Complex Geometry

Filip Marić Danijela Simić

December 14, 2021

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 $cos\ x < 0$
  ⟨*proof*⟩

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

lemma *sin-kpi* [*simp*]:
  fixes *k*::*int*
  shows $sin\ (k * pi) = 0$
  ⟨*proof*⟩

lemma *cos-odd-kpi* [*simp*]:
  fixes *k*::*int*
  assumes *odd k*
  shows $cos\ (k * pi) = -1$
  ⟨*proof*⟩

lemma *cos-even-kpi* [*simp*]:
  fixes *k*::*int*
  assumes *even k*
  shows $cos\ (k * pi) = 1$
  ⟨*proof*⟩

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

assumes *odd k*
shows *sin (pi / 2 + k * pi) = −1*
⟨*proof*⟩

lemma *sin-pi2-plus-even-kpi* [*simp*]:
fixes *k*::*int*
assumes *even k*
shows *sin (pi / 2 + k * pi) = 1*
⟨*proof*⟩

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*
⟨*proof*⟩

lemma *sin-0-iff-canon*:
assumes *sin φ = 0* **and** *−pi < φ* **and** *φ ≤ pi*
shows *φ = 0 ∨ φ = pi*
⟨*proof*⟩

lemma *cos0-sin1*:
assumes *sin φ = 1*
shows ∃ *k*::*int*. *φ = pi/2 + 2∗k∗pi*
⟨*proof*⟩

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′*
⟨*proof*⟩

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*
⟨*proof*⟩

lemma *sin-periodic-int* [*simp*]:
fixes *i* :: *int*
shows *sin (x + i * (2 * pi)) = sin x*
⟨*proof*⟩

lemma *cos-periodic-nat* [*simp*]:
fixes *n* :: *nat*
shows *cos (x + n * (2 * pi)) = cos x*
⟨*proof*⟩

lemma *cos-periodic-int* [*simp*]:
fixes *i* :: *int*
shows *cos (x + i * (2 * pi)) = cos x*
⟨*proof*⟩

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*
⟨*proof*⟩

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

lemma *ex-cos-eq*:
assumes $-pi/2 \leq \alpha \wedge \alpha \leq pi/2$
assumes $a \geq 0$ **and** $a < 1$
shows $\exists\ \alpha'.\ -pi/2 \leq \alpha' \wedge \alpha' \leq pi/2 \wedge \alpha' \neq \alpha \wedge cos\ (\alpha - \alpha') = a$
⟨*proof*⟩

lemma *ex-cos-gt*:
assumes $-pi/2 \leq \alpha \wedge \alpha \leq pi/2$
assumes $a < 1$
shows $\exists\ \alpha'.\ -pi/2 \leq \alpha' \wedge \alpha' \leq pi/2 \wedge \alpha' \neq \alpha \wedge cos\ (\alpha - \alpha') > a$
⟨*proof*⟩

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 $-pi \leq atan2\ y\ x \wedge atan2\ y\ x < pi$
⟨*proof*⟩

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*⟩

Canonical angle of any angle is unique.

lemma *canon-ang-unique*:
assumes *canon-ang-P* $\alpha\ \alpha_1$ **and** *canon-ang-P* $\alpha\ \alpha_2$
shows $\alpha_1 = \alpha_2$
⟨*proof*⟩

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** $\exists\ k{::}int.\ \alpha - \lfloor\alpha\rfloor = 2{*}k{*}pi$
⟨*proof*⟩

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

lemma *canon-ang-id*:
assumes $-pi < \alpha \wedge \alpha \leq pi$
shows $\lfloor\alpha\rfloor = \alpha$
⟨*proof*⟩

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

lemma *canon-ang-eq*:
assumes $\exists\ k{::}int.\ \alpha_1 - \alpha_2 = 2{*}k{*}pi$
shows $\lfloor\alpha_1\rfloor = \lfloor\alpha_2\rfloor$
⟨*proof*⟩

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 $\lfloor\alpha\rfloor = \alpha'$
⟨*proof*⟩

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*⟩

Canonical angle of opposite angle

lemma *canon-ang-uminus*:
assumes $\lfloor\alpha\rfloor \neq pi$
shows $\lfloor-\alpha\rfloor = -\lfloor\alpha\rfloor$
⟨*proof*⟩

lemma *canon-ang-uminus-pi*:
assumes $\lfloor\alpha\rfloor = pi$
shows $\lfloor-\alpha\rfloor = \lfloor\alpha\rfloor$
⟨*proof*⟩

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*⟩

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*⟩

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*⟩

lemma *canon-ang-minus-pi1*:
assumes $0 < \alpha$ **and** $\alpha \leq 2{*}pi$
shows $\lfloor\alpha - pi\rfloor = \alpha - pi$
⟨*proof*⟩

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*⟩

lemma *canon-ang-minus-pi2*:
assumes $-2{*}pi < \alpha$ **and** $\alpha \leq 0$
shows $\lfloor\alpha - pi\rfloor = \alpha + pi$
⟨*proof*⟩

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*⟩

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*⟩

Canonical angles for some special angles

lemma *zero-canonical* [*simp*]:
shows $\lfloor 0\rfloor = 0$
⟨*proof*⟩

lemma *pi-canonical* [*simp*]:
shows $\lfloor pi\rfloor = pi$
⟨*proof*⟩

lemma *two-pi-canonical* [*simp*]:
shows $\lfloor 2 * pi\rfloor = 0$
⟨*proof*⟩

Canonization preserves sine and cosine

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

lemma *canon-ang-cos* [*simp*]:
shows *cos* $\lfloor\alpha\rfloor$ = *cos* α
⟨*proof*⟩

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*
⟨*proof*⟩

lemma *cis-cmod* [*simp*]:
assumes $a \neq 0$
shows *cis* (*Arg a*) ∗ *of-real* (*cmod a*) = *a*
⟨*proof*⟩

lemma *cor-squared*:
shows $(cor\ x)^2 = cor\ (x^2)$
⟨*proof*⟩

lemma *cor-sqrt-mult-cor-sqrt* [*simp*]:

shows *cor (sqrt A) * cor (sqrt A) = cor |A|*
⟨*proof*⟩

lemma *cor-eq-0*: *cor x + i * cor y = 0 ⟷ x = 0 ∧ y = 0*
⟨*proof*⟩

lemma *one-plus-square-neq-zero* [*simp*]:
shows *1 + (cor x)^2 ≠ 0*
⟨*proof*⟩

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

lemma *complex-real-two* [*simp*]:
shows *Complex 2 0 = 2*
⟨*proof*⟩

lemma *complex-double* [*simp*]:
shows *(Complex a b) * 2 = Complex (2*a) (2*b)*
⟨*proof*⟩

lemma *complex-half* [*simp*]:
shows *(Complex a b) / 2 = Complex (a/2) (b/2)*
⟨*proof*⟩

lemma *Complex-scale1*:
shows *Complex (a * b) (a * c) = cor a * Complex b c*
⟨*proof*⟩

lemma *Complex-scale2*:
shows *Complex (a * c) (b * c) = Complex a b * cor c*
⟨*proof*⟩

lemma *Complex-scale3*:
shows *Complex (a / b) (a / c) = cor a * Complex (1 / b) (1 / c)*
⟨*proof*⟩

lemma *Complex-scale4*:
shows *c ≠ 0 ⟹ Complex (a / c) (b / c) = Complex a b / cor c*
⟨*proof*⟩

lemma *Complex-Re-express-cnj*:
shows *Complex (Re z) 0 = (z + cnj z) / 2*
⟨*proof*⟩

lemma *Complex-Im-express-cnj*:
shows *Complex 0 (Im z) = (z − cnj z)/2*
⟨*proof*⟩

Additional properties of *cmod*.

lemma *complex-mult-cnj-cmod*:
shows *z * cnj z = cor ((cmod z)^2)*
⟨*proof*⟩

lemma *cmod-square*:
shows *(cmod z)^2 = Re (z * cnj z)*
⟨*proof*⟩

lemma *cor-cmod-power-4* [*simp*]:
shows *cor (cmod z) ^ 4 = (z * cnj z)^2*
⟨*proof*⟩

lemma *cnjE*:
assumes *x ≠ 0*
shows *cnj x = cor ((cmod x)^2) / x*
⟨*proof*⟩

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

shows *cmod* (*z* / *cor k*) = *cmod z* / |*k*|
⟨*proof*⟩

lemma *cmod-mult-minus-left-distrib* [*simp*]:
shows *cmod* (*z*∗*z1* − *z*∗*z2*) = *cmod z* ∗ *cmod*(*z1* − *z2*)
⟨*proof*⟩

lemma *cmod-eqI*:
assumes *z1* ∗ *cnj z1* = *z2* ∗ *cnj z2*
shows *cmod z1* = *cmod z2*
⟨*proof*⟩

lemma *cmod-eqE*:
assumes *cmod z1* = *cmod z2*
shows *z1* ∗ *cnj z1* = *z2* ∗ *cnj z2*
⟨*proof*⟩

lemma *cmod-eq-one* [*simp*]:
shows *cmod a* = *1* ⟷ *a*∗*cnj a* = *1*
⟨*proof*⟩

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*
⟨*proof*⟩

lemma *complex-eq-if-Re-eq*:
assumes *is-real z1* **and** *is-real z2*
shows *z1* = *z2* ⟷ *Re z1* = *Re z2*
⟨*proof*⟩

lemma *mult-reals* [*simp*]:
assumes *is-real a* **and** *is-real b*
shows *is-real* (*a* ∗ *b*)
⟨*proof*⟩

lemma *div-reals* [*simp*]:
assumes *is-real a* **and** *is-real b*
shows *is-real* (*a* / *b*)
⟨*proof*⟩

lemma *complex-of-real-Re* [*simp*]:
assumes *is-real k*
shows *cor* (*Re k*) = *k*
⟨*proof*⟩

lemma *cor-cmod-real*:
assumes *is-real a*
shows *cor* (*cmod a*) = *a* ∨ *cor* (*cmod a*) = −*a*
⟨*proof*⟩

lemma *eq-cnj-iff-real*:
shows *cnj z* = *z* ⟷ *is-real z*
⟨*proof*⟩

lemma *eq-minus-cnj-iff-imag*:
shows *cnj z* = −*z* ⟷ *is-imag z*
⟨*proof*⟩

lemma *Re-divide-real*:
assumes *is-real b* **and** $b \neq 0$
shows *Re* $(a \ / \ b) =$ (*Re a*) / (*Re b*)
⟨*proof*⟩

lemma *Re-mult-real*:
assumes *is-real a*
shows *Re* $(a * b) =$ (*Re a*) $*$ (*Re b*)
⟨*proof*⟩

lemma *Im-mult-real*:
assumes *is-real a*
shows *Im* $(a * b) =$ (*Re a*) $*$ (*Im b*)
⟨*proof*⟩

lemma *Im-divide-real*:
assumes *is-real b* **and** $b \neq 0$
shows *Im* $(a \ / \ b) =$ (*Im a*) / (*Re b*)
⟨*proof*⟩

lemma *Re-sgn*:
assumes *is-real R*
shows *Re* (*sgn R*) = *sgn* (*Re R*)
⟨*proof*⟩

lemma *is-real-div*:
assumes $b \neq 0$
shows *is-real* $(a \ / \ b) \longleftrightarrow a*cnj\ b = b*cnj\ a$
⟨*proof*⟩

lemma *is-real-mult-real*:
assumes *is-real a* **and** $a \neq 0$
shows *is-real b* $\longleftrightarrow$ *is-real* $(a * b)$
⟨*proof*⟩

lemma *Im-express-cnj*:
shows *Im z* = $(z - cnj\ z)$ / $(2 * \mathrm{i})$
⟨*proof*⟩

lemma *Re-express-cnj*:
shows *Re z* = $(z + cnj\ z)$ / 2
⟨*proof*⟩

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* = $z * \mathrm{i}$
⟨*proof*⟩

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* = $\mathrm{i} *$ *cnj-mix* (*rot90 z1*) *z2*
⟨*proof*⟩

lemma *cnj-mix-minus′*:
shows *cnj z1*$*$*z2* − *z1*$*$*cnj z2* = *rot90* (*cnj-mix* (*rot90 z1*) *z2*)

⟨*proof*⟩

lemma *cnj-mix-real* [*simp*]:
shows *is-real* (*cnj-mix z1 z2*)
⟨*proof*⟩

lemma *scalprod-real* [*simp*]:
shows *is-real* (*scalprod z1 z2*)
⟨*proof*⟩

Additional properties of *cis* function.

lemma *cis-minus-pi2* [*simp*]:
shows *cis* (−*pi*/*2*) = −i
⟨*proof*⟩

lemma *cis-pi2-minus-x* [*simp*]:
shows *cis* (*pi*/*2* − *x*) = i ∗ *cis*(−*x*)
⟨*proof*⟩

lemma *cis-pm-pi* [*simp*]:
shows *cis* (*x* − *pi*) = − *cis x* **and** *cis* (*x* + *pi*) = − *cis x*
⟨*proof*⟩

lemma *cis-times-cis-opposite* [*simp*]:
shows *cis* φ ∗ *cis* (− φ) = *1*
⟨*proof*⟩

cis repeats only after $2k\pi$

lemma *cis-eq*:
assumes *cis a* = *cis b*
shows ∃ *k*::*int*. *a* − *b* = *2* ∗ *k* ∗ *pi*
⟨*proof*⟩

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′
⟨*proof*⟩

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
⟨*proof*⟩

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

lemma *cis-add-cis-opposite* [*simp*]:
shows *cis* φ + *cis* (−φ) = *2* ∗ *cos* φ
⟨*proof*⟩

cis equal to 1 or -1

lemma *cis-one* [*simp*]:
assumes *sin* φ = *0* **and** *cos* φ = *1*
shows *cis* φ = *1*
⟨*proof*⟩

lemma *cis-minus-one* [*simp*]:
assumes *sin* φ = *0* **and** *cos* φ = −*1*
shows *cis* φ = −*1*
⟨*proof*⟩

3.3.1 Additional properties of complex number argument

Arg of real numbers

lemma *is-real-arg1*:
  assumes *Arg z = 0 ∨ Arg z = pi*
  shows *is-real z*
  ⟨*proof*⟩

lemma *is-real-arg2*:
  assumes *is-real z*
  shows *Arg z = 0 ∨ Arg z = pi*
⟨*proof*⟩

lemma *arg-complex-of-real-positive* [*simp*]:
  assumes *k > 0*
  shows *Arg (cor k) = 0*
⟨*proof*⟩

lemma *arg-complex-of-real-negative* [*simp*]:
  assumes *k < 0*
  shows *Arg (cor k) = pi*
⟨*proof*⟩

lemma *arg-0-iff*:
  shows *z ≠ 0 ∧ Arg z = 0 ⟷ is-real z ∧ Re z > 0*
  ⟨*proof*⟩

lemma *arg-pi-iff*:
  shows *Arg z = pi ⟷ is-real z ∧ Re z < 0*
  ⟨*proof*⟩

Arg of imaginary numbers

lemma *is-imag-arg1*:
  assumes *Arg z = pi/2 ∨ Arg z = −pi/2*
  shows *is-imag z*
  ⟨*proof*⟩

lemma *is-imag-arg2*:
  assumes *is-imag z* **and** *z ≠ 0*
  shows *Arg z = pi/2 ∨ Arg z = −pi/2*
  ⟨*proof*⟩

lemma *arg-complex-of-real-times-i-positive* [*simp*]:
  assumes *k > 0*
  shows *Arg (cor k ∗ i) = pi / 2*
⟨*proof*⟩

lemma *arg-complex-of-real-times-i-negative* [*simp*]:
  assumes *k < 0*
  shows *Arg (cor k ∗ i) = − pi / 2*
⟨*proof*⟩

lemma *arg-pi2-iff*:
  shows *z ≠ 0 ∧ Arg z = pi / 2 ⟷ is-imag z ∧ Im z > 0*
  ⟨*proof*⟩

lemma *arg-minus-pi2-iff*:
  shows *z ≠ 0 ∧ Arg z = − pi / 2 ⟷ is-imag z ∧ Im z < 0*
  ⟨*proof*⟩

Argument is a canonical angle

lemma *canon-ang-arg*:
  shows *⌊Arg z⌋ = Arg z*
  ⟨*proof*⟩

lemma *arg-cis*:
shows *Arg* (*cis* φ) = $\lfloor\varphi\rfloor$
⟨*proof*⟩

Cosine and sine of *Arg*

lemma *cos-arg*:
assumes $z \neq 0$
shows *cos* (*Arg z*) = *Re z* / *cmod z*
⟨*proof*⟩

lemma *sin-arg*:
assumes $z \neq 0$
shows *sin* (*Arg z*) = *Im z* / *cmod z*
⟨*proof*⟩

Argument of product

lemma *cis-arg-mult*:
assumes $z1 * z2 \neq 0$
shows *cis* (*Arg* (*z1* ∗ *z2*)) = *cis* (*Arg z1* + *Arg z2*)
⟨*proof*⟩

lemma *arg-mult-2kpi*:
assumes $z1 * z2 \neq 0$
shows ∃ *k*::*int*. *Arg* (*z1* ∗ *z2*) = *Arg z1* + *Arg z2* + 2∗*k*∗*pi*
⟨*proof*⟩

lemma *arg-mult*:
assumes $z1 * z2 \neq 0$
shows *Arg*(*z1* ∗ *z2*) = $\lfloor$*Arg z1* + *Arg z2*$\rfloor$
⟨*proof*⟩

lemma *arg-mult-real-positive* [*simp*]:
assumes $k > 0$
shows *Arg* (*cor k* ∗ *z*) = *Arg z*
⟨*proof*⟩

lemma *arg-mult-real-negative* [*simp*]:
assumes $k < 0$
shows *Arg* (*cor k* ∗ *z*) = *Arg* (−*z*)
⟨*proof*⟩

lemma *arg-div-real-positive* [*simp*]:
assumes $k > 0$
shows *Arg* (*z* / *cor k*) = *Arg z*
⟨*proof*⟩

lemma *arg-div-real-negative* [*simp*]:
assumes $k < 0$
shows *Arg* (*z* / *cor k*) = *Arg* (−*z*)
⟨*proof*⟩

lemma *arg-mult-eq*:
assumes $z * z1 \neq 0$ **and** $z * z2 \neq 0$
assumes *Arg* (*z* ∗ *z1*) = *Arg* (*z* ∗ *z2*)
shows *Arg z1* = *Arg z2*
⟨*proof*⟩

Argument of conjugate

lemma *arg-cnj-pi*:
assumes *Arg z* = *pi*
shows *Arg* (*cnj z*) = *pi*
⟨*proof*⟩

lemma *arg-cnj-not-pi*:
assumes *Arg z* ≠ *pi*

shows *Arg (cnj z) = −Arg z*
⟨*proof*⟩

Argument of reciprocal

lemma *arg-inv-not-pi*:
assumes *z ≠ 0* **and** *Arg z ≠ pi*
shows *Arg (1 / z) = − Arg z*
⟨*proof*⟩

lemma *arg-inv-pi*:
assumes *z ≠ 0* **and** *Arg z = pi*
shows *Arg (1 / z) = pi*
⟨*proof*⟩

lemma *arg-inv-2kpi*:
assumes *z ≠ 0*
shows ∃ *k*::*int*. *Arg (1 / z) = − Arg z + 2∗k∗pi*
⟨*proof*⟩

lemma *arg-inv*:
assumes *z ≠ 0*
shows *Arg (1 / z) = ⌊− Arg z⌋*
⟨*proof*⟩

Argument of quotient

lemma *arg-div-2kpi*:
assumes *z1 ≠ 0* **and** *z2 ≠ 0*
shows ∃ *k*::*int*. *Arg (z1 / z2) = Arg z1 − Arg z2 + 2∗k∗pi*
⟨*proof*⟩

lemma *arg-div*:
assumes *z1 ≠ 0* **and** *z2 ≠ 0*
shows *Arg(z1 / z2) = ⌊Arg z1 − Arg z2⌋*
⟨*proof*⟩

Argument of opposite

lemma *arg-uminus*:
assumes *z ≠ 0*
shows *Arg (−z) = ⌊Arg z + pi⌋*
⟨*proof*⟩

lemma *arg-uminus-opposite-sign*:
assumes *z ≠ 0*
shows *Arg z > 0 ⟷ ¬ Arg (−z) > 0*
⟨*proof*⟩

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*⟩

3.3.2 Complex square root

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

lemma *square-ccsqrt* [*simp*]:
shows $(ccsqrt\ x)^2 = x$
⟨*proof*⟩

lemma *ex-complex-sqrt*:
shows ∃ *s*::*complex*. *s∗s = z*
⟨*proof*⟩

lemma *ccsqrt*:
assumes $s * s = z$
shows $s = ccsqrt\ z \vee s = -ccsqrt\ z$
⟨*proof*⟩

lemma *null-ccsqrt* [*simp*]:
shows $ccsqrt\ x = 0 \longleftrightarrow x = 0$
⟨*proof*⟩

lemma *ccsqrt-mult*:
shows $ccsqrt\ (a * b) = ccsqrt\ a * ccsqrt\ b \vee$
$ccsqrt\ (a * b) = -\ ccsqrt\ a * ccsqrt\ b$
⟨*proof*⟩

lemma *csqrt-real*:
assumes *is-real x*
shows $(Re\ x \geq 0 \wedge ccsqrt\ x = cor\ (sqrt\ (Re\ x))) \vee$
$(Re\ x < 0 \wedge ccsqrt\ x = \mathrm{i} * cor\ (sqrt\ (-\ (Re\ x))))$
⟨*proof*⟩

Rotation of complex vector to x-axis.

lemma *is-real-rot-to-x-axis*:
assumes $z \neq 0$
shows $is\text{-}real\ (cis\ (-Arg\ z) * z)$
⟨*proof*⟩

lemma *positive-rot-to-x-axis*:
assumes $z \neq 0$
shows $Re\ (cis\ (-Arg\ z) * z) > 0$
⟨*proof*⟩

Inequalities involving *cmod*.

lemma *cmod-1-plus-mult-le*:
shows $cmod\ (1 + z*w) \leq sqrt((1 + (cmod\ z)^2) * (1 + (cmod\ w)^2))$
⟨*proof*⟩

lemma *cmod-diff-ge*:
shows $cmod\ (b - c) \geq sqrt\ (1 + (cmod\ b)^2) - sqrt\ (1 + (cmod\ c)^2)$
⟨*proof*⟩

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

Definition of Euclidean distance between two complex numbers.

definition *cdist* **where**
[*simp*]: $cdist\ z1\ z2 \equiv cmod\ (z2 - z1)$

Misc. properties of complex numbers.

lemma *ex-complex-to-complex* [*simp*]:
fixes *z1 z2* :: *complex*
assumes $z1 \neq 0$ **and** $z2 \neq 0$
shows $\exists k.\ k \neq 0 \wedge z2 = k * z1$
⟨*proof*⟩

lemma *ex-complex-to-one* [*simp*]:
fixes *z*::*complex*
assumes $z \neq 0$
shows $\exists k.\ k \neq 0 \wedge k * z = 1$
⟨*proof*⟩

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

lemma *complex-sqrt-1*:
fixes *z*::*complex*
assumes $z \neq 0$
shows $z = 1 \ / \ z \longleftrightarrow z = 1 \vee z = -1$
⟨*proof*⟩

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* ($\angle$) **where**
[*simp*]: $\angle\ z1\ z2 \equiv \lfloor Arg\ z2 - Arg\ z1 \rfloor$

lemma *ang-vec-bounded*:
shows $-pi < \angle\ z1\ z2 \wedge \angle\ z1\ z2 \leq pi$
⟨*proof*⟩

lemma *ang-vec-sym*:
assumes $\angle\ z1\ z2 \neq pi$
shows $\angle\ z1\ z2 = -\angle\ z2\ z1$
⟨*proof*⟩

lemma *ang-vec-sym-pi*:
assumes $\angle\ z1\ z2 = pi$
shows $\angle\ z1\ z2 = \angle\ z2\ z1$
⟨*proof*⟩

lemma *ang-vec-plus-pi1*:
assumes $\angle\ z1\ z2 > 0$
shows $\lfloor \angle\ z1\ z2 + pi \rfloor = \angle\ z1\ z2 - pi$
⟨*proof*⟩

lemma *ang-vec-plus-pi2*:
assumes $\angle\ z1\ z2 \leq 0$
shows $\lfloor \angle\ z1\ z2 + pi \rfloor = \angle\ z1\ z2 + pi$
⟨*proof*⟩

lemma *ang-vec-opposite1*:
assumes $z1 \neq 0$
shows $\angle\ (-z1)\ z2 = \lfloor \angle\ z1\ z2 - pi \rfloor$
⟨*proof*⟩

lemma *ang-vec-opposite2*:
assumes $z2 \neq 0$
shows $\angle\ z1\ (-z2) = \lfloor \angle\ z1\ z2 + pi \rfloor$
⟨*proof*⟩

lemma *ang-vec-opposite-opposite*:
assumes $z1 \neq 0$ **and** $z2 \neq 0$
shows $\angle\ (-z1)\ (-z2) = \angle\ z1\ z2$
⟨*proof*⟩

lemma *ang-vec-opposite-opposite'*:
assumes $z1 \neq z$ **and** $z2 \neq z$
shows $\angle\ (z - z1)\ (z - z2) = \angle\ (z1 - z)\ (z2 - z)$

⟨*proof*⟩

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*⟩

lemma *cos0-scalprod0*:
assumes *z1 ≠ 0* **and** *z2 ≠ 0*
shows *cos (∠ z1 z2) = 0 ⟷ scalprod z1 z2 = 0*
⟨*proof*⟩

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

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*⟩

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*
⟨*proof*⟩

lemma *ang-vec-c-bounded*: *0 ≤ ∠c z1 z2 ∧ ∠c z1 z2 ≤ pi*
⟨*proof*⟩

Cosine and scalar product

lemma *cos-c-*: *cos (∠c z1 z2) = cos (∠ z1 z2)*
⟨*proof*⟩

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

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
⟨*proof*⟩

lemma *ang-vec-a-opposite2*:
∠a z1 z2 = ∠a z1 (−z2)
⟨*proof*⟩

lemma *ang-vec-a-opposite1*:
shows *∠a z1 z2 = ∠a (−z1) z2*
⟨*proof*⟩

lemma *ang-vec-a-scale1*:
assumes $k \neq 0$
shows $\angle a\ (cor\ k * z1)\ z2 = \angle a\ z1\ z2$
⟨*proof*⟩

lemma *ang-vec-a-scale2*:
assumes $k \neq 0$
shows $\angle a\ z1\ (cor\ k * z2) = \angle a\ z1\ z2$
⟨*proof*⟩

lemma *ang-vec-a-scale*:
assumes $k1 \neq 0$ **and** $k2 \neq 0$
shows $\angle a\ (cor\ k1 * z1)\ (cor\ k2 * z2) = \angle a\ z1\ z2$
⟨*proof*⟩

lemma *ang-a-cnj-cnj*:
shows $\angle a\ z1\ z2 = \angle a\ (cnj\ z1)\ (cnj\ z2)$
⟨*proof*⟩

Cosine and scalar product

lemma *ortho-a-scalprod0*:
assumes $z1 \neq 0$ **and** $z2 \neq 0$
shows $\angle a\ z1\ z2 = pi/2 \longleftrightarrow scalprod\ z1\ z2 = 0$
⟨*proof*⟩

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

lemma *cos-a-c*: $cos\ (\angle a\ z1\ z2) = abs\ (cos\ (\angle c\ z1\ z2))$
⟨*proof*⟩

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 $\exists\ x\ y.\ x \neq y \wedge x \in A \wedge y \in A$
⟨*proof*⟩

Every infinite set has at least three different elements

lemma *infinite-contains-3-elems*:
assumes *infinite A*
shows $\exists\ x\ y\ z.\ x \neq y \wedge x \neq z \wedge y \neq z \wedge x \in A \wedge y \in A \wedge z \in A$
⟨*proof*⟩

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*⟩

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*⟩

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*: $card\ A = 2 \longleftrightarrow (\exists\ x\ y.\ x \neq y \wedge A = \{x, y\})$

⟨*proof*⟩

lemma *card-eq-2-doubleton*:
assumes *card A = 2* **and** *x ≠ y* **and** *x ∈ A* **and** *y ∈ A*
shows *A = {x, y}*
⟨*proof*⟩

Bijections map singleton to singleton sets

lemma *bij-image-singleton*:
shows *⟦f ‘ A = {b}; f a = b; bij f⟧ ⟹ A = {a}*
⟨*proof*⟩

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*⟩

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*⟩

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*⟩

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 $\exists\ k.\ x1 = k * x2 \wedge y1 = k * y2 \wedge z1 = k * z2$
⟨*proof*⟩

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 $b\ c\ \xi1\ \xi2 :: real$
assumes $b^2 - 4*c \geq 0$ **and** $\xi1^2 + b*\xi1 + c = 0$ **and** $\xi2^2 + b*\xi2 + c = 0$ **and** $\xi1 \neq \xi2$
shows $\xi1*\xi2 = c$
⟨*proof*⟩

lemma *viette2*:
fixes $a\ b\ c\ \xi1\ \xi2 :: real$
assumes $a \neq 0$ **and** $b^2 - 4*a*c \geq 0$ **and** $a*\xi1^2 + b*\xi1 + c = 0$ **and** $a*\xi2^2 + b*\xi2 + c = 0$ **and** $\xi1 \neq \xi2$
shows $\xi1*\xi2 = c/a$
⟨*proof*⟩

lemma *viette2'-monic*:
fixes $b\ c\ \xi :: real$
assumes $b^2 - 4*c = 0$ **and** $\xi^2 + b*\xi + c = 0$
shows $\xi*\xi = c$
⟨*proof*⟩

lemma *viette2'*:
fixes $a\ b\ c\ \xi :: real$
assumes $a \neq 0$ **and** $b^2 - 4*a*c = 0$ **and** $a*\xi^2 + b*\xi + c = 0$
shows $\xi*\xi = c/a$
⟨*proof*⟩

3.7.2 Complex quadratic equations

lemma *complex-quadratic-equation-monic-only-two-roots*:
fixes $\xi :: complex$
assumes $\xi^2 + b * \xi + c = 0$
shows $\xi = (-b + ccsqrt(b^2 - 4*c))\ /\ 2 \vee \xi = (-b - ccsqrt(b^2 - 4*c))\ /\ 2$
⟨*proof*⟩

lemma *complex-quadratic-equation-monic-roots*:
fixes $\xi :: complex$
assumes $\xi = (-b + ccsqrt(b^2 - 4*c))\ /\ 2 \vee$
$\xi = (-b - ccsqrt(b^2 - 4*c))\ /\ 2$
shows $\xi^2 + b * \xi + c = 0$
⟨*proof*⟩

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*⟩

lemma *complex-quadratic-equation-two-roots*:
fixes $\xi :: complex$
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*⟩

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 $x = x1 \vee x = x2$
⟨*proof*⟩

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*⟩

lemma *quadratic-linear-at-most-2-intersections-help'*:
fixes $x\ y$ :: *complex*
assumes $qf = (\lambda\ x\ y.\ a11*x^2 + 2*a12*x*y + a22*y^2 + b1*x + b2*y + c)$
$x = -n/k1$ **and** $k1 \neq 0$ **and** $qf\ x\ y = 0$
$yf = (\lambda\ y.\ k1^2*a22*y^2 + (-2*a12*n*k1 + b2*k1^2)*y + a11*n^2 - b1*n*k1 + c*k1^2)$
shows $yf\ y = 0$
⟨*proof*⟩

lemma *quadratic-linear-at-most-2-intersections*:
fixes $x\ y\ x1\ y1\ x2\ y2$ :: *complex*
assumes $(a11,\ a12,\ a22) \neq (0,\ 0,\ 0)$ **and** $(k1,\ k2) \neq (0,\ 0)$
assumes $a11*k2^2 - 2*a12*k1*k2 + a22*k1^2 \neq 0$
assumes $qf = (\lambda\ x\ y.\ a11*x^2 + 2*a12*x*y + a22*y^2 + b1*x + b2*y + c)$ **and** $lf = (\lambda\ x\ y.\ k1*x + k2*y + n)$
$qf\ x1\ y1 = 0$ **and** $lf\ x1\ y1 = 0$
$qf\ x2\ y2 = 0$ **and** $lf\ x2\ y2 = 0$
$(x1,\ y1) \neq (x2,\ y2)$
$qf\ x\ y = 0$ **and** $lf\ x\ y = 0$
shows $(x,\ y) = (x1,\ y1) \vee (x,\ y) = (x2,\ y2)$
⟨*proof*⟩

lemma *quadratic-quadratic-at-most-2-intersections'*:
fixes $x\ y\ x1\ y1\ x2\ y2$ :: *complex*
assumes $b2 \neq B2 \vee b1 \neq B1$
$(b2 - B2)^2 + (b1 - B1)^2 \neq 0$
assumes $qf1 = (\lambda\ x\ y.\ x^2 + y^2 + b1*x + b2*y + c)$
$qf2 = (\lambda\ x\ y.\ x^2 + y^2 + B1*x + B2*y + C)$
$qf1\ x1\ y1 = 0\ qf2\ x1\ y1 = 0$
$qf1\ x2\ y2 = 0\ qf2\ x2\ y2 = 0$
$(x1,\ y1) \neq (x2,\ y2)$
$qf1\ x\ y = 0\ qf2\ x\ y = 0$
shows $(x,\ y) = (x1,\ y1) \vee (x,\ y) = (x2,\ y2)$
⟨*proof*⟩

lemma *quadratic-change-coefficients*:
fixes $x\ y$ :: *complex*
assumes $A1 \neq 0$
assumes $qf = (\lambda\ x\ y.\ A1*x^2 + A1*y^2 + b1*x + b2*y + c)$
$qf\ x\ y = 0$
$qf\text{-}1 = (\lambda\ x\ y.\ x^2 + y^2 + (b1/A1)*x + (b2/A1)*y + c/A1)$
shows $qf\text{-}1\ x\ y = 0$
⟨*proof*⟩

lemma *quadratic-quadratic-at-most-2-intersections*:
fixes *x y x1 y1 x2 y2* :: *complex*
assumes $A1 \neq 0$ **and** $A2 \neq 0$
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**
$b2{*}A2 \neq B2{*}A1 \vee b1{*}A2 \neq B1{*}A1$
shows $(x,\ y) = (x1,\ y1) \vee (x,\ y) = (x2,\ y2)$
⟨*proof*⟩

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* $(k *_{sv} v) = k * fst\ v$
⟨*proof*⟩

lemma *snd-mult-sv* [*simp*]:
shows *snd* $(k *_{sv} v) = k * snd\ v$
⟨*proof*⟩

lemma *mult-sv-mult-sv* [*simp*]:
shows $k1 *_{sv} (k2 *_{sv} v) = (k1{*}k2) *_{sv} v$
⟨*proof*⟩

lemma *one-mult-sv* [*simp*]:
shows $1 *_{sv} v = v$
⟨*proof*⟩

lemma *mult-sv-ex-id1* [*simp*]:
shows $\exists\ k{::}complex.\ k \neq 0 \wedge k *_{sv} v = v$
⟨*proof*⟩

lemma *mult-sv-ex-id2* [*simp*]:
shows $\exists\ k{::}complex.\ k \neq 0 \wedge v = k *_{sv} v$
⟨*proof*⟩

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 $v1 *_{vv} v2 = v2 *_{vv} v1$
⟨*proof*⟩

lemma *mult-vv-scale-sv1*:
shows $(k *_{sv} v1) *_{vv} v2 = k * (v1 *_{vv} v2)$
⟨*proof*⟩

lemma *mult-vv-scale-sv2*:
shows $v1 *_{vv} (k *_{sv} v2) = k * (v1 *_{vv} v2)$
⟨*proof*⟩

Conjugate vector

fun *vec-map* **where**
$vec\text{-}map\ f\ (x,\ y) = (f\ x,\ f\ y)$

definition *vec-cnj* **where**
$vec\text{-}cnj = vec\text{-}map\ cnj$

lemma *vec-cnj-vec-cnj* [*simp*]:
shows $vec\text{-}cnj\ (vec\text{-}cnj\ v) = v$
⟨*proof*⟩

lemma *cnj-mult-vv*:
shows $cnj\ (v1 *_{vv} v2) = (vec\text{-}cnj\ v1) *_{vv} (vec\text{-}cnj\ v2)$
⟨*proof*⟩

lemma *vec-cnj-sv* [*simp*]:
shows $vec\text{-}cnj\ (k *_{sv} A) = cnj\ k *_{sv} vec\text{-}cnj\ A$
⟨*proof*⟩

lemma *scalsquare-vv-zero*:
shows $(vec\text{-}cnj\ v) *_{vv} v = 0 \longleftrightarrow v = vec\text{-}zero$
⟨*proof*⟩

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 $k1 *_{sm} (k2 *_{sm} A) = (k1*k2) *_{sm} A$
⟨*proof*⟩

lemma *mult-sm-neutral* [*simp*]:
shows $1 *_{sm} A = A$
⟨*proof*⟩

lemma *mult-sm-inv-l*:
assumes $k \neq 0$ **and** $k *_{sm} A = B$
shows $A = (1/k) *_{sm} B$
⟨*proof*⟩

lemma *mult-sm-ex-id1* [*simp*]:
shows $\exists\ k{::}complex.\ k \neq 0 \wedge k *_{sm} M = M$
⟨*proof*⟩

lemma *mult-sm-ex-id2* [*simp*]:
shows $\exists\ k{::}complex.\ k \neq 0 \wedge M = k *_{sm} M$
⟨*proof*⟩

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*
⟨*proof*⟩

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*
⟨*proof*⟩

lemma *mult-assoc-5*:
shows *A* $*_{mm}$ (*B* $*_{mm}$ *C* $*_{mm}$ *D*) $*_{mm}$ *E* = (*A* $*_{mm}$ *B*) $*_{mm}$ *C* $*_{mm}$ (*D* $*_{mm}$ *E*)
⟨*proof*⟩

lemma *mat-zero-r* [*simp*]:
shows *A* $*_{mm}$ *mat-zero* = *mat-zero*
⟨*proof*⟩

lemma *mat-zero-l* [*simp*]:
shows *mat-zero* $*_{mm}$ *A* = *mat-zero*
⟨*proof*⟩

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

lemma *mat-eye-l*:
shows *eye* $*_{mm}$ *A* = *A*
⟨*proof*⟩

lemma *mat-eye-r*:
shows *A* $*_{mm}$ *eye* = *A*
⟨*proof*⟩

lemma *mult-mm-sm* [*simp*]:
shows *A* $*_{mm}$ (*k* $*_{sm}$ *B*) = *k* $*_{sm}$ (*A* $*_{mm}$ *B*)
⟨*proof*⟩

lemma *mult-sm-mm* [*simp*]:
shows (*k* $*_{sm}$ *A*) $*_{mm}$ *B* = *k* $*_{sm}$ (*A* $*_{mm}$ *B*)
⟨*proof*⟩

lemma *mult-sm-eye-mm* [*simp*]:
shows *k* $*_{sm}$ *eye* $*_{mm}$ *A* = *k* $*_{sm}$ *A*
⟨*proof*⟩

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*
⟨*proof*⟩

lemma *mat-det-mult-sm* [*simp*]:
shows *mat-det* $(k *_{sm} A) = (k*k) * \textit{mat-det } A$
⟨*proof*⟩

Matrix inverse

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

lemma *mat-inv-r*:
assumes *mat-det* $A \neq 0$
shows $A *_{mm} (\textit{mat-inv } A) = \textit{eye}$
⟨*proof*⟩

lemma *mat-inv-l*:
assumes *mat-det* $A \neq 0$
shows $(\textit{mat-inv } A) *_{mm} A = \textit{eye}$
⟨*proof*⟩

lemma *mat-det-inv*:
assumes *mat-det* $A \neq 0$
shows $\textit{mat-det } (\textit{mat-inv } A) = 1 \ / \ \textit{mat-det } A$
⟨*proof*⟩

lemma *mult-mm-inv-l*:
assumes *mat-det* $A \neq 0$ **and** $A *_{mm} B = C$
shows $B = \textit{mat-inv } A *_{mm} C$
⟨*proof*⟩

lemma *mult-mm-inv-r*:
assumes *mat-det* $B \neq 0$ **and** $A *_{mm} B = C$
shows $A = C *_{mm} \textit{mat-inv } B$
⟨*proof*⟩

lemma *mult-mm-non-zero-l*:
assumes *mat-det* $A \neq 0$ **and** $B \neq \textit{mat-zero}$
shows $A *_{mm} B \neq \textit{mat-zero}$
⟨*proof*⟩

lemma *mat-inv-mult-mm*:
assumes *mat-det* $A \neq 0$ **and** *mat-det* $B \neq 0$
shows $\textit{mat-inv } (A *_{mm} B) = \textit{mat-inv } B *_{mm} \textit{mat-inv } A$
⟨*proof*⟩

lemma *mult-mm-cancel-l*:
assumes *mat-det* $M \neq 0$ $M *_{mm} A = M *_{mm} B$
shows $A = B$
⟨*proof*⟩

lemma *mult-mm-cancel-r*:
assumes *mat-det* $M \neq 0$ $A *_{mm} M = B *_{mm} M$
shows $A = B$
⟨*proof*⟩

lemma *mult-mm-non-zero-r*:
assumes $A \neq \textit{mat-zero}$ **and** *mat-det* $B \neq 0$
shows $A *_{mm} B \neq \textit{mat-zero}$
⟨*proof*⟩

lemma *mat-inv-mult-sm*:
assumes $k \neq 0$
shows $\textit{mat-inv } (k *_{sm} A) = (1 \ / \ k) *_{sm} \textit{mat-inv } A$
⟨*proof*⟩

lemma *mat-inv-inv* [*simp*]:
assumes *mat-det* $M \neq 0$

shows *mat-inv* (*mat-inv M*) = *M*
⟨*proof*⟩

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*
⟨*proof*⟩

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

lemma *mat-t-mult-mm* [*simp*]:
shows *mat-transpose* (*A* $*_{mm}$ *B*) = *mat-transpose B* $*_{mm}$ *mat-transpose A*
⟨*proof*⟩

lemma *mat-inv-transpose*:
shows *mat-transpose* (*mat-inv M*) = *mat-inv* (*mat-transpose M*)
⟨*proof*⟩

lemma *mat-det-transpose* [*simp*]:
fixes *M* :: *complex-mat*
shows *mat-det* (*mat-transpose M*) = *mat-det M*
⟨*proof*⟩

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*
⟨*proof*⟩

lemma *mat-cnj-sm* [*simp*]:
shows *mat-cnj* (*k* $*_{sm}$ *A*) = *cnj k* $*_{sm}$ (*mat-cnj A*)
⟨*proof*⟩

lemma *mat-det-cnj* [*simp*]:
shows *mat-det* (*mat-cnj A*) = *cnj* (*mat-det A*)
⟨*proof*⟩

lemma *nonzero-mat-cnj*:
shows *mat-cnj A* = *mat-zero* ⟷ *A* = *mat-zero*
⟨*proof*⟩

lemma *mat-inv-cnj*:
shows *mat-cnj* (*mat-inv M*) = *mat-inv* (*mat-cnj M*)
⟨*proof*⟩

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*

⟨*proof*⟩

lemma *mat-adj-mult-sm* [*simp*]:
shows *mat-adj* (*k* $*_{sm}$ *A*) = *cnj k* $*_{sm}$ *mat-adj A*
⟨*proof*⟩

lemma *mat-det-adj*:
shows *mat-det* (*mat-adj A*) = *cnj* (*mat-det A*)
⟨*proof*⟩

lemma *mat-adj-inv*:
assumes *mat-det M* ≠ *0*
shows *mat-adj* (*mat-inv M*) = *mat-inv* (*mat-adj M*)
⟨*proof*⟩

lemma *mat-transpose-mat-cnj*:
shows *mat-transpose* (*mat-cnj A*) = *mat-adj A*
⟨*proof*⟩

lemma *mat-adj-adj* [*simp*]:
shows *mat-adj* (*mat-adj A*) = *A*
⟨*proof*⟩

lemma *mat-adj-eye* [*simp*]:
shows *mat-adj eye* = *eye*
⟨*proof*⟩

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*
⟨*proof*⟩

lemma *mult-mv-mv* [*simp*]:
shows *B* $*_{mv}$ (*A* $*_{mv}$ *v*) = (*B* $*_{mm}$ *A*) $*_{mv}$ *v*
⟨*proof*⟩

lemma *mult-vm-vm* [*simp*]:
shows (*v* $*_{vm}$ *A*) $*_{vm}$ *B* = *v* $*_{vm}$ (*A* $*_{mm}$ *B*)
⟨*proof*⟩

lemma *mult-mv-inv*:
assumes *x* = *A* $*_{mv}$ *y* **and** *mat-det A* ≠ *0*
shows *y* = (*mat-inv A*) $*_{mv}$ *x*
⟨*proof*⟩

lemma *mult-vm-inv*:
assumes *x* = *y* $*_{vm}$ *A* **and** *mat-det A* ≠ *0*
shows *y* = *x* $*_{vm}$ (*mat-inv A*)
⟨*proof*⟩

lemma *mult-mv-cancel-l*:
assumes *mat-det A* ≠ *0* **and** *A* $*_{mv}$ *v* = *A* $*_{mv}$ *v*′
shows *v* = *v*′
⟨*proof*⟩

lemma *mult-vm-cancel-r*:

assumes *mat-det A* $\neq$ *0* **and** *v* $*_{vm}$ *A* = *v'* $*_{vm}$ *A*
shows *v* = *v'*
⟨*proof*⟩

lemma *vec-zero-l* [*simp*]:
shows *A* $*_{mv}$ *vec-zero* = *vec-zero*
⟨*proof*⟩

lemma *vec-zero-r* [*simp*]:
shows *vec-zero* $*_{vm}$ *A* = *vec-zero*
⟨*proof*⟩

lemma *mult-mv-nonzero*:
assumes *v* $\neq$ *vec-zero* **and** *mat-det A* $\neq$ *0*
shows *A* $*_{mv}$ *v* $\neq$ *vec-zero*
⟨*proof*⟩

lemma *mult-vm-nonzero*:
assumes *v* $\neq$ *vec-zero* **and** *mat-det A* $\neq$ *0*
shows *v* $*_{vm}$ *A* $\neq$ *vec-zero*
⟨*proof*⟩

lemma *mult-sv-mv*:
shows *k* $*_{sv}$ (*A* $*_{mv}$ *v*) = (*A* $*_{mv}$ (*k* $*_{sv}$ *v*))
⟨*proof*⟩

lemma *mult-mv-mult-vm*:
shows *A* $*_{mv}$ *x* = *x* $*_{vm}$ (*mat-transpose A*)
⟨*proof*⟩

lemma *mult-mv-vv*:
shows *A* $*_{mv}$ *v1* $*_{vv}$ *v2* = *v1* $*_{vv}$ (*mat-transpose A* $*_{mv}$ *v2*)
⟨*proof*⟩

lemma *mult-vv-mv*:
shows *x* $*_{vv}$ (*A* $*_{mv}$ *y*) = (*x* $*_{vm}$ *A*) $*_{vv}$ *y*
⟨*proof*⟩

lemma *vec-cnj-mult-mv*:
shows *vec-cnj* (*A* $*_{mv}$ *x*) = (*mat-cnj A*) $*_{mv}$ (*vec-cnj x*)
⟨*proof*⟩

lemma *vec-cnj-mult-vm*:
shows *vec-cnj* (*v* $*_{vm}$ *A*) = *vec-cnj v* $*_{vm}$ *mat-cnj A*
⟨*proof*⟩

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* $\longleftrightarrow$ ($\exists$ *v*. *v* $\neq$ *vec-zero* $\wedge$ *H* $*_{mv}$ *v* = *k* $*_{sv}$ *v*)

lemma *eigen-equation*:
shows *eigenval k H* $\longleftrightarrow$ k^2 − *mat-trace H* ∗ *k* + *mat-det H* = *0* (**is** *?lhs* $\longleftrightarrow$ *?rhs*)
⟨*proof*⟩

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*

⟨*proof*⟩

lemma *bilinear-form-scale-v1*:
shows *bilinear-form* $(k *_{sv} v1)$ *v2 H* = *cnj k* ∗ *bilinear-form v1 v2 H*
⟨*proof*⟩

lemma *bilinear-form-scale-v2*:
shows *bilinear-form v1* $(k *_{sv} v2)$ *H* = *k* ∗ *bilinear-form v1 v2 H*
⟨*proof*⟩

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*
⟨*proof*⟩

lemma *quad-form-scale-v*:
shows *quad-form* $(k *_{sv} v)$ *H* = *cor* $((cmod\ k)^2)$ ∗ *quad-form v H*
⟨*proof*⟩

lemma *quad-form-scale-m*:
shows *quad-form v* $(k *_{sm} H)$ = *k* ∗ *quad-form v H*
⟨*proof*⟩

lemma *cnj-quad-form* [*simp*]:
shows *cnj* (*quad-form z H*) = *quad-form z* (*mat-adj H*)
⟨*proof*⟩

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*
⟨*proof*⟩

lemma *congruence-congruence*:
shows *congruence M1* (*congruence M2 H*) = *congruence* (*M2* $*_{mm}$ *M1*) *H*
⟨*proof*⟩

lemma *congruence-eye* [*simp*]:
shows *congruence eye H* = *H*
⟨*proof*⟩

lemma *congruence-congruence-inv* [*simp*]:
assumes *mat-det M* ≠ *0*
shows *congruence M* (*congruence* (*mat-inv M*) *H*) = *H*
⟨*proof*⟩

lemma *congruence-inv*:
assumes *mat-det M* ≠ *0* **and** *congruence M H* = *H′*
shows *congruence* (*mat-inv M*) *H′* = *H*
⟨*proof*⟩

lemma *congruence-scale-m* [*simp*]:
shows *congruence M* $(k *_{sm} H)$ = $k *_{sm}$ (*congruence M H*)
⟨*proof*⟩

lemma *inj-congruence*:
assumes *mat-det M* ≠ *0* **and** *congruence M H* = *congruence M H′*

shows $H = H'$
⟨*proof*⟩

lemma *mat-det-congruence* [*simp*]:
shows *mat-det* (*congruence M H*) = (*cor* ((*cmod* (*mat-det M*))^2)) ∗ *mat-det H*
⟨*proof*⟩

lemma *det-sgn-congruence* [*simp*]:
assumes *mat-det M* ≠ *0*
shows *sgn* (*mat-det* (*congruence M H*)) = *sgn* (*mat-det H*)
⟨*proof*⟩

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*⟩

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

lemma *bilinear-form-congruence* [*simp*]:
assumes *mat-det M* ≠ *0*
shows *bilinear-form* (*M* $*_{mv}$ *v1*) (*M* $*_{mv}$ *v2*) (*congruence* (*mat-inv M*) *H*) =
bilinear-form v1 v2 H
⟨*proof*⟩

lemma *quad-form-congruence* [*simp*]:
assumes *mat-det M* ≠ *0*
shows *quad-form* (*M* $*_{mv}$ *z*) (*congruence* (*mat-inv M*) *H*) = *quad-form z H*
⟨*proof*⟩

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* ≠ *0*
shows *mat-det* (*similarity A M*) = *mat-det M*
⟨*proof*⟩

lemma *mat-trace-similarity* [*simp*]:
assumes *mat-det A* ≠ *0*
shows *mat-trace* (*similarity A M*) = *mat-trace M*
⟨*proof*⟩

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

lemma *similarity-eye′* [*simp*]:
shows *similarity* (*1*, *0*, *0*, *1*) *M* = *M*
⟨*proof*⟩

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*
⟨*proof*⟩

lemma *similarity-inv*:
assumes *similarity A M1* = *M2* **and** *mat-det A* ≠ *0*
shows *similarity* (*mat-inv A*) *M2* = *M1*
⟨*proof*⟩

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 $\longleftrightarrow$ *mat-adj M* $*_{mm}$ *M* = *eye*

Generalized unitary matrices

definition *unitary-gen* **where**
  unitary-gen M $\longleftrightarrow$
  ($\exists$ *k*::*complex*. *k* $\neq$ *0* $\wedge$ *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 ($\exists$ *k*::*real*. *k* > *0* $\wedge$ *mat-adj M* $*_{mm}$ *M* = *cor k* $*_{sm}$ *eye*)
⟨*proof*⟩

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* $\longleftrightarrow$
      ($\exists$ *k M'*. *k* > *0* $\wedge$ *unitary M'* $\wedge$ *M* = (*cor k* $*_{sm}$ *eye*) $*_{mm}$ *M'*) (**is** *?lhs* = *?rhs*)
⟨*proof*⟩

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* $\longleftrightarrow$
      ($\exists$ *k*::*complex*. *k* $\neq$ *0* $\wedge$ *congruence M* (*1*, *0*, *0*, *1*) = *k* $*_{sm}$ (*1*, *0*, *0*, *1*))
  ⟨*proof*⟩

Unitary matrices are special cases of general unitary matrices

lemma *unitary-unitary-gen* [*simp*]:
  assumes *unitary M*
  shows *unitary-gen M*
  ⟨*proof*⟩

Generalized unitary matrices are regular

lemma *unitary-gen-regular*:
  assumes *unitary-gen M*
  shows *mat-det M* $\neq$ *0*
⟨*proof*⟩

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*)
  ⟨*proof*⟩

lemma *unitary-comp*:
  assumes *unitary M1* **and** *unitary M2*
  shows *unitary* (*M1* $*_{mm}$ *M2*)

⟨*proof*⟩

lemma *unitary-gen-comp*:
assumes *unitary-gen M1* **and** *unitary-gen M2*
shows *unitary-gen* (*M1* $*_{mm}$ *M2*)
⟨*proof*⟩

lemma *unitary-adj-eq-inv*:
shows *unitary M* $\longleftrightarrow$ *mat-det M* $\neq$ *0* $\wedge$ *mat-adj M* = *mat-inv M*
⟨*proof*⟩

lemma *unitary-inv*:
assumes *unitary M*
shows *unitary* (*mat-inv M*)
⟨*proof*⟩

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

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*⟩

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*⟩

A characterization of all generalized unitary matrices

lemma *unitary-gen-iff*:
shows *unitary-gen M* $\longleftrightarrow$
($\exists$ *a b k*. *k* $\neq$ *0* $\wedge$ *mat-det* (*a*, *b*, $-$*cnj b*, *cnj a*) $\neq$ *0* $\wedge$
M = *k* $*_{sm}$ (*a*, *b*, $-$*cnj b*, *cnj a*)) (**is** *?lhs* = *?rhs*)
⟨*proof*⟩

A characterization of unitary matrices

lemma *unitary-iff*:
shows *unitary M* $\longleftrightarrow$
($\exists$ *a b k*. (*cmod a*)^2 + (*cmod b*)^2 $\neq$ *0* $\wedge$
(*cmod k*)^2 = *1* / ((*cmod a*)^2 + (*cmod b*)^2) $\wedge$
M = *k* $*_{sm}$ (*a*, *b*, $-$*cnj b*, *cnj a*)) (**is** *?lhs* = *?rhs*)
⟨*proof*⟩

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 $\longleftrightarrow$ *congruence M* (*1*, *0*, *0*, -1) = (*1*, *0*, *0*, -1)

Generalized unitary11 matrices

definition *unitary11-gen* **where**
unitary11-gen M $\longleftrightarrow$ ($\exists$ *k*. *k* $\neq$ *0* $\wedge$ *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* $\longleftrightarrow$ ($\exists$ *k*. *k* $\neq$ *0* $\wedge$ *congruence M* (*1*, *0*, *0*, -1) = *cor k* $*_{sm}$ (*1*, *0*, *0*, -1))
⟨*proof*⟩

Unitary11 matrices are special cases of generalized unitary 11 matrices

lemma *unitary11-unitary11-gen* [*simp*]:
assumes *unitary11 M*
shows *unitary11-gen M*
⟨*proof*⟩

All generalized unitary11 matrices are regular

lemma *unitary11-gen-regular*:
assumes *unitary11-gen M*
shows *mat-det M* $\neq$ *0*
⟨*proof*⟩

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 $\exists$ *a b*. *M* = (*a*, *b*, *cnj b*, *cnj a*)
⟨*proof*⟩

lemma *unitary11-gen-special*:
assumes *unitary11-gen M* **and** *mat-det M* = *1*
shows $\exists$ *a b*. *M* = (*a*, *b*, *cnj b*, *cnj a*) $\vee$ *M* = (*a*, *b*, $-$*cnj b*, $-$*cnj a*)
⟨*proof*⟩

A characterization of all generalized unitary11 matrices

lemma *unitary11-gen-iff′*:
shows *unitary11-gen M* $\longleftrightarrow$
($\exists$ *a b k*. *k* $\neq$ *0* $\wedge$ *mat-det* (*a*, *b*, *cnj b*, *cnj a*) $\neq$ *0* $\wedge$
(*M* = *k* $*_{sm}$ (*a*, *b*, *cnj b*, *cnj a*) $\vee$
M = *k* $*_{sm}$ (-1, *0*, *0*, *1*) $*_{mm}$ (*a*, *b*, *cnj b*, *cnj a*))) (**is** *?lhs* = *?rhs*)
⟨*proof*⟩

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

lemma *unitary11-gen-cis-blaschke*:
assumes *k* $\neq$ *0* **and** *M* = *k* $*_{sm}$ (*a*, *b*, *cnj b*, *cnj a*) **and**
a $\neq$ *0* **and** *mat-det* (*a*, *b*, *cnj b*, *cnj a*) $\neq$ *0*
shows $\exists$ *k′ φ a′*. *k′* $\neq$ *0* $\wedge$ *a′* $*$ *cnj a′* $\neq$ *1* $\wedge$
M = *k′* $*_{sm}$ (*cis φ*, *0*, *0*, *1*) $*_{mm}$ (*1*, $-$*a′*, $-$*cnj a′*, *1*)
⟨*proof*⟩

lemma *unitary11-gen-cis-blaschke′*:
assumes *k* $\neq$ *0* **and** *M* = *k* $*_{sm}$ (-1, *0*, *0*, *1*) $*_{mm}$ (*a*, *b*, *cnj b*, *cnj a*) **and**
a $\neq$ *0* **and** *mat-det* (*a*, *b*, *cnj b*, *cnj a*) $\neq$ *0*
shows $\exists$ *k′ φ a′*. *k′* $\neq$ *0* $\wedge$ *a′* $*$ *cnj a′* $\neq$ *1* $\wedge$
M = *k′* $*_{sm}$ (*cis φ*, *0*, *0*, *1*) $*_{mm}$ (*1*, $-$*a′*, $-$*cnj a′*, *1*)
⟨*proof*⟩

lemma *unitary11-gen-cis-blaschke-rev*:

assumes $k' \neq 0$ **and** $M = k' *_{sm} (cis\ \varphi,\ 0,\ 0,\ 1) *_{mm} (1,\ -a',\ -cnj\ a',\ 1)$ **and**
$a' * cnj\ a' \neq 1$
shows $\exists\ k\ a\ b.\ k \neq 0 \wedge mat\text{-}det\ (a,\ b,\ cnj\ b,\ cnj\ a) \neq 0\ \wedge$
$M = k *_{sm} (a,\ b,\ cnj\ b,\ cnj\ a)$
⟨*proof*⟩

lemma *unitary11-gen-cis-inversion*:
assumes $k \neq 0$ **and** $M = k *_{sm} (0,\ b,\ cnj\ b,\ 0)$ **and** $b \neq 0$
shows $\exists\ k'\ \varphi.\ k' \neq 0 \wedge$
$M = k' *_{sm} (cis\ \varphi,\ 0,\ 0,\ 1) *_{mm} (0,\ 1,\ 1,\ 0)$
⟨*proof*⟩

lemma *unitary11-gen-cis-inversion'*:
assumes $k \neq 0$ **and** $M = k *_{sm} (-1,\ 0,\ 0,\ 1) *_{mm} (0,\ b,\ cnj\ b,\ 0)$ **and** $b \neq 0$
shows $\exists\ k'\ \varphi.\ k' \neq 0 \wedge$
$M = k' *_{sm} (cis\ \varphi,\ 0,\ 0,\ 1) *_{mm} (0,\ 1,\ 1,\ 0)$
⟨*proof*⟩

lemma *unitary11-gen-cis-inversion-rev*:
assumes $k' \neq 0$ **and** $M = k' *_{sm} (cis\ \varphi,\ 0,\ 0,\ 1) *_{mm} (0,\ 1,\ 1,\ 0)$
shows $\exists\ k\ a\ b.\ k \neq 0 \wedge mat\text{-}det\ (a,\ b,\ cnj\ b,\ cnj\ a) \neq 0 \wedge$
$M = k *_{sm} (a,\ b,\ cnj\ b,\ cnj\ a)$
⟨*proof*⟩

Another characterization of generalized unitary11 matrices

lemma *unitary11-gen-iff*:
shows *unitary11-gen* $M \longleftrightarrow$
$(\exists\ k\ a\ b.\ k \neq 0 \wedge mat\text{-}det\ (a,\ b,\ cnj\ b,\ cnj\ a) \neq 0 \wedge$
$M = k *_{sm} (a,\ b,\ cnj\ b,\ cnj\ a))$ (**is** *?lhs* = *?rhs*)
⟨*proof*⟩

lemma *unitary11-iff*:
shows *unitary11* $M \longleftrightarrow$
$(\exists\ a\ b\ k.\ (cmod\ a)^2 > (cmod\ b)^2 \wedge$
$(cmod\ k)^2 = 1\ /\ ((cmod\ a)^2 - (cmod\ b)^2) \wedge$
$M = k *_{sm} (a,\ b,\ cnj\ b,\ cnj\ a))$ (**is** *?lhs* = *?rhs*)
⟨*proof*⟩

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*⟩

lemma *unitary11-gen-div-sm*:
assumes $k \neq 0$ **and** *unitary11-gen* $(k *_{sm} M)$
shows *unitary11-gen M*
⟨*proof*⟩

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 $\exists\ k'\ a'\ b'.\ k' \neq 0 \wedge mat\text{-}inv\ M = k' *_{sm} (a',\ b',\ cnj\ b',\ cnj\ a') \wedge mat\text{-}det\ (a',\ b',\ cnj\ b',\ cnj\ a') \neq 0$
⟨*proof*⟩

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 $\exists\ k\ a\ b.\ k \neq 0 \wedge M1 *_{mm} M2 = k *_{sm} (a,\ b,\ cnj\ b,\ cnj\ a) \wedge mat\text{-}det\ (a,\ b,\ cnj\ b,\ cnj\ a) \neq 0$
⟨*proof*⟩

lemma *unitary11-gen-mat-inv*:
assumes *unitary11-gen M* **and** *mat-det M* $\neq$ *0*
shows *unitary11-gen* (*mat-inv M*)
⟨*proof*⟩

lemma *unitary11-gen-comp*:
assumes *unitary11-gen M1* **and** *mat-det M1* $\neq$ *0* **and** *unitary11-gen M2* **and** *mat-det M2* $\neq$ *0*
shows *unitary11-gen* (*M1* $*_{mm}$ *M2*)
⟨*proof*⟩

Classification into orientation-preserving and orientation-reversing matrices

lemma *unitary11-sgn-det-orientation*:
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*)
shows $\exists$ *k'*. *sgn k'* = *sgn* (*Re* (*mat-det* (*a*, *b*, *cnj b*, *cnj a*))) $\wedge$ *congruence M* (*1*, *0*, *0*, *−1*) = *cor k'* $*_{sm}$ (*1*, *0*, *0*, *−1*)
⟨*proof*⟩

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*⟩

lemma *unitary11-orientation*:
assumes *unitary11-gen M* **and** *M* = (*A*, *B*, *C*, *D*)
shows $\exists$ *k'*. *sgn k'* = *sgn* (*if B* = *0 then 1 else sgn* (*Re* ((*A*∗*D*)/(*B*∗*C*)) − *1*)) $\wedge$ *congruence M* (*1*, *0*, *0*, *−1*) = *cor k'* $*_{sm}$ (*1*, *0*, *0*, *−1*)
⟨*proof*⟩

lemma *unitary11-sgn-det-orientation'*:
assumes *congruence M* (*1*, *0*, *0*, *−1*) = *cor k'* $*_{sm}$ (*1*, *0*, *0*, *−1*) **and** *k'* $\neq$ *0*
shows $\exists$ *a b k*. *k* $\neq$ *0* $\wedge$ *M* = *k* $*_{sm}$ (*a*, *b*, *cnj b*, *cnj a*) $\wedge$ *sgn k'* = *sgn* (*Re* (*mat-det* (*a*, *b*, *cnj b*, *cnj a*)))
⟨*proof*⟩

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*
⟨*proof*⟩

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*)
⟨*proof*⟩

lemma *hermitean-mk-circline'* [*simp*]:
assumes *is-real A* **and** *is-real D*
shows *hermitean* (*A*, *B*, *cnj B*, *D*)
⟨*proof*⟩

lemma *hermitean-elems*:
assumes *hermitean* (*A*, *B*, *C*, *D*)
shows *is-real A* **and** *is-real D* **and** $B = cnj\ C$ **and** $cnj\ B = C$
⟨*proof*⟩

Operations that preserve the Hermitean property

lemma *hermitean-mat-cnj*:
shows *hermitean H* $\longleftrightarrow$ *hermitean* (*mat-cnj H*)
⟨*proof*⟩

lemma *hermitean-mult-real*:
assumes *hermitean H*
shows *hermitean* ((*cor k*) $*_{sm}$ *H*)
⟨*proof*⟩

lemma *hermitean-congruence*:
assumes *hermitean H*
shows *hermitean* (*congruence M H*)
⟨*proof*⟩

Identity matrix is Hermitean

lemma *hermitean-eye* [*simp*]:
shows *hermitean eye*
⟨*proof*⟩

lemma *hermitean-eye′* [*simp*]:
shows *hermitean* (*1*, *0*, *0*, *1*)
⟨*proof*⟩

Unit circle matrix is Hermitean

lemma *hermitean-unit-circle* [*simp*]:
shows *hermitean* (*1*, *0*, *0*, -1)
⟨*proof*⟩

Hermitean matrices have real determinant

lemma *mat-det-hermitean-real*:
assumes *hermitean A*
shows *is-real* (*mat-det A*)
⟨*proof*⟩

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*
⟨*proof*⟩

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*⟩

lemma *quad-form-hermitean-real*:
assumes *hermitean H*
shows *is-real* (*quad-form z H*)
⟨*proof*⟩

lemma *quad-form-vec-cnj-mat-cnj*:
assumes *hermitean H*
shows *quad-form* (*vec-cnj z*) (*mat-cnj H*) = *quad-form z H*
⟨*proof*⟩

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*⟩

Non-diagonal Hermitean matrices have distinct eigenvalues

lemma *hermitean-distinct-eigenvals*:
assumes *hermitean H*
shows (∃ k_1 k_2. $k_1 \neq k_2 \wedge$ *eigenval* k_1 *H* $\wedge$ *eigenval* k_2 *H*) $\vee$ *mat-diagonal H*
⟨*proof*⟩

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* $\neq$ *k2*
shows *vec-cnj v2* $*_{vv}$ *v1* = *0* **and** *vec-cnj v1* $*_{vv}$ *v2* = *0*
⟨*proof*⟩

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* $\neq$ *0* $\wedge$ *unitary M* $\wedge$ *congruence M H* = (*k1*, *0*, *0*, *k2*) $\wedge$
is-real k1 $\wedge$ *is-real k2* $\wedge$ *sgn* (*Re k1* $*$ *Re k2*) = *sgn* (*Re* (*mat-det H*))
⟨*proof*⟩

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 $\mathbb{C}$ (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* $\vee$ *Im* ((*z3* − *z1*) / (*z2* − *z1*)) = *0*

lemma *collinear-ex-real*:
shows *collinear z1 z2 z3* ⟷
(∃ *k*::*real*. *z1* = *z2* $\vee$ *z3* − *z1* = *complex-of-real k* $*$ (*z2* − *z1*))
⟨*proof*⟩

Collinearity characterization using determinants

lemma *collinear-det*:
assumes ¬ *collinear z1 z2 z3*
shows *det2* (*z3* − *z1*) (*cnj* (*z3* − *z1*)) (*z1* − *z2*) (*cnj* (*z1* − *z2*)) $\neq$ *0*
⟨*proof*⟩

Properties of three collinear points

lemma *collinear-sym1*:
shows *collinear z1 z2 z3* ⟷ *collinear z1 z3 z2*
⟨*proof*⟩

lemma *collinear-sym2'*:
assumes *collinear z1 z2 z3*
shows *collinear z2 z1 z3*
⟨*proof*⟩

lemma *collinear-sym2*:
shows *collinear z1 z2 z3* ⟷ *collinear z2 z1 z3*
⟨*proof*⟩

Properties of four collinear points

lemma *collinear-trans1*:
assumes *collinear z0 z2 z1* **and** *collinear z0 z3 z1* **and** *z0* ≠ *z1*
shows *collinear z0 z2 z3*
⟨*proof*⟩

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* ∈ *line z z'* **and** *z2* ∈ *line z z'* **and** *z3* ∈ *line z z'* **and** *z* ≠ *z'*
shows *collinear z1 z2 z3*
⟨*proof*⟩

Parametric equation of a line

lemma *line-param*:
shows *z1* + *cor k* ∗ (*z2* − *z1*) ∈ *line z1 z2*
⟨*proof*⟩

Equation of the line containing two different given points

lemma *line-equation*:
assumes *z1* ≠ *z2* **and** *μ* = *rot90* (*z2* − *z1*)
shows *line z1 z2* = {*z*. *cnj μ*∗*z* + *μ*∗*cnj z* − (*cnj μ* ∗ *z1* + *μ* ∗ *cnj z1*) = *0*}
⟨*proof*⟩

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 μ r = {*z*. *cmod* (*z* − *μ*) = *r*}

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

lemma *circle-equation*:
assumes *r* ≥ *0*
shows *circle μ r* = {*z*. *z*∗*cnj z* − *z*∗*cnj μ* − *cnj z*∗*μ* + *μ*∗*cnj μ* − *cor* (*r*∗*r*) = *0*}
⟨*proof*⟩

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*⟩

lemma *circline-ex-circle*:
assumes $A \neq 0$ **and** $A * D \leq (cmod\ BC)^2$ **and** $cl = circline\ A\ BC\ D$
shows $\exists\ \mu\ r.\ cl = circle\ \mu\ r$
⟨*proof*⟩

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*⟩

lemma *circle-ex-circline*:
assumes $cl = circle\ \mu\ r$ **and** $r \geq 0$
shows $\exists\ A\ BC\ D.\ A \neq 0 \wedge A*D \leq (cmod\ BC)^2 \wedge cl = circline\ A\ BC\ D$
⟨*proof*⟩

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*⟩

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*⟩

Every Euclidean line can be represented by a circline

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

lemma *circline-line′*:
assumes $z1 \neq z2$
shows $circline\ 0\ (\mathrm{i} * (z2 - z1))\ (Re\ (-\ cnj\text{-}mix\ (\mathrm{i} * (z2 - z1))\ z1)) = line\ z1\ z2$
⟨*proof*⟩

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**
$sgn\text{-}bool\ p \equiv if\ p\ then\ 1\ else\ -1$

definition *circ-tang-vec* :: *complex* ⇒ *complex* ⇒ *bool* ⇒ *complex* **where**
$circ\text{-}tang\text{-}vec\ \mu\ E\ p = sgn\text{-}bool\ p * \mathrm{i} * (E - \mu)$

Tangent vector is orthogonal to the radius.

lemma *circ-tang-vec-ortho*:
shows *scalprod* $(E - \mu)$ $(circ\text{-}tang\text{-}vec\ \mu\ E\ p) = 0$
⟨*proof*⟩

Changing the circle orientation gives the opposite tangent vector.

lemma *circ-tang-vec-opposite-orient*:
shows *circ-tang-vec* $\mu\ E\ p = -\ circ\text{-}tang\text{-}vec\ \mu\ E\ (\neg\ p)$
⟨*proof*⟩

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**
$ang\text{-}circ\ E\ \mu1\ \mu2\ p1\ p2 = \angle\ (circ\text{-}tang\text{-}vec\ \mu1\ E\ p1)\ (circ\text{-}tang\text{-}vec\ \mu2\ E\ p2)$

The unoriented angle between the two circles

definition *ang-circ-c* **where**
$ang\text{-}circ\text{-}c\ E\ \mu1\ \mu2\ p1\ p2 = \angle c\ (circ\text{-}tang\text{-}vec\ \mu1\ E\ p1)\ (circ\text{-}tang\text{-}vec\ \mu2\ E\ p2)$

The acute angle between the two circles

definition *ang-circ-a* **where**
$ang\text{-}circ\text{-}a\ E\ \mu1\ \mu2\ p1\ p2 = \angle a\ (circ\text{-}tang\text{-}vec\ \mu1\ E\ p1)\ (circ\text{-}tang\text{-}vec\ \mu2\ E\ p2)$

Explicit expression for oriented angle between two circles

lemma *ang-circ-simp*:
assumes $E \neq \mu1$ **and** $E \neq \mu2$
shows $ang\text{-}circ\ E\ \mu1\ \mu2\ p1\ p2 =$
$\lfloor Arg\ (E - \mu2) - Arg\ (E - \mu1) + sgn\text{-}bool\ p1 * pi\ /\ 2 - sgn\text{-}bool\ p2 * pi\ /\ 2 \rfloor$
⟨*proof*⟩

Explicit expression for the cosine of angle between two circles

lemma *cos-ang-circ-simp*:
assumes $E \neq \mu1$ **and** $E \neq \mu2$
shows $cos\ (ang\text{-}circ\ E\ \mu1\ \mu2\ p1\ p2) =$
$sgn\text{-}bool\ (p1 = p2) * cos\ (Arg\ (E - \mu2) - Arg\ (E - \mu1))$
⟨*proof*⟩

Explicit expression for the unoriented angle between two circles

lemma *ang-circ-c-simp*:
assumes $E \neq \mu1$ **and** $E \neq \mu2$
shows $ang\text{-}circ\text{-}c\ E\ \mu1\ \mu2\ p1\ p2 =$
$|\lfloor Arg\ (E - \mu2) - Arg\ (E - \mu1) + sgn\text{-}bool\ p1 * pi\ /\ 2 - sgn\text{-}bool\ p2 * pi\ /\ 2 \rfloor|$
⟨*proof*⟩

Explicit expression for the acute angle between two circles

lemma *ang-circ-a-simp*:
assumes $E \neq \mu1$ **and** $E \neq \mu2$
shows $ang\text{-}circ\text{-}a\ E\ \mu1\ \mu2\ p1\ p2 =$
$acute\text{-}ang\ (abs\ (canon\text{-}ang\ (Arg(E - \mu2) - Arg(E - \mu1) + (sgn\text{-}bool\ p1) * pi/2 - (sgn\text{-}bool\ p2) * pi/2)))$
⟨*proof*⟩

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

lemma *ang-circ-a-pTrue*:
assumes $E \neq \mu1$ **and** $E \neq \mu2$
shows $ang\text{-}circ\text{-}a\ E\ \mu1\ \mu2\ p1\ p2 = ang\text{-}circ\text{-}a\ E\ \mu1\ \mu2\ True\ True$
⟨*proof*⟩

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* $= \angle a\ (E - \mu 1)\ (E - \mu 2)$
⟨*proof*⟩

lemma *ang-circ-a′-simp*:
assumes $E \neq \mu 1$ **and** $E \neq \mu 2$
shows *ang-circ-a′ E μ1 μ2* $= \angle a\ (E - \mu 1)\ (E - \mu 2)$
⟨*proof*⟩

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** $\approx_v$ *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*⟩

lemma *complex-eq-cvec-reflp* [*simp*]:
shows *reflp* ($\approx_v$)
⟨*proof*⟩

lemma *complex-eq-cvec-symp* [*simp*]:
shows *symp* ($\approx_v$)
⟨*proof*⟩

lemma *complex-eq-cvec-transp* [*simp*]:
shows *transp* ($\approx_v$)
⟨*proof*⟩

lemma *complex-eq-cvec-equivp* [*simp*]:
shows *equivp* ($\approx_v$)
⟨*proof*⟩

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

typedef *complex-homo-coords* = {*v*::*complex-vec*. *v* ≠ *vec-zero*}
⟨*proof*⟩

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*
⟨*proof*⟩

lemma *complex-homo-coords-eq-reflp* [*simp*]:
shows *reflp* (≈)
⟨*proof*⟩

lemma *complex-homo-coords-eq-symp* [*simp*]:
shows *symp* (≈)
⟨*proof*⟩

lemma *complex-homo-coords-eq-transp* [*simp*]:
shows *transp* (≈)
⟨*proof*⟩

lemma *complex-homo-coords-eq-equivp*:
shows *equivp* (≈)
⟨*proof*⟩

lemma *complex-homo-coords-eq-refl* [*simp*]:
shows $z \approx z$
⟨*proof*⟩

lemma *complex-homo-coords-eq-sym*:
assumes *z1* ≈ *z2*
shows *z2* ≈ *z1*
⟨*proof*⟩

lemma *complex-homo-coords-eq-trans*:
assumes *z1* ≈ *z2* **and** *z2* ≈ *z3*
shows *z1* ≈ *z3*
⟨*proof*⟩

Quotient type of homogeneous coordinates

quotient-type
complex-homo = *complex-homo-coords* / *complex-homo-coords-eq*
⟨*proof*⟩

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

Infinite point

definition *inf-cvec* :: *complex-vec* (∞_v) **where**
[*simp*]: *inf-cvec* = (*1*, *0*)
lift-definition *inf-hcoords* :: *complex-homo-coords* (∞_{hc}) **is** *inf-cvec*
⟨*proof*⟩
lift-definition *inf* :: *complex-homo* (∞_h) **is** *inf-hcoords*
⟨*proof*⟩

lemma *inf-cvec-z2-zero-iff*:
assumes (*z1*, *z2*) ≠ *vec-zero*
shows (*z1*, *z2*) ≈$_v$ ∞$_v$ ⟷ *z2* = *0*
⟨*proof*⟩

Zero

definition *zero-cvec* :: *complex-vec* (0_v) **where**
[*simp*]: *zero-cvec* = (*0*, *1*)
lift-definition *zero-hcoords* :: *complex-homo-coords* (0_{hc}) **is** *zero-cvec*
⟨*proof*⟩
lift-definition *zero* :: *complex-homo* (0_h) **is** *zero-hcoords*
⟨*proof*⟩

lemma *zero-cvec-z1-zero-iff*:
assumes (*z1*, *z2*) ≠ *vec-zero*
shows (*z1*, *z2*) ≈$_v$ *0*$_v$ ⟷ *z1* = *0*
⟨*proof*⟩

One

definition *one-cvec* :: *complex-vec* (1_v)**where**

[*simp*]: *one-cvec* = (*1*, *1*)
lift-definition *one-hcoords* :: *complex-homo-coords* (1_{hc}) **is** *one-cvec*
⟨*proof*⟩
lift-definition *one* :: *complex-homo* (1_h) **is** *one-hcoords*
⟨*proof*⟩

lemma *zero-one-infty-not-equal* [*simp*]:
shows $1_h \neq \infty_h$ **and** $0_h \neq \infty_h$ **and** $0_h \neq 1_h$ **and** $1_h \neq 0_h$ **and** $\infty_h \neq 0_h$ **and** $\infty_h \neq 1_h$
⟨*proof*⟩

Imaginary unit

definition *ii-cvec* :: *complex-vec* (ii_v) **where**
[*simp*]: *ii-cvec* = (i, *1*)
lift-definition *ii-hcoords* :: *complex-homo-coords* (ii_{hc}) **is** *ii-cvec*
⟨*proof*⟩
lift-definition *ii* :: *complex-homo* (ii_h) **is** *ii-hcoords*
⟨*proof*⟩

lemma *ex-3-different-points*:
fixes *z*::*complex-homo*
shows ∃ *z1 z2*. *z* ≠ *z1* ∧ *z1* ≠ *z2* ∧ *z* ≠ *z2*
⟨*proof*⟩

5.3 Connection to ordinary complex plane 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*
⟨*proof*⟩
lift-definition *of-complex* :: *complex* ⇒ *complex-homo* **is** *of-complex-hcoords*
⟨*proof*⟩

lemma *of-complex-inj*:
assumes *of-complex x* = *of-complex y*
shows *x* = *y*
⟨*proof*⟩

lemma *of-complex-image-inj*:
assumes *of-complex* ‘ *A* = *of-complex* ‘ *B*
shows *A* = *B*
⟨*proof*⟩

lemma *of-complex-not-inf* [*simp*]:
shows *of-complex x* ≠ ∞_h
⟨*proof*⟩

lemma *inf-not-of-complex* [*simp*]:
shows ∞_h ≠ *of-complex x*
⟨*proof*⟩

lemma *inf-or-of-complex*:
shows *z* = ∞_h ∨ (∃ *x*. *z* = *of-complex x*)
⟨*proof*⟩

lemma *of-complex-zero* [*simp*]:
shows *of-complex 0* = 0_h
⟨*proof*⟩

lemma *of-complex-one* [*simp*]:
shows *of-complex 1* = 1_h
⟨*proof*⟩

lemma *of-complex-ii* [*simp*]:
shows *of-complex* i = ii_h

⟨*proof*⟩

lemma *of-complex-zero-iff* [*simp*]:
shows *of-complex x* = 0_h ⟷ *x* = *0*
⟨*proof*⟩

lemma *of-complex-one-iff* [*simp*]:
shows *of-complex x* = 1_h ⟷ *x* = *1*
⟨*proof*⟩

lemma *of-complex-ii-iff* [*simp*]:
shows *of-complex x* = ii_h ⟷ *x* = i
⟨*proof*⟩

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*
⟨*proof*⟩
lift-definition *to-complex* :: *complex-homo* ⇒ *complex* **is** *to-complex-homo-coords*
⟨*proof*⟩

lemma *to-complex-of-complex* [*simp*]:
shows *to-complex* (*of-complex z*) = *z*
⟨*proof*⟩

lemma *of-complex-to-complex* [*simp*]:
assumes *z* ≠ ∞_h
shows (*of-complex* (*to-complex z*)) = *z*
⟨*proof*⟩

lemma *to-complex-zero-zero* [*simp*]:
shows *to-complex* 0_h = *0*
⟨*proof*⟩

lemma *to-complex-one-one* [*simp*]:
shows *to-complex* 1_h = *1*
⟨*proof*⟩

lemma *to-complex-img-one* [*simp*]:
shows *to-complex* ii_h = i
⟨*proof*⟩

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*
⟨*proof*⟩

lift-definition *add* :: *complex-homo* ⇒ *complex-homo* ⇒ *complex-homo* (**infixl** $+_h$ *60*) **is** *add-hcoords*
⟨*proof*⟩

lemma *add-commute*:
shows $z +_h w = w +_h z$
⟨*proof*⟩

lemma *add-zero-right* [*simp*]:
shows $z +_h 0_h = z$
⟨*proof*⟩

lemma *add-zero-left* [*simp*]:
shows $0_h +_h z = z$
⟨*proof*⟩

lemma *of-complex-add-of-complex* [*simp*]:
shows (*of-complex x*) $+_h$ (*of-complex y*) = *of-complex* (*x* + *y*)
⟨*proof*⟩

lemma *of-complex-add-inf* [*simp*]:
shows (*of-complex x*) $+_h \infty_h = \infty_h$
⟨*proof*⟩

lemma *inf-add-of-complex* [*simp*]:
shows $\infty_h +_h$ (*of-complex x*) $= \infty_h$
⟨*proof*⟩

lemma *inf-add-right*:
assumes $z \neq \infty_h$
shows $z +_h \infty_h = \infty_h$
⟨*proof*⟩

lemma *inf-add-left*:
assumes $z \neq \infty_h$
shows $\infty_h +_h z = \infty_h$
⟨*proof*⟩

This is ill-defined, but holds by our definition

lemma *inf-add-inf*:
shows $\infty_h +_h \infty_h = \infty_h$
⟨*proof*⟩

5.4.2 Unary minus

definition *uminus-cvec* :: *complex-vec* ⇒ *complex-vec* ($\sim_v$) **where**
[*simp*]: $\sim_v z$ = (*let* (*z1*, *z2*) = *z* *in* (−*z1*, *z2*))
lift-definition *uminus-hcoords* :: *complex-homo-coords* ⇒ *complex-homo-coords* ($\sim_{hc}$) **is** *uminus-cvec*
⟨*proof*⟩
lift-definition *uminus* :: *complex-homo* ⇒ *complex-homo* ($\sim_h$) **is** *uminus-hcoords*
⟨*proof*⟩

lemma *uminus-of-complex* [*simp*]:
shows $\sim_h$ (*of-complex z*) = *of-complex* (−*z*)
⟨*proof*⟩

lemma *uminus-zero* [*simp*]:
shows $\sim_h 0_h = 0_h$
⟨*proof*⟩

lemma *uminus-inf* [*simp*]:
shows $\sim_h \infty_h = \infty_h$
⟨*proof*⟩

lemma *uminus-inf-iff*:
shows $\sim_h z = \infty_h \longleftrightarrow z = \infty_h$
⟨*proof*⟩

lemma *uminus-id-iff*:

shows $\sim_h\ z = z \longleftrightarrow z = 0_h \vee z = \infty_h$
⟨*proof*⟩

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** $-_h$ *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*)
⟨*proof*⟩

lemma *zero-sub-right*[*simp*]:
shows $z -_h 0_h = z$
⟨*proof*⟩

lemma *zero-sub-left*[*simp*]:
shows $0_h -_h$ *of-complex x* = *of-complex* (−*x*)
⟨*proof*⟩

lemma *zero-sub-one*[*simp*]:
shows $0_h -_h 1_h$ = *of-complex* (−*1*)
⟨*proof*⟩

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

lemma *sub-eq-zero* [*simp*]:
assumes $z \neq \infty_h$
shows $z -_h z = 0_h$
⟨*proof*⟩

lemma *sub-eq-zero-iff*:
assumes $z \neq \infty_h \vee w \neq \infty_h$
shows $z -_h w = 0_h \longleftrightarrow z = w$
⟨*proof*⟩

lemma *inf-sub-left* [*simp*]:
assumes $z \neq \infty_h$
shows $\infty_h -_h z = \infty_h$
⟨*proof*⟩

lemma *inf-sub-right* [*simp*]:
assumes $z \neq \infty_h$
shows $z -_h \infty_h = \infty_h$
⟨*proof*⟩

This is ill-defined, but holds by our definition

lemma *inf-sub-inf*:
shows $\infty_h -_h \infty_h = 0_h$
⟨*proof*⟩

lemma *sub-noteq-inf*:
assumes $z \neq \infty_h$ **and** $w \neq \infty_h$
shows $z -_h w \neq \infty_h$
⟨*proof*⟩

lemma *sub-eq-inf*:
assumes $z -_h w = \infty_h$
shows $z = \infty_h \vee w = \infty_h$
⟨*proof*⟩

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*
⟨*proof*⟩

lift-definition *mult* :: *complex-homo* ⇒ *complex-homo* ⇒ *complex-homo* (**infixl** $*_h$ *70*) **is** *mult-hcoords*
⟨*proof*⟩

lemma *of-complex-mult-of-complex* [*simp*]:
shows (*of-complex z1*) $*_h$ (*of-complex z2*) = *of-complex* (*z1* ∗ *z2*)
⟨*proof*⟩

lemma *mult-commute*:
shows *z1* $*_h$ *z2* = *z2* $*_h$ *z1*
⟨*proof*⟩

lemma *mult-zero-left* [*simp*]:
assumes *z* ≠ ∞_h
shows 0_h $*_h$ *z* = 0_h
⟨*proof*⟩

lemma *mult-zero-right* [*simp*]:
assumes *z* ≠ ∞_h
shows *z* $*_h$ 0_h = 0_h
⟨*proof*⟩

lemma *mult-inf-right* [*simp*]:
assumes *z* ≠ 0_h
shows *z* $*_h$ ∞_h = ∞_h
⟨*proof*⟩

lemma *mult-inf-left* [*simp*]:
assumes *z* ≠ 0_h
shows ∞_h $*_h$ *z* = ∞_h
⟨*proof*⟩

lemma *mult-one-left* [*simp*]:
shows 1_h $*_h$ *z* = *z*
⟨*proof*⟩

lemma *mult-one-right* [*simp*]:
shows *z* $*_h$ 1_h = *z*
⟨*proof*⟩

This is ill-defined, but holds by our definition

lemma *inf-mult-zero*:
shows ∞_h $*_h$ 0_h = 1_h
⟨*proof*⟩
lemma *zero-mult-inf*:
shows 0_h $*_h$ ∞_h = 1_h
⟨*proof*⟩

lemma *mult-eq-inf*:
assumes *z* $*_h$ *w* = ∞_h
shows *z* = ∞_h ∨ *w* = ∞_h
⟨*proof*⟩

lemma *mult-noteq-inf*:
assumes $z \neq \infty_h$ **and** $w \neq \infty_h$
shows $z *_h w \neq \infty_h$
⟨*proof*⟩

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*
⟨*proof*⟩

lift-definition *reciprocal* :: *complex-homo* ⇒ *complex-homo* **is** *reciprocal-hcoords*
⟨*proof*⟩

lemma *reciprocal-involution* [*simp*]: *reciprocal* (*reciprocal z*) = *z*
⟨*proof*⟩

lemma *reciprocal-zero* [*simp*]: *reciprocal* 0_h = ∞_h
⟨*proof*⟩

lemma *reciprocal-inf* [*simp*]: *reciprocal* ∞_h = 0_h
⟨*proof*⟩

lemma *reciprocal-one* [*simp*]: *reciprocal* 1_h = 1_h
⟨*proof*⟩

lemma *reciprocal-inf-iff* [*iff*]: *reciprocal z* = ∞_h ⟷ *z* = 0_h
⟨*proof*⟩

lemma *reciprocal-zero-iff* [*iff*]: *reciprocal z* = 0_h ⟷ *z* = ∞_h
⟨*proof*⟩

lemma *reciprocal-of-complex* [*simp*]:
assumes *z* ≠ *0*
shows *reciprocal* (*of-complex z*) = *of-complex* (*1* / *z*)
⟨*proof*⟩

lemma *reciprocal-real*:
assumes *is-real* (*to-complex z*) **and** *z* ≠ 0_h **and** *z* ≠ ∞_h
shows *Re* (*to-complex* (*reciprocal z*)) = *1* / *Re* (*to-complex z*)
⟨*proof*⟩

lemma *reciprocal-id-iff*:
shows *reciprocal z* = *z* ⟷ *z* = *of-complex 1* ∨ *z* = *of-complex* (−*1*)
⟨*proof*⟩

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* ≠ 0_h
shows *z* $:_h$ 0_h = ∞_h
⟨*proof*⟩

lemma *divide-zero-left* [*simp*]:
assumes *z* ≠ 0_h
shows 0_h $:_h$ *z* = 0_h
⟨*proof*⟩

lemma *divide-inf-right* [*simp*]:
assumes *z* ≠ ∞_h

shows $z :_h \infty_h = 0_h$
⟨*proof*⟩

lemma *divide-inf-left* [*simp*]:
assumes $z \neq \infty_h$
shows $\infty_h :_h z = \infty_h$
⟨*proof*⟩

lemma *divide-eq-inf*:
assumes $z :_h w = \infty_h$
shows $z = \infty_h \vee w = 0_h$
⟨*proof*⟩

lemma *inf-divide-zero* [*simp*]:
shows $\infty_h :_h 0_h = \infty_h$
⟨*proof*⟩

lemma *zero-divide-inf* [*simp*]:
shows $0_h :_h \infty_h = 0_h$
⟨*proof*⟩

lemma *divide-one-right* [*simp*]:
shows $z :_h 1_h = z$
⟨*proof*⟩

lemma *of-complex-divide-of-complex* [*simp*]:
assumes *z2* $\neq$ *0*
shows (*of-complex z1*) $:_h$ (*of-complex z2*) = *of-complex* (*z1 / z2*)
⟨*proof*⟩

lemma *one-div-of-complex* [*simp*]:
assumes $x \neq 0$
shows $1_h :_h$ *of-complex x* = *of-complex* (*1 / x*)
⟨*proof*⟩

This is ill-defined, but holds by our definition

lemma *inf-divide-inf*:
shows $\infty_h :_h \infty_h = 1_h$
⟨*proof*⟩

This is ill-defined, but holds by our definition

lemma *zero-divide-zero*:
shows $0_h :_h 0_h = 1_h$
⟨*proof*⟩

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*
⟨*proof*⟩
lift-definition *conjugate* :: *complex-homo* ⇒ *complex-homo* **is** *conjugate-hcoords*
⟨*proof*⟩

lemma *conjugate-involution* [*simp*]:
shows *conjugate* (*conjugate z*) = *z*
⟨*proof*⟩

lemma *conjugate-conjugate-comp* [*simp*]:
shows *conjugate* ∘ *conjugate* = *id*
⟨*proof*⟩

lemma *inv-conjugate* [*simp*]:
shows *inv conjugate* = *conjugate*
⟨*proof*⟩

lemma *conjugate-of-complex* [*simp*]:
shows *conjugate* (*of-complex z*) = *of-complex* (*cnj z*)
⟨*proof*⟩

lemma *conjugate-inf* [*simp*]:
shows *conjugate* $\infty_h = \infty_h$
⟨*proof*⟩

lemma *conjugate-zero* [*simp*]:
shows *conjugate* $0_h = 0_h$
⟨*proof*⟩

lemma *conjugate-one* [*simp*]:
shows *conjugate* $1_h = 1_h$
⟨*proof*⟩

lemma *conjugate-inj*:
assumes *conjugate x* = *conjugate y*
shows *x* = *y*
⟨*proof*⟩

lemma *bij-conjugate* [*simp*]:
shows *bij conjugate*
⟨*proof*⟩

lemma *conjugate-id-iff*:
shows *conjugate a* = *a* ⟷ *is-real* (*to-complex a*) ∨ *a* = ∞_h
⟨*proof*⟩

5.4.8 Inversion

Geometric inversion wrt. the unit circle

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

lemma *inversion-sym*:
shows *inversion* = *reciprocal* ∘ *conjugate*
⟨*proof*⟩

lemma *inversion-involution* [*simp*]:
shows *inversion* (*inversion z*) = *z*
⟨*proof*⟩

lemma *inversion-inversion-id* [*simp*]:
shows *inversion* ∘ *inversion* = *id*
⟨*proof*⟩

lemma *inversion-zero* [*simp*]:
shows *inversion* $0_h = \infty_h$
⟨*proof*⟩

lemma *inversion-infty* [*simp*]:
shows *inversion* $\infty_h = 0_h$
⟨*proof*⟩

lemma *inversion-of-complex* [*simp*]:
assumes *z* ≠ *0*
shows *inversion* (*of-complex z*) = *of-complex* (*1 / cnj z*)
⟨*proof*⟩

lemma *is-real-inversion*:
assumes *is-real x* **and** *x* ≠ *0*
shows *is-real* (*to-complex* (*inversion* (*of-complex x*)))
⟨*proof*⟩

lemma *inversion-id-iff*:

shows $a = \mathit{inversion}\ a \longleftrightarrow a \neq \infty_h \wedge (\mathit{to\text{-}complex}\ a) * \mathit{cnj}\ (\mathit{to\text{-}complex}\ a) = 1$ (**is** *?lhs = ?rhs*)
⟨*proof*⟩

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*$_h$
⟨*proof*⟩

lemma
assumes *za* ≠ ∞$_h$ **and** *zc* ≠ ∞$_h$
shows *ratio za* ∞$_h$ *zc* = ∞$_h$
⟨*proof*⟩

lemma
assumes *za* ≠ ∞$_h$ **and** *zb* ≠ ∞$_h$
shows *ratio za zb* ∞$_h$ = *0*$_h$
⟨*proof*⟩

lemma
assumes *z1* ≠ *z2* **and** *z1* ≠ ∞$_h$
shows *ratio z1 z2 z1* = ∞$_h$
⟨*proof*⟩

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*
⟨*proof*⟩

lift-definition *cross-ratio* :: *complex-homo* ⇒ *complex-homo* ⇒ *complex-homo* ⇒ *complex-homo* ⇒ *complex-homo* **is** *cross-ratio-hcoords*
⟨*proof*⟩

lemma *cross-ratio-01inf-id* [*simp*]:
shows *cross-ratio z 0*$_h$ *1*$_h$ ∞$_h$ = *z*
⟨*proof*⟩

lemma *cross-ratio-0*:
assumes $u \neq v$ **and** $u \neq w$
shows *cross-ratio* $u\ u\ v\ w = 0_h$
⟨*proof*⟩

lemma *cross-ratio-1*:
assumes $u \neq v$ **and** $v \neq w$
shows *cross-ratio* $v\ u\ v\ w = 1_h$
⟨*proof*⟩

lemma *cross-ratio-inf*:
assumes $u \neq w$ **and** $v \neq w$
shows *cross-ratio* $w\ u\ v\ w = \infty_h$
⟨*proof*⟩

lemma *cross-ratio-0inf*:
assumes $y \neq 0$
shows *cross-ratio* (*of-complex* x) 0_h (*of-complex* y) ∞_h = (*of-complex* $(x\ /\ y)$)
⟨*proof*⟩

lemma *cross-ratio-commute-13*:
shows *cross-ratio* $z\ u\ v\ w$ = *reciprocal* (*cross-ratio* $v\ u\ z\ w$)
⟨*proof*⟩

lemma *cross-ratio-commute-24*:
shows *cross-ratio* $z\ u\ v\ w$ = *reciprocal* (*cross-ratio* $z\ w\ v\ u$)
⟨*proof*⟩

lemma *cross-ratio-not-inf*:
assumes $z \neq w$ **and** $u \neq v$
shows *cross-ratio* $z\ u\ v\ w \neq \infty_h$
⟨*proof*⟩

lemma *cross-ratio-not-zero*:
assumes $z \neq u$ **and** $v \neq w$
shows *cross-ratio* $z\ u\ v\ w \neq 0_h$
⟨*proof*⟩

lemma *cross-ratio-real*:
assumes *is-real* z **and** *is-real* u **and** *is-real* v **and** *is-real* w
assumes $z \neq u \wedge v \neq w \vee z \neq w \wedge u \neq v$
shows *is-real* (*to-complex* (*cross-ratio* (*of-complex* z) (*of-complex* u) (*of-complex* v) (*of-complex* w)))
⟨*proof*⟩

lemma *cross-ratio*:
assumes $(z \neq u \wedge v \neq w) \vee (z \neq w \wedge u \neq v)$ **and**
$z \neq \infty_h$ **and** $u \neq \infty_h$ **and** $v \neq \infty_h$ **and** $w \neq \infty_h$
shows *cross-ratio* $z\ u\ v\ w = ((z -_h u) *_h (v -_h w)) :_h ((z -_h w) *_h (v -_h u))$
⟨*proof*⟩

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*}
⟨*proof*⟩

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*
⟨*proof*⟩

lemma *moebius-mat-eq-refl* [*simp*]:
shows *moebius-mat-eq x x*
⟨*proof*⟩

quotient-type *moebius* = *moebius-mat* / *moebius-mat-eq*
⟨*proof*⟩

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*
⟨*proof*⟩

lift-definition *mk-moebius* :: *complex* ⇒ *complex* ⇒ *complex* ⇒ *complex* ⇒ *moebius* **is** *mk-moebius-mat*
⟨*proof*⟩

lemma *ex-mk-moebius*:
shows ∃ *a b c d*. *M* = *mk-moebius a b c d* ∧ *mat-det* (*a*, *b*, *c*, *d*) ≠ *0*
⟨*proof*⟩

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*
⟨*proof*⟩

lift-definition *moebius-pt* :: *moebius* ⇒ *complex-homo* ⇒ *complex-homo* **is** *moebius-pt-mmat-hcoords*
⟨*proof*⟩

lemma *bij-moebius-pt* [*simp*]:
shows *bij* (*moebius-pt M*)
⟨*proof*⟩

lemma *moebius-pt-eq-I*:
assumes *moebius-pt M z1* = *moebius-pt M z2*
shows *z1* = *z2*
⟨*proof*⟩

lemma *moebius-pt-neq-I* [*simp*]:
assumes *z1* ≠ *z2*
shows *moebius-pt M z1* ≠ *moebius-pt M z2*

⟨*proof*⟩

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*))
⟨*proof*⟩

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*
⟨*proof*⟩

lift-definition *id-moebius* :: *moebius* **is** *id-moebius-mmat*
⟨*proof*⟩

lemma *moebius-pt-moebius-id* [*simp*]:
shows *moebius-pt id-moebius* = *id*
⟨*proof*⟩

lemma *mk-moeibus-id* [*simp*]:
shows *mk-moebius a 0 0 a* = *id-moebius*
⟨*proof*⟩

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*
⟨*proof*⟩

lift-definition *moebius-inv* :: *moebius* ⇒ *moebius* **is** *moebius-inv-mmat*
⟨*proof*⟩

lemma *moebius-inv*:
shows *moebius-pt* (*moebius-inv M*) = *inv* (*moebius-pt M*)
⟨*proof*⟩

lemma *is-moebius-inv* [*simp*]:
assumes *is-moebius m*
shows *is-moebius* (*inv m*)
⟨*proof*⟩

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*))

⟨*proof*⟩

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*
⟨*proof*⟩

lift-definition *moebius-comp* :: *moebius* ⇒ *moebius* ⇒ *moebius* **is** *moebius-comp-mmat*
⟨*proof*⟩

lemma *moebius-comp*:
shows *moebius-pt* (*moebius-comp M1 M2*) = *moebius-pt M1* ∘ *moebius-pt M2*
⟨*proof*⟩

lemma *moebius-pt-comp* [*simp*]:
shows *moebius-pt* (*moebius-comp M1 M2*) *z* = *moebius-pt M1* (*moebius-pt M2 z*)
⟨*proof*⟩

lemma *is-moebius-comp* [*simp*]:
assumes *is-moebius m1* **and** *is-moebius m2*
shows *is-moebius* (*m1* ∘ *m2*)
⟨*proof*⟩

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′*)
⟨*proof*⟩

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*⟩
end

Composition with inverse

lemma *moebius-comp-inv-left* [*simp*]:
shows *moebius-comp* (*moebius-inv M*) *M* = *id-moebius*
⟨*proof*⟩

lemma *moebius-comp-inv-right* [*simp*]:
shows *moebius-comp M* (*moebius-inv M*) = *id-moebius*
⟨*proof*⟩

lemma *moebius-pt-comp-inv-left* [*simp*]:
shows *moebius-pt* (*moebius-inv M*) (*moebius-pt M z*) = *z*
⟨*proof*⟩

lemma *moebius-pt-comp-inv-right* [*simp*]:
shows *moebius-pt M* (*moebius-pt* (*moebius-inv M*) *z*) = *z*
⟨*proof*⟩

lemma *moebius-pt-comp-inv-image-left* [*simp*]:

shows *moebius-pt* (*moebius-inv M*) ‘ *moebius-pt M ‘ A* = *A*
⟨*proof*⟩

lemma *moebius-pt-comp-inv-image-right* [*simp*]:
shows *moebius-pt M ‘ moebius-pt* (*moebius-inv M*) ‘ *A* = *A*
⟨*proof*⟩

lemma *moebius-pt-invert*:
assumes *moebius-pt M z1* = *z2*
shows *moebius-pt* (*moebius-inv M*) *z2* = *z1*
⟨*proof*⟩

lemma *moebius-pt-moebius-inv-in-set* [*simp*]:
assumes *moebius-pt M z* ∈ *A*
shows *z* ∈ *moebius-pt* (*moebius-inv M*) ‘ *A*
⟨*proof*⟩

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*
⟨*proof*⟩

lemma *moebius-reciprocal-inv* [*simp*]:
shows *moebius-inv moebius-reciprocal* = *moebius-reciprocal*
⟨*proof*⟩

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* ≠ *0*
shows *moebius-pt* (*moebius-similarity a b*) *z* = (*of-complex a*) ∗$_h$ *z* +$_h$ (*of-complex b*)
⟨*proof*⟩

Their action is a linear transformation of ℂ.

lemma *moebius-pt-moebius-similarity′*:
assumes *a* ≠ *0*
shows *moebius-pt* (*moebius-similarity a b*) = (λ *z*. (*of-complex a*) ∗$_h$ *z* +$_h$ (*of-complex b*))
⟨*proof*⟩

lemma *is-moebius-similarity′*:
assumes *a* ≠ *0*$_h$ **and** *a* ≠ ∞$_h$ **and** *b* ≠ ∞$_h$
shows (λ *z*. *a* ∗$_h$ *z* +$_h$ *b*) = *moebius-pt* (*moebius-similarity* (*to-complex a*) (*to-complex b*))
⟨*proof*⟩

lemma *is-moebius-similarity*:
assumes *a* ≠ *0*$_h$ **and** *a* ≠ ∞$_h$ **and** *b* ≠ ∞$_h$
shows *is-moebius* (λ *z*. *a* ∗$_h$ *z* +$_h$ *b*)
⟨*proof*⟩

Euclidean similarities form a group.

lemma *moebius-similarity-id* [*simp*]:
shows *moebius-similarity 1 0* = *id-moebius*
⟨*proof*⟩

lemma *moebius-similarity-inv* [*simp*]:
assumes *a* ≠ *0*

shows *moebius-inv* (*moebius-similarity a b*) = *moebius-similarity* (*1*/*a*) (−*b*/*a*)
⟨*proof*⟩

lemma *moebius-similarity-uminus* [*simp*]:
assumes *a* ≠ *0*
shows − *moebius-similarity a b* = *moebius-similarity* (*1*/*a*) (−*b*/*a*)
⟨*proof*⟩

lemma *moebius-similarity-comp* [*simp*]:
assumes *a* ≠ *0* **and** *c* ≠ *0*
shows *moebius-comp* (*moebius-similarity a b*) (*moebius-similarity c d*) = *moebius-similarity* (*a*∗*c*) (*a*∗*d*+*b*)
⟨*proof*⟩

lemma *moebius-similarity-plus* [*simp*]:
assumes *a* ≠ *0* **and** *c* ≠ *0*
shows *moebius-similarity a b* + *moebius-similarity c d* = *moebius-similarity* (*a*∗*c*) (*a*∗*d*+*b*)
⟨*proof*⟩

Euclidean similarities are the only Möbius group elements such that their action leaves the ∞_h fixed.

lemma *moebius-similarity-inf* [*simp*]:
assumes *a* ≠ *0*
shows *moebius-pt* (*moebius-similarity a b*) ∞_h = ∞_h
⟨*proof*⟩

lemma *moebius-similarity-only-inf-to-inf*:
assumes *a* ≠ *0* *moebius-pt* (*moebius-similarity a b*) *z* = ∞_h
shows *z* = ∞_h
⟨*proof*⟩

lemma *moebius-similarity-inf-iff* [*simp*]:
assumes *a* ≠ *0*
shows *moebius-pt* (*moebius-similarity a b*) *z* = ∞_h ⟷ *z* = ∞_h
⟨*proof*⟩

lemma *inf-fixed-only-moebius-similarity*:
assumes *moebius-pt M* ∞_h = ∞_h
shows ∃ *a b*. *a* ≠ *0* ∧ *M* = *moebius-similarity a b*
⟨*proof*⟩

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*)
⟨*proof*⟩

lemma *moebius-translation-plus* [*simp*]:
shows (*moebius-translation v1*) + (*moebius-translation v2*) = *moebius-translation* (*v1* + *v2*)
⟨*proof*⟩

lemma *moebius-translation-zero* [*simp*]:
shows *moebius-translation 0* = *id-moebius*
⟨*proof*⟩

lemma *moebius-translation-inv* [*simp*]:
shows *moebius-inv* (*moebius-translation v1*) = *moebius-translation* (−*v1*)
⟨*proof*⟩

lemma *moebius-translation-uminus* [*simp*]:
shows − (*moebius-translation v1*) = *moebius-translation* (−*v1*)
⟨*proof*⟩

lemma *moebius-translation-inv-translation* [*simp*]:
shows *moebius-pt* (*moebius-translation v*) (*moebius-pt* (*moebius-translation* (−*v*)) *z*) = *z*
⟨*proof*⟩

lemma *moebius-inv-translation-translation* [*simp*]:
shows *moebius-pt* (*moebius-translation* (−*v*)) (*moebius-pt* (*moebius-translation v*) *z*) = *z*
⟨*proof*⟩

lemma *moebius-pt-moebius-translation* [*simp*]:
shows *moebius-pt* (*moebius-translation v*) (*of-complex z*) = *of-complex* (*z* + *v*)
⟨*proof*⟩

lemma *moebius-pt-moebius-translation-inf* [*simp*]:
shows *moebius-pt* (*moebius-translation v*) ∞$_h$ = ∞$_h$
⟨*proof*⟩

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*)
⟨*proof*⟩

lemma *moebius-rotation-plus* [*simp*]:
shows (*moebius-rotation φ1*) + (*moebius-rotation φ2*) = *moebius-rotation* (*φ1* + *φ2*)
⟨*proof*⟩

lemma *moebius-rotation-zero* [*simp*]:
shows *moebius-rotation 0* = *id-moebius*
⟨*proof*⟩

lemma *moebius-rotation-inv* [*simp*]:
shows *moebius-inv* (*moebius-rotation φ*) = *moebius-rotation* (− *φ*)
⟨*proof*⟩

lemma *moebius-rotation-uminus* [*simp*]:
shows − (*moebius-rotation φ*) = *moebius-rotation* (− *φ*)
⟨*proof*⟩

lemma *moebius-rotation-inv-rotation* [*simp*]:
shows *moebius-pt* (*moebius-rotation φ*) (*moebius-pt* (*moebius-rotation* (−*φ*)) *z*) = *z*
⟨*proof*⟩

lemma *moebius-inv-rotation-rotation* [*simp*]:
shows *moebius-pt* (*moebius-rotation* (−*φ*)) (*moebius-pt* (*moebius-rotation φ*) *z*) = *z*
⟨*proof*⟩

lemma *moebius-pt-moebius-rotation* [*simp*]:
shows *moebius-pt* (*moebius-rotation φ*) (*of-complex z*) = *of-complex* (*cis φ* ∗ *z*)
⟨*proof*⟩

lemma *moebius-pt-moebius-rotation-inf* [*simp*]:
shows *moebius-pt* (*moebius-rotation v*) ∞$_h$ = ∞$_h$
⟨*proof*⟩

lemma *moebius-pt-rotation-inf-iff* [*simp*]:
shows *moebius-pt* (*moebius-rotation v*) *x* = ∞$_h$ ⟷ *x* = ∞$_h$
⟨*proof*⟩

lemma *moebius-pt-moebius-rotation-zero* [*simp*]:
shows *moebius-pt* (*moebius-rotation φ*) *0*$_h$ = *0*$_h$
⟨*proof*⟩

lemma *moebius-pt-moebius-rotation-zero-iff* [*simp*]:

shows *moebius-pt* (*moebius-rotation* φ) $x = 0_h \longleftrightarrow x = 0_h$
⟨*proof*⟩

lemma *moebius-rotation-preserve-cmod* [*simp*]:
assumes $u \neq \infty_h$
shows *cmod* (*to-complex* (*moebius-pt* (*moebius-rotation* φ) *u*)) = *cmod* (*to-complex u*)
⟨*proof*⟩

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*)
⟨*proof*⟩

lemma *moebius-dilatation-plus* [*simp*]:
assumes *a1* > *0* **and** *a2* > *0*
shows (*moebius-dilatation a1*) + (*moebius-dilatation a2*) = *moebius-dilatation* (*a1* ∗ *a2*)
⟨*proof*⟩

lemma *moebius-dilatation-zero* [*simp*]:
shows *moebius-dilatation 1* = *id-moebius*
⟨*proof*⟩

lemma *moebius-dilatation-inverse* [*simp*]:
assumes *a* > *0*
shows *moebius-inv* (*moebius-dilatation a*) = *moebius-dilatation* (*1*/*a*)
⟨*proof*⟩

lemma *moebius-dilatation-uminus* [*simp*]:
assumes *a* > *0*
shows − (*moebius-dilatation a*) = *moebius-dilatation* (*1*/*a*)
⟨*proof*⟩

lemma *moebius-pt-dilatation* [*simp*]:
assumes *a* ≠ *0*
shows *moebius-pt* (*moebius-dilatation a*) (*of-complex z*) = *of-complex* (*cor a* ∗ *z*)
⟨*proof*⟩

6.4.6 Rotation-dilatation

definition *moebius-rotation-dilatation* **where**
moebius-rotation-dilatation a = *moebius-similarity a 0*

lemma *moebius-rotation-dilatation*:
assumes *a* ≠ *0*
shows *moebius-rotation-dilatation a* = *moebius-rotation* (*Arg a*) + *moebius-dilatation* (*cmod a*)
⟨*proof*⟩

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* ⇒ *moebius-mat* **is** *mat-cnj*
⟨*proof*⟩
lift-definition *conjugate-moebius* :: *moebius* ⇒ *moebius* **is** *conjugate-moebius-mmat*
⟨*proof*⟩

lemma *conjugate-moebius*:
shows *conjugate* ∘ *moebius-pt M* = *moebius-pt* (*conjugate-moebius M*) ∘ *conjugate*
⟨*proof*⟩

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*⟩

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)
⟨*proof*⟩

lemma *moebius-decomposition-similarity*:
assumes $a \neq 0$
shows *mk-moebius a b 0 d* = *moebius-similarity* (a/d) (b/d)
⟨*proof*⟩

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: $\bigwedge \alpha.$ *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*⟩

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*⟩

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 $z1 \neq z2$ **and** $z1 \neq z3$ **and** $z2 \neq z3$
shows $\exists\ M.$ ((*moebius-pt M z1* = 0_h) $\wedge$ (*moebius-pt M z2* = 1_h) $\wedge$ (*moebius-pt M z3* = ∞_h))
⟨*proof*⟩

There is a Möbius transformation mapping any three different points to any three different points.

lemma *ex-moebius*:
assumes $z1 \neq z2$ **and** $z1 \neq z3$ **and** $z2 \neq z3$
$w1 \neq w2$ **and** $w1 \neq w3$ **and** $w2 \neq w3$
shows $\exists\ M.$ ((*moebius-pt M z1* = *w1*) $\wedge$ (*moebius-pt M z2* = *w2*) $\wedge$ (*moebius-pt M z3* = *w3*))
⟨*proof*⟩

lemma *ex-moebius-1*:
shows $\exists\ M.$ *moebius-pt M z1* = *w1*
⟨*proof*⟩

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* 0_h 1_h ∞_h **and** *z1* $\neq$ *z2* **and** *z2* $\neq$ *z3* **and** *z1* $\neq$ *z3*
$\bigwedge$ *M a b c*. *P a b c* $\Longrightarrow$ *P* (*moebius-pt M a*) (*moebius-pt M b*) (*moebius-pt M c*)
shows *P z1 z2 z3*
⟨*proof*⟩

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*
⟨*proof*⟩

lemma *three-fixed-points*:
assumes *z1* $\neq$ *z2* **and** *z1* $\neq$ *z3* **and** *z2* $\neq$ *z3*
assumes *moebius-pt M z1* = *z1* **and** *moebius-pt M z2* = *z2* **and** *moebius-pt M z3* = *z3*
shows *M* = *id-moebius*
⟨*proof*⟩

lemma *unique-moebius-three-points*:
assumes *z1* $\neq$ *z2* **and** *z1* $\neq$ *z3* **and** *z2* $\neq$ *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*⟩

There is a unique Möbius transformation mapping three different points to other three different points.

lemma *ex-unique-moebius-three-points*:
assumes *z1* $\neq$ *z2* **and** *z1* $\neq$ *z3* **and** *z2* $\neq$ *z3*
w1 $\neq$ *w2* **and** *w1* $\neq$ *w3* **and** *w2* $\neq$ *w3*
shows $\exists!$ *M*. ((*moebius-pt M z1* = *w1*) $\wedge$ (*moebius-pt M z2* = *w2*) $\wedge$ (*moebius-pt M z3* = *w3*))
⟨*proof*⟩

lemma *ex-unique-moebius-three-points-fun*:
assumes *z1* $\neq$ *z2* **and** *z1* $\neq$ *z3* **and** *z2* $\neq$ *z3*
w1 $\neq$ *w2* **and** *w1* $\neq$ *w3* **and** *w2* $\neq$ *w3*
shows $\exists!$ *f*. *is-moebius f* $\wedge$ (*f z1* = *w1*) $\wedge$ (*f z2* = *w2*) $\wedge$ (*f z3* = *w3*)
⟨*proof*⟩

Different Möbius transformations produce different actions.

lemma *unique-moebius-pt*:
assumes *moebius-pt M1* = *moebius-pt M2*
shows *M1* = *M2*
⟨*proof*⟩

lemma *is-cross-ratio-01inf*:
assumes *z1* $\neq$ *z2* **and** *z1* $\neq$ *z3* **and** *z2* $\neq$ *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*)
⟨*proof*⟩

Möbius transformations preserve cross-ratio.

lemma *moebius-preserve-cross-ratio* [*simp*]:
assumes *z1* $\neq$ *z2* **and** *z1* $\neq$ *z3* **and** *z2* $\neq$ *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*⟩

lemma *conjugate-cross-ratio* [*simp*]:
assumes *z1* $\neq$ *z2* **and** *z1* $\neq$ *z3* **and** *z2* $\neq$ *z3*
shows *cross-ratio* (*conjugate z*) (*conjugate z1*) (*conjugate z2*) (*conjugate z3*) =
conjugate (*cross-ratio z z1 z2 z3*)

⟨*proof*⟩

lemma *cross-ratio-reciprocal* [*simp*]:
assumes $u \neq v$ **and** $v \neq w$ **and** $u \neq w$
shows *cross-ratio* (*reciprocal z*) (*reciprocal u*) (*reciprocal v*) (*reciprocal w*) =
cross-ratio z u v w
⟨*proof*⟩

lemma *cross-ratio-inversion* [*simp*]:
assumes $u \neq v$ **and** $v \neq w$ **and** $u \neq w$
shows *cross-ratio* (*inversion z*) (*inversion u*) (*inversion v*) (*inversion w*) =
conjugate (*cross-ratio z u v w*)
⟨*proof*⟩

lemma *fixed-points-0inf'*:
assumes *moebius-pt M* $0_h = 0_h$ **and** *moebius-pt M* $\infty_h = \infty_h$
shows $\exists$ *k*::*complex-homo*. $(k \neq 0_h \wedge k \neq \infty_h) \wedge (\forall$ *z*. *moebius-pt M z* = *k* $*_h$ *z*)
⟨*proof*⟩

lemma *fixed-points-0inf*:
assumes *moebius-pt M* $0_h = 0_h$ **and** *moebius-pt M* $\infty_h = \infty_h$
shows $\exists$ *k*::*complex-homo*. $(k \neq 0_h \wedge k \neq \infty_h) \wedge$ *moebius-pt M* = $(\lambda$ *z*. *k* $*_h$ *z*)
⟨*proof*⟩

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*⟩

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*⟩

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*
⟨*proof*⟩

6.8 Pole

definition *is-pole* :: *moebius* ⇒ *complex-homo* ⇒ *bool* **where**
is-pole M z ⟷ *moebius-pt M z* = ∞_h

lemma *ex1-pole*:
shows $\exists!$ *z*. *is-pole M z*
⟨*proof*⟩

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*⟩

lemma *pole-similarity*:
assumes *is-pole* (*moebius-similarity a b*) *z* **and** $a \neq 0$
shows $z = \infty_h$
⟨*proof*⟩

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*⟩

lemma *conjugation-is-antihomography*[*simp*]:
shows *is-antihomography conjugate*
⟨*proof*⟩

lemma *inversion-is-antihomography* [*simp*]:
shows *is-antihomography inversion*
⟨*proof*⟩

Functions cannot simultaneously be homographies and antihomographies - the disjunction is exclusive.

lemma *homography-antihomography-exclusive*:
assumes *is-antihomography f*
shows ¬ *is-homography f*
⟨*proof*⟩

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*
⟨*proof*⟩

lemma *similarity-scale-2*:
shows *similarity I* (*k* $*_{sm}$ *M*) = *k* $*_{sm}$ (*similarity I M*)
⟨*proof*⟩

lemma *mat-trace-mult-sm* [*simp*]:
shows *mat-trace* (*k* $*_{sm}$ *M*) = *k* ∗ *mat-trace M*
⟨*proof*⟩

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*
⟨*proof*⟩

lift-definition *moebius-mb* :: *moebius* ⇒ *moebius* ⇒ *moebius* **is** *moebius-mb-mmat*
⟨*proof*⟩

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*
⟨*proof*⟩

lift-definition *similarity-invar* :: *moebius* ⇒ *complex* **is** *similarity-invar-mmat*

⟨*proof*⟩

lemma *similarity-invar-moeibus-mb*:
shows *similarity-invar* (*moebius-mb I M*) = *similarity-invar M*
⟨*proof*⟩

definition *similar* :: *moebius* ⇒ *moebius* ⇒ *bool* **where**
similar M1 M2 ⟷ (∃ *I*. *moebius-mb I M1* = *M2*)

lemma *similar-refl* [*simp*]:
shows *similar M M*
⟨*proof*⟩

lemma *similar-sym*:
assumes *similar M1 M2*
shows *similar M2 M1*
⟨*proof*⟩

lemma *similar-trans*:
assumes *similar M1 M2* **and** *similar M2 M3*
shows *similar M1 M3*
⟨*proof*⟩

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 ≡ {*H*. *hermitean H* ∧ *H* ≠ *mat-zero*}

typedef *circline-mat* = *hermitean-nonzero*
⟨*proof*⟩

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* ≠ *0* ∧ *B* = *cor k* ∗$_{sm}$ *A*)

lemma *symp-circline-eq-cmat*: *symp circline-eq-cmat*
⟨*proof*⟩

Hermitean non-zero matrices are equivalent only to such matrices

lemma *circline-eq-cmat-hermitean-nonzero*:
assumes *hermitean H* ∧ *H* ≠ *mat-zero circline-eq-cmat H H′*
shows *hermitean H′* ∧ *H′* ≠ *mat-zero*
⟨*proof*⟩

lift-definition *circline-eq-clmat* :: *circline-mat* ⇒ *circline-mat* ⇒ *bool* **is** *circline-eq-cmat*
⟨*proof*⟩

lemma *circline-eq-clmat-refl* [*simp*]: *circline-eq-clmat H H*
⟨*proof*⟩

quotient-type *circline* = *circline-mat* / *circline-eq-clmat*
⟨*proof*⟩

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*
⟨*proof*⟩

lift-definition *mk-circline* :: *complex* ⇒ *complex* ⇒ *complex* ⇒ *complex* ⇒ *circline* **is** *mk-circline-clmat*
⟨*proof*⟩

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*⟩

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*
⟨*proof*⟩

lift-definition *circline-type* :: *circline* ⇒ *real* **is** *circline-type-clmat*
⟨*proof*⟩

lemma *circline-type*: *circline-type H* = −*1* ∨ *circline-type H* = *0* ∨ *circline-type H* = *1*
⟨*proof*⟩

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*))
⟨*proof*⟩

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*
⟨*proof*⟩

lift-definition *on-circline* :: *circline* ⇒ *complex-homo* ⇒ *bool* **is** *on-circline-clmat-hcoords*
⟨*proof*⟩

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 $z \in$ *circline-set H*
⟨*proof*⟩

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) $(z1, z2) \longleftrightarrow$ *circline-equation A B C D z1 z2*
⟨*proof*⟩

lemma *circline-equation*:
assumes $H =$ *mk-circline A B C D* **and** $(A, B, C, D) \in$ *hermitean-nonzero*
shows *of-complex z* $\in$ *circline-set H* $\longleftrightarrow$ *circline-equation A B C D z 1*
⟨*proof*⟩

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* $\longleftrightarrow$ (*let* $(A, B, C, D) = H$ *in* $A = 0$)
lift-definition *circline-A0-clmat* :: *circline-mat* ⇒ *bool* **is** *circline-A0-cmat*
⟨*proof*⟩
lift-definition *circline-A0* :: *circline* ⇒ *bool* **is** *circline-A0-clmat*
⟨*proof*⟩

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* $\longleftrightarrow$ (*let* $(A, B, C, D) = H$ *in* $D = 0$)
lift-definition *circline-D0-clmat* :: *circline-mat* ⇒ *bool* **is** *circline-D0-cmat*
⟨*proof*⟩
lift-definition *circline-D0* :: *circline* ⇒ *bool* **is** *circline-D0-clmat*
⟨*proof*⟩

lemma *inf-on-circline*: *on-circline H* $\infty_h \longleftrightarrow$ *circline-A0 H*
⟨*proof*⟩

lemma
inf-in-circline-set: $\infty_h \in$ *circline-set H* $\longleftrightarrow$ *is-line H*
⟨*proof*⟩

lemma *zero-on-circline*: *on-circline H* $0_h \longleftrightarrow$ *circline-D0 H*
⟨*proof*⟩

lemma
zero-in-circline-set: $0_h \in$ *circline-set H* $\longleftrightarrow$ *circline-D0 H*
⟨*proof*⟩

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* $(Re\ A)\ B\ (Re\ D)$
⟨*proof*⟩

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*

⟨*proof*⟩

lift-definition *mk-circle* :: *complex* ⇒ *real* ⇒ *circline* **is** *mk-circle-clmat*
⟨*proof*⟩

lemma *is-circle-mk-circle*: *is-circle* (*mk-circle a r*)
⟨*proof*⟩

lemma *circline-set-mk-circle* [*simp*]:
assumes $r \geq 0$
shows *circline-set* (*mk-circle a r*) = *of-complex* ‘ *circle a r*
⟨*proof*⟩

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*
⟨*proof*⟩

lift-definition *mk-line* :: *complex* ⇒ *complex* ⇒ *circline* **is** *mk-line-clmat*
⟨*proof*⟩

lemma *circline-set-mk-line* [*simp*]:
assumes *z1* ≠ *z2*
shows *circline-set* (*mk-line z1 z2*) − {∞_h} = *of-complex* ‘ *line z1 z2*
⟨*proof*⟩

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*
⟨*proof*⟩

lift-definition *euclidean-circle* :: *circline* ⇒ *euclidean-circle* **is** *euclidean-circle-clmat*
⟨*proof*⟩

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*⟩

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*
⟨*proof*⟩

lift-definition *euclidean-line* :: *circline* ⇒ *complex* × *complex* **is** *euclidean-line-clmat*
⟨*proof*⟩

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*⟩

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*
⟨*proof*⟩
lift-definition *unit-circle* :: *circline* **is** *unit-circle-clmat*
⟨*proof*⟩

lemma *on-circline-cmat-cvec-unit*:
shows *on-circline-cmat-cvec unit-circle-cmat (z1, z2) ⟷*
z1 ∗ cnj z1 = z2 ∗ cnj z2
⟨*proof*⟩

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*
⟨*proof*⟩

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*
⟨*proof*⟩

lemma *is-circle-unit-circle* [*simp*]:
shows *is-circle unit-circle*
⟨*proof*⟩

lemma *not-inf-on-unit-circle′* [*simp*]:
shows *¬ on-circline unit-circle ∞h*
⟨*proof*⟩

lemma *not-inf-on-unit-circle″* [*simp*]:
shows *∞h ∉ circline-set unit-circle*
⟨*proof*⟩

lemma *euclidean-circle-unit-circle* [*simp*]:
shows *euclidean-circle unit-circle = (0, 1)*
⟨*proof*⟩

lemma *circline-type-unit-circle* [*simp*]:
shows *circline-type unit-circle = −1*
⟨*proof*⟩

lemma *on-circline-unit-circle* [*simp*]:
shows *on-circline unit-circle (of-complex z) ⟷ cmod z = 1*
⟨*proof*⟩

lemma *circline-set-unit-circle* [*simp*]:
shows *circline-set unit-circle = of-complex ‘ {z. cmod z = 1}*
⟨*proof*⟩

lemma *circline-set-unit-circle-I* [*simp*]:
assumes *cmod z = 1*
shows *of-complex z ∈ circline-set unit-circle*
⟨*proof*⟩

lemma *inversion-unit-circle* [*simp*]:
assumes *on-circline unit-circle x*
shows *inversion x = x*
⟨*proof*⟩

lemma *inversion-id-iff-on-unit-circle*:
shows *inversion a = a ⟷ on-circline unit-circle a*
⟨*proof*⟩

lemma *on-unit-circle-conjugate* [*simp*]:
shows *on-circline unit-circle* (*conjugate z*) ⟷ *on-circline unit-circle z*
⟨*proof*⟩

lemma *conjugate-unit-circle-set* [*simp*]:
shows *conjugate ' (circline-set unit-circle) = circline-set unit-circle*
⟨*proof*⟩

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*
⟨*proof*⟩
lift-definition *x-axis* :: *circline* **is** *x-axis-clmat*
⟨*proof*⟩

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
⟨*proof*⟩

lemma *special-points-on-x-axis″* [*simp*]:
shows $0_h \in$ *circline-set x-axis* **and** $1_h \in$ *circline-set x-axis* **and** $\infty_h \in$ *circline-set x-axis*
⟨*proof*⟩

lemma *is-line-x-axis* [*simp*]:
shows *is-line x-axis*
⟨*proof*⟩

lemma *circline-type-x-axis* [*simp*]:
shows *circline-type x-axis = −1*
⟨*proof*⟩

lemma *on-circline-x-axis*:
shows *on-circline x-axis z* ⟷ (∃ *c*. *is-real c* ∧ *z = of-complex c*) ∨ $z = \infty_h$
⟨*proof*⟩

lemma *on-circline-x-axis-I* [*simp*]:
assumes *is-real z*
shows *on-circline x-axis* (*of-complex z*)
⟨*proof*⟩

lemma *circline-set-x-axis*:
shows *circline-set x-axis = of-complex ' {x. is-real x}* ∪ $\{\infty_h\}$
⟨*proof*⟩

lemma *circline-set-x-axis-I*:
assumes *is-real z*
shows *of-complex z* ∈ *circline-set x-axis*
⟨*proof*⟩

lemma *circline-equation-x-axis*:
shows *of-complex z* ∈ *circline-set x-axis* ⟷ *z = cnj z*
⟨*proof*⟩

Positive and negative part of x-axis

definition *positive-x-axis* **where**

positive-x-axis = $\{z.\ z \in$ *circline-set x-axis* $\wedge\ z \neq \infty_h \wedge$ *Re* (*to-complex z*) $> 0\}$

definition *negative-x-axis* **where**
negative-x-axis = $\{z.\ z \in$ *circline-set x-axis* $\wedge\ z \neq \infty_h \wedge$ *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* $\in$ *positive-x-axis*
⟨*proof*⟩

lemma *circline-set-negative-x-axis-I* [*simp*]:
assumes *is-real z* **and** *Re z* < 0
shows *of-complex z* $\in$ *negative-x-axis*
⟨*proof*⟩

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*
⟨*proof*⟩
lift-definition *y-axis* :: *circline* **is** *y-axis-clmat*
⟨*proof*⟩

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
⟨*proof*⟩

lemma *special-points-on-y-axis″* [*simp*]:
shows $0_h \in$ *circline-set y-axis* **and** $ii_h \in$ *circline-set y-axis* **and** $\infty_h \in$ *circline-set y-axis*
⟨*proof*⟩

lemma *on-circline-y-axis*:
shows *on-circline y-axis z* $\longleftrightarrow$ $(\exists\ c.$ *is-imag c* $\wedge$ *z* = *of-complex c*$) \vee z = \infty_h$
⟨*proof*⟩

lemma *on-circline-y-axis-I* [*simp*]:
assumes *is-imag z*
shows *on-circline y-axis* (*of-complex z*)
⟨*proof*⟩

lemma *circline-set-y-axis*:
shows *circline-set y-axis* = *of-complex* ‘ $\{x.$ *is-imag x*$\} \cup \{\infty_h\}$
⟨*proof*⟩

lemma *circline-set-y-axis-I*:
assumes *is-imag z*
shows *of-complex z* $\in$ *circline-set y-axis*
⟨*proof*⟩

Positive and negative part of y-axis

definition *positive-y-axis* **where**
positive-y-axis = $\{z.\ z \in$ *circline-set y-axis* $\wedge\ z \neq \infty_h \wedge$ *Im* (*to-complex z*) $> 0\}$

definition *negative-y-axis* **where**
negative-y-axis = $\{z.\ z \in$ *circline-set y-axis* $\wedge\ z \neq \infty_h \wedge$ *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* $\in$ *positive-y-axis*
⟨*proof*⟩

lemma *circline-set-negative-y-axis-I* [*simp*]:
assumes *is-imag z* **and** *Im z* < 0
shows *of-complex z* $\in$ *negative-y-axis*
⟨*proof*⟩

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*
⟨*proof*⟩
lift-definition *circline-point-0* :: *circline* **is** *circline-point-0-clmat*
⟨*proof*⟩

lemma *circline-type-circline-point-0* [*simp*]:
shows *circline-type circline-point-0* = *0*
⟨*proof*⟩

lemma *zero-in-circline-point-0* [*simp*]:
shows $0_h \in$ *circline-set circline-point-0*
⟨*proof*⟩

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*
⟨*proof*⟩
lift-definition *imag-unit-circle* :: *circline* **is** *imag-unit-circle-clmat*
⟨*proof*⟩

lemma *circline-type-imag-unit-circle* [*simp*]:
shows *circline-type imag-unit-circle* = *1*
⟨*proof*⟩

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*
⟨*proof*⟩

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 $A*x^2 + A*y^2 + 2*bx*x + 2*by*y + D = 0$
⟨*proof*⟩

lemma *circline-intersection-symetry*:
shows *circline-intersection H1 H2* = *circline-intersection H2 H1*
⟨*proof*⟩

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*
⟨*proof*⟩

lift-definition *moebius-circline* :: *moebius* ⇒ *circline* ⇒ *circline* **is** *moebius-circline-mmat-clmat*
⟨*proof*⟩

lemma *moebius-preserve-circline-type* [*simp*]:
shows *circline-type* (*moebius-circline M H*) = *circline-type H*
⟨*proof*⟩

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*⟩

lemma *on-circline-moebius-circline-I* [*simp*]:
assumes *on-circline H z*
shows *on-circline* (*moebius-circline M H*) (*moebius-pt M z*)
⟨*proof*⟩

lemma *circline-set-moebius-circline* [*simp*]:
shows *circline-set* (*moebius-circline M H*) = *moebius-pt M ` circline-set H*
⟨*proof*⟩

lemma *circline-set-moebius-circline-I* [*simp*]:
assumes $z \in circline\text{-}set\ H$
shows *moebius-pt M z* ∈ *circline-set* (*moebius-circline M H*)
⟨*proof*⟩

lemma *circline-set-moebius-circline-E*:
assumes *moebius-pt M z* ∈ *circline-set* (*moebius-circline M H*)
shows $z \in circline\text{-}set\ H$
⟨*proof*⟩

lemma *circline-set-moebius-circline-iff* [*simp*]:
shows *moebius-pt M z* ∈ *circline-set* (*moebius-circline M H*) ⟷
$z \in circline\text{-}set\ H$
⟨*proof*⟩

lemma *inj-moebius-circline*:
shows *inj* (*moebius-circline M*)
⟨*proof*⟩

lemma *moebius-circline-eq-I*:
assumes *moebius-circline M H1* = *moebius-circline M H2*
shows *H1* = *H2*
⟨*proof*⟩

lemma *moebius-circline-neq-I* [*simp*]:
assumes *H1* ≠ *H2*
shows *moebius-circline M H1* ≠ *moebius-circline M H2*
⟨*proof*⟩

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*
⟨*proof*⟩

lemma *moebius-circline-comp* [*simp*]:
shows *moebius-circline* (*moebius-comp M1 M2*) *H* = *moebius-circline M1* (*moebius-circline M2 H*)
⟨*proof*⟩

lemma *moebius-circline-comp-inv-left* [*simp*]:
shows *moebius-circline* (*moebius-inv M*) (*moebius-circline M H*) = *H*
⟨*proof*⟩

lemma *moebius-circline-comp-inv-right* [*simp*]:
shows *moebius-circline M* (*moebius-circline* (*moebius-inv M*) *H*) = *H*
⟨*proof*⟩

7.8 Action of Euclidean similarities on circlines

lemma *moebius-similarity-lines-to-lines* [*simp*]:
assumes *a* ≠ *0*
shows $\infty_h \in circline\text{-}set\ (moebius\text{-}circline\ (moebius\text{-}similarity\ a\ b)\ H) \longleftrightarrow$

$\infty_h \in$ *circline-set H*
⟨*proof*⟩

lemma *moebius-similarity-lines-to-lines'*:
assumes $a \neq 0$
shows *on-circline* (*moebius-circline* (*moebius-similarity a b*) *H*) $\infty_h \longleftrightarrow$
$\infty_h \in$ *circline-set H*
⟨*proof*⟩

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*
⟨*proof*⟩
lift-definition *conjugate-circline* :: *circline* ⇒ *circline* **is** *conjugate-circline-clmat*
⟨*proof*⟩

lemma *conjugate-circline-set'*:
shows *conjugate* ‘ *circline-set H* ⊆ *circline-set* (*conjugate-circline H*)
⟨*proof*⟩

lemma *conjugate-conjugate-circline* [*simp*]:
shows *conjugate-circline* (*conjugate-circline H*) = *H*
⟨*proof*⟩

lemma *circline-set-conjugate-circline* [*simp*]:
shows *circline-set* (*conjugate-circline H*) = *conjugate* ‘ *circline-set H* (**is** *?lhs* = *?rhs*)
⟨*proof*⟩

lemma *on-circline-conjugate-circline* [*simp*]:
shows *on-circline* (*conjugate-circline H*) *z* ⟷ *on-circline H* (*conjugate z*)
⟨*proof*⟩

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*
⟨*proof*⟩
lift-definition *circline-inversion* :: *circline* ⇒ *circline* **is** *circline-inversion-clmat*
⟨*proof*⟩

lemma *on-circline-circline-inversion* [*simp*]:
shows *on-circline* (*circline-inversion H*) *z* ⟷ *on-circline H* (*reciprocal* (*conjugate z*))
⟨*proof*⟩

lemma *circline-set-circline-inversion* [*simp*]:
shows *circline-set* (*circline-inversion H*) = *inversion* ‘ *circline-set H*
⟨*proof*⟩

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*
⟨*proof*⟩

Rotation of circlines

lemma *rotation-pi-2-y-axis* [*simp*]:
shows *moebius-circline* (*moebius-rotation* (*pi*/2)) *y-axis* = *x-axis*
⟨*proof*⟩

lemma *rotation-minus-pi-2-y-axis* [*simp*]:

shows *moebius-circline* (*moebius-rotation* (−*pi*/2)) *y-axis* = *x-axis*
⟨*proof*⟩

lemma *rotation-minus-pi-2-x-axis* [*simp*]:
shows *moebius-circline* (*moebius-rotation* (−*pi*/2)) *x-axis* = *y-axis*
⟨*proof*⟩

lemma *rotation-pi-2-x-axis* [*simp*]:
shows *moebius-circline* (*moebius-rotation* (*pi*/2)) *x-axis* = *y-axis*
⟨*proof*⟩

lemma *rotation-minus-pi-2-positive-y-axis* [*simp*]:
shows (*moebius-pt* (*moebius-rotation* (−*pi*/2))) ‘ *positive-y-axis* = *positive-x-axis*
⟨*proof*⟩

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*)
⟨*proof*⟩

lemma *unique-circline-type-zero-0*:
shows ∃! *H*. *circline-type H* = 0 ∧ 0_h ∈ *circline-set H*
⟨*proof*⟩

lemma *unique-circline-type-zero*:
shows ∃! *H*. *circline-type H* = 0 ∧ *z* ∈ *circline-set H*
⟨*proof*⟩

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*⟩

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*⟩

lemma *zero-one-inf-x-axis* [*simp*]:
assumes 0_h ∈ *circline-set H* **and** 1_h ∈ *circline-set H* **and** ∞_h ∈ *circline-set H*
shows *H* = *x-axis*
⟨*proof*⟩

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*
⟨*proof*⟩
lift-definition *circline-diag* :: *circline* ⇒ *bool* **is** *is-diag-circline-clmat*
⟨*proof*⟩

lemma *circline-diagonalize*:
shows ∃ *M H′*. *moebius-circline M H* = *H′* ∧ *circline-diag H′*
⟨*proof*⟩

lemma *wlog-circline-diag*:
assumes ⋀ *H*. *circline-diag H* ⟹ *P H*
⋀ *M H*. *P H* ⟹ *P* (*moebius-circline M H*)

shows *P H*
⟨*proof*⟩

7.11.2 Zero type circline set cardinality

lemma *circline-type-zero-card-eq1-0*:
assumes *circline-type H = 0* **and** $0_h \in$ *circline-set H*
shows *circline-set H* = $\{0_h\}$
⟨*proof*⟩

lemma *circline-type-zero-card-eq1*:
assumes *circline-type H = 0*
shows ∃ *z*. *circline-set H* = $\{z\}$
⟨*proof*⟩

7.11.3 Negative type circline set cardinality

lemma *quad-form-diagonal-iff*:
assumes *k1* ≠ *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* ⟷ (∃ *φ*. *z1* = *rcis* (*sqrt* (*Re* (−*k2* /*k1*))) *φ*)
⟨*proof*⟩

lemma *circline-type-neg-card-gt3-diag*:
assumes *circline-type H* < *0* **and** *circline-diag H*
shows ∃ *A B C*. *A* ≠ *B* ∧ *A* ≠ *C* ∧ *B* ≠ *C* ∧ {*A*, *B*, *C*} ⊆ *circline-set H*
⟨*proof*⟩

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*⟩

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* = {}
⟨*proof*⟩

lemma *circline-type-pos-card-eq0*:
assumes *circline-type H* > *0*
shows *circline-set H* = {}
⟨*proof*⟩

7.11.5 Cardinality determines type

lemma *card-eq1-circline-type-zero*:
assumes ∃ *z*. *circline-set H* = $\{z\}$
shows *circline-type H* = *0*
⟨*proof*⟩

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*⟩

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*⟩

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*
⟨*proof*⟩
lift-definition *circline-symmetric* :: *complex-homo* ⇒ *complex-homo* ⇒ *circline* ⇒ *bool* **is** *circline-symmetric-hcoords-clmat*
⟨*proof*⟩

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*)
⟨*proof*⟩

Symmetry wrt. *unit-circle*

lemma *circline-symmetric-0inf-disc* [*simp*]:
shows *circline-symmetric* 0_h ∞_h *unit-circle*
⟨*proof*⟩

lemma *circline-symmetric-inv-homo-disc* [*simp*]:
shows *circline-symmetric a* (*inversion a*) *unit-circle*
⟨*proof*⟩

lemma *circline-symmetric-inv-homo-disc′*:
assumes *circline-symmetric a a′ unit-circle*
shows *a′ = inversion a*
⟨*proof*⟩

lemma *ex-moebius-circline-x-axis*:
assumes *circline-type H < 0*
shows ∃ *M*. *moebius-circline M H = x-axis*
⟨*proof*⟩

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*⟩

lemma *circline-intersection-at-most-2-points*:
assumes *H1 ≠ H2*
shows *finite* (*circline-intersection H1 H2*) ∧ *card* (*circline-intersection H1 H2*) ≤ *2*
⟨*proof*⟩

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*
  ⟨*proof*⟩

lemma *ocircline-eq-cmat-id* [*simp*]:
  shows *ocircline-eq-cmat H H*
  ⟨*proof*⟩

quotient-type *ocircline* = *circline-mat* / *ocircline-eq-clmat*
⟨*proof*⟩

8.2 Points on oriented circlines

Boundary of the circline.

lift-definition *on-ocircline* :: *ocircline* ⇒ *complex-homo* ⇒ *bool* **is** *on-circline-clmat-hcoords*
  ⟨*proof*⟩

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*
  ⟨*proof*⟩

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*
  ⟨*proof*⟩
lift-definition *in-ocircline* :: *ocircline* ⇒ *complex-homo* ⇒ *bool* **is** *in-ocircline-clmat-hcoords*
⟨*proof*⟩

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*
  ⟨*proof*⟩

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*
  ⟨*proof*⟩
lift-definition *out-ocircline* :: *ocircline* ⇒ *complex-homo* ⇒ *bool* **is** *out-ocircline-clmat-hcoords*
⟨*proof*⟩

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*
  ⟨*proof*⟩

lemma *in-on-out*:
  shows *in-ocircline H z* ∨ *on-ocircline H z* ∨ *out-ocircline H z*

⟨*proof*⟩

lemma *in-on-out-univ*:
shows *disc H* ∪ *disc-compl H* ∪ *ocircline-set H* = *UNIV*
⟨*proof*⟩

lemma *disc-inter-disc-compl* [*simp*]:
shows *disc H* ∩ *disc-compl H* = {}
⟨*proof*⟩

lemma *disc-inter-ocircline-set* [*simp*]:
shows *disc H* ∩ *ocircline-set H* = {}
⟨*proof*⟩

lemma *disc-compl-inter-ocircline-set* [*simp*]:
shows *disc-compl H* ∩ *ocircline-set H* = {}
⟨*proof*⟩

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*
⟨*proof*⟩
lift-definition *opposite-ocircline* :: *ocircline* ⇒ *ocircline* **is** *opposite-ocircline-clmat*
⟨*proof*⟩

lemma *opposite-ocircline-involution* [*simp*]:
shows *opposite-ocircline* (*opposite-ocircline H*) = *H*
⟨*proof*⟩

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*
⟨*proof*⟩

lemma *on-circline-opposite-ocircline* [*simp*]:
shows *on-ocircline* (*opposite-ocircline H*) *z* ⟷ *on-ocircline H z*
⟨*proof*⟩

lemma *ocircline-set-opposite-ocircline* [*simp*]:
shows *ocircline-set* (*opposite-ocircline H*) = *ocircline-set H*
⟨*proof*⟩

lemma *disc-compl-opposite-ocircline* [*simp*]:
shows *disc-compl* (*opposite-ocircline H*) = *disc H*
⟨*proof*⟩

lemma *disc-opposite-ocircline* [*simp*]:
shows *disc* (*opposite-ocircline H*) = *disc-compl H*
⟨*proof*⟩

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*
⟨*proof*⟩

lemma *of-ocircline-opposite-ocircline* [*simp*]:
shows *of-ocircline* (*opposite-ocircline H*) = *of-ocircline H*
⟨*proof*⟩

lemma *on-ocircline-of-circline* [*simp*]:
shows *on-circline* (*of-ocircline H*) *z* ⟷ *on-ocircline H z*
⟨*proof*⟩

lemma *circline-set-of-ocircline* [*simp*]:
shows *circline-set* (*of-ocircline H*) = *ocircline-set H*
⟨*proof*⟩

lemma *inj-of-ocircline*:
assumes *of-ocircline H* = *of-ocircline H′*
shows *H* = *H′* ∨ *H* = *opposite-ocircline H′*
⟨*proof*⟩

lemma *inj-ocircline-set*:
assumes *ocircline-set H* = *ocircline-set H′* **and** *ocircline-set H* ≠ {}
shows *H* = *H′* ∨ *H* = *opposite-ocircline H′*
⟨*proof*⟩

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*
⟨*proof*⟩

lift-definition *pos-oriented* :: *ocircline* ⇒ *bool* **is** *pos-oriented-clmat*
⟨*proof*⟩

lemma *pos-oriented*:
shows *pos-oriented H* ∨ *pos-oriented* (*opposite-ocircline H*)
⟨*proof*⟩

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*⟩

lemma *pos-oriented-opposite-ocircline* [*simp*]:
shows *pos-oriented* (*opposite-ocircline H*) ⟷ ¬ *pos-oriented H*
⟨*proof*⟩

lemma *pos-oriented-circle-inf*:
assumes ∞$_h$ ∉ *ocircline-set H*
shows *pos-oriented H* ⟷ ∞$_h$ ∉ *disc H*
⟨*proof*⟩

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*
⟨*proof*⟩

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*
⟨*proof*⟩

lemma *of-circline-clmat-def'*:
shows *of-circline-clmat H* = (*if pos-oriented-clmat H then H else opposite-ocircline-clmat H*)
⟨*proof*⟩

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*⟩

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*⟩

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*
⟨*proof*⟩

lift-definition *of-circline* :: *circline* ⇒ *ocircline* **is** *of-circline-clmat*
⟨*proof*⟩

lemma *pos-oriented-of-circline* [*simp*]:
shows *pos-oriented* (*of-circline H*)
⟨*proof*⟩

lemma *of-ocircline-of-circline* [*simp*]:
shows *of-ocircline* (*of-circline H*) = *H*
⟨*proof*⟩

lemma *of-circline-of-ocircline-pos-oriented* [*simp*]:
assumes *pos-oriented H*
shows *of-circline* (*of-ocircline H*) = *H*
⟨*proof*⟩

lemma *inj-of-circline*:
assumes *of-circline H* = *of-circline H'*
shows *H* = *H'*
⟨*proof*⟩

lemma *of-circline-of-ocircline*:
shows *of-circline* (*of-ocircline H'*) = *H'* ∨
of-circline (*of-ocircline H'*) = *opposite-ocircline H'*
⟨*proof*⟩

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*
⟨*proof*⟩

8.6 Some special oriented circlines and discs

lift-definition *mk-ocircline* :: *complex* ⇒ *complex* ⇒ *complex* ⇒ *complex* ⇒ *ocircline* **is** *mk-circline-clmat*
⟨*proof*⟩

oriented unit circle and unit disc

lift-definition *ounit-circle* :: *ocircline* **is** *unit-circle-clmat*
⟨*proof*⟩

lemma *pos-oriented-ounit-circle* [*simp*]:
shows *pos-oriented ounit-circle*
⟨*proof*⟩

lemma *of-ocircline-ounit-circle* [*simp*]:
shows *of-ocircline ounit-circle* = *unit-circle*
⟨*proof*⟩

lemma *of-circline-unit-circle* [*simp*]:
shows *of-circline* (*unit-circle*) = *ounit-circle*
⟨*proof*⟩

lemma *ocircline-set-ounit-circle* [*simp*]:
shows *ocircline-set ounit-circle* = *circline-set unit-circle*
⟨*proof*⟩

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*
⟨*proof*⟩

lemma *one-notin-unit-dic* [*simp*]:
shows $1_h \notin$ *unit-disc*
⟨*proof*⟩

lemma *inf-notin-unit-disc* [*simp*]:
shows $\infty_h \notin$ *unit-disc*
⟨*proof*⟩

lemma *unit-disc-iff-cmod-lt-1* [*simp*]:
shows *of-complex c* ∈ *unit-disc* ⟷ *cmod c* < *1*
⟨*proof*⟩

lemma *unit-disc-cmod-square-lt-1* [*simp*]:
assumes *z* ∈ *unit-disc*
shows (*cmod* (*to-complex z*))^2 < *1*
⟨*proof*⟩

lemma *unit-disc-to-complex-inj*:
assumes *u* ∈ *unit-disc* **and** *v* ∈ *unit-disc*
assumes *to-complex u* = *to-complex v*
shows *u* = *v*
⟨*proof*⟩

lemma *inversion-unit-disc* [*simp*]:
shows *inversion* ‘ *unit-disc* = *unit-disc-compl*
⟨*proof*⟩

lemma *inversion-unit-disc-compl* [*simp*]:
shows *inversion* ‘ *unit-disc-compl* = *unit-disc*

⟨*proof*⟩

lemma *inversion-noteq-unit-disc*:
assumes $u \in$ *unit-disc* **and** $v \in$ *unit-disc*
shows *inversion* $u \neq v$
⟨*proof*⟩

lemma *in-ocircline-ounit-circle-conjugate* [*simp*]:
assumes *in-ocircline ounit-circle z*
shows *in-ocircline ounit-circle* (*conjugate z*)
⟨*proof*⟩

lemma *conjugate-unit-disc* [*simp*]:
shows *conjugate ‘ unit-disc* = *unit-disc*
⟨*proof*⟩

lemma *conjugate-in-unit-disc* [*simp*]:
assumes $z \in$ *unit-disc*
shows *conjugate z* $\in$ *unit-disc*
⟨*proof*⟩

lemma *out-ocircline-ounit-circle-conjugate* [*simp*]:
assumes *out-ocircline ounit-circle z*
shows *out-ocircline ounit-circle* (*conjugate z*)
⟨*proof*⟩

lemma *conjugate-unit-disc-compl* [*simp*]:
shows *conjugate ‘ unit-disc-compl* = *unit-disc-compl*
⟨*proof*⟩

lemma *conjugate-in-unit-disc-compl* [*simp*]:
assumes $z \in$ *unit-disc-compl*
shows *conjugate z* $\in$ *unit-disc-compl*
⟨*proof*⟩

8.6.1 Oriented x axis and lower half plane

lift-definition *o-x-axis* :: *ocircline* **is** *x-axis-clmat*
⟨*proof*⟩

lemma *o-x-axis-pos-oriented* [*simp*]:
shows *pos-oriented o-x-axis*
⟨*proof*⟩

lemma *of-ocircline-o-x-axis* [*simp*]:
shows *of-ocircline o-x-axis* = *x-axis*
⟨*proof*⟩

lemma *of-circline-x-axis* [*simp*]:
shows *of-circline x-axis* = *o-x-axis*
⟨*proof*⟩

lemma *ocircline-set-circline-set-x-axis* [*simp*]:
shows *ocircline-set o-x-axis* = *circline-set x-axis*
⟨*proof*⟩

lemma *ii-in-disc-o-x-axis* [*simp*]:
shows $ii_h \notin$ *disc o-x-axis*
⟨*proof*⟩

lemma *ii-notin-disc-o-x-axis* [*simp*]:
shows $ii_h \in$ *disc-compl o-x-axis*
⟨*proof*⟩

lemma *of-complex-in-o-x-axis-disc* [*simp*]:
shows *of-complex z* $\in$ *disc o-x-axis* $\longleftrightarrow$ *Im z* $<$ *0*

⟨*proof*⟩

lemma *inf-notin-disc-o-x-axis* [*simp*]:
shows $\infty_h \notin$ *disc o-x-axis*
⟨*proof*⟩

lemma *disc-o-x-axis*:
shows *disc o-x-axis* = *of-complex* ' $\{z.\ Im\ z < 0\}$
⟨*proof*⟩

8.6.2 Oriented single point circline

lift-definition *o-circline-point-0* :: *ocircline* **is** *circline-point-0-clmat*
⟨*proof*⟩

lemma *of-ocircline-o-circline-point-0* [*simp*]:
shows *of-ocircline o-circline-point-0* = *circline-point-0*
⟨*proof*⟩

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*
⟨*proof*⟩

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*))
⟨*proof*⟩

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*)))
⟨*proof*⟩

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*)
⟨*proof*⟩

lemma *moebius-ocircline-id-moebius* [*simp*]:
shows *moebius-ocircline id-moebius H* = *H*
⟨*proof*⟩

lemma *moebius-ocircline-comp* [*simp*]:
shows *moebius-ocircline* (*moebius-comp M1 M2*) *H* = *moebius-ocircline M1* (*moebius-ocircline M2 H*)
⟨*proof*⟩

lemma *moebius-ocircline-comp-inv-left* [*simp*]:
shows *moebius-ocircline* (*moebius-inv M*) (*moebius-ocircline M H*) = *H*
⟨*proof*⟩

lemma *moebius-ocircline-comp-inv-right* [*simp*]:
shows *moebius-ocircline M* (*moebius-ocircline* (*moebius-inv M*) *H*) = *H*
⟨*proof*⟩

lemma *moebius-ocircline-opposite-ocircline* [*simp*]:
shows *moebius-ocircline M* (*opposite-ocircline H*) = *opposite-ocircline* (*moebius-ocircline M H*)
⟨*proof*⟩

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*⟩

lemma *ocircline-set-fix-iff-ocircline-fix*:
assumes *ocircline-set* $H' \neq \{\}$
shows *ocircline-set* (*moebius-ocircline M H*) = *ocircline-set* $H' \longleftrightarrow$
moebius-ocircline M H = $H' \vee$ *moebius-ocircline M H* = *opposite-ocircline* H'
⟨*proof*⟩

lemma *disc-moebius-ocircline* [*simp*]:
shows *disc* (*moebius-ocircline M H*) = *moebius-pt M* ‘ (*disc H*)
⟨*proof*⟩

lemma *disc-compl-moebius-ocircline* [*simp*]:
shows *disc-compl* (*moebius-ocircline M H*) = *moebius-pt M* ‘ (*disc-compl H*)
⟨*proof*⟩

8.8 Orientation after Möbius transformations

All Euclidean similarities preserve circline orientation.

lemma *moebius-similarity-oriented-lines-to-oriented-lines*:
assumes $a \neq 0$
shows $\infty_h \in$ *ocircline-set H* $\longleftrightarrow$ $\infty_h \in$ *ocircline-set* (*moebius-ocircline* (*moebius-similarity a b*) *H*)
⟨*proof*⟩

lemma *moebius-similarity-preserve-orientation*′:
assumes $a \neq 0$ **and** $\infty_h \notin$ *ocircline-set H* **and** *pos-oriented H*
shows *pos-oriented* (*moebius-ocircline* (*moebius-similarity a b*) *H*)
⟨*proof*⟩

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*⟩

lemma *reciprocal-preserve-orientation*:
assumes $0_h \in$ *disc-compl H*
shows *pos-oriented* (*moebius-ocircline moebius-reciprocal H*)
⟨*proof*⟩

lemma *reciprocal-not-preserve-orientation*:
assumes $0_h \in$ *disc H*
shows ¬ *pos-oriented* (*moebius-ocircline moebius-reciprocal H*)
⟨*proof*⟩

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*⟩

lemma *pole-in-disc-compl*:
assumes *M* = *mk-moebius a b c d* **and** $c \neq 0$ **and** $a*d - b*c \neq 0$
assumes *is-pole M z* **and** *z* ∈ *disc-compl H*
shows *pos-oriented* (*moebius-ocircline M H*)
⟨*proof*⟩

8.9 Oriented circlines uniqueness

lemma *ocircline-01inf*:
assumes $0_h \in$ *ocircline-set H* $\wedge$ $1_h \in$ *ocircline-set H* $\wedge$ $\infty_h \in$ *ocircline-set H*
shows *H* = *o-x-axis* $\vee$ *H* = *opposite-ocircline o-x-axis*

⟨*proof*⟩

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*⟩

lemma *unique-ocircline-set*:
assumes $A \neq B$ **and** $A \neq C$ **and** $B \neq C$
shows ∃! *H*. *pos-oriented H* ∧ (*A* ∈ *ocircline-set H* ∧ *B* ∈ *ocircline-set H* ∧ *C* ∈ *ocircline-set H*)
⟨*proof*⟩

lemma *ocircline-set-0h*:
assumes *ocircline-set H* = $\{0_h\}$
shows *H* = *o-circline-point-0* ∨ *H* = *opposite-ocircline* (*o-circline-point-0*)
⟨*proof*⟩

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*
⟨*proof*⟩

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*
⟨*proof*⟩

lemma *mat-det-12-sm-l* [*simp*]:
shows *mat-det-12* (*k* $*_{sm}$ *A*) *B* = *k* ∗ *mat-det-12 A B*
⟨*proof*⟩

lemma *mat-det-12-sm-r* [*simp*]:
shows *mat-det-12 A* (*k* $*_{sm}$ *B*) = *k* ∗ *mat-det-12 A B*
⟨*proof*⟩

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*
⟨*proof*⟩

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*
⟨*proof*⟩

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*⟩

lift-definition *cos-angle* :: *ocircline* ⇒ *ocircline* ⇒ *real* **is** *cos-angle-clmat*
⟨*proof*⟩

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'*
⟨*proof*⟩

lemma *cos-angle-opposite2* [*simp*]:
shows *cos-angle H* (*opposite-ocircline H'*) = − *cos-angle H H'*
⟨*proof*⟩

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*)
⟨*proof*⟩

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*
⟨*proof*⟩

lemma *sgn-sqrt* [*simp*]:
shows *sgn* (*sqrt x*) = *sgn x*
⟨*proof*⟩

lemma *real-circle-sgn-r*:
assumes *is-circle H* **and** (*a*, *r*) = *euclidean-circle H*
shows *sgn r* = − *circline-type H*
⟨*proof*⟩

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*⟩

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 ⟷ *cos-angle* (*of-circline H1*) (*of-circline H2*) = *0*

lemma *perpendicular-sym*:
shows *perpendicular H1 H2* ⟷ *perpendicular H2 H1*
⟨*proof*⟩

8.12 Möbius transforms preserve angles and perpendicularity

Möbius transformations are *conformal* i.e., they preserve angles between circlines.

lemma *moebius-preserve-circline-angle* [*simp*]:
shows *cos-angle* (*moebius-ocircline M H1*) (*moebius-ocircline M H2*) =
cos-angle H1 H2
⟨*proof*⟩

lemma *perpendicular-moebius* [*simp*]:
assumes *perpendicular H1 H2*

shows *perpendicular* (*moebius-circline M H1*) (*moebius-circline M H2*)
⟨*proof*⟩

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*
⟨*proof*⟩

lift-definition *unit-circle-fix* :: *moebius* ⇒ *bool* **is** *unit-circle-fix-mmat*
⟨*proof*⟩

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*⟩

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′*
⟨*proof*⟩

lemma *unit-circle-fix-iff-unit-circle-set*:
shows *unit-circle-fix M* ⟷ *moebius-pt M ‘ unit-circle-set* = *unit-circle-set*
⟨*proof*⟩

Unit circle preserving Möbius transformations form a group.

lemma *unit-circle-fix-id-moebius* [*simp*]:
shows *unit-circle-fix id-moebius*
⟨*proof*⟩

lemma *unit-circle-fix-moebius-add* [*simp*]:
assumes *unit-circle-fix M1* **and** *unit-circle-fix M2*
shows *unit-circle-fix* (*M1* + *M2*)
⟨*proof*⟩

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*)
⟨*proof*⟩

lemma *unit-circle-fix-moebius-uminus* [*simp*]:
assumes *unit-circle-fix M*
shows *unit-circle-fix* (−*M*)
⟨*proof*⟩

lemma *unit-circle-fix-moebius-inv* [*simp*]:
assumes *unit-circle-fix M*
shows *unit-circle-fix* (*moebius-inv M*)

⟨*proof*⟩

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*)
⟨*proof*⟩

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*⟩

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*
⟨*proof*⟩

lift-definition *unit-disc-fix* :: *moebius* ⇒ *bool* **is** *unit-disc-fix-mmat*
⟨*proof*⟩

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*
⟨*proof*⟩

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*⟩

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*
⟨*proof*⟩

lemma *unit-disc-fix-discI* [*simp*]:
assumes *unit-disc-fix M* **and** *u* ∈ *unit-disc*
shows *moebius-pt M u* ∈ *unit-disc*
⟨*proof*⟩

Unit disc preserving transformations form a group.

lemma *unit-disc-fix-id-moebius* [*simp*]:
shows *unit-disc-fix id-moebius*
⟨*proof*⟩

lemma *unit-disc-fix-moebius-add* [*simp*]:

assumes *unit-disc-fix M1* **and** *unit-disc-fix M2*
shows *unit-disc-fix* (*M1* + *M2*)
⟨*proof*⟩

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*)
⟨*proof*⟩

lemma *unit-disc-fix-moebius-uminus* [*simp*]:
assumes *unit-disc-fix M*
shows *unit-disc-fix* (−*M*)
⟨*proof*⟩

lemma *unit-disc-fix-moebius-inv* [*simp*]:
assumes *unit-disc-fix M*
shows *unit-disc-fix* (*moebius-inv M*)
⟨*proof*⟩

9.4 Rotations are unit disc preserving transformations

lemma *unit-disc-fix-rotation* [*simp*]:
shows *unit-disc-fix* (*moebius-rotation* φ)
⟨*proof*⟩

lemma *moebius-rotation-unit-circle-fix* [*simp*]:
shows *moebius-pt* (*moebius-rotation* φ) $u \in$ *unit-circle-set* $\longleftrightarrow$ $u \in$ *unit-circle-set*
⟨*proof*⟩

lemma *ex-rotation-mapping-u-to-positive-x-axis*:
assumes $u \neq 0_h$ **and** $u \neq \infty_h$
shows $\exists$ φ. *moebius-pt* (*moebius-rotation* φ) $u \in$ *positive-x-axis*
⟨*proof*⟩

lemma *ex-rotation-mapping-u-to-positive-y-axis*:
assumes $u \neq 0_h$ **and** $u \neq \infty_h$
shows $\exists$ φ. *moebius-pt* (*moebius-rotation* φ) $u \in$ *positive-y-axis*
⟨*proof*⟩

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. $[\![u \in$ *unit-disc*; $u \neq 0_h$; P (*moebius-pt* (*moebius-rotation* φ) $u)]\!] \Longrightarrow P\ u$
assumes *x-axis*: $\bigwedge x$. $[\![$*is-real* x; $0 <$ *Re* x; *Re* $x < 1]\!] \Longrightarrow P$ (*of-complex* x)
shows $P\ u$
⟨*proof*⟩

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 (*moebius-pt* (*moebius-rotation* φ) $u)]\!] \Longrightarrow P\ u$
assumes *x-axis*: $\bigwedge x$. $[\![$*is-real* x; $0 <$ *Re* $x]\!] \Longrightarrow P$ (*of-complex* x)
shows $P\ u$
⟨*proof*⟩

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*⟩

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*
⟨*proof*⟩
lift-definition *blaschke* :: *complex* ⇒ *moebius* **is** *blaschke-mmat*
⟨*proof*⟩

lemma *blaschke-0-id* [*simp*]: *blaschke 0* = *id-moebius*
⟨*proof*⟩

lemma *blaschke-a-to-zero* [*simp*]:
assumes *cmod a* ≠ *1*
shows *moebius-pt* (*blaschke a*) (*of-complex a*) = 0_h
⟨*proof*⟩

lemma *blaschke-inv-a-inf* [*simp*]:
assumes *cmod a* ≠ *1*
shows *moebius-pt* (*blaschke a*) (*inversion* (*of-complex a*)) = ∞_h
⟨*proof*⟩

lemma *blaschke-inf* [*simp*]:
assumes *cmod a* < *1* **and** *a* ≠ *0*
shows *moebius-pt* (*blaschke a*) ∞_h = *of-complex* (− *1* / *cnj a*)
⟨*proof*⟩

lemma *blaschke-0-minus-a* [*simp*]:
assumes *cmod a* ≠ *1*
shows *moebius-pt* (*blaschke a*) 0_h = $^{\sim}{}_h$ (*of-complex a*)
⟨*proof*⟩

lemma *blaschke-unit-circle-fix* [*simp*]:
assumes *cmod a* ≠ *1*
shows *unit-circle-fix* (*blaschke a*)
⟨*proof*⟩

lemma *blaschke-unit-disc-fix* [*simp*]:
assumes *cmod a* < *1*
shows *unit-disc-fix* (*blaschke a*)
⟨*proof*⟩

lemma *blaschke-unit-circle-fix′*:
assumes *cmod a* ≠ *1*
shows *moebius-circline* (*blaschke a*) *unit-circle* = *unit-circle*
⟨*proof*⟩

lemma *blaschke-ounit-circle-fix′*:
assumes *cmod a* < *1*
shows *moebius-ocircline* (*blaschke a*) *ounit-circle* = *ounit-circle*
⟨*proof*⟩

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*))
⟨*proof*⟩

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*⟩

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*⟩

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*⟩

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*
⟨*proof*⟩
lift-definition *inv-blaschke* :: *complex* ⇒ *moebius* **is** *inv-blaschke-mmat*
⟨*proof*⟩

lemma *inv-blaschke-neg* [*simp*]: *inv-blaschke a* = *blaschke* (−*a*)
⟨*proof*⟩

lemma *inv-blaschke*:
assumes *cmod a* ≠ *1*
shows *blaschke a* + *inv-blaschke a* = *0*
⟨*proof*⟩

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*⟩

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*⟩

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*⟩

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*⟩

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*
⟨*proof*⟩

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*
⟨*proof*⟩

lemma *ex-unit-disc-fix-to-zero-positive-x-axis*:

assumes $u \in$ *unit-disc* **and** $v \in$ *unit-disc* **and** $u \neq v$
shows $\exists\ M.$ *unit-disc-fix* $M \wedge$
moebius-pt $M\ u = 0_h \wedge$ *moebius-pt* $M\ v \in$ *positive-x-axis*
⟨*proof*⟩

lemma *wlog-x-axis*:
assumes *in-disc*: $u \in$ *unit-disc* $v \in$ *unit-disc*
assumes *preserved*: $\bigwedge M\ u\ v.$ ⟦*unit-disc-fix* M; $u \in$ *unit-disc*; $v \in$ *unit-disc*; P (*moebius-pt* $M\ u$) (*moebius-pt* $M\ v$)⟧ $\Longrightarrow P\ u\ v$
assumes *axis*: $\bigwedge x.$ ⟦*is-real* x; $0 \leq$ *Re* x; *Re* $x < 1$⟧ $\Longrightarrow P\ 0_h$ (*of-complex* x)
shows $P\ u\ v$
⟨*proof*⟩

lemma *wlog-positive-x-axis*:
assumes *in-disc*: $u \in$ *unit-disc* $v \in$ *unit-disc* $u \neq v$
assumes *preserved*: $\bigwedge M\ u\ v.$ ⟦*unit-disc-fix* M; $u \in$ *unit-disc*; $v \in$ *unit-disc*; $u \neq v$; P (*moebius-pt* $M\ u$) (*moebius-pt* $M\ v$)⟧ $\Longrightarrow P\ u\ v$
assumes *axis*: $\bigwedge x.$ ⟦*is-real* x; $0 <$ *Re* x; *Re* $x < 1$⟧ $\Longrightarrow P\ 0_h$ (*of-complex* x)
shows $P\ u\ v$
⟨*proof*⟩

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*
⟨*proof*⟩

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)
⟨*proof*⟩

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*⟩

lemma *unit-disc-fix-conjugate-comp-moebius* [*simp*]:
assumes *unit-disc-fix* M
shows *unit-disc-fix-f* (*conjugate* $\circ$ *moebius-pt* M)
⟨*proof*⟩

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* $\circ$ *f2*)
⟨*proof*⟩

lemma *unit-disc-fix-f-inv*:
assumes *unit-disc-fix-f* M
shows *unit-disc-fix-f* (*inv* M)
⟨*proof*⟩

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*⟩

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*⟩

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*⟩

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*
⟨*proof*⟩

setup-lifting *type-definition-riemann-sphere*

lemma *sphere-bounds′*:
assumes *x*∗*x* + *y*∗*y* + *z*∗*z* = (*1*::*real*)
shows −*1* ≤ *x* ∧ *x* ≤ *1*
⟨*proof*⟩

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*
⟨*proof*⟩

10.1 Parametrization of the unit sphere in polar coordinates

lemma *sphere-params-on-sphere*:
fixes α β :: *real*
assumes $x = cos\ \alpha * cos\ \beta$ **and** $y = cos\ \alpha * sin\ \beta$ $z = sin\ \alpha$
shows $x*x + y*y + z*z = 1$
⟨*proof*⟩

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*⟩

lemma *ex-sphere-params*:
assumes $x*x + y*y + z*z = 1$
shows $\exists\ \alpha\ \beta.\ x = cos\ \alpha * cos\ \beta \wedge y = cos\ \alpha * sin\ \beta \wedge z = sin\ \alpha \wedge -pi\ /\ 2 \leq \alpha \wedge \alpha \leq pi\ /\ 2 \wedge -pi \leq \beta \wedge \beta < pi$
⟨*proof*⟩

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*) ≠ (*0*, *0*, *1*) *then*
(*x* + i ∗ *y*, *cor* (*1* − *z*))
else
(*1*, *0*)
))

lift-definition *stereographic-r3-hcoords* :: *R3* ⇒ *complex-homo-coords* **is** *stereographic-r3-cvec*
⟨*proof*⟩

lift-definition *stereographic* :: *riemann-sphere* ⇒ *complex-homo* **is** *stereographic-r3-hcoords*
⟨*proof*⟩

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)
⟨*proof*⟩

lemma *Im-stereographic*:
shows *Im* (*2* ∗ *z* / (*1* + *z* ∗ *cnj z*)) = *2* ∗ *Im z* / (*1* + (*cmod z*)^2)
⟨*proof*⟩

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*⟩

lift-definition *inv-stereographic-hcoords-r3* :: *complex-homo-coords* ⇒ *R3* **is** *inv-stereographic-cvec-r3*
⟨*proof*⟩

lift-definition *inv-stereographic* :: *complex-homo* ⇒ *riemann-sphere* **is** *inv-stereographic-hcoords-r3*
⟨*proof*⟩

North pole

definition *North-R3* :: *R3* **where**
[*simp*]: *North-R3* = (*0*, *0*, *1*)
lift-definition *North* :: *riemann-sphere* **is** *North-R3*
⟨*proof*⟩

lemma *stereographic-North*:
shows *stereographic x* = ∞$_h$ ⟷ *x* = *North*
⟨*proof*⟩

Stereographic and inverse stereographic projection are mutually inverse.

lemma *stereographic-inv-stereographic′*:
assumes
z: *z* = *z1*/*z2* **and** *z2* ≠ *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 ∃ *k*. *k* ≠ *0* ∧ (*X* + i∗*Y*, *complex-of-real* (*1* − *Z*)) = *k* ∗$_{sv}$ (*z1*, *z2*)
⟨*proof*⟩

lemma *stereographic-inv-stereographic* [*simp*]:
shows *stereographic* (*inv-stereographic w*) = *w*
⟨*proof*⟩

Stereographic projection is bijective function

lemma *bij-stereographic*:
shows *bij stereographic*
⟨*proof*⟩

lemma *inv-stereographic-stereographic* [*simp*]:
shows *inv-stereographic* (*stereographic x*) = *x*
⟨*proof*⟩

lemma *inv-stereographic-is-inv*:
shows *inv-stereographic* = *inv stereographic*
⟨*proof*⟩

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*
⟨*proof*⟩

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*
⟨*proof*⟩

lemma *mult-sv-one* [*simp*]:
shows *1* ∗$_{sv4}$ *x* = *x*
⟨*proof*⟩

lemma *mult-sv-distb* [*simp*]:
shows *x* ∗$_{sv4}$ (*y* ∗$_{sv4}$ *v*) = (*x*∗*y*) ∗$_{sv4}$ *v*
⟨*proof*⟩

quotient-type *plane* = *plane-vec* / *plane-vec-eq*
⟨*proof*⟩

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*
⟨*proof*⟩

lift-definition *on-sphere-circle* :: *plane* ⇒ *riemann-sphere* ⇒ *bool* **is** *on-sphere-circle-vec*
⟨*proof*⟩

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*
⟨*proof*⟩

lift-definition *inv-stereographic-circline* :: *circline* ⇒ *plane* **is** *inv-stereographic-circline-clmat-pv*
⟨*proof*⟩

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*
⟨*proof*⟩

lift-definition *stereographic-circline* :: *plane* ⇒ *circline* **is** *stereographic-circline-pv-clmat*
⟨*proof*⟩

Stereographic and inverse stereographic projection of circlines are mutually inverse.

lemma *stereographic-circline-inv-stereographic-circline*:
shows *stereographic-circline* ∘ *inv-stereographic-circline* = *id*

⟨*proof*⟩

Stereographic and inverse stereographic projection of circlines are mutually inverse.

lemma *inv-stereographic-circline-stereographic-circline*:
inv-stereographic-circline ∘ *stereographic-circline* = *id*
⟨*proof*⟩

lemma *stereographic-sphere-circle-set''*:
shows *on-sphere-circle* (*inv-stereographic-circline H*) *z* ⟷
on-circline H (*stereographic z*)
⟨*proof*⟩

lemma *stereographic-sphere-circle-set'* [*simp*]:
shows *stereographic* ‘ *sphere-circle-set* (*inv-stereographic-circline H*) =
circline-set H
⟨*proof*⟩

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*)
⟨*proof*⟩

Stereographic projection of circlines is bijective.

lemma *bij-stereographic-circline*:
shows *bij stereographic-circline*
⟨*proof*⟩

Inverse stereographic projection is bijective.

lemma *bij-inv-stereographic-circline*:
shows *bij inv-stereographic-circline*
⟨*proof*⟩

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* (⟨-,-⟩)
translations
⟨*z*,*w*⟩ == *CONST inprod-cvec z w*

lemma *real-inprod-cvec* [*simp*]:
shows *is-real* ⟨*z*,*z*⟩
⟨*proof*⟩

lemma *inprod-cvec-ge-zero* [*simp*]:
shows *Re* ⟨*z*,*z*⟩ ≥ *0*
⟨*proof*⟩

lemma *inprod-cvec-bilinear1* [*simp*]:
assumes $z' = k *_{sv} z$
shows $\langle z',w\rangle = cnj\ k * \langle z,w\rangle$
⟨*proof*⟩

lemma *inprod-cvec-bilinear2* [*simp*]:
assumes $z' = k *_{sv} z$
shows $\langle w, z'\rangle = k * \langle w, z\rangle$
⟨*proof*⟩

lemma *inprod-cvec-g-zero* [*simp*]:
assumes $z \neq vec\text{-}zero$
shows $Re\ \langle z, z\rangle > 0$
⟨*proof*⟩

definition *norm-cvec* :: *complex-vec* ⇒ *real* **where**
[*simp*]: $norm\text{-}cvec\ z = sqrt\ (Re\ \langle z,z\rangle)$
syntax
-norm-cvec :: *complex-vec* ⇒ *complex* (⟨-⟩)
translations
$\langle z\rangle == CONST\ norm\text{-}cvec\ z$

lemma *norm-cvec-square*:
shows $\langle z\rangle^2 = Re\ (\langle z,z\rangle)$
⟨*proof*⟩

lemma *norm-cvec-gt-0*:
assumes $z \neq vec\text{-}zero$
shows $\langle z\rangle > 0$
⟨*proof*⟩

lemma *norm-cvec-scale*:
assumes $z' = k *_{sv} z$
shows $\langle z'\rangle^2 = Re\ (cnj\ k * k) * \langle z\rangle^2$
⟨*proof*⟩

lift-definition *inprod-hcoords* :: *complex-homo-coords* ⇒ *complex-homo-coords* ⇒ *complex* **is** *inprod-cvec*
⟨*proof*⟩

lift-definition *norm-hcoords* :: *complex-homo-coords* ⇒ *real* **is** *norm-cvec*
⟨*proof*⟩

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 $z \neq vec\text{-}zero$ **and** $w \neq vec\text{-}zero$
shows $dist\text{-}fs\text{-}cvec\ z\ w = 2*sqrt(1 - (cmod\ \langle z,w\rangle)^2\ /\ (\langle z\rangle^2 * \langle w\rangle^2))$
⟨*proof*⟩

lift-definition *dist-fs-hcoords* :: *complex-homo-coords* ⇒ *complex-homo-coords* ⇒ *real* **is** *dist-fs-cvec*
⟨*proof*⟩

lift-definition *dist-fs* :: *complex-homo* ⇒ *complex-homo* ⇒ *real* **is** *dist-fs-hcoords*
⟨*proof*⟩

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))
⟨*proof*⟩

lemma *dist-fs-infinite1*:
shows *dist-fs* (*of-complex z1*) ∞_h = *2* / *sqrt* (*1*+(*cmod z1*)^2)
⟨*proof*⟩

lemma *dist-fs-infinite2*:
shows *dist-fs* ∞_h (*of-complex z1*) = *2* / *sqrt* (*1*+(*cmod z1*)^2)
⟨*proof*⟩

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})
⟨*proof*⟩

lemma *dist-fs-zero1* [*simp*]:
shows *dist-fs z z* = *0*
⟨*proof*⟩

lemma *dist-fs-zero2* [*simp*]:
assumes *dist-fs z1 z2* = *0*
shows *z1* = *z2*
⟨*proof*⟩

lemma *dist-fs-sym*:
shows *dist-fs z1 z2* = *dist-fs z2 z1*
⟨*proof*⟩

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*⟩

lemma *dist-fs-triangle-infinite1*:
shows *1* / *sqrt*(*1* + (*cmod b*)^2) ≤ *1* / *sqrt*(*1* + (*cmod c*)^2) + *cmod* (*b* − *c*) / (*sqrt*(*1* + (*cmod b*)^2) ∗ *sqrt*(*1* + (*cmod c*)^2))
⟨*proof*⟩

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)
⟨*proof*⟩

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*⟩

lemma *dist-fs-triangle*:
shows *dist-fs A B* ≤ *dist-fs A C* + *dist-fs C B*
⟨*proof*⟩

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*⟩
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*
⟨*proof*⟩

lift-definition *dist-riemann-sphere′* :: *riemann-sphere* ⇒ *riemann-sphere* ⇒ *real* **is** *dist-riemann-sphere-r3*
⟨*proof*⟩

lemma *dist-riemann-sphere-ge-0* [*simp*]:
shows *dist-riemann-sphere′ M1 M2* ≥ *0*
⟨*proof*⟩

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))
⟨*proof*⟩

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*⟩

lemma *dist-rieman-sphere-zero* [*simp*]:
shows *dist-riemann-sphere′ M M* = *0*
⟨*proof*⟩

lemma *dist-riemann-sphere-sym*:
shows *dist-riemann-sphere′ M1 M2* = *dist-riemann-sphere′ M2 M1*
⟨*proof*⟩

Central theorem that connects the two metrics.

lemma *dist-stereographic*:
shows *dist-riemann-sphere′ M1 M2* = *dist-fs* (*stereographic M1*) (*stereographic M2*)
⟨*proof*⟩

Other direction

lemma *dist-stereographic′*:
shows *dist-fs A B* = *dist-riemann-sphere′* (*inv-stereographic A*) (*inv-stereographic B*)
⟨*proof*⟩

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*⟩
end

The *riemann-sphere* metric space is perfect, i.e., it does not have isolated points.

instantiation *riemann-sphere* :: *perfect-space*
begin
instance ⟨*proof*⟩
end

The *complex-homo* metric space is perfect, i.e., it does not have isolated points.

instantiation *complex-homo* :: *perfect-space*
begin
instance ⟨*proof*⟩

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*)
⟨*proof*⟩

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*)
⟨*proof*⟩

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*
⟨*proof*⟩

lemma *continuous-on UNIV inv-stereographic*
⟨*proof*⟩

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*
⟨*proof*⟩

lift-definition *chordal-circle-hcoords-clmat* :: *complex-homo-coords* ⇒ *real* ⇒ *circline-mat* **is** *chordal-circle-cvec-cmat*
⟨*proof*⟩

lift-definition *chordal-circle* :: *complex-homo* ⇒ *real* ⇒ *circline* **is** *chordal-circle-hcoords-clmat*
⟨*proof*⟩

lemma *sqrt-1-plus-square*:
shows *sqrt* (1 + *a*²) ≠ 0
⟨*proof*⟩

lemma
assumes *dist-fs z a* = *r*
shows *z* ∈ *circline-set* (*chordal-circle a r*)
⟨*proof*⟩

lemma
assumes *z* ∈ *circline-set* (*chordal-circle a r*) **and** *r* ≥ *0*
shows *dist-fs z a* = *r*
⟨*proof*⟩

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*
⟨*proof*⟩

lift-definition *chordal-circles* :: *ocircline* ⇒ (*complex* × *real*) × (*complex* × *real*) **is** *chordal-circles-clmat*
⟨*proof*⟩

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*⟩

lemma *chordal-circle-det-positive*:
fixes *x y* :: *real*
assumes *x* ∗ *y* < *0*
shows *x* / (*x* − *y*) > *0*
⟨*proof*⟩

lemma *cor-sqrt-squared*: *x* ≥ *0* ⟹ (*cor* (*sqrt x*))² = *cor x*
⟨*proof*⟩

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*
⟨*proof*⟩

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*
⟨*proof*⟩

lemma *chordal-circle′*:
assumes *B* ≠ *0* **and** (*A*, *B*, *C*, *D*) ∈ *hermitean-nonzero* **and** *Re* (*mat-det* (*A*, *B*, *C*, *D*)) ≤ *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*
⟨*proof*⟩

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.