*
shows *in-ocircline ounit-circle* (*conjugate z*)
using *assms*
by (*transfer*, *transfer*, *auto simp add*: *vec-cnj-def*)

lemma *conjugate-unit-disc* [*simp*]:
shows *conjugate ‘ unit-disc = unit-disc*
unfolding *unit-disc-def disc-def*
apply (*auto simp add*: *image-iff*)
apply (*rule-tac x=conjugate x* **in** *exI*, *simp*)
done

lemma *conjugate-in-unit-disc* [*simp*]:
assumes *z* ∈ *unit-disc*
shows *conjugate z* ∈ *unit-disc*
using *conjugate-unit-disc*
using *assms*
by *blast*

lemma *out-ocircline-ounit-circle-conjugate* [*simp*]:
assumes *out-ocircline ounit-circle z*
shows *out-ocircline ounit-circle* (*conjugate z*)
using *assms*
by (*transfer*, *transfer*, *auto simp add*: *vec-cnj-def*)

lemma *conjugate-unit-disc-compl* [*simp*]:
shows *conjugate ‘ unit-disc-compl = unit-disc-compl*
unfolding *unit-disc-compl-def disc-compl-def*
apply (*auto simp add*: *image-iff*)
apply (*rule-tac x=conjugate x* **in** *exI*, *simp*)
done

lemma *conjugate-in-unit-disc-compl* [*simp*]:
assumes *z* ∈ *unit-disc-compl*
shows *conjugate z* ∈ *unit-disc-compl*
using *conjugate-unit-disc-compl*
using *assms*
by *blast*

8.6.1 Oriented x axis and lower half plane

lift-definition *o-x-axis* :: *ocircline* **is** *x-axis-clmat*
done

lemma *o-x-axis-pos-oriented* [*simp*]:
shows *pos-oriented o-x-axis*
by (*transfer*, *transfer*, *simp*)

lemma *of-ocircline-o-x-axis* [*simp*]:
shows *of-ocircline o-x-axis = x-axis*
by (*transfer*, *transfer*, *simp*)

lemma *of-circline-x-axis* [*simp*]:
shows *of-circline x-axis = o-x-axis*
using *of-circline-of-ocircline-pos-oriented*[*of o-x-axis*]
using *o-x-axis-pos-oriented*
by *simp*

lemma *ocircline-set-circline-set-x-axis* [*simp*]:
shows *ocircline-set o-x-axis = circline-set x-axis*

by (*subst of-circline-x-axis*[*symmetric*], *subst ocircline-set-of-circline*, *simp*)

lemma *ii-in-disc-o-x-axis* [*simp*]:
shows $ii_h \notin$ *disc o-x-axis*
unfolding *disc-def*
by *simp* (*transfer*, *transfer*, *simp add*: *Let-def vec-cnj-def*)

lemma *ii-notin-disc-o-x-axis* [*simp*]:
shows $ii_h \in$ *disc-compl o-x-axis*
unfolding *disc-compl-def*
by *simp* (*transfer*, *transfer*, *simp add*: *Let-def vec-cnj-def*)

lemma *of-complex-in-o-x-axis-disc* [*simp*]:
shows *of-complex z* $\in$ *disc o-x-axis* $\longleftrightarrow$ *Im z* $<$ *0*
unfolding *disc-def*
by *auto* (*transfer*, *transfer*, *simp add*: *vec-cnj-def*)+

lemma *inf-notin-disc-o-x-axis* [*simp*]:
shows $\infty_h \notin$ *disc o-x-axis*
unfolding *disc-def*
by *simp* (*transfer*, *transfer*, *simp add*: *vec-cnj-def*)

lemma *disc-o-x-axis*:
shows *disc o-x-axis* = *of-complex* ‘ {*z*. *Im z* $<$ *0*}
proof−
{
fix *z*
assume *z* $\in$ *disc o-x-axis*
hence $\exists$ *x*. *Im x* $<$ *0* $\wedge$ *z* = *of-complex x*
using *inf-or-of-complex*[*of z*]
by *auto*
}
thus *?thesis*
by (*auto simp add*: *image-iff*)
qed

8.6.2 Oriented single point circline

lift-definition *o-circline-point-0* :: *ocircline* **is** *circline-point-0-clmat*
done

lemma *of-ocircline-o-circline-point-0* [*simp*]:
shows *of-ocircline o-circline-point-0* = *circline-point-0*
by (*transfer*, *transfer*, *simp*)

8.7 Möbius action on oriented circlines and discs

Möbius action on an oriented circline is the same as on to an unoriented circline.

lift-definition *moebius-ocircline* :: *moebius* ⇒ *ocircline* ⇒ *ocircline* **is** *moebius-circline-mmat-clmat*
apply (*transfer*, *transfer*)
apply *simp*
apply ((*erule exE*)+, (*erule conjE*)+)
apply (*simp add*: *mat-inv-mult-sm*)
apply (*rule-tac x*=*ka* / *Re* (*k* ∗ *cnj k*) **in** *exI*, *auto simp add*: *complex-mult-cnj-cmod power2-eq-square*)
done

Möbius action on (unoriented) circlines could have been defined using the action on oriented circlines, but not the other way around.

lemma *moebius-circline-ocircline*:
shows *moebius-circline M H* = *of-ocircline* (*moebius-ocircline M* (*of-circline H*))
apply (*transfer*, *simp add*: *of-circline-clmat-def*′, *safe*)
apply (*transfer*, *simp*, *rule-tac x*=−*1* **in** *exI*, *simp*)
done

lemma *moebius-ocircline-circline*:

shows *moebius-ocircline M H = of-circline (moebius-circline M (of-ocircline H)) ∨*
moebius-ocircline M H = opposite-ocircline (of-circline (moebius-circline M (of-ocircline H)))
apply (*transfer*, *simp add*: *of-circline-clmat-def'*, *safe*)
apply (*transfer*, *simp*, *rule-tac x=1* **in** *exI*, *simp*)
apply (*transfer*, *simp*, *erule-tac x=1* **in** *allE*, *simp*)
done

Möbius action on oriented circlines have many nice properties as it was the case with Möbius action on (unoriented) circlines. These transformations are injective and form group under composition.

lemma *inj-moebius-ocircline* [*simp*]:
shows *inj (moebius-ocircline M)*
unfolding *inj-on-def*
proof (*safe*)
fix *H H'*
assume *moebius-ocircline M H = moebius-ocircline M H'*
thus *H = H'*
proof (*transfer*, *transfer*)
fix *M H H' :: complex-mat*
assume *mat-det M ≠ 0*
let *?iM = mat-inv M*
assume *ocircline-eq-cmat (moebius-circline-cmat-cmat M H) (moebius-circline-cmat-cmat M H')*
then obtain *k* **where** *congruence ?iM H' = congruence ?iM (cor k ∗sm H) k > 0*
by (*auto simp del*: *congruence-def*)
thus *ocircline-eq-cmat H H'*
using ‹*mat-det M ≠ 0*› *inj-congruence*[*of ?iM H' cor k ∗sm H*] *mat-det-inv*[*of M*]
by *auto*
qed
qed

lemma *moebius-ocircline-id-moebius* [*simp*]:
shows *moebius-ocircline id-moebius H = H*
by (*transfer*, *transfer*) (*force simp add*: *mat-adj-def mat-cnj-def*)

lemma *moebius-ocircline-comp* [*simp*]:
shows *moebius-ocircline (moebius-comp M1 M2) H = moebius-ocircline M1 (moebius-ocircline M2 H)*
by (*transfer*, *transfer*, *simp*, *rule-tac x=1* **in** *exI*, *simp add*: *mat-inv-mult-mm mult-mm-assoc*)

lemma *moebius-ocircline-comp-inv-left* [*simp*]:
shows *moebius-ocircline (moebius-inv M) (moebius-ocircline M H) = H*
by (*subst moebius-ocircline-comp*[*symmetric*]) *simp*

lemma *moebius-ocircline-comp-inv-right* [*simp*]:
shows *moebius-ocircline M (moebius-ocircline (moebius-inv M) H) = H*
by (*subst moebius-ocircline-comp*[*symmetric*]) *simp*

lemma *moebius-ocircline-opposite-ocircline* [*simp*]:
shows *moebius-ocircline M (opposite-ocircline H) = opposite-ocircline (moebius-ocircline M H)*
by (*transfer*, *transfer*, *simp*, *rule-tac x=1* **in** *exI*, *simp*)

Möbius action on oriented circlines preserve the set of points of the circline.

lemma *ocircline-set-moebius-ocircline* [*simp*]:
shows *ocircline-set (moebius-ocircline M H) = moebius-pt M ‘ ocircline-set H* (**is** *?lhs = ?rhs*)
proof−
have *?rhs = circline-set (moebius-circline M (of-ocircline H))*
by *simp*
thus *?thesis*
using *moebius-ocircline-circline*[*of M H*]
by *auto*
qed

lemma *ocircline-set-fix-iff-ocircline-fix*:
assumes *ocircline-set H' ≠ {}*
shows *ocircline-set (moebius-ocircline M H) = ocircline-set H' ⟷*
moebius-ocircline M H = H' ∨ moebius-ocircline M H = opposite-ocircline H'
using *assms*
using *inj-ocircline-set*[*of moebius-ocircline M H H'*]

```
  by (auto simp del: ocircline-set-moebius-ocircline)

lemma disc-moebius-ocircline [simp]:
  shows disc (moebius-ocircline M H) = moebius-pt M ‘ (disc H)
proof (safe)
  fix z
  assume z ∈ disc H
  thus moebius-pt M z ∈ disc (moebius-ocircline M H)
    unfolding disc-def
  proof (safe)
    assume in-ocircline H z
    thus in-ocircline (moebius-ocircline M H) (moebius-pt M z)
    proof (transfer, transfer)
      fix H M :: complex-mat and z :: complex-vec
      assume mat-det M ≠ 0
      assume in-ocircline-cmat-cvec H z
      thus in-ocircline-cmat-cvec (moebius-circline-cmat-cmat M H) (moebius-pt-cmat-cvec M z)
        using ‹mat-det M ≠ 0› quad-form-congruence[of M z]
        by simp
    qed
  qed
next
  fix z
  assume z ∈ disc (moebius-ocircline M H)
  thus z ∈ moebius-pt M ‘ disc H
    unfolding disc-def
  proof(safe)
    assume in-ocircline (moebius-ocircline M H) z
    show z ∈ moebius-pt M ‘ Collect (in-ocircline H)
    proof
      show z = moebius-pt M (moebius-pt (moebius-inv M) z)
        by simp
    next
      show moebius-pt (moebius-inv M) z ∈ Collect (in-ocircline H)
        using ‹in-ocircline (moebius-ocircline M H) z›
      proof (safe, transfer, transfer)
        fix M H :: complex-mat and z :: complex-vec
        assume mat-det M ≠ 0
        hence congruence (mat-inv (mat-inv M)) (congruence (mat-inv M) H) = H
          by (simp del: congruence-def)
        hence quad-form z (congruence (mat-inv M) H) = quad-form (mat-inv M *mv z) H
          using quad-form-congruence[of mat-inv M z congruence (mat-inv M) H]
          using ‹mat-det M ≠ 0› mat-det-inv[of M]
          by simp
        moreover
        assume in-ocircline-cmat-cvec (moebius-circline-cmat-cmat M H) z
        ultimately
        show in-ocircline-cmat-cvec H (moebius-pt-cmat-cvec (moebius-inv-cmat M) z)
          by simp
      qed
    qed
  qed
qed

lemma disc-compl-moebius-ocircline [simp]:
  shows disc-compl (moebius-ocircline M H) = moebius-pt M ‘ (disc-compl H)
proof (safe)
  fix z
  assume z ∈ disc-compl H
  thus moebius-pt M z ∈ disc-compl (moebius-ocircline M H)
    unfolding disc-compl-def
  proof (safe)
    assume out-ocircline H z
    thus out-ocircline (moebius-ocircline M H) (moebius-pt M z)
    proof (transfer, transfer)
      fix H M :: complex-mat and z :: complex-vec
```

```
    assume mat-det M ≠ 0
    assume out-ocircline-cmat-cvec H z
    thus out-ocircline-cmat-cvec (moebius-circline-cmat-cmat M H) (moebius-pt-cmat-cvec M z)
      using ‹mat-det M ≠ 0› quad-form-congruence[of M z]
      by simp
  qed
 qed
next
 fix z
 assume z ∈ disc-compl (moebius-ocircline M H)
 thus z ∈ moebius-pt M ‘ disc-compl H
  unfolding disc-compl-def
 proof(safe)
  assume out-ocircline (moebius-ocircline M H) z
  show z ∈ moebius-pt M ‘ Collect (out-ocircline H)
  proof
   show z = moebius-pt M (moebius-pt (moebius-inv M) z)
    by simp
  next
   show moebius-pt (moebius-inv M) z ∈ Collect (out-ocircline H)
    using ‹out-ocircline (moebius-ocircline M H) z›
   proof (safe, transfer, transfer)
    fix M H :: complex-mat and z :: complex-vec
    assume mat-det M ≠ 0
    hence congruence (mat-inv (mat-inv M)) (congruence (mat-inv M) H) = H
     by (simp del: congruence-def)
    hence quad-form z (congruence (mat-inv M) H) = quad-form (mat-inv M *mv z) H
     using quad-form-congruence[of mat-inv M z congruence (mat-inv M) H]
     using ‹mat-det M ≠ 0› mat-det-inv[of M]
     by simp
    moreover
    assume out-ocircline-cmat-cvec (moebius-circline-cmat-cmat M H) z
    ultimately
    show out-ocircline-cmat-cvec H (moebius-pt-cmat-cvec (moebius-inv-cmat M) z)
     by simp
   qed
  qed
 qed
qed
```

8.8 Orientation after Möbius transformations

All Euclidean similarities preserve circline orientation.

```
lemma moebius-similarity-oriented-lines-to-oriented-lines:
 assumes a ≠ 0
 shows ∞h ∈ ocircline-set H ⟷ ∞h ∈ ocircline-set (moebius-ocircline (moebius-similarity a b) H)
 using moebius-similarity-lines-to-lines[OF ‹a ≠ 0›, of b of-ocircline H]
 by simp

lemma moebius-similarity-preserve-orientation′:
 assumes a ≠ 0 and ∞h ∉ ocircline-set H and pos-oriented H
 shows pos-oriented (moebius-ocircline (moebius-similarity a b) H)
proof−
 let ?M = moebius-similarity a b
 let ?H = moebius-ocircline ?M H
 have ∞h ∉ ocircline-set ?H
  using ‹∞h ∉ ocircline-set H› moebius-similarity-oriented-lines-to-oriented-lines[OF ‹a ≠ 0›]
  by simp

 have ∞h ∈ disc-compl H
  using ‹∞h ∉ ocircline-set H› ‹pos-oriented H› pos-oriented-circle-inf[of H] in-on-out
  unfolding disc-def disc-compl-def ocircline-set-def
  by auto
 hence ∞h ∈ disc-compl ?H
  using moebius-similarity-inf[OF ‹a ≠ 0›, of b]
  by force
```

thus *pos-oriented ?H*
using *pos-oriented-circle-inf*[*of ?H*] *disc-inter-disc-compl*[*of ?H*] ‹$\infty_h \notin$ *ocircline-set ?H*›
by *auto*
qed

lemma *moebius-similarity-preserve-orientation*:
assumes $a \neq 0$ **and** $\infty_h \notin$ *ocircline-set H*
shows *pos-oriented H* $\longleftrightarrow$ *pos-oriented*(*moebius-ocircline* (*moebius-similarity a b*) *H*)
proof−
let *?M* = *moebius-similarity a b*
let *?H* = *moebius-ocircline ?M H*
have $\infty_h \notin$ *ocircline-set ?H*
using ‹$\infty_h \notin$ *ocircline-set H*› *moebius-similarity-oriented-lines-to-oriented-lines*[*OF* ‹$a \neq 0$›]
by *simp*

have ∗: *H* = *moebius-ocircline* (− *moebius-similarity a b*) *?H*
by *simp*
show *?thesis*
using ‹$a \neq 0$›
using *moebius-similarity-preserve-orientation*′ [*OF* ‹$a \neq 0$› ‹$\infty_h \notin$ *ocircline-set H*›]
using *moebius-similarity-preserve-orientation*′[*OF* - ‹$\infty_h \notin$ *ocircline-set ?H*›, *of* $1/a$ $-b/a$]
using *moebius-similarity-inv*[*of a b*, *OF* ‹$a \neq 0$›] ∗
by *auto*
qed

lemma *reciprocal-preserve-orientation*:
assumes $0_h \in$ *disc-compl H*
shows *pos-oriented* (*moebius-ocircline moebius-reciprocal H*)
proof−
have $\infty_h \in$ *disc-compl* (*moebius-ocircline moebius-reciprocal H*)
using *assms*
by *force*
thus *pos-oriented* (*moebius-ocircline moebius-reciprocal H*)
using *pos-oriented-circle-inf*[*of moebius-ocircline moebius-reciprocal H*]
using *disc-inter-disc-compl*[*of moebius-ocircline moebius-reciprocal H*]
using *disc-compl-inter-ocircline-set*[*of moebius-ocircline moebius-reciprocal H*]
by *auto*
qed

lemma *reciprocal-not-preserve-orientation*:
assumes $0_h \in$ *disc H*
shows ¬ *pos-oriented* (*moebius-ocircline moebius-reciprocal H*)
proof−
let *?H* = *moebius-ocircline moebius-reciprocal H*
have $\infty_h \in$ *disc ?H*
using *assms*
by *force*
thus ¬ *pos-oriented ?H*
using *pos-oriented-circle-inf*[*of ?H*] *disc-inter-ocircline-set*[*of ?H*]
by *auto*
qed

Orientation of the image of a given oriented circline H under a given Möbius transformation M depends on whether the pole of M (the point that M maps to ∞_{hc}) lies in the disc or in the disc complement of H (if it is on the set of H, then it maps onto a line and we do not discuss the orientation).

lemma *pole-in-disc*:
assumes *M* = *mk-moebius a b c d* **and** $c \neq 0$ **and** $a*d - b*c \neq 0$
assumes *is-pole M z z* ∈ *disc H*
shows ¬ *pos-oriented* (*moebius-ocircline M H*)
proof−
let *?t1* = *moebius-translation* (*a* / *c*)
let *?rd* = *moebius-rotation-dilatation* ((*b* ∗ *c* − *a* ∗ *d*) / (*c* ∗ *c*))
let *?r* = *moebius-reciprocal*
let *?t2* = *moebius-translation* (*d* / *c*)

have $0_h = moebius\text{-}pt\ (moebius\text{-}translation\ (d/c))\ z$
using *pole-mk-moebius*[*of a b c d z*] *assms*
by *simp*

have $z \notin ocircline\text{-}set\ H$
using ‹$z \in disc\ H$› *disc-inter-ocircline-set*[*of H*]
by *blast*

hence $0_h \notin ocircline\text{-}set\ (moebius\text{-}ocircline\ ?t2\ H)$
using ‹$0_h = moebius\text{-}pt\ ?t2\ z$›
using *moebius-pt-neq-I*[*of z - ?t2*]
by *force*

hence $*: \infty_h \notin ocircline\text{-}set\ (moebius\text{-}ocircline\ (?r + ?t2)\ H)$
using ‹$0_h = moebius\text{-}pt\ (moebius\text{-}translation\ (d\ /\ c))\ z$›
by (*metis circline-set-moebius-circline circline-set-moebius-circline-iff circline-set-of-ocircline moebius-pt-comp moebius-reciprocal ocircline-set-moebius-ocircline plus-moebius-def reciprocal-zero*)

hence $**: \infty_h \notin ocircline\text{-}set\ (moebius\text{-}ocircline\ (?rd + ?r + ?t2)\ H)$
using ‹$a*d - b*c \neq 0$› ‹$c \neq 0$›
unfolding *moebius-rotation-dilatation-def*
using *moebius-similarity-oriented-lines-to-oriented-lines*[*of - moebius-ocircline (?r + ?t2) H*]
by (*metis divide-eq-0-iff divisors-zero moebius-ocircline-comp plus-moebius-def right-minus-eq*)

have $\neg\ pos\text{-}oriented\ (moebius\text{-}ocircline\ (?r + ?t2)\ H)$
using *pole-mk-moebius*[*of a b c d z*] *assms*
using *reciprocal-not-preserve-orientation*
by *force*
hence $\neg\ pos\text{-}oriented\ (moebius\text{-}ocircline\ (?rd + ?r + ?t2)\ H)$
using $*$
using ‹$a*d - b*c \neq 0$› ‹$c \neq 0$›
using *moebius-similarity-preserve-orientation*[*of - moebius-ocircline (?r + ?t2) H*]
unfolding *moebius-rotation-dilatation-def*
by *simp*
hence $\neg\ pos\text{-}oriented\ (moebius\text{-}ocircline\ (?t1 + ?rd + ?r + ?t2)\ H)$
using $**$
using *moebius-similarity-preserve-orientation*[*of - moebius-ocircline (?rd + ?r + ?t2) H*]
unfolding *moebius-translation-def*
by *simp*

thus *?thesis*
using *assms*
by *simp* (*subst moebius-decomposition*, *simp-all*)
qed

lemma *pole-in-disc-compl*:
assumes $M = mk\text{-}moebius\ a\ b\ c\ d$ **and** $c \neq 0$ **and** $a*d - b*c \neq 0$
assumes *is-pole M z* **and** $z \in disc\text{-}compl\ H$
shows *pos-oriented* (*moebius-ocircline M H*)
proof−
let *?t1* = *moebius-translation* (*a* / *c*)
let *?rd* = *moebius-rotation-dilatation* ((*b* ∗ *c* − *a* ∗ *d*) / (*c* ∗ *c*))
let *?r* = *moebius-reciprocal*
let *?t2* = *moebius-translation* (*d* / *c*)

have $0_h = moebius\text{-}pt\ (moebius\text{-}translation\ (d/c))\ z$
using *pole-mk-moebius*[*of a b c d z*] *assms*
by *simp*

have $z \notin ocircline\text{-}set\ H$
using ‹$z \in disc\text{-}compl\ H$› *disc-compl-inter-ocircline-set*[*of H*]
by *blast*
hence $0_h \notin ocircline\text{-}set\ (moebius\text{-}ocircline\ ?t2\ H)$
using ‹$0_h = moebius\text{-}pt\ ?t2\ z$›
using *moebius-pt-neq-I*[*of z - ?t2*]

by *force*
hence ∗: $\infty_h \notin$ *ocircline-set* (*moebius-ocircline* (*?r* + *?t2*) *H*)
using ‹0_h = *moebius-pt* (*moebius-translation* (*d* / *c*)) *z*›
by (*metis circline-set-moebius-circline circline-set-moebius-circline-iff circline-set-of-ocircline moebius-pt-comp moebius-reciprocal ocircline-set-moebius-ocircline plus-moebius-def reciprocal-zero*)

hence ∗∗: $\infty_h \notin$ *ocircline-set* (*moebius-ocircline* (*?rd* + *?r* + *?t2*) *H*)
using ‹*a*∗*d* − *b*∗*c* ≠ *0*› ‹*c* ≠ *0*›
unfolding *moebius-rotation-dilatation-def*
using *moebius-similarity-oriented-lines-to-oriented-lines*[*of* - *moebius-ocircline* (*?r* + *?t2*) *H*]
by (*metis divide-eq-0-iff divisors-zero moebius-ocircline-comp plus-moebius-def right-minus-eq*)

have *pos-oriented* (*moebius-ocircline* (*?r* + *?t2*) *H*)
using *pole-mk-moebius*[*of a b c d z*] *assms*
using *reciprocal-preserve-orientation*
by *force*
hence *pos-oriented* (*moebius-ocircline* (*?rd* + *?r* + *?t2*) *H*)
using ∗
using ‹*a*∗*d* − *b*∗*c* ≠ *0*› ‹*c* ≠ *0*›
using *moebius-similarity-preserve-orientation*[*of* - *moebius-ocircline* (*?r* + *?t2*) *H*]
unfolding *moebius-rotation-dilatation-def*
by *simp*
hence *pos-oriented* (*moebius-ocircline* (*?t1* + *?rd* + *?r* + *?t2*) *H*)
using ∗∗
using *moebius-similarity-preserve-orientation*[*of* - *moebius-ocircline* (*?rd* + *?r* + *?t2*) *H*]
unfolding *moebius-translation-def*
by *simp*

thus *?thesis*
using *assms*
by *simp* (*subst moebius-decomposition*, *simp-all*)
qed

8.9 Oriented circlines uniqueness

lemma *ocircline-01inf*:
assumes $0_h \in$ *ocircline-set H* ∧ $1_h \in$ *ocircline-set H* ∧ $\infty_h \in$ *ocircline-set H*
shows *H* = *o-x-axis* ∨ *H* = *opposite-ocircline o-x-axis*
proof−
have $0_h \in$ *circline-set* (*of-ocircline H*) ∧ $1_h \in$ *circline-set* (*of-ocircline H*) ∧ $\infty_h \in$ *circline-set* (*of-ocircline H*)
using *assms*
by *simp*
hence *of-ocircline H* = *x-axis*
using *unique-circline-01inf′*
by *auto*
thus *H* = *o-x-axis* ∨ *H* = *opposite-ocircline o-x-axis*
by (*metis inj-of-ocircline of-ocircline-o-x-axis*)
qed

lemma *unique-ocircline-01inf*:
shows ∃! *H*. $0_h \in$ *ocircline-set H* ∧ $1_h \in$ *ocircline-set H* ∧ $\infty_h \in$ *ocircline-set H* ∧ $ii_h \notin$ *disc H*
proof
show $0_h \in$ *ocircline-set o-x-axis* ∧ $1_h \in$ *ocircline-set o-x-axis* ∧ $\infty_h \in$ *ocircline-set o-x-axis* ∧ $ii_h \notin$ *disc o-x-axis*
by *simp*
next
fix *H*
assume $0_h \in$ *ocircline-set H* ∧ $1_h \in$ *ocircline-set H* ∧ $\infty_h \in$ *ocircline-set H* ∧ $ii_h \notin$ *disc H*
hence $0_h \in$ *ocircline-set H* ∧ $1_h \in$ *ocircline-set H* ∧ $\infty_h \in$ *ocircline-set H* $ii_h \notin$ *disc H*
by *auto*
hence *H* = *o-x-axis* ∨ *H* = *opposite-ocircline o-x-axis*
using *ocircline-01inf*
by *simp*
thus *H* = *o-x-axis*
using ‹$ii_h \notin$ *disc H*›
by *auto*
qed

```
lemma unique-ocircline-set:
  assumes A ≠ B and A ≠ C and B ≠ C
  shows ∃! H. pos-oriented H ∧ (A ∈ ocircline-set H ∧ B ∈ ocircline-set H ∧ C ∈ ocircline-set H)
proof−
  obtain M where ∗: moebius-pt M A = 0_h  moebius-pt M B = 1_h moebius-pt M C = ∞_h
    using ex-moebius-01inf[OF assms]
    by auto
  let ?iM = moebius-pt (moebius-inv M)
  have ∗∗: ?iM 0_h = A  ?iM 1_h = B  ?iM ∞_h = C
    using ∗
    by (auto simp add: moebius-pt-invert)
  let ?H = moebius-ocircline (moebius-inv M) o-x-axis
  have 1: A ∈ ocircline-set ?H B ∈ ocircline-set ?H C ∈ ocircline-set ?H
    using ∗∗
    by auto
 have 2: ⋀ H'. A ∈ ocircline-set H' ∧ B ∈ ocircline-set H' ∧ C ∈ ocircline-set H' ⟹ H' = ?H ∨ H' = opposite-ocircline
?H
  proof−
    fix H'
    let ?H' = ocircline-set H' and ?H'' = ocircline-set (moebius-ocircline M H')
    assume A ∈ ocircline-set H' ∧ B ∈ ocircline-set H' ∧ C ∈ ocircline-set H'
    hence moebius-pt M A ∈ ?H'' moebius-pt M B ∈ ?H'' moebius-pt M C ∈ ?H''
      by auto
    hence 0_h ∈ ?H'' 1_h ∈ ?H''  ∞_h ∈ ?H''
      using ∗
      by auto
    hence moebius-ocircline M H' = o-x-axis ∨ moebius-ocircline M H' = opposite-ocircline o-x-axis
      using ocircline-01inf
      by auto
    hence o-x-axis = moebius-ocircline M H' ∨  o-x-axis = opposite-ocircline (moebius-ocircline M H')
      by auto
    thus H' = ?H ∨ H' = opposite-ocircline ?H
    proof
      assume ∗: o-x-axis = moebius-ocircline M H'
      show H' = moebius-ocircline (moebius-inv M) o-x-axis ∨ H' = opposite-ocircline (moebius-ocircline (moebius-inv
M) o-x-axis)
        by (rule disjI1) (subst ∗, simp)
    next
      assume ∗: o-x-axis = opposite-ocircline (moebius-ocircline M H')
      show H' = moebius-ocircline (moebius-inv M) o-x-axis ∨ H' = opposite-ocircline (moebius-ocircline (moebius-inv
M) o-x-axis)
        by (rule disjI2) (subst ∗, simp)
    qed
  qed

  show ?thesis (is ∃! x. ?P x)
  proof (cases pos-oriented ?H)
    case True
    show ?thesis
    proof
      show ?P ?H
        using 1 True
        by auto
    next
      fix H
      assume ?P H
      thus H = ?H
        using 1 2[of H] True
        by auto
    qed
  next
    case False
    let ?OH = opposite-ocircline ?H
    show ?thesis
    proof
```

show *?P ?OH*
using *1 False*
by *auto*
next
fix *H*
assume *?P H*
thus *H = ?OH*
using *False 2*[*of H*]
by *auto*
qed
qed
qed

lemma *ocircline-set-0h*:
assumes *ocircline-set H* = {0_h}
shows *H = o-circline-point-0* ∨ *H = opposite-ocircline* (*o-circline-point-0*)
proof−
have *of-ocircline H = circline-point-0*
using *assms*
using *unique-circline-type-zero-0′ card-eq1-circline-type-zero*[*of of-ocircline H*]
by *auto*
thus *?thesis*
by (*metis inj-of-ocircline of-ocircline-o-circline-point-0*)
qed

end
theory *Circlines-Angle*
imports *Oriented-Circlines Elementary-Complex-Geometry*
begin

8.10 Angle between circlines

Angle between circlines can be defined in purely algebraic terms (following Schwerdtfeger [13]) and using this definitions many properties can be easily proved.

fun *mat-det-12* :: *complex-mat* ⇒ *complex-mat* ⇒ *complex* **where**
mat-det-12 (*A1*, *B1*, *C1*, *D1*) (*A2*, *B2*, *C2*, *D2*) = *A1*∗*D2* + *A2*∗*D1* − *B1*∗*C2* − *B2*∗*C1*

lemma *mat-det-12-mm-l* [*simp*]:
shows *mat-det-12* (*M* ∗$_{mm}$ *A*) (*M* ∗$_{mm}$ *B*) = *mat-det M* ∗ *mat-det-12 A B*
by (*cases M*, *cases A*, *cases B*) (*simp add*: *field-simps*)

lemma *mat-det-12-mm-r* [*simp*]:
shows *mat-det-12* (*A* ∗$_{mm}$ *M*) (*B* ∗$_{mm}$ *M*) = *mat-det M* ∗ *mat-det-12 A B*
by (*cases M*, *cases A*, *cases B*) (*simp add*: *field-simps*)

lemma *mat-det-12-sm-l* [*simp*]:
shows *mat-det-12* (*k* ∗$_{sm}$ *A*) *B* = *k* ∗ *mat-det-12 A B*
by (*cases A*, *cases B*) (*simp add*: *field-simps*)

lemma *mat-det-12-sm-r* [*simp*]:
shows *mat-det-12 A* (*k* ∗$_{sm}$ *B*) = *k* ∗ *mat-det-12 A B*
by (*cases A*, *cases B*) (*simp add*: *field-simps*)

lemma *mat-det-12-congruence* [*simp*]:
shows *mat-det-12* (*congruence M A*) (*congruence M B*) = (*cor* ((*cmod* (*mat-det M*))^2)) ∗ *mat-det-12 A B*
unfolding *congruence-def*
by ((*subst mult-mm-assoc*[*symmetric*])+, *subst mat-det-12-mm-l*, *subst mat-det-12-mm-r*, *subst mat-det-adj*) (*auto simp add*: *field-simps complex-mult-cnj-cmod*)

definition *cos-angle-cmat* :: *complex-mat* ⇒ *complex-mat* ⇒ *real* **where**
[*simp*]: *cos-angle-cmat H1 H2* = − *Re* (*mat-det-12 H1 H2*) / (2 ∗ (*sqrt* (*Re* (*mat-det H1* ∗ *mat-det H2*))))

lift-definition *cos-angle-clmat* :: *circline-mat* ⇒ *circline-mat* ⇒ *real* **is** *cos-angle-cmat*

done

lemma *cos-angle-den-scale* [*simp*]:
assumes *k1* > *0* **and** *k2* > *0*
shows *sqrt* (*Re* ((*k1*^{2} ∗ *mat-det H1*) ∗ (*k2*^{2} ∗ *mat-det H2*))) =
k1 ∗ *k2* ∗ *sqrt* (*Re* (*mat-det H1* ∗ *mat-det H2*))
proof−
let *?lhs* = (*k1*^{2} ∗ *mat-det H1*) ∗ (*k2*^{2} ∗ *mat-det H2*)
let *?rhs* = *mat-det H1* ∗ *mat-det H2*
have *1*: *?lhs* = (*k1*^{2}∗*k2*^{2}) ∗ *?rhs*
by *simp*
hence *Re ?lhs* = (*k1*^{2}∗*k2*^{2}) ∗ *Re ?rhs*
by (*simp add*: *field-simps*)
thus *?thesis*
using *assms*
by (*simp add*: *real-sqrt-mult*)
qed

lift-definition *cos-angle* :: *ocircline* ⇒ *ocircline* ⇒ *real* **is** *cos-angle-clmat*
proof *transfer*
fix *H1 H2 H1′ H2′*
assume *ocircline-eq-cmat H1 H1′ ocircline-eq-cmat H2 H2′*
then obtain *k1 k2* :: *real* **where**
∗: *k1* > *0 H1′* = *cor k1* ∗$_{sm}$ *H1*
k2 > *0 H2′* = *cor k2* ∗$_{sm}$ *H2*
by *auto*
thus *cos-angle-cmat H1 H2* = *cos-angle-cmat H1′ H2′*
unfolding *cos-angle-cmat-def*
apply (*subst* ∗)+
apply (*subst mat-det-12-sm-l*, *subst mat-det-12-sm-r*)
apply (*subst mat-det-mult-sm*)+
apply (*subst power2-eq-square*[*symmetric*])+
apply (*subst cos-angle-den-scale*, *simp*, *simp*)
apply *simp*
done
qed

Möbius transformations are conformal, meaning that they preserve oriented angle between oriented circlines.

lemma *cos-angle-opposite1* [*simp*]:
shows *cos-angle* (*opposite-ocircline H*) *H′* = − *cos-angle H H′*
by (*transfer*, *transfer*, *simp*)

lemma *cos-angle-opposite2* [*simp*]:
shows *cos-angle H* (*opposite-ocircline H′*) = − *cos-angle H H′*
by (*transfer*, *transfer*, *simp*)

8.10.1 Connection with the elementary angle definition between circles

We want to connect algebraic definition of an angle with a traditional one and to prove equivalency between these two definitions. For the traditional definition of an angle we follow the approach suggested by Needham [10].

lemma *Re-sgn*:
assumes *is-real A* **and** *A* ≠ *0*
shows *Re* (*sgn A*) = *sgn-bool* (*Re A* > *0*)
using *assms*
using *More-Complex.Re-sgn complex-eq-if-Re-eq*
by *auto*

lemma *Re-mult-real3*:
assumes *is-real z1* **and** *is-real z2* **and** *is-real z3*
shows *Re* (*z1* ∗ *z2* ∗ *z3*) = *Re z1* ∗ *Re z2* ∗ *Re z3*
using *assms*
by (*metis Re-mult-real mult-reals*)

lemma *sgn-sqrt* [*simp*]:

```
  shows sgn (sqrt x) = sgn x
  by (simp add: sgn-root sqrt-def)

lemma real-circle-sgn-r:
  assumes is-circle H and (a, r) = euclidean-circle H
  shows sgn r = − circline-type H
  using assms
proof (transfer, transfer)
  fix H :: complex-mat and a r
  assume hh: hermitean H ∧ H ≠ mat-zero
  obtain A B C D where HH: H = (A, B, C, D)
    by (cases H) auto
  hence is-real A is-real D
    using hermitean-elems hh
    by auto
  assume ¬ circline-A0-cmat H (a, r) = euclidean-circle-cmat H
  hence A ≠ 0
    using ‹¬ circline-A0-cmat H› HH
    by simp
  hence Re A * Re A > 0
    using ‹is-real A›
    using complex-eq-if-Re-eq not-real-square-gt-zero
    by fastforce
  thus sgn r = − circline-type-cmat H
    using HH ‹(a, r) = euclidean-circle-cmat H› ‹is-real A› ‹is-real D› ‹A ≠ 0›
    by (simp add: Re-divide-real sgn-minus[symmetric])
qed
```

The definition of an angle using algebraic terms is not intuitive, and we want to connect it to the more common definition given earlier that defines an angle between circlines as the angle between tangent vectors in the point of the intersection of the circlines.

```
lemma cos-angle-eq-cos-ang-circ:
  assumes
  is-circle (of-ocircline H1) and is-circle (of-ocircline H2) and
  circline-type (of-ocircline H1) < 0 and circline-type (of-ocircline H2) < 0
  (a1, r1) = euclidean-circle (of-ocircline H1) and (a2, r2) = euclidean-circle (of-ocircline H2) and
  of-complex E ∈ ocircline-set H1 ∩ ocircline-set H2
  shows cos-angle H1 H2 = cos (ang-circ E a1 a2 (pos-oriented H1) (pos-oriented H2))
proof−
  let ?p1 = pos-oriented H1 and ?p2 = pos-oriented H2
  have E ∈ circle a1 r1 E ∈ circle a2 r2
    using classic-circle[of of-ocircline H1 a1 r1]  classic-circle[of of-ocircline H2 a2 r2]
    using assms of-complex-inj
    by auto
  hence ∗: cdist E a1 = r1 cdist E a2 = r2
    unfolding circle-def
    by (simp-all add: norm-minus-commute)
  have r1 > 0 r2 > 0
    using assms(1−6) real-circle-sgn-r[of of-ocircline H1 a1 r1]  real-circle-sgn-r[of of-ocircline H2 a2 r2]
    using sgn-greater
    by fastforce+
  hence E ≠ a1 E ≠ a2
    using ‹cdist E a1 = r1› ‹cdist E a2 = r2›
    by auto

  let ?k = sgn-bool (?p1 = ?p2)
  let ?xx = ?k * (r1^2 + r2^2 − (cdist a2 a1)^2) / (2 * r1 * r2)

  have cos (ang-circ E a1 a2 ?p1 ?p2) = ?xx
    using law-of-cosines[of a2 a1 E] ∗ ‹r1 > 0› ‹r2 > 0› cos-ang-circ-simp[OF ‹E ≠ a1› ‹E ≠ a2›]
    by (subst (asm) ang-vec-opposite-opposite′[OF ‹E ≠ a1›[symmetric] ‹E ≠ a2›[symmetric], symmetric]) simp
  moreover
  have cos-angle H1 H2 = ?xx
    using ‹r1 > 0› ‹r2 > 0›
    using ‹(a1, r1) = euclidean-circle (of-ocircline H1)› ‹(a2, r2) = euclidean-circle (of-ocircline H2)›
    using ‹is-circle (of-ocircline H1)› ‹is-circle (of-ocircline H2)›
```

```
  using ‹circline-type (of-ocircline H1) < 0› ‹circline-type (of-ocircline H2) < 0›
proof (transfer, transfer)
  fix a1 r1 H1 H2 a2 r2
  assume hh: hermitean H1 ∧ H1 ≠ mat-zero hermitean H2 ∧ H2 ≠ mat-zero
  obtain A1 B1 C1 D1 where HH1: H1 = (A1, B1, C1, D1)
    by (cases H1) auto
  obtain A2 B2 C2 D2 where HH2: H2 = (A2, B2, C2, D2)
    by (cases H2) auto
  have ∗: is-real A1 is-real A2 is-real D1 is-real D2 cnj B1 = C1 cnj B2 = C2
    using hh hermitean-elems[of A1 B1 C1 D1] hermitean-elems[of A2 B2 C2 D2] HH1 HH2
    by auto
  have cnj A1 = A1 cnj A2 = A2
    using ‹is-real A1› ‹is-real A2›
    by (case-tac[!] A1, case-tac[!] A2, auto simp add: Complex-eq)

  assume ¬ circline-A0-cmat (id H1) ¬ circline-A0-cmat (id H2)
  hence A1 ≠ 0 A2 ≠ 0
    using HH1 HH2
    by auto
  hence Re A1 ≠ 0 Re A2 ≠ 0
    using ‹is-real A1› ‹is-real A2›
    using complex.expand
    by auto

  assume circline-type-cmat (id H1) < 0 circline-type-cmat (id H2) < 0
  assume (a1, r1) = euclidean-circle-cmat (id H1) (a2, r2) = euclidean-circle-cmat (id H2)
  assume r1 > 0 r2 > 0

  let ?D12 = mat-det-12 H1 H2 and ?D1 = mat-det H1 and ?D2 = mat-det H2
  let ?x1 = (cdist a2 a1)^2 − r1^2 − r2^2 and ?x2 = 2∗r1∗r2
  let ?x = ?x1 / ?x2
  have ∗: Re (?D12) / (2 ∗ (sqrt (Re (?D1 ∗ ?D2)))) = Re (sgn A1) ∗ Re (sgn A2) ∗ ?x
  proof−
    let ?M1 = (A1, B1, C1, D1) and ?M2 = (A2, B2, C2, D2)
    let ?d1 = B1 ∗ C1 − A1 ∗ D1 and ?d2 = B2 ∗ C2 − A2 ∗ D2
    have Re ?d1 > 0 Re ?d2 > 0
      using HH1 HH2 ‹circline-type-cmat (id H1) < 0› ‹circline-type-cmat (id H2) < 0›
      by auto
    hence ∗∗: Re (?d1 / (A1 ∗ A1)) > 0 Re (?d2 / (A2 ∗ A2)) > 0
      using ‹is-real A1› ‹is-real A2› ‹A1 ≠ 0› ‹A2 ≠ 0›
      by (subst Re-divide-real, simp-all add: complex-neq-0 power2-eq-square)+
    have ∗∗∗: is-real (?d1 / (A1 ∗ A1)) ∧ is-real (?d2 / (A2 ∗ A2))
      using ‹is-real A1› ‹is-real A2› ‹A1 ≠ 0› ‹A2 ≠ 0› ‹cnj B1 = C1›[symmetric] ‹cnj B2 = C2›[symmetric]
‹is-real D1› ‹is-real D2›
      by (subst div-reals, simp, simp, simp)+

    have cor ?x = mat-det-12 ?M1 ?M2 / (2 ∗ sgn A1 ∗ sgn A2 ∗ cor (sqrt (Re ?d1) ∗ sqrt (Re ?d2)))
    proof−
      have A1∗A2∗cor ?x1 = mat-det-12 ?M1 ?M2
      proof−
        have 1: A1∗A2∗(cor ((cdist a2 a1)^2)) = ((B2∗A1 − A2∗B1)∗(C2∗A1 − C1∗A2)) / (A1∗A2)
          using ‹(a1, r1) = euclidean-circle-cmat (id H1)› ‹(a2, r2) = euclidean-circle-cmat (id H2)›
          unfolding cdist-def cmod-square
          using HH1 HH2 ∗ ‹A1 ≠ 0› ‹A2 ≠ 0› ‹cnj A1 = A1› ‹cnj A2 = A2›
          unfolding Let-def
          apply (subst complex-of-real-Re)
          apply (simp add: field-simps)
          apply (simp add: complex-mult-cnj-cmod power2-eq-square)
          apply (simp add: field-simps)
          done
        have 2: A1∗A2∗cor (−r1^2) = A2∗D1 − B1∗C1∗A2/A1
          using ‹(a1, r1) = euclidean-circle-cmat (id H1)›
          using HH1 ∗∗ ∗ ∗∗∗ ‹A1 ≠ 0›
          by (simp add: power2-eq-square field-simps)
        have 3: A1∗A2∗cor (−r2^2) = A1∗D2 − B2∗C2∗A1/A2
          using ‹(a2, r2) = euclidean-circle-cmat (id H2)›
```

```
    using HH2 ** * *** ‹A2 ≠ 0›
    by (simp add: power2-eq-square field-simps)
  have A1*A2*cor((cdist a2 a1)^2) + A1*A2*cor(-r1^2) + A1*A2*cor(-r2^2) = mat-det-12 ?M1 ?M2
    using ‹A1 ≠ 0› ‹A2 ≠ 0›
    by (subst 1, subst 2, subst 3) (simp add: field-simps)
  thus ?thesis
    by (simp add: field-simps)
qed

moreover

have A1 * A2 * cor (?x2) = 2 * sgn A1 * sgn A2 * cor (sqrt (Re ?d1) * sqrt (Re ?d2))
proof−
  have 1: sqrt (Re (?d1/ (A1 * A1))) = sqrt (Re ?d1) / |Re A1|
    using ‹A1 ≠ 0› ‹is-real A1›
    by (subst Re-divide-real, simp, simp, subst real-sqrt-divide, simp)

  have 2: sqrt (Re (?d2/ (A2 * A2))) = sqrt (Re ?d2) / |Re A2|
    using ‹A2 ≠ 0› ‹is-real A2›
    by (subst Re-divide-real, simp, simp, subst real-sqrt-divide, simp)
  have sgn A1 = A1 / cor |Re A1|
    using ‹is-real A1›
    unfolding sgn-eq
    by (simp add: cmod-eq-Re)
  moreover
  have sgn A2 = A2 / cor |Re A2|
    using ‹is-real A2›
    unfolding sgn-eq
    by (simp add: cmod-eq-Re)
  ultimately
  show ?thesis
    using ‹(a1, r1) = euclidean-circle-cmat (id H1)› ‹(a2, r2) = euclidean-circle-cmat (id H2)›  HH1 HH2
    using *** ‹is-real A1› ‹is-real A2›
    by simp (subst 1, subst 2, simp)
qed

ultimately

have (A1 * A2 * cor ?x1) / (A1 * A2 * (cor ?x2)) =
      mat-det-12 ?M1 ?M2 / (2 * sgn A1 * sgn A2 * cor (sqrt (Re ?d1) * sqrt (Re ?d2)))
  by simp
thus ?thesis
  using ‹A1 ≠ 0› ‹A2 ≠ 0›
  by simp
qed
hence cor ?x * sgn A1 * sgn A2 = mat-det-12 ?M1 ?M2 / (2 * cor (sqrt (Re ?d1) * sqrt (Re ?d2)))
  using ‹A1 ≠ 0› ‹A2 ≠ 0›
  by (simp add: sgn-zero-iff)
moreover
have Re (cor ?x * sgn A1 * sgn A2) = Re (sgn A1) * Re (sgn A2) * ?x
proof−
  have is-real (cor ?x) is-real (sgn A1) is-real (sgn A2)
    using ‹is-real A1› ‹is-real A2› Im-complex-of-real[of ?x]
    by auto
  thus ?thesis
    using Re-complex-of-real[of ?x]
    by (subst Re-mult-real3, auto simp add: field-simps)
qed
moreover
have *: sqrt (Re ?D1) * sqrt (Re ?D2) = sqrt (Re ?d1) * sqrt (Re ?d2)
  using HH1 HH2
  by (subst real-sqrt-mult[symmetric])+ (simp add: field-simps)
have 2 * (sqrt (Re (?D1 * ?D2))) ≠ 0
  using ‹Re ?d1 > 0›  ‹Re ?d2 > 0› HH1 HH2 ‹is-real A1› ‹is-real A2›  ‹is-real D1› ‹is-real D2›
  using hh mat-det-hermitean-real[of H1]
  by (subst Re-mult-real, auto)
```

hence $**$: $Re\ (?D12\ /\ (2 * cor\ (sqrt\ (Re\ (?D1 * ?D2))))) = Re\ (?D12)\ /\ (2 * (sqrt\ (Re\ (?D1 * ?D2))))$
using ‹$Re\ ?d1 > 0$› ‹$Re\ ?d2 > 0$› $HH1\ HH2$ ‹$is\text{-}real\ A1$› ‹$is\text{-}real\ A2$› ‹$is\text{-}real\ D1$› ‹$is\text{-}real\ D2$›
by ($subst\ Re\text{-}divide\text{-}real$) $auto$
have $Re\ (mat\text{-}det\text{-}12\ ?M1\ ?M2\ /\ (2 * cor\ (sqrt\ (Re\ ?d1) * sqrt\ (Re\ ?d2)))) = Re\ (?D12)\ /\ (2 * (sqrt\ (Re\ (?D1 * ?D2))))$
using $HH1\ HH2\ hh\ mat\text{-}det\text{-}hermitean\text{-}real[of\ H1]$
by ($subst\ **[symmetric]$, $subst\ Re\text{-}mult\text{-}real$, $simp$, $subst\ real\text{-}sqrt\text{-}mult$, $subst\ *$, $simp$)
ultimately
show $?thesis$
by $simp$
qed
have $**$: $pos\text{-}oriented\text{-}cmat\ H1 \longleftrightarrow Re\ A1 > 0\ \ pos\text{-}oriented\text{-}cmat\ H2 \longleftrightarrow Re\ A2 > 0$
using ‹$Re\ A1 \neq 0$› $HH1$ ‹$Re\ A2 \neq 0$› $HH2$
by $auto$
show $cos\text{-}angle\text{-}cmat\ H1\ H2 = sgn\text{-}bool\ (pos\text{-}oriented\text{-}cmat\ H1 = pos\text{-}oriented\text{-}cmat\ H2) * (r1^2 + r2^2 - (cdist\ a2\ a1)^2)\ /\ (2 * r1 * r2)$
unfolding $Let\text{-}def$
using ‹$r1 > 0$› ‹$r2 > 0$›
unfolding $cos\text{-}angle\text{-}cmat\text{-}def$
apply ($subst\ divide\text{-}minus\text{-}left$)
apply ($subst\ *$)
apply ($subst\ Re\text{-}sgn[OF$ ‹$is\text{-}real\ A1$› ‹$A1 \neq 0$›$]$, $subst\ Re\text{-}sgn[OF$ ‹$is\text{-}real\ A2$› ‹$A2 \neq 0$›$]$)
apply ($subst\ **$, $subst\ **$)
apply ($simp\ add$: $field\text{-}simps$)
done
qed
ultimately
show $?thesis$
by $simp$
qed

8.11 Perpendicularity

Two circlines are perpendicular if the intersect at right angle i.e., the angle with the cosine 0.

definition $perpendicular$ **where**
$perpendicular\ H1\ H2 \longleftrightarrow cos\text{-}angle\ (of\text{-}circline\ H1)\ (of\text{-}circline\ H2) = 0$

lemma $perpendicular\text{-}sym$:
shows $perpendicular\ H1\ H2 \longleftrightarrow perpendicular\ H2\ H1$
unfolding $perpendicular\text{-}def$
by ($transfer$, $transfer$, $auto\ simp\ add$: $field\text{-}simps$)

8.12 Möbius transforms preserve angles and perpendicularity

Möbius transformations are *conformal* i.e., they preserve angles between circlines.

lemma $moebius\text{-}preserve\text{-}circline\text{-}angle$ [$simp$]:
shows $cos\text{-}angle\ (moebius\text{-}ocircline\ M\ H1)\ (moebius\text{-}ocircline\ M\ H2) =$
$cos\text{-}angle\ H1\ H2$
proof ($transfer$, $transfer$)
fix $H1\ H2\ M :: complex\text{-}mat$
assume hh: $mat\text{-}det\ M \neq 0$
show $cos\text{-}angle\text{-}cmat\ (moebius\text{-}circline\text{-}cmat\text{-}cmat\ M\ H1)\ (moebius\text{-}circline\text{-}cmat\text{-}cmat\ M\ H2) = cos\text{-}angle\text{-}cmat\ H1\ H2$
unfolding $cos\text{-}angle\text{-}cmat\text{-}def\ moebius\text{-}circline\text{-}cmat\text{-}cmat\text{-}def$
unfolding $Let\text{-}def\ mat\text{-}det\text{-}12\text{-}congruence\ mat\text{-}det\text{-}congruence$
using $hh\ mat\text{-}det\text{-}inv[of\ M]$
apply ($subst\ cor\text{-}squared[symmetric]$)+
apply ($subst\ cos\text{-}angle\text{-}den\text{-}scale$, $simp$)
apply ($auto\ simp\ add$: $power2\text{-}eq\text{-}square\ real\text{-}sqrt\text{-}mult\ field\text{-}simps$)
done
qed

lemma $perpendicular\text{-}moebius$ [$simp$]:
assumes $perpendicular\ H1\ H2$
shows $perpendicular\ (moebius\text{-}circline\ M\ H1)\ (moebius\text{-}circline\ M\ H2)$

using *assms*
unfolding *perpendicular-def*
using *moebius-preserve-circline-angle*[*of M of-circline H1 of-circline H2*]
using *moebius-ocircline-circline*[*of M of-circline H1*]
using *moebius-ocircline-circline*[*of M of-circline H2*]
by (*auto simp del*: *moebius-preserve-circline-angle*)

end

9 Unit circle preserving Möbius transformations

In this section we shall examine Möbius transformations that map the unit circle onto itself. We shall say that they fix or preserve the unit circle (although, they do not need to fix each of its points).

theory *Unit-Circle-Preserving-Moebius*
imports *Unitary11-Matrices Moebius Oriented-Circlines*
begin

9.1 Möbius transformations that fix the unit circle

We define Möbius transformations that preserve unit circle as transformations represented by generalized unitary matrices with the $1-1$ signature (elements of the gruop $GU_{1,1}(2,\mathbb{C})$, defined earlier in the theory Unitary11Matrices).

lift-definition *unit-circle-fix-mmat* :: *moebius-mat* ⇒ *bool* **is** *unitary11-gen*
done

lift-definition *unit-circle-fix* :: *moebius* ⇒ *bool* **is** *unit-circle-fix-mmat*
apply *transfer*
apply (*auto simp del*: *mult-sm.simps*)
apply (*simp del*: *mult-sm.simps add*: *unitary11-gen-mult-sm*)
apply (*simp del*: *mult-sm.simps add*: *unitary11-gen-div-sm*)
done

Our algebraic characterisation (by matrices) is geometrically correct.

lemma *unit-circle-fix-iff*:
shows *unit-circle-fix M* ⟷
moebius-circline M unit-circle = *unit-circle* (**is** *?rhs* = *?lhs*)
proof
assume *?lhs*
thus *?rhs*
proof (*transfer*, *transfer*)
fix *M* :: *complex-mat*
assume *mat-det M* ≠ *0*
assume *circline-eq-cmat* (*moebius-circline-cmat-cmat M unit-circle-cmat*) *unit-circle-cmat*
then obtain *k* **where** *k* ≠ *0* (*1*, *0*, *0*, −*1*) = *cor k* ∗$_{sm}$ *congruence* (*mat-inv M*) (*1*, *0*, *0*, −*1*)
by *auto*
hence (*1*/*cor k*, *0*, *0*, −*1*/*cor k*) = *congruence* (*mat-inv M*) (*1*, *0*, *0*, −*1*)
using *mult-sm-inv-l*[*of cor k congruence* (*mat-inv M*) (*1*, *0*, *0*, −*1*)]
by *simp*
hence *congruence M* (*1*/*cor k*, *0*, *0*, −*1*/*cor k*) = (*1*, *0*, *0*, −*1*)
using ‹*mat-det M* ≠ *0*› *mat-det-inv*[*of M*]
using *congruence-inv*[*of mat-inv M* (*1*, *0*, *0*, −*1*) (*1*/*cor k*, *0*, *0*, −*1*/*cor k*)]
by *simp*
hence *congruence M* (*1*, *0*, *0*, −*1*) = *cor k* ∗$_{sm}$ (*1*, *0*, *0*, −*1*)
using *congruence-scale-m*[*of M 1*/*cor k* (*1*, *0*, *0*, −*1*)]
using *mult-sm-inv-l*[*of 1*/ *cor k congruence M* (*1*, *0*, *0*, −*1*) (*1*, *0*, *0*, −*1*)] ‹*k* ≠ *0*›
by *simp*
thus *unitary11-gen M*
using ‹*k* ≠ *0*›
unfolding *unitary11-gen-def*
by *simp*
qed
next
assume *?rhs*

```
  thus ?lhs
  proof (transfer, transfer)
    fix M :: complex-mat
    assume mat-det M ≠ 0
    assume unitary11-gen M
    hence unitary11-gen (mat-inv M)
      using ‹mat-det M ≠ 0›
      using unitary11-gen-mat-inv
      by simp
    thus  circline-eq-cmat (moebius-circline-cmat-cmat M unit-circle-cmat) unit-circle-cmat
      unfolding unitary11-gen-real
      by auto (rule-tac x=1/k in exI, simp)
  qed
qed

lemma circline-set-fix-iff-circline-fix:
  assumes circline-set H' ≠ {}
  shows circline-set (moebius-circline M H) = circline-set H' ⟷
        moebius-circline M H = H'
  using assms
  by auto (rule inj-circline-set, auto)

lemma unit-circle-fix-iff-unit-circle-set:
  shows unit-circle-fix M ⟷ moebius-pt M ` unit-circle-set = unit-circle-set
proof−
  have circline-set unit-circle ≠ {}
    using one-in-unit-circle-set
    by auto
  thus ?thesis
    using unit-circle-fix-iff[of M] circline-set-fix-iff-circline-fix[of unit-circle M unit-circle]
    by (simp add: unit-circle-set-def)
qed
```

Unit circle preserving Möbius transformations form a group.

```
lemma unit-circle-fix-id-moebius [simp]:
  shows unit-circle-fix id-moebius
  by (transfer, transfer, simp add: unitary11-gen-def mat-adj-def mat-cnj-def)

lemma unit-circle-fix-moebius-add [simp]:
  assumes unit-circle-fix M1 and unit-circle-fix M2
  shows unit-circle-fix (M1 + M2)
  using assms
  unfolding unit-circle-fix-iff
  by auto

lemma unit-circle-fix-moebius-comp [simp]:
  assumes unit-circle-fix M1 and unit-circle-fix M2
  shows unit-circle-fix (moebius-comp M1 M2)
  using unit-circle-fix-moebius-add[OF assms]
  by simp

lemma unit-circle-fix-moebius-uminus [simp]:
  assumes unit-circle-fix M
  shows unit-circle-fix (−M)
  using assms
  unfolding unit-circle-fix-iff
  by (metis moebius-circline-comp-inv-left uminus-moebius-def)

lemma unit-circle-fix-moebius-inv [simp]:
  assumes unit-circle-fix M
  shows unit-circle-fix (moebius-inv M)
  using unit-circle-fix-moebius-uminus[OF assms]
  by simp
```

Unit circle fixing transforms preserve inverse points.

```
lemma unit-circle-fix-moebius-pt-inversion [simp]:
```

assumes *unit-circle-fix M*
shows *moebius-pt M (inversion z) = inversion (moebius-pt M z)*
using *assms*
using *symmetry-principle*[*of z inversion z unit-circle M*]
using *unit-circle-fix-iff*[*of M, symmetric*]
using *circline-symmetric-inv-homo-disc*[*of z*]
using *circline-symmetric-inv-homo-disc'*[*of moebius-pt M z moebius-pt M (inversion z)*]
by *metis*

9.2 Möbius transformations that fix the imaginary unit circle

Only for completeness we show that Möbius transformations that preserve the imaginary unit circle are exactly those characterised by generalized unitary matrices (with the (2, 0) signature).

lemma *imag-unit-circle-fixed-iff-unitary-gen*:
assumes *mat-det (A, B, C, D) ≠ 0*
shows *moebius-circline (mk-moebius A B C D) imag-unit-circle = imag-unit-circle ⟷*
unitary-gen (A, B, C, D) (**is** *?lhs = ?rhs*)
proof
assume *?lhs*
thus *?rhs*
using *assms*
proof (*transfer, transfer*)
fix *A B C D :: complex*
let *?M = (A, B, C, D)* **and** *?E = (1, 0, 0, 1)*
assume *circline-eq-cmat (moebius-circline-cmat-cmat (mk-moebius-cmat A B C D) imag-unit-circle-cmat) imag-unit-circle-cmat*
mat-det ?M ≠ 0
then obtain *k* **where** *k ≠ 0 ?E = cor k ∗sm congruence (mat-inv ?M) ?E*
by *auto*
hence *unitary-gen (mat-inv ?M)*
using *mult-sm-inv-l*[*of cor k congruence (mat-inv ?M) ?E ?E*]
unfolding *unitary-gen-def*
by (*metis congruence-def divide-eq-0-iff eye-def mat-eye-r of-real-eq-0-iff one-neq-zero*)
thus *unitary-gen ?M*
using *unitary-gen-inv*[*of mat-inv ?M*] ‹*mat-det ?M ≠ 0*›
by (*simp del*: *mat-inv.simps*)
qed
next
assume *?rhs*
thus *?lhs*
using *assms*
proof (*transfer, transfer*)
fix *A B C D :: complex*
let *?M = (A, B, C, D)* **and** *?E = (1, 0, 0, 1)*
assume *unitary-gen ?M mat-det ?M ≠ 0*
hence *unitary-gen (mat-inv ?M)*
using *unitary-gen-inv*[*of ?M*]
by *simp*
then obtain *k* **where** *k ≠ 0 mat-adj (mat-inv ?M) ∗mm (mat-inv ?M) = cor k ∗sm eye*
using *unitary-gen-real*[*of mat-inv ?M*] *mat-det-inv*[*of ?M*]
by *auto*
hence ∗: *?E = (1 / cor k) ∗sm (mat-adj (mat-inv ?M) ∗mm (mat-inv ?M))*
using *mult-sm-inv-l*[*of cor k eye mat-adj (mat-inv ?M) ∗mm (mat-inv ?M)*]
by *simp*
have ∃ *k. k ≠ 0 ∧*
(1, 0, 0, 1) = cor k ∗sm (mat-adj (mat-inv (A, B, C, D)) ∗mm (1, 0, 0, 1) ∗mm mat-inv (A, B, C, D))
using ‹*mat-det ?M ≠ 0*› ‹*k ≠ 0*›
by (*metis ∗ Im-complex-of-real Re-complex-of-real ‹mat-adj (mat-inv ?M) ∗mm mat-inv ?M = cor k ∗sm eye› complex-of-real-Re eye-def mat-eye-l mult-mm-assoc mult-mm-sm mult-sm-eye-mm of-real-1 of-real-divide of-real-eq-1-iff zero-eq-1-divide-iff*)
thus *circline-eq-cmat (moebius-circline-cmat-cmat (mk-moebius-cmat A B C D) imag-unit-circle-cmat) imag-unit-circle-cmat*
using ‹*mat-det ?M ≠ 0*› ‹*k ≠ 0*›
by (*simp del*: *mat-inv.simps*)
qed
qed

9.3 Möbius transformations that fix the oriented unit circle and the unit disc

Möbius transformations that fix the unit circle either map the unit disc onto itself or exchange it with its exterior. The transformations that fix the unit disc can be recognized from their matrices – they have the form as before, but additionally it must hold that $|a|^2 > |b|^2$.

definition *unit-disc-fix-cmat* :: *complex-mat* ⇒ *bool* **where**
[*simp*]: *unit-disc-fix-cmat M* ⟷
(*let* (*A*, *B*, *C*, *D*) = *M*
in unitary11-gen (*A*, *B*, *C*, *D*) ∧ (*B* = *0* ∨ *Re* ((*A*∗*D*)/(*B*∗*C*)) > *1*))

lift-definition *unit-disc-fix-mmat* :: *moebius-mat* ⇒ *bool* **is** *unit-disc-fix-cmat*
done

lift-definition *unit-disc-fix* :: *moebius* ⇒ *bool* **is** *unit-disc-fix-mmat*
proof *transfer*
fix *M M′* :: *complex-mat*
assume *det*: *mat-det M* ≠ *0 mat-det M′* ≠ *0*
assume *moebius-cmat-eq M M′*
then obtain *k* **where** ∗: *k* ≠ *0 M′* = *k* ∗$_{sm}$ *M*
by *auto*
hence ∗∗: *unitary11-gen M* ⟷ *unitary11-gen M′*
using *unitary11-gen-mult-sm*[*of k M*] *unitary11-gen-div-sm*[*of k M*]
by *auto*
obtain *A B C D* **where** *MM*: (*A*, *B*, *C*, *D*) = *M*
by (*cases M*) *auto*
obtain *A′ B′ C′ D′* **where** *MM′*: (*A′*, *B′*, *C′*, *D′*) = *M′*
by (*cases M′*) *auto*

show *unit-disc-fix-cmat M* = *unit-disc-fix-cmat M′*
using ∗ ∗∗ *MM MM′*
by *auto*
qed

Transformations that fix the unit disc also fix the unit circle.

lemma *unit-disc-fix-unit-circle-fix* [*simp*]:
assumes *unit-disc-fix M*
shows *unit-circle-fix M*
using *assms*
by (*transfer*, *transfer*, *auto*)

Transformations that preserve the unit disc preserve the orientation of the unit circle.

lemma *unit-disc-fix-iff-ounit-circle*:
shows *unit-disc-fix M* ⟷
moebius-ocircline M ounit-circle = *ounit-circle* (**is** *?rhs* ⟷ *?lhs*)
proof
assume ∗: *?lhs*
have *moebius-circline M unit-circle* = *unit-circle*
apply (*subst moebius-circline-ocircline*[*of M unit-circle*])
apply (*subst of-circline-unit-circle*)
apply (*subst* ∗)
by *simp*

hence *unit-circle-fix M*
by (*simp add*: *unit-circle-fix-iff*)
thus *?rhs*
using ∗
proof (*transfer*, *transfer*)
fix *M* :: *complex-mat*
assume *mat-det M* ≠ *0*
let *?H* = (*1*, *0*, *0*, −*1*)
obtain *A B C D* **where** *MM*: (*A*, *B*, *C*, *D*) = *M*
by (*cases M*) *auto*
assume *unitary11-gen M ocircline-eq-cmat* (*moebius-circline-cmat-cmat M unit-circle-cmat*) *unit-circle-cmat*
then obtain *k* **where** *0* < *k ?H* = *cor k* ∗$_{sm}$ *congruence* (*mat-inv M*) *?H*
by *auto*

```
    hence congruence M ?H = cor k *sm ?H
      using congruence-inv[of mat-inv M ?H (1/cor k) *sm ?H] ‹mat-det M ≠ 0›
      using mult-sm-inv-l[of cor k congruence (mat-inv M) ?H ?H]
      using mult-sm-inv-l[of 1/cor k congruence M ?H]
      using congruence-scale-m[of M 1/cor k ?H]
      using ‹⋀B. ⟦1 / cor k ≠ 0; (1 / cor k) *sm congruence M (1, 0, 0, − 1) = B⟧ ⟹ congruence M (1, 0, 0, −
1) = (1 / (1 / cor k)) *sm B›
      by (auto simp add: mat-det-inv)
    then obtain a b k' where k' ≠ 0 M = k' *sm (a, b, cnj b, cnj a) sgn (Re (mat-det (a, b, cnj b, cnj a))) = 1
      using unitary11-sgn-det-orientation'[of M k] ‹k > 0›
      by auto
    moreover
    have mat-det (a, b, cnj b, cnj a) ≠ 0
      using ‹sgn (Re (mat-det (a, b, cnj b, cnj a))) = 1›
      by (smt (verit) sgn-0 zero-complex.simps(1))
    ultimately
    show unit-disc-fix-cmat M
      using unitary11-sgn-det[of k' a b M A B C D]
      using MM[symmetric] ‹k > 0› ‹unitary11-gen M›
      by (simp add: sgn-1-pos split: if-split-asm)
  qed
next
  assume ?rhs
  thus ?lhs
  proof (transfer, transfer)
    fix M :: complex-mat
    assume mat-det M ≠ 0

    obtain A B C D where MM: (A, B, C, D) = M
      by (cases M) auto
    assume unit-disc-fix-cmat M
    hence unitary11-gen M B = 0 ∨ 1 < Re (A * D / (B * C))
      using MM[symmetric]
      by auto
    have sgn (if B = 0 then 1 else sgn (Re (A * D / (B * C)) − 1)) = 1
      using ‹B = 0 ∨ 1 < Re (A * D / (B * C))›
      by auto
    then obtain k' where k' > 0 congruence M (1, 0, 0, −1) = cor k' *sm (1, 0, 0, −1)
      using unitary11-orientation[OF ‹unitary11-gen M› MM[symmetric]]
      by (auto simp add: sgn-1-pos)
    thus ocircline-eq-cmat (moebius-circline-cmat-cmat M unit-circle-cmat) unit-circle-cmat
      using congruence-inv[of M (1, 0, 0, −1) cor k' *sm (1, 0, 0, −1)] ‹mat-det M ≠ 0›
      using congruence-scale-m[of mat-inv M cor k' (1, 0, 0, −1)]
      by auto
  qed
qed
```

Our algebraic characterisation (by matrices) is geometrically correct.

```
lemma unit-disc-fix-iff [simp]:
  assumes unit-disc-fix M
  shows moebius-pt M ‘ unit-disc = unit-disc
  using assms
  using unit-disc-fix-iff-ounit-circle[of M]
  unfolding unit-disc-def
  by (subst disc-moebius-ocircline[symmetric], simp)

lemma unit-disc-fix-discI [simp]:
  assumes unit-disc-fix M and u ∈ unit-disc
  shows moebius-pt M u ∈ unit-disc
  using unit-disc-fix-iff assms
  by blast
```

Unit disc preserving transformations form a group.

```
lemma unit-disc-fix-id-moebius [simp]:
  shows unit-disc-fix id-moebius
  by (transfer, transfer, simp add: unitary11-gen-def mat-adj-def mat-cnj-def)
```

lemma *unit-disc-fix-moebius-add* [*simp*]:
assumes *unit-disc-fix M1* **and** *unit-disc-fix M2*
shows *unit-disc-fix* (*M1* + *M2*)
using *assms*
unfolding *unit-disc-fix-iff-ounit-circle*
by *auto*

lemma *unit-disc-fix-moebius-comp* [*simp*]:
assumes *unit-disc-fix M1* **and** *unit-disc-fix M2*
shows *unit-disc-fix* (*moebius-comp M1 M2*)
using *unit-disc-fix-moebius-add*[*OF assms*]
by *simp*

lemma *unit-disc-fix-moebius-uminus* [*simp*]:
assumes *unit-disc-fix M*
shows *unit-disc-fix* (−*M*)
using *assms*
unfolding *unit-disc-fix-iff-ounit-circle*
by (*metis moebius-ocircline-comp-inv-left uminus-moebius-def*)

lemma *unit-disc-fix-moebius-inv* [*simp*]:
assumes *unit-disc-fix M*
shows *unit-disc-fix* (*moebius-inv M*)
using *unit-disc-fix-moebius-uminus*[*OF assms*]
by *simp*

9.4 Rotations are unit disc preserving transformations

lemma *unit-disc-fix-rotation* [*simp*]:
shows *unit-disc-fix* (*moebius-rotation* φ)
unfolding *moebius-rotation-def moebius-similarity-def*
by (*transfer*, *transfer*, *simp add*: *unitary11-gen-def mat-adj-def mat-cnj-def cis-mult*)

lemma *moebius-rotation-unit-circle-fix* [*simp*]:
shows *moebius-pt* (*moebius-rotation* φ) *u* ∈ *unit-circle-set* ⟷ *u* ∈ *unit-circle-set*
using *moebius-pt-moebius-inv-in-set unit-circle-fix-iff-unit-circle-set*
by *fastforce*

lemma *ex-rotation-mapping-u-to-positive-x-axis*:
assumes *u* ≠ 0_h **and** *u* ≠ ∞_h
shows ∃ φ. *moebius-pt* (*moebius-rotation* φ) *u* ∈ *positive-x-axis*
proof−
from *assms* **obtain** *c* **where** ∗: *u* = *of-complex c*
using *inf-or-of-complex*
by *blast*
have *is-real* (*cis* (− *Arg c*) ∗ *c*) *Re* (*cis* (−*Arg c*) ∗ *c*) > *0*
using ∗ *assms is-real-rot-to-x-axis positive-rot-to-x-axis of-complex-zero-iff*
by *blast*+
thus *?thesis*
using ∗
by (*rule-tac x*=−*Arg c* **in** *exI*) (*simp add*: *positive-x-axis-def circline-set-x-axis*)
qed

lemma *ex-rotation-mapping-u-to-positive-y-axis*:
assumes *u* ≠ 0_h **and** *u* ≠ ∞_h
shows ∃ φ. *moebius-pt* (*moebius-rotation* φ) *u* ∈ *positive-y-axis*
proof−
from *assms* **obtain** *c* **where** ∗: *u* = *of-complex c*
using *inf-or-of-complex*
by *blast*
have *is-imag* (*cis* (*pi*/*2* − *Arg c*) ∗ *c*) *Im* (*cis* (*pi*/*2* − *Arg c*) ∗ *c*) > *0*
using ∗ *assms is-real-rot-to-x-axis positive-rot-to-x-axis of-complex-zero-iff*
by − (*simp*, *simp*, *simp add*: *field-simps*)
thus *?thesis*
using ∗

by (*rule-tac x=pi/2−Arg c* **in** *exI*) (*simp add*: *positive-y-axis-def circline-set-y-axis*)
qed

lemma *wlog-rotation-to-positive-x-axis*:
assumes *in-disc*: $u \in$ *unit-disc* **and** *not-zero*: $u \neq 0_h$
assumes *preserving*: $\bigwedge\varphi\ u.\ [\![u \in \textit{unit-disc};\ u \neq 0_h;\ P\ (\textit{moebius-pt}\ (\textit{moebius-rotation}\ \varphi)\ u)]\!] \Longrightarrow P\ u$
assumes *x-axis*: $\bigwedge x.\ [\![\textit{is-real}\ x;\ 0 < \textit{Re}\ x;\ \textit{Re}\ x < 1]\!] \Longrightarrow P\ (\textit{of-complex}\ x)$
shows $P\ u$
proof−
from *in-disc* **obtain** φ **where** ∗:
moebius-pt (*moebius-rotation* φ) $u \in$ *positive-x-axis*
using *ex-rotation-mapping-u-to-positive-x-axis*[*of u*]
using *inf-notin-unit-disc not-zero*
by *blast*
let *?Mu* = *moebius-pt* (*moebius-rotation* φ) u
have P *?Mu*
proof−
let *?x* = *to-complex ?Mu*
have *?Mu* ∈ *unit-disc ?Mu* ≠ 0_h *?Mu* ≠ ∞_h
using ‹$u \in$ *unit-disc*› ‹$u \neq 0_h$›
by *auto*
hence *is-real* (*to-complex ?Mu*) 0 < *Re ?x Re ?x* < 1
using ∗
unfolding *positive-x-axis-def circline-set-x-axis*
by (*auto simp add*: *cmod-eq-Re*)
thus *?thesis*
using *x-axis*[*of ?x*] ‹*?Mu* ≠ ∞_h›
by *simp*
qed
thus *?thesis*
using *preserving*[*OF in-disc*] *not-zero*
by *simp*
qed

lemma *wlog-rotation-to-positive-x-axis′*:
assumes *not-zero*: $u \neq 0_h$ **and** *not-inf*: $u \neq \infty_h$
assumes *preserving*: $\bigwedge\varphi\ u.\ [\![u \neq 0_h;\ u \neq \infty_h;\ P\ (\textit{moebius-pt}\ (\textit{moebius-rotation}\ \varphi)\ u)]\!] \Longrightarrow P\ u$
assumes *x-axis*: $\bigwedge x.\ [\![\textit{is-real}\ x;\ 0 < \textit{Re}\ x]\!] \Longrightarrow P\ (\textit{of-complex}\ x)$
shows $P\ u$
proof−
from *not-zero* **and** *not-inf* **obtain** φ **where** ∗:
moebius-pt (*moebius-rotation* φ) $u \in$ *positive-x-axis*
using *ex-rotation-mapping-u-to-positive-x-axis*[*of u*]
using *inf-notin-unit-disc*
by *blast*
let *?Mu* = *moebius-pt* (*moebius-rotation* φ) u
have P *?Mu*
proof−
let *?x* = *to-complex ?Mu*
have *?Mu* ≠ 0_h *?Mu* ≠ ∞_h
using ‹$u \neq \infty_h$› ‹$u \neq 0_h$›
by *auto*
hence *is-real* (*to-complex ?Mu*) 0 < *Re ?x*
using ∗
unfolding *positive-x-axis-def circline-set-x-axis*
by (*auto simp add*: *cmod-eq-Re*)
thus *?thesis*
using *x-axis*[*of ?x*] ‹*?Mu* ≠ ∞_h›
by *simp*
qed
thus *?thesis*
using *preserving*[*OF not-zero not-inf*]
by *simp*
qed

lemma *wlog-rotation-to-positive-y-axis*:

assumes *in-disc*: $u \in$ *unit-disc* **and** *not-zero*: $u \neq 0_h$
assumes *preserving*: $\bigwedge\varphi\ u.\ [\![u \in$ *unit-disc*; $u \neq 0_h$; P (*moebius-pt* (*moebius-rotation* φ) $u)]\!] \Longrightarrow P\ u$
assumes *y-axis*: $\bigwedge x.\ [\![$*is-imag* x; $0 <$ *Im* x; *Im* $x < 1]\!] \Longrightarrow P$ (*of-complex* x)
shows $P\ u$
proof−
from *in-disc* **and** *not-zero* **obtain** φ **where** $*$:
moebius-pt (*moebius-rotation* φ) $u \in$ *positive-y-axis*
using *ex-rotation-mapping-u-to-positive-y-axis*[*of u*]
using *inf-notin-unit-disc*
by *blast*
let *?Mu* = *moebius-pt* (*moebius-rotation* φ) u
have P *?Mu*
proof−
let *?y* = *to-complex ?Mu*
have *?Mu* $\in$ *unit-disc ?Mu* $\neq 0_h$ *?Mu* $\neq \infty_h$
using ‹$u \in$ *unit-disc*› ‹$u \neq 0_h$›
by *auto*
hence *is-imag* (*to-complex ?Mu*) $0 <$ *Im ?y Im ?y* < 1
using $*$
unfolding *positive-y-axis-def circline-set-y-axis*
by (*auto simp add*: *cmod-eq-Im*)
thus *?thesis*
using *y-axis*[*of ?y*] ‹*?Mu* $\neq \infty_h$›
by *simp*
qed
thus *?thesis*
using *preserving*[*OF in-disc not-zero*]
by *simp*
qed

9.5 Blaschke factors are unit disc preserving transformations

For a given point a, Blaschke factor transformations are of the form $k \cdot \begin{pmatrix} 1 & -a \\ -\overline{a} & 1 \end{pmatrix}$. It is a disc preserving Möbius transformation that maps the point a to zero (by the symmetry principle, it must map the inverse point of a to infinity).

definition *blaschke-cmat* :: *complex* ⇒ *complex-mat* **where**
[*simp*]: *blaschke-cmat a* = (*if cmod a* ≠ *1 then* (*1*, −*a*, −*cnj a*, *1*) *else eye*)
lift-definition *blaschke-mmat* :: *complex* ⇒ *moebius-mat* **is** *blaschke-cmat*
by *simp*
lift-definition *blaschke* :: *complex* ⇒ *moebius* **is** *blaschke-mmat*
done

lemma *blaschke-0-id* [*simp*]: *blaschke 0* = *id-moebius*
by (*transfer*, *transfer*, *simp*)

lemma *blaschke-a-to-zero* [*simp*]:
assumes *cmod a* ≠ *1*
shows *moebius-pt* (*blaschke a*) (*of-complex a*) = 0_h
using *assms*
by (*transfer*, *transfer*, *simp*)

lemma *blaschke-inv-a-inf* [*simp*]:
assumes *cmod a* ≠ *1*
shows *moebius-pt* (*blaschke a*) (*inversion* (*of-complex a*)) = ∞_h
using *assms*
unfolding *inversion-def*
by (*transfer*, *transfer*) (*simp add*: *vec-cnj-def*, *rule-tac x=1/*(*1* − *a*∗*cnj a*) **in** *exI*, *simp*)

lemma *blaschke-inf* [*simp*]:
assumes *cmod a* < *1* **and** *a* ≠ *0*
shows *moebius-pt* (*blaschke a*) ∞_h = *of-complex* (− *1* / *cnj a*)
using *assms*
by (*transfer*, *transfer*, *simp add*: *complex-mod-sqrt-Re-mult-cnj*)

lemma *blaschke-0-minus-a* [*simp*]:
assumes *cmod a* ≠ *1*
shows *moebius-pt* (*blaschke a*) *0*$_h$ = ~$_h$ (*of-complex a*)
using *assms*
by (*transfer*, *transfer*, *simp*)

lemma *blaschke-unit-circle-fix* [*simp*]:
assumes *cmod a* ≠ *1*
shows *unit-circle-fix* (*blaschke a*)
using *assms*
by (*transfer*, *transfer*) (*simp add*: *unitary11-gen-def mat-adj-def mat-cnj-def*)

lemma *blaschke-unit-disc-fix* [*simp*]:
assumes *cmod a* < *1*
shows *unit-disc-fix* (*blaschke a*)
using *assms*
proof (*transfer*, *transfer*)
fix *a*
assume ∗: *cmod a* < *1*
show *unit-disc-fix-cmat* (*blaschke-cmat a*)
proof (*cases a* = *0*)
case *True*
thus *?thesis*
by (*simp add*: *unitary11-gen-def mat-adj-def mat-cnj-def*)
next
case *False*
hence *Re* (*a* ∗ *cnj a*) < *1*
using ∗
by (*metis complex-mod-sqrt-Re-mult-cnj real-sqrt-lt-1-iff*)
hence *1* / *Re* (*a* ∗ *cnj a*) > *1*
using *False*
by (*smt* (*verit*) *complex-div-gt-0 less-divide-eq-1-pos one-complex.simps*(*1*) *right-inverse-eq*)
hence *Re* (*1* / (*a* ∗ *cnj a*)) > *1*
by (*simp add*: *complex-is-Real-iff*)
thus *?thesis*
by (*simp add*: *unitary11-gen-def mat-adj-def mat-cnj-def*)
qed
qed

lemma *blaschke-unit-circle-fix′*:
assumes *cmod a* ≠ *1*
shows *moebius-circline* (*blaschke a*) *unit-circle* = *unit-circle*
using *assms*
using *blaschke-unit-circle-fix unit-circle-fix-iff*
by *simp*

lemma *blaschke-ounit-circle-fix′*:
assumes *cmod a* < *1*
shows *moebius-ocircline* (*blaschke a*) *ounit-circle* = *ounit-circle*
proof−
have *Re* (*a* ∗ *cnj a*) < *1*
using *assms*
by (*metis complex-mod-sqrt-Re-mult-cnj real-sqrt-lt-1-iff*)
thus *?thesis*
using *assms*
using *blaschke-unit-disc-fix unit-disc-fix-iff-ounit-circle*
by *simp*
qed

lemma *moebius-pt-blaschke* [*simp*]:
assumes *cmod a* ≠ *1* **and** *z* ≠ *1* / *cnj a*
shows *moebius-pt* (*blaschke a*) (*of-complex z*) = *of-complex* ((*z* − *a*) / (*1* − *cnj a* ∗ *z*))
using *assms*
proof (*cases a* = *0*)
case *True*
thus *?thesis*

by *auto*
next
case *False*
thus *?thesis*
using *assms*
apply (*transfer*, *transfer*)
apply (*simp add*: *complex-mod-sqrt-Re-mult-cnj*)
apply (*rule-tac x=1 / (1 − cnj a ∗ z)* **in** *exI*)
apply (*simp add*: *field-simps*)
done
qed

9.5.1 Blaschke factors for a real point a

If the point a is real, the Blaschke factor preserve x-axis and the upper and the lower halfplane.

lemma *blaschke-real-preserve-x-axis* [*simp*]:
assumes *is-real a* **and** *cmod a < 1*
shows *moebius-pt* (*blaschke a*) *z* ∈ *circline-set x-axis* ⟷ *z* ∈ *circline-set x-axis*
proof (*cases a = 0*)
case *True*
thus *?thesis*
by *simp*
next
case *False*
have *cmod a ≠ 1*
using *assms*
by *linarith*
let *?a = of-complex a*
let *?i = inversion ?a*
let *?M = moebius-pt* (*blaschke a*)
have ∗: *?M ?a = 0*$_h$ *?M ?i* = ∞$_h$ *?M 0*$_h$ *= of-complex* (−*a*)
using ‹*cmod a ≠ 1*› *blaschke-a-to-zero*[*of a*] *blaschke-inv-a-inf*[*of a*] *blaschke-0-minus-a*[*of a*]
by *auto*
let *?Mx = moebius-circline* (*blaschke a*) *x-axis*
have *?a* ∈ *circline-set x-axis ?i* ∈ *circline-set x-axis 0*$_h$ ∈ *circline-set x-axis*
using ‹*is-real a*› ‹*a ≠ 0*› *eq-cnj-iff-real*[*of a*]
by *auto*
hence *0*$_h$ ∈ *circline-set ?Mx* ∞$_h$ ∈ *circline-set ?Mx of-complex* (−*a*) ∈ *circline-set ?Mx*
using ∗
apply −
apply (*force simp add*: *image-iff*)+
apply (*simp add*: *image-iff*, *rule-tac x=0*$_h$ **in** *bexI*, *simp-all*)
done
moreover
have *0*$_h$ ∈ *circline-set x-axis* ∞$_h$ ∈ *circline-set x-axis of-complex* (−*a*) ∈ *circline-set x-axis*
using ‹*is-real a*›
by *auto*
moreover
have *of-complex* (−*a*) ≠ *0*$_h$
using ‹*a ≠ 0*›
by *simp*
hence *0*$_h$ ≠ *of-complex* (−*a*)
by *metis*
hence ∃!*H*. *0*$_h$ ∈ *circline-set H* ∧ ∞$_h$ ∈ *circline-set H* ∧ *of-complex* (− *a*) ∈ *circline-set H*
using *unique-circline-set*[*of 0*$_h$ ∞$_h$ *of-complex* (−*a*)] ‹*a ≠ 0*›
by *simp*
ultimately
have *moebius-circline* (*blaschke a*) *x-axis = x-axis*
by *auto*
thus *?thesis*
by (*metis circline-set-moebius-circline-iff*)
qed

lemma *blaschke-real-preserve-sgn-Im* [*simp*]:
assumes *is-real a* **and** *cmod a < 1* **and** *z ≠* ∞$_h$ **and** *z ≠ inversion* (*of-complex a*)
shows *sgn* (*Im* (*to-complex* (*moebius-pt* (*blaschke a*) *z*))) = *sgn* (*Im* (*to-complex z*))

proof (*cases a = 0*)
case *True*
thus *?thesis*
by *simp*
next
case *False*
obtain *z'* **where** *z'*: *z = of-complex z'*
using *inf-or-of-complex*[*of z*] ‹*z ≠ ∞ₕ*›
by *auto*
have *z' ≠ 1 / cnj a*
using *assms z'* ‹*a ≠ 0*›
by (*auto simp add*: *of-complex-inj*)
moreover
have *a * cnj a ≠ 1*
using ‹*cmod a < 1*›
by *auto* (*simp add*: *complex-mod-sqrt-Re-mult-cnj*)
moreover
have *sgn* (*Im* ((*z' − a*) / (*1 − a * z'*))) = *sgn* (*Im z'*)
proof−
have *a * z' ≠ 1*
using ‹*is-real a*› ‹*z' ≠ 1 / cnj a*› ‹*a ≠ 0*› *eq-cnj-iff-real*[*of a*]
by (*simp add*: *field-simps*)
moreover
have *Re* (*1 − a²*) > *0*
using ‹*is-real a*› ‹*cmod a < 1*›
by (*smt* (*verit*) *Re-power2 minus-complex.simps*(*1*) *norm-complex-def one-complex.simps*(*1*) *power2-less-0 real-sqrt-lt-1-iff*)
moreover
have *Im* ((*z' − a*) / (*1 − a * z'*)) = *Re* (((*1 − a²*) * *Im z'*) / (*cmod* (*1 − a*z'*))²)
proof−
have *1 − a * cnj z' ≠ 0*
using ‹*z' ≠ 1 / cnj a*›
by (*metis Im-complex-div-eq-0 complex-cnj-zero-iff diff-eq-diff-eq diff-numeral-special*(*9*) *eq-divide-imp is-real-div mult-not-zero one-complex.simps*(*2*) *zero-neq-one*)
hence *Im* ((*z' − a*) / (*1 − a * z'*)) = *Im* (((*z' − a*) * (*1 − a * cnj z'*)) / ((*1 − a * z'*) * *cnj* (*1 − a * z'*)))
using ‹*is-real a*› *eq-cnj-iff-real*[*of a*]
by *simp*
also have ... = *Im* ((*z' − a − a * z' * cnj z' + a² * cnj z'*) / (*cmod* (*1 − a*z'*))²)
unfolding *complex-mult-cnj-cmod*
by (*simp add*: *power2-eq-square field-simps*)
finally show *?thesis*
using ‹*is-real a*›
by (*simp add*: *field-simps*)
qed
moreover
have *0 < (1 − (Re a)²) * Im z' / (cmod (1 − a * z'))² ⟹ Im z' > 0*
using ‹*is-real a*› ‹*0 < Re* (*1 − a²*)›
by (*smt* (*verit*) *Re-power-real divide-le-0-iff minus-complex.simps*(*1*) *not-sum-power2-lt-zero one-complex.simps*(*1*) *zero-less-mult-pos*)
ultimately
show *?thesis*
unfolding *sgn-real-def*
using ‹*cmod a < 1*› ‹*a * z' ≠ 1*› ‹*is-real a*›
by (*auto simp add*: *cmod-eq-Re*)
qed
ultimately
show *?thesis*
using *assms z' moebius-pt-blaschke*[*of a z'*] ‹*is-real a*› *eq-cnj-iff-real*[*of a*]
by *simp*
qed

lemma *blaschke-real-preserve-sgn-arg* [*simp*]:
assumes *is-real a* **and** *cmod a < 1* **and** *z ∉ circline-set x-axis*
shows *sgn* (*Arg* (*to-complex* (*moebius-pt* (*blaschke a*) *z*))) = *sgn* (*Arg* (*to-complex z*))
proof−
have *z ≠ ∞ₕ*
using *assms*

using *special-points-on-x-axis''*(*3*) **by** *blast*
moreover
have $z \neq$ *inversion* (*of-complex a*)
using *assms*
by (*metis calculation circline-equation-x-axis circline-set-x-axis-I conjugate-of-complex inversion-of-complex inversion-sym is-real-inversion o-apply of-complex-zero reciprocal-zero to-complex-of-complex*)
ultimately
show *?thesis*
using *blaschke-real-preserve-sgn-Im*[*OF assms*(*1*) *assms*(*2*), *of z*]
by (*smt* (*verit*) *arg-Im-sgn assms*(*3*) *circline-set-x-axis-I norm-sgn of-complex-to-complex*)
qed

9.5.2 Inverse Blaschke transform

definition *inv-blaschke-cmat* :: *complex* ⇒ *complex-mat* **where**
[*simp*]: *inv-blaschke-cmat a* = (*if cmod a* ≠ *1 then* (*1*, *a*, *cnj a*, *1*) *else eye*)
lift-definition *inv-blaschke-mmat* :: *complex* ⇒ *moebius-mat* **is** *inv-blaschke-cmat*
by *simp*
lift-definition *inv-blaschke* :: *complex* ⇒ *moebius* **is** *inv-blaschke-mmat*
done

lemma *inv-blaschke-neg* [*simp*]: *inv-blaschke a* = *blaschke* (−*a*)
by (*transfer*, *transfer*) *simp*

lemma *inv-blaschke*:
assumes *cmod a* ≠ *1*
shows *blaschke a* + *inv-blaschke a* = *0*
apply *simp*
apply (*transfer*, *transfer*)
by *auto* (*rule-tac x*=*1*/(*1* − *a*∗*cnj a*) **in** *exI*, *simp*)

lemma *ex-unit-disc-fix-mapping-u-to-zero*:
assumes *u* ∈ *unit-disc*
shows ∃ *M*. *unit-disc-fix M* ∧ *moebius-pt M u* = 0_h
proof−
from *assms* **obtain** *c* **where** ∗: *u* = *of-complex c*
by (*metis inf-notin-unit-disc inf-or-of-complex*)
hence *cmod c* < *1*
using *assms unit-disc-iff-cmod-lt-1*
by *simp*
thus *?thesis*
using ∗
by (*rule-tac x*=*blaschke c* **in** *exI*)
(*smt* (*verit*) *blaschke-a-to-zero blaschke-ounit-circle-fix' unit-disc-fix-iff-ounit-circle*)
qed

lemma *wlog-zero*:
assumes *in-disc*: *u* ∈ *unit-disc*
assumes *preserving*: ⋀ *a u*. ⟦*u* ∈ *unit-disc*; *cmod a* < *1*; *P* (*moebius-pt* (*blaschke a*) *u*)⟧ ⟹ *P u*
assumes *zero*: *P* 0_h
shows *P u*
proof−
have ∗: *moebius-pt* (*blaschke* (*to-complex u*)) *u* = 0_h
by (*smt* (*verit*) *blaschke-a-to-zero in-disc inf-notin-unit-disc of-complex-to-complex unit-disc-iff-cmod-lt-1*)
thus *?thesis*
using *preserving*[*of u to-complex u*] *in-disc zero*
using *inf-or-of-complex*[*of u*]
by *auto*
qed

lemma *wlog-real-zero*:
assumes *in-disc*: *u* ∈ *unit-disc* **and** *real*: *is-real* (*to-complex u*)
assumes *preserving*: ⋀ *a u*. ⟦*u* ∈ *unit-disc*; *is-real a*; *cmod a* < *1*; *P* (*moebius-pt* (*blaschke a*) *u*)⟧ ⟹ *P u*
assumes *zero*: *P* 0_h
shows *P u*
proof−

```
  have ∗: moebius-pt (blaschke (to-complex u)) u = 0_h
    by (smt (verit) blaschke-a-to-zero in-disc inf-notin-unit-disc of-complex-to-complex unit-disc-iff-cmod-lt-1)
  thus ?thesis
    using preserving[of u to-complex u] in-disc zero real
    using inf-or-of-complex[of u]
    by auto
qed

lemma unit-disc-fix-transitive:
  assumes in-disc: u ∈ unit-disc and u' ∈ unit-disc
  shows ∃ M. unit-disc-fix M ∧ moebius-pt M u = u'
proof−
  have ∀ u ∈ unit-disc. ∃ M. unit-disc-fix M ∧ moebius-pt M u = u' (is ?P u')
  proof (rule wlog-zero)
    show u' ∈ unit-disc by fact
  next
    show ?P 0_h
      by (simp add: ex-unit-disc-fix-mapping-u-to-zero)
  next
    fix a u
    assume cmod a < 1 and ∗: ?P (moebius-pt (blaschke a) u)
    show ?P u
    proof
      fix u'
      assume u' ∈ unit-disc
      then obtain M' where unit-disc-fix M' moebius-pt M' u' = moebius-pt (blaschke a) u
        using ∗
        by auto
      thus ∃ M. unit-disc-fix M ∧ moebius-pt M u' = u
        using ‹cmod a < 1› blaschke-unit-disc-fix[of a]
        using unit-disc-fix-moebius-comp[of − blaschke a M']
        using unit-disc-fix-moebius-inv[of blaschke a]
        by (rule-tac x=(− (blaschke a)) + M' in exI, simp)
    qed
  qed
  thus ?thesis
    using assms
    by auto
qed
```

9.6 Decomposition of unit disc preserving Möbius transforms

Each transformation preserving unit disc can be decomposed to a rotation around the origin and a Blaschke factors that maps a point within the unit disc to zero.

```
lemma unit-disc-fix-decompose-blaschke-rotation:
  assumes unit-disc-fix M
  shows ∃ k φ. cmod k < 1 ∧ M = moebius-rotation φ + blaschke k
  using assms
  unfolding moebius-rotation-def moebius-similarity-def
proof (simp, transfer, transfer)
  fix M
  assume ∗: mat-det M ≠ 0 unit-disc-fix-cmat M
  then obtain k a b :: complex where
    ∗∗: k ≠ 0 mat-det (a, b, cnj b, cnj a) ≠ 0 M = k ∗_sm (a, b, cnj b, cnj a)
    using unitary11-gen-iff[of M]
    by auto
  have a ≠ 0
    using ∗ ∗∗
    by auto
  then obtain a' k' φ
    where ∗∗∗: k' ≠ 0 ∧ a' ∗ cnj a' ≠ 1 ∧ M = k' ∗_sm (cis φ, 0, 0, 1) ∗_mm (1, − a', − cnj a', 1)
    using ∗∗ unitary11-gen-cis-blaschke[of k M a b]
    by auto blast
  have a' = 0 ∨ 1 < 1 / (cmod a')²
    using ∗ ∗∗∗ complex-mult-cnj-cmod[of a']
```

```
    by simp
  hence cmod a' < 1
    by (smt (verit) less-divide-eq-1-pos norm-zero one-less-power one-power2 pos2)
  thus ∃k. cmod k < 1 ∧
          (∃φ. moebius-cmat-eq M (moebius-comp-cmat (mk-moebius-cmat (cis φ) 0 0 1) (blaschke-cmat k)))
    using ***
    apply (rule-tac x=a' in exI)
    apply simp
    apply (rule-tac x=φ in exI)
    apply simp
    apply (rule-tac x=1/k' in exI)
    by auto
qed

lemma wlog-unit-disc-fix:
  assumes unit-disc-fix M
  assumes b: ⋀ k. cmod k < 1 ⟹ P (blaschke k)
  assumes r: ⋀ φ. P (moebius-rotation φ)
  assumes comp: ⋀M1 M2. ⟦unit-disc-fix M1; P M1; unit-disc-fix M2; P M2⟧ ⟹ P (M1 + M2)
  shows P M
  using assms
  using unit-disc-fix-decompose-blaschke-rotation[OF assms(1)]
  using blaschke-unit-disc-fix
  by auto

lemma ex-unit-disc-fix-to-zero-positive-x-axis:
  assumes u ∈ unit-disc and v ∈ unit-disc and u ≠ v
  shows ∃ M. unit-disc-fix M ∧
          moebius-pt M u = 0_h ∧ moebius-pt M v ∈ positive-x-axis
proof−
  from assms obtain B where
    ∗: unit-disc-fix B moebius-pt B u = 0_h
    using ex-unit-disc-fix-mapping-u-to-zero
    by blast

  let ?v = moebius-pt B v
  have ?v ∈ unit-disc
    using ‹v ∈ unit-disc› ∗
    by auto
  hence ?v ≠ ∞_h
    using inf-notin-unit-disc by auto
  have ?v ≠ 0_h
    using ‹u ≠ v› ∗
    by (metis moebius-pt-invert)

  obtain R where
    unit-disc-fix R
    moebius-pt R 0_h = 0_h moebius-pt R ?v ∈ positive-x-axis
    using ex-rotation-mapping-u-to-positive-x-axis[of ?v] ‹?v ≠ 0_h› ‹?v ≠ ∞_h›
    using moebius-pt-rotation-inf-iff moebius-pt-moebius-rotation-zero unit-disc-fix-rotation
    by blast
  thus ?thesis
    using ∗ moebius-comp[of R B, symmetric]
    using unit-disc-fix-moebius-comp
    by (rule-tac x=R + B in exI) (simp add: comp-def)
qed

lemma wlog-x-axis:
  assumes in-disc: u ∈ unit-disc v ∈ unit-disc
  assumes preserved: ⋀ M u v. ⟦unit-disc-fix M; u ∈ unit-disc; v ∈ unit-disc; P (moebius-pt M u) (moebius-pt M v)⟧
⟹ P u v
  assumes axis: ⋀ x. ⟦is-real x; 0 ≤ Re x;  Re x < 1⟧ ⟹ P 0_h (of-complex x)
  shows P u v
proof (cases u = v)
  case True
  have P u u (is ?Q u)
```

```
  proof (rule wlog-zero[where P=?Q])
    show u ∈ unit-disc
      by fact
  next
    show ?Q 0_h
      using axis[of 0]
      by simp
  next
    fix a u
    assume u ∈ unit-disc cmod a < 1 ?Q (moebius-pt (blaschke a) u)
    thus ?Q u
      using preserved[of blaschke a u u]
      using blaschke-unit-disc-fix[of a]
      by simp
  qed
  thus ?thesis
    using True
    by simp
next
  case False
  from in-disc obtain M where
    *: 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 False
    by auto
  then obtain x where **: moebius-pt M v = of-complex x is-real x
    unfolding positive-x-axis-def circline-set-x-axis
    by auto
  moreover
  have of-complex x ∈ unit-disc
    using ‹unit-disc-fix M› ‹v ∈ unit-disc› **
    using unit-disc-fix-discI
    by fastforce
  hence 0 < Re x Re x < 1
    using ‹moebius-pt M v ∈ positive-x-axis› **
    by (auto simp add: positive-x-axis-def cmod-eq-Re)
  ultimately
  have P 0_h (of-complex x)
    using ‹is-real x› axis
    by auto
  thus ?thesis
    using preserved[OF *(1) assms(1−2)] *(2) **(1)
    by simp
qed

lemma wlog-positive-x-axis:
  assumes in-disc: u ∈ unit-disc v ∈ unit-disc u ≠ v
  assumes preserved: ⋀ M u v. ⟦unit-disc-fix M; u ∈ unit-disc; v ∈ unit-disc; u ≠ v; P (moebius-pt M u) (moebius-pt
M v)⟧ ⟹ P u v
  assumes axis: ⋀ x. ⟦is-real x; 0 < Re x;  Re x < 1⟧ ⟹ P 0_h (of-complex x)
  shows P u v
proof−
  have u ≠ v ⟶ P u v (is ?Q u v)
  proof (rule wlog-x-axis)
    show u ∈ unit-disc v ∈ unit-disc
      by fact+
  next
    fix M u v
    assume unit-disc-fix M u ∈ unit-disc v ∈ unit-disc
          ?Q (moebius-pt M u) (moebius-pt M v)
    thus ?Q u v
      using preserved[of M u v]
      using moebius-pt-invert
      by blast
  next
    fix x
    assume is-real x 0 ≤ Re x Re x < 1
```

thus *?Q* 0_h (*of-complex x*)
using *axis*[*of x*] *of-complex-zero-iff*[*of x*] *complex.expand*[*of x 0*]
by *fastforce*
qed
thus *?thesis*
using ‹$u \neq v$›
by *simp*
qed

9.7 All functions that fix the unit disc

It can be proved that continuous functions that fix the unit disc are either actions of Möbius transformations that fix the unit disc (homographies), or are compositions of actions of Möbius transformations that fix the unit disc and the conjugation (antihomographies). We postulate this as a definition, but it this characterisation could also be formally shown (we do not need this for our further applications).

definition *unit-disc-fix-f* **where**
unit-disc-fix-f f $\longleftrightarrow$
($\exists$ *M*. *unit-disc-fix M* $\wedge$ (*f* = *moebius-pt M* $\vee$ *f* = *moebius-pt M* $\circ$ *conjugate*))

Unit disc fixing functions really fix unit disc.

lemma *unit-disc-fix-f-unit-disc*:
assumes *unit-disc-fix-f M*
shows *M* ‘ *unit-disc* = *unit-disc*
using *assms*
unfolding *unit-disc-fix-f-def*
using *image-comp*
by *force*

Actions of unit disc fixing Möbius transformations (unit disc fixing homographies) are unit disc fixing functions.

lemma *unit-disc-fix-f-moebius-pt* [*simp*]:
assumes *unit-disc-fix M*
shows *unit-disc-fix-f* (*moebius-pt M*)
using *assms*
unfolding *unit-disc-fix-f-def*
by *auto*

Compositions of unit disc fixing Möbius transformations and conjugation (unit disc fixing antihomographies) are unit disc fixing functions.

lemma *unit-disc-fix-conjugate-moebius* [*simp*]:
assumes *unit-disc-fix M*
shows *unit-disc-fix* (*conjugate-moebius M*)
proof −
have $\bigwedge$*a aa ab b*. ⟦*1 < Re* (*a * b / (aa * ab)*); ¬ *1 < Re* (*cnj a * cnj b / (cnj aa * cnj ab)*)⟧ ⟹ *aa = 0*
by (*metis cnj.simps*(*1*) *complex-cnj-divide complex-cnj-mult*)
thus *?thesis*
using *assms*
by (*transfer*, *transfer*)
(*auto simp add*: *mat-cnj-def unitary11-gen-def mat-adj-def field-simps*)
qed

lemma *unit-disc-fix-conjugate-comp-moebius* [*simp*]:
assumes *unit-disc-fix M*
shows *unit-disc-fix-f* (*conjugate* ∘ *moebius-pt M*)
using *assms*
apply (*subst conjugate-moebius*)
apply (*simp add*: *unit-disc-fix-f-def*)
apply (*rule-tac x*=*conjugate-moebius M* **in** *exI*, *simp*)
done

Uniti disc fixing functions form a group under function composition.

lemma *unit-disc-fix-f-comp* [*simp*]:
assumes *unit-disc-fix-f f1* **and** *unit-disc-fix-f f2*
shows *unit-disc-fix-f* (*f1* ∘ *f2*)
using *assms*

apply (*subst* (*asm*) *unit-disc-fix-f-def*)
apply (*subst* (*asm*) *unit-disc-fix-f-def*)
proof *safe*
fix *M M'*
assume *unit-disc-fix M unit-disc-fix M'*
thus *unit-disc-fix-f* (*moebius-pt M* ∘ *moebius-pt M'*)
unfolding *unit-disc-fix-f-def*
by (*rule-tac x*=*M* + *M'* **in** *exI*) *auto*
next
fix *M M'*
assume *unit-disc-fix M unit-disc-fix M'*
thus *unit-disc-fix-f* (*moebius-pt M* ∘ (*moebius-pt M'* ∘ *conjugate*))
unfolding *unit-disc-fix-f-def*
by (*subst comp-assoc*[*symmetric*])+
(*rule-tac x*=*M* + *M'* **in** *exI*, *auto*)
next
fix *M M'*
assume *unit-disc-fix M unit-disc-fix M'*
thus *unit-disc-fix-f* ((*moebius-pt M* ∘ *conjugate*) ∘ *moebius-pt M'*)
unfolding *unit-disc-fix-f-def*
by (*subst comp-assoc*, *subst conjugate-moebius*, *subst comp-assoc*[*symmetric*])+
(*rule-tac x*=*M* + *conjugate-moebius M'* **in** *exI*, *auto*)
next
fix *M M'*
assume *unit-disc-fix M unit-disc-fix M'*
thus *unit-disc-fix-f* ((*moebius-pt M* ∘ *conjugate*) ∘ (*moebius-pt M'* ∘ *conjugate*))
apply (*subst comp-assoc*[*symmetric*], *subst comp-assoc*)
apply (*subst conjugate-moebius*, *subst comp-assoc*, *subst comp-assoc*)
apply (*simp add*: *unit-disc-fix-f-def*)
apply (*rule-tac x*=*M* + *conjugate-moebius M'* **in** *exI*, *auto*)
done
qed

lemma *unit-disc-fix-f-inv*:
assumes *unit-disc-fix-f M*
shows *unit-disc-fix-f* (*inv M*)
using *assms*
apply (*subst* (*asm*) *unit-disc-fix-f-def*)
proof *safe*
fix *M*
assume *unit-disc-fix M*
have *inv* (*moebius-pt M*) = *moebius-pt* (−*M*)
by (*rule ext*) (*simp add*: *moebius-inv*)
thus *unit-disc-fix-f* (*inv* (*moebius-pt M*))
using ‹*unit-disc-fix M*›
unfolding *unit-disc-fix-f-def*
by (*rule-tac x*=−*M* **in** *exI*, *simp*)
next
fix *M*
assume *unit-disc-fix M*
have *inv* (*moebius-pt M* ∘ *conjugate*) = *conjugate* ∘ *inv* (*moebius-pt M*)
by (*subst o-inv-distrib*, *simp-all*)
also have ... = *conjugate* ∘ (*moebius-pt* (−*M*))
using *moebius-inv*
by *auto*
also have ... = *moebius-pt* (*conjugate-moebius* (−*M*)) ∘ *conjugate*
by (*simp add*: *conjugate-moebius*)
finally
show *unit-disc-fix-f* (*inv* (*moebius-pt M* ∘ *conjugate*))
using ‹*unit-disc-fix M*›
unfolding *unit-disc-fix-f-def*
by (*rule-tac x*=*conjugate-moebius* (−*M*) **in** *exI*, *simp*)
qed

9.7.1 Action of unit disc fixing functions on circlines

definition *unit-disc-fix-f-circline* **where**
unit-disc-fix-f-circline f H =
(if ∃ M. unit-disc-fix M ∧ f = moebius-pt M then
moebius-circline (THE M. unit-disc-fix M ∧ f = moebius-pt M) H
else if ∃ M. unit-disc-fix M ∧ f = moebius-pt M ∘ conjugate then
(moebius-circline (THE M. unit-disc-fix M ∧ f = moebius-pt M ∘ conjugate) ∘ conjugate-circline) H
else
H)

lemma *unique-moebius-pt-conjugate*:
assumes *moebius-pt M1 ∘ conjugate = moebius-pt M2 ∘ conjugate*
shows *M1 = M2*
proof−
from *assms* **have** *moebius-pt M1 = moebius-pt M2*
using *conjugate-conjugate-comp rewriteL-comp-comp2* **by** *fastforce*
thus *?thesis*
using *unique-moebius-pt*
by *auto*
qed

lemma *unit-disc-fix-f-circline-direct*:
assumes *unit-disc-fix M* **and** *f = moebius-pt M*
shows *unit-disc-fix-f-circline f H = moebius-circline M H*
proof−
have *M = (THE M. unit-disc-fix M ∧ f = moebius-pt M)*
using *assms*
using *theI-unique*[*of* λ *M. unit-disc-fix M ∧ f = moebius-pt M M*]
using *unique-moebius-pt*[*of M*]
by *auto*
thus *?thesis*
using *assms*
unfolding *unit-disc-fix-f-circline-def*
by *auto*
qed

lemma *unit-disc-fix-f-circline-indirect*:
assumes *unit-disc-fix M* **and** *f = moebius-pt M ∘ conjugate*
shows *unit-disc-fix-f-circline f H = ((moebius-circline M) ∘ conjugate-circline) H*
proof−
have *¬ (∃ M. unit-disc-fix M ∧ f = moebius-pt M)*
using *assms homography-antihomography-exclusive*[*of f*]
unfolding *is-homography-def is-antihomography-def is-moebius-def*
by *auto*
moreover
have *M = (THE M. unit-disc-fix M ∧ f = moebius-pt M ∘ conjugate)*
using *assms*
using *theI-unique*[*of* λ *M. unit-disc-fix M ∧ f = moebius-pt M ∘ conjugate M*]
using *unique-moebius-pt-conjugate*[*of M*]
by *auto*
ultimately
show *?thesis*
using *assms*
unfolding *unit-disc-fix-f-circline-def*
by *metis*
qed

Disc automorphisms - it would be nice to show that there are no disc automorphisms other than unit disc fixing homographies and antihomographies, but this part of the theory is not yet developed.

definition *is-disc-aut* **where** *is-disc-aut f ⟷ bij-betw f unit-disc unit-disc*

end

10 Riemann sphere

The extended complex plane $\mathbb{C}P^1$ can be identified with a Riemann (unit) sphere Σ by means of stereographic projection. The sphere is projected from its north pole N to the xOy plane (identified with $\mathbb{C}$). This projection establishes a bijective map *sp* between $\Sigma \setminus \{N\}$ and the finite complex plane $\mathbb{C}$. The infinite point is defined as the image of N.

theory *Riemann-Sphere*
imports *Homogeneous-Coordinates Circlines HOL−Analysis.Product-Vector*
begin

Coordinates in $\mathbb{R}^3$

type-synonym *R3* = *real* × *real* × *real*

Type of points of Σ

abbreviation *unit-sphere* **where**
unit-sphere ≡ {(*x*::*real*, *y*::*real*, *z*::*real*). *x*∗*x* + *y*∗*y* + *z*∗*z* = *1*}

typedef *riemann-sphere* = *unit-sphere*
by (*rule-tac x*=(*1*, *0*, *0*) **in** *exI*) *simp*

setup-lifting *type-definition-riemann-sphere*

lemma *sphere-bounds′*:
assumes *x*∗*x* + *y*∗*y* + *z*∗*z* = (*1*::*real*)
shows −*1* ≤ *x* ∧ *x* ≤ *1*
proof−
from *assms* **have** *x*∗*x* ≤ *1*
by (*smt* (*verit*) *real-minus-mult-self-le*)
hence $x^2 \le 1^2$ $(-x)^2 \le 1^2$
by (*auto simp add*: *power2-eq-square*)
show −*1* ≤ *x* ∧ *x* ≤ *1*
proof (*cases x* ≥ *0*)
case *True*
thus *?thesis*
using ‹$x^2 \le 1^2$›
by (*smt* (*verit*) *power2-le-imp-le*)
next
case *False*
thus *?thesis*
using ‹$(-x)^2 \le 1^2$›
by (*smt* (*verit*) *power2-le-imp-le*)
qed
qed

lemma *sphere-bounds*:
assumes *x*∗*x* + *y*∗*y* + *z*∗*z* = (*1*::*real*)
shows −*1* ≤ *x* ∧ *x* ≤ *1* −*1* ≤ *y* ∧ *y* ≤ *1* −*1* ≤ *z* ∧ *z* ≤ *1*
using *assms*
using *sphere-bounds′*[*of x y z*] *sphere-bounds′*[*of y x z*] *sphere-bounds′*[*of z x y*]
by (*auto simp add*: *field-simps*)

10.1 Parametrization of the unit sphere in polar coordinates

lemma *sphere-params-on-sphere*:
fixes *α β* :: *real*
assumes *x* = *cos α* ∗ *cos β* **and** *y* = *cos α* ∗ *sin β z* = *sin α*
shows *x*∗*x* + *y*∗*y* + *z*∗*z* = *1*
proof−
have *x*∗*x* + *y*∗*y* = (*cos α* ∗ *cos α*) ∗ (*cos β* ∗ *cos β*) + (*cos α* ∗ *cos α*) ∗ (*sin β* ∗ *sin β*)
using *assms*
by *simp*
hence *x*∗*x* + *y*∗*y* = *cos α* ∗ *cos α*
using *sin-cos-squared-add3*[*of β*]
by (*subst* (*asm*) *distrib-left*[*symmetric*]) (*simp add*: *field-simps*)
thus *?thesis*

using *assms*
using *sin-cos-squared-add3*[*of* α]
by *simp*
qed

lemma *sphere-params*:
fixes $x\ y\ z$ :: *real*
assumes $x*x + y*y + z*z = 1$
shows $x = cos\ (arcsin\ z) * cos\ (atan2\ y\ x) \wedge y = cos\ (arcsin\ z) * sin\ (atan2\ y\ x) \wedge z = sin\ (arcsin\ z)$
proof (*cases* $z{=}1 \vee z = -1$)
case *True*
hence $x = 0 \wedge y = 0$
using *assms*
by *auto*
thus *?thesis*
using ‹$z = 1 \vee z = -1$›
by (*auto simp add*: *cos-arcsin*)
next
case *False*
hence $x \neq 0 \vee y \neq 0$
using *assms*
by (*auto simp add*: *square-eq-1-iff*)
thus *?thesis*
using *real-sqrt-unique*[*of* $y\ 1 - z*z$]
using *real-sqrt-unique*[*of* $-y\ 1 - z*z$]
using *sphere-bounds*[*OF assms*] *assms*
by (*auto simp add*: *cos-arcsin cos-arctan sin-arctan power2-eq-square field-simps real-sqrt-divide atan2-def*)
qed

lemma *ex-sphere-params*:
assumes $x*x + y*y + z*z = 1$
shows $\exists\ \alpha\ \beta.\ x = cos\ \alpha * cos\ \beta \wedge y = cos\ \alpha * sin\ \beta \wedge z = sin\ \alpha \wedge -pi\ /\ 2 \leq \alpha \wedge \alpha \leq pi\ /\ 2 \wedge -pi \leq \beta \wedge \beta < pi$
using *assms arcsin-bounded*[*of z*] *sphere-bounds*[*of x y z*]
by (*rule-tac* $x{=}arcsin\ z$ **in** *exI*, *rule-tac* $x{=}atan2\ y\ x$ **in** *exI*) (*simp add*: *sphere-params arcsin-bounded atan2-bounded*)

10.2 Stereographic and inverse stereographic projection

Stereographic projection

definition *stereographic-r3-cvec* :: *R3* ⇒ *complex-vec* **where**
[*simp*]: *stereographic-r3-cvec M* = (*let* $(x,\ y,\ z) =$ *M in*
(*if* $(x,\ y,\ z) \neq (0,\ 0,\ 1)$ *then*
$(x + \mathrm{i} * y,\ cor\ (1 - z))$
else
$(1,\ 0)$
))

lift-definition *stereographic-r3-hcoords* :: *R3* ⇒ *complex-homo-coords* **is** *stereographic-r3-cvec*
by (*auto split*: *if-split-asm simp add*: *cor-eq-0*)

lift-definition *stereographic* :: *riemann-sphere* ⇒ *complex-homo* **is** *stereographic-r3-hcoords*
done

Inverse stereographic projection

definition *inv-stereographic-cvec-r3* :: *complex-vec* ⇒ *R3* **where** [*simp*]:
inv-stereographic-cvec-r3 z = (
let $(z1,\ z2) = z$
in if $z2 = 0$ *then*
$(0,\ 0,\ 1)$
else
let $z = z1/z2$;
$X = Re\ (2*z\ /\ (1 + z*cnj\ z))$;
$Y = Im\ (2*z\ /\ (1 + z*cnj\ z))$;
$Z = ((cmod\ z)^2 - 1)\ /\ (1 + (cmod\ z)^2)$
in $(X,\ Y,\ Z)$)

lemma *Re-stereographic*:
shows *Re* (*2* ∗ *z* / (*1* + *z* ∗ *cnj z*)) = *2* ∗ *Re z* / (*1* + (*cmod z*)^2)
using *one-plus-square-neq-zero*
by (*subst complex-mult-cnj-cmod*, *subst Re-divide-real*) (*auto simp add*: *power2-eq-square*)

lemma *Im-stereographic*:
shows *Im* (*2* ∗ *z* / (*1* + *z* ∗ *cnj z*)) = *2* ∗ *Im z* / (*1* + (*cmod z*)^2)
using *one-plus-square-neq-zero*
by (*subst complex-mult-cnj-cmod*, *subst Im-divide-real*) (*auto simp add*: *power2-eq-square*)

lemma *inv-stereographic-on-sphere*:
assumes *X* = *Re* (*2*∗*z* / (*1* + *z*∗*cnj z*)) **and** *Y* = *Im* (*2*∗*z* / (*1* + *z*∗*cnj z*)) **and** *Z* = ((*cmod z*)^2 − *1*) / (*1* + (*cmod z*)^2)
shows *X*∗*X* + *Y*∗*Y* + *Z*∗*Z* = *1*
proof−
have *1* + (*cmod z*)^2 ≠ *0*
by (*smt* (*verit*) *power2-less-0*)
thus *?thesis*
using *assms*
by (*simp add*: *Re-stereographic Im-stereographic*)
(*cases z*, *simp add*: *power2-eq-square real-sqrt-mult*[*symmetric*] *add-divide-distrib*[*symmetric*], *simp add*: *complex-norm power2-eq-square field-simps*)
qed

lift-definition *inv-stereographic-hcoords-r3* :: *complex-homo-coords* ⇒ *R3* **is** *inv-stereographic-cvec-r3*
done

lift-definition *inv-stereographic* :: *complex-homo* ⇒ *riemann-sphere* **is** *inv-stereographic-hcoords-r3*
proof *transfer*
fix *v v′*
assume *1*: *v* ≠ *vec-zero v′* ≠ *vec-zero v* ≈$_v$ *v′*
obtain *v1 v2 v′1 v′2* **where** ∗: *v* = (*v1*, *v2*) *v′* = (*v′1*, *v′2*)
by (*cases v*, *cases v′*, *auto*)
obtain *x y z* **where**
∗∗: *inv-stereographic-cvec-r3 v* = (*x*, *y*, *z*)
by (*cases inv-stereographic-cvec-r3 v*, *blast*)
have *inv-stereographic-cvec-r3 v* ∈ *unit-sphere*
proof (*cases v2* = *0*)
case *True*
thus *?thesis*
using ∗
by *simp*
next
case *False*
thus *?thesis*
using ∗ ∗∗ *inv-stereographic-on-sphere*[*of x v1* / *v2 y z*]
by (*simp add*: *norm-divide*)
qed
moreover
have *inv-stereographic-cvec-r3 v* = *inv-stereographic-cvec-r3 v′*
using *1* ∗ ∗∗
by (*auto split*: *if-split if-split-asm*)
ultimately
show *inv-stereographic-cvec-r3 v* ∈ *unit-sphere* ∧
inv-stereographic-cvec-r3 v = *inv-stereographic-cvec-r3 v′*
by *simp*
qed

North pole

definition *North-R3* :: *R3* **where**
[*simp*]: *North-R3* = (*0*, *0*, *1*)
lift-definition *North* :: *riemann-sphere* **is** *North-R3*
by *simp*

lemma *stereographic-North*:

shows *stereographic* $x = \infty_h \longleftrightarrow x = North$
by (*transfer*, *transfer*, *auto split*: *if-split-asm*)

Stereographic and inverse stereographic projection are mutually inverse.

lemma *stereographic-inv-stereographic'*:
assumes
z: $z = z1/z2$ **and** $z2 \neq 0$ **and**
X: $X = Re\ (2{*}z\ /\ (1 + z{*}cnj\ z))$ **and** Y: $Y = Im\ (2{*}z\ /\ (1 + z{*}cnj\ z))$ **and** Z: $Z = ((cmod\ z)^2 - 1)\ /\ (1 + (cmod\ z)^2)$
shows $\exists\ k.\ k \neq 0 \wedge (X + \mathrm{i}{*}Y,\ complex\text{-}of\text{-}real\ (1 - Z)) = k *_{sv} (z1,\ z2)$
proof−
have $1 + (cmod\ z)^2 \neq 0$
by (*metis one-power2 sum-power2-eq-zero-iff zero-neq-one*)
hence $(1 - Z) = 2\ /\ (1 + (cmod\ z)^2)$
using Z
by (*auto simp add*: *field-simps*)
hence $cor\ (1 - Z) = 2\ /\ cor\ (1 + (cmod\ z)^2)$
by *auto*
moreover
have $X = 2 * Re(z)\ /\ (1 + (cmod\ z)^2)$
using X
by (*simp add*: *Re-stereographic*)
have $Y = 2 * Im(z)\ /\ (1 + (cmod\ z)^2)$
using Y
by (*simp add*: *Im-stereographic*)
have $X + \mathrm{i}{*}Y = 2 * z\ /\ cor\ (1 + (cmod\ z)^2)$
using ‹$1 + (cmod\ z)^2 \neq 0$›
unfolding *Complex-eq*[*of X Y*, *symmetric*]
by (*subst* ‹$X = 2{*}Re(z)\ /\ (1 + (cmod\ z)^2)$›, *subst* ‹$Y = 2{*}Im(z)\ /\ (1 + (cmod\ z)^2)$›, *simp add*: *Complex-scale4 Complex-scale1*)
moreover
have $1 + (cor\ (cmod\ (z1\ /\ z2)))^2 \neq 0$
by (*rule one-plus-square-neq-zero*)
ultimately
show *?thesis*
using ‹$z2 \neq 0$› ‹$1 + (cmod\ z)^2 \neq 0$›
by (*simp*, *subst z*)+
(*rule-tac x*=$(2\ /\ (1 + (cor\ (cmod\ (z1\ /\ z2)))^2))\ /\ z2$ **in** *exI*, *auto*)
qed

lemma *stereographic-inv-stereographic* [*simp*]:
shows *stereographic* (*inv-stereographic w*) = *w*
proof−
have *w* = *stereographic* (*inv-stereographic w*)
proof (*transfer*, *transfer*)
fix *w*
assume *w* ≠ *vec-zero*
obtain *w1 w2* **where** ∗: *w* = (*w1*, *w2*)
by (*cases w*, *auto*)
obtain *x y z* **where** ∗∗: *inv-stereographic-cvec-r3 w* = (*x*, *y*, *z*)
by (*cases inv-stereographic-cvec-r3 w*, *blast*)
show *w* ≈*v* *stereographic-r3-cvec* (*inv-stereographic-cvec-r3 w*)
using ‹*w* ≠ *vec-zero*› *stereographic-inv-stereographic'*[*of w1*/*w2 w1 w2 x y z*] ∗ ∗∗
by (*auto simp add*: *split-def Let-def split*: *if-split-asm*)
qed
thus *?thesis*
by *simp*
qed

Stereographic projection is bijective function

lemma *bij-stereographic*:
shows *bij stereographic*
unfolding *bij-def inj-on-def surj-def*
proof (*safe*)
fix *a b*
assume *stereographic a* = *stereographic b*

```
  thus a = b
  proof (transfer, transfer)
    fix a b :: R3
    obtain xa ya za xb yb zb where
      *: a = (xa, ya, za) b = (xb, yb, zb)
      by (cases a, cases b, auto)
    assume **: a ∈ unit-sphere b ∈ unit-sphere stereographic-r3-cvec a ≈v stereographic-r3-cvec b
    show a = b
    proof (cases a = (0, 0, 1) ∨ b = (0, 0, 1))
      case True
      thus ?thesis
        using * **
        by (simp split: if-split-asm) force+
    next
      case False
      then obtain k where ++: k ≠ 0 cor xb + i * cor yb = k * (cor xa + i * cor ya) 1 - cor zb = k * (1 - cor za)
        using * **
        by (auto split: if-split-asm)

      {
        assume xb + xa*zb = xa + xb*za
               yb + ya*zb = ya + yb*za
               xa*xa + ya*ya + za*za = 1 xb*xb + yb*yb + zb*zb = 1
               za ≠ 1 zb ≠ 1
        hence xa = xb ∧ ya = yb ∧ za = zb
          by algebra
      } note *** = this

      have za ≠ 1 zb ≠ 1
        using False * **
        by auto
      have k = (1 - cor zb) / (1 - cor za)
        using ‹1 - cor zb = k * (1 - cor za)› ‹za ≠ 1›
        by simp
      hence (1 - cor za) * (cor xb + i * cor yb) = (1 - cor zb) * (cor xa + i * cor ya)
        using ‹za ≠ 1› ++(2)
        by simp
      hence xb + xa*zb = xa + xb*za
            yb + ya*zb = ya + yb*za
            xa*xa + ya*ya + za*za = 1 xb*xb + yb*yb + zb*zb = 1
        using * ** ‹za ≠ 1›
        apply (simp-all add: field-simps)
        unfolding complex-of-real-def imaginary-unit.ctr
        by (simp-all add: legacy-Complex-simps)
      thus ?thesis
        using * ** *** ‹za ≠ 1› ‹zb ≠ 1›
        by simp
      qed
  qed
next
  fix y
  show ∃ x. y = stereographic x
    by (rule-tac x=inv-stereographic y in exI, simp)
qed


lemma inv-stereographic-stereographic [simp]:
  shows inv-stereographic (stereographic x) = x
  using stereographic-inv-stereographic[of stereographic x]
  using bij-stereographic
  unfolding bij-def inj-on-def
  by simp

lemma inv-stereographic-is-inv:
  shows inv-stereographic = inv stereographic
  by (rule inv-equality[symmetric], simp-all)
```

10.3 Circles on the sphere

Circlines in the plane correspond to circles on the Riemann sphere, and we formally establish this connection. Every circle in three–dimensional space can be obtained as the intersection of a sphere and a plane. We establish a one-to-one correspondence between circles on the Riemann sphere and planes in space. Note that the plane need not intersect the sphere, but we will still say that it defines a single imaginary circle. However, for one special circline (the one with the identity representative matrix), there does not exist a plane in $\mathbb{R}^3$ that would correspond to it — in order to have this, instead of considering planes in $\mathbb{R}^3$, we must consider three dimensional projective space and consider the infinite (hyper)plane.

Planes in R^3 are given by equations $ax + by + cz = d$. Two four-tuples of coefficients (a, b, c, d) give the same plane iff they are proportional.

type-synonym *R4* = *real* × *real* × *real* × *real*

fun *mult-sv* :: *real* ⇒ *R4* ⇒ *R4* (**infixl** ‹$*_{sv4}$› *100*) **where**
  k $*_{sv4}$ (*a*, *b*, *c*, *d*) = (*k*∗*a*, *k*∗*b*, *k*∗*c*, *k*∗*d*)

abbreviation *plane-vectors* **where**
  plane-vectors ≡ {(*a*::*real*, *b*::*real*, *c*::*real*, *d*::*real*). *a* ≠ *0* ∨ *b* ≠ *0* ∨ *c* ≠ *0* ∨ *d* ≠ *0*}

typedef *plane-vec* = *plane-vectors*
  by (*rule-tac x*=(*1*, *1*, *1*, *1*) **in** *exI*) *simp*

setup-lifting *type-definition-plane-vec*

definition *plane-vec-eq-r4* :: *R4* ⇒ *R4* ⇒ *bool* **where**
  [*simp*]: *plane-vec-eq-r4 v1 v2* ⟷ (∃ *k*. *k* ≠ *0* ∧ *v2* = *k* $*_{sv4}$ *v1*)

lift-definition *plane-vec-eq* :: *plane-vec* ⇒ *plane-vec* ⇒ *bool* **is** *plane-vec-eq-r4*
  done

lemma *mult-sv-one* [*simp*]:
  shows *1* $*_{sv4}$ *x* = *x*
  by (*cases x*) *simp*

lemma *mult-sv-distb* [*simp*]:
  shows *x* $*_{sv4}$ (*y* $*_{sv4}$ *v*) = (*x*∗*y*) $*_{sv4}$ *v*
  by (*cases v*) *simp*

quotient-type *plane* = *plane-vec* / *plane-vec-eq*
proof (*rule equivpI*)
  show *reflp plane-vec-eq*
    unfolding *reflp-def*
    by (*auto simp add*: *plane-vec-eq-def*) (*rule-tac x=1* **in** *exI*, *simp*)
next
  show *symp plane-vec-eq*
    unfolding *symp-def*
    by (*auto simp add*: *plane-vec-eq-def*) (*rule-tac x=1/k* **in** *exI*, *simp*)
next
  show *transp plane-vec-eq*
    unfolding *transp-def*
    by (*auto simp add*: *plane-vec-eq-def*) (*rule-tac x=ka∗k* **in** *exI*, *simp*)
qed

Plane coefficients give a linear equation and the point on the Riemann sphere lies on the circle determined by the plane iff its representation satisfies that linear equation.

definition *on-sphere-circle-r4-r3* :: *R4* ⇒ *R3* ⇒ *bool* **where**
  [*simp*]: *on-sphere-circle-r4-r3* α *A* ⟷
    (*let* (*X*, *Y*, *Z*) = *A*;
        (*a*, *b*, *c*, *d*) = α
      in a∗*X* + *b*∗*Y* + *c*∗*Z* + *d* = *0*)

lift-definition *on-sphere-circle-vec* :: *plane-vec* ⇒ *R3* ⇒ *bool* **is** *on-sphere-circle-r4-r3*
  done

lift-definition *on-sphere-circle* :: *plane* ⇒ *riemann-sphere* ⇒ *bool* **is** *on-sphere-circle-vec*
proof (*transfer*)
fix *pv1 pv2* :: *R4* **and** *w* :: *R3*
obtain *a1 b1 c1 d1 a2 b2 c2 d2 x y z* **where**
∗: *pv1* = (*a1*, *b1*, *c1*, *d1*) *pv2* = (*a2*, *b2*, *c2*, *d2*) *w* = (*x*, *y*, *z*)
by (*cases pv1*, *cases pv2*, *cases w*, *auto*)
assume *pv1* ∈ *plane-vectors pv2* ∈ *plane-vectors w* ∈ *unit-sphere plane-vec-eq-r4 pv1 pv2*
then obtain *k* **where** ∗∗: *a2* = *k*∗*a1 b2* = *k*∗*b1 c2* = *k*∗*c1 d2* = *k*∗*d1 k* ≠ *0*
using ∗
by *auto*
have *k* ∗ *a1* ∗ *x* + *k* ∗ *b1* ∗ *y* + *k* ∗ *c1* ∗ *z* + *k* ∗ *d1* = *k*∗(*a1*∗*x* + *b1*∗*y* + *c1*∗*z* + *d1*)
by (*simp add*: *field-simps*)
thus *on-sphere-circle-r4-r3 pv1 w* = *on-sphere-circle-r4-r3 pv2 w*
using ∗ ∗∗
by *simp*
qed

definition *sphere-circle-set* **where**
sphere-circle-set α = {*A*. *on-sphere-circle* α *A*}

10.4 Connections of circlines in the plane and circles on the Riemann sphere

We introduce stereographic and inverse stereographic projection between circles on the Riemann sphere and circlines in the extended complex plane.

definition *inv-stereographic-circline-cmat-r4* :: *complex-mat* ⇒ *R4* **where**
[*simp*]: *inv-stereographic-circline-cmat-r4 H* =
(*let* (*A*, *B*, *C*, *D*) = *H*
in (*Re* (*B*+*C*), *Re*(i∗(*C*−*B*)), *Re*(*A*−*D*), *Re*(*D*+*A*)))

lift-definition *inv-stereographic-circline-clmat-pv* :: *circline-mat* ⇒ *plane-vec* **is** *inv-stereographic-circline-cmat-r4*
by (*auto simp add*: *hermitean-def mat-adj-def mat-cnj-def real-imag-0 eq-cnj-iff-real*)

lift-definition *inv-stereographic-circline* :: *circline* ⇒ *plane* **is** *inv-stereographic-circline-clmat-pv*
apply *transfer*
apply *simp*
apply (*erule exE*)
apply (*rule-tac x*=*k* **in** *exI*)
apply (*case-tac circline-mat1*, *case-tac circline-mat2*)
apply (*simp add*: *field-simps*)
done

definition *stereographic-circline-r4-cmat* :: *R4* ⇒ *complex-mat* **where**
[*simp*]: *stereographic-circline-r4-cmat* α =
(*let* (*a*, *b*, *c*, *d*) = α
in (*cor* ((*c*+*d*)/*2*) , ((*cor a* + i ∗ *cor b*)/*2*), ((*cor a* − i ∗ *cor b*)/*2*), *cor* ((*d*−*c*)/*2*)))

lift-definition *stereographic-circline-pv-clmat* :: *plane-vec* ⇒ *circline-mat* **is** *stereographic-circline-r4-cmat*
by (*auto simp add*: *hermitean-def mat-adj-def mat-cnj-def*)

lift-definition *stereographic-circline* :: *plane* ⇒ *circline* **is** *stereographic-circline-pv-clmat*
apply *transfer*
apply *transfer*
apply (*case-tac plane-vec1*, *case-tac plane-vec2*, *simp*, *erule exE*, *rule-tac x*=*k* **in** *exI*, *simp add*: *field-simps*)
done

Stereographic and inverse stereographic projection of circlines are mutually inverse.

lemma *stereographic-circline-inv-stereographic-circline*:
shows *stereographic-circline* ∘ *inv-stereographic-circline* = *id*
proof (*rule ext*, *simp*)
fix *H*
show *stereographic-circline* (*inv-stereographic-circline H*) = *H*
proof (*transfer*, *transfer*)
fix *H*
assume *hh*: *hermitean H* ∧ *H* ≠ *mat-zero*
obtain *A B C D* **where** *HH*: *H* = (*A*, *B*, *C*, *D*)

```
    by (cases H) auto
  have is-real A ∧ is-real D ∧ C = cnj B
    using HH hh hermitean-elems[of A B C D]
    by auto
  thus circline-eq-cmat (stereographic-circline-r4-cmat (inv-stereographic-circline-cmat-r4 H)) H
    using HH
    apply simp
    apply (rule-tac x=1 in exI, cases B)
     by (smt (verit) add-uminus-conv-diff complex-cnj-add complex-cnj-complex-of-real complex-cnj-i complex-cnj-mult
complex-cnj-one complex-eq distrib-left-numeral mult.commute mult.left-commute mult.left-neutral mult-cancel-right2 mult-minus-left
of-real-1 one-add-one)
 qed
qed
```

Stereographic and inverse stereographic projection of circlines are mutually inverse.

```
lemma inv-stereographic-circline-stereographic-circline:
  inv-stereographic-circline ∘ stereographic-circline = id
proof (rule ext, simp)
  fix α
  show inv-stereographic-circline (stereographic-circline α) = α
  proof (transfer, transfer)
    fix α
    assume aa: α ∈ plane-vectors
    obtain a b c d where AA: α = (a, b, c, d)
      by (cases α) auto
    thus plane-vec-eq-r4 (inv-stereographic-circline-cmat-r4 (stereographic-circline-r4-cmat α)) α
      using AA
      by simp (rule-tac x=1 in exI, auto simp add: field-simps complex-of-real-def)
  qed
qed

lemma stereographic-sphere-circle-set'':
  shows on-sphere-circle (inv-stereographic-circline H) z ⟷
        on-circline H (stereographic z)
proof (transfer, transfer)
  fix M :: R3 and H :: complex-mat
  assume hh: hermitean H ∧ H ≠ mat-zero M ∈ unit-sphere
  obtain A B C D where HH: H = (A, B, C, D)
    by (cases H) auto
  have ∗: is-real A ∧ is-real D ∧ C = cnj B
    using hh HH hermitean-elems[of A B C D]
    by auto
  obtain x y z where MM: M = (x, y, z)
    by (cases M) auto
  show on-sphere-circle-r4-r3 (inv-stereographic-circline-cmat-r4 H) M ⟷
       on-circline-cmat-cvec H (stereographic-r3-cvec M) (is ?lhs = ?rhs)
  proof
    assume ?lhs
    show ?rhs
    proof (cases z=1)
      case True
      hence x = 0 ∧ y = 0
        using MM hh
        by auto
      thus ?thesis
        using ∗ ‹?lhs› HH MM ‹z=1›
        by (cases A, simp add: vec-cnj-def Complex-eq Let-def)
    next
      case False
      hence Re A∗(1+z) + 2∗Re B∗x + 2∗Im B∗y + Re D∗(1−z) = 0
        using ∗ ‹?lhs› HH MM
        by (simp add: Let-def field-simps)
      hence (Re A∗(1+z) + 2∗Re B∗x + 2∗Im B∗y + Re D∗(1−z))∗(1−z) = 0
        by simp
      hence Re A∗(1+z)∗(1−z) + 2∗Re B∗x∗(1−z) + 2∗Im B∗y∗(1−z) + Re D∗(1−z)∗(1−z) = 0
        by (simp add: field-simps)
```

```
      moreover
      have x*x+y*y = (1+z)*(1−z)
        using MM hh
        by (simp add: field-simps)
      ultimately
      have Re A*(x*x+y*y) + 2*Re B*x*(1−z) + 2*Im B*y*(1−z) + Re D*(1−z)*(1−z) = 0
        by simp
      hence (x * Re A + (1 − z) * Re B) * x − (− (y * Re A) + − ((1 − z) * Im B)) * y + (x * Re B + y * Im B +
(1 − z) * Re D) * (1 − z) = 0
        by (simp add: field-simps)
      thus ?thesis
        using ‹z ≠ 1› HH MM * ‹Re A*(1+z) + 2*Re B*x + 2*Im B*y + Re D*(1−z) = 0›
        apply (simp add: Let-def vec-cnj-def)
        apply (subst complex-eq-iff)
        apply (simp add: field-simps)
        done
    qed
  next
    assume ?rhs
    show ?lhs
    proof (cases z=1)
      case True
      hence x = 0 y = 0
        using MM hh
        by auto
      thus ?thesis
        using HH MM ‹?rhs› ‹z = 1›
        by (simp add: Let-def vec-cnj-def)
    next
      case False
      hence (x * Re A + (1 − z) * Re B) * x − (− (y * Re A) + − ((1 − z) * Im B)) * y + (x * Re B + y * Im B +
(1 − z) * Re D) * (1 − z) = 0
        using HH MM * ‹?rhs›
        by (simp add: Let-def vec-cnj-def complex-eq-iff)
      hence Re A*(x*x+y*y) + 2*Re B*x*(1−z) + 2*Im B*y*(1−z) + Re D*(1−z)*(1−z) = 0
        by (simp add: field-simps)
      moreover
      have x*x + y*y = (1+z)*(1−z)
        using MM hh
        by (simp add: field-simps)
      ultimately
      have Re A*(1+z)*(1−z) + 2*Re B*x*(1−z) + 2*Im B*y*(1−z) + Re D*(1−z)*(1−z) = 0
        by simp
      hence (Re A*(1+z) + 2*Re B*x + 2*Im B*y + Re D*(1−z))*(1−z) = 0
        by (simp add: field-simps)
      hence Re A*(1+z) + 2*Re B*x + 2*Im B*y + Re D*(1−z) = 0
        using ‹z ≠ 1›
        by simp
      thus ?thesis
        using MM HH *
        by (simp add: field-simps)
    qed
  qed
qed

lemma stereographic-sphere-circle-set' [simp]:
  shows stereographic ‘ sphere-circle-set (inv-stereographic-circline H) =
        circline-set H
unfolding sphere-circle-set-def circline-set-def
apply safe
proof−
  fix x
  assume on-sphere-circle (inv-stereographic-circline H) x
  thus on-circline H (stereographic x)
    using stereographic-sphere-circle-set''
    by simp
```

next
fix *x*
assume *on-circline H x*
show *x* ∈ *stereographic* ‘ {*z*. *on-sphere-circle* (*inv-stereographic-circline H*) *z*}
proof
show *x* = *stereographic* (*inv-stereographic x*)
by *simp*
next
show *inv-stereographic x* ∈ {*z*. *on-sphere-circle* (*inv-stereographic-circline H*) *z*}
using *stereographic-sphere-circle-set''*[*of H inv-stereographic x*] ‹*on-circline H x*›
by *simp*
qed
qed

The projection of the set of points on a circle on the Riemann sphere is exactly the set of points on the circline obtained by the just introduced circle stereographic projection.

lemma *stereographic-sphere-circle-set*:
shows *stereographic* ‘ *sphere-circle-set H* = *circline-set* (*stereographic-circline H*)
using *stereographic-sphere-circle-set'*[*of stereographic-circline H*]
using *inv-stereographic-circline-stereographic-circline*
unfolding *comp-def*
by (*metis id-apply*)

Stereographic projection of circlines is bijective.

lemma *bij-stereographic-circline*:
shows *bij stereographic-circline*
using *stereographic-circline-inv-stereographic-circline inv-stereographic-circline-stereographic-circline*
using *o-bij* **by** *blast*

Inverse stereographic projection is bijective.

lemma *bij-inv-stereographic-circline*:
shows *bij inv-stereographic-circline*
using *stereographic-circline-inv-stereographic-circline inv-stereographic-circline-stereographic-circline*
using *o-bij* **by** *blast*

end

10.5 Chordal Metric

Riemann sphere can be made a metric space. We are going to introduce distance on Riemann sphere and to prove that it is a metric space. The distance between two points on the sphere is defined as the length of the chord that connects them. This metric can be used in formalization of elliptic geometry.

theory *Chordal-Metric*
imports *Homogeneous-Coordinates Riemann-Sphere Oriented-Circlines HOL−Analysis.Inner-Product HOL−Analysis.Euclidean-Spac*
begin

10.5.1 Inner product and norm

definition *inprod-cvec* :: *complex-vec* ⇒ *complex-vec* ⇒ *complex* **where**
[*simp*]: *inprod-cvec z w* =
(*let* (*z1*, *z2*) = *z*;
(*w1*, *w2*) = *w*
in vec-cnj (*z1*, *z2*) ∗$_{vv}$ (*w1*, *w2*))
syntax
-inprod-cvec :: *complex-vec* ⇒ *complex-vec* ⇒ *complex* (‹⟨-,-⟩›)
syntax-consts
-inprod-cvec == *inprod-cvec*
translations
⟨*z*,*w*⟩ == *CONST inprod-cvec z w*

lemma *real-inprod-cvec* [*simp*]:
shows *is-real* ⟨*z*,*z*⟩
by (*cases z*, *simp add*: *vec-cnj-def*)

lemma *inprod-cvec-ge-zero* [*simp*]:

shows *Re* $\langle z,z\rangle \geq 0$
by (*cases z*, *simp add*: *vec-cnj-def*)

lemma *inprod-cvec-bilinear1* [*simp*]:
assumes $z' = k *_{sv} z$
shows $\langle z',w\rangle = cnj\ k * \langle z,w\rangle$
using *assms*
by (*cases z*, *cases z'*, *cases w*) (*simp add*: *vec-cnj-def field-simps*)

lemma *inprod-cvec-bilinear2* [*simp*]:
assumes $z' = k *_{sv} z$
shows $\langle w, z'\rangle = k * \langle w, z\rangle$
using *assms*
by (*cases z*, *cases z'*, *cases w*) (*simp add*: *vec-cnj-def field-simps*)

lemma *inprod-cvec-g-zero* [*simp*]:
assumes $z \neq vec\text{-}zero$
shows $Re\ \langle z, z\rangle > 0$
proof−
have $\forall\ a\ b.\ a \neq 0 \vee b \neq 0 \longrightarrow 0 < (Re\ a * Re\ a + Im\ a * Im\ a) + (Re\ b * Re\ b + Im\ b * Im\ b)$
by (*smt* (*verit*) *complex-eq-0 not-sum-squares-lt-zero power2-eq-square*)
thus *?thesis*
using *assms*
by (*cases z*, *simp add*: *vec-cnj-def*)
qed

definition *norm-cvec* :: *complex-vec* ⇒ *real* **where**
[*simp*]: *norm-cvec z* = *sqrt* (*Re* $\langle z,z\rangle$)
syntax
-norm-cvec :: *complex-vec* ⇒ *complex* (‹⟨-⟩›)
syntax-consts
-norm-cvec == *norm-cvec*
translations
$\langle z\rangle$ == *CONST norm-cvec z*

lemma *norm-cvec-square*:
shows $\langle z\rangle^2 = Re\ (\langle z,z\rangle)$
by (*simp del*: *inprod-cvec-def*)

lemma *norm-cvec-gt-0*:
assumes $z \neq vec\text{-}zero$
shows $\langle z\rangle > 0$
using *assms*
by (*simp del*: *inprod-cvec-def*)

lemma *norm-cvec-scale*:
assumes $z' = k *_{sv} z$
shows $\langle z'\rangle^2 = Re\ (cnj\ k * k) * \langle z\rangle^2$
unfolding *norm-cvec-square*
using *inprod-cvec-bilinear1*[*OF assms*, *of z'*]
using *inprod-cvec-bilinear2*[*OF assms*, *of z*]
by (*simp del*: *inprod-cvec-def add*: *field-simps*)

lift-definition *inprod-hcoords* :: *complex-homo-coords* ⇒ *complex-homo-coords* ⇒ *complex* **is** *inprod-cvec*
done

lift-definition *norm-hcoords* :: *complex-homo-coords* ⇒ *real* **is** *norm-cvec*
done

10.5.2 Distance in $\mathbb{C}P^1$ - defined by Fubini-Study metric.

Formula for the chordal distance between the two points can be given directly based on the homogenous coordinates of their stereographic projections in the plane. This is called the Fubini-Study metric.

definition *dist-fs-cvec* :: *complex-vec* ⇒ *complex-vec* ⇒ *real* **where** [*simp*]:
dist-fs-cvec z1 z2 =
(*let* (*z1x*, *z1y*) = *z1*;

```
      (z2x, z2y) = z2;
      num = (z1x*z2y − z2x*z1y) * (cnj z1x*cnj z2y − cnj z2x*cnj z1y);
      den = (z1x*cnj z1x + z1y*cnj z1y) * (z2x*cnj z2x + z2y*cnj z2y)
    in 2*sqrt(Re num / Re den))

lemma dist-fs-cvec-iff:
  assumes z ≠ vec-zero and w ≠ vec-zero
  shows dist-fs-cvec z w = 2*sqrt(1 − (cmod ⟨z,w⟩)² / (⟨z⟩² * ⟨w⟩²))
proof−
  obtain z1 z2 w1 w2 where *: z = (z1, z2) w = (w1, w2)
    by (cases z, cases w) auto
  have 1: 2*sqrt(1 − (cmod ⟨z,w⟩)² / (⟨z⟩² * ⟨w⟩²)) = 2*sqrt((⟨z⟩² * ⟨w⟩² − (cmod ⟨z,w⟩)²) / (⟨z⟩² * ⟨w⟩²))
    using norm-cvec-gt-0[of z] norm-cvec-gt-0[of w] assms
    by (simp add: field-simps)

  have 2: ⟨z⟩² * ⟨w⟩² = Re ((z1*cnj z1 + z2*cnj z2) * (w1*cnj w1 + w2*cnj w2))
    using assms *
    by (simp add: vec-cnj-def)

  have 3: ⟨z⟩² * ⟨w⟩² − (cmod ⟨z,w⟩)² = Re ((z1*w2 − w1*z2) * (cnj z1*cnj w2 − cnj w1*cnj z2))
    apply (subst cmod-square, (subst norm-cvec-square)+)
    using *
    by (simp add: vec-cnj-def field-simps)

  thus ?thesis
    using 1 2 3
    using *
    unfolding dist-fs-cvec-def Let-def
    by simp
qed

lift-definition dist-fs-hcoords :: complex-homo-coords ⇒ complex-homo-coords ⇒ real is dist-fs-cvec
  done

lift-definition dist-fs :: complex-homo ⇒ complex-homo ⇒ real is dist-fs-hcoords
proof transfer
  fix z1 z2 z1' z2' :: complex-vec
  obtain z1x z1y z2x z2y z1'x z1'y z2'x z2'y where
    zz: z1 = (z1x, z1y) z2 = (z2x, z2y) z1' = (z1'x, z1'y) z2' = (z2'x, z2'y)
    by (cases z1, cases z2, cases z1', cases z2') blast

  assume 1: z1 ≠ vec-zero z2 ≠ vec-zero z1' ≠ vec-zero z2' ≠ vec-zero z1 ≈v z1' z2 ≈v z2'
  then obtain k1 k2 where
    *: k1 ≠ 0 z1' = k1 *sv z1 and
    **: k2 ≠ 0 z2' = k2 *sv z2
    by auto
  have (cmod ⟨z1,z2⟩)² / (⟨z1⟩² * ⟨z2⟩²) = (cmod ⟨z1',z2'⟩)² / (⟨z1'⟩² * ⟨z2'⟩²)
    using ‹k1 ≠ 0› ‹k2 ≠ 0›
    using cmod-square[symmetric, of k1] cmod-square[symmetric, of k2]
    apply (subst norm-cvec-scale[OF *(2)])
    apply (subst norm-cvec-scale[OF **(2)])
    apply (subst inprod-cvec-bilinear1[OF *(2)])
    apply (subst inprod-cvec-bilinear2[OF **(2)])
    by (simp add: power2-eq-square norm-mult)
  thus dist-fs-cvec z1 z2 = dist-fs-cvec z1' z2'
    using 1 dist-fs-cvec-iff
    by simp
qed

lemma dist-fs-finite:
  shows dist-fs (of-complex z1) (of-complex z2) = 2 * cmod(z1 − z2) / (sqrt (1+(cmod z1)²) * sqrt (1+(cmod z2)²))
  apply transfer
  apply transfer
  apply (subst cmod-square)+
  apply (simp add: real-sqrt-divide cmod-def power2-eq-square)
  apply (subst real-sqrt-mult[symmetric])
```

apply (*simp add*: *field-simps*)
done

lemma *dist-fs-infinite1*:
shows *dist-fs* (*of-complex z1*) ∞_h = *2* / *sqrt* (*1*+(*cmod z1*)^2)
by (*transfer*, *transfer*) (*subst cmod-square*, *simp add*: *real-sqrt-divide*)

lemma *dist-fs-infinite2*:
shows *dist-fs* ∞_h (*of-complex z1*) = *2* / *sqrt* (*1*+(*cmod z1*)^2)
by (*transfer*, *transfer*) (*subst cmod-square*, *simp add*: *real-sqrt-divide*)

lemma *dist-fs-cvec-zero*:
assumes *z* ≠ *vec-zero* **and** *w* ≠ *vec-zero*
shows *dist-fs-cvec z w* = *0* ⟷ (*cmod* ⟨*z*,*w*⟩)^2 = (⟨*z*⟩^{2} * ⟨*w*⟩^{2})
using *assms norm-cvec-gt-0*[*of z*] *norm-cvec-gt-0*[*of w*]
by (*subst dist-fs-cvec-iff*) *auto*

lemma *dist-fs-zero1* [*simp*]:
shows *dist-fs z z* = *0*
by (*transfer*, *transfer*)
(*subst dist-fs-cvec-zero*, *simp*, (*subst norm-cvec-square*)+, *subst cmod-square*, *simp del*: *inprod-cvec-def*)

lemma *dist-fs-zero2* [*simp*]:
assumes *dist-fs z1 z2* = *0*
shows *z1* = *z2*
using *assms*
proof (*transfer*, *transfer*)
fix *z w* :: *complex-vec*
obtain *z1 z2 w1 w2* **where** ∗: *z* = (*z1*, *z2*) *w* = (*w1*, *w2*)
by (*cases z*, *cases w*, *auto*)
let *?x* = (*z1*∗*w2* − *w1*∗*z2*) ∗ (*cnj z1*∗*cnj w2* − *cnj w1*∗*cnj z2*)
assume *z* ≠ *vec-zero w* ≠ *vec-zero dist-fs-cvec z w* = *0*
hence (*cmod* ⟨*z*,*w*⟩)^2 = ⟨*z*⟩^{2} * ⟨*w*⟩^{2}
by (*subst* (*asm*) *dist-fs-cvec-zero*, *simp-all*)
hence *Re ?x* = *0*
using ∗
by (*subst* (*asm*) *cmod-square*) ((*subst* (*asm*) *norm-cvec-square*)+, *simp add*: *vec-cnj-def field-simps*)
hence *?x* = *0*
using *complex-mult-cnj-cmod*[*of z1*∗*w2* − *w1*∗*z2*] *zero-complex.simps*
by (*subst complex-eq-if-Re-eq*[*of ?x 0*]) (*simp add*: *power2-eq-square*, *simp*, *linarith*)
moreover
have *z1* ∗ *w2* − *w1* ∗ *z2* = *0* ⟷ *cnj z1* ∗ *cnj w2* − *cnj w1* ∗ *cnj z2* = *0*
by (*metis complex-cnj-diff complex-cnj-mult complex-cnj-zero-iff*)
ultimately
show *z* ≈$_v$ *w*
using ∗ ‹*z* ≠ *vec-zero*› ‹*w* ≠ *vec-zero*›
using *complex-cvec-eq-mix*[*of z1 z2 w1 w2*]
by *auto*
qed

lemma *dist-fs-sym*:
shows *dist-fs z1 z2* = *dist-fs z2 z1*
by (*transfer*, *transfer*) (*simp add*: *split-def field-simps*)

10.5.3 Triangle inequality for Fubini-Study metric

lemma *dist-fs-triangle-finite*:
shows *cmod*(*a* − *b*) / (*sqrt* (*1*+(*cmod a*)^2) ∗ *sqrt* (*1*+(*cmod b*)^2)) ≤ *cmod* (*a* − *c*) / (*sqrt* (*1*+(*cmod a*)^2) ∗ *sqrt* (*1*+(*cmod c*)^2)) + *cmod* (*c* − *b*) / (*sqrt* (*1*+(*cmod b*)^2) ∗ *sqrt* (*1*+(*cmod c*)^2))
proof−
let *?cc* = *1*+(*cmod c*)^2 **and** *?bb* = *1*+(*cmod b*)^2 **and** *?aa* = *1*+(*cmod a*)^2
have *sqrt ?cc* > *0 sqrt ?aa* > *0 sqrt ?bb* > *0*
by (*smt* (*verit*) *real-sqrt-gt-zero zero-compare-simps*(*12*))+
have (*a* − *b*)∗(*1*+*cnj c*∗*c*) = (*a*−*c*)∗(*1*+*cnj c*∗*b*) + (*c*−*b*)∗(*1* + *cnj c*∗*a*)
by (*simp add*: *field-simps*)
moreover

have $1 + cnj\ c * c = 1 + (cmod\ c)^2$
using *complex-norm-square*
by *auto*
hence $cmod\ ((a - b)*(1+cnj\ c*c)) = cmod(a - b) * (1+(cmod\ c)^2)$
by (*smt* (*verit*) *norm-mult norm-of-real zero-compare-simps*(12))
ultimately
have $cmod(a - b) * (1+(cmod\ c)^2) \leq cmod\ (a-c) * cmod\ (1+cnj\ c*b) + cmod\ (c-b) * cmod(1 + cnj\ c*a)$
using *complex-mod-triangle-ineq2*[*of* $(a-c)*(1+cnj\ c*b)\ (c-b)*(1 + cnj\ c*a)$]
by (*simp add*: *norm-mult*)
moreover
have $*$: $\bigwedge a\ b\ c\ d\ b'\ d'.\ [\![b \leq b';\ d \leq d';\ a \geq (0::real);\ c \geq 0]\!] \Longrightarrow a*b + c*d \leq a*b' + c*d'$
by (*simp add*: *add-mono-thms-linordered-semiring*(1) *mult-left-mono*)
have $cmod\ (a-c) * cmod\ (1+cnj\ c*b) + cmod\ (c-b) * cmod(1 + cnj\ c*a) \leq cmod\ (a - c) * (sqrt\ (1+(cmod\ c)^2) * sqrt\ (1+(cmod\ b)^2)) + cmod\ (c - b) * (sqrt\ (1+(cmod\ c)^2) * sqrt\ (1+(cmod\ a)^2))$
using $*$[*OF cmod-1-plus-mult-le*[*of cnj c b*] *cmod-1-plus-mult-le*[*of cnj c a*], *of cmod* $(a-c)$ *cmod* $(c-b)$]
by (*simp add*: *field-simps real-sqrt-mult*[*symmetric*])
ultimately
have $cmod(a - b) * ?cc \leq cmod\ (a - c) * sqrt\ ?cc * sqrt\ ?bb + cmod\ (c - b) * sqrt\ ?cc * sqrt\ ?aa$
by *simp*
moreover
hence $0 \leq ?cc * sqrt\ ?aa * sqrt\ ?bb$
using *mult-right-mono*[*of 0 sqrt ?aa sqrt ?bb*]
using *mult-right-mono*[*of 0 ?cc sqrt ?aa * sqrt ?bb*]
by *simp*
moreover
have $sqrt\ ?cc\ /\ ?cc = 1\ /\ sqrt\ ?cc$
using ‹$sqrt\ ?cc > 0$›
by (*simp add*: *field-simps*)
hence $sqrt\ ?cc\ /\ (?cc * sqrt\ ?aa) = 1\ /\ (sqrt\ ?aa * sqrt\ ?cc)$
using *times-divide-eq-right*[*of 1*/*sqrt ?aa sqrt ?cc ?cc*]
using ‹$sqrt\ ?aa > 0$›
by *simp*
hence $cmod\ (a - c) * sqrt\ ?cc\ /\ (?cc * sqrt\ ?aa) = cmod\ (a - c)\ /\ (sqrt\ ?aa * sqrt\ ?cc)$
using *times-divide-eq-right*[*of cmod* $(a - c)$ *sqrt ?cc* $(?cc * sqrt\ ?aa)$]
by *simp*
moreover
have $sqrt\ ?cc\ /\ ?cc = 1\ /\ sqrt\ ?cc$
using ‹$sqrt\ ?cc > 0$›
by (*simp add*: *field-simps*)
hence $sqrt\ ?cc\ /\ (?cc * sqrt\ ?bb) = 1\ /\ (sqrt\ ?bb * sqrt\ ?cc)$
using *times-divide-eq-right*[*of 1*/*sqrt ?bb sqrt ?cc ?cc*]
using ‹$sqrt\ ?bb > 0$›
by *simp*
hence $cmod\ (c - b) * sqrt\ ?cc\ /\ (?cc * sqrt\ ?bb) = cmod\ (c - b)\ /\ (sqrt\ ?bb * sqrt\ ?cc)$
using *times-divide-eq-right*[*of cmod* $(c - b)$ *sqrt ?cc ?cc * sqrt ?bb*]
by *simp*
ultimately
show *?thesis*
using *divide-right-mono*[*of cmod* $(a - b) * ?cc\ cmod\ (a - c) * sqrt\ ?cc * sqrt\ ?bb + cmod\ (c - b) * sqrt\ ?cc * sqrt\ ?aa\ ?cc * sqrt\ ?aa * sqrt\ ?bb$] ‹$sqrt\ ?aa > 0$› ‹$sqrt\ ?bb > 0$› ‹$sqrt\ ?cc > 0$›
by (*simp add*: *add-divide-distrib*)
qed

lemma *dist-fs-triangle-infinite1*:
shows $1\ /\ sqrt(1 + (cmod\ b)^2) \leq 1\ /\ sqrt(1 + (cmod\ c)^2) + cmod\ (b - c)\ /\ (sqrt(1 + (cmod\ b)^2) * sqrt(1 + (cmod\ c)^2))$
proof−
let $?bb = sqrt\ (1 + (cmod\ b)^2)$ **and** $?cc = sqrt\ (1 + (cmod\ c)^2)$
have $?bb > 0\ ?cc > 0$
by (*metis add-strict-increasing real-sqrt-gt-0-iff zero-le-power2 zero-less-one*)+
hence $*$: $?bb * ?cc \geq 0$
by *simp*
have $**$: $(?cc - ?bb)\ /\ (?bb * ?cc) = 1\ /\ ?bb - 1\ /\ ?cc$
using ‹$sqrt\ (1 + (cmod\ b)^2) > 0$› ‹$sqrt\ (1 + (cmod\ c)^2) > 0$›
by (*simp add*: *field-simps*)
show $1\ /\ ?bb \leq 1\ /\ ?cc + cmod\ (b - c)\ /\ (?bb * ?cc)$

```
    using divide-right-mono[OF cmod-diff-ge[of c b] *]
    by (subst (asm) **) (simp add: field-simps norm-minus-commute)
qed

lemma dist-fs-triangle-infinite2:
  shows 1 / sqrt(1 + (cmod a)^2) ≤ cmod (a − c) / (sqrt (1+(cmod a)^2) * sqrt (1+(cmod c)^2)) + 1 / sqrt(1 + (cmod c)^2)
  using dist-fs-triangle-infinite1[of a c]
  by simp

lemma dist-fs-triangle-infinite3:
  shows cmod(a − b) / (sqrt (1+(cmod a)^2) * sqrt (1+(cmod b)^2)) ≤ 1 / sqrt(1 + (cmod a)^2) + 1 / sqrt(1 + (cmod b)^2)
proof−
  let ?aa = sqrt (1 + (cmod a)^2) and ?bb = sqrt (1 + (cmod b)^2)
  have ?aa > 0 ?bb > 0
    by (metis add-strict-increasing real-sqrt-gt-0-iff zero-le-power2 zero-less-one)+
  hence *: ?aa * ?bb ≥ 0
    by simp
  have **: (?aa + ?bb) / (?aa * ?bb) = 1 / ?aa + 1 / ?bb
    using ‹?aa > 0› ‹?bb > 0›
    by (simp add: field-simps)
  show cmod (a − b) / (?aa * ?bb) ≤ 1 / ?aa + 1 / ?bb
    using divide-right-mono[OF cmod-diff-le[of a b] *]
    by (subst (asm) **) (simp add: field-simps norm-minus-commute)
qed

lemma dist-fs-triangle:
  shows dist-fs A B ≤ dist-fs A C + dist-fs C B
proof (cases A = ∞h)
  case True
  show ?thesis
  proof (cases B = ∞h)
    case True
    show ?thesis
    proof (cases C = ∞h)
      case True
      show ?thesis
        using ‹A = ∞h› ‹B = ∞h› ‹C = ∞h›
        by simp
    next
      case False
      then obtain c where C = of-complex c
        using inf-or-of-complex[of C]
        by auto
      show ?thesis
        using ‹A = ∞h› ‹B = ∞h› ‹C = of-complex c›
        by (simp add: dist-fs-infinite2 dist-fs-sym)
    qed
  next
    case False
    then obtain b where B = of-complex b
      using inf-or-of-complex[of B]
      by auto
    show ?thesis
    proof (cases C = ∞h)
      case True
      show ?thesis
        using ‹A = ∞h› ‹C = ∞h› ‹B = of-complex b›
        by simp
    next
      case False
      then obtain c where C = of-complex c
        using inf-or-of-complex[of C]
        by auto
      show ?thesis
```

```
      using ‹A = ∞ₕ› ‹B = of-complex b› ‹C = of-complex c›
      using mult-left-mono[OF dist-fs-triangle-infinite1[of b c], of 2]
      by (simp add: dist-fs-finite dist-fs-infinite1 dist-fs-infinite2 dist-fs-sym)
    qed
  qed
next
  case False
  then obtain a where A = of-complex a
    using inf-or-of-complex[of A]
    by auto
  show ?thesis
  proof (cases B = ∞ₕ)
    case True
    show ?thesis
    proof (cases C = ∞ₕ)
      case True
      show ?thesis
        using ‹B = ∞ₕ› ‹C = ∞ₕ› ‹A = of-complex a›
        by (simp add: dist-fs-infinite2)
    next
      case False
      then obtain c where C = of-complex c
        using inf-or-of-complex[of C]
        by auto
      show ?thesis
        using ‹B = ∞ₕ› ‹C = of-complex c› ‹A = of-complex a›
        using mult-left-mono[OF dist-fs-triangle-infinite2[of a c], of 2]
        by (simp add: dist-fs-finite dist-fs-infinite1 dist-fs-infinite2)
    qed
  next
    case False
    then obtain b where B = of-complex b
      using inf-or-of-complex[of B]
      by auto
    show ?thesis
    proof (cases C = ∞ₕ)
      case True
      thus ?thesis
        using ‹C = ∞ₕ› ‹B = of-complex b› ‹A = of-complex a›
        using mult-left-mono[OF dist-fs-triangle-infinite3[of a b], of 2]
        by (simp add: dist-fs-finite dist-fs-infinite1 dist-fs-infinite2)
    next
      case False
      then obtain c where C = of-complex c
        using inf-or-of-complex[of C]
        by auto
      show ?thesis
        using ‹A = of-complex a› ‹B = of-complex b› ‹C = of-complex c›
        using mult-left-mono[OF dist-fs-triangle-finite[of a b c], of 2]
        by (simp add: dist-fs-finite norm-minus-commute dist-fs-sym)
    qed
  qed
qed
```

10.5.4 $\mathbb{C}P^1$ with Fubini-Study metric is a metric space

Using the (already available) fact that $\mathbb{R}^3$ is a metric space (under the distance function $\lambda\ x\ y.\ norm(x-y)$), it was not difficult to show that the type *complex-homo* equipped with *dist-fs* is a metric space, i.e., an instantiation of the *metric-space* locale.

```
instantiation complex-homo :: metric-space
begin
definition dist-complex-homo = dist-fs
definition (uniformity-complex-homo :: (complex-homo × complex-homo) filter) = (INF e∈{0<..}. principal {(x, y). dist-class.dist x y < e})
definition open-complex-homo (U :: complex-homo set) = (∀ x ∈ U. eventually (λ(x', y). x' = x ⟶ y ∈ U) uniformity)
instance
```

proof
fix *x y* :: *complex-homo*
show (*dist-class.dist x y* = *0*) = (*x* = *y*)
unfolding *dist-complex-homo-def*
using *dist-fs-zero1*[*of x*] *dist-fs-zero2*[*of x y*]
by *auto*
next
fix *x y z* :: *complex-homo*
show *dist-class.dist x y* ≤ *dist-class.dist x z* + *dist-class.dist y z*
unfolding *dist-complex-homo-def*
using *dist-fs-triangle*[*of x y z*]
by (*simp add*: *dist-fs-sym*)
qed (*simp-all add*: *open-complex-homo-def uniformity-complex-homo-def*)
end

10.5.5 Chordal distance on the Riemann sphere

Distance of the two points is given by the length of the chord. We show that it corresponds to the Fubini-Study metric in the plane.

definition *dist-riemann-sphere-r3* :: *R3* ⇒ *R3* ⇒ *real* **where** [*simp*]:
dist-riemann-sphere-r3 M1 M2 =
(*let* (*x1*, *y1*, *z1*) = *M1*;
(*x2*, *y2*, *z2*) = *M2*
in norm (*x1* − *x2*, *y1* − *y2*, *z1* − *z2*))

lemma *dist-riemann-sphere-r3-inner*:
assumes *M1* ∈ *unit-sphere* **and** *M2* ∈ *unit-sphere*
shows (*dist-riemann-sphere-r3 M1 M2*)^2 = *2* − *2* ∗ *inner M1 M2*
using *assms*
apply (*cases M1*, *cases M2*)
apply (*auto simp add*: *norm-prod-def*)
apply (*simp add*: *power2-eq-square field-simps*)
done

lift-definition *dist-riemann-sphere′* :: *riemann-sphere* ⇒ *riemann-sphere* ⇒ *real* **is** *dist-riemann-sphere-r3*
done

lemma *dist-riemann-sphere-ge-0* [*simp*]:
shows *dist-riemann-sphere′ M1 M2* ≥ *0*
apply *transfer*
using *norm-ge-zero*
by (*simp add*: *split-def Let-def*)

Using stereographic projection we prove the connection between chordal metric on the spehere and Fubini-Study metric in the plane.

lemma *dist-stereographic-finite*:
assumes *stereographic M1* = *of-complex m1* **and** *stereographic M2* = *of-complex m2*
shows *dist-riemann-sphere′ M1 M2* = *2* ∗ *cmod* (*m1* − *m2*) / (*sqrt* (*1* + (*cmod m1*)^2) ∗ *sqrt* (*1* + (*cmod m2*)^2))
using *assms*
proof−
have ∗: *M1* = *inv-stereographic* (*of-complex m1*) *M2* = *inv-stereographic* (*of-complex m2*)
using *inv-stereographic-is-inv assms*
by (*metis inv-stereographic-stereographic*)+
have (*1* + (*cmod m1*)^2) ≠ *0* (*1* + (*cmod m2*)^2) ≠ *0*
by (*smt* (*verit*) *power2-less-0*)+
have (*1* + (*cmod m1*)^2) > *0* (*1* + (*cmod m2*)^2) > *0*
by (*smt* (*verit*) *realpow-square-minus-le*)+
hence (*1* + (*cmod m1*)^2) ∗ (*1* + (*cmod m2*)^2) > *0*
by (*metis norm-mult-less norm-zero power2-eq-square zero-power2*)
hence ++: *sqrt* ((*1* + *cmod m1* ∗ *cmod m1*) ∗ (*1* + *cmod m2* ∗ *cmod m2*)) > *0*
using *real-sqrt-gt-0-iff*
by (*simp add*: *power2-eq-square*)
hence ∗∗: (*2* ∗ *cmod* (*m1* − *m2*) / *sqrt* ((*1* + *cmod m1* ∗ *cmod m1*) ∗ (*1* + *cmod m2* ∗ *cmod m2*))) ≥ *0* ⟷ *cmod* (*m1* − *m2*) ≥ *0*

```
    by (metis diff-self divide-nonneg-pos mult-2 norm-ge-zero norm-triangle-ineq4 norm-zero)

  have (dist-riemann-sphere' M1 M2)^2 * (1 + (cmod m1)^2) * (1 + (cmod m2)^2) = 4 * (cmod (m1 - m2))^2
    using *
  proof (transfer, transfer)
    fix m1 m2 M1 M2
    assume us: M1 ∈ unit-sphere M2 ∈ unit-sphere and
       *: M1 = inv-stereographic-cvec-r3 (of-complex-cvec m1) M2 = inv-stereographic-cvec-r3 (of-complex-cvec m2)
    have (1 + (cmod m1)^2) ≠ 0  (1 + (cmod m2)^2) ≠ 0
      by (smt (verit) power2-less-0)+
    thus (dist-riemann-sphere-r3 M1 M2)^2 * (1 + (cmod m1)^2) * (1 + (cmod m2)^2) =
         4 * (cmod (m1 - m2))^2
      apply (subst dist-riemann-sphere-r3-inner[OF us])
      apply (subst *)+
      apply (simp add: dist-riemann-sphere-r3-inner[OF us] complex-mult-cnj-cmod)
      apply (subst left-diff-distrib[of 2])
      apply (subst left-diff-distrib[of 2*(1+(cmod m1)^2)])
      apply (subst distrib-right[of - - (1 + (cmod m1)^2)])
      apply (subst distrib-right[of - - (1 + (cmod m1)^2)])
      apply simp
      apply (subst distrib-right[of - - (1 + (cmod m2)^2)])
      apply (subst distrib-right[of - - (1 + (cmod m2)^2)])
      apply (subst distrib-right[of - - (1 + (cmod m2)^2)])
      apply simp
      apply (subst (asm) cmod-square)+
      apply (subst cmod-square)+
      apply (simp add: field-simps)
      done
  qed
  hence (dist-riemann-sphere' M1 M2)^2 = 4 * (cmod (m1 - m2))^2 / ((1 + (cmod m1)^2) * (1 + (cmod m2)^2))
    using ‹(1 + (cmod m1)^2) ≠ 0›  ‹(1 + (cmod m2)^2) ≠ 0›
    using eq-divide-imp[of (1 + (cmod m1)^2) * (1 + (cmod m2)^2) (dist-riemann-sphere' M1 M2)^2 4 * (cmod (m1 -
m2))^2]
    by simp
  thus dist-riemann-sphere' M1 M2 = 2 * cmod (m1 - m2) / (sqrt (1 + (cmod m1)^2) * sqrt (1 + (cmod m2)^2))
    using power2-eq-iff[of dist-riemann-sphere' M1 M2 2 * (cmod (m1 - m2)) / sqrt ((1 + (cmod m1)^2) * (1 + (cmod
m2)^2))]
    using ‹(1 + (cmod m1)^2) * (1 + (cmod m2)^2) > 0›  ‹(1 + (cmod m1)^2) > 0› ‹(1 + (cmod m2)^2) > 0›
    apply (auto simp add: power2-eq-square real-sqrt-mult[symmetric])
    using dist-riemann-sphere-ge-0[of M1 M2] **
    using ++ divide-le-0-iff by force
qed


lemma dist-stereographic-infinite:
  assumes stereographic M1 = ∞_h and stereographic M2 = of-complex m2
  shows dist-riemann-sphere' M1 M2 = 2 / sqrt (1 + (cmod m2)^2)
proof−
  have *: M1 = inv-stereographic ∞_h  M2 = inv-stereographic (of-complex m2)
    using inv-stereographic-is-inv assms
    by (metis inv-stereographic-stereographic)+
  have (1 + (cmod m2)^2) ≠ 0
    by (smt (verit) power2-less-0)
  have (1 + (cmod m2)^2) > 0
    by (smt (verit) realpow-square-minus-le)+
  hence sqrt (1 + cmod m2 * cmod m2) > 0
    using real-sqrt-gt-0-iff
    by (simp add: power2-eq-square)
  hence **: 2 / sqrt (1 + cmod m2 * cmod m2) > 0
    by simp

  have (dist-riemann-sphere' M1 M2)^2 * (1 + (cmod m2)^2) = 4
    using *
    apply transfer
    apply transfer
  proof−
```

```
    fix M1 M2 m2
    assume us: M1 ∈ unit-sphere M2 ∈ unit-sphere and
          *: M1 = inv-stereographic-cvec-r3 ∞v M2 = inv-stereographic-cvec-r3 (of-complex-cvec m2)
    have (1 + (cmod m2)²) ≠ 0
      by (smt (verit) power2-less-0)
    thus (dist-riemann-sphere-r3 M1 M2)² * (1 + (cmod m2)²) = 4
      apply (subst dist-riemann-sphere-r3-inner[OF us])
      apply (subst *)+
      apply (simp add: complex-mult-cnj-cmod)
      apply (subst left-diff-distrib[of 2], simp)
      done
  qed
  hence (dist-riemann-sphere' M1 M2)² = 4 / (1 + (cmod m2)²)
    using ‹(1 + (cmod m2)²) ≠ 0›
    by (simp add: field-simps)
  thus dist-riemann-sphere' M1 M2 = 2 / sqrt (1 + (cmod m2)²)
    using power2-eq-iff[of dist-riemann-sphere' M1 M2 2 / sqrt (1 + (cmod m2)²)]
    using ‹(1 + (cmod m2)²) > 0›
    apply (auto simp add: power2-eq-square real-sqrt-mult[symmetric])
    using dist-riemann-sphere-ge-0[of M1 M2] **
    by simp
qed

lemma dist-rieman-sphere-zero [simp]:
  shows dist-riemann-sphere' M M = 0
  by transfer auto

lemma dist-riemann-sphere-sym:
  shows dist-riemann-sphere' M1 M2 = dist-riemann-sphere' M2 M1
proof transfer
  fix M1 M2 :: R3
  obtain x1 y1 z1 x2 y2 z2 where MM: (x1, y1, z1) = M1 (x2, y2, z2) = M2
    by (cases M1, cases M2, auto)
  show dist-riemann-sphere-r3 M1 M2 = dist-riemann-sphere-r3 M2 M1
    using norm-minus-cancel[of (x1 − x2, y1 − y2, z1 − z2)] MM[symmetric]
    by simp
qed
```

Central theorem that connects the two metrics.

```
lemma dist-stereographic:
  shows dist-riemann-sphere' M1 M2 = dist-fs (stereographic M1) (stereographic M2)
proof (cases M1 = North)
  case True
  hence stereographic M1 = ∞h
    by (simp add: stereographic-North)
  show ?thesis
  proof (cases M2 = North)
    case True
    show ?thesis
      using ‹M1 = North› ‹M2 = North›
      by auto
  next
    case False
    hence stereographic M2 ≠ ∞h
      using stereographic-North[of M2]
      by simp
    then obtain m2 where stereographic M2 = of-complex m2
      using inf-or-of-complex[of stereographic M2]
      by auto
    show ?thesis
      using ‹stereographic M2 = of-complex m2› ‹stereographic M1 = ∞h›
      using dist-fs-infinite1 dist-stereographic-infinite
      by (simp add: dist-fs-sym)
  qed
next
  case False
```

```
  hence stereographic M1 ≠ ∞h
    by (simp add: stereographic-North)
  then obtain m1 where stereographic M1 = of-complex m1
    using inf-or-of-complex[of stereographic M1]
    by auto
  show ?thesis
  proof (cases M2 = North)
    case True
    hence stereographic M2 = ∞h
      by (simp add: stereographic-North)
    show ?thesis
      using ‹stereographic M1 = of-complex m1› ‹stereographic M2 = ∞h›
      using dist-fs-infinite2 dist-stereographic-infinite
      by (subst dist-riemann-sphere-sym, simp add: dist-fs-sym)
  next
    case False
    hence stereographic M2 ≠ ∞h
      by (simp add: stereographic-North)
    then obtain m2 where stereographic M2 = of-complex m2
      using inf-or-of-complex[of stereographic M2]
      by auto
    show ?thesis
      using ‹stereographic M1 = of-complex m1› ‹stereographic M2 = of-complex m2›
      using dist-fs-finite dist-stereographic-finite
      by simp
  qed
qed
```

Other direction

```
lemma dist-stereographic':
  shows dist-fs A B = dist-riemann-sphere' (inv-stereographic A) (inv-stereographic B)
  by (subst dist-stereographic) (metis stereographic-inv-stereographic)
```

The *riemann-sphere* equipped with *dist-riemann-sphere'* is a metric space, i.e., an instantiation of the *metric-space* locale.

```
instantiation riemann-sphere :: metric-space
begin
definition dist-riemann-sphere = dist-riemann-sphere'
definition (uniformity-riemann-sphere :: (riemann-sphere × riemann-sphere) filter) = (INF e∈{0<..}. principal {(x, y). dist-class.dist x y < e})
definition open-riemann-sphere (U :: riemann-sphere set) = (∀ x ∈ U. eventually (λ(x', y). x' = x ⟶ y ∈ U) uniformity)
instance
proof
  fix x y :: riemann-sphere
  show (dist-class.dist x y = 0) = (x = y)
    unfolding dist-riemann-sphere-def
  proof transfer
    fix x y :: R3
    obtain x1 y1 z1 x2 y2 z2 where *: (x1, y1, z1) = x (x2, y2, z2) = y
      by (cases x, cases y, auto)
    assume x ∈ unit-sphere y ∈ unit-sphere
    thus (dist-riemann-sphere-r3 x y = 0) = (x = y)
      using norm-eq-zero[of (x1 − y2, y1 − y2, z1 − z2)] using *[symmetric]
      by (simp add: zero-prod-def)
  qed
next
  fix x y z :: riemann-sphere
  show dist-class.dist x y ≤ dist-class.dist x z + dist-class.dist y z
    unfolding dist-riemann-sphere-def
  proof transfer
    fix x y z :: R3
    obtain x1 y1 z1 x2 y2 z2 x3 y3 z3 where MM: (x1, y1, z1) = x (x2, y2, z2) = y (x3, y3, z3) = z
      by (cases x, cases y, cases z, auto)

    assume x ∈ unit-sphere y ∈ unit-sphere z ∈ unit-sphere
```

```
    thus dist-riemann-sphere-r3 x y ≤ dist-riemann-sphere-r3 x z + dist-riemann-sphere-r3 y z
      using MM[symmetric] norm-minus-cancel[of (x3 − x2, y3 − y2, z3 − z2)] norm-triangle-ineq[of (x1 − x3, y1
− y3, z1 − z3) (x3 − x2, y3 − y2, z3 − z2)]
      by simp
  qed
qed (simp-all add: uniformity-riemann-sphere-def open-riemann-sphere-def)
end
```

The *riemann-sphere* metric space is perfect, i.e., it does not have isolated points.

```
instantiation riemann-sphere :: perfect-space
begin
instance proof
  fix M :: riemann-sphere
  show ¬ open {M}
    unfolding open-dist dist-riemann-sphere-def
    apply (subst dist-riemann-sphere-sym)
  proof transfer
    fix M
    assume M ∈ unit-sphere
    obtain x y z where MM: M = (x, y, z)
      by (cases M) auto
    then obtain α β where ∗: x = cos α ∗ cos β y = cos α ∗ sin β z = sin α −pi / 2 ≤ α ∧ α ≤ pi / 2
      using ‹M ∈ unit-sphere›
      using ex-sphere-params[of x y z]
      by auto
    have ⋀ e. e > 0 ⟹ (∃ y. y ∈ unit-sphere ∧ dist-riemann-sphere-r3 M y < e ∧ y ≠ M)
    proof−
      fix e :: real
      assume e > 0
      then obtain α′ where 1 − (e∗e/2) < cos (α − α′) α ≠ α′ −pi/2 ≤ α′ α′ ≤ pi/2
        using ex-cos-gt[of α 1 − (e∗e/2)] ‹− pi / 2 ≤ α ∧ α ≤ pi / 2›
        by auto
      hence sin α ≠ sin α′
        using ‹−pi / 2 ≤ α ∧ α ≤ pi / 2› sin-inj[of α α′]
        by auto

      have 2 − 2 ∗ cos (α − α′) < e∗e
        using mult-strict-right-mono[OF ‹1 − (e∗e/2) < cos (α − α′)›, of 2]
        by (simp add: field-simps)
      have 2 − 2 ∗ cos (α − α′) ≥ 0
        using cos-le-one[of α − α′]
        by (simp add: algebra-split-simps)
      let ?M′ = (cos α′ ∗ cos β,  cos α′ ∗ sin β, sin α′)
      have dist-riemann-sphere-r3 M ?M′ = sqrt ((cos α − cos α′)² + (sin α − sin α′)²)
        using MM ∗ sphere-params-on-sphere[of - α′ β]
        using sin-cos-squared-add[of β]
        apply (simp add: dist-riemann-sphere′-def Abs-riemann-sphere-inverse norm-prod-def)
        apply (subst left-diff-distrib[symmetric])+
        apply (subst power-mult-distrib)+
        apply (subst distrib-left[symmetric])
        apply simp
        done
      also have ... = sqrt (2 − 2∗cos (α − α′))
        by (simp add: power2-eq-square field-simps cos-diff)
      finally
      have (dist-riemann-sphere-r3 M ?M′)² = 2 − 2∗cos (α − α′)
        using ‹2 − 2 ∗ cos (α − α′) ≥ 0›
        by simp
      hence (dist-riemann-sphere-r3 M ?M′)² < e²
        using ‹2 − 2 ∗ cos (α − α′) < e∗e›
        by (simp add: power2-eq-square)
      hence dist-riemann-sphere-r3 M ?M′ < e
        apply (rule power2-less-imp-less)
        using ‹e > 0›
        by simp
      moreover
```

```
    have M ≠ ?M'
      using MM ‹sin α ≠ sin α'› *
      by simp
    moreover
    have ?M' ∈ unit-sphere
      using sphere-params-on-sphere by auto
    ultimately
    show ∃ y. y ∈ unit-sphere ∧ dist-riemann-sphere-r3 M y < e ∧ y ≠ M
      unfolding dist-riemann-sphere-def
      by (rule-tac x=?M' in exI, simp)
  qed
  thus ¬ (∀ x∈{M}. ∃ e>0. ∀ y∈{x. x ∈ unit-sphere}. dist-riemann-sphere-r3 x y < e ⟶ y ∈ {M})
    by auto
 qed
qed
end
```

The *complex-homo* metric space is perfect, i.e., it does not have isolated points.

```
instantiation complex-homo :: perfect-space
begin
instance proof
 fix x::complex-homo
 show ¬ open {x}
   unfolding open-dist
 proof (auto)
   fix e::real
   assume e > 0
   thus ∃ y. dist-class.dist y x < e ∧ y ≠ x
     using not-open-singleton[of inv-stereographic x]
     unfolding open-dist
     unfolding dist-complex-homo-def dist-riemann-sphere-def
     apply (subst dist-stereographic', auto)
     apply (erule-tac x=e in allE, auto)
     apply (rule-tac x=stereographic y in exI, auto)
     done
 qed
qed

end

lemma Lim-within:
 shows (f ⟶⟶ l) (at a within S) ⟷
   (∀ e >0. ∃ d>0. ∀ x ∈ S. 0 < dist-class.dist x a ∧ dist-class.dist x a  < d ⟶ dist-class.dist (f x) l < e)
 by (auto simp: tendsto-iff eventually-at)

lemma continuous-on-iff:
 shows continuous-on s f ⟷
   (∀ x∈s. ∀ e>0. ∃ d>0. ∀ x'∈s. dist-class.dist x' x < d ⟶ dist-class.dist (f x') (f x) < e)
 unfolding continuous-on-def Lim-within
 by (metis dist-pos-lt dist-self)
```

Using the chordal metric in the extended plane, and the Euclidean metric on the sphere in $\mathbb{R}^3$, the stereographic and inverse stereographic projections are proved to be continuous.

```
lemma continuous-on UNIV stereographic
unfolding continuous-on-iff
unfolding dist-complex-homo-def dist-riemann-sphere-def
by (subst dist-stereographic', auto)

lemma continuous-on UNIV inv-stereographic
unfolding continuous-on-iff
unfolding dist-complex-homo-def dist-riemann-sphere-def
by (subst dist-stereographic, auto)
```

10.5.6 Chordal circles

Real circlines are sets of points that are equidistant from some given point in the chordal metric. There are exactly two such points (two chordal centers). On the Riemann sphere, these two points are obtained as intersections of the sphere and a line that goes trough center of the circle, and its orthogonal to its plane.

The circline for the given chordal center and radius.

definition *chordal-circle-cvec-cmat* :: *complex-vec* ⇒ *real* ⇒ *complex-mat* **where**
[*simp*]: *chordal-circle-cvec-cmat a r* =
(*let* (*a1*, *a2*) = *a*
in ((4∗*a2*∗*cnj a2* − (*cor r*)²∗(*a1*∗*cnj a1* + *a2*∗*cnj a2*)), (−4∗*a1*∗*cnj a2*), (−4∗*cnj a1*∗*a2*), (4∗*a1*∗*cnj a1* − (*cor r*)²∗(*a1*∗*cnj a1* + *a2*∗*cnj a2*))))

lemma *chordal-circle-cmat-hermitean-nonzero* [*simp*]:
assumes *a* ≠ *vec-zero*
shows *chordal-circle-cvec-cmat a r* ∈ *hermitean-nonzero*
using *assms*
by (*cases a*) (*auto simp add*: *hermitean-def mat-adj-def mat-cnj-def Let-def*)

lift-definition *chordal-circle-hcoords-clmat* :: *complex-homo-coords* ⇒ *real* ⇒ *circline-mat* **is** *chordal-circle-cvec-cmat*
using *chordal-circle-cmat-hermitean-nonzero*
by (*simp del*: *chordal-circle-cvec-cmat-def*)

lift-definition *chordal-circle* :: *complex-homo* ⇒ *real* ⇒ *circline* **is** *chordal-circle-hcoords-clmat*
proof *transfer*
fix *a b* :: *complex-vec* **and** *r* :: *real*
assume ∗: *a* ≠ *vec-zero b* ≠ *vec-zero*
obtain *a1 a2* **where** *aa*: *a* = (*a1*, *a2*)
by (*cases a*, *auto*)
obtain *b1 b2* **where** *bb*: *b* = (*b1*, *b2*)
by (*cases b*, *auto*)
assume *a* ≈*v* *b*
then obtain *k* **where** *b* = (*k* ∗ *a1*, *k* ∗ *a2*) *k* ≠ *0*
using *aa bb*
by *auto*
moreover
have *cor* (*Re* (*k* ∗ *cnj k*)) = *k* ∗ *cnj k*
by (*metis complex-In-mult-cnj-zero complex-of-real-Re*)
ultimately
show *circline-eq-cmat* (*chordal-circle-cvec-cmat a r*) (*chordal-circle-cvec-cmat b r*)
using ∗ *aa bb*
by *simp* (*rule-tac x*=*Re* (*k*∗*cnj k*) **in** *exI*, *auto simp add*: *Let-def field-simps*)
qed

lemma *sqrt-1-plus-square*:
shows *sqrt* (*1* + *a*²) ≠ *0*
by (*smt* (*verit*) *real-sqrt-less-mono real-sqrt-zero realpow-square-minus-le*)

lemma
assumes *dist-fs z a* = *r*
shows *z* ∈ *circline-set* (*chordal-circle a r*)
proof (*cases a* ≠ ∞ₕ)
case *True*
then obtain *a'* **where** *a* = *of-complex a'*
using *inf-or-of-complex*
by *auto*
let *?A* = *4* − (*cor r*)² ∗ (*1* + (*a'*∗*cnj a'*)) **and** *?B* = −*4*∗*a'* **and** *?C*=−*4*∗*cnj a'* **and** *?D* = *4*∗*a'*∗*cnj a'* − (*cor r*)² ∗ (*1* + (*a'*∗*cnj a'*))
have *hh*: (*?A*, *?B*, *?C*, *?D*) ∈ {*H*. *hermitean H* ∧ *H* ≠ *mat-zero*}
by (*auto simp add*: *hermitean-def mat-adj-def mat-cnj-def power2-eq-square*)
hence ∗: *chordal-circle* (*of-complex a'*) *r* = *mk-circline ?A ?B ?C ?D*
by (*transfer*, *transfer*, *simp*, *rule-tac x=1* **in** *exI*, *simp*)

show *?thesis*
proof (*cases z* ≠ ∞ₕ)
case *True*

then obtain z' **where** $z = \textit{of-complex}\ z'$
using *inf-or-of-complex*[*of z*] *inf-or-of-complex*[*of a*]
by *auto*
have $2 * \textit{cmod}\ (z' - a') / (\textit{sqrt}\ (1 + (\textit{cmod}\ z')^2) * \textit{sqrt}\ (1 + (\textit{cmod}\ a')^2)) = r$
using *dist-fs-finite*[*of z' a'*] *assms* ‹$z = \textit{of-complex}\ z'$› ‹$a = \textit{of-complex}\ a'$›
by *auto*
hence $4 * (\textit{cmod}\ (z' - a'))^2 / ((1 + (\textit{cmod}\ z')^2) * (1 + (\textit{cmod}\ a')^2)) = r^2$
by (*auto simp add*: *field-simps*)
moreover
have $\textit{sqrt}\ (1 + (\textit{cmod}\ z')^2) \neq 0\ \textit{sqrt}\ (1 + (\textit{cmod}\ a')^2) \neq 0$
using *sqrt-1-plus-square*
by *simp+*
hence $(1 + (\textit{cmod}\ z')^2) * (1 + (\textit{cmod}\ a')^2) \neq 0$
by *simp*
ultimately
have $4 * (\textit{cmod}\ (z' - a'))^2 = r^2 * ((1 + (\textit{cmod}\ z')^2) * (1 + (\textit{cmod}\ a')^2))$
by (*simp add*: *field-simps*)
hence $4 * \textit{Re}\ ((z' - a') * \textit{cnj}\ (z' - a')) = r^2 * (1 + \textit{Re}\ (z' * \textit{cnj}\ z')) * (1 + \textit{Re}\ (a' * \textit{cnj}\ a'))$
by ((*subst cmod-square*[*symmetric*])+, *simp*)
hence $4 * (\textit{Re}(z' * \textit{cnj}\ z') - \textit{Re}(a' * \textit{cnj}\ z') - \textit{Re}(\textit{cnj}\ a' * z') + \textit{Re}(a' * \textit{cnj}\ a')) = r^2 * (1 + \textit{Re}\ (z' * \textit{cnj}\ z')) * (1 + \textit{Re}\ (a' * \textit{cnj}\ a'))$
by (*simp add*: *field-simps*)
hence $\textit{Re}\ (?A * z' * \textit{cnj}\ z' + ?B * \textit{cnj}\ z' + ?C * z' + ?D) = 0$
by (*simp add*: *power2-eq-square field-simps*)
hence $?A * z' * \textit{cnj}\ z' + ?B * \textit{cnj}\ z' + ?C * z' + ?D = 0$
by (*subst complex-eq-if-Re-eq*) (*auto simp add*: *power2-eq-square*)
hence $(\textit{cnj}\ z' * ?A + ?C) * z' + (\textit{cnj}\ z' * ?B + ?D) = 0$
by *algebra*
hence $z \in \textit{circline-set}\ (\textit{mk-circline}\ ?A\ ?B\ ?C\ ?D)$
using ‹$z = \textit{of-complex}\ z'$› *hh*
unfolding *circline-set-def*
by *simp* (*transfer*, *transfer*, *simp add*: *vec-cnj-def*)
thus *?thesis*
using $*$
by (*subst* ‹$a = \textit{of-complex}\ a'$›) *simp*
next
case *False*
hence $2 / \textit{sqrt}\ (1 + (\textit{cmod}\ a')^2) = r$
using *assms* ‹$a = \textit{of-complex}\ a'$›
using *dist-fs-infinite2*[*of a'*]
by *simp*
moreover
have $\textit{sqrt}\ (1 + (\textit{cmod}\ a')^2) \neq 0$
using *sqrt-1-plus-square*
by *simp*
ultimately
have $2 = r * \textit{sqrt}\ (1 + (\textit{cmod}\ a')^2)$
by (*simp add*: *field-simps*)
hence $4 = (r * \textit{sqrt}\ (1 + (\textit{cmod}\ a')^2))^2$
by *simp*
hence $4 = r^2 * (1 + (\textit{cmod}\ a')^2)$
by (*simp add*: *power-mult-distrib*)
hence $\textit{Re}\ (4 - (\textit{cor}\ r)^2 * (1 + (a' * \textit{cnj}\ a'))) = 0$
by (*subst* (*asm*) *cmod-square*) (*simp add*: *field-simps power2-eq-square*)
hence $4 - (\textit{cor}\ r)^2 * (1 + (a' * \textit{cnj}\ a')) = 0$
by (*subst complex-eq-if-Re-eq*) (*auto simp add*: *power2-eq-square*)
hence *circline-A0* (*mk-circline ?A ?B ?C ?D*)
using *hh*
by (*simp*, *transfer*, *transfer*, *simp*)
hence $z \in \textit{circline-set}\ (\textit{mk-circline}\ ?A\ ?B\ ?C\ ?D)$
using *inf-in-circline-set*[*of mk-circline ?A ?B ?C ?D*]
using ‹$\neg\ z \neq \infty_h$›
by *simp*
thus *?thesis*
using $*$
by (*subst* ‹$a = \textit{of-complex}\ a'$›) *simp*

```
  qed
next
  case False
  let ?A = -(cor r)^2 and ?B = 0 and ?C = 0 and ?D = 4 -(cor r)^2
  have hh: (?A, ?B, ?C, ?D) ∈ {H. hermitean H ∧ H ≠ mat-zero}
    by (auto simp add: hermitean-def mat-adj-def mat-cnj-def power2-eq-square)
  hence *: chordal-circle a r = mk-circline ?A ?B ?C ?D
    using ‹¬ a ≠ ∞h›
    by simp (transfer, transfer, simp, rule-tac x=1 in exI, simp)

  show ?thesis
  proof (cases z = ∞h)
    case True
    show ?thesis
      using assms ‹z = ∞h› ‹¬ a ≠ ∞h›
      using * hh
      by (simp, subst inf-in-circline-set, transfer, transfer, simp)
  next
    case False
    then obtain z' where z = of-complex z'
      using inf-or-of-complex[of z]
      by auto
    have 2 / sqrt (1 + (cmod z')^2) = r
      using assms ‹z = of-complex z'›‹¬ a ≠ ∞h›
      using dist-fs-infinite2[of z']
      by (simp add: dist-fs-sym)
    moreover
    have sqrt (1 + (cmod z')^2) ≠ 0
      using sqrt-1-plus-square
      by simp
    ultimately
    have 2 = r * sqrt (1 + (cmod z')^2)
      by (simp add: field-simps)
    hence 4 = (r * sqrt (1 + (cmod z')^2))^2
      by simp
    hence 4 = r^2 * (1 + (cmod z')^2)
      by (simp add: power-mult-distrib)
    hence Re (4 - (cor r)^2 * (1 + (z' * cnj z'))) = 0
      by (subst (asm) cmod-square) (simp add: field-simps power2-eq-square)
    hence - (cor r)^2 * z'*cnj z' + 4 - (cor r)^2 = 0
      by (subst complex-eq-if-Re-eq) (auto simp add: power2-eq-square field-simps)
    hence z ∈ circline-set (mk-circline ?A ?B ?C ?D)
      using hh
      unfolding circline-set-def
      by (subst ‹z = of-complex z'›, simp) (transfer, transfer, auto simp add: vec-cnj-def field-simps)
    thus ?thesis
      using *
      by simp
  qed
qed

lemma
  assumes z ∈ circline-set (chordal-circle a r) and r ≥ 0
  shows dist-fs z a = r
proof (cases a = ∞h)
  case False
  then obtain a' where a = of-complex a'
    using inf-or-of-complex
    by auto

  let ?A = 4 - (cor r)^2 * (1 + (a'*cnj a')) and ?B = -4*a' and ?C=-4*cnj a' and ?D = 4*a'*cnj a' - (cor r)^2
* (1 + (a'*cnj a'))
  have hh: (?A, ?B, ?C, ?D) ∈ {H. hermitean H ∧ H ≠ mat-zero}
    by (auto simp add: hermitean-def mat-adj-def mat-cnj-def power2-eq-square)
  hence *: chordal-circle (of-complex a') r = mk-circline ?A ?B ?C ?D
    by (transfer, transfer, simp, rule-tac x=1 in exI, simp)
```

show *?thesis*
proof (*cases* $z = \infty_h$)
case *False*
then obtain z' **where** $z =$ *of-complex* z'
using *inf-or-of-complex*[*of z*] *inf-or-of-complex*[*of a*]
by *auto*
hence $z \in$ *circline-set* (*mk-circline ?A ?B ?C ?D*)
using *assms* ‹$a =$ *of-complex* a'› $*$
by *simp*
hence (*cnj* $z' * ?A + ?C) * z' + (cnj\ z' * ?B + ?D) = 0$
using *hh*
unfolding *circline-set-def*
by (*subst* (*asm*) ‹$z =$ *of-complex* z'›, *simp*) (*transfer*, *transfer*, *simp add*: *vec-cnj-def*)
hence $?A * z' * cnj\ z' + ?B * cnj\ z' + ?C * z' + ?D = 0$
by *algebra*
hence *Re* $(?A * z' * cnj\ z' + ?B * cnj\ z' + ?C * z' + ?D) = 0$
by (*simp add*: *power2-eq-square field-simps*)
hence $4 * Re\ ((z' - a') * cnj\ (z' - a')) = r^2 * (1 + Re\ (z' * cnj\ z')) * (1 + Re\ (a' * cnj\ a'))$
by (*simp add*: *field-simps power2-eq-square*)
hence $4 * (cmod\ (z' - a'))^2 = r^2 * ((1 + (cmod\ z')^2) * (1 + (cmod\ a')^2))$
by (*subst cmod-square*)+ *simp*
moreover
have $sqrt\ (1 + (cmod\ z')^2) \neq 0\ sqrt\ (1 + (cmod\ a')^2) \neq 0$
using *sqrt-1-plus-square*
by *simp*+
hence $(1 + (cmod\ z')^2) * (1 + (cmod\ a')^2) \neq 0$
by *simp*
ultimately
have $4 * (cmod\ (z' - a'))^2 / ((1 + (cmod\ z')^2) * (1 + (cmod\ a')^2)) = r^2$
by (*simp add*: *field-simps*)
hence $2 * cmod\ (z' - a') / (sqrt\ (1 + (cmod\ z')^2) * sqrt\ (1 + (cmod\ a')^2)) = r$
using ‹$r \geq 0$›
by (*subst* (*asm*) *real-sqrt-eq-iff*[*symmetric*]) (*simp add*: *real-sqrt-mult real-sqrt-divide*)
thus *?thesis*
using ‹$z =$ *of-complex* z'› ‹$a =$ *of-complex* a'›
using *dist-fs-finite*[*of* z' a']
by *simp*
next
case *True*
have $z \in$ *circline-set* (*mk-circline ?A ?B ?C ?D*)
using *assms* ‹$a =$ *of-complex* a'› $*$
by *simp*
hence *circline-A0* (*mk-circline ?A ?B ?C ?D*)
using *inf-in-circline-set*[*of mk-circline ?A ?B ?C ?D*]
using ‹$z = \infty_h$›
by *simp*
hence $4 - (cor\ r)^2 * (1 + (a' * cnj\ a')) = 0$
using *hh*
by (*transfer*, *transfer*, *simp*)
hence $Re\ (4 - (cor\ r)^2 * (1 + (a' * cnj\ a'))) = 0$
by *simp*
hence $4 = r^2 * (1 + (cmod\ a')^2)$
by (*subst cmod-square*) (*simp add*: *power2-eq-square*)
hence $2 = r * sqrt\ (1 + (cmod\ a')^2)$
using ‹$r \geq 0$›
by (*subst* (*asm*) *real-sqrt-eq-iff*[*symmetric*]) (*simp add*: *real-sqrt-mult*)
moreover
have $sqrt\ (1 + (cmod\ a')^2) \neq 0$
using *sqrt-1-plus-square*
by *simp*
ultimately
have $2 / sqrt\ (1 + (cmod\ a')^2) = r$
by (*simp add*: *field-simps*)
thus *?thesis*
using ‹$a =$ *of-complex* a'› ‹$z = \infty_h$›

```
      using dist-fs-infinite2[of a']
      by simp
  qed
next
  case True
  let ?A = -(cor r)^2 and ?B = 0 and ?C = 0 and ?D = 4 -(cor r)^2
  have hh: (?A, ?B, ?C, ?D) ∈ {H. hermitean H ∧ H ≠ mat-zero}
    by (auto simp add: hermitean-def mat-adj-def mat-cnj-def power2-eq-square)
  hence *: chordal-circle a r = mk-circline ?A ?B ?C ?D
    using ‹a = ∞h›
    by simp (transfer, transfer, simp, rule-tac x=1 in exI, simp)

  show ?thesis
  proof (cases z = ∞h)
    case True
    thus ?thesis
      using ‹a = ∞h› assms * hh
      by simp (subst (asm) inf-in-circline-set, transfer, transfer, simp)
  next
    case False
    then obtain z' where z = of-complex z'
      using inf-or-of-complex
      by auto
    hence z ∈ circline-set (mk-circline ?A ?B ?C ?D)
      using assms *
      by simp
    hence - (cor r)^2 * z'*cnj z' + 4 - (cor r)^2 = 0
      using hh
      unfolding circline-set-def
      apply (subst (asm) ‹z = of-complex z'›)
      by (simp, transfer, transfer, simp add: vec-cnj-def, algebra)
    hence 4 - (cor r)^2 * (1 + (z'*cnj z')) = 0
      by (simp add: field-simps)
    hence Re (4 - (cor r)^2 * (1 + (z' * cnj z'))) = 0
      by simp
    hence 4 = r^2 * (1 + (cmod z')^2)
      by (subst cmod-square) (simp add: power2-eq-square)
    hence 2 = r * sqrt (1 + (cmod z')^2)
      using ‹r ≥ 0›
      by (subst (asm) real-sqrt-eq-iff[symmetric]) (simp add: real-sqrt-mult)
    moreover
    have sqrt (1 + (cmod z')^2) ≠ 0
      using sqrt-1-plus-square
      by simp
    ultimately
    have 2 / sqrt (1 + (cmod z')^2) = r
      by (simp add: field-simps)
    thus ?thesis
      using ‹z = of-complex z'› ‹a = ∞h›
      using dist-fs-infinite2[of z']
      by (simp add: dist-fs-sym)
  qed
qed
```

Two chordal centers and radii for the given circline

```
definition chordal-circles-cmat :: complex-mat ⇒ (complex × real) × (complex × real) where
 [simp]: chordal-circles-cmat H =
    (let (A, B, C, D) = H;
         dsc = sqrt(Re ((D-A)^2 + 4 * (B*cnj B)));
         a1 = (A - D + cor dsc) / (2 * C);
         r1 = sqrt((4 - Re((-4 * a1/B) * A)) / (1 + Re (a1*cnj a1)));
         a2 = (A - D - cor dsc) / (2 * C);
         r2 = sqrt((4 - Re((-4 * a2/B) * A)) / (1 + Re (a2*cnj a2)))
      in ((a1, r1), (a2, r2)))
```

```
lift-definition chordal-circles-clmat :: circline-mat ⇒ (complex × real) × (complex × real) is chordal-circles-cmat
```

done

lift-definition *chordal-circles* :: *ocircline* ⇒ (*complex* × *real*) × (*complex* × *real*) **is** *chordal-circles-clmat*
proof *transfer*
fix *H1 H2* :: *complex-mat*
obtain *A1 B1 C1 D1* **where** *hh1*: (*A1*, *B1*, *C1*, *D1*) = *H1*
by (*cases H1*) *auto*
obtain *A2 B2 C2 D2* **where** *hh2*: (*A2*, *B2*, *C2*, *D2*) = *H2*
by (*cases H2*) *auto*

assume *ocircline-eq-cmat H1 H2*
then obtain *k* **where** ∗: *k* > *0 A2* = *cor k* ∗ *A1 B2* = *cor k* ∗ *B1 C2* = *cor k* ∗ *C1 D2* = *cor k* ∗ *D1*
using *hh1*[*symmetric*] *hh2*[*symmetric*]
by *auto*
let *?dsc1* = *sqrt* (*Re* ((*D1* − *A1*)² + *4* ∗ (*B1* ∗ *cnj B1*))) **and** *?dsc2* = *sqrt* (*Re* ((*D2* − *A2*)² + *4* ∗ (*B2* ∗ *cnj B2*)))
let *?a11* = (*A1* − *D1* + *cor ?dsc1*) / (*2* ∗ *C1*) **and** *?a12* = (*A2* − *D2* + *cor ?dsc2*) / (*2* ∗ *C2*)
let *?a21* = (*A1* − *D1* − *cor ?dsc1*) / (*2* ∗ *C1*) **and** *?a22* = (*A2* − *D2* − *cor ?dsc2*) / (*2* ∗ *C2*)
let *?r11* = *sqrt*((*4* − *Re*((−*4* ∗ *?a11*/*B1*) ∗ *A1*)) / (*1* + *Re* (*?a11*∗*cnj ?a11*)))
let *?r12* = *sqrt*((*4* − *Re*((−*4* ∗ *?a12*/*B2*) ∗ *A2*)) / (*1* + *Re* (*?a12*∗*cnj ?a12*)))
let *?r21* = *sqrt*((*4* − *Re*((−*4* ∗ *?a21*/*B1*) ∗ *A1*)) / (*1* + *Re* (*?a21*∗*cnj ?a21*)))
let *?r22* = *sqrt*((*4* − *Re*((−*4* ∗ *?a22*/*B2*) ∗ *A2*)) / (*1* + *Re* (*?a22*∗*cnj ?a22*)))

have *Re* ((*D2* − *A2*)² + *4* ∗ (*B2* ∗ *cnj B2*)) = *k*² ∗ *Re* ((*D1* − *A1*)² + *4* ∗ (*B1* ∗ *cnj B1*))
using ∗
by (*simp add*: *power2-eq-square field-simps*)
hence *?dsc2* = *k* ∗ *?dsc1*
using ‹*k* > *0*›
by (*simp add*: *real-sqrt-mult*)
hence *A2* − *D2* + *cor ?dsc2* = *cor k* ∗ (*A1* − *D1* + *cor ?dsc1*) *A2* − *D2* − *cor ?dsc2* = *cor k* ∗ (*A1* − *D1* − *cor ?dsc1*) *2*∗*C2* = *cor k* ∗ (*2*∗*C1*)
using ∗
by (*auto simp add*: *field-simps*)
hence *?a12* = *?a11 ?a22* = *?a21*
using ‹*k* > *0*›
by *simp-all*
moreover
have *Re*((−*4* ∗ *?a12*/*B2*) ∗ *A2*) = *Re*((−*4* ∗ *?a11*/*B1*) ∗ *A1*)
using ∗
by (*subst* ‹*?a12* = *?a11*›) (*simp*, *simp add*: *field-simps*)
have *?r12* = *?r11*
by (*subst* ‹*Re*((−*4* ∗ *?a12*/*B2*) ∗ *A2*) = *Re*((−*4* ∗ *?a11*/*B1*) ∗ *A1*)›, (*subst* ‹*?a12* = *?a11*›)+) *simp*
moreover
have *Re*((−*4* ∗ *?a22*/*B2*) ∗ *A2*) = *Re*((−*4* ∗ *?a21*/*B1*) ∗ *A1*)
using ∗
by (*subst* ‹*?a22* = *?a21*›) (*simp*, *simp add*: *field-simps*)
have *?r22* = *?r21*
by (*subst* ‹*Re*((−*4* ∗ *?a22*/*B2*) ∗ *A2*) = *Re*((−*4* ∗ *?a21*/*B1*) ∗ *A1*)›, (*subst* ‹*?a22* = *?a21*›)+) *simp*
moreover
have *chordal-circles-cmat H1* = ((*?a11*, *?r11*), (*?a21*, *?r21*))
using *hh1*[*symmetric*]
unfolding *chordal-circles-cmat-def Let-def*
by (*simp del*: *times-complex.sel*)
moreover
have *chordal-circles-cmat H2* = ((*?a12*, *?r12*), (*?a22*, *?r22*))
using *hh2*[*symmetric*]
unfolding *chordal-circles-cmat-def Let-def*
by (*simp del*: *times-complex.sel*)
ultimately
show *chordal-circles-cmat H1* = *chordal-circles-cmat H2*
by *metis*
qed

lemma *chordal-circle-radius-positive*:
assumes *hermitean* (*A*, *B*, *C*, *D*) **and** *Re* (*mat-det* (*A*, *B*, *C*, *D*)) ≤ *0* **and** *B* ≠ *0* **and**
dsc = *sqrt*(*Re* ((*D*−*A*)² + *4* ∗ (*B*∗*cnj B*))) **and**

```
  a1 = (A − D + cor dsc) / (2 * C) and a2 = (A − D − cor dsc) / (2 * C)
  shows Re (A*a1/B) ≥ −1 ∧ Re (A*a2/B) ≥ −1
proof−
  from assms have is-real A is-real D C = cnj B
    using hermitean-elems
    by auto
  have *: A*a1/B = ((A − D + cor dsc) / (2 * (B * cnj B))) * A
    using ‹B ≠ 0› ‹C = cnj B› ‹a1 = (A − D + cor dsc) / (2 * C)›
    by (simp add: field-simps) algebra
  have **: A*a2/B = ((A − D − cor dsc) / (2 * (B * cnj B))) * A
    using ‹B ≠ 0› ‹C = cnj B› ‹a2 = (A − D − cor dsc) / (2 * C)›
    by (simp add: field-simps) algebra
  have dsc ≥ 0
  proof−
    have 0 ≤ Re ((D − A)^2) + 4 * Re ((cor (cmod B))^2)
      using ‹is-real A› ‹is-real D› by simp
    thus ?thesis
      using ‹dsc = sqrt(Re ((D−A)^2 + 4*(B*cnj B)))›
      by (subst (asm) complex-mult-cnj-cmod) simp
  qed
  hence Re (A − D − cor dsc) ≤ Re (A − D + cor dsc)
    by simp
  moreover
  have Re (2 * (B * cnj B)) > 0
    using ‹B ≠ 0›
    by (subst complex-mult-cnj-cmod, simp add: power2-eq-square)
  ultimately
  have xxx: Re (A − D + cor dsc) / Re (2 * (B * cnj B)) ≥ Re (A − D − cor dsc) / Re (2 * (B * cnj B)) (is ?lhs ≥
?rhs)
    by (metis divide-right-mono less-eq-real-def)

  have Re A * Re D ≤ Re (B*cnj B)
    using ‹Re (mat-det (A, B, C, D)) ≤ 0› ‹C = cnj B› ‹is-real A› ‹is-real D›
    by simp


  have (Re (2 * (B * cnj B)) / Re A) / Re (2 * B * cnj B) = 1 / Re A
    using ‹Re (2 * (B * cnj B)) > 0›
    apply (subst divide-divide-eq-left)
    apply (subst mult.assoc)
    apply (subst nonzero-divide-mult-cancel-right)
    by simp-all

  show ?thesis
  proof (cases Re A > 0)
    case True
    hence Re (A*a1/B) ≥ Re (A*a2/B)
      using * ** ‹Re (2 * (B * cnj B)) > 0› ‹B ≠ 0› ‹is-real A› ‹is-real D› xxx
      using mult-right-mono[of ?rhs ?lhs Re A]
      apply simp
      apply (subst Re-divide-real, simp, simp)
      apply (subst Re-divide-real, simp, simp)
      apply (subst Re-mult-real, simp)+
      apply simp
      done
    moreover
    have Re (A*a2/B) ≥ −1
    proof−
      from ‹Re A * Re D ≤ Re (B*cnj B)›
      have Re (A^2) ≤ Re (B*cnj B) + Re ((A − D)*A)
        using ‹Re A > 0› ‹is-real A› ‹is-real D›
        by (simp add: power2-eq-square field-simps)
      have 1 ≤ Re (B*cnj B) / Re (A^2) + Re (A − D) / Re A
        using ‹Re A > 0› ‹is-real A› ‹is-real D›
        using divide-right-mono[OF ‹Re (A^2) ≤ Re (B*cnj B) + Re ((A − D)*A)›, of Re (A^2)]
        by (simp add: power2-eq-square add-divide-distrib)
```

have *4 * Re(B*cnj B) ≤ 4 * (Re (B*cnj B))² / Re (A²) + 2*Re (A − D) / Re A * 2 * Re(B*cnj B)*
using *mult-right-mono[OF ‹1 ≤ Re (B*cnj B) / Re (A²) + Re (A − D) / Re A›, of 4 * Re (B*cnj B)]*
by (*simp add*: *distrib-right*) (*simp add*: *power2-eq-square field-simps*)
moreover
have *A ≠ 0*
using ‹*Re A > 0*›
by *auto*
hence *4 * (Re (B*cnj B))² / Re (A²) = Re (4 * (B*cnj B)² / A²)*
using *Re-divide-real[of A² 4 * (B*cnj B)²] ‹Re A > 0› ‹is-real A›*
by (*auto simp add*: *power2-eq-square*)
moreover
have *2*Re (A − D) / Re A * 2 * Re(B*cnj B) = Re (2 * (A − D) / A * 2 * B * cnj B)*
using ‹*is-real A*› ‹*is-real D*› ‹*A ≠ 0*›
using *Re-divide-real[of A (4 * A − 4 * D) * B * cnj B]*
by (*simp add*: *field-simps*)
ultimately
have *Re ((D − A)² + 4 * B*cnj B) ≤ Re((A − D)² + 4 * (B*cnj B)² / A² + 2*(A − D) / A * 2 * B*cnj B)*
by (*simp add*: *field-simps power2-eq-square*)
hence *Re ((D − A)² + 4 * B*cnj B) ≤ Re(((A − D) + 2 * B*cnj B / A)²)*
using ‹*A ≠ 0*›
by (*subst power2-sum*) (*simp add*: *power2-eq-square field-simps*)
hence *dsc ≤ sqrt (Re(((A − D) + 2 * B*cnj B / A)²))*
using ‹*dsc = sqrt(Re ((D−A)² + 4*(B*cnj B)))*›
by *simp*
moreover
have *Re(((A − D) + 2 * B*cnj B / A)²) = (Re((A − D) + 2 * B*cnj B / A))²*
using ‹*is-real A*› ‹*is-real D*› *div-reals*
by (*simp add*: *power2-eq-square*)
ultimately
have *dsc ≤ |Re (A − D + 2 * B * cnj B / A)|*
by *simp*
moreover
have *Re (A − D + 2 * B * cnj B / A) ≥ 0*
proof−
have ∗: *Re (A² + B*cnj B) ≥ 0*
using ‹*is-real A*›
by (*simp add*: *power2-eq-square*)
also have *Re (A² + 2*B*cnj B − A*D) ≥ Re (A² + B*cnj B)*
using ‹*Re A * Re D ≤ Re (B*cnj B)*›
using ‹*is-real A*› ‹*is-real D*›
by *simp*
finally
have *Re (A² + 2*B*cnj B − A*D) ≥ 0*
by *simp*
show *?thesis*
using *divide-right-mono[OF ‹Re (A² + 2*B*cnj B − A*D) ≥ 0›, of Re A] ‹Re A > 0› ‹is-real A› ‹A ≠ 0›*
by (*simp add*: *add-divide-distrib diff-divide-distrib*) (*subst Re-divide-real*, *auto simp add*: *power2-eq-square field-simps*)
qed
ultimately
have *dsc ≤ Re (A − D + 2 * B * cnj B / A)*
by *simp*
hence *− Re (2 * (B * cnj B) / A) ≤ Re ((A − D − cor dsc))*
by (*simp add*: *field-simps*)
hence ∗: *− (Re (2 * (B * cnj B)) / Re A) ≤ Re (A − D − cor dsc)*
using ‹*is-real A*› ‹*A ≠ 0*›
by (*subst* (*asm*) *Re-divide-real*, *auto*)
from *divide-right-mono[OF this, of Re (2 * B * cnj B)]*
have *− 1 / Re A ≤ Re (A − D − cor dsc) / Re (2 * B * cnj B)*
using ‹*Re A > 0*› ‹*B ≠ 0*› ‹*A ≠ 0*› ‹*0 < Re (2 * (B * cnj B))*›
using ‹*(Re (2 * (B * cnj B)) / Re A) / Re (2 * B * cnj B) = 1 / Re A*›
by *simp*
from *mult-right-mono[OF this, of Re A]*
show *?thesis*
using ‹*is-real A*› ‹*is-real D*› ‹*B ≠ 0*› ‹*Re A > 0*› ‹*A ≠ 0*›
apply (*subst* ∗∗)

```
      apply (subst Re-mult-real, simp)
      apply (subst Re-divide-real, simp, simp)
      apply (simp add: field-simps)
      done
  qed
  ultimately
  show ?thesis
    by simp
next
  case False
  show ?thesis
  proof (cases Re A < 0)
    case True
    hence Re (A*a1/B) ≤ Re (A*a2/B)
      using * ** ‹Re (2 * (B * cnj B)) > 0› ‹B ≠ 0› ‹is-real A› ‹is-real D› xxx
      using mult-right-mono-neg[of ?rhs ?lhs Re A]
      apply simp
      apply (subst Re-divide-real, simp, simp)
      apply (subst Re-divide-real, simp, simp)
      apply (subst Re-mult-real, simp)+
      apply simp
      done
    moreover
    have Re (A*a1/B) ≥ -1
    proof-
      from ‹Re A * Re D ≤ Re (B*cnj B)›
      have Re (A²) ≤ Re (B*cnj B) - Re ((D - A)*A)
        using ‹Re A < 0› ‹is-real A› ‹is-real D›
        by (simp add: power2-eq-square field-simps)
      hence 1 ≤ Re (B*cnj B) / Re (A²) - Re (D - A) / Re A
        using ‹Re A < 0› ‹is-real A› ‹is-real D›
        using divide-right-mono[OF ‹Re (A²) ≤ Re (B*cnj B) - Re ((D - A)*A)›, of Re (A²)]
        by (simp add: power2-eq-square diff-divide-distrib)
      have 4 * Re(B*cnj B) ≤ 4 * (Re (B*cnj B))² / Re (A²) - 2*Re (D - A) / Re A * 2 * Re(B*cnj B)
        using mult-right-mono[OF ‹1 ≤ Re (B*cnj B) / Re (A²) - Re (D - A) / Re A›, of 4 * Re (B*cnj B)]
        by (simp add: left-diff-distrib) (simp add: power2-eq-square field-simps)
      moreover
      have A ≠ 0
        using ‹Re A < 0›
        by auto
      hence 4 * (Re (B*cnj B))² / Re (A²) = Re (4 * (B*cnj B)² / A²)
        using Re-divide-real[of A² 4 * (B*cnj B)²] ‹Re A < 0› ‹is-real A›
        by (auto simp add: power2-eq-square)
      moreover
      have 2*Re (D - A) / Re A * 2 * Re(B*cnj B) = Re (2 * (D - A) / A * 2 * B * cnj B)
        using ‹is-real A› ‹is-real D› ‹A ≠ 0›
        using Re-divide-real[of A (4 * D - 4 * A) * B * cnj B]
        by (simp add: field-simps)
      ultimately
      have Re ((D - A)² + 4 * B*cnj B) ≤ Re((D - A)² + 4 * (B*cnj B)² / A² - 2*(D - A) / A * 2 * B*cnj B)
        by (simp add: field-simps power2-eq-square)
      hence Re ((D - A)² + 4 * B*cnj B) ≤ Re(((D - A) - 2 * B*cnj B / A)²)
        using ‹A ≠ 0›
        by (subst power2-diff) (simp add: power2-eq-square field-simps)
      hence dsc ≤ sqrt (Re(((D - A) - 2 * B*cnj B / A)²))
        using ‹dsc = sqrt(Re ((D-A)² + 4*(B*cnj B)))›
        by simp
      moreover
      have Re(((D - A) - 2 * B*cnj B / A)²) = (Re((D - A) - 2 * B*cnj B / A))²
        using ‹is-real A› ‹is-real D› div-reals
        by (simp add: power2-eq-square)
      ultimately
      have dsc ≤ |Re (D - A - 2 * B * cnj B / A)|
        by simp
      moreover
      have Re (D - A - 2 * B * cnj B / A) ≥ 0
```

```
      proof −
        have Re (A^2 + B*cnj B) ≥ 0
          using ‹is-real A›
          by (simp add: power2-eq-square)
        also have Re (A^2 + 2*B*cnj B − A*D) ≥ Re (A^2 + B*cnj B)
          using ‹Re A * Re D ≤ Re (B*cnj B)›
          using ‹is-real A› ‹is-real D›
          by simp
        finally have Re (A^2 + 2*B*cnj B − A*D) ≥ 0
          by simp
        show ?thesis
          using divide-right-mono-neg[OF ‹Re (A^2 + 2*B*cnj B − A*D) ≥ 0›, of Re A] ‹Re A < 0› ‹is-real A› ‹A ≠
0›
            by (simp add: add-divide-distrib diff-divide-distrib) (subst Re-divide-real, auto simp add: power2-eq-square
field-simps)
      qed
      ultimately
      have dsc ≤ Re (D − A − 2 * B * cnj B / A)
        by simp
      hence − Re (2 * (B * cnj B) / A) ≥ Re ((A − D + cor dsc))
        by (simp add: field-simps)
      hence − (Re (2 * (B * cnj B)) / Re A) ≥ Re (A − D + cor dsc)
        using ‹is-real A› ‹A ≠ 0›
        by (subst (asm) Re-divide-real, auto)
      from divide-right-mono[OF this, of Re (2 * B * cnj B)]
      have − 1 / Re A ≥ Re (A − D + cor dsc) / Re (2 * B * cnj B)
        using ‹Re A < 0› ‹B ≠ 0› ‹A ≠ 0› ‹0 < Re (2 * (B * cnj B))›
        using ‹(Re (2 * (B * cnj B)) / Re A) / Re (2 * B * cnj B) = 1 / Re A›
        by simp
      from mult-right-mono-neg[OF this, of Re A]
      show ?thesis
        using ‹is-real A› ‹is-real D› ‹B ≠ 0› ‹Re A < 0› ‹A ≠ 0›
        apply (subst *)
        apply (subst Re-mult-real, simp)
        apply (subst Re-divide-real, simp, simp)
        apply (simp add: field-simps)
        done
    qed
    ultimately
    show ?thesis
      by simp
  next
    case False
    hence A = 0
      using ‹¬ Re A > 0› ‹is-real A›
      using complex-eq-if-Re-eq by auto
    thus ?thesis
      by simp
  qed
 qed
qed


lemma chordal-circle-det-positive:
  fixes x y :: real
  assumes x * y < 0
  shows x / (x − y) > 0
proof (cases x > 0)
  case True
  hence y < 0
    using ‹x * y < 0›
    by (smt (verit) mult-nonneg-nonneg)
  have x − y > 0
    using ‹x > 0› ‹y < 0›
    by auto
  thus ?thesis
```

```
    using ‹x > 0›
    by (metis zero-less-divide-iff)
next
  case False
  hence *: y > 0 ∧ x < 0
    using ‹x * y < 0›
    using mult-nonpos-nonpos[of x y]
    by (cases x=0) force+

  have x - y < 0
    using *
    by auto
  thus ?thesis
    using *
    by (metis zero-less-divide-iff)
qed

lemma cor-sqrt-squared: x ≥ 0 ⟹ (cor (sqrt x))^2 = cor x
  by (simp add: power2-eq-square)

lemma chordal-circle1:
  assumes is-real A and is-real D and Re (A * D) < 0 and r = sqrt(Re ((4*A)/(A-D)))
  shows mk-circline A 0 0 D = chordal-circle ∞_h r
using assms
proof (transfer, transfer)
  fix A D r
  assume *: is-real A is-real D Re (A * D) < 0 r = sqrt (Re ((4*A)/(A-D)))
  hence A ≠ 0 ∨ D ≠ 0
    by auto
  hence (A, 0, 0, D) ∈ hermitean-nonzero
    using eq-cnj-iff-real[of A] eq-cnj-iff-real[of D] *
    unfolding hermitean-def
    by (simp add: mat-adj-def mat-cnj-def)
  moreover
  have (- (cor r)^2, 0, 0, 4 - (cor r)^2) ∈ hermitean-nonzero
    by (simp add: hermitean-def mat-adj-def mat-cnj-def power2-eq-square)
  moreover
  have A ≠ D
    using ‹Re (A * D) < 0› ‹is-real A› ‹is-real D›
    by auto
  have Re ((4*A)/(A-D)) ≥ 0
  proof-
    have Re A / Re (A - D) ≥ 0
      using ‹Re (A * D) < 0› ‹is-real A› ‹is-real D›
      using chordal-circle-det-positive[of Re A Re D]
      by simp
    thus ?thesis
      using ‹is-real A› ‹is-real D› ‹A ≠ D›
      by (subst Re-divide-real, auto)
  qed
  moreover
  have - (cor (sqrt (Re (4 * A / (A - D)))))^2 = cor (Re (4 / (D - A))) * A
    using ‹is-real A› ‹is-real D› ‹A ≠ D› ‹Re ((4*A)/(A-D)) ≥ 0›
    by (simp add: cor-sqrt-squared field-simps)
  moreover
  have 4 - 4 * A / (A - D) = 4 * D / (D - A)
    using‹A ≠ D›
    by (simp add: divide-simps split: if-split-asm) (simp add: minus-mult-right)
  hence **: 4 - (cor (sqrt (Re (4 * A / (A - D)))))^2 = cor (Re (4 / (D - A))) * D
    using ‹Re ((4*A)/(A-D)) ≥ 0› ‹is-real A› ‹is-real D› ‹A ≠ D›
    by (simp add: cor-sqrt-squared field-simps)
  ultimately
  show circline-eq-cmat (mk-circline-cmat A 0 0 D) (chordal-circle-cvec-cmat ∞_v r)
    using * ‹is-real A› ‹is-real D› ‹A ≠ D› ‹r = sqrt(Re ((4*A)/(A-D)))›
    by (simp, rule-tac x=Re(4/(D-A)) in exI, auto, simp-all add: **)
qed
```

lemma *chordal-circle2*:
assumes *is-real A* **and** *is-real D* **and** *Re* $(A * D) < 0$ **and** $r = sqrt(Re\ ((4*D)/(D-A)))$
shows *mk-circline A 0 0 D* = *chordal-circle* 0_h *r*
using *assms*
proof (*transfer*, *transfer*)
fix *A D r*
assume ∗: *is-real A is-real D Re* $(A * D) < 0$ *r* = *sqrt* (*Re* $((4*D)/(D-A))$)
hence $A \neq 0 \vee D \neq 0$
by *auto*
hence $(A, 0, 0, D) \in \{H.\ hermitean\ H \wedge H \neq mat\text{-}zero\}$
using *eq-cnj-iff-real*[*of A*] *eq-cnj-iff-real*[*of D*] ∗
unfolding *hermitean-def*
by (*simp add*: *mat-adj-def mat-cnj-def*)
moreover
have $(4 - (cor\ r)^2, 0, 0, -(cor\ r)^2) \in \{H.\ hermitean\ H \wedge H \neq mat\text{-}zero\}$
by (*auto simp add*: *hermitean-def mat-adj-def mat-cnj-def power2-eq-square*)
moreover
have $A \neq D$
using ‹*Re* $(A * D) < 0$› ‹*is-real A*› ‹*is-real D*›
by *auto*
have $Re((4*D)/(D-A)) \geq 0$
proof−
have *Re D* / *Re* $(D - A) \geq 0$
using ‹*Re* $(A * D) < 0$› ‹*is-real A*› ‹*is-real D*›
using *chordal-circle-det-positive*[*of Re D Re A*]
by (*simp add*: *field-simps*)
thus *?thesis*
using ‹*is-real A*› ‹*is-real D*› ‹$A \neq D$› *Re-divide-real* **by** *force*
qed
have $4 - 4 * D / (D - A) = 4 * A / (A - D)$
by (*simp add*: *divide-simps split*: *if-split-asm*) (*simp add*: ‹$A \neq D$› *minus-mult-right*)
hence ∗∗: $4 - (cor\ (sqrt\ (Re\ ((4*D)/(D-A)))))^2 = cor\ (Re\ (4 / (A - D))) * A$
using ‹*is-real A*› ‹*is-real D*› ‹$A \neq D$› ‹*Re* $(4 * D / (D - A)) \geq 0$›
by (*simp add*: *cor-sqrt-squared field-simps*)
moreover
have $-(cor\ (sqrt\ (Re\ ((4*D)/(D-A)))))^2 = cor\ (Re\ (4 / (A - D))) * D$
using ‹*is-real A*› ‹*is-real D*› ‹$A \neq D$› ‹*Re* $(4 * D / (D - A)) \geq 0$›
by (*simp add*: *cor-sqrt-squared field-simps*)
ultimately
show *circline-eq-cmat* (*mk-circline-cmat A 0 0 D*) (*chordal-circle-cvec-cmat* 0_v *r*)
using ‹*is-real A*› ‹*is-real D*› ‹$A \neq 0 \vee D \neq 0$› ‹$r = sqrt\ (Re\ ((4*D)/(D-A)))$›
using ∗
by (*simp*, *rule-tac x*=*Re* $(4/(A-D))$ **in** *exI*, *auto*, *simp-all add*: ∗∗)
qed

lemma *chordal-circle′*:
assumes $B \neq 0$ **and** $(A, B, C, D) \in hermitean\text{-}nonzero$ **and** *Re* (*mat-det* $(A, B, C, D)) \leq 0$ **and**
$C * a^2 + (D - A) * a - B = 0$ **and** $r = sqrt((4 - Re((-4 * a/B) * A)) / (1 + Re\ (a*cnj\ a)))$
shows *mk-circline A B C D* = *chordal-circle* (*of-complex a*) *r*
using *assms*
proof (*transfer*, *transfer*)
fix *A B C D a* :: *complex* **and** *r* :: *real*

let *?k* $= (-4) * a / B$

assume ∗: $(A, B, C, D) \in \{H.\ hermitean\ H \wedge H \neq mat\text{-}zero\}$ **and** ∗∗: $B \neq 0\ C * a^2 + (D - A) * a - B = 0$ **and**
rr: $r = sqrt\ ((4 - Re\ (?k * A)) / (1 + Re\ (a * cnj\ a)))$ **and** *det*: *Re* (*mat-det* $(A, B, C, D)) \leq 0$

have *is-real A is-real D C* = *cnj B*
using ∗ *hermitean-elems*
by *auto*

from ∗∗ **have** *a12*: *let dsc* = $sqrt(Re\ ((D-A)^2 + 4 * (B*cnj\ B)))$
in a = $(A - D + cor\ dsc) / (2 * C) \vee a = (A - D - cor\ dsc) / (2 * C)$
proof−

```
    have Re ((D−A)^2 + 4 * (B*cnj B)) ≥ 0
      using ‹is-real A› ‹is-real D›
      by (subst complex-mult-cnj-cmod) (simp add: power2-eq-square)
    hence ccsqrt ((D − A)^2 − 4 * C * − B) = cor (sqrt (Re ((D − A)^2 + 4 * (B * cnj B))))
      using csqrt-real[of ((D − A)^2 + 4 * (B * cnj B))] ‹is-real A› ‹is-real D› ‹C = cnj B›
      by (auto simp add: power2-eq-square field-simps)
    thus ?thesis
      using complex-quadratic-equation-two-roots[of C a D − A −B]
      using ‹C * a^2 + (D − A) * a − B = 0› ‹B ≠ 0› ‹C = cnj B›
      by (simp add: Let-def)
  qed

  have is-real ?k
    using a12 ‹C = cnj B› ‹is-real A› ‹is-real D›
    by (auto simp add: Let-def)
  have a ≠ 0
    using **
    by auto
  hence Re ?k ≠ 0
    using ‹is-real (−4*a / B)› ‹B ≠ 0›
   by (metis complex.expand divide-eq-0-iff divisors-zero zero-complex.simps(1) zero-complex.simps(2) zero-neq-neg-numeral)
  moreover
  have (−4) * a = cor (Re ?k) * B
    using complex-of-real-Re[OF ‹is-real (−4*a/B)›] ‹B ≠ 0›
    by simp
  moreover
  have is-real (a/B)
    using ‹is-real ?k› is-real-mult-real[of −4 a / B]
    by simp
  hence is-real (B * cnj a)
    using * ‹C = cnj B›
   by (metis (no-types, lifting) Im-complex-div-eq-0 complex-cnj-divide eq-cnj-iff-real hermitean-elems(3) mem-Collect-eq
mult.commute)
  hence B * cnj a = cnj B * a
    using eq-cnj-iff-real[of B * cnj a]
    by simp
  hence −4 * cnj a = cor (Re ?k) * C
    using ‹C = cnj B›
    using complex-of-real-Re[OF ‹is-real ?k›] ‹B ≠ 0›
    by (simp, simp add: field-simps)
  moreover
  have 1 + a * cnj a ≠ 0
    by (simp add: complex-mult-cnj-cmod)
  have r^2 = (4 − Re (?k * A)) / (1 + Re (a * cnj a))
  proof−
    have Re (a / B * A) ≥ −1
      using a12 chordal-circle-radius-positive[of A B C D] * ‹B ≠ 0› det
      by (auto simp add: Let-def field-simps)
    from mult-right-mono-neg[OF this, of −4]
    have 4 − Re (?k * A) ≥ 0
      using Re-mult-real[of −4 a / B * A]
      by (simp add: field-simps)
    moreover
    have 1 + Re (a * cnj a) > 0
      using ‹a ≠ 0› complex-mult-cnj complex-neq-0
      by auto
    ultimately
    have (4 − Re (?k * A)) / (1 + Re (a * cnj a)) ≥ 0
      by (metis divide-nonneg-pos)
    thus ?thesis
      using rr
      by simp
  qed
  hence r^2 = Re ((4 − ?k * A) / (1 + a * cnj a))
    using ‹is-real ?k› ‹is-real A› ‹1 + a * cnj a ≠ 0›
    by (subst Re-divide-real, auto)
```

hence $(cor\ r)^2 = (4 - ?k * A) / (1 + a * cnj\ a)$
using ‹*is-real ?k*› ‹*is-real A*› *mult-reals*[*of ?k A*]
by (*simp add*: *cor-squared*)
hence $4 - (cor\ r)^2 * (a * cnj\ a + 1) = cor\ (Re\ ?k) * A$
using *complex-of-real-Re*[*OF* ‹*is-real* $(-4*a/B)$›]
using ‹$1 + a * cnj\ a \neq 0$›
by (*simp add*: *field-simps*)
moreover

have $?k = cnj\ ?k$
using ‹*is-real ?k*›
using *eq-cnj-iff-real*[*of* $-4*a/B$]
by *simp*

have $?k^2 = cor\ ((cmod\ ?k)^2)$
using *cor-cmod-real*[*OF* ‹*is-real ?k*›]
unfolding *power2-eq-square* **by** *force*
hence $?k^2 = ?k * cnj\ ?k$
using *complex-mult-cnj-cmod*[*of ?k*]
by *simp*
hence ***: $a * cnj\ a = (cor\ ((Re\ ?k)^2) * B * C) / 16$
using *complex-of-real-Re*[*OF* ‹*is-real* $(-4*a/B)$›] ‹$C = cnj\ B$› ‹*is-real* $(-4*a/B)$› ‹$B \neq 0$›
by *simp*
from ** **have** $cor\ ((Re\ ?k)^2) * B * C - 4 * cor\ (Re\ ?k) * (D-A) - 16 = 0$
using *complex-of-real-Re*[*OF* ‹*is-real ?k*›]
by (*simp add*: *power2-eq-square*, *simp add*: *field-simps*, *algebra*)
hence $?k * (D-A) = 4 * (cor\ ((Re\ ?k)^2) * B * C / 16 - 1)$
by (*subst* (*asm*) *complex-of-real-Re*[*OF* ‹*is-real ?k*›]) *algebra*
hence $?k * (D-A) = 4 * (a*cnj\ a - 1)$
by (*subst* (*asm*) ***[*symmetric*]) *simp*

hence $4 * a * cnj\ a - (cor\ r)^2 * (a * cnj\ a + 1) = cor\ (Re\ ?k) * D$
using ‹$4 - (cor\ r)^2 * (a * cnj\ a + 1) = cor\ (Re\ ?k) * A$›
using *complex-of-real-Re*[*OF* ‹*is-real* $(-4*a/B)$›]
by *simp algebra*
ultimately
show *circline-eq-cmat* (*mk-circline-cmat A B C D*) (*chordal-circle-cvec-cmat* (*of-complex-cvec a*) *r*)
using * ‹$a \neq 0$›
by (*simp*, *rule-tac x*=$Re\ (-4*a / B)$ **in** *exI*, *simp*)
qed

end

References

[1] C. Dehlinger, J.-F. Dufourd, and P. Schreck. Higher-order intuitionistic formalization and proofs in Hilberts elementary geometry. In *Automated Deduction in Geometry*, pages 306–323. Springer, 2001.

[2] J. Duprat. Une axiomatique de la géométrie plane en Coq. *Actes des JFLA*, pages 123–136, 2008.

[3] F. Guilhot. Formalisation en Coq et visualisation d'un cours de géométrie pour le lycée. *Technique et Science informatiques*, 24(9):1113–1138, 2005.

[4] J. Harrison. A HOL theory of Euclidean space. In *Theorem proving in higher order logics*, pages 114–129. Springer, 2005.

[5] J. Harrison. Without loss of generality. In *Theorem Proving in Higher Order Logics*, pages 43–59. Springer, 2009.

[6] D. Hilbert. *Grundlagen der geometrie*. Springer-Verlag, 2013.

[7] G. Kahn. Constructive geometry according to Jan von Plato. *Coq contribution. Coq*, 5:10, 1995.

[8] L. I. Meikle and J. D. Fleuriot. Formalizing Hilberts Grundlagen in Isabelle/Isar. In *Theorem proving in higher order logics*, pages 319–334. Springer, 2003.

[9] J. Narboux. Mechanical theorem proving in Tarskis geometry. In *Automated Deduction in Geometry*, pages 139–156. Springer, 2007.

[10] T. Needham. *Visual complex analysis*. Oxford University Press, 1998.

[11] D. Petrovic and F. Maric. Formalizing analytic geometries. In *This volume contains the papers presented at ADG 2012: The 9th International Workshop on Automated Deduction in Geometry, held on September 17–19, 2012 at the University of Edinburgh. The submissions were each reviewed by at least 3 program committee mem-bers, and the committee decided to accept 15 papers for the workshop. The*, page 107, 2012.

[12] W. Schwabhäuser, W. Szmielew, and A. Tarski. *Metamathematische Methoden in der Geometrie*. Springer-Verlag, Berlin, 1983.

[13] H. Schwerdtfeger. *Geometry of complex numbers: circle geometry, Moebius transformation, non-euclidean geometry*. Courier Corporation, 1979.

[14] P. Scott. Mechanising Hilberts foundations of geometry in Isabelle. *Master's thesis, University of Edinburgh*, 2008.

[15] J. von Plato. The axioms of constructive geometry. *Annals of pure and applied logic*, 76(2):169–200, 1995.