The independence of Tarski's Euclidean axiom

T. J. M. Makarios

April 20, 2020

Abstract

Tarski's axioms of plane geometry are formalized and, using the standard real Cartesian model, shown to be consistent. A substantial theory of the projective plane is developed. Building on this theory, the Klein–Beltrami model of the hyperbolic plane is defined and shown to satisfy all of Tarski's axioms except his Euclidean axiom; thus Tarski's Euclidean axiom is shown to be independent of his other axioms of plane geometry.

An earlier version of this work was the subject of the author's MSc thesis [2], which contains natural-language explanations of some of the more interesting proofs.

Contents

1 Metric and semimetric spaces

theory *Metric*
imports *HOL−Analysis.Multivariate-Analysis*
begin

locale *semimetric* =
  fixes *dist* :: $'p \Rightarrow 'p \Rightarrow real$
  assumes *nonneg* [*simp*]: $dist\ x\ y \geq 0$
  and *eq-0* [*simp*]: $dist\ x\ y = 0 \longleftrightarrow x = y$
  and *symm*: $dist\ x\ y = dist\ y\ x$
begin

```
  lemma refl [simp]: dist x x = 0
    by simp
end

locale metric =
  fixes dist :: 'p ⇒ 'p ⇒ real
  assumes [simp]: dist x y = 0 ⟷ x = y
  and triangle: dist x z ≤ dist y x + dist y z

sublocale metric < semimetric
proof
  { fix w
    have dist w w = 0 by simp }
  note [simp] = this
  fix x y
  show 0 ≤ dist x y
  proof −
    from triangle [of y y x] show 0 ≤ dist x y by simp
  qed
  show dist x y = 0 ⟷ x = y by simp
  show dist x y = dist y x
  proof −
    { fix w z
      have dist w z ≤ dist z w
      proof −
        from triangle [of w z z] show dist w z ≤ dist z w by simp
      qed }
    hence dist x y ≤ dist y x and dist y x ≤ dist x y by simp+
    thus dist x y = dist y x by simp
  qed
qed

definition norm-dist :: ('a::real-normed-vector) ⇒ 'a ⇒ real where
[simp]: norm-dist x y ≜ norm (x − y)

interpretation norm-metric: metric norm-dist
proof
  fix x y
  show norm-dist x y = 0 ⟷ x = y by simp
  fix z
  from norm-triangle-ineq [of x − y y − z] have
    norm (x − z) ≤ norm (x − y) + norm (y − z) by simp
  with norm-minus-commute [of x y] show
    norm-dist x z ≤ norm-dist y x + norm-dist y z by simp
qed

end
```

2 Miscellaneous results

theory *Miscellany*
imports *Metric*
begin

lemma *unordered-pair-element-equality*:
  assumes $\{p, q\} = \{r, s\}$ **and** $p = r$
  shows $q = s$
  using *assms* **by** (*auto simp*: *doubleton-eq-iff*)

lemma *unordered-pair-equality*: $\{p, q\} = \{q, p\}$
  by *auto*

lemma *cosine-rule*:
  fixes $a\ b\ c$ :: *real* ^ ('*n*::*finite*)
  shows $(\textit{norm-dist}\ a\ c)^2 =$
  $(\textit{norm-dist}\ a\ b)^2 + (\textit{norm-dist}\ b\ c)^2 + 2 * ((a - b) \cdot (b - c))$
proof −
  have $(a - b) + (b - c) = a - c$ **by** *simp*
  with *dot-norm* [*of* $a - b\ b - c$]
    have $(a - b) \cdot (b - c) =$
      $((\textit{norm}\ (a - c))^2 - (\textit{norm}\ (a - b))^2 - (\textit{norm}\ (b - c))^2) / 2$
      by *simp*
  thus *?thesis* **by** *simp*
qed

lemma *scalar-equiv*: $r *s\ x = r *_R x$
  by *vector*

lemma *norm-dist-dot*: $(\textit{norm-dist}\ x\ y)^2 = (x - y) \cdot (x - y)$
  by (*simp add*: *power2-norm-eq-inner*)

definition *dep2* :: '*a*::*real-vector* ⇒ '*a* ⇒ *bool* **where**
  $\textit{dep2}\ u\ v \triangleq \exists\, w\ r\ s.\ u = r *_R w \wedge v = s *_R w$

lemma *real2-eq*:
  fixes $u\ v$ :: *real*^*2*
  assumes $u\$1 = v\$1$ **and** $u\$2 = v\$2$
  shows $u = v$
  by (*simp add*: *vec-eq-iff* [*of u v*] *forall-2 assms*)

definition *rotate2* :: *real*^*2* ⇒ *real*^*2* **where**
  $\textit{rotate2}\ x \triangleq \textit{vector}\ [-x\$2, x\$1]$

declare *vector-2* [*simp*]

lemma *rotate2* [*simp*]:
  $(\textit{rotate2}\ x)\$1 = -x\$2$

```
  (rotate2 x)$2 = x$1
  by (simp add: rotate2-def)+

lemma rotate2-rotate2 [simp]: rotate2 (rotate2 x) = −x
proof −
  have (rotate2 (rotate2 x))$1 = −x$1 and (rotate2 (rotate2 x))$2 = −x$2
    by simp+
  with real2-eq show rotate2 (rotate2 x) = −x by simp
qed

lemma rotate2-dot [simp]: (rotate2 u) · (rotate2 v) = u · v
  unfolding inner-vec-def
  by (simp add: sum-2)

lemma rotate2-scaleR [simp]: rotate2 (k *R x) = k *R (rotate2 x)
proof −
  have (rotate2 (k *R x))$1 = (k *R (rotate2 x))$1 and
    (rotate2 (k *R x))$2 = (k *R (rotate2 x))$2 by simp+
  with real2-eq show ?thesis by simp
qed

lemma rotate2-uminus [simp]: rotate2 (−x) = −(rotate2 x)
proof −
  from scaleR-minus-left [of 1] have
    −1 *R x = −x and −1 *R (rotate2 x) = −(rotate2 x) by auto
  with rotate2-scaleR [of −1 x] show ?thesis by simp
qed

lemma rotate2-eq [iff]: rotate2 x = rotate2 y ⟷ x = y
proof
  assume x = y
  thus rotate2 x = rotate2 y by simp
next
  assume rotate2 x = rotate2 y
  hence rotate2 (rotate2 x) = rotate2 (rotate2 y) by simp
  hence −(−x) = −(−y) by simp
  thus x = y by simp
qed

lemma dot2-rearrange-1:
  fixes u x :: real^2
  assumes u · x = 0 and x$1 ≠ 0
  shows u = (u$2 / x$1) *R (rotate2 x) (is u = ?u')
proof −
  from ‹u · x = 0› have u$1 * x$1 = −(u$2) * (x$2)
    unfolding inner-vec-def
    by (simp add: sum-2)
  hence u$1 * x$1 / x$1 = −u$2 / x$1 * x$2 by simp
  with ‹x$1 ≠ 0› have u$1 = ?u'$1 by simp
```

from ‹$x\$1 \neq 0$› **have** $u\$2 = ?u'\$2$ **by** *simp*
with ‹$u\$1 = ?u'\$1$› **and** *real2-eq* **show** $u = ?u'$ **by** *simp*
qed

lemma *dot2-rearrange-2*:
fixes *u x* :: *real^2*
assumes $u \cdot x = 0$ **and** $x\$2 \neq 0$
shows $u = -(u\$1 / x\$2) *_R (rotate2\ x)$ (**is** $u = ?u'$)
proof −
from *assms* **and** *dot2-rearrange-1* [*of rotate2 u rotate2 x*] **have**
rotate2 u = *rotate2 ?u'* **by** *simp*
thus $u = ?u'$ **by** *blast*
qed

lemma *dot2-rearrange*:
fixes *u x* :: *real^2*
assumes $u \cdot x = 0$ **and** $x \neq 0$
shows $\exists k.\ u = k *_R (rotate2\ x)$
proof *cases*
assume $x\$1 = 0$
with *real2-eq* [*of x 0*] **and** ‹$x \neq 0$› **have** $x\$2 \neq 0$ **by** *auto*
with *dot2-rearrange-2* **and** ‹$u \cdot x = 0$› **show** *?thesis* **by** *blast*
next
assume $x\$1 \neq 0$
with *dot2-rearrange-1* **and** ‹$u \cdot x = 0$› **show** *?thesis* **by** *blast*
qed

lemma *real2-orthogonal-dep2*:
fixes *u v x* :: *real^2*
assumes $x \neq 0$ **and** $u \cdot x = 0$ **and** $v \cdot x = 0$
shows *dep2 u v*
proof −
let *?w* = *rotate2 x*
from *dot2-rearrange* **and** *assms* **have**
$\exists r\ s.\ u = r *_R ?w \wedge v = s *_R ?w$ **by** *simp*
with *dep2-def* **show** *?thesis* **by** *auto*
qed

lemma *dot-left-diff-distrib*:
fixes *u v x* :: *real^'n*
shows $(u - v) \cdot x = (u \cdot x) - (v \cdot x)$
proof −
have $(u \cdot x) - (v \cdot x) = (\sum i{\in}UNIV.\ u\$i * x\$i) - (\sum i{\in}UNIV.\ v\$i * x\$i)$
unfolding *inner-vec-def*
by *simp*
also from *sum-subtractf* [*of* $\lambda\ i.\ u\$i * x\$i\ \lambda\ i.\ v\$i * x\$i$] **have**
$\ldots = (\sum i{\in}UNIV.\ u\$i * x\$i - v\$i * x\$i)$ **by** *simp*
also from *left-diff-distrib* [**where** *'a* = *real*] **have**
$\ldots = (\sum i{\in}UNIV.\ (u\$i - v\$i) * x\$i)$ **by** *simp*

also have
$\ldots = (u - v) \cdot x$
unfolding *inner-vec-def*
by *simp*
finally show *?thesis* ..
qed

lemma *dot-right-diff-distrib*:
fixes *u v x* :: *real^'n*
shows $x \cdot (u - v) = (x \cdot u) - (x \cdot v)$
proof –
from *inner-commute* have $x \cdot (u - v) = (u - v) \cdot x$ by *auto*
also from *dot-left-diff-distrib* [*of u v x*] have
$\ldots = u \cdot x - v \cdot x$.
also from *inner-commute* [*of x*] have
$\ldots = x \cdot u - x \cdot v$ by *simp*
finally show *?thesis* .
qed

lemma *am-gm2*:
fixes *a b* :: *real*
assumes $a \geq 0$ and $b \geq 0$
shows $sqrt\ (a * b) \leq (a + b) / 2$
and $sqrt\ (a * b) = (a + b) / 2 \longleftrightarrow a = b$
proof –
have $0 \leq (a - b) * (a - b)$ and $0 = (a - b) * (a - b) \longleftrightarrow a = b$ by *simp+*
with *right-diff-distrib* [*of a − b a b*] and *left-diff-distrib* [*of a b*] have
$0 \leq a * a - 2 * a * b + b * b$
and $0 = a * a - 2 * a * b + b * b \longleftrightarrow a = b$ by *auto*
hence $4 * a * b \leq a * a + 2 * a * b + b * b$
and $4 * a * b = a * a + 2 * a * b + b * b \longleftrightarrow a = b$ by *auto*
with *distrib-right* [*of a + b a b*] and *distrib-left* [*of a b*] have
$4 * a * b \leq (a + b) * (a + b)$
and $4 * a * b = (a + b) * (a + b) \longleftrightarrow a = b$ by (*simp add*: *field-simps*)+
with *real-sqrt-le-mono* [*of 4 ∗ a ∗ b (a + b) ∗ (a + b)*]
and *real-sqrt-eq-iff* [*of 4 ∗ a ∗ b (a + b) ∗ (a + b)*] have
$sqrt\ (4 * a * b) \leq sqrt\ ((a + b) * (a + b))$
and $sqrt\ (4 * a * b) = sqrt\ ((a + b) * (a + b)) \longleftrightarrow a = b$ by *simp+*
with ‹$a \geq 0$› and ‹$b \geq 0$› have $sqrt\ (4 * a * b) \leq a + b$
and $sqrt\ (4 * a * b) = a + b \longleftrightarrow a = b$ by *simp+*
with *real-sqrt-abs2* [*of 2*] and *real-sqrt-mult* [*of 4 a ∗ b*] show
$sqrt\ (a * b) \leq (a + b) / 2$
and $sqrt\ (a * b) = (a + b) / 2 \longleftrightarrow a = b$ by (*simp add*: *ac-simps*)+
qed

lemma *refl-on-allrel*: *refl-on A* $(A \times A)$
unfolding *refl-on-def*
by *simp*

```
lemma refl-on-restrict:
  assumes refl-on A r
  shows refl-on (A ∩ B) (r ∩ B × B)
proof –
  from ‹refl-on A r› and refl-on-allrel [of B] and refl-on-Int
  show ?thesis by auto
qed

lemma sym-allrel: sym (A × A)
  unfolding sym-def
  by simp

lemma sym-restrict:
  assumes sym r
  shows sym (r ∩ A × A)
proof –
  from ‹sym r› and sym-allrel and sym-Int
  show ?thesis by auto
qed

lemma trans-allrel: trans (A × A)
  unfolding trans-def
  by simp

lemma equiv-Int:
  assumes equiv A r and equiv B s
  shows equiv (A ∩ B) (r ∩ s)
proof –
  from assms and refl-on-Int [of A r B s] and sym-Int and trans-Int
  show ?thesis
    unfolding equiv-def
    by auto
qed

lemma equiv-allrel: equiv A (A × A)
  unfolding equiv-def
  by (simp add: refl-on-allrel sym-allrel trans-allrel)

lemma equiv-restrict:
  assumes equiv A r
  shows equiv (A ∩ B) (r ∩ B × B)
proof –
  from ‹equiv A r› and equiv-allrel [of B] and equiv-Int
  show ?thesis by auto
qed

lemma invertible-times-eq-zero:
  fixes x :: real^'n and A :: real^'n^'n
  assumes invertible A and A *v x = 0
```

shows $x = 0$
using *assms invertible-def matrix-left-invertible-ker* **by** *blast*

lemma *times-invertible-eq-zero*:
fixes x :: *real^'n* **and** A :: *real^'n^'n*
assumes *invertible* A **and** x *v∗* $A = 0$
shows $x = 0$
using *transpose-invertible assms invertible-times-eq-zero* **by** *fastforce*

lemma *matrix-id-invertible*:
invertible (*mat 1* :: (*'a::semiring-1*)*^'n^'n*)
by (*simp add*: *invertible-def*)

lemma *Image-refl-on-nonempty*:
assumes *refl-on* A r **and** $x \in A$
shows $x \in r``\{x\}$
proof
from ‹*refl-on* A r› **and** ‹$x \in A$› **show** $(x, x) \in r$
unfolding *refl-on-def*
by *simp*
qed

lemma *quotient-element-nonempty*:
assumes *equiv* A r **and** $X \in A//r$
shows $\exists\ x.\ x \in X$
using *assms in-quotient-imp-non-empty* **by** *fastforce*

lemma *zero-3*: $(3::3) = 0$
by *simp*

lemma *card-suc-ge-insert*:
fixes A **and** x
shows *card* $A + 1 \geq$ *card* (*insert* x A)
using *card-insert-le-m1* **by** *fastforce*

lemma *card-le-UNIV*:
fixes A :: (*'n::finite*) *set*
shows *card* $A \leq$ *CARD*(*'n*)
by (*simp add*: *card-mono*)

lemma *partition-Image-element*:
assumes *equiv* A r **and** $X \in A//r$ **and** $x \in X$
shows $r``\{x\} = X$
by (*metis Image-singleton-iff assms equiv-class-eq-iff quotientE*)

lemma *card-insert-ge*: *card* (*insert* x A) $\geq$ *card* A
by (*metis card-infinite card-insert-le zero-le*)

lemma *choose-1*:

assumes *card S = 1*
shows ∃ *x. S = {x}*
using ‹*card S = 1*› **and** *card-eq-SucD* [*of S 0*]
by *simp*

lemma *choose-2*:
assumes *card S = 2*
shows ∃ *x y. S = {x,y}*
proof −
from ‹*card S = 2*› **and** *card-eq-SucD* [*of S 1*]
obtain *x* **and** *T* **where** *S = insert x T* **and** *card T = 1* **by** *auto*
from ‹*card T = 1*› **and** *choose-1* **obtain** *y* **where** *T = {y}* **by** *auto*
with ‹*S = insert x T*› **have** *S = {x,y}* **by** *simp*
thus ∃ *x y. S = {x,y}* **by** *auto*
qed

lemma *choose-3*:
assumes *card S = 3*
shows ∃ *x y z. S = {x,y,z}*
proof −
from ‹*card S = 3*› **and** *card-eq-SucD* [*of S 2*]
obtain *x* **and** *T* **where** *S = insert x T* **and** *card T = 2* **by** *auto*
from ‹*card T = 2*› **and** *choose-2* [*of T*] **obtain** *y* **and** *z* **where** *T = {y,z}* **by** *auto*
with ‹*S = insert x T*› **have** *S = {x,y,z}* **by** *simp*
thus ∃ *x y z. S = {x,y,z}* **by** *auto*
qed

lemma *card-gt-0-diff-singleton*:
assumes *card S > 0* **and** *x* ∈ *S*
shows *card (S − {x}) = card S − 1*
proof −
from ‹*card S > 0*› **have** *finite S* **by** (*rule card-ge-0-finite*)
with ‹*x* ∈ *S*›
show *card (S − {x}) = card S − 1* **by** (*simp add*: *card-Diff-singleton*)
qed

lemma *eq-3-or-of-3*:
fixes *j* :: *4*
shows *j = 3* ∨ (∃ *j'*::*3*. *j = of-int* (*Rep-bit1 j'*))
proof (*induct j*)
fix *j-int* :: *int*
assume *0* ≤ *j-int*
assume *j-int* < *int CARD*(*4*)
hence *j-int* ≤ *3* **by** *simp*

show *of-int j-int* = (*3*::*4*) ∨ (∃ *j'*::*3*. *of-int j-int = of-int* (*Rep-bit1 j'*))
proof *cases*
assume *j-int = 3*

```
    thus
      of-int j-int = (3::4) ∨ (∃ j'::3. of-int j-int = of-int (Rep-bit1 j'))
      by simp
  next
    assume j-int ≠ 3
    with ‹j-int ≤ 3› have j-int < 3 by simp
    with ‹0 ≤ j-int› have j-int ∈ {0..<3} by simp
    hence Rep-bit1 (Abs-bit1 j-int :: 3) = j-int
      by (simp add: bit1.Abs-inverse)
    hence of-int j-int = of-int (Rep-bit1 (Abs-bit1 j-int :: 3)) by simp
    thus
      of-int j-int = (3::4) ∨ (∃ j'::3. of-int j-int = of-int (Rep-bit1 j'))
      by auto
  qed
qed

lemma sgn-plus:
  fixes x y :: 'a::linordered-idom
  assumes sgn x = sgn y
  shows sgn (x + y) = sgn x
  by (simp add: assms same-sgn-sgn-add)

lemma sgn-div:
  fixes x y :: 'a::linordered-field
  assumes y ≠ 0 and sgn x = sgn y
  shows x / y > 0
  using assms sgn-1-pos sgn-eq-0-iff by fastforce

lemma abs-plus:
  fixes x y :: 'a::linordered-idom
  assumes sgn x = sgn y
  shows |x + y| = |x| + |y|
  by (simp add: assms same-sgn-abs-add)

lemma sgn-plus-abs:
  fixes x y :: 'a::linordered-idom
  assumes |x| > |y|
  shows sgn (x + y) = sgn x
  by (cases x > 0) (use assms in auto)

end
```

3 Tarski's geometry

```
theory Tarski
  imports Complex-Main Miscellany Metric
begin
```

3.1 The axioms

The axioms, and all theorems beginning with *th* followed by a number, are based on corresponding axioms and theorems in [3].

locale *tarski-first3* =
  fixes $C :: 'p \Rightarrow 'p \Rightarrow 'p \Rightarrow 'p \Rightarrow bool$ $\quad$ (- - ≡ - - [99,99,99,99] 50)
  assumes $A1$: $\forall\, a\ b.\ a\ b \equiv b\ a$
  and $A2$: $\forall\, a\ b\ p\ q\ r\ s.\ a\ b \equiv p\ q \wedge a\ b \equiv r\ s \longrightarrow p\ q \equiv r\ s$
  and $A3$: $\forall\, a\ b\ c.\ a\ b \equiv c\ c \longrightarrow a = b$

locale *tarski-first5* = *tarski-first3* +
  fixes $B :: 'p \Rightarrow 'p \Rightarrow 'p \Rightarrow bool$
  assumes $A4$: $\forall\, q\ a\ b\ c.\ \exists\, x.\ B\ q\ a\ x \wedge a\ x \equiv b\ c$
  and $A5$: $\forall\, a\ b\ c\ d\ a'\ b'\ c'\ d'.\ a \neq b \wedge B\ a\ b\ c \wedge B\ a'\ b'\ c'$
  $\wedge\ a\ b \equiv a'\ b' \wedge b\ c \equiv b'\ c' \wedge a\ d \equiv a'\ d' \wedge$
$b\ d \equiv b'\ d'$
  $\longrightarrow c\ d \equiv c'\ d'$

locale *tarski-absolute-space* = *tarski-first5* +
  assumes $A6$: $\forall\, a\ b.\ B\ a\ b\ a \longrightarrow a = b$
  and $A7$: $\forall\, a\ b\ c\ p\ q.\ B\ a\ p\ c \wedge B\ b\ q\ c \longrightarrow (\exists\, x.\ B\ p\ x\ b \wedge B\ q\ x\ a)$
  and $A11$: $\forall\, X\ Y.\ (\exists\, a.\ \forall\, x\ y.\ x \in X \wedge y \in Y \longrightarrow B\ a\ x\ y)$
  $\longrightarrow (\exists\, b.\ \forall\, x\ y.\ x \in X \wedge y \in Y \longrightarrow B\ x\ b\ y)$

locale *tarski-absolute* = *tarski-absolute-space* +
  assumes $A8$: $\exists\, a\ b\ c.\ \neg\ B\ a\ b\ c \wedge \neg\ B\ b\ c\ a \wedge \neg\ B\ c\ a\ b$
  and $A9$: $\forall\, p\ q\ a\ b\ c.\ p \neq q \wedge a\ p \equiv a\ q \wedge b\ p \equiv b\ q \wedge c\ p \equiv c\ q$
  $\longrightarrow B\ a\ b\ c \vee B\ b\ c\ a \vee B\ c\ a\ b$

locale *tarski-space* = *tarski-absolute-space* +
  assumes $A10$: $\forall\, a\ b\ c\ d\ t.\ B\ a\ d\ t \wedge B\ b\ d\ c \wedge a \neq d$
  $\longrightarrow (\exists\, x\ y.\ B\ a\ b\ x \wedge B\ a\ c\ y \wedge B\ x\ t\ y)$

locale *tarski* = *tarski-absolute* + *tarski-space*

3.2 Semimetric spaces satisfy the first three axioms

context *semimetric*
begin
  definition $smC :: 'p \Rightarrow 'p \Rightarrow 'p \Rightarrow 'p \Rightarrow bool$ (- - $\equiv_{sm}$ - - [99,99,99,99] 50)
    where [*simp*]: $a\ b \equiv_{sm} c\ d \triangleq dist\ a\ b = dist\ c\ d$
end

sublocale *semimetric* < *tarski-first3 smC*
proof
  from *symm* **show** $\forall\, a\ b.\ a\ b \equiv_{sm} b\ a$ **by** *simp*
  show $\forall\, a\ b\ p\ q\ r\ s.\ a\ b \equiv_{sm} p\ q \wedge a\ b \equiv_{sm} r\ s \longrightarrow p\ q \equiv_{sm} r\ s$ **by** *simp*
  show $\forall\, a\ b\ c.\ a\ b \equiv_{sm} c\ c \longrightarrow a = b$ **by** *simp*
qed

3.3 Some consequences of the first three axioms

context *tarski-first3*
begin
lemma *A1'*: $a\ b \equiv b\ a$
by (*simp add*: *A1*)

lemma *A2'*: $[\![a\ b \equiv p\ q;\ a\ b \equiv r\ s]\!] \Longrightarrow p\ q \equiv r\ s$
proof −
assume $a\ b \equiv p\ q$ **and** $a\ b \equiv r\ s$
with *A2* **show** *?thesis* **by** *blast*
qed

lemma *A3'*: $a\ b \equiv c\ c \Longrightarrow a = b$
by (*simp add*: *A3*)

theorem *th2-1*: $a\ b \equiv a\ b$
proof −
from *A2'* [*of* $b\ a\ a\ b\ a\ b$] **and** *A1'* [*of* $b\ a$] **show** *?thesis* **by** *simp*
qed

theorem *th2-2*: $a\ b \equiv c\ d \Longrightarrow c\ d \equiv a\ b$
proof −
assume $a\ b \equiv c\ d$
with *A2'* [*of* $a\ b\ c\ d\ a\ b$] **and** *th2-1* [*of* $a\ b$] **show** *?thesis* **by** *simp*
qed

theorem *th2-3*: $[\![a\ b \equiv c\ d;\ c\ d \equiv e\ f]\!] \Longrightarrow a\ b \equiv e\ f$
proof −
assume $a\ b \equiv c\ d$
with *th2-2* [*of* $a\ b\ c\ d$] **have** $c\ d \equiv a\ b$ **by** *simp*
assume $c\ d \equiv e\ f$
with *A2'* [*of* $c\ d\ a\ b\ e\ f$] **and** ‹$c\ d \equiv a\ b$› **show** *?thesis* **by** *simp*
qed

theorem *th2-4*: $a\ b \equiv c\ d \Longrightarrow b\ a \equiv c\ d$
proof −
assume $a\ b \equiv c\ d$
with *th2-3* [*of* $b\ a\ a\ b\ c\ d$] **and** *A1'* [*of* $b\ a$] **show** *?thesis* **by** *simp*
qed

theorem *th2-5*: $a\ b \equiv c\ d \Longrightarrow a\ b \equiv d\ c$
proof −
assume $a\ b \equiv c\ d$
with *th2-3* [*of* $a\ b\ c\ d\ d\ c$] **and** *A1'* [*of* $c\ d$] **show** *?thesis* **by** *simp*
qed

definition *is-segment* :: *'p set* ⇒ *bool* **where**
is-segment $X \triangleq \exists x\ y.\ X = \{x,\ y\}$

definition *segments* :: $'p$ *set set* **where**
segments = $\{X.$ *is-segment* $X\}$

definition *SC* :: $'p$ *set* $\Rightarrow$ $'p$ *set* $\Rightarrow$ *bool* **where**
$SC\ X\ Y \triangleq \exists\, w\ x\ y\ z.\ X = \{w,\ x\} \land Y = \{y,\ z\} \land w\ x \equiv y\ z$

definition *SC-rel* :: $('p\ set \times 'p\ set)$ *set* **where**
SC-rel = $\{(X,\ Y) \mid X\ Y.\ SC\ X\ Y\}$

lemma *left-segment-congruence*:
  assumes $\{a,\ b\} = \{p,\ q\}$ **and** $p\ q \equiv c\ d$
  shows $a\ b \equiv c\ d$
proof *cases*
  assume $a = p$
  with *unordered-pair-element-equality* [*of a b p q*] **and** ‹$\{a,\ b\} = \{p,\ q\}$›
    have $b = q$ **by** *simp*
  with ‹$p\ q \equiv c\ d$› **and** ‹$a = p$› **show** *?thesis* **by** *simp*
next
  assume $a \neq p$
  with ‹$\{a,\ b\} = \{p,\ q\}$› **have** $a = q$ **by** *auto*
  with *unordered-pair-element-equality* [*of a b q p*] **and** ‹$\{a,\ b\} = \{p,\ q\}$›
    have $b = p$ **by** *auto*
  with ‹$p\ q \equiv c\ d$› **and** ‹$a = q$› **have** $b\ a \equiv c\ d$ **by** *simp*
  with *th2-4* [*of b a c d*] **show** *?thesis* **by** *simp*
qed

lemma *right-segment-congruence*:
  assumes $\{c,\ d\} = \{p,\ q\}$ **and** $a\ b \equiv p\ q$
  shows $a\ b \equiv c\ d$
proof −
  from *th2-2* [*of a b p q*] **and** ‹$a\ b \equiv p\ q$› **have** $p\ q \equiv a\ b$ **by** *simp*
  with *left-segment-congruence* [*of c d p q a b*] **and** ‹$\{c,\ d\} = \{p,\ q\}$›
    have $c\ d \equiv a\ b$ **by** *simp*
  with *th2-2* [*of c d a b*] **show** *?thesis* **by** *simp*
qed

lemma *C-SC-equiv*: $a\ b \equiv c\ d = SC\ \{a,\ b\}\ \{c,\ d\}$
proof
  assume $a\ b \equiv c\ d$
  with *SC-def* [*of* $\{a,\ b\}\ \{c,\ d\}$] **show** $SC\ \{a,\ b\}\ \{c,\ d\}$ **by** *auto*
next
  assume $SC\ \{a,\ b\}\ \{c,\ d\}$
  with *SC-def* [*of* $\{a,\ b\}\ \{c,\ d\}$]
    obtain $w\ x\ y\ z$ **where** $\{a,\ b\} = \{w,\ x\}$ **and** $\{c,\ d\} = \{y,\ z\}$ **and** $w\ x \equiv y\ z$
      by *blast*
  from *left-segment-congruence* [*of a b w x y z*] **and**
      ‹$\{a,\ b\} = \{w,\ x\}$› **and**
      ‹$w\ x \equiv y\ z$›
    have $a\ b \equiv y\ z$ **by** *simp*

```
  with right-segment-congruence [of c d y z a b] and ‹{c, d} = {y, z}›
    show a b ≡ c d by simp
qed

lemmas SC-refl = th2-1 [simplified]

lemma SC-rel-refl: refl-on segments SC-rel
proof -
  note refl-on-def [of segments SC-rel]
  moreover
  { fix Z
    assume Z ∈ SC-rel
    with SC-rel-def obtain X Y where Z = (X, Y) and SC X Y by auto
    from ‹SC X Y› and SC-def [of X Y]
      have ∃ w x. X = {w, x} and ∃ y z. Y = {y, z} by auto
    with is-segment-def [of X] and is-segment-def [of Y]
      have is-segment X and is-segment Y by auto
    with segments-def have X ∈ segments and Y ∈ segments by auto
    with ‹Z = (X, Y)› have Z ∈ segments × segments by simp }
  hence SC-rel ⊆ segments × segments by auto
  moreover
  { fix X
    assume X ∈ segments
    with segments-def have is-segment X by auto
    with is-segment-def [of X] obtain x y where X = {x, y} by auto
   with SC-def [of X X] and SC-refl have SC X X by (simp add: C-SC-equiv)
    with SC-rel-def have (X, X) ∈ SC-rel by simp }
  hence ∀ X. X ∈ segments ⟶ (X, X) ∈ SC-rel by simp
  ultimately show ?thesis by simp
qed

lemma SC-sym:
  assumes SC X Y
  shows SC Y X
proof -
  from SC-def [of X Y] and ‹SC X Y›
    obtain w x y z where X = {w, x} and Y = {y, z} and w x ≡ y z
      by auto
  from th2-2 [of w x y z] and ‹w x ≡ y z› have y z ≡ w x by simp
  with SC-def [of Y X] and ‹X = {w, x}› and ‹Y = {y, z}›
    show SC Y X by (simp add: C-SC-equiv)
qed

lemma SC-sym': SC X Y = SC Y X
proof
  assume SC X Y
  with SC-sym [of X Y] show SC Y X by simp
next
  assume SC Y X
```

```
  with SC-sym [of Y X] show SC X Y by simp
qed

lemma SC-rel-sym: sym SC-rel
proof -
  { fix X Y
    assume (X, Y) ∈ SC-rel
    with SC-rel-def have SC X Y by simp
    with SC-sym' have SC Y X by simp
    with SC-rel-def have (Y, X) ∈ SC-rel by simp }
  with sym-def [of SC-rel] show ?thesis by blast
qed

lemma SC-trans:
  assumes SC X Y and SC Y Z
  shows SC X Z
proof -
  from SC-def [of X Y] and ‹SC X Y›
    obtain w x y z where X = {w, x} and Y = {y, z} and w x ≡ y z
      by auto
  from SC-def [of Y Z] and ‹SC Y Z›
    obtain p q r s where Y = {p, q} and Z = {r, s} and p q ≡ r s by auto
  from ‹Y = {y, z}› and ‹Y = {p, q}› and ‹p q ≡ r s›
    have y z ≡ r s by (simp add: C-SC-equiv)
  with th2-3 [of w x y z r s] and ‹w x ≡ y z› have w x ≡ r s by simp
  with SC-def [of X Z] and ‹X = {w, x}› and ‹Z = {r, s}›
    show SC X Z by (simp add: C-SC-equiv)
qed

lemma SC-rel-trans: trans SC-rel
proof -
  { fix X Y Z
    assume (X, Y) ∈ SC-rel and (Y, Z) ∈ SC-rel
    with SC-rel-def have SC X Y and SC Y Z by auto
    with SC-trans [of X Y Z] have SC X Z by simp
    with SC-rel-def have (X, Z) ∈ SC-rel by simp }
  with trans-def [of SC-rel] show ?thesis by blast
qed

lemma A3-reversed:
  assumes a a ≡ b c
  shows b = c
proof -
  from ‹a a ≡ b c› have b c ≡ a a by (rule th2-2)
  thus b = c by (rule A3')
qed

lemma equiv-segments-SC-rel: equiv segments SC-rel
  by (simp add: equiv-def SC-rel-refl SC-rel-sym SC-rel-trans)
```

end

3.4 Some consequences of the first five axioms

context *tarski-first5*
begin
lemma *A4'*: $\exists x.\ B\ q\ a\ x \wedge a\ x \equiv b\ c$
by (*simp add*: *A4* [*simplified*])

theorem *th2-8*: $a\ a \equiv b\ b$
proof –
from *A4'* [*of - a b b*] **obtain** x **where** $a\ x \equiv b\ b$ **by** *auto*
with *A3'* [*of a x b*] **have** $x = a$ **by** *simp*
with ‹$a\ x \equiv b\ b$› **show** *?thesis* **by** *simp*
qed

definition *OFS* :: $['p,'p,'p,'p,'p,'p,'p,'p] \Rightarrow$ *bool* **where**
OFS $a\ b\ c\ d\ a'\ b'\ c'\ d' \triangleq$
$B\ a\ b\ c \wedge B\ a'\ b'\ c' \wedge a\ b \equiv a'\ b' \wedge b\ c \equiv b'\ c' \wedge a\ d \equiv a'\ d' \wedge b\ d \equiv b'\ d'$

lemma *A5'*: $[\![$*OFS* $a\ b\ c\ d\ a'\ b'\ c'\ d'$; $a \neq b]\!] \Longrightarrow c\ d \equiv c'\ d'$
proof –
assume *OFS* $a\ b\ c\ d\ a'\ b'\ c'\ d'$ **and** $a \neq b$
with *A5* **and** *OFS-def* **show** *?thesis* **by** *blast*
qed

theorem *th2-11*:
assumes *hypotheses*:
$B\ a\ b\ c$
$B\ a'\ b'\ c'$
$a\ b \equiv a'\ b'$
$b\ c \equiv b'\ c'$
shows $a\ c \equiv a'\ c'$
proof *cases*
assume $a = b$
with ‹$a\ b \equiv a'\ b'$› **have** $a' = b'$ **by** (*simp add*: *A3-reversed*)
with ‹$b\ c \equiv b'\ c'$› **and** ‹$a = b$› **show** *?thesis* **by** *simp*
next
assume $a \neq b$
moreover
note *A5'* [*of a b c a a' b' c' a'*] **and**
unordered-pair-equality [*of a c*] **and**
unordered-pair-equality [*of a' c'*]
moreover
from *OFS-def* [*of a b c a a' b' c' a'*] **and**
hypotheses **and**
th2-8 [*of a a'*] **and**
unordered-pair-equality [*of a b*] **and**

unordered-pair-equality [*of* a' b']
have *OFS* a b c a a' b' c' a' **by** (*simp add*: *C-SC-equiv*)
ultimately show *?thesis* **by** (*simp add*: *C-SC-equiv*)
qed

lemma *A4-unique*:
assumes $q \neq a$ **and** $B\ q\ a\ x$ **and** $a\ x \equiv b\ c$
and $B\ q\ a\ x'$ **and** $a\ x' \equiv b\ c$
shows $x = x'$
proof −
from *SC-sym'* **and** *SC-trans* **and** *C-SC-equiv* **and** ‹$a\ x' \equiv b\ c$› **and** ‹$a\ x \equiv b$
c›
have $a\ x \equiv a\ x'$ **by** *blast*
with *th2-11* [*of* q a x q a x'] **and** ‹$B\ q\ a\ x$› **and** ‹$B\ q\ a\ x'$› **and** *SC-refl*
have $q\ x \equiv q\ x'$ **by** *simp*
with *OFS-def* [*of* q a x x q a x x'] **and**
‹$B\ q\ a\ x$› **and**
SC-refl **and**
‹$a\ x \equiv a\ x'$›
have *OFS* q a x x q a x x' **by** *simp*
with *A5'* [*of* q a x x q a x x'] **and** ‹$q \neq a$› **have** $x\ x \equiv x\ x'$ **by** *simp*
thus $x = x'$ **by** (*rule A3-reversed*)
qed

theorem *th2-12*:
assumes $q \neq a$
shows $\exists!x.\ B\ q\ a\ x \wedge a\ x \equiv b\ c$
using ‹$q \neq a$› **and** *A4'* **and** *A4-unique*
by *blast*
end

3.5 Simple theorems about betweenness

theorem (**in** *tarski-first5*) *th3-1*: $B\ a\ b\ b$
proof −
from *A4* [*rule-format*, *of* a b b b] **obtain** x **where** $B\ a\ b\ x$ **and** $b\ x \equiv b\ b$ **by**
auto
from *A3* [*rule-format*, *of* b x b] **and** ‹$b\ x \equiv b\ b$› **have** $b = x$ **by** *simp*
with ‹$B\ a\ b\ x$› **show** $B\ a\ b\ b$ **by** *simp*
qed

context *tarski-absolute-space*
begin
lemma *A6'*:
assumes $B\ a\ b\ a$
shows $a = b$
proof −
from *A6* **and** ‹$B\ a\ b\ a$› **show** $a = b$ **by** *simp*
qed

lemma *A7′*:
  assumes $B\ a\ p\ c$ **and** $B\ b\ q\ c$
  shows $\exists x.\ B\ p\ x\ b \wedge B\ q\ x\ a$
proof –
  from *A7* **and** ‹$B\ a\ p\ c$› **and** ‹$B\ b\ q\ c$› **show** *?thesis* **by** *blast*
qed

lemma *A11′*:
  assumes $\forall\ x\ y.\ x \in X \wedge y \in Y \longrightarrow B\ a\ x\ y$
  shows $\exists\ b.\ \forall\ x\ y.\ x \in X \wedge y \in Y \longrightarrow B\ x\ b\ y$
proof –
  from *assms* **have** $\exists\ a.\ \forall\ x\ y.\ x \in X \wedge y \in Y \longrightarrow B\ a\ x\ y$ **by** (*rule exI*)
  thus $\exists\ b.\ \forall\ x\ y.\ x \in X \wedge y \in Y \longrightarrow B\ x\ b\ y$ **by** (*rule A11* [*rule-format*])
qed

theorem *th3-2*:
  assumes $B\ a\ b\ c$
  shows $B\ c\ b\ a$
proof –
  from *th3-1* **have** $B\ b\ c\ c$ **by** *simp*
  with *A7′* **and** ‹$B\ a\ b\ c$› **obtain** x **where** $B\ b\ x\ b$ **and** $B\ c\ x\ a$ **by** *blast*
  from *A6′* **and** ‹$B\ b\ x\ b$› **have** $x = b$ **by** *auto*
  with ‹$B\ c\ x\ a$› **show** $B\ c\ b\ a$ **by** *simp*
qed

theorem *th3-4*:
  assumes $B\ a\ b\ c$ **and** $B\ b\ a\ c$
  shows $a = b$
proof –
  from ‹$B\ a\ b\ c$› **and** ‹$B\ b\ a\ c$› **and** *A7′* [*of* $a\ b\ c\ b\ a$]
  obtain x **where** $B\ b\ x\ b$ **and** $B\ a\ x\ a$ **by** *auto*
  hence $b = x$ **and** $a = x$ **by** (*simp-all add*: *A6′*)
  thus $a = b$ **by** *simp*
qed

theorem *th3-5-1*:
  assumes $B\ a\ b\ d$ **and** $B\ b\ c\ d$
  shows $B\ a\ b\ c$
proof –
  from ‹$B\ a\ b\ d$› **and** ‹$B\ b\ c\ d$› **and** *A7′* [*of* $a\ b\ d\ b\ c$]
  obtain x **where** $B\ b\ x\ b$ **and** $B\ c\ x\ a$ **by** *auto*
  from ‹$B\ b\ x\ b$› **have** $b = x$ **by** (*rule A6′*)
  with ‹$B\ c\ x\ a$› **have** $B\ c\ b\ a$ **by** *simp*
  thus $B\ a\ b\ c$ **by** (*rule th3-2*)
qed

theorem *th3-6-1*:
  assumes $B\ a\ b\ c$ **and** $B\ a\ c\ d$

shows $B\ b\ c\ d$
proof $-$
from $\langle B\ a\ c\ d\rangle$ **and** $\langle B\ a\ b\ c\rangle$ **and** *th3-2* **have** $B\ d\ c\ a$ **and** $B\ c\ b\ a$ **by** *fast+*
hence $B\ d\ c\ b$ **by** (*rule th3-5-1*)
thus $B\ b\ c\ d$ **by** (*rule th3-2*)
qed

theorem *th3-7-1*:
assumes $b \neq c$ **and** $B\ a\ b\ c$ **and** $B\ b\ c\ d$
shows $B\ a\ c\ d$
proof $-$
from *A4'* **obtain** x **where** $B\ a\ c\ x$ **and** $c\ x \equiv c\ d$ **by** *fast*
from $\langle B\ a\ b\ c\rangle$ **and** $\langle B\ a\ c\ x\rangle$ **have** $B\ b\ c\ x$ **by** (*rule th3-6-1*)
have $c\ d \equiv c\ d$ **by** (*rule th2-1*)
with $\langle b \neq c\rangle$ **and** $\langle B\ b\ c\ x\rangle$ **and** $\langle c\ x \equiv c\ d\rangle$ **and** $\langle B\ b\ c\ d\rangle$
have $x = d$ **by** (*rule A4-unique*)
with $\langle B\ a\ c\ x\rangle$ **show** $B\ a\ c\ d$ **by** *simp*
qed

theorem *th3-7-2*:
assumes $b \neq c$ **and** $B\ a\ b\ c$ **and** $B\ b\ c\ d$
shows $B\ a\ b\ d$
proof $-$
from $\langle B\ b\ c\ d\rangle$ **and** $\langle B\ a\ b\ c\rangle$ **and** *th3-2* **have** $B\ d\ c\ b$ **and** $B\ c\ b\ a$ **by** *fast+*
with $\langle b \neq c\rangle$ **and** *th3-7-1* [*of* $c\ b\ d\ a$] **have** $B\ d\ b\ a$ **by** *simp*
thus $B\ a\ b\ d$ **by** (*rule th3-2*)
qed
end

3.6 Simple theorems about congruence and betweenness

definition (**in** *tarski-first5*) *Col* :: $'p \Rightarrow 'p \Rightarrow 'p \Rightarrow$ *bool* **where**
$Col\ a\ b\ c \triangleq B\ a\ b\ c \vee B\ b\ c\ a \vee B\ c\ a\ b$

end

4 Real Euclidean space and Tarski's axioms

theory *Euclid-Tarski*
imports *Tarski*
begin

4.1 Real Euclidean space satisfies the first five axioms

abbreviation
real-euclid-C :: [*real*ˆ($'n$::*finite*), *real*ˆ($'n$), *real*ˆ($'n$), *real*ˆ($'n$)] $\Rightarrow$ *bool*
(- - $\equiv_{\mathbb{R}}$ - - [*99*,*99*,*99*,*99*] *50*) **where**
real-euclid-C $\triangleq$ *norm-metric.smC*

definition *real-euclid-B* :: $[real\hat{}('n::finite), real\hat{}('n), real\hat{}('n)] \Rightarrow bool$
($B_{\mathbb{R}}$ - - - [99,99,99] 50) **where**
$B_{\mathbb{R}}\ a\ b\ c \triangleq \exists l.\ 0 \leq l \wedge l \leq 1 \wedge b - a = l *_R (c - a)$

interpretation *real-euclid*: *tarski-first5 real-euclid-C real-euclid-B*
proof

By virtue of being a semimetric space, real Euclidean space is already known to satisfy the first three axioms.

```
{ fix q a b c
  have ∃x. B_ℝ q a x ∧ a x ≡_ℝ b c
  proof cases
    assume q = a
    let ?x = a + c − b
    have B_ℝ q a ?x
    proof −
      let ?l = 0 :: real
      note real-euclid-B-def [of q a ?x]
      moreover
        have ?l ≥ 0 and ?l ≤ 1 by auto
      moreover
        from ‹q = a› have a − q = 0 by simp
        hence a − q = ?l *_R (?x − q) by simp
      ultimately show ?thesis by auto
    qed
    moreover
      have a − ?x = b − c by simp
      hence a ?x ≡_ℝ b c by (simp add: field-simps)
    ultimately show ?thesis by blast
  next
    assume q ≠ a
    hence norm-dist q a > 0 by simp
    let ?k = norm-dist b c / norm-dist q a
    let ?x = a + ?k *_R (a − q)
    have B_ℝ q a ?x
    proof −
      let ?l = 1 / (1 + ?k)
      have ?l > 0 by (simp add: add-pos-nonneg)
      note real-euclid-B-def [of q a ?x]
      moreover
        have ?l ≥ 0 and ?l ≤ 1 by (auto simp add: add-pos-nonneg)
      moreover
        from scaleR-left-distrib [of 1 ?k a − q]
          have (1 + ?k) *_R (a − q) = ?x − q by simp
        hence ?l *_R ((1 + ?k) *_R (a − q)) = ?l *_R (?x − q) by simp
        with ‹?l > 0› and scaleR-right-diff-distrib [of ?l ?x q]
          have a − q = ?l *_R (?x − q) by simp
      ultimately show B_ℝ q a ?x by blast
    qed
    moreover
```

have $a\ ?x \equiv_{\mathbb{R}} b\ c$
proof $-$
from *norm-scaleR* [*of* $?k\ a - q$] **have**
$norm\text{-}dist\ a\ ?x = |?k| * norm\ (a - q)$ **by** *simp*
also have
$\ldots = ?k * norm\ (a - q)$ **by** *simp*
also from *norm-metric.symm* [*of* $q\ a$] **have**
$\ldots = ?k * norm\text{-}dist\ q\ a$ **by** *simp*
finally have
$norm\text{-}dist\ a\ ?x = norm\text{-}dist\ b\ c\ /\ norm\text{-}dist\ q\ a * norm\text{-}dist\ q\ a$ **.**
with ‹$norm\text{-}dist\ q\ a > 0$› **show** $a\ ?x \equiv_{\mathbb{R}} b\ c$ **by** *auto*
qed
ultimately show *?thesis* **by** *blast*
qed }
thus $\forall q\ a\ b\ c.\ \exists x.\ B_{\mathbb{R}}\ q\ a\ x \wedge a\ x \equiv_{\mathbb{R}} b\ c$ **by** *auto*
{ fix $a\ b\ c\ d\ a'\ b'\ c'\ d'$
assume $a \neq b$ **and**
$B_{\mathbb{R}}\ a\ b\ c$ **and**
$B_{\mathbb{R}}\ a'\ b'\ c'$ **and**
$a\ b \equiv_{\mathbb{R}} a'\ b'$ **and**
$b\ c \equiv_{\mathbb{R}} b'\ c'$ **and**
$a\ d \equiv_{\mathbb{R}} a'\ d'$ **and**
$b\ d \equiv_{\mathbb{R}} b'\ d'$
have $c\ d \equiv_{\mathbb{R}} c'\ d'$
proof $-$
{ fix m
fix $p\ q\ r$ :: *real*^(*'n::finite*)
assume $0 \leq m$ **and**
$m \leq 1$ **and**
$p \neq q$ **and**
$q - p = m *_R (r - p)$
from ‹$p \neq q$› **and** ‹$q - p = m *_R (r - p)$› **have** $m \neq 0$
proof $-$
{ assume $m = 0$
with ‹$q - p = m *_R (r - p)$› **have** $q - p = 0$ **by** *simp*
with ‹$p \neq q$› **have** *False* **by** *simp* **}**
thus *?thesis* **..**
qed
with ‹$m \geq 0$› **have** $m > 0$ **by** *simp*
from ‹$q - p = m *_R (r - p)$› **and**
scaleR-right-diff-distrib [*of* $m\ r\ p$]
have $q - p = m *_R r - m *_R p$ **by** *simp*
hence $q - p - q + p - m *_R r =$
$m *_R r - m *_R p - q + p - m *_R r$
by *simp*
with *scaleR-left-diff-distrib* [*of* $1\ m\ p$] **and**
scaleR-left-diff-distrib [*of* $1\ m\ q$]
have $(1 - m) *_R p - (1 - m) *_R q = m *_R q - m *_R r$ **by** *auto*
with *scaleR-right-diff-distrib* [*of* $1 - m\ p\ q$] **and**

scaleR-right-diff-distrib [*of* m q r]
have $(1 - m) *_R (p - q) = m *_R (q - r)$ **by** *simp*
with *norm-scaleR* [*of* $1 - m$ $p - q$] **and** *norm-scaleR* [*of* m $q - r$]
have $|1 - m| * norm\ (p - q) = |m| * norm\ (q - r)$ **by** *simp*
with ‹$m > 0$› **and** ‹$m \leq 1$›
have $norm\ (q - r) = (1 - m) / m * norm\ (p - q)$ **by** *simp*
moreover from ‹$p \neq q$› **have** $norm\ (p - q) \neq 0$ **by** *simp*
ultimately
have $norm\ (q - r) / norm\ (p - q) = (1 - m) / m$ **by** *simp*
with ‹$m \neq 0$› **have**
norm-dist q r / *norm-dist* p q $= (1 - m) / m$ **and** $m \neq 0$ **by** *auto* **}**
note *linelemma* = *this*
from *real-euclid-B-def* [*of* a b c] **and** ‹$B_{\mathbb{R}}$ a b c›
obtain l **where** $0 \leq l$ **and** $l \leq 1$ **and** $b - a = l *_R (c - a)$ **by** *auto*
from *real-euclid-B-def* [*of* a' b' c'] **and** ‹$B_{\mathbb{R}}$ a' b' c'›
obtain l' **where** $0 \leq l'$ **and** $l' \leq 1$ **and** $b' - a' = l' *_R (c' - a')$ **by** *auto*
from ‹$a \neq b$› **and** ‹a b $\equiv_{\mathbb{R}}$ a' b'› **have** $a' \neq b'$ **by** *auto*
from *linelemma* [*of* l a b c] **and**
‹$l \geq 0$› **and**
‹$l \leq 1$› **and**
‹$a \neq b$› **and**
‹$b - a = l *_R (c - a)$›
have $l \neq 0$ **and** $(1 - l) / l$ = *norm-dist* b c / *norm-dist* a b **by** *auto*
from ‹$(1 - l) / l$ = *norm-dist* b c / *norm-dist* a b› **and**
‹a b $\equiv_{\mathbb{R}}$ a' b'› **and**
‹b c $\equiv_{\mathbb{R}}$ b' c'›
have $(1 - l) / l$ = *norm-dist* b' c' / *norm-dist* a' b' **by** *simp*
with *linelemma* [*of* l' a' b' c'] **and**
‹$l' \geq 0$› **and**
‹$l' \leq 1$› **and**
‹$a' \neq b'$› **and**
‹$b' - a' = l' *_R (c' - a')$›
have $l' \neq 0$ **and** $(1 - l) / l = (1 - l') / l'$ **by** *auto*
from ‹$(1 - l) / l = (1 - l') / l'$›
have $(1 - l) / l * l * l' = (1 - l') / l' * l * l'$ **by** *simp*
with ‹$l \neq 0$› **and** ‹$l' \neq 0$› **have** $(1 - l) * l' = (1 - l') * l$ **by** *simp*
with *left-diff-distrib* [*of* 1 l l'] **and** *left-diff-distrib* [*of* 1 l' l]
have $l = l'$ **by** *simp*
{ fix m
fix p q r s :: *real*^('*n*::*finite*)
assume $m \neq 0$ **and**
$q - p = m *_R (r - p)$
with *scaleR-scaleR* **have** $r - p = (1/m) *_R (q - p)$ **by** *simp*
with *cosine-rule* [*of* r s p]
have $(norm\text{-}dist\ r\ s)^2 = (norm\text{-}dist\ r\ p)^2 + (norm\text{-}dist\ p\ s)^2 +$
$2 * (((1/m) *_R (q - p)) \cdot (p - s))$
by *simp*
also from *inner-scaleR-left* [*of* $1/m$ $q - p$ $p - s$]
have ... =

$(\textit{norm-dist}\ r\ p)^2 + (\textit{norm-dist}\ p\ s)^2 + 2/m * ((q - p) \cdot (p - s))$
by *simp*
also from $\langle m \neq 0 \rangle$ **and** *cosine-rule* [*of q s p*]
have $\ldots = (\textit{norm-dist}\ r\ p)^2 + (\textit{norm-dist}\ p\ s)^2 +$
$1/m * ((\textit{norm-dist}\ q\ s)^2 - (\textit{norm-dist}\ q\ p)^2 - (\textit{norm-dist}\ p\ s)^2)$
by *simp*
finally have $(\textit{norm-dist}\ r\ s)^2 = (\textit{norm-dist}\ r\ p)^2 + (\textit{norm-dist}\ p\ s)^2 +$
$1/m * ((\textit{norm-dist}\ q\ s)^2 - (\textit{norm-dist}\ q\ p)^2 - (\textit{norm-dist}\ p\ s)^2)$ **.**
moreover
{ from *norm-dist-dot* [*of r p*] **and** $\langle r - p = (1/m) *_R (q - p) \rangle$
have $(\textit{norm-dist}\ r\ p)^2 = ((1/m) *_R (q - p)) \cdot ((1/m) *_R (q - p))$
by *simp*
also from *inner-scaleR-left* [*of* $1/m\ q - p$] **and**
inner-scaleR-right [*of* - $1/m\ q - p$]
have $\ldots = 1/m^2 * ((q - p) \cdot (q - p))$
by (*simp add*: *power2-eq-square*)
also from *norm-dist-dot* [*of q p*] **have** $\ldots = 1/m^2 * (\textit{norm-dist}\ q\ p)^2$
by *simp*
finally have $(\textit{norm-dist}\ r\ p)^2 = 1/m^2 * (\textit{norm-dist}\ q\ p)^2$ **. }**
ultimately have
$(\textit{norm-dist}\ r\ s)^2 = 1/m^2 * (\textit{norm-dist}\ q\ p)^2 + (\textit{norm-dist}\ p\ s)^2 +$
$1/m * ((\textit{norm-dist}\ q\ s)^2 - (\textit{norm-dist}\ q\ p)^2 - (\textit{norm-dist}\ p\ s)^2)$
by *simp*
with *norm-metric.symm* [*of q p*]
have $(\textit{norm-dist}\ r\ s)^2 = 1/m^2 * (\textit{norm-dist}\ p\ q)^2 + (\textit{norm-dist}\ p\ s)^2 +$
$1/m * ((\textit{norm-dist}\ q\ s)^2 - (\textit{norm-dist}\ p\ q)^2 - (\textit{norm-dist}\ p\ s)^2)$
by *simp* **}**
note *fiveseglemma* = *this*
from *fiveseglemma* [*of l b a c d*] **and** $\langle l \neq 0 \rangle$ **and** $\langle b - a = l *_R (c - a) \rangle$
have $(\textit{norm-dist}\ c\ d)^2 = 1/l^2 * (\textit{norm-dist}\ a\ b)^2 + (\textit{norm-dist}\ a\ d)^2 +$
$1/l * ((\textit{norm-dist}\ b\ d)^2 - (\textit{norm-dist}\ a\ b)^2 - (\textit{norm-dist}\ a\ d)^2)$
by *simp*
also from $\langle l = l' \rangle$ **and**
$\langle a\ b \equiv_{\mathbb{R}} a'\ b' \rangle$ **and**
$\langle a\ d \equiv_{\mathbb{R}} a'\ d' \rangle$ **and**
$\langle b\ d \equiv_{\mathbb{R}} b'\ d' \rangle$
have $\ldots = 1/l'^2 * (\textit{norm-dist}\ a'\ b')^2 + (\textit{norm-dist}\ a'\ d')^2 +$
$1/l' * ((\textit{norm-dist}\ b'\ d')^2 - (\textit{norm-dist}\ a'\ b')^2 - (\textit{norm-dist}\ a'\ d')^2)$
by *simp*
also from *fiveseglemma* [*of* $l'\ b'\ a'\ c'\ d'$] **and**
$\langle l' \neq 0 \rangle$ **and**
$\langle b' - a' = l' *_R (c' - a') \rangle$
have $\ldots = (\textit{norm-dist}\ c'\ d')^2$ **by** *simp*
finally have $(\textit{norm-dist}\ c\ d)^2 = (\textit{norm-dist}\ c'\ d')^2$ **.**
hence $\textit{sqrt}\ ((\textit{norm-dist}\ c\ d)^2) = \textit{sqrt}\ ((\textit{norm-dist}\ c'\ d')^2)$ **by** *simp*
with *real-sqrt-abs* **show** $c\ d \equiv_{\mathbb{R}} c'\ d'$ **by** *simp*
qed }
thus $\forall a\ b\ c\ d\ a'\ b'\ c'\ d'.$
$a \neq b \wedge B_{\mathbb{R}}\ a\ b\ c \wedge B_{\mathbb{R}}\ a'\ b'\ c' \wedge$

$a\ b \equiv_{\mathbb{R}} a'\ b' \wedge b\ c \equiv_{\mathbb{R}} b'\ c' \wedge a\ d \equiv_{\mathbb{R}} a'\ d' \wedge b\ d \equiv_{\mathbb{R}} b'\ d' \longrightarrow$
$c\ d \equiv_{\mathbb{R}} c'\ d'$
by *blast*
qed

4.2 Real Euclidean space also satisfies axioms 6, 7, and 11

lemma *rearrange-real-euclid-B*:
fixes $w\ y\ z$ :: *real*ˆ('*n*) **and** h
shows $y - w = h *_R (z - w) \longleftrightarrow y = h *_R z + (1 - h) *_R w$
proof
assume $y - w = h *_R (z - w)$
hence $y - w + w = h *_R (z - w) + w$ **by** *simp*
hence $y = h *_R (z - w) + w$ **by** *simp*
with *scaleR-right-diff-distrib* [*of h z w*]
have $y = h *_R z + w - h *_R w$ **by** *simp*
with *scaleR-left-diff-distrib* [*of 1 h w*]
show $y = h *_R z + (1 - h) *_R w$ **by** *simp*
next
assume $y = h *_R z + (1 - h) *_R w$
with *scaleR-left-diff-distrib* [*of 1 h w*]
have $y = h *_R z + w - h *_R w$ **by** *simp*
with *scaleR-right-diff-distrib* [*of h z w*]
have $y = h *_R (z - w) + w$ **by** *simp*
hence $y - w + w = h *_R (z - w) + w$ **by** *simp*
thus $y - w = h *_R (z - w)$ **by** *simp*
qed

interpretation *real-euclid*: *tarski-absolute-space real-euclid-C real-euclid-B*
proof
{ fix $a\ b$
assume $B_{\mathbb{R}}\ a\ b\ a$
with *real-euclid-B-def* [*of a b a*]
obtain l **where** $b - a = l *_R (a - a)$ **by** *auto*
hence $a = b$ **by** *simp* **}**
thus $\forall a\ b.\ B_{\mathbb{R}}\ a\ b\ a \longrightarrow a = b$ **by** *auto*
{ fix $a\ b\ c\ p\ q$
assume $B_{\mathbb{R}}\ a\ p\ c$ **and** $B_{\mathbb{R}}\ b\ q\ c$
from *real-euclid-B-def* [*of a p c*] **and** ‹$B_{\mathbb{R}}\ a\ p\ c$›
obtain i **where** $i \geq 0$ **and** $i \leq 1$ **and** $p - a = i *_R (c - a)$ **by** *auto*
have $\exists x.\ B_{\mathbb{R}}\ p\ x\ b \wedge B_{\mathbb{R}}\ q\ x\ a$
proof *cases*
assume $i = 0$
with ‹$p - a = i *_R (c - a)$› **have** $p = a$ **by** *simp*
hence $p - a = 0 *_R (b - p)$ **by** *simp*
moreover have $(0::real) \geq 0$ **and** $(0::real) \leq 1$ **by** *auto*
moreover note *real-euclid-B-def* [*of p a b*]
ultimately have $B_{\mathbb{R}}\ p\ a\ b$ **by** *auto*
moreover

{ **have** $a - q = 1 *_R (a - q)$ **by** *simp*
  moreover have $(1::real) \geq 0$ **and** $(1::real) \leq 1$ **by** *auto*
  moreover note *real-euclid-B-def* [*of q a a*]
  ultimately have $B_{\mathbb{R}}\ q\ a\ a$ **by** *blast* }
ultimately have $B_{\mathbb{R}}\ p\ a\ b \wedge B_{\mathbb{R}}\ q\ a\ a$ **by** *simp*
thus $\exists x.\ B_{\mathbb{R}}\ p\ x\ b \wedge B_{\mathbb{R}}\ q\ x\ a$ **by** *auto*

next

assume $i \neq 0$
from *real-euclid-B-def* [*of b q c*] **and** ‹$B_{\mathbb{R}}\ b\ q\ c$›
  obtain j **where** $j \geq 0$ **and** $j \leq 1$ **and** $q - b = j *_R (c - b)$ **by** *auto*
from ‹$i \geq 0$› **and** ‹$i \leq 1$›
  have $1 - i \geq 0$ **and** $1 - i \leq 1$ **by** *auto*
from ‹$j \geq 0$› **and** ‹$1 - i \geq 0$›
  have $j * (1 - i) \geq 0$ **by** *auto*
with ‹$i \geq 0$› **and** ‹$i \neq 0$› **have** $i + j * (1 - i) > 0$ **by** *simp*
hence $i + j * (1 - i) \neq 0$ **by** *simp*
let $?l = j * (1 - i) / (i + j * (1 - i))$
from *diff-divide-distrib* [*of* $i + j * (1 - i)$ $j * (1 - i)$ $i + j * (1 - i)$] **and**
    ‹$i + j * (1 - i) \neq 0$›
  have $1 - ?l = i / (i + j * (1 - i))$ **by** *simp*
let $?k = i * (1 - j) / (j + i * (1 - j))$
from *right-diff-distrib* [*of i 1 j*] **and**
    right-diff-distrib [*of j 1 i*] **and**
    mult.commute [*of i j*] **and**
    add.commute [*of i j*]
  have $j + i * (1 - j) = i + j * (1 - i)$ **by** *simp*
with ‹$i + j * (1 - i) \neq 0$› **have** $j + i * (1 - j) \neq 0$ **by** *simp*
with *diff-divide-distrib* [*of* $j + i * (1 - j)$ $i * (1 - j)$ $j + i * (1 - j)$]
  have $1 - ?k = j / (j + i * (1 - j))$ **by** *simp*
with ‹$1 - ?l = i / (i + j * (1 - i))$› **and**
    ‹$j + i * (1 - j) = i + j * (1 - i)$› **and**
    times-divide-eq-left [*of* - $i + j * (1 - i)$] **and**
    mult.commute [*of i j*]
  have $(1 - ?l) * j = (1 - ?k) * i$ **by** *simp*
moreover
{ **from** ‹$1 - ?k = j / (j + i * (1 - j))$› **and**
      ‹$j + i * (1 - j) = i + j * (1 - i)$›
    have $?l = (1 - ?k) * (1 - i)$ **by** *simp* }
moreover
{ **from** ‹$1 - ?l = i / (i + j * (1 - i))$› **and**
      ‹$j + i * (1 - j) = i + j * (1 - i)$›
    have $(1 - ?l) * (1 - j) = ?k$ **by** *simp* }
ultimately
  have $?l *_R a + ((1 - ?l) * j) *_R c + ((1 - ?l) * (1 - j)) *_R b =$
      $?k *_R b + ((1 - ?k) * i) *_R c + ((1 - ?k) * (1 - i)) *_R a$
    by *simp*
with *scaleR-scaleR*
  have $?l *_R a + (1 - ?l) *_R j *_R c + (1 - ?l) *_R (1 - j) *_R b =$
      $?k *_R b + (1 - ?k) *_R i *_R c + (1 - ?k) *_R (1 - i) *_R a$

by *simp*
with *scaleR-right-distrib* [*of* $(1 - ?l)$ $j *_R c$ $(1 - j) *_R b$] **and**
scaleR-right-distrib [*of* $(1 - ?k)$ $i *_R c$ $(1 - i) *_R a$] **and**
add.assoc [*of* $?l *_R a$ $(1 - ?l) *_R j *_R c$ $(1 - ?l) *_R (1 - j) *_R b$] **and**
add.assoc [*of* $?k *_R b$ $(1 - ?k) *_R i *_R c$ $(1 - ?k) *_R (1 - i) *_R a$]
have $?l *_R a + (1 - ?l) *_R (j *_R c + (1 - j) *_R b) =$
$?k *_R b + (1 - ?k) *_R (i *_R c + (1 - i) *_R a)$
by *arith*
from ‹$?l *_R a + (1 - ?l) *_R (j *_R c + (1 - j) *_R b) =$
$?k *_R b + (1 - ?k) *_R (i *_R c + (1 - i) *_R a)$› **and**
‹$p - a = i *_R (c - a)$› **and**
‹$q - b = j *_R (c - b)$› **and**
rearrange-real-euclid-B [*of p a i c*] **and**
rearrange-real-euclid-B [*of q b j c*]
have $?l *_R a + (1 - ?l) *_R q = ?k *_R b + (1 - ?k) *_R p$ **by** *simp*
let $?x = ?l *_R a + (1 - ?l) *_R q$
from *rearrange-real-euclid-B* [*of ?x q ?l a*]
have $?x - q = ?l *_R (a - q)$ **by** *simp*
from ‹$?x = ?k *_R b + (1 - ?k) *_R p$› **and**
rearrange-real-euclid-B [*of ?x p ?k b*]
have $?x - p = ?k *_R (b - p)$ **by** *simp*
from ‹$i + j * (1 - i) > 0$› **and**
‹$j * (1 - i) \geq 0$› **and**
zero-le-divide-iff [*of* $j * (1 - i)$ $i + j * (1 - i)$]
have $?l \geq 0$ **by** *simp*
from ‹$i + j * (1 - i) > 0$› **and**
‹$i \geq 0$› **and**
zero-le-divide-iff [*of* i $i + j * (1 - i)$] **and**
‹$1 - ?l = i / (i + j * (1 - i))$›
have $1 - ?l \geq 0$ **by** *simp*
hence $?l \leq 1$ **by** *simp*
with ‹$?l \geq 0$› **and**
‹$?x - q = ?l *_R (a - q)$› **and**
real-euclid-B-def [*of q ?x a*]
have $B_{\mathbb{R}}$ *q ?x a* **by** *auto*
from ‹$j \leq 1$› **have** $1 - j \geq 0$ **by** *simp*
with ‹$1 - ?l \geq 0$› **and**
‹$(1 - ?l) * (1 - j) = ?k$› **and**
zero-le-mult-iff [*of* $1 - ?l$ $1 - j$]
have $?k \geq 0$ **by** *simp*
from ‹$j \geq 0$› **have** $1 - j \leq 1$ **by** *simp*
from ‹$?l \geq 0$› **have** $1 - ?l \leq 1$ **by** *simp*
with ‹$1 - j \leq 1$› **and**
‹$1 - j \geq 0$› **and**
mult-mono [*of* $1 - ?l$ 1 $1 - j$ 1] **and**
‹$(1 - ?l) * (1 - j) = ?k$›
have $?k \leq 1$ **by** *simp*
with ‹$?k \geq 0$› **and**
‹$?x - p = ?k *_R (b - p)$› **and**

real-euclid-B-def $[of\ p\ ?x\ b]$
have $B_{\mathbb{R}}\ p\ ?x\ b$ **by** *auto*
with $\langle B_{\mathbb{R}}\ q\ ?x\ a\rangle$ **show** *?thesis* **by** *auto*
qed }
thus $\forall\, a\ b\ c\ p\ q.\ B_{\mathbb{R}}\ a\ p\ c \wedge B_{\mathbb{R}}\ b\ q\ c \longrightarrow (\exists\, x.\ B_{\mathbb{R}}\ p\ x\ b \wedge B_{\mathbb{R}}\ q\ x\ a)$ **by** *auto*
{ fix $X\ Y$
assume $\exists\, a.\ \forall\, x\ y.\ x \in X \wedge y \in Y \longrightarrow B_{\mathbb{R}}\ a\ x\ y$
then obtain a **where** $\forall\, x\ y.\ x \in X \wedge y \in Y \longrightarrow B_{\mathbb{R}}\ a\ x\ y$ **by** *auto*
have $\exists\, b.\ \forall\, x\ y.\ x \in X \wedge y \in Y \longrightarrow B_{\mathbb{R}}\ x\ b\ y$
proof *cases*
assume $X \subseteq \{a\} \vee Y = \{\}$
let $?b = a$
{ fix $x\ y$
assume $x \in X$ **and** $y \in Y$
with $\langle X \subseteq \{a\} \vee Y = \{\}\rangle$ **have** $x = a$ **by** *auto*
from $\langle \forall\, x\ y.\ x \in X \wedge y \in Y \longrightarrow B_{\mathbb{R}}\ a\ x\ y\rangle$ **and** $\langle x \in X\rangle$ **and** $\langle y \in Y\rangle$
have $B_{\mathbb{R}}\ a\ x\ y$ **by** *simp*
with $\langle x = a\rangle$ **have** $B_{\mathbb{R}}\ x\ ?b\ y$ **by** *simp* **}**
hence $\forall\, x\ y.\ x \in X \wedge y \in Y \longrightarrow B_{\mathbb{R}}\ x\ ?b\ y$ **by** *simp*
thus *?thesis* **by** *auto*
next
assume $\neg(X \subseteq \{a\} \vee Y = \{\})$
hence $X - \{a\} \neq \{\}$ **and** $Y \neq \{\}$ **by** *auto*
from $\langle X - \{a\} \neq \{\}\rangle$ **obtain** c **where** $c \in X$ **and** $c \neq a$ **by** *auto*
from $\langle c \neq a\rangle$ **have** $c - a \neq 0$ **by** *simp*
{ fix y
assume $y \in Y$
with $\langle \forall\, x\ y.\ x \in X \wedge y \in Y \longrightarrow B_{\mathbb{R}}\ a\ x\ y\rangle$ **and** $\langle c \in X\rangle$
have $B_{\mathbb{R}}\ a\ c\ y$ **by** *simp*
with *real-euclid-B-def* $[of\ a\ c\ y]$
obtain l **where** $l \geq 0$ **and** $l \leq 1$ **and** $c - a = l *_R (y - a)$ **by** *auto*
from $\langle c - a = l *_R (y - a)\rangle$ **and** $\langle c - a \neq 0\rangle$ **have** $l \neq 0$ **by** *simp*
with $\langle l \geq 0\rangle$ **have** $l > 0$ **by** *simp*
with $\langle c - a = l *_R (y - a)\rangle$ **have** $y - a = (1/l) *_R (c - a)$ **by** *simp*
from $\langle l > 0\rangle$ **and** $\langle l \leq 1\rangle$ **have** $1/l \geq 1$ **by** *simp*
with $\langle y - a = (1/l) *_R (c - a)\rangle$
have $\exists j{\geq}1.\ y - a = j *_R (c - a)$ **by** *auto* **}**
note *ylemma* = *this*
from $\langle Y \neq \{\}\rangle$ **obtain** d **where** $d \in Y$ **by** *auto*
with *ylemma* $[of\ d]$
obtain jd **where** $jd \geq 1$ **and** $d - a = jd *_R (c - a)$ **by** *auto*
{ fix x
assume $x \in X$
with $\langle \forall\, x\ y.\ x \in X \wedge y \in Y \longrightarrow B_{\mathbb{R}}\ a\ x\ y\rangle$ **and** $\langle d \in Y\rangle$
have $B_{\mathbb{R}}\ a\ x\ d$ **by** *simp*
with *real-euclid-B-def* $[of\ a\ x\ d]$
obtain l **where** $l \geq 0$ **and** $x - a = l *_R (d - a)$ **by** *auto*
from $\langle x - a = l *_R (d - a)\rangle$ **and**
$\langle d - a = jd *_R (c - a)\rangle$ **and**

scaleR-scaleR
have $x - a = (l * jd) *_R (c - a)$ **by** *simp*
hence $\exists i.\ x - a = i *_R (c - a)$ **by** *auto* **}**
note *xlemma* = *this*
let $?S = \{j.\ j \geq 1 \wedge (\exists y{\in}Y.\ y - a = j *_R (c - a))\}$
from $\langle d \in Y\rangle$ **and** $\langle jd \geq 1\rangle$ **and** $\langle d - a = jd *_R (c - a)\rangle$
have $?S \neq \{\}$ **by** *auto*
let $?k = Inf\ ?S$
let $?b = ?k *_R c + (1 - ?k) *_R a$
from *rearrange-real-euclid-B* [*of* $?b$ a $?k$ c]
have $?b - a = ?k *_R (c - a)$ **by** *simp*
{ fix x y
assume $x \in X$ **and** $y \in Y$
from *xlemma* [*of* x] **and** $\langle x \in X\rangle$
obtain i **where** $x - a = i *_R (c - a)$ **by** *auto*
from *ylemma* [*of* y] **and** $\langle y \in Y\rangle$
obtain j **where** $j \geq 1$ **and** $y - a = j *_R (c - a)$ **by** *auto*
with $\langle y \in Y\rangle$ **have** $j \in ?S$ **by** *auto*
then have $?k \leq j$ **by** (*auto intro*: *cInf-lower*)
{ fix h
assume $h \in ?S$
hence $h \geq 1$ **by** *simp*
from $\langle h \in ?S\rangle$
obtain z **where** $z \in Y$ **and** $z - a = h *_R (c - a)$ **by** *auto*
from $\langle \forall x\, y.\ x \in X \wedge y \in Y \longrightarrow B_{\mathbb{R}}\ a\ x\ y\rangle$ **and** $\langle x \in X\rangle$ **and** $\langle z \in Y\rangle$
have $B_{\mathbb{R}}\ a\ x\ z$ **by** *simp*
with *real-euclid-B-def* [*of* a x z]
obtain l **where** $l \leq 1$ **and** $x - a = l *_R (z - a)$ **by** *auto*
with $\langle z - a = h *_R (c - a)\rangle$ **and** *scaleR-scaleR*
have $x - a = (l * h) *_R (c - a)$ **by** *simp*
with $\langle x - a = i *_R (c - a)\rangle$
have $i *_R (c - a) = (l * h) *_R (c - a)$ **by** *auto*
with *scaleR-cancel-right* **and** $\langle c - a \neq 0\rangle$ **have** $i = l * h$ **by** *blast*
with $\langle l \leq 1\rangle$ **and** $\langle h \geq 1\rangle$ **have** $i \leq h$ **by** *simp* **}**
with $\langle ?S \neq \{\}\rangle$ **and** *cInf-greatest* [*of* $?S$] **have** $i \leq ?k$ **by** *simp*
have $y - x = (y - a) - (x - a)$ **by** *simp*
with $\langle y - a = j *_R (c - a)\rangle$ **and** $\langle x - a = i *_R (c - a)\rangle$
have $y - x = j *_R (c - a) - i *_R (c - a)$ **by** *simp*
with *scaleR-left-diff-distrib* [*of* j i $c - a$]
have $y - x = (j - i) *_R (c - a)$ **by** *simp*
have $?b - x = (?b - a) - (x - a)$ **by** *simp*
with $\langle ?b - a = ?k *_R (c - a)\rangle$ **and** $\langle x - a = i *_R (c - a)\rangle$
have $?b - x = ?k *_R (c - a) - i *_R (c - a)$ **by** *simp*
with *scaleR-left-diff-distrib* [*of* $?k$ i $c - a$]
have $?b - x = (?k - i) *_R (c - a)$ **by** *simp*
have $B_{\mathbb{R}}\ x\ ?b\ y$
proof *cases*
assume $i = j$
with $\langle i \leq ?k\rangle$ **and** $\langle ?k \leq j\rangle$ **have** $?k = i$ **by** *simp*

with ‹*?b* − *x* = (*?k* − *i*) $*_R$ (*c* − *a*)› **have** *?b* − *x* = *0* **by** *simp*
hence *?b* − *x* = *0* $*_R$ (*y* − *x*) **by** *simp*
with *real-euclid-B-def* [*of x ?b y*] **show** $B_{\mathbb{R}}$ *x ?b y* **by** *auto*
next
assume *i* ≠ *j*
with ‹*i* ≤ *?k*› **and** ‹*?k* ≤ *j*› **have** *j* − *i* > *0* **by** *simp*
with ‹*y* − *x* = (*j* − *i*) $*_R$ (*c* − *a*)› **and** *scaleR-scaleR*
have *c* − *a* = (*1* / (*j* − *i*)) $*_R$ (*y* − *x*) **by** *simp*
with ‹*?b* − *x* = (*?k* − *i*) $*_R$ (*c* − *a*)› **and** *scaleR-scaleR*
have *?b* − *x* = ((*?k* − *i*) / (*j* − *i*)) $*_R$ (*y* − *x*) **by** *simp*
let *?l* = (*?k* − *i*) / (*j* − *i*)
from ‹*?k* ≤ *j*› **have** *?k* − *i* ≤ *j* − *i* **by** *simp*
with ‹*j* − *i* > *0*› **have** *?l* ≤ *1* **by** *simp*
from ‹*i* ≤ *?k*› **and** ‹*j* − *i* > *0*› **and** *pos-le-divide-eq* [*of j* − *i 0 ?k* − *i*]
have *?l* ≥ *0* **by** *simp*
with *real-euclid-B-def* [*of x ?b y*] **and**
‹*?l* ≤ *1*› **and**
‹*?b* − *x* = *?l* $*_R$ (*y* − *x*)›
show $B_{\mathbb{R}}$ *x ?b y* **by** *auto*
qed **}**
thus ∃ *b*. ∀ *x y*. *x* ∈ *X* ∧ *y* ∈ *Y* ⟶ $B_{\mathbb{R}}$ *x b y* **by** *auto*
qed **}**
thus ∀ *X Y*. (∃ *a*. ∀ *x y*. *x* ∈ *X* ∧ *y* ∈ *Y* ⟶ $B_{\mathbb{R}}$ *a x y*) ⟶
(∃ *b*. ∀ *x y*. *x* ∈ *X* ∧ *y* ∈ *Y* ⟶ $B_{\mathbb{R}}$ *x b y*)
by *auto*
qed

4.3 Real Euclidean space satisfies the Euclidean axiom

lemma *rearrange-real-euclid-B-2*:
fixes *a b c* :: *real*ˆ(′*n*::*finite*)
assumes *l* ≠ *0*
shows *b* − *a* = *l* $*_R$ (*c* − *a*) ⟷ *c* = (*1*/*l*) $*_R$ *b* + (*1* − *1*/*l*) $*_R$ *a*
proof
from *scaleR-right-diff-distrib* [*of 1/l b a*]
have (*1*/*l*) $*_R$ (*b* − *a*) = *c* − *a* ⟷ (*1*/*l*) $*_R$ *b* − (*1*/*l*) $*_R$ *a* + *a* = *c* **by** *auto*
also with *scaleR-left-diff-distrib* [*of 1 1/l a*]
have . . . ⟷ *c* = (*1*/*l*) $*_R$ *b* + (*1* − *1*/*l*) $*_R$ *a* **by** *auto*
finally have *eq*:
(*1*/*l*) $*_R$ (*b* − *a*) = *c* − *a* ⟷ *c* = (*1*/*l*) $*_R$ *b* + (*1* − *1*/*l*) $*_R$ *a* **.**
{ assume *b* − *a* = *l* $*_R$ (*c* − *a*)
with ‹*l* ≠ *0*› **have** (*1*/*l*) $*_R$ (*b* − *a*) = *c* − *a* **by** *simp*
with *eq* **show** *c* = (*1*/*l*) $*_R$ *b* + (*1* − *1*/*l*) $*_R$ *a* **.. }**
{ assume *c* = (*1*/*l*) $*_R$ *b* + (*1* − *1*/*l*) $*_R$ *a*
with *eq* **have** (*1*/*l*) $*_R$ (*b* − *a*) = *c* − *a* **..**
hence *l* $*_R$ (*1*/*l*) $*_R$ (*b* − *a*) = *l* $*_R$ (*c* − *a*) **by** *simp*
with ‹*l* ≠ *0*› **show** *b* − *a* = *l* $*_R$ (*c* − *a*) **by** *simp* **}**
qed

interpretation *real-euclid*: *tarski-space real-euclid-C real-euclid-B*
proof
  { fix $a\ b\ c\ d\ t$
    assume $B_{\mathbb{R}}\ a\ d\ t$ **and** $B_{\mathbb{R}}\ b\ d\ c$ **and** $a \neq d$
    from *real-euclid-B-def* [*of* $a\ d\ t$] **and** ‹$B_{\mathbb{R}}\ a\ d\ t$›
      obtain j **where** $j \geq 0$ **and** $j \leq 1$ **and** $d - a = j *_R (t - a)$ **by** *auto*
    from ‹$d - a = j *_R (t - a)$› **and** ‹$a \neq d$› **have** $j \neq 0$ **by** *auto*
    with ‹$d - a = j *_R (t - a)$› **and** *rearrange-real-euclid-B-2*
      have $t = (1/j) *_R d + (1 - 1/j) *_R a$ **by** *auto*
    let $?x = (1/j) *_R b + (1 - 1/j) *_R a$
    let $?y = (1/j) *_R c + (1 - 1/j) *_R a$
    from ‹$j \neq 0$› **and** *rearrange-real-euclid-B-2* **have**
      $b - a = j *_R (?x - a)$ **and** $c - a = j *_R (?y - a)$ **by** *auto*
    with *real-euclid-B-def* **and** ‹$j \geq 0$› **and** ‹$j \leq 1$› **have**
      $B_{\mathbb{R}}\ a\ b\ ?x$ **and** $B_{\mathbb{R}}\ a\ c\ ?y$ **by** *auto*
    from *real-euclid-B-def* **and** ‹$B_{\mathbb{R}}\ b\ d\ c$› **obtain** k **where**
      $k \geq 0$ **and** $k \leq 1$ **and** $d - b = k *_R (c - b)$ **by** *blast*
    from ‹$t = (1/j) *_R d + (1 - 1/j) *_R a$› **have**
      $t - ?x = (1/j) *_R d - (1/j) *_R b$ **by** *simp*
    also from *scaleR-right-diff-distrib* [*of* $1/j\ d\ b$] **have**
      $\ldots = (1/j) *_R (d - b)$ **by** *simp*
    also from ‹$d - b = k *_R (c - b)$› **have**
      $\ldots = k *_R (1/j) *_R (c - b)$ **by** *simp*
    also from *scaleR-right-diff-distrib* [*of* $1/j\ c\ b$] **have**
      $\ldots = k *_R (?y - ?x)$ **by** *simp*
    finally have $t - ?x = k *_R (?y - ?x)$ **.**
    with *real-euclid-B-def* **and** ‹$k \geq 0$› **and** ‹$k \leq 1$› **have** $B_{\mathbb{R}}\ ?x\ t\ ?y$ **by** *blast*
    with ‹$B_{\mathbb{R}}\ a\ b\ ?x$› **and** ‹$B_{\mathbb{R}}\ a\ c\ ?y$› **have**
      $\exists\, x\ y.\ B_{\mathbb{R}}\ a\ b\ x \wedge B_{\mathbb{R}}\ a\ c\ y \wedge B_{\mathbb{R}}\ x\ t\ y$ **by** *auto* **}**
  thus $\forall\, a\ b\ c\ d\ t.\ B_{\mathbb{R}}\ a\ d\ t \wedge B_{\mathbb{R}}\ b\ d\ c \wedge a \neq d \longrightarrow$
          $(\exists\, x\ y.\ B_{\mathbb{R}}\ a\ b\ x \wedge B_{\mathbb{R}}\ a\ c\ y \wedge B_{\mathbb{R}}\ x\ t\ y)$
    by *auto*
qed

4.4 The real Euclidean plane

lemma *Col-dep2*:
  real-euclid.Col $a\ b\ c \longleftrightarrow$ *dep2* $(b - a)\ (c - a)$
proof −
  from *real-euclid.Col-def* **have**
    real-euclid.Col $a\ b\ c \longleftrightarrow B_{\mathbb{R}}\ a\ b\ c \vee B_{\mathbb{R}}\ b\ c\ a \vee B_{\mathbb{R}}\ c\ a\ b$ **by** *auto*
  moreover from *dep2-def* **have**
    dep2 $(b - a)\ (c - a) \longleftrightarrow (\exists\, w\ r\ s.\ b - a = r *_R w \wedge c - a = s *_R w)$
    by *auto*
  moreover
  { assume $B_{\mathbb{R}}\ a\ b\ c \vee B_{\mathbb{R}}\ b\ c\ a \vee B_{\mathbb{R}}\ c\ a\ b$
    moreover
    { assume $B_{\mathbb{R}}\ a\ b\ c$
      with *real-euclid-B-def* **obtain** l **where** $b - a = l *_R (c - a)$ **by** *blast*

moreover have $c - a = \mathit{1} *_R (c - a)$ **by** *simp*
ultimately have $\exists\, w\ r\ s.\ b - a = r *_R w \wedge c - a = s *_R w$ **by** *blast* **}**
moreover
{ assume $B_{\mathbb{R}}\ b\ c\ a$
with *real-euclid-B-def* **obtain** l **where** $c - b = l *_R (a - b)$ **by** *blast*
moreover have $c - a = (c - b) - (a - b)$ **by** *simp*
ultimately have $c - a = l *_R (a - b) - (a - b)$ **by** *simp*
with *scaleR-left-diff-distrib* [*of* l *1* $a - b$] **have**
$c - a = (l - \mathit{1}) *_R (a - b)$ **by** *simp*
moreover from *scaleR-minus-left* [*of* *1* $a - b$] **have**
$b - a = (-\mathit{1}) *_R (a - b)$ **by** *simp*
ultimately have $\exists\, w\ r\ s.\ b - a = r *_R w \wedge c - a = s *_R w$ **by** *blast* **}**
moreover
{ assume $B_{\mathbb{R}}\ c\ a\ b$
with *real-euclid-B-def* **obtain** l **where** $a - c = l *_R (b - c)$ **by** *blast*
moreover have $c - a = -(a - c)$ **by** *simp*
ultimately have $c - a = -(l *_R (b - c))$ **by** *simp*
with *scaleR-minus-left* **have** $c - a = (-l) *_R (b - c)$ **by** *simp*
moreover have $b - a = (b - c) + (c - a)$ **by** *simp*
ultimately have $b - a = \mathit{1} *_R (b - c) + (-l) *_R (b - c)$ **by** *simp*
with *scaleR-left-distrib* [*of* *1* $-l$ $b - c$] **have**
$b - a = (\mathit{1} + (-l)) *_R (b - c)$ **by** *simp*
with ‹$c - a = (-l) *_R (b - c)$› **have**
$\exists\, w\ r\ s.\ b - a = r *_R w \wedge c - a = s *_R w$ **by** *blast* **}**
ultimately have $\exists\, w\ r\ s.\ b - a = r *_R w \wedge c - a = s *_R w$ **by** *auto* **}**
moreover
{ assume $\exists\, w\ r\ s.\ b - a = r *_R w \wedge c - a = s *_R w$
then obtain $w\ r\ s$ **where** $b - a = r *_R w$ **and** $c - a = s *_R w$ **by** *auto*
have $B_{\mathbb{R}}\ a\ b\ c \vee B_{\mathbb{R}}\ b\ c\ a \vee B_{\mathbb{R}}\ c\ a\ b$
proof *cases*
assume $s = \mathit{0}$
with ‹$c - a = s *_R w$› **have** $a = c$ **by** *simp*
with *real-euclid.th3-1* **have** $B_{\mathbb{R}}\ b\ c\ a$ **by** *simp*
thus *?thesis* **by** *simp*
next
assume $s \neq \mathit{0}$
with ‹$c - a = s *_R w$› **have** $w = (\mathit{1}/s) *_R (c - a)$ **by** *simp*
with ‹$b - a = r *_R w$› **have** $b - a = (r/s) *_R (c - a)$ **by** *simp*
have $r/s < \mathit{0} \vee (r/s \geq \mathit{0} \wedge r/s \leq \mathit{1}) \vee r/s > \mathit{1}$ **by** *arith*
moreover
{ assume $r/s \geq \mathit{0} \wedge r/s \leq \mathit{1}$
with *real-euclid-B-def* **and** ‹$b - a = (r/s) *_R (c - a)$› **have** $B_{\mathbb{R}}\ a\ b\ c$
by *auto*
hence *?thesis* **by** *simp* **}**
moreover
{ assume $r/s > \mathit{1}$
with ‹$b - a = (r/s) *_R (c - a)$› **have** $c - a = (s/r) *_R (b - a)$ **by** *auto*
from ‹$r/s > \mathit{1}$› **and** *le-imp-inverse-le* [*of* *1* r/s] **have**
$s/r \leq \mathit{1}$ **by** *simp*

```
      from ‹r/s > 1› and inverse-positive-iff-positive [of r/s] have
        s/r ≥ 0 by simp
      with real-euclid-B-def
        and ‹c − a = (s/r) *R (b − a)›
        and ‹s/r ≤ 1›
      have BIR a c b by auto
      with real-euclid.th3-2 have BIR b c a by auto
      hence ?thesis by simp }
    moreover
    { assume r/s < 0
      have b − c = (b − a) + (a − c) by simp
      with ‹b − a = (r/s) *R (c − a)› have
        b − c = (r/s) *R (c − a) + (a − c) by simp
      have c − a = −(a − c) by simp
      with scaleR-minus-right [of r/s a − c] have
        (r/s) *R (c − a) = −((r/s) *R (a − c)) by arith
      with ‹b − c = (r/s) *R (c − a) + (a − c)› have
        b − c = −(r/s) *R (a − c) + (a − c) by simp
      with scaleR-left-distrib [of −(r/s) 1 a − c] have
        b − c = (−(r/s) + 1) *R (a − c) by simp
      moreover from ‹r/s < 0› have −(r/s) + 1 > 1 by simp
      ultimately have a − c = (1 / (−(r/s) + 1)) *R (b − c) by auto
      let ?l = 1 / (−(r/s) + 1)
      from ‹−(r/s) + 1 > 1› and le-imp-inverse-le [of 1 −(r/s) + 1] have
        ?l ≤ 1 by simp
      from ‹−(r/s) + 1 > 1›
        and inverse-positive-iff-positive [of −(r/s) + 1]
      have
        ?l ≥ 0 by simp
      with real-euclid-B-def and ‹?l ≤ 1› and ‹a − c = ?l *R (b − c)› have
        BIR c a b by blast
      hence ?thesis by simp }
    ultimately show ?thesis by auto
  qed }
  ultimately show ?thesis by blast
qed

lemma non-Col-example:
  ¬(real-euclid.Col 0 (vector [1/2,0] :: real^2) (vector [0,1/2]))
  (is ¬ (real-euclid.Col ?a ?b ?c))
proof −
  { assume dep2 (?b − ?a) (?c − ?a)
    with dep2-def [of ?b − ?a ?c − ?a] obtain w r s where
      ?b − ?a = r *R w and ?c − ?a = s *R w by auto
    have ?b$1 = 1/2 by simp
    with ‹?b − ?a = r *R w› have r * (w$1) = 1/2 by simp
    hence w$1 ≠ 0 by auto
    have ?c$1 = 0 by simp
    with ‹?c − ?a = s *R w› have s * (w$1) = 0 by simp
```

```
    with ‹w$1 ≠ 0› have s = 0 by simp
    have ?c$2 = 1/2 by simp
    with ‹?c − ?a = s *R w› have s * (w$2) = 1/2 by simp
    with ‹s = 0› have False by simp }
  hence ¬(dep2 (?b − ?a) (?c − ?a)) by auto
  with Col-dep2 show ¬(real-euclid.Col ?a ?b ?c) by blast
qed

interpretation real-euclid:
  tarski real-euclid-C::([real^2, real^2, real^2, real^2] ⇒ bool) real-euclid-B
proof
  { let ?a = 0 :: real^2
    let ?b = vector [1/2, 0] :: real^2
    let ?c = vector [0, 1/2] :: real^2
    from non-Col-example and real-euclid.Col-def have
      ¬ B_ℝ ?a ?b ?c ∧ ¬ B_ℝ ?b ?c ?a ∧ ¬ B_ℝ ?c ?a ?b by auto }
  thus ∃ a b c :: real^2. ¬ B_ℝ a b c ∧ ¬ B_ℝ b c a ∧ ¬ B_ℝ c a b
    by auto
  { fix p q a b c :: real^2
    assume p ≠ q and a p ≡_ℝ a q and b p ≡_ℝ b q and c p ≡_ℝ c q
    let ?m = (1/2) *R (p + q)
    from scaleR-right-distrib [of 1/2 p q] and
      scaleR-right-diff-distrib [of 1/2 q p] and
      scaleR-left-diff-distrib [of 1/2 1 p]
    have ?m − p = (1/2) *R (q − p) by simp
    with ‹p ≠ q› have ?m − p ≠ 0 by simp
    from scaleR-right-distrib [of 1/2 p q] and
      scaleR-right-diff-distrib [of 1/2 p q] and
      scaleR-left-diff-distrib [of 1/2 1 q]
    have ?m − q = (1/2) *R (p − q) by simp
    with ‹?m − p = (1/2) *R (q − p)›
      and scaleR-minus-right [of 1/2 q − p]
    have ?m − q = −(?m − p) by simp
    with norm-minus-cancel [of ?m − p] have
      (norm (?m − q))^2 = (norm (?m − p))^2 by (simp only: norm-minus-cancel)
    { fix d
      assume d p ≡_ℝ d q
      hence (norm (d − p))^2 = (norm (d − q))^2 by simp
      have (d − ?m) · (?m − p) = 0
      proof −
        have d + (−q) = d − q by simp
        have d + (−p) = d − p by simp
        with dot-norm [of d − ?m ?m − p] have
          (d − ?m) · (?m − p) =
          ((norm (d − p))^2 − (norm (d − ?m))^2 − (norm(?m − p))^2) / 2
          by simp
        also from ‹(norm (d − p))^2 = (norm (d − q))^2›
          and ‹(norm (?m − q))^2 = (norm (?m − p))^2›
        have
```

$\ldots = ((norm\ (d - ?m))^2 - (norm\ (d - ?m))^2 - (norm(?m - q))^2)\ /\ 2$
by *simp*
also from *dot-norm* [*of* $d - ?m\ ?m - q$]
and ‹$d + (-q) = d - q$›
have
$\ldots = (d - ?m) \cdot (?m - q)$ **by** *simp*
also from *inner-minus-right* [*of* $d - ?m\ ?m - p$]
and ‹$?m - q = -(?m - p)$›
have
$\ldots = -((d - ?m) \cdot (?m - p))$ **by** (*simp only*: *inner-minus-left*)
finally have $(d - ?m) \cdot (?m - p) = -((d - ?m) \cdot (?m - p))$ **.**
thus $(d - ?m) \cdot (?m - p) = 0$ **by** *arith*
qed }
note *m-lemma* = *this*
with ‹$a\ p \equiv_{\mathbb{R}} a\ q$› **have** $(a - ?m) \cdot (?m - p) = 0$ **by** *simp*
{ fix d
assume $d\ p \equiv_{\mathbb{R}} d\ q$
with *m-lemma* **have** $(d - ?m) \cdot (?m - p) = 0$ **by** *simp*
with *dot-left-diff-distrib* [*of* $d - ?m\ a - ?m\ ?m - p$]
and ‹$(a - ?m) \cdot (?m - p) = 0$›
have $(d - a) \cdot (?m - p) = 0$ **by** (*simp add*: *inner-diff-left inner-diff-right*) **}**
with ‹$b\ p \equiv_{\mathbb{R}} b\ q$› **and** ‹$c\ p \equiv_{\mathbb{R}} c\ q$› **have**
$(b - a) \cdot (?m - p) = 0$ **and** $(c - a) \cdot (?m - p) = 0$ **by** *simp+*
with *real2-orthogonal-dep2* **and** ‹$?m - p \neq 0$› **have** *dep2* $(b - a)\ (c - a)$
by *blast*
with *Col-dep2* **have** *real-euclid.Col* $a\ b\ c$ **by** *auto*
with *real-euclid.Col-def* **have** $B_{\mathbb{R}}\ a\ b\ c \vee B_{\mathbb{R}}\ b\ c\ a \vee B_{\mathbb{R}}\ c\ a\ b$ **by** *auto* **}**
thus $\forall\, p\ q\ a\ b\ c :: real\hat{}2.$
$p \neq q \wedge a\ p \equiv_{\mathbb{R}} a\ q \wedge b\ p \equiv_{\mathbb{R}} b\ q \wedge c\ p \equiv_{\mathbb{R}} c\ q \longrightarrow$
$B_{\mathbb{R}}\ a\ b\ c \vee B_{\mathbb{R}}\ b\ c\ a \vee B_{\mathbb{R}}\ c\ a\ b$
by *blast*
qed

4.5 Special cases of theorems of Tarski's geometry

lemma *real-euclid-B-disjunction*:
assumes $l \geq 0$ **and** $b - a = l *_R (c - a)$
shows $B_{\mathbb{R}}\ a\ b\ c \vee B_{\mathbb{R}}\ a\ c\ b$
proof *cases*
assume $l \leq 1$
with ‹$l \geq 0$› **and** ‹$b - a = l *_R (c - a)$›
have $B_{\mathbb{R}}\ a\ b\ c$ **by** (*unfold real-euclid-B-def*) (*simp add*: *exI* [*of* - l])
thus $B_{\mathbb{R}}\ a\ b\ c \vee B_{\mathbb{R}}\ a\ c\ b$ **..**
next
assume $\neg\ (l \leq 1)$
hence $1/l \leq 1$ **by** *simp*

from ‹$l \geq 0$› **have** $1/l \geq 0$ **by** *simp*

from ‹$b - a = l *_R (c - a)$›
have $(1/l) *_R (b - a) = (1/l) *_R (l *_R (c - a))$ **by** *simp*
with ‹$\neg (l \le 1)$› **have** $c - a = (1/l) *_R (b - a)$ **by** *simp*
with ‹$1/l \ge 0$› **and** ‹$1/l \le 1$›
have $B_{\mathbb{R}}\ a\ c\ b$ **by** (*unfold real-euclid-B-def*) (*simp add*: *exI* [*of* - $1/l$])
thus $B_{\mathbb{R}}\ a\ b\ c \vee B_{\mathbb{R}}\ a\ c\ b$ **..**
qed

The following are true in Tarski's geometry, but to prove this would require much more development of it, so only the Euclidean case is proven here.

theorem *real-euclid-th5-1*:
assumes $a \neq b$ **and** $B_{\mathbb{R}}\ a\ b\ c$ **and** $B_{\mathbb{R}}\ a\ b\ d$
shows $B_{\mathbb{R}}\ a\ c\ d \vee B_{\mathbb{R}}\ a\ d\ c$
proof −
from ‹$B_{\mathbb{R}}\ a\ b\ c$› **and** ‹$B_{\mathbb{R}}\ a\ b\ d$›
obtain l **and** m **where** $l \ge 0$ **and** $b - a = l *_R (c - a)$
and $m \ge 0$ **and** $b - a = m *_R (d - a)$
by (*unfold real-euclid-B-def*) *auto*
from ‹$b - a = m *_R (d - a)$› **and** ‹$a \neq b$› **have** $m \neq 0$ **by** *auto*

from ‹$l \ge 0$› **and** ‹$m \ge 0$› **have** $l/m \ge 0$ **by** (*simp add*: *zero-le-divide-iff*)

from ‹$b - a = l *_R (c - a)$› **and** ‹$b - a = m *_R (d - a)$›
have $m *_R (d - a) = l *_R (c - a)$ **by** *simp*
hence $(1/m) *_R (m *_R (d - a)) = (1/m) *_R (l *_R (c - a))$ **by** *simp*
with ‹$m \neq 0$› **have** $d - a = (l/m) *_R (c - a)$ **by** *simp*
with ‹$l/m \ge 0$› **and** *real-euclid-B-disjunction*
show $B_{\mathbb{R}}\ a\ c\ d \vee B_{\mathbb{R}}\ a\ d\ c$ **by** *auto*
qed

theorem *real-euclid-th5-3*:
assumes $B_{\mathbb{R}}\ a\ b\ d$ **and** $B_{\mathbb{R}}\ a\ c\ d$
shows $B_{\mathbb{R}}\ a\ b\ c \vee B_{\mathbb{R}}\ a\ c\ b$
proof −
from ‹$B_{\mathbb{R}}\ a\ b\ d$› **and** ‹$B_{\mathbb{R}}\ a\ c\ d$›
obtain l **and** m **where** $l \ge 0$ **and** $b - a = l *_R (d - a)$
and $m \ge 0$ **and** $c - a = m *_R (d - a)$
by (*unfold real-euclid-B-def*) *auto*

show $B_{\mathbb{R}}\ a\ b\ c \vee B_{\mathbb{R}}\ a\ c\ b$
proof *cases*
assume $l = 0$
with ‹$b - a = l *_R (d - a)$› **have** $b - a = l *_R (c - a)$ **by** *simp*
with ‹$l = 0$›
have $B_{\mathbb{R}}\ a\ b\ c$ **by** (*unfold real-euclid-B-def*) (*simp add*: *exI* [*of* - l])
thus $B_{\mathbb{R}}\ a\ b\ c \vee B_{\mathbb{R}}\ a\ c\ b$ **..**
next
assume $l \neq 0$

from ‹$l \geq 0$› **and** ‹$m \geq 0$› **have** $m/l \geq 0$ **by** (*simp add*: *zero-le-divide-iff*)

from ‹$b - a = l *_R (d - a)$›
have $(1/l) *_R (b - a) = (1/l) *_R (l *_R (d - a))$ **by** *simp*
with ‹$l \neq 0$› **have** $d - a = (1/l) *_R (b - a)$ **by** *simp*
with ‹$c - a = m *_R (d - a)$› **have** $c - a = (m/l) *_R (b - a)$ **by** *simp*
with ‹$m/l \geq 0$› **and** *real-euclid-B-disjunction*
show $B_{\mathbb{R}}\ a\ b\ c \vee B_{\mathbb{R}}\ a\ c\ b$ **by** *auto*
qed
qed

end

5 Linear algebra

theory *Linear-Algebra2*
imports *Miscellany*
begin

lemma *exhaust-4*:
fixes $x :: 4$
shows $x = 1 \vee x = 2 \vee x = 3 \vee x = 4$
proof (*induct x*)
case (*of-int z*)
hence $0 \leq z$ **and** $z < 4$ **by** *simp-all*
hence $z = 0 \vee z = 1 \vee z = 2 \vee z = 3$ **by** *arith*
thus *?case* **by** *auto*
qed

lemma *forall-4*: $(\forall\ i::4.\ P\ i) \longleftrightarrow P\ 1 \wedge P\ 2 \wedge P\ 3 \wedge P\ 4$
by (*metis exhaust-4*)

lemma *UNIV-4*: $(UNIV::(4\ set)) = \{1, 2, 3, 4\}$
using *exhaust-4*
by *auto*

lemma *vector-4*:
fixes $w :: 'a::zero$
shows $(vector\ [w, x, y, z] :: 'a\hat{\ }4)\$1 = w$
and $(vector\ [w, x, y, z] :: 'a\hat{\ }4)\$2 = x$
and $(vector\ [w, x, y, z] :: 'a\hat{\ }4)\$3 = y$
and $(vector\ [w, x, y, z] :: 'a\hat{\ }4)\$4 = z$
unfolding *vector-def*
by *simp-all*

definition
$is\text{-}basis :: (real\hat{\ }'n)\ set \Rightarrow bool$ **where**
$is\text{-}basis\ S \triangleq independent\ S \wedge span\ S = UNIV$

lemma *card-finite*:
  assumes *card S* = *CARD*(*'n*::*finite*)
  shows *finite S*
proof –
  from ‹*card S* = *CARD*(*'n*)› **have** *card S* ≠ *0* **by** *simp*
  with *card-eq-0-iff* [*of S*] **show** *finite S* **by** *simp*
qed

lemma *independent-is-basis*:
  fixes *B* :: (*real*ˆ*'n*) *set*
  shows *independent B* ∧ *card B* = *CARD*(*'n*) ⟷ *is-basis B*
proof
  assume *L*: *independent B* ∧ *card B* = *CARD*(*'n*)
  then have *card* (*Basis*::(*real*ˆ*'n*) *set*) = *card B*
    by *simp*
  with *L* **show** *is-basis B*
      by (*metis* (*no-types*) *card-eq-dim dim-UNIV independent-bound is-basis-def subset-antisym top-greatest*)
next
  assume *is-basis B*
  then show *independent B* ∧ *card B* = *CARD*(*'n*)
   by (*metis DIM-cart DIM-real basis-card-eq-dim dim-UNIV is-basis-def mult.right-neutral top.extremum*)
qed

lemma *basis-finite*:
  fixes *B* :: (*real*ˆ*'n*) *set*
  assumes *is-basis B*
  shows *finite B*
proof –
  from *independent-is-basis* [*of B*] **and** ‹*is-basis B*› **have** *card B* = *CARD*(*'n*)
    by *simp*
  with *card-finite* [*of B*, **where** *'n* = *'n*] **show** *finite B* **by** *simp*
qed

lemma *basis-expand*:
  assumes *is-basis B*
  shows ∃ *c*. *v* = ($\sum$ *w*∈*B*. (*c w*) $*_R$ *w*)
proof –
  from ‹*is-basis B*› **have** *v* ∈ *span B* **unfolding** *is-basis-def* **by** *simp*
  from *basis-finite* [*of B*] **and** ‹*is-basis B*› **have** *finite B* **by** *simp*
  with *span-finite* [*of B*] **and** ‹*v* ∈ *span B*›
  show ∃ *c*. *v* = ($\sum$ *w*∈*B*. (*c w*) $*_R$ *w*) **by** (*simp add*: *scalar-equiv*) *auto*
qed

lemma *not-span-independent-insert*:
  fixes *v* :: (*'a*::*real-vector*)ˆ*'n*
  assumes *independent S* **and** *v* ∉ *span S*

shows *independent* (*insert v S*)
by (*simp add*: *assms independent-insert*)

lemma *orthogonal-sum*:
fixes v :: *real*^'*n*
assumes $\bigwedge w.\ w \in S \Longrightarrow$ *orthogonal v w*
shows *orthogonal v* ($\sum w \in S.\ c\ w *s\ w$)
by (*metis* (*no-types*, *lifting*) *assms orthogonal-clauses*(*1*,*2*) *orthogonal-rvsum scalar-equiv sum.infinite*)

lemma *orthogonal-self-eq-0*:
fixes v :: ('*a*::*real-inner*)^'*n*
assumes *orthogonal v v*
shows $v = 0$
using *inner-eq-zero-iff* [*of v*] **and** *assms*
unfolding *orthogonal-def*
by *simp*

lemma *orthogonal-in-span-eq-0*:
fixes v :: *real*^'*n*
assumes $v \in$ *span S* **and** $\bigwedge w.\ w \in S \Longrightarrow$ *orthogonal v w*
shows $v = 0$
using *assms orthogonal-self orthogonal-to-span* **by** *blast*

lemma *orthogonal-independent*:
fixes v :: *real*^'*n*
assumes *independent S* **and** $v \neq 0$ **and** $\bigwedge w.\ w \in S \Longrightarrow$ *orthogonal v w*
shows *independent* (*insert v S*)
using *assms not-span-independent-insert orthogonal-in-span-eq-0* **by** *blast*

lemma *dot-scaleR-mult*:
shows $(k *_R a) \cdot b = k * (a \cdot b)$ **and** $a \cdot (k *_R b) = k * (a \cdot b)$
by *auto*

lemma *dependent-explicit-finite*:
fixes S :: (('*a*::{*real-vector*,*field*})^'*n*) *set*
assumes *finite S*
shows *dependent S* $\longleftrightarrow (\exists\ u.\ (\exists\ v \in S.\ u\ v \neq 0) \wedge (\sum\ v \in S.\ u\ v *_R v) = 0)$
by (*simp add*: *assms dependent-finite*)

lemma *dependent-explicit-2*:
fixes $v\ w$ :: ('*a*::{*field*,*real-vector*})^'*n*
assumes $v \neq w$
shows *dependent* $\{v, w\} \longleftrightarrow (\exists\ i\ j.\ (i \neq 0 \vee j \neq 0) \wedge i *_R v + j *_R w = 0)$
proof
let $?S = \{v, w\}$
have *finite ?S* **by** *simp*

{ **assume** *dependent ?S*

with *dependent-explicit-finite* [*of ?S*] **and** ‹*finite ?S*› **and** ‹$v \neq w$›
show $\exists\ i\ j.\ (i \neq 0 \lor j \neq 0) \land i *_R v + j *_R w = 0$ **by** *auto* **}**

{ assume $\exists\ i\ j.\ (i \neq 0 \lor j \neq 0) \land i *_R v + j *_R w = 0$
then obtain i **and** j **where** $i \neq 0 \lor j \neq 0$ **and** $i *_R v + j *_R w = 0$ **by** *auto*
let $?u = \lambda\ x.\ if\ x = v\ then\ i\ else\ j$
from ‹$i \neq 0 \lor j \neq 0$› **and** ‹$v \neq w$› **have** $\exists\ x{\in}?S.\ ?u\ x \neq 0$ **by** *simp*
from ‹$i *_R v + j *_R w = 0$› **and** ‹$v \neq w$›
have $(\sum\ x{\in}?S.\ ?u\ x *_R x) = 0$ **by** *simp*
with *dependent-explicit-finite* [*of ?S*]
and ‹*finite ?S*› **and** ‹$\exists\ x{\in}?S.\ ?u\ x \neq 0$›
show *dependent ?S* **by** *best* **}**
qed

5.1 Matrices

lemma *zero-not-invertible*:
$\neg$ (*invertible* (*0*::*real*^'*n*^'*n*))
using *invertible-times-eq-zero matrix-vector-mult-0* **by** *blast*

Based on matrix-vector-column in HOL/Multivariate_Analysis/Euclidean_Space.thy in Isabelle 2009-1:

lemma *vector-matrix-row*:
fixes x :: ('*a*::*comm-semiring-1*)^'*m* **and** A :: ('*a*^'*n*^'*m*)
shows $x\ v{*}\ A = (\sum\ i{\in}UNIV.\ (x\$i) *s\ (A\$i))$
unfolding *vector-matrix-mult-def*
by (*simp add*: *vec-eq-iff mult.commute*)

lemma *matrix-inv*:
assumes *invertible M*
shows *matrix-inv M* $**$ M = *mat 1*
and M $**$ *matrix-inv M* = *mat 1*
using ‹*invertible M*› **and** *someI-ex* [*of* $\lambda\ N.\ M ** N = mat\ 1 \land N ** M = mat\ 1$]
unfolding *invertible-def* **and** *matrix-inv-def*
by *simp-all*

lemma *matrix-inv-invertible*:
assumes *invertible M*
shows *invertible* (*matrix-inv M*)
using ‹*invertible M*› **and** *matrix-inv*
unfolding *invertible-def* [*of matrix-inv M*]
by *auto*

lemma *invertible-times-non-zero*:
fixes M :: *real*^'*n*^'*n*
assumes *invertible M* **and** $v \neq 0$
shows $M *v\ v \neq 0$

```
  using ‹invertible M› and ‹v ≠ 0› and invertible-times-eq-zero [of M v]
  by auto

lemma matrix-right-invertible-ker:
  fixes M :: real^('m::finite)^'n
  shows (∃ M'. M ** M' = mat 1) ⟷ (∀ x. x v* M = 0 ⟶ x = 0)
  using left-invertible-transpose matrix-left-invertible-ker by force

lemma left-invertible-iff-invertible:
  fixes M :: real^'n^'n
  shows (∃ N. N ** M = mat 1) ⟷ invertible M
  by (simp add: invertible-def matrix-left-right-inverse)

lemma right-invertible-iff-invertible:
  fixes M :: real^'n^'n
  shows (∃ N. M ** N = mat 1) ⟷ invertible M
  by (simp add: invertible-def matrix-left-right-inverse)

definition symmatrix :: 'a^'n^'n ⇒ bool where
  symmatrix M ≜ transpose M = M

lemma symmatrix-preserve:
  fixes M N :: ('a::comm-semiring-1)^'n^'n
  assumes symmatrix M
  shows symmatrix (N ** M ** transpose N)
proof −
  have transpose (N ** M ** transpose N) = N ** (M ** transpose N)
   by (metis (no-types) transpose-transpose assms matrix-transpose-mul symmatrix-def)
  then show ?thesis
    by (simp add: matrix-mul-assoc symmatrix-def)
qed

lemma non-zero-mult-invertible-non-zero:
  fixes M :: real^'n^'n
  assumes v ≠ 0 and invertible M
  shows v v* M ≠ 0
  using ‹v ≠ 0› and ‹invertible M› and times-invertible-eq-zero
  by auto

end
```

6 Right group actions

```
theory Action
  imports HOL−Algebra.Group
begin

locale action = group +
  fixes act :: 'b ⇒ 'a ⇒ 'b (infixl <o 69)
```

assumes *id-act* [*simp*]: $b <o \mathbf{1} = b$
and *act-act'*:
$g \in carrier\ G \wedge h \in carrier\ G \longrightarrow (b <o\ g) <o\ h = b <o\ (g \otimes h)$
begin

lemma *act-act*:
assumes $g \in carrier\ G$ **and** $h \in carrier\ G$
shows $(b <o\ g) <o\ h = b <o\ (g \otimes h)$
proof −
from ‹$g \in carrier\ G$› **and** ‹$h \in carrier\ G$› **and** *act-act'*
show $(b <o\ g) <o\ h = b <o\ (g \otimes h)$ **by** *simp*
qed

lemma *act-act-inv* [*simp*]:
assumes $g \in carrier\ G$
shows $b <o\ g <o\ inv\ g = b$
proof −
from ‹$g \in carrier\ G$› **have** $inv\ g \in carrier\ G$ **by** (*rule inv-closed*)
with ‹$g \in carrier\ G$› **have** $b <o\ g <o\ inv\ g = b <o\ g \otimes inv\ g$ **by** (*rule act-act*)
with ‹$g \in carrier\ G$› **show** $b <o\ g <o\ inv\ g = b$ **by** *simp*
qed

lemma *act-inv-act* [*simp*]:
assumes $g \in carrier\ G$
shows $b <o\ inv\ g <o\ g = b$
using ‹$g \in carrier\ G$› **and** *act-act-inv* [*of inv g*]
by *simp*

lemma *act-inv-iff*:
assumes $g \in carrier\ G$
shows $b <o\ inv\ g = c \longleftrightarrow b = c <o\ g$
proof
assume $b <o\ inv\ g = c$
hence $b <o\ inv\ g <o\ g = c <o\ g$ **by** *simp*
with ‹$g \in carrier\ G$› **show** $b = c <o\ g$ **by** *simp*
next
assume $b = c <o\ g$
hence $b <o\ inv\ g = c <o\ g <o\ inv\ g$ **by** *simp*
with ‹$g \in carrier\ G$› **show** $b <o\ inv\ g = c$ **by** *simp*
qed

end

end

7 Projective geometry

theory *Projective*
imports *Linear-Algebra2*

Euclid-Tarski
Action
begin

7.1 Proportionality on non-zero vectors

context *vector-space*
begin

definition *proportionality* :: $('b \times 'b)$ *set* **where**
proportionality $\triangleq \{(x, y).\ x \neq 0 \wedge y \neq 0 \wedge (\exists k.\ x = \mathit{scale}\ k\ y)\}$

definition *non-zero-vectors* :: $'b$ *set* **where**
non-zero-vectors $\triangleq \{x.\ x \neq 0\}$

lemma *proportionality-refl-on*: *refl-on local.non-zero-vectors local.proportionality*
proof −
have *local.proportionality* $\subseteq$ *local.non-zero-vectors* $\times$ *local.non-zero-vectors*
unfolding *proportionality-def non-zero-vectors-def*
by *auto*
moreover have $\forall x \in$*local.non-zero-vectors*. $(x, x) \in$ *local.proportionality*
proof
fix x
assume $x \in$ *local.non-zero-vectors*
hence $x \neq 0$ **unfolding** *non-zero-vectors-def* **..**
moreover have $x = \mathit{scale}\ 1\ x$ **by** *simp*
ultimately show $(x, x) \in$ *local.proportionality*
unfolding *proportionality-def*
by *blast*
qed
ultimately show *refl-on local.non-zero-vectors local.proportionality*
unfolding *refl-on-def* **..**
qed

lemma *proportionality-sym*: *sym local.proportionality*
proof −
{ fix $x\ y$
assume $(x, y) \in$ *local.proportionality*
hence $x \neq 0$ **and** $y \neq 0$ **and** $\exists k.\ x = \mathit{scale}\ k\ y$
unfolding *proportionality-def*
by *simp+*
from ‹$\exists k.\ x = \mathit{scale}\ k\ y$› **obtain** k **where** $x = \mathit{scale}\ k\ y$ **by** *auto*
with ‹$x \neq 0$› **have** $k \neq 0$ **by** *simp*
with ‹$x = \mathit{scale}\ k\ y$› **have** $y = \mathit{scale}\ (1/k)\ x$ **by** *simp*
with ‹$x \neq 0$› **and** ‹$y \neq 0$› **have** $(y, x) \in$ *local.proportionality*
unfolding *proportionality-def*
by *auto*
}
thus *sym local.proportionality*

unfolding *sym-def*
by *blast*
qed

lemma *proportionality-trans*: *trans local.proportionality*
proof −
{ **fix** $x\ y\ z$
assume $(x,\ y) \in$ *local.proportionality* **and** $(y,\ z) \in$ *local.proportionality*
hence $x \neq 0$ **and** $z \neq 0$ **and** $\exists\, j.\ x = \mathit{scale}\ j\ y$ **and** $\exists\, k.\ y = \mathit{scale}\ k\ z$
unfolding *proportionality-def*
by *simp+*
from $\langle\exists\, j.\ x = \mathit{scale}\ j\ y\rangle$ **and** $\langle\exists\, k.\ y = \mathit{scale}\ k\ z\rangle$
obtain j **and** k **where** $x = \mathit{scale}\ j\ y$ **and** $y = \mathit{scale}\ k\ z$ **by** *auto+*
hence $x = \mathit{scale}\ (j * k)\ z$ **by** *simp*
with $\langle x \neq 0\rangle$ **and** $\langle z \neq 0\rangle$ **have** $(x,\ z) \in$ *local.proportionality*
unfolding *proportionality-def*
by *auto*
}
thus *trans local.proportionality*
unfolding *trans-def*
by *blast*
qed

theorem *proportionality-equiv*: *equiv local.non-zero-vectors local.proportionality*
unfolding *equiv-def*
by (*simp add*:
proportionality-refl-on
proportionality-sym
proportionality-trans)

end

definition *invertible-proportionality* ::
((*real*ˆ(′*n*::*finite*)ˆ′*n*) × (*real*ˆ′*n*ˆ′*n*)) *set* **where**
invertible-proportionality ≜
real-vector.proportionality ∩ (*Collect invertible* × *Collect invertible*)

lemma *invertible-proportionality-equiv*:
equiv (*Collect invertible* :: (*real*ˆ(′*n*::*finite*)ˆ′*n*) *set*)
invertible-proportionality
(**is** *equiv ?invs* -)
proof −
from *zero-not-invertible*
have *real-vector.non-zero-vectors* ∩ *?invs* = *?invs*
unfolding *real-vector.non-zero-vectors-def*
by *auto*
from *equiv-restrict* **and** *real-vector.proportionality-equiv*
have *equiv* (*real-vector.non-zero-vectors* ∩ *?invs*) *invertible-proportionality*
unfolding *invertible-proportionality-def*

by *auto*
with ‹*real-vector.non-zero-vectors* ∩ *?invs* = *?invs*›
show *equiv ?invs invertible-proportionality*
by *simp*
qed

7.2 Points of the real projective plane

typedef *proj2* = (*real-vector.non-zero-vectors* :: (*real^3*) *set*)//*real-vector.proportionality*
proof
have (*axis 1 1* :: *real^3*) ∈ *real-vector.non-zero-vectors*
unfolding *real-vector.non-zero-vectors-def*
by (*simp add*: *axis-def vec-eq-iff*[**where** *'a=real*])
thus *real-vector.proportionality* `` {*axis 1 1*} ∈ (*real-vector.non-zero-vectors* :: (*real^3*) *set*)//*real-vector.proportionality*
unfolding *quotient-def*
by *auto*
qed

definition *proj2-rep* :: *proj2* ⇒ *real^3* **where**
proj2-rep x ≜ ϵ *v*. *v* ∈ *Rep-proj2 x*

definition *proj2-abs* :: *real^3* ⇒ *proj2* **where**
proj2-abs v ≜ *Abs-proj2* (*real-vector.proportionality* `` {*v*})

lemma *proj2-rep-in*: *proj2-rep x* ∈ *Rep-proj2 x*
proof −
let *?v* = *proj2-rep x*
from *quotient-element-nonempty* **and**
real-vector.proportionality-equiv **and**
Rep-proj2 [*of x*]
have ∃ *w*. *w* ∈ *Rep-proj2 x*
by *auto*
with *someI-ex* [*of* λ *z*. *z* ∈ *Rep-proj2 x*]
show *?v* ∈ *Rep-proj2 x*
unfolding *proj2-rep-def*
by *simp*
qed

lemma *proj2-rep-non-zero*: *proj2-rep x* ≠ *0*
proof −
from
Union-quotient [*of real-vector.non-zero-vectors real-vector.proportionality*]
and *real-vector.proportionality-equiv*
and *Rep-proj2* [*of x*] **and** *proj2-rep-in* [*of x*]
have *proj2-rep x* ∈ *real-vector.non-zero-vectors*
unfolding *quotient-def*
by *auto*
thus *proj2-rep x* ≠ *0*

unfolding *real-vector.non-zero-vectors-def*
by *simp*
qed

lemma *proj2-rep-abs*:
fixes v :: *real^3*
assumes $v \in$ *real-vector.non-zero-vectors*
shows (v, *proj2-rep* (*proj2-abs* v)) $\in$ *real-vector.proportionality*
proof −
from ‹$v \in$ *real-vector.non-zero-vectors*›
have *real-vector.proportionality* `` $\{v\}$ $\in$ (*real-vector.non-zero-vectors* :: (*real^3*) *set*)//*real-vector.proportionality*
unfolding *quotient-def*
by *auto*
with *Abs-proj2-inverse*
have *Rep-proj2* (*proj2-abs* v) = *real-vector.proportionality* `` $\{v\}$
unfolding *proj2-abs-def*
by *simp*
with *proj2-rep-in*
have *proj2-rep* (*proj2-abs* v) $\in$ *real-vector.proportionality* `` $\{v\}$ **by** *auto*
thus (v, *proj2-rep* (*proj2-abs* v)) $\in$ *real-vector.proportionality* **by** *simp*
qed

lemma *proj2-abs-rep*: *proj2-abs* (*proj2-rep* x) = x
proof −
from *partition-Image-element*
[*of real-vector.non-zero-vectors*
real-vector.proportionality
Rep-proj2 x
proj2-rep x]
and *real-vector.proportionality-equiv*
and *Rep-proj2* [*of* x] **and** *proj2-rep-in* [*of* x]
have *real-vector.proportionality* `` {*proj2-rep* x} = *Rep-proj2* x
by *simp*
with *Rep-proj2-inverse* **show** *proj2-abs* (*proj2-rep* x) = x
unfolding *proj2-abs-def*
by *simp*
qed

lemma *proj2-abs-mult*:
assumes $c \neq 0$
shows *proj2-abs* ($c *_R v$) = *proj2-abs* v
proof *cases*
assume $v = 0$
thus *proj2-abs* ($c *_R v$) = *proj2-abs* v **by** *simp*
next
assume $v \neq 0$
with ‹$c \neq 0$›
have ($c *_R v$, v) $\in$ *real-vector.proportionality*

and $c *_R v \in$ *real-vector.non-zero-vectors*
and $v \in$ *real-vector.non-zero-vectors*
unfolding *real-vector.proportionality-def*
and *real-vector.non-zero-vectors-def*
by *simp-all*
with *eq-equiv-class-iff*
[*of real-vector.non-zero-vectors*
real-vector.proportionality
$c *_R v$
v]
and *real-vector.proportionality-equiv*
have *real-vector.proportionality* ‘‘ $\{c *_R v\}$ =
real-vector.proportionality ‘‘ $\{v\}$
by *simp*
thus *proj2-abs* $(c *_R v)$ = *proj2-abs* v
unfolding *proj2-abs-def*
by *simp*
qed

lemma *proj2-abs-mult-rep*:
assumes $c \neq 0$
shows *proj2-abs* $(c *_R$ *proj2-rep* $x) = x$
using *proj2-abs-mult* **and** *proj2-abs-rep* **and** *assms*
by *simp*

lemma *proj2-rep-inj*: *inj proj2-rep*
by (*simp add*: *inj-on-inverseI* [*of UNIV proj2-abs proj2-rep*] *proj2-abs-rep*)

lemma *proj2-rep-abs2*:
assumes $v \neq 0$
shows $\exists$ k. $k \neq 0 \wedge$ *proj2-rep* (*proj2-abs* v) = $k *_R v$
proof −
from *proj2-rep-abs* [*of* v] **and** ‹$v \neq 0$›
have (v, *proj2-rep* (*proj2-abs* v)) $\in$ *real-vector.proportionality*
unfolding *real-vector.non-zero-vectors-def*
by *simp*
then obtain c **where** $v = c *_R$ *proj2-rep* (*proj2-abs* v)
unfolding *real-vector.proportionality-def*
by *auto*
with ‹$v \neq 0$› **have** $c \neq 0$ **by** *auto*
hence $1/c \neq 0$ **by** *simp*

from ‹$v = c *_R$ *proj2-rep* (*proj2-abs* v)›
have $(1/c) *_R v = (1/c) *_R c *_R$ *proj2-rep* (*proj2-abs* v)
by *simp*
with ‹$c \neq 0$› **have** *proj2-rep* (*proj2-abs* v) = $(1/c) *_R v$ **by** *simp*

with ‹$1/c \neq 0$› **show** $\exists$ k. $k \neq 0 \wedge$ *proj2-rep* (*proj2-abs* v) = $k *_R v$
by *blast*

qed

lemma *proj2-abs-abs-mult*:
  assumes *proj2-abs* v = *proj2-abs* w **and** $w \neq 0$
  shows $\exists\ c.\ v = c *_R w$
proof *cases*
  assume $v = 0$
  hence $v = 0 *_R w$ **by** *simp*
  thus $\exists\ c.\ v = c *_R w$ **..**
next
  assume $v \neq 0$
  from ‹*proj2-abs* v = *proj2-abs* w›
  have *proj2-rep* (*proj2-abs* v) = *proj2-rep* (*proj2-abs* w) **by** *simp*
  with *proj2-rep-abs2* **and** ‹$w \neq 0$›
  obtain k **where** *proj2-rep* (*proj2-abs* v) = $k *_R w$ **by** *auto*
  with *proj2-rep-abs2* [*of* v] **and** ‹$v \neq 0$›
  obtain j **where** $j \neq 0$ **and** $j *_R v = k *_R w$ **by** *auto*
  hence $(1/j) *_R j *_R v = (1/j) *_R k *_R w$ **by** *simp*
  with ‹$j \neq 0$› **have** $v = (k/j) *_R w$ **by** *simp*
  thus $\exists\ c.\ v = c *_R w$ **..**
qed

lemma *dependent-proj2-abs*:
  assumes $p \neq 0$ **and** $q \neq 0$ **and** $i \neq 0 \lor j \neq 0$ **and** $i *_R p + j *_R q = 0$
  shows *proj2-abs* p = *proj2-abs* q
proof −
  have $i \neq 0$
  proof
    assume $i = 0$
    with ‹$i \neq 0 \lor j \neq 0$› **have** $j \neq 0$ **by** *simp*
    with ‹$i *_R p + j *_R q = 0$› **and** ‹$q \neq 0$› **have** $i *_R p \neq 0$ **by** *auto*
    with ‹$i = 0$› **show** *False* **by** *simp*
  qed
  with ‹$p \neq 0$› **and** ‹$i *_R p + j *_R q = 0$› **have** $j \neq 0$ **by** *auto*

  from ‹$i \neq 0$›
  have *proj2-abs* p = *proj2-abs* $(i *_R p)$ **by** (*rule proj2-abs-mult* [*symmetric*])
  also from ‹$i *_R p + j *_R q = 0$› **and** *proj2-abs-mult* [*of* -1 $j *_R q$]
  have ... = *proj2-abs* $(j *_R q)$ **by** (*simp add*: *algebra-simps* [*symmetric*])
  also from ‹$j \neq 0$› **have** ... = *proj2-abs* q **by** (*rule proj2-abs-mult*)
  finally show *proj2-abs* p = *proj2-abs* q **.**
qed

lemma *proj2-rep-dependent*:
  assumes $i *_R$ *proj2-rep* v + $j *_R$ *proj2-rep* w = 0
  (**is** $i *_R\ ?p + j *_R\ ?q = 0$)
  and $i \neq 0 \lor j \neq 0$
  shows $v = w$
proof −

have $?p \neq 0$ **and** $?q \neq 0$ **by** (*rule proj2-rep-non-zero*)+
with ‹$i \neq 0 \vee j \neq 0$› **and** ‹$i *_R ?p + j *_R ?q = 0$›
have *proj2-abs ?p = proj2-abs ?q* **by** (*simp add: dependent-proj2-abs*)
thus $v = w$ **by** (*simp add: proj2-abs-rep*)
qed

lemma *proj2-rep-independent*:
assumes $p \neq q$
shows *independent* {*proj2-rep p*, *proj2-rep q*}
proof
let *?p′ = proj2-rep p*
let *?q′ = proj2-rep q*
let *?S* = {*?p′*, *?q′*}
assume *dependent ?S*
from *proj2-rep-inj* **and** ‹$p \neq q$› **have** $?p' \neq ?q'$
unfolding *inj-on-def*
by *auto*
with *dependent-explicit-2* [*of ?p′ ?q′*] **and** ‹*dependent ?S*›
obtain *i* **and** *j* **where** $i *_R ?p' + j *_R ?q' = 0$ **and** $i \neq 0 \vee j \neq 0$
by (*simp add: scalar-equiv*) *auto*
with *proj2-rep-dependent* **have** $p = q$ **by** *simp*
with ‹$p \neq q$› **show** *False* **..**
qed

7.3 Lines of the real projective plane

definition *proj2-Col* :: [*proj2*, *proj2*, *proj2*] ⇒ *bool* **where**
proj2-Col p q r ≜
($\exists$ *i j k*. $i *_R$ *proj2-rep p* + $j *_R$ *proj2-rep q* + $k *_R$ *proj2-rep r* = 0
$\wedge$ ($i \neq 0 \vee j \neq 0 \vee k \neq 0$))

lemma *proj2-Col-abs*:
assumes $p \neq 0$ **and** $q \neq 0$ **and** $r \neq 0$ **and** $i \neq 0 \vee j \neq 0 \vee k \neq 0$
and $i *_R p + j *_R q + k *_R r = 0$
shows *proj2-Col* (*proj2-abs p*) (*proj2-abs q*) (*proj2-abs r*)
(**is** *proj2-Col ?pp ?pq ?pr*)
proof −
from ‹$p \neq 0$› **and** *proj2-rep-abs2*
obtain i' **where** $i' \neq 0$ **and** *proj2-rep ?pp* = $i' *_R p$ (**is** *?rp* = -) **by** *auto*
from ‹$q \neq 0$› **and** *proj2-rep-abs2*
obtain j' **where** $j' \neq 0$ **and** *proj2-rep ?pq* = $j' *_R q$ (**is** *?rq* = -) **by** *auto*
from ‹$r \neq 0$› **and** *proj2-rep-abs2*
obtain k' **where** $k' \neq 0$ **and** *proj2-rep ?pr* = $k' *_R r$ (**is** *?rr* = -) **by** *auto*
with ‹$i *_R p + j *_R q + k *_R r = 0$›
and ‹$i' \neq 0$› **and** ‹*proj2-rep ?pp* = $i' *_R p$›
and ‹$j' \neq 0$› **and** ‹*proj2-rep ?pq* = $j' *_R q$›
have $(i/i') *_R ?rp + (j/j') *_R ?rq + (k/k') *_R ?rr = 0$ **by** *simp*

from ‹$i' \neq 0$› **and** ‹$j' \neq 0$› **and** ‹$k' \neq 0$› **and** ‹$i \neq 0 \vee j \neq 0 \vee k \neq 0$›

have $i/i' \neq 0 \vee j/j' \neq 0 \vee k/k' \neq 0$ **by** *simp*
with ‹$(i/i') *_R$?rp + $(j/j') *_R$?rq + $(k/k') *_R$?rr = 0›
show *proj2-Col ?pp ?pq ?pr* **by** (*unfold proj2-Col-def*, *best*)
qed

lemma *proj2-Col-permute*:
assumes *proj2-Col a b c*
shows *proj2-Col a c b*
and *proj2-Col b a c*
proof −
let $?a' = $ *proj2-rep a*
let $?b' = $ *proj2-rep b*
let $?c' = $ *proj2-rep c*
from ‹*proj2-Col a b c*›
obtain i **and** j **and** k **where**
$i *_R ?a' + j *_R ?b' + k *_R ?c' = 0$
and $i \neq 0 \vee j \neq 0 \vee k \neq 0$
unfolding *proj2-Col-def*
by *auto*

from ‹$i *_R ?a' + j *_R ?b' + k *_R ?c' = 0$›
have $i *_R ?a' + k *_R ?c' + j *_R ?b' = 0$
and $j *_R ?b' + i *_R ?a' + k *_R ?c' = 0$
by (*simp-all add*: *ac-simps*)
moreover from ‹$i \neq 0 \vee j \neq 0 \vee k \neq 0$›
have $i \neq 0 \vee k \neq 0 \vee j \neq 0$ **and** $j \neq 0 \vee i \neq 0 \vee k \neq 0$ **by** *auto*
ultimately show *proj2-Col a c b* **and** *proj2-Col b a c*
unfolding *proj2-Col-def*
by *auto*
qed

lemma *proj2-Col-coincide*: *proj2-Col a a c*
proof −
have $1 *_R$ *proj2-rep a* + $(-1) *_R$ *proj2-rep a* + $0 *_R$ *proj2-rep c* = 0
by *simp*
moreover have $(1::real) \neq 0$ **by** *simp*
ultimately show *proj2-Col a a c*
unfolding *proj2-Col-def*
by *blast*
qed

lemma *proj2-Col-iff*:
assumes $a \neq r$
shows *proj2-Col a r t* $\longleftrightarrow$
$t = a \vee (\exists\ i.\ t = $ *proj2-abs* $(i *_R$ (*proj2-rep a*) + (*proj2-rep r*)))
proof
let $?a' = $ *proj2-rep a*
let $?r' = $ *proj2-rep r*
let $?t' = $ *proj2-rep t*

{ **assume** *proj2-Col a r t*
  then obtain h **and** j **and** k **where**
    $h *_R\ ?a' + j *_R\ ?r' + k *_R\ ?t' = 0$
    and $h \neq 0 \vee j \neq 0 \vee k \neq 0$
    unfolding *proj2-Col-def*
    by *auto*

  show $t = a \vee (\exists\ i.\ t = \textit{proj2-abs}\ (i *_R\ ?a' + ?r'))$
  proof *cases*
    assume $j = 0$
    with ‹$h \neq 0 \vee j \neq 0 \vee k \neq 0$› **have** $h \neq 0 \vee k \neq 0$ **by** *simp*
    with *proj2-rep-dependent*
      and ‹$h *_R\ ?a' + j *_R\ ?r' + k *_R\ ?t' = 0$›
      and ‹$j = 0$›
    have $t = a$ **by** *auto*
    thus $t = a \vee (\exists\ i.\ t = \textit{proj2-abs}\ (i *_R\ ?a' + ?r'))$ **..**
  next
    assume $j \neq 0$
    have $k \neq 0$
    proof (*rule ccontr*)
      assume $\neg\ k \neq 0$
      with *proj2-rep-dependent*
        and ‹$h *_R\ ?a' + j *_R\ ?r' + k *_R\ ?t' = 0$›
        and ‹$j \neq 0$›
      have $a = r$ **by** *simp*
      with ‹$a \neq r$› **show** *False* **..**
    qed

    from ‹$h *_R\ ?a' + j *_R\ ?r' + k *_R\ ?t' = 0$›
    have $h *_R\ ?a' + j *_R\ ?r' + k *_R\ ?t' - k *_R\ ?t' = -k *_R\ ?t'$ **by** *simp*
    hence $h *_R\ ?a' + j *_R\ ?r' = -k *_R\ ?t'$ **by** *simp*
    with *proj2-abs-mult-rep* [*of* $-k$] **and** ‹$k \neq 0$›
    have *proj2-abs* $(h *_R\ ?a' + j *_R\ ?r') = t$ **by** *simp*
    with *proj2-abs-mult* [*of* $1/j$ $h *_R\ ?a' + j *_R\ ?r'$] **and** ‹$j \neq 0$›
    have *proj2-abs* $((h/j) *_R\ ?a' + ?r') = t$
      by (*simp add*: *scaleR-right-distrib*)
    hence $\exists\ i.\ t = \textit{proj2-abs}\ (i *_R\ ?a' + ?r')$ **by** *auto*
    thus $t = a \vee (\exists\ i.\ t = \textit{proj2-abs}\ (i *_R\ ?a' + ?r'))$ **..**
  qed
}

{ **assume** $t = a \vee (\exists\ i.\ t = \textit{proj2-abs}\ (i *_R\ ?a' + ?r'))$
  show *proj2-Col a r t*
  proof *cases*
    assume $t = a$
    with *proj2-Col-coincide* **and** *proj2-Col-permute*
    show *proj2-Col a r t* **by** *blast*
  next

assume $t \neq a$
with ‹$t = a \vee (\exists\ i.\ t = proj2\text{-}abs\ (i *_R\ ?a' + ?r'))$›
obtain i **where** $t = proj2\text{-}abs\ (i *_R\ ?a' + ?r')$ **by** *auto*
from *proj2-rep-dependent* [*of i a 1 r*] **and** ‹$a \neq r$›
have $i *_R\ ?a' + ?r' \neq 0$ **by** *auto*
with *proj2-rep-abs2* **and** ‹$t = proj2\text{-}abs\ (i *_R\ ?a' + ?r')$›
obtain j **where** $?t' = j *_R (i *_R\ ?a' + ?r')$ **by** *auto*
hence $?t' - ?t' = (j * i) *_R\ ?a' + j *_R\ ?r' + (-1) *_R\ ?t'$
by (*simp add*: *scaleR-right-distrib*)
hence $(j * i) *_R\ ?a' + j *_R\ ?r' + (-1) *_R\ ?t' = 0$ **by** *simp*
have $\exists\ h\ j\ k.\ h *_R\ ?a' + j *_R\ ?r' + k *_R\ ?t' = 0$
$\wedge (h \neq 0 \vee j \neq 0 \vee k \neq 0)$
proof *standard+*
from ‹$(j * i) *_R\ ?a' + j *_R\ ?r' + (-1) *_R\ ?t' = 0$›
show $(j * i) *_R\ ?a' + j *_R\ ?r' + (-1) *_R\ ?t' = 0$ **.**
show $j * i \neq 0 \vee j \neq 0 \vee (-1::real) \neq 0$ **by** *simp*
qed
thus *proj2-Col a r t*
unfolding *proj2-Col-def* **.**
qed
}
qed

definition *proj2-Col-coeff* :: *proj2* ⇒ *proj2* ⇒ *proj2* ⇒ *real* **where**
$proj2\text{-}Col\text{-}coeff\ a\ r\ t \triangleq \epsilon\ i.\ t = proj2\text{-}abs\ (i *_R proj2\text{-}rep\ a + proj2\text{-}rep\ r)$

lemma *proj2-Col-coeff*:
assumes *proj2-Col a r t* **and** $a \neq r$ **and** $t \neq a$
shows $t = proj2\text{-}abs\ ((proj2\text{-}Col\text{-}coeff\ a\ r\ t) *_R proj2\text{-}rep\ a + proj2\text{-}rep\ r)$
proof −
from ‹$a \neq r$› **and** ‹*proj2-Col a r t*› **and** ‹$t \neq a$› **and** *proj2-Col-iff*
have $\exists\ i.\ t = proj2\text{-}abs\ (i *_R proj2\text{-}rep\ a + proj2\text{-}rep\ r)$ **by** *simp*
thus $t = proj2\text{-}abs\ ((proj2\text{-}Col\text{-}coeff\ a\ r\ t) *_R proj2\text{-}rep\ a + proj2\text{-}rep\ r)$
by (*unfold proj2-Col-coeff-def*) (*rule someI-ex*)
qed

lemma *proj2-Col-coeff-unique′*:
assumes $a \neq 0$ **and** $r \neq 0$ **and** $proj2\text{-}abs\ a \neq proj2\text{-}abs\ r$
and $proj2\text{-}abs\ (i *_R a + r) = proj2\text{-}abs\ (j *_R a + r)$
shows $i = j$
proof −
from ‹$a \neq 0$› **and** ‹$r \neq 0$› **and** ‹$proj2\text{-}abs\ a \neq proj2\text{-}abs\ r$›
and *dependent-proj2-abs* [*of a r - 1*]
have $i *_R a + r \neq 0$ **and** $j *_R a + r \neq 0$ **by** *auto*
with *proj2-rep-abs2* [*of* $i *_R a + r$]
and *proj2-rep-abs2* [*of* $j *_R a + r$]
obtain k **and** l **where** $k \neq 0$
and $proj2\text{-}rep\ (proj2\text{-}abs\ (i *_R a + r)) = k *_R (i *_R a + r)$
and $proj2\text{-}rep\ (proj2\text{-}abs\ (j *_R a + r)) = l *_R (j *_R a + r)$

```
    by auto
  with ‹proj2-abs (i *R a + r) = proj2-abs (j *R a + r)›
  have (k * i) *R a + k *R r = (l * j) *R a + l *R r
    by (simp add: scaleR-right-distrib)
  hence (k * i − l * j) *R a + (k − l) *R r = 0
    by (simp add: algebra-simps vec-eq-iff)
  with ‹a ≠ 0› and ‹r ≠ 0› and ‹proj2-abs a ≠ proj2-abs r›
    and dependent-proj2-abs [of a r k * i − l * j k − l]
  have k * i − l * j = 0 and k − l = 0 by auto
  from ‹k − l = 0› have k = l by simp
  with ‹k * i − l * j = 0› have k * i = k * j by simp
  with ‹k ≠ 0› show i = j by simp
qed

lemma proj2-Col-coeff-unique:
  assumes a ≠ r
  and proj2-abs (i *R proj2-rep a + proj2-rep r)
  = proj2-abs (j *R proj2-rep a + proj2-rep r)
  shows i = j
proof −
  let ?a' = proj2-rep a
  let ?r' = proj2-rep r
  have ?a' ≠ 0 and ?r' ≠ 0 by (rule proj2-rep-non-zero)+

  from ‹a ≠ r› have proj2-abs ?a' ≠ proj2-abs ?r' by (simp add: proj2-abs-rep)
  with ‹?a' ≠ 0› and ‹?r' ≠ 0›
    and ‹proj2-abs (i *R ?a' + ?r') = proj2-abs (j *R ?a' + ?r')›
    and proj2-Col-coeff-unique'
  show i = j by simp
qed

datatype proj2-line = P2L proj2

definition L2P :: proj2-line ⇒ proj2 where
  L2P l ≜ case l of P2L p ⇒ p

lemma L2P-P2L [simp]: L2P (P2L p) = p
  unfolding L2P-def
  by simp

lemma P2L-L2P [simp]: P2L (L2P l) = l
  by (induct l) simp

lemma L2P-inj [simp]:
  assumes L2P l = L2P m
  shows l = m
  using P2L-L2P [of l] and assms
  by simp
```

lemma *P2L-to-L2P*: *P2L p* = *l* ⟷ *p* = *L2P l*
proof
  assume *P2L p* = *l*
  hence *L2P* (*P2L p*) = *L2P l* **by** *simp*
  thus *p* = *L2P l* **by** *simp*
next
  assume *p* = *L2P l*
  thus *P2L p* = *l* **by** *simp*
qed

definition *proj2-line-abs* :: *real^3* ⇒ *proj2-line* **where**
  proj2-line-abs v ≜ *P2L* (*proj2-abs v*)

definition *proj2-line-rep* :: *proj2-line* ⇒ *real^3* **where**
  proj2-line-rep l ≜ *proj2-rep* (*L2P l*)

lemma *proj2-line-rep-abs*:
  assumes *v* ≠ *0*
  shows ∃ *k*. *k* ≠ *0* ∧ *proj2-line-rep* (*proj2-line-abs v*) = *k* $*_R$ *v*
  unfolding *proj2-line-rep-def* **and** *proj2-line-abs-def*
  using *proj2-rep-abs2* **and** ‹*v* ≠ *0*›
  by *simp*

lemma *proj2-line-abs-rep* [*simp*]: *proj2-line-abs* (*proj2-line-rep l*) = *l*
  unfolding *proj2-line-abs-def* **and** *proj2-line-rep-def*
  by (*simp add*: *proj2-abs-rep*)

lemma *proj2-line-rep-non-zero*: *proj2-line-rep l* ≠ *0*
  unfolding *proj2-line-rep-def*
  using *proj2-rep-non-zero*
  by *simp*

lemma *proj2-line-rep-dependent*:
  assumes *i* $*_R$ *proj2-line-rep l* + *j* $*_R$ *proj2-line-rep m* = *0*
  and *i* ≠ *0* ∨ *j* ≠ *0*
  shows *l* = *m*
  using *proj2-rep-dependent* [*of i L2P l j L2P m*] **and** *assms*
  unfolding *proj2-line-rep-def*
  by *simp*

lemma *proj2-line-abs-mult*:
  assumes *k* ≠ *0*
  shows *proj2-line-abs* (*k* $*_R$ *v*) = *proj2-line-abs v*
  unfolding *proj2-line-abs-def*
  using ‹*k* ≠ *0*›
  by (*subst proj2-abs-mult*) *simp-all*

lemma *proj2-line-abs-abs-mult*:
  assumes *proj2-line-abs v* = *proj2-line-abs w* **and** *w* ≠ *0*

shows $\exists\ k.\ v = k *_R w$
using *assms*
by (*unfold proj2-line-abs-def*) (*simp add*: *proj2-abs-abs-mult*)

definition *proj2-incident* :: *proj2* $\Rightarrow$ *proj2-line* $\Rightarrow$ *bool* **where**
proj2-incident p l $\triangleq$ (*proj2-rep p*) $\cdot$ (*proj2-line-rep l*) = *0*

lemma *proj2-points-define-line*:
shows $\exists\ l.$ *proj2-incident p l* $\wedge$ *proj2-incident q l*
proof –
let *?p'* = *proj2-rep p*
let *?q'* = *proj2-rep q*
let *?B* = {*?p'*, *?q'*}
from *card-suc-ge-insert* [*of ?p'* {*?q'*}] **have** *card ?B* $\leq$ *2* **by** *simp*
with *dim-le-card'* [*of ?B*] **have** *dim ?B* < *3* **by** *simp*
with *lowdim-subset-hyperplane* [*of ?B*]
obtain *l'* **where** *l'* $\neq$ *0* **and** *span ?B* $\subseteq$ {*x*. *l'* $\cdot$ *x* = *0*} **by** *auto*
let *?l* = *proj2-line-abs l'*
let *?l''* = *proj2-line-rep ?l*
from *proj2-line-rep-abs* **and** ‹*l'* $\neq$ *0*›
obtain *k* **where** *?l''* = $k *_R l'$ **by** *auto*

have *?p'* $\in$ *?B* **and** *?q'* $\in$ *?B* **by** *simp-all*
with *span-superset* [*of ?B*] **and** ‹*span ?B* $\subseteq$ {*x*. *l'* $\cdot$ *x* = *0*}›
have *l'* $\cdot$ *?p'* = *0* **and** *l'* $\cdot$ *?q'* = *0* **by** *auto*
hence *?p'* $\cdot$ *l'* = *0* **and** *?q'* $\cdot$ *l'* = *0* **by** (*simp-all add*: *inner-commute*)
with *dot-scaleR-mult*(*2*) [*of - k l'*] **and** ‹*?l''* = $k *_R l'$›
have *proj2-incident p ?l* $\wedge$ *proj2-incident q ?l*
unfolding *proj2-incident-def*
by *simp*
thus $\exists\ l.$ *proj2-incident p l* $\wedge$ *proj2-incident q l* **by** *auto*
qed

definition *proj2-line-through* :: *proj2* $\Rightarrow$ *proj2* $\Rightarrow$ *proj2-line* **where**
proj2-line-through p q $\triangleq$ $\epsilon\ l.$ *proj2-incident p l* $\wedge$ *proj2-incident q l*

lemma *proj2-line-through-incident*:
shows *proj2-incident p* (*proj2-line-through p q*)
and *proj2-incident q* (*proj2-line-through p q*)
unfolding *proj2-line-through-def*
using *proj2-points-define-line*
and *someI-ex* [*of* λ *l*. *proj2-incident p l* $\wedge$ *proj2-incident q l*]
by *simp-all*

lemma *proj2-line-through-unique*:
assumes *p* $\neq$ *q* **and** *proj2-incident p l* **and** *proj2-incident q l*
shows *l* = *proj2-line-through p q*
proof –
let *?l'* = *proj2-line-rep l*

let *?m* = *proj2-line-through p q*
let *?m′* = *proj2-line-rep ?m*
let *?p′* = *proj2-rep p*
let *?q′* = *proj2-rep q*
let *?A* = {*?p′*, *?q′*}
let *?B* = *insert ?m′ ?A*
from *proj2-line-through-incident*
have *proj2-incident p ?m* **and** *proj2-incident q ?m* **by** *simp-all*
with ‹*proj2-incident p l*› **and** ‹*proj2-incident q l*›
have *ortho*: $\bigwedge w.\ w\in ?A \Longrightarrow$ *orthogonal ?m′ w* $\bigwedge w.\ w\in ?A \Longrightarrow$ *orthogonal ?l′ w*
unfolding *proj2-incident-def* **and** *orthogonal-def*
by (*metis empty-iff inner-commute insert-iff*)+
from *proj2-rep-independent* **and** ‹$p \neq q$› **have** *independent ?A* **by** *simp*
from *proj2-line-rep-non-zero* **have** $?m' \neq 0$ **by** *simp*
with *orthogonal-independent* ‹*independent ?A*› *ortho*
have *independent ?B* **by** *auto*

from *proj2-rep-inj* **and** ‹$p \neq q$› **have** $?p' \neq ?q'$
unfolding *inj-on-def*
by *auto*
hence *card ?A* = *2* **by** *simp*
moreover have $?m' \notin ?A$
using *ortho*(*1*) *orthogonal-self proj2-line-rep-non-zero* **by** *auto*
ultimately have *card ?B* = *3* **by** *simp*
with *independent-is-basis* [*of ?B*] **and** ‹*independent ?B*›
have *is-basis ?B* **by** *simp*
with *basis-expand* **obtain** *c* **where** $?l' = (\sum\ v\in ?B.\ c\ v *_R v)$ **by** *auto*
let $?l'' = ?l' - c\ ?m' *_R ?m'$
from ‹$?l' = (\sum\ v\in ?B.\ c\ v *_R v)$› **and** ‹$?m' \notin ?A$›
have $?l'' = (\sum\ v\in ?A.\ c\ v *_R v)$ **by** *simp*
with *orthogonal-sum* [*of ?A*] *ortho*
have *orthogonal ?l′ ?l″* **and** *orthogonal ?m′ ?l″*
by (*simp-all add*: *scalar-equiv*)
from ‹*orthogonal ?m′ ?l″*›
have *orthogonal* $(c\ ?m' *_R ?m')$ *?l″* **by** (*simp add*: *orthogonal-clauses*)
with ‹*orthogonal ?l′ ?l″*›
have *orthogonal ?l″ ?l″* **by** (*simp add*: *orthogonal-clauses*)
with *orthogonal-self-eq-0* [*of ?l″*] **have** $?l'' = 0$ **by** *simp*
with *proj2-line-rep-dependent* [*of 1 l* − *c ?m′ ?m*] **show** *l* = *?m* **by** *simp*
qed

lemma *proj2-incident-unique*:
assumes *proj2-incident p l*
and *proj2-incident q l*
and *proj2-incident p m*
and *proj2-incident q m*
shows $p = q \vee l = m$
proof *cases*
assume $p = q$

thus $p = q \vee l = m$ **..**
next
assume $p \neq q$
with ‹*proj2-incident p l*› **and** ‹*proj2-incident q l*›
and *proj2-line-through-unique*
have l = *proj2-line-through p q* **by** *simp*
moreover from ‹$p \neq q$› **and** ‹*proj2-incident p m*› **and** ‹*proj2-incident q m*›
have m = *proj2-line-through p q* **by** (*rule proj2-line-through-unique*)
ultimately show $p = q \vee l = m$ **by** *simp*
qed

lemma *proj2-lines-define-point*: $\exists$ *p. proj2-incident p l* $\wedge$ *proj2-incident p m*
proof −
let *?l′* = *L2P l*
let *?m′* = *L2P m*
from *proj2-points-define-line* [*of ?l′ ?m′*]
obtain *p′* **where** *proj2-incident ?l′ p′* $\wedge$ *proj2-incident ?m′ p′* **by** *auto*
hence *proj2-incident* (*L2P p′*) *l* $\wedge$ *proj2-incident* (*L2P p′*) *m*
unfolding *proj2-incident-def* **and** *proj2-line-rep-def*
by (*simp add*: *inner-commute*)
thus $\exists$ *p. proj2-incident p l* $\wedge$ *proj2-incident p m* **by** *auto*
qed

definition *proj2-intersection* :: *proj2-line* $\Rightarrow$ *proj2-line* $\Rightarrow$ *proj2* **where**
proj2-intersection l m $\triangleq$ *L2P* (*proj2-line-through* (*L2P l*) (*L2P m*))

lemma *proj2-incident-switch*:
assumes *proj2-incident p l*
shows *proj2-incident* (*L2P l*) (*P2L p*)
using *assms*
unfolding *proj2-incident-def* **and** *proj2-line-rep-def*
by (*simp add*: *inner-commute*)

lemma *proj2-intersection-incident*:
shows *proj2-incident* (*proj2-intersection l m*) *l*
and *proj2-incident* (*proj2-intersection l m*) *m*
using *proj2-line-through-incident*(*1*) [*of L2P l L2P m*]
and *proj2-line-through-incident*(*2*) [*of L2P m L2P l*]
and *proj2-incident-switch* [*of L2P l*]
and *proj2-incident-switch* [*of L2P m*]
unfolding *proj2-intersection-def*
by *simp-all*

lemma *proj2-intersection-unique*:
assumes $l \neq m$ **and** *proj2-incident p l* **and** *proj2-incident p m*
shows *p* = *proj2-intersection l m*
proof −
from ‹$l \neq m$› **have** *L2P l* $\neq$ *L2P m* **by** *auto*
from ‹*proj2-incident p l*› **and** ‹*proj2-incident p m*›

and *proj2-incident-switch*
have *proj2-incident* (*L2P l*) (*P2L p*) **and** *proj2-incident* (*L2P m*) (*P2L p*)
by *simp-all*
with ‹*L2P l* ≠ *L2P m*› **and** *proj2-line-through-unique*
have *P2L p* = *proj2-line-through* (*L2P l*) (*L2P m*) **by** *simp*
thus *p* = *proj2-intersection l m*
unfolding *proj2-intersection-def*
by (*simp add*: *P2L-to-L2P*)
qed

lemma *proj2-not-self-incident*:
¬ (*proj2-incident p* (*P2L p*))
unfolding *proj2-incident-def* **and** *proj2-line-rep-def*
using *proj2-rep-non-zero* **and** *inner-eq-zero-iff* [*of proj2-rep p*]
by *simp*

lemma *proj2-another-point-on-line*:
∃ *q*. *q* ≠ *p* ∧ *proj2-incident q l*
proof −
let *?m* = *P2L p*
let *?q* = *proj2-intersection l ?m*
from *proj2-intersection-incident*
have *proj2-incident ?q l* **and** *proj2-incident ?q ?m* **by** *simp-all*
from ‹*proj2-incident ?q ?m*› **and** *proj2-not-self-incident* **have** *?q* ≠ *p* **by** *auto*
with ‹*proj2-incident ?q l*› **show** ∃ *q*. *q* ≠ *p* ∧ *proj2-incident q l* **by** *auto*
qed

lemma *proj2-another-line-through-point*:
∃ *m*. *m* ≠ *l* ∧ *proj2-incident p m*
proof −
from *proj2-another-point-on-line*
obtain *q* **where** *q* ≠ *L2P l* ∧ *proj2-incident q* (*P2L p*) **by** *auto*
with *proj2-incident-switch* [*of q P2L p*]
have *P2L q* ≠ *l* ∧ *proj2-incident p* (*P2L q*) **by** *auto*
thus ∃ *m*. *m* ≠ *l* ∧ *proj2-incident p m* **..**
qed

lemma *proj2-incident-abs*:
assumes *v* ≠ *0* **and** *w* ≠ *0*
shows *proj2-incident* (*proj2-abs v*) (*proj2-line-abs w*) ⟷ *v* · *w* = *0*
proof −
from ‹*v* ≠ *0*› **and** *proj2-rep-abs2*
obtain *j* **where** *j* ≠ *0* **and** *proj2-rep* (*proj2-abs v*) = *j* $*_R$ *v* **by** *auto*

from ‹*w* ≠ *0*› **and** *proj2-line-rep-abs*
obtain *k* **where** *k* ≠ *0*
and *proj2-line-rep* (*proj2-line-abs w*) = *k* $*_R$ *w*
by *auto*
with ‹*j* ≠ *0*› **and** ‹*proj2-rep* (*proj2-abs v*) = *j* $*_R$ *v*›

show *proj2-incident* (*proj2-abs v*) (*proj2-line-abs w*) $\longleftrightarrow$ $v \cdot w = 0$
unfolding *proj2-incident-def*
by (*simp add*: *dot-scaleR-mult*)
qed

lemma *proj2-incident-left-abs*:
assumes $v \neq 0$
shows *proj2-incident* (*proj2-abs v*) *l* $\longleftrightarrow$ $v \cdot$ (*proj2-line-rep l*) $= 0$
proof –
have *proj2-line-rep l* $\neq 0$ **by** (*rule proj2-line-rep-non-zero*)
with ‹$v \neq 0$› **and** *proj2-incident-abs* [*of v proj2-line-rep l*]
show *proj2-incident* (*proj2-abs v*) *l* $\longleftrightarrow$ $v \cdot$ (*proj2-line-rep l*) $= 0$ **by** *simp*
qed

lemma *proj2-incident-right-abs*:
assumes $v \neq 0$
shows *proj2-incident p* (*proj2-line-abs v*) $\longleftrightarrow$ (*proj2-rep p*) $\cdot v = 0$
proof –
have *proj2-rep p* $\neq 0$ **by** (*rule proj2-rep-non-zero*)
with ‹$v \neq 0$› **and** *proj2-incident-abs* [*of proj2-rep p v*]
show *proj2-incident p* (*proj2-line-abs v*) $\longleftrightarrow$ (*proj2-rep p*) $\cdot v = 0$
by (*simp add*: *proj2-abs-rep*)
qed

definition *proj2-set-Col* :: *proj2 set* $\Rightarrow$ *bool* **where**
proj2-set-Col S $\triangleq$ $\exists$ *l*. $\forall$ *p*$\in$*S*. *proj2-incident p l*

lemma *proj2-subset-Col*:
assumes $T \subseteq S$ **and** *proj2-set-Col S*
shows *proj2-set-Col T*
using ‹$T \subseteq S$› **and** ‹*proj2-set-Col S*›
by (*unfold proj2-set-Col-def*) *auto*

definition *proj2-no-3-Col* :: *proj2 set* $\Rightarrow$ *bool* **where**
proj2-no-3-Col S $\triangleq$ *card S* $= 4 \wedge (\forall$ *p*$\in$*S*. $\neg$ *proj2-set-Col* $(S - \{p\}))$

lemma *proj2-Col-iff-not-invertible*:
proj2-Col p q r
$\longleftrightarrow$ $\neg$ *invertible* (*vector* [*proj2-rep p*, *proj2-rep q*, *proj2-rep r*] :: *real^3^3*)
(**is** - $\longleftrightarrow$ $\neg$ *invertible* (*vector* [*?u*, *?v*, *?w*]))
proof –
let *?M* = *vector* [*?u*,*?v*,*?w*] :: *real^3^3*
have *proj2-Col p q r* $\longleftrightarrow$ ($\exists$ *x*. $x \neq 0 \wedge x$ *v** *?M* $= 0$)
proof
assume *proj2-Col p q r*
then obtain *i* **and** *j* **and** *k*
where $i \neq 0 \vee j \neq 0 \vee k \neq 0$ **and** $i *_R$ *?u* $+ j *_R$ *?v* $+ k *_R$ *?w* $= 0$
unfolding *proj2-Col-def*
by *auto*

```
    let ?x = vector [i,j,k] :: real^3
    from ‹i ≠ 0 ∨ j ≠ 0 ∨ k ≠ 0›
    have ?x ≠ 0
      unfolding vector-def
      by (simp add: vec-eq-iff forall-3)
    moreover {
      from ‹i *R ?u + j *R ?v + k *R ?w = 0›
      have ?x v* ?M = 0
        unfolding vector-def and vector-matrix-mult-def
        by (simp add: sum-3 vec-eq-iff algebra-simps) }
    ultimately show ∃ x. x ≠ 0 ∧ x v* ?M = 0 by auto
  next
    assume ∃ x. x ≠ 0 ∧ x v* ?M = 0
    then obtain x where x ≠ 0 and x v* ?M = 0 by auto
    let ?i = x$1
    let ?j = x$2
    let ?k = x$3
   from ‹x ≠ 0› have ?i ≠ 0 ∨ ?j ≠ 0 ∨ ?k ≠ 0 by (simp add: vec-eq-iff forall-3)
    moreover {
      from ‹x v* ?M = 0›
      have ?i *R ?u + ?j *R ?v + ?k *R ?w = 0
        unfolding vector-matrix-mult-def and sum-3 and vector-def
        by (simp add: vec-eq-iff algebra-simps) }
    ultimately show proj2-Col p q r
      unfolding proj2-Col-def
      by auto
  qed
  also from matrix-right-invertible-ker [of ?M]
  have ... ⟷ ¬ (∃ M′. ?M ** M′ = mat 1) by auto
  also from matrix-left-right-inverse
  have ... ⟷ ¬ invertible ?M
    unfolding invertible-def
    by auto
  finally show proj2-Col p q r ⟷ ¬ invertible ?M .
qed

lemma not-invertible-iff-proj2-set-Col:
  ¬ invertible (vector [proj2-rep p, proj2-rep q, proj2-rep r] :: real^3^3)
  ⟷ proj2-set-Col {p,q,r}
  (is ¬ invertible ?M ⟷ -)
proof −
  from left-invertible-iff-invertible
  have ¬ invertible ?M ⟷ ¬ (∃ M′. M′ ** ?M = mat 1) by auto
  also from matrix-left-invertible-ker [of ?M]
  have ... ⟷ (∃ y. y ≠ 0 ∧ ?M *v y = 0) by auto
  also have ... ⟷ (∃ l. ∀ s∈{p,q,r}. proj2-incident s l)
  proof
    assume ∃ y. y ≠ 0 ∧ ?M *v y = 0
    then obtain y where y ≠ 0 and ?M *v y = 0 by auto
```

```
    let ?l = proj2-line-abs y
    from ‹?M *v y = 0›
    have ∀ s∈{p,q,r}. proj2-rep s · y = 0
      unfolding vector-def
        and matrix-vector-mult-def
        and inner-vec-def
        and sum-3
      by (simp add: vec-eq-iff forall-3)
    with ‹y ≠ 0› and proj2-incident-right-abs
    have ∀ s∈{p,q,r}. proj2-incident s ?l by simp
    thus ∃ l. ∀ s∈{p,q,r}. proj2-incident s l ..
  next
    assume ∃ l. ∀ s∈{p,q,r}. proj2-incident s l
    then obtain l where ∀ s∈{p,q,r}. proj2-incident s l ..
    let ?y = proj2-line-rep l
    have ?y ≠ 0 by (rule proj2-line-rep-non-zero)
    moreover {
      from ‹∀ s∈{p,q,r}. proj2-incident s l›
      have ?M *v ?y = 0
        unfolding vector-def
          and matrix-vector-mult-def
          and inner-vec-def
          and sum-3
          and proj2-incident-def
        by (simp add: vec-eq-iff) }
    ultimately show ∃ y. y ≠ 0 ∧ ?M *v y = 0 by auto
  qed
  finally show ¬ invertible ?M ⟷ proj2-set-Col {p,q,r}
    unfolding proj2-set-Col-def .
qed

lemma proj2-Col-iff-set-Col:
  proj2-Col p q r ⟷ proj2-set-Col {p,q,r}
  by (simp add: proj2-Col-iff-not-invertible
    not-invertible-iff-proj2-set-Col)

lemma proj2-incident-Col:
  assumes proj2-incident p l and proj2-incident q l and proj2-incident r l
  shows proj2-Col p q r
proof −
  from ‹proj2-incident p l› and ‹proj2-incident q l› and ‹proj2-incident r l›
  have proj2-set-Col {p,q,r} by (unfold proj2-set-Col-def) auto
  thus proj2-Col p q r by (subst proj2-Col-iff-set-Col)
qed

lemma proj2-incident-iff-Col:
  assumes p ≠ q and proj2-incident p l and proj2-incident q l
  shows proj2-incident r l ⟷ proj2-Col p q r
proof
```

assume *proj2-incident r l*
with ‹*proj2-incident p l*› **and** ‹*proj2-incident q l*›
show *proj2-Col p q r* **by** (*rule proj2-incident-Col*)
next
assume *proj2-Col p q r*
hence *proj2-set-Col* {*p*,*q*,*r*} **by** (*simp add*: *proj2-Col-iff-set-Col*)
then obtain *m* **where** $\forall$ *s*$\in${*p*,*q*,*r*}. *proj2-incident s m*
unfolding *proj2-set-Col-def* **..**
hence *proj2-incident p m* **and** *proj2-incident q m* **and** *proj2-incident r m*
by *simp-all*
from ‹$p \neq q$› **and** ‹*proj2-incident p l*› **and** ‹*proj2-incident q l*›
and ‹*proj2-incident p m*› **and** ‹*proj2-incident q m*›
and *proj2-incident-unique*
have $m = l$ **by** *auto*
with ‹*proj2-incident r m*› **show** *proj2-incident r l* **by** *simp*
qed

lemma *proj2-incident-iff*:
assumes $p \neq q$ **and** *proj2-incident p l* **and** *proj2-incident q l*
shows *proj2-incident r l*
$\longleftrightarrow r = p \vee (\exists\ k.\ r =$ *proj2-abs* ($k *_R$ *proj2-rep p* + *proj2-rep q*))
proof −
from ‹$p \neq q$› **and** ‹*proj2-incident p l*› **and** ‹*proj2-incident q l*›
have *proj2-incident r l* $\longleftrightarrow$ *proj2-Col p q r* **by** (*rule proj2-incident-iff-Col*)
with ‹$p \neq q$› **and** *proj2-Col-iff*
show *proj2-incident r l*
$\longleftrightarrow r = p \vee (\exists\ k.\ r =$ *proj2-abs* ($k *_R$ *proj2-rep p* + *proj2-rep q*))
by *simp*
qed

lemma *not-proj2-set-Col-iff-span*:
assumes *card S* = *3*
shows $\neg$ *proj2-set-Col S* $\longleftrightarrow$ *span* (*proj2-rep* ‘ *S*) = *UNIV*
proof −
from ‹*card S* = *3*› **and** *choose-3* [*of S*]
obtain *p* **and** *q* **and** *r* **where** *S* = {*p*,*q*,*r*} **by** *auto*
let *?u* = *proj2-rep p*
let *?v* = *proj2-rep q*
let *?w* = *proj2-rep r*
let *?M* = *vector* [*?u*, *?v*, *?w*] :: *real^3^3*
from ‹*S* = {*p*,*q*,*r*}› **and** *not-invertible-iff-proj2-set-Col* [*of p q r*]
have $\neg$ *proj2-set-Col S* $\longleftrightarrow$ *invertible ?M* **by** *auto*
also from *left-invertible-iff-invertible*
have ... $\longleftrightarrow$ ($\exists$ *N*. *N* ** *?M* = *mat 1*) **..**
also from *matrix-left-invertible-span-rows*
have ... $\longleftrightarrow$ *span* (*rows ?M*) = *UNIV* **by** *auto*
finally have $\neg$ *proj2-set-Col S* $\longleftrightarrow$ *span* (*rows ?M*) = *UNIV* **.**

have *rows ?M* = {*?u*, *?v*, *?w*}

proof
{ **fix** x
assume $x \in rows\ ?M$
then obtain $i :: 3$ **where** $x = ?M \$ i$
unfolding *rows-def* **and** *row-def*
by (*auto simp add*: *vec-lambda-beta vec-lambda-eta*)
with *exhaust-3* **have** $x = ?u \lor x = ?v \lor x = ?w$
unfolding *vector-def*
by *auto*
hence $x \in \{?u, ?v, ?w\}$ **by** *simp* }
thus $rows\ ?M \subseteq \{?u, ?v, ?w\}$ **..**
{ **fix** x
assume $x \in \{?u, ?v, ?w\}$
hence $x = ?u \lor x = ?v \lor x = ?w$ **by** *simp*
hence $x = ?M \$ 1 \lor x = ?M \$ 2 \lor x = ?M \$ 3$
unfolding *vector-def*
by *simp*
hence $x \in rows\ ?M$
unfolding *rows-def row-def vec-lambda-eta*
by *blast* }
thus $\{?u, ?v, ?w\} \subseteq rows\ ?M$ **..**
qed
with ‹$S = \{p,q,r\}$›
have $rows\ ?M = proj2\text{-}rep$ ‘ S
unfolding *image-def*
by *auto*
with ‹$\neg$ *proj2-set-Col* $S \longleftrightarrow span\ (rows\ ?M) = UNIV$›
show $\neg$ *proj2-set-Col* $S \longleftrightarrow span\ (proj2\text{-}rep$ ‘ $S) = UNIV$ **by** *simp*
qed

lemma *proj2-no-3-Col-span*:
assumes *proj2-no-3-Col* S **and** $p \in S$
shows *span* (*proj2-rep* ‘ $(S - \{p\})) = UNIV$
proof −
from ‹*proj2-no-3-Col* S› **have** *card* $S = 4$ **unfolding** *proj2-no-3-Col-def* **..**
with ‹$p \in S$› **and** ‹*card* $S = 4$› **and** *card-gt-0-diff-singleton* [*of* S p]
have *card* $(S - \{p\}) = 3$ **by** *simp*

from ‹*proj2-no-3-Col* S› **and** ‹$p \in S$›
have $\neg$ *proj2-set-Col* $(S - \{p\})$
unfolding *proj2-no-3-Col-def*
by *simp*
with ‹*card* $(S - \{p\}) = 3$› **and** *not-proj2-set-Col-iff-span*
show *span* (*proj2-rep* ‘ $(S - \{p\})) = UNIV$ **by** *simp*
qed

lemma *fourth-proj2-no-3-Col*:
assumes $\neg$ *proj2-Col* $p\ q\ r$
shows $\exists\ s.$ *proj2-no-3-Col* $\{s,r,p,q\}$

proof −
  from ‹¬ *proj2-Col p q r*› **and** *proj2-Col-coincide* **have** $p \neq q$ **by** *auto*
  hence *card* $\{p,q\} = 2$ **by** *simp*

  from ‹¬ *proj2-Col p q r*› **and** *proj2-Col-coincide* **and** *proj2-Col-permute*
  have $r \notin \{p,q\}$ **by** *fast*
  with ‹*card* $\{p,q\} = 2$› **have** *card* $\{r,p,q\} = 3$ **by** *simp*

  have *finite* $\{r,p,q\}$ **by** *simp*

  let *?s* = *proj2-abs* ($\sum$ $t\in\{r,p,q\}$. *proj2-rep t*)
  have $\exists$ *j*. ($\sum$ $t\in\{r,p,q\}$. *proj2-rep t*) = *j* $*_R$ *proj2-rep ?s*
  proof *cases*
    assume ($\sum$ $t\in\{r,p,q\}$. *proj2-rep t*) = *0*
    hence ($\sum$ $t\in\{r,p,q\}$. *proj2-rep t*) = *0* $*_R$ *proj2-rep ?s* **by** *simp*
    thus $\exists$ *j*. ($\sum$ $t\in\{r,p,q\}$. *proj2-rep t*) = *j* $*_R$ *proj2-rep ?s* **..**
  next
    assume ($\sum$ $t\in\{r,p,q\}$. *proj2-rep t*) $\neq$ *0*
    with *proj2-rep-abs2*
    obtain *k* **where** $k \neq 0$
      and *proj2-rep ?s* = *k* $*_R$ ($\sum$ $t\in\{r,p,q\}$. *proj2-rep t*)
      by *auto*
    hence (*1*/*k*) $*_R$ *proj2-rep ?s* = ($\sum$ $t\in\{r,p,q\}$. *proj2-rep t*) **by** *simp*
    from *this* [*symmetric*]
    show $\exists$ *j*. ($\sum$ $t\in\{r,p,q\}$. *proj2-rep t*) = *j* $*_R$ *proj2-rep ?s* **..**
  qed
  then obtain *j* **where** ($\sum$ $t\in\{r,p,q\}$. *proj2-rep t*) = *j* $*_R$ *proj2-rep ?s* **..**
  let *?c* = λ *t*. *if t = ?s then 1* − *j else 1*
  from ‹$p \neq q$› **have** *?c p* $\neq$ *0* $\vee$ *?c q* $\neq$ *0* **by** *simp*

  let *?d* = λ *t*. *if t = ?s then j else* −*1*

  let *?S* = {*?s*,*r*,*p*,*q*}

  have *?s* $\notin$ $\{r,p,q\}$
  proof
    assume *?s* $\in$ $\{r,p,q\}$

    from ‹$r \notin \{p,q\}$› **and** ‹$p \neq q$›
    have *?c r* $*_R$ *proj2-rep r* + *?c p* $*_R$ *proj2-rep p* + *?c q* $*_R$ *proj2-rep q*
      = ($\sum$ $t\in\{r,p,q\}$. *?c t* $*_R$ *proj2-rep t*)
      by (*simp add*: *sum.insert* [*of* - - λ *t*. *?c t* $*_R$ *proj2-rep t*])
    also from ‹*finite* $\{r,p,q\}$› **and** ‹*?s* $\in$ $\{r,p,q\}$›
    have . . . = *?c ?s* $*_R$ *proj2-rep ?s* + ($\sum$ $t\in\{r,p,q\}-\{?s\}$. *?c t* $*_R$ *proj2-rep t*)
      by (*simp only*:
        sum.remove [*of* $\{r,p,q\}$ *?s* λ *t*. *?c t* $*_R$ *proj2-rep t*])
    also have . . .
      = −*j* $*_R$ *proj2-rep ?s* + (*proj2-rep ?s* + ($\sum$ $t\in\{r,p,q\}-\{?s\}$. *proj2-rep t*))
      by (*simp add*: *algebra-simps*)

```
  also from ‹finite {r,p,q}› and ‹?s ∈ {r,p,q}›
  have ... = −j *R proj2-rep ?s + (∑ t∈{r,p,q}. proj2-rep t)
    by (simp only:
      sum.remove [of {r,p,q} ?s λ t. proj2-rep t,symmetric])
  also from ‹(∑ t∈{r,p,q}. proj2-rep t) = j *R proj2-rep ?s›
  have ... = 0 by simp
  finally
  have ?c r *R proj2-rep r + ?c p *R proj2-rep p + ?c q *R proj2-rep q = 0
    .
  with ‹?c p ≠ 0 ∨ ?c q ≠ 0›
  have proj2-Col p q r
    by (unfold proj2-Col-def) (auto simp add: algebra-simps)
  with ‹¬ proj2-Col p q r› show False ..
qed
with ‹card {r,p,q} = 3› have card ?S = 4 by simp

from ‹¬ proj2-Col p q r› and proj2-Col-permute
have ¬ proj2-Col r p q by fast
hence ¬ proj2-set-Col {r,p,q} by (subst proj2-Col-iff-set-Col [symmetric])

have ∀ u∈?S. ¬ proj2-set-Col (?S − {u})
proof
  fix u
  assume u ∈ ?S
  with ‹card ?S = 4› have card (?S − {u}) = 3 by simp
  show ¬ proj2-set-Col (?S − {u})
  proof cases
    assume u = ?s
    with ‹?s ∉ {r,p,q}› have ?S − {u} = {r,p,q} by simp
    with ‹¬ proj2-set-Col {r,p,q}› show ¬ proj2-set-Col (?S − {u}) by simp
  next
    assume u ≠ ?s
    hence insert ?s ({r,p,q} − {u}) = ?S − {u} by auto

    from ‹finite {r,p,q}› have finite ({r,p,q} − {u}) by simp

    from ‹?s ∉ {r,p,q}› have ?s ∉ {r,p,q} − {u} by simp
    hence ∀ t∈{r,p,q}−{u}. ?d t = −1 by auto

    from ‹u ≠ ?s› and ‹u ∈ ?S› have u ∈ {r,p,q} by simp
    hence (∑ t∈{r,p,q}. proj2-rep t)
      = proj2-rep u + (∑ t∈{r,p,q}−{u}. proj2-rep t)
      by (simp add: sum.remove)
    with ‹(∑ t∈{r,p,q}. proj2-rep t) = j *R proj2-rep ?s›
    have proj2-rep u
      = j *R proj2-rep ?s − (∑ t∈{r,p,q}−{u}. proj2-rep t)
      by simp
    also from ‹∀ t∈{r,p,q}−{u}. ?d t = −1›
    have ... = j *R proj2-rep ?s + (∑ t∈{r,p,q}−{u}. ?d t *R proj2-rep t)
```

```
      by (simp add: sum-negf)
    also from ‹finite ({r,p,q} − {u})› and ‹?s ∉ {r,p,q} − {u}›
    have ... = (∑ t∈insert ?s ({r,p,q}−{u}). ?d t *R proj2-rep t)
      by (simp add: sum.insert)
    also from ‹insert ?s ({r,p,q} − {u}) = ?S − {u}›
    have ... = (∑ t∈?S−{u}. ?d t *R proj2-rep t) by simp
    finally have proj2-rep u = (∑ t∈?S−{u}. ?d t *R proj2-rep t) .
    moreover
    have ∀ t∈?S−{u}. ?d t *R proj2-rep t ∈ span (proj2-rep ` (?S − {u}))
      by (simp add: span-clauses)
    ultimately have proj2-rep u ∈ span (proj2-rep ` (?S − {u}))
      by (metis (no-types, lifting) span-sum)

    have ∀ t∈{r,p,q}. proj2-rep t ∈ span (proj2-rep ` (?S − {u}))
    proof
      fix t
      assume t ∈ {r,p,q}
      show proj2-rep t ∈ span (proj2-rep ` (?S − {u}))
      proof cases
        assume t = u
        from ‹proj2-rep u ∈ span (image proj2-rep (?S − {u}))›
        show proj2-rep t ∈ span (proj2-rep ` (?S − {u}))
          by (subst ‹t = u›)
      next
        assume t ≠ u
        with ‹t ∈ {r,p,q}›
        have proj2-rep t ∈ proj2-rep ` (?S − {u}) by simp
        with span-superset [of proj2-rep ` (?S − {u})]
        show proj2-rep t ∈ span (proj2-rep ` (?S − {u})) by fast
      qed
    qed
    hence proj2-rep ` {r,p,q} ⊆ span (proj2-rep ` (?S − {u}))
      by (simp only: image-subset-iff)
    hence
      span (proj2-rep ` {r,p,q}) ⊆ span (span (proj2-rep ` (?S − {u})))
      by (simp only: span-mono)
    hence span (proj2-rep ` {r,p,q}) ⊆ span (proj2-rep ` (?S − {u}))
      by (simp only: span-span)
    moreover
    from ‹¬ proj2-set-Col {r,p,q}›
      and ‹card {r,p,q} = 3›
      and not-proj2-set-Col-iff-span
    have span (proj2-rep ` {r,p,q}) = UNIV by simp
    ultimately have span (proj2-rep ` (?S − {u})) = UNIV by auto
    with ‹card (?S − {u}) = 3› and not-proj2-set-Col-iff-span
    show ¬ proj2-set-Col (?S − {u}) by simp
  qed
qed
with ‹card ?S = 4›
```

have *proj2-no-3-Col ?S* **by** (*unfold proj2-no-3-Col-def*) *fast*
thus ∃ *s. proj2-no-3-Col* {*s*,*r*,*p*,*q*} **..**
qed

lemma *proj2-set-Col-expand*:
assumes *proj2-set-Col S* **and** {*p*,*q*,*r*} ⊆ *S* **and** *p* ≠ *q* **and** *r* ≠ *p*
shows ∃ *k*. *r* = *proj2-abs* (*k* $*_R$ *proj2-rep p* + *proj2-rep q*)
proof −
from ‹*proj2-set-Col S*›
obtain *l* **where** ∀ *t*∈*S*. *proj2-incident t l* **unfolding** *proj2-set-Col-def* **..**
with ‹{*p*,*q*,*r*} ⊆ *S*› **and** ‹*p* ≠ *q*› **and** ‹*r* ≠ *p*› **and** *proj2-incident-iff* [*of p q l r*]
show ∃ *k*. *r* = *proj2-abs* (*k* $*_R$ *proj2-rep p* + *proj2-rep q*) **by** *simp*
qed

7.4 Collineations of the real projective plane

typedef *cltn2* =
(*Collect invertible* :: (*real*ˆ*3*ˆ*3*) *set*)//*invertible-proportionality*
proof
from *matrix-id-invertible* **have** (*mat 1* :: *real*ˆ*3*ˆ*3*) ∈ *Collect invertible*
by *simp*
thus *invertible-proportionality* ‘‘ {*mat 1*} ∈
(*Collect invertible* :: (*real*ˆ*3*ˆ*3*) *set*)//*invertible-proportionality*
unfolding *quotient-def*
by *auto*
qed

definition *cltn2-rep* :: *cltn2* ⇒ *real*ˆ*3*ˆ*3* **where**
cltn2-rep A ≜ ϵ *B*. *B* ∈ *Rep-cltn2 A*

definition *cltn2-abs* :: *real*ˆ*3*ˆ*3* ⇒ *cltn2* **where**
cltn2-abs B ≜ *Abs-cltn2* (*invertible-proportionality* ‘‘ {*B*})

definition *cltn2-independent* :: *cltn2 set* ⇒ *bool* **where**
cltn2-independent X ≜ *independent* {*cltn2-rep A* | *A*. *A* ∈ *X*}

definition *apply-cltn2* :: *proj2* ⇒ *cltn2* ⇒ *proj2* **where**
apply-cltn2 x A ≜ *proj2-abs* (*proj2-rep x v*∗ *cltn2-rep A*)

lemma *cltn2-rep-in*: *cltn2-rep B* ∈ *Rep-cltn2 B*
proof −
let *?A* = *cltn2-rep B*
from *quotient-element-nonempty* **and**
invertible-proportionality-equiv **and**
Rep-cltn2 [*of B*]
have ∃ *C*. *C* ∈ *Rep-cltn2 B*
by *auto*
with *someI-ex* [*of* λ *C*. *C* ∈ *Rep-cltn2 B*]
show *?A* ∈ *Rep-cltn2 B*

unfolding *cltn2-rep-def*
by *simp*
qed

lemma *cltn2-rep-invertible*: *invertible* (*cltn2-rep* *A*)
proof −
from
Union-quotient [*of Collect invertible invertible-proportionality*]
and *invertible-proportionality-equiv*
and *Rep-cltn2* [*of A*] **and** *cltn2-rep-in* [*of A*]
have *cltn2-rep A* ∈ *Collect invertible*
unfolding *quotient-def*
by *auto*
thus *invertible* (*cltn2-rep A*)
unfolding *invertible-proportionality-def*
by *simp*
qed

lemma *cltn2-rep-abs*:
fixes *A* :: *real^3^3*
assumes *invertible A*
shows (*A*, *cltn2-rep* (*cltn2-abs A*)) ∈ *invertible-proportionality*
proof −
from ‹*invertible A*›
have *invertible-proportionality* `` {*A*} ∈ (*Collect invertible* :: (*real^3^3*) *set*)//*invertible-proportionality*
unfolding *quotient-def*
by *auto*
with *Abs-cltn2-inverse*
have *Rep-cltn2* (*cltn2-abs A*) = *invertible-proportionality* `` {*A*}
unfolding *cltn2-abs-def*
by *simp*
with *cltn2-rep-in*
have *cltn2-rep* (*cltn2-abs A*) ∈ *invertible-proportionality* `` {*A*} **by** *auto*
thus (*A*, *cltn2-rep* (*cltn2-abs A*)) ∈ *invertible-proportionality* **by** *simp*
qed

lemma *cltn2-rep-abs2*:
assumes *invertible A*
shows ∃ *k*. *k* ≠ *0* ∧ *cltn2-rep* (*cltn2-abs A*) = *k* $*_R$ *A*
proof −
from ‹*invertible A*› **and** *cltn2-rep-abs*
have (*A*, *cltn2-rep* (*cltn2-abs A*)) ∈ *invertible-proportionality* **by** *simp*
then obtain *c* **where** *A* = *c* $*_R$ *cltn2-rep* (*cltn2-abs A*)
unfolding *invertible-proportionality-def* **and** *real-vector.proportionality-def*
by *auto*
with ‹*invertible A*› **and** *zero-not-invertible* **have** *c* ≠ *0* **by** *auto*
hence *1*/*c* ≠ *0* **by** *simp*

let *?k* = *1*/*c*

from ‹$A = c *_R$ *cltn2-rep* (*cltn2-abs* A)›
have $?k *_R A = ?k *_R c *_R$ *cltn2-rep* (*cltn2-abs* A) **by** *simp*
with ‹$c \neq 0$› **have** *cltn2-rep* (*cltn2-abs* A) $= ?k *_R A$ **by** *simp*
with ‹$?k \neq 0$›
show $\exists\ k.\ k \neq 0 \wedge$ *cltn2-rep* (*cltn2-abs* A) $= k *_R A$ **by** *blast*
qed

lemma *cltn2-abs-rep*: *cltn2-abs* (*cltn2-rep* A) $= A$
proof −
from *partition-Image-element*
[*of Collect invertible*
invertible-proportionality
Rep-cltn2 A
cltn2-rep A]
and *invertible-proportionality-equiv*
and *Rep-cltn2* [*of A*] **and** *cltn2-rep-in* [*of A*]
have *invertible-proportionality* ‘‘ {*cltn2-rep A*} = *Rep-cltn2 A*
by *simp*
with *Rep-cltn2-inverse*
show *cltn2-abs* (*cltn2-rep* A) $= A$
unfolding *cltn2-abs-def*
by *simp*
qed

lemma *cltn2-abs-mult*:
assumes $k \neq 0$ **and** *invertible A*
shows *cltn2-abs* ($k *_R A$) = *cltn2-abs A*
proof −
from ‹$k \neq 0$› **and** ‹*invertible A*› **and** *scalar-invertible*
have *invertible* ($k *_R A$) **by** *auto*
with ‹*invertible A*›
have ($k *_R A$, A) $\in$ *invertible-proportionality*
unfolding *invertible-proportionality-def*
and *real-vector.proportionality-def*
by (*auto simp add*: *zero-not-invertible*)
with *eq-equiv-class-iff*
[*of Collect invertible invertible-proportionality* $k *_R A$ A]
and *invertible-proportionality-equiv*
and ‹*invertible A*› **and** ‹*invertible* ($k *_R A$)›
have *invertible-proportionality* ‘‘ {$k *_R A$}
= *invertible-proportionality* ‘‘ {A}
by *simp*
thus *cltn2-abs* ($k *_R A$) = *cltn2-abs A*
unfolding *cltn2-abs-def*
by *simp*
qed

lemma *cltn2-abs-mult-rep*:
assumes $k \neq 0$

shows *cltn2-abs* $(k *_R$ *cltn2-rep* $A) = A$
using *cltn2-rep-invertible* **and** *cltn2-abs-mult* **and** *cltn2-abs-rep* **and** *assms*
by *simp*

lemma *apply-cltn2-abs*:
assumes $x \neq 0$ **and** *invertible* A
shows *apply-cltn2* (*proj2-abs* x) (*cltn2-abs* A) = *proj2-abs* (x $v*$ A)
proof −
from *proj2-rep-abs2* **and** ‹$x \neq 0$›
obtain k **where** $k \neq 0$ **and** *proj2-rep* (*proj2-abs* x) = $k *_R x$ **by** *auto*

from *cltn2-rep-abs2* **and** ‹*invertible* A›
obtain c **where** $c \neq 0$ **and** *cltn2-rep* (*cltn2-abs* A) = $c *_R A$ **by** *auto*

from ‹$k \neq 0$› **and** ‹$c \neq 0$› **have** $k * c \neq 0$ **by** *simp*

from ‹*proj2-rep* (*proj2-abs* x) = $k *_R x$› **and** ‹*cltn2-rep* (*cltn2-abs* A) = $c *_R A$›
have *proj2-rep* (*proj2-abs* x) $v*$ *cltn2-rep* (*cltn2-abs* A) = $(k*c) *_R (x\ v*\ A)$
by (*simp add*: *scaleR-vector-matrix-assoc vector-scaleR-matrix-ac*)
with ‹$k * c \neq 0$›
show *apply-cltn2* (*proj2-abs* x) (*cltn2-abs* A) = *proj2-abs* (x $v*$ A)
unfolding *apply-cltn2-def*
by (*simp add*: *proj2-abs-mult*)
qed

lemma *apply-cltn2-left-abs*:
assumes $v \neq 0$
shows *apply-cltn2* (*proj2-abs* v) C = *proj2-abs* (v $v*$ *cltn2-rep* C)
proof −
have *cltn2-abs* (*cltn2-rep* C) = C **by** (*rule cltn2-abs-rep*)
with ‹$v \neq 0$› **and** *cltn2-rep-invertible* **and** *apply-cltn2-abs* [*of v cltn2-rep C*]
show *apply-cltn2* (*proj2-abs* v) C = *proj2-abs* (v $v*$ *cltn2-rep* C)
by *simp*
qed

lemma *apply-cltn2-right-abs*:
assumes *invertible* M
shows *apply-cltn2* p (*cltn2-abs* M) = *proj2-abs* (*proj2-rep* p $v*$ M)
proof −
from *proj2-rep-non-zero* **and** ‹*invertible* M› **and** *apply-cltn2-abs*
have *apply-cltn2* (*proj2-abs* (*proj2-rep* p)) (*cltn2-abs* M)
= *proj2-abs* (*proj2-rep* p $v*$ M)
by *simp*
thus *apply-cltn2* p (*cltn2-abs* M) = *proj2-abs* (*proj2-rep* p $v*$ M)
by (*simp add*: *proj2-abs-rep*)
qed

lemma *non-zero-mult-rep-non-zero*:
assumes $v \neq 0$

shows $v\ v{*}\ cltn2\text{-}rep\ C \neq 0$
using ‹$v \neq 0$› **and** *cltn2-rep-invertible* **and** *times-invertible-eq-zero*
by *auto*

lemma *rep-mult-rep-non-zero*: $proj2\text{-}rep\ p\ v{*}\ cltn2\text{-}rep\ A \neq 0$
using *proj2-rep-non-zero*
by (*rule non-zero-mult-rep-non-zero*)

definition *cltn2-image* :: $proj2\ set \Rightarrow cltn2 \Rightarrow proj2\ set$ **where**
$cltn2\text{-}image\ P\ A \triangleq \{apply\text{-}cltn2\ p\ A \mid p.\ p \in P\}$

7.4.1 As a group

definition *cltn2-id* :: *cltn2* **where**
$cltn2\text{-}id \triangleq cltn2\text{-}abs\ (mat\ 1)$

definition *cltn2-compose* :: $cltn2 \Rightarrow cltn2 \Rightarrow cltn2$ **where**
$cltn2\text{-}compose\ A\ B \triangleq cltn2\text{-}abs\ (cltn2\text{-}rep\ A\ {**}\ cltn2\text{-}rep\ B)$

definition *cltn2-inverse* :: $cltn2 \Rightarrow cltn2$ **where**
$cltn2\text{-}inverse\ A \triangleq cltn2\text{-}abs\ (matrix\text{-}inv\ (cltn2\text{-}rep\ A))$

lemma *cltn2-compose-abs*:
assumes *invertible M* **and** *invertible N*
shows $cltn2\text{-}compose\ (cltn2\text{-}abs\ M)\ (cltn2\text{-}abs\ N) = cltn2\text{-}abs\ (M\ {**}\ N)$
proof −
from ‹*invertible M*› **and** ‹*invertible N*› **and** *invertible-mult*
have *invertible* $(M\ {**}\ N)$ **by** *auto*

from ‹*invertible M*› **and** ‹*invertible N*› **and** *cltn2-rep-abs2*
obtain j **and** k **where** $j \neq 0$ **and** $k \neq 0$
and $cltn2\text{-}rep\ (cltn2\text{-}abs\ M) = j *_R M$
and $cltn2\text{-}rep\ (cltn2\text{-}abs\ N) = k *_R N$
by *blast*

from ‹$j \neq 0$› **and** ‹$k \neq 0$› **have** $j * k \neq 0$ **by** *simp*

from ‹$cltn2\text{-}rep\ (cltn2\text{-}abs\ M) = j *_R M$› **and** ‹$cltn2\text{-}rep\ (cltn2\text{-}abs\ N) = k *_R N$›
have $cltn2\text{-}rep\ (cltn2\text{-}abs\ M)\ {**}\ cltn2\text{-}rep\ (cltn2\text{-}abs\ N)$
$= (j * k) *_R (M\ {**}\ N)$
by (*simp add*: *matrix-scalar-ac scalar-matrix-assoc* [*symmetric*])
with ‹$j * k \neq 0$› **and** ‹*invertible* $(M\ {**}\ N)$›
show $cltn2\text{-}compose\ (cltn2\text{-}abs\ M)\ (cltn2\text{-}abs\ N) = cltn2\text{-}abs\ (M\ {**}\ N)$
unfolding *cltn2-compose-def*
by (*simp add*: *cltn2-abs-mult*)
qed

lemma *cltn2-compose-left-abs*:

assumes *invertible M*
shows *cltn2-compose* (*cltn2-abs M*) *A* = *cltn2-abs* (*M* ∗∗ *cltn2-rep A*)
proof −
from ⟨*invertible M*⟩ **and** *cltn2-rep-invertible* **and** *cltn2-compose-abs*
have *cltn2-compose* (*cltn2-abs M*) (*cltn2-abs* (*cltn2-rep A*))
= *cltn2-abs* (*M* ∗∗ *cltn2-rep A*)
by *simp*
thus *cltn2-compose* (*cltn2-abs M*) *A* = *cltn2-abs* (*M* ∗∗ *cltn2-rep A*)
by (*simp add*: *cltn2-abs-rep*)
qed

lemma *cltn2-compose-right-abs*:
assumes *invertible M*
shows *cltn2-compose A* (*cltn2-abs M*) = *cltn2-abs* (*cltn2-rep A* ∗∗ *M*)
proof −
from ⟨*invertible M*⟩ **and** *cltn2-rep-invertible* **and** *cltn2-compose-abs*
have *cltn2-compose* (*cltn2-abs* (*cltn2-rep A*)) (*cltn2-abs M*)
= *cltn2-abs* (*cltn2-rep A* ∗∗ *M*)
by *simp*
thus *cltn2-compose A* (*cltn2-abs M*) = *cltn2-abs* (*cltn2-rep A* ∗∗ *M*)
by (*simp add*: *cltn2-abs-rep*)
qed

lemma *cltn2-abs-rep-abs-mult*:
assumes *invertible M* **and** *invertible N*
shows *cltn2-abs* (*cltn2-rep* (*cltn2-abs M*) ∗∗ *N*) = *cltn2-abs* (*M* ∗∗ *N*)
proof −
from ⟨*invertible M*⟩ **and** ⟨*invertible N*⟩
have *invertible* (*M* ∗∗ *N*) **by** (*simp add*: *invertible-mult*)

from ⟨*invertible M*⟩ **and** *cltn2-rep-abs2*
obtain *k* **where** $k \neq 0$ **and** *cltn2-rep* (*cltn2-abs M*) = $k *_R M$ **by** *auto*
from ⟨*cltn2-rep* (*cltn2-abs M*) = $k *_R M$⟩
have *cltn2-rep* (*cltn2-abs M*) ∗∗ *N* = $k *_R M$ ∗∗ *N* **by** *simp*
with ⟨$k \neq 0$⟩ **and** ⟨*invertible* (*M* ∗∗ *N*)⟩ **and** *cltn2-abs-mult*
show *cltn2-abs* (*cltn2-rep* (*cltn2-abs M*) ∗∗ *N*) = *cltn2-abs* (*M* ∗∗ *N*)
by (*simp add*: *scalar-matrix-assoc* [*symmetric*])
qed

lemma *cltn2-assoc*:
cltn2-compose (*cltn2-compose A B*) *C* = *cltn2-compose A* (*cltn2-compose B C*)
proof −
let *?A′* = *cltn2-rep A*
let *?B′* = *cltn2-rep B*
let *?C′* = *cltn2-rep C*
from *cltn2-rep-invertible*
have *invertible ?A′* **and** *invertible ?B′* **and** *invertible ?C′* **by** *simp-all*
with *invertible-mult*
have *invertible* (*?A′* ∗∗ *?B′*) **and** *invertible* (*?B′* ∗∗ *?C′*)

and *invertible* (*?A′* ∗∗ *?B′* ∗∗ *?C′*)
by *auto*
from ‹*invertible* (*?A′* ∗∗ *?B′*)› **and** ‹*invertible* *?C′*› **and** *cltn2-abs-rep-abs-mult*
have *cltn2-abs* (*cltn2-rep* (*cltn2-abs* (*?A′* ∗∗ *?B′*)) ∗∗ *?C′*)
= *cltn2-abs* (*?A′* ∗∗ *?B′* ∗∗ *?C′*)
by *simp*

from ‹*invertible* (*?B′* ∗∗ *?C′*)› **and** *cltn2-rep-abs2* [*of ?B′* ∗∗ *?C′*]
obtain *k* **where** $k \neq 0$
and *cltn2-rep* (*cltn2-abs* (*?B′* ∗∗ *?C′*)) = $k *_R$ (*?B′* ∗∗ *?C′*)
by *auto*
from ‹*cltn2-rep* (*cltn2-abs* (*?B′* ∗∗ *?C′*)) = $k *_R$ (*?B′* ∗∗ *?C′*)›
have *?A′* ∗∗ *cltn2-rep* (*cltn2-abs* (*?B′* ∗∗ *?C′*)) = $k *_R$ (*?A′* ∗∗ *?B′* ∗∗ *?C′*)
by (*simp add*: *matrix-scalar-ac matrix-mul-assoc scalar-matrix-assoc*)
with ‹$k \neq 0$› **and** ‹*invertible* (*?A′* ∗∗ *?B′* ∗∗ *?C′*)›
and *cltn2-abs-mult* [*of k ?A′* ∗∗ *?B′* ∗∗ *?C′*]
have *cltn2-abs* (*?A′* ∗∗ *cltn2-rep* (*cltn2-abs* (*?B′* ∗∗ *?C′*)))
= *cltn2-abs* (*?A′* ∗∗ *?B′* ∗∗ *?C′*)
by *simp*
with ‹*cltn2-abs* (*cltn2-rep* (*cltn2-abs* (*?A′* ∗∗ *?B′*)) ∗∗ *?C′*)
= *cltn2-abs* (*?A′* ∗∗ *?B′* ∗∗ *?C′*)›
show
cltn2-compose (*cltn2-compose A B*) *C* = *cltn2-compose A* (*cltn2-compose B C*)
unfolding *cltn2-compose-def*
by *simp*
qed

lemma *cltn2-left-id*: *cltn2-compose cltn2-id A* = *A*
proof −
let *?A′* = *cltn2-rep A*
from *cltn2-rep-invertible* **have** *invertible ?A′* **by** *simp*
with *matrix-id-invertible* **and** *cltn2-abs-rep-abs-mult* [*of mat 1 ?A′*]
have *cltn2-compose cltn2-id A* = *cltn2-abs* (*cltn2-rep A*)
unfolding *cltn2-compose-def* **and** *cltn2-id-def*
by (*auto simp add*: *matrix-mul-lid*)
with *cltn2-abs-rep* **show** *cltn2-compose cltn2-id A* = *A* **by** *simp*
qed

lemma *cltn2-left-inverse*: *cltn2-compose* (*cltn2-inverse A*) *A* = *cltn2-id*
proof −
let *?M* = *cltn2-rep A*
let *?M′* = *matrix-inv ?M*
from *cltn2-rep-invertible* **have** *invertible ?M* **by** *simp*
with *matrix-inv-invertible* **have** *invertible ?M′* **by** *auto*
with ‹*invertible ?M*› **and** *cltn2-abs-rep-abs-mult*
have *cltn2-compose* (*cltn2-inverse A*) *A* = *cltn2-abs* (*?M′* ∗∗ *?M*)
unfolding *cltn2-compose-def* **and** *cltn2-inverse-def*
by *simp*
with ‹*invertible ?M*›

show *cltn2-compose* (*cltn2-inverse A*) *A* = *cltn2-id*
unfolding *cltn2-id-def*
by (*simp add*: *matrix-inv*)
qed

lemma *cltn2-left-inverse-ex*:
∃ *B*. *cltn2-compose B A* = *cltn2-id*
using *cltn2-left-inverse* **..**

interpretation *cltn2*:
group (|*carrier* = *UNIV*, *mult* = *cltn2-compose*, *one* = *cltn2-id*|)
using *cltn2-assoc* **and** *cltn2-left-id* **and** *cltn2-left-inverse-ex*
and *groupI* [*of* (|*carrier* = *UNIV*, *mult* = *cltn2-compose*, *one* = *cltn2-id*|)]
by *simp-all*

lemma *cltn2-inverse-inv* [*simp*]:
inv(|*carrier* = *UNIV*, *mult* = *cltn2-compose*, *one* = *cltn2-id*|) *A*
= *cltn2-inverse A*
using *cltn2-left-inverse* [*of A*] **and** *cltn2.inv-equality*
by *simp*

lemmas *cltn2-inverse-id* [*simp*] = *cltn2.inv-one* [*simplified*]
and *cltn2-inverse-compose* = *cltn2.inv-mult-group* [*simplified*]

7.4.2 As a group action

lemma *apply-cltn2-id* [*simp*]: *apply-cltn2 p cltn2-id* = *p*
proof −
from *matrix-id-invertible* **and** *apply-cltn2-right-abs*
have *apply-cltn2 p cltn2-id* = *proj2-abs* (*proj2-rep p v* mat 1*)
unfolding *cltn2-id-def* **by** *blast*
thus *apply-cltn2 p cltn2-id* = *p*
by (*simp add*: *proj2-abs-rep*)
qed

lemma *apply-cltn2-compose*:
apply-cltn2 (*apply-cltn2 p A*) *B* = *apply-cltn2 p* (*cltn2-compose A B*)
proof −
from *rep-mult-rep-non-zero* **and** *cltn2-rep-invertible* **and** *apply-cltn2-abs*
have *apply-cltn2* (*apply-cltn2 p A*) (*cltn2-abs* (*cltn2-rep B*))
= *proj2-abs* ((*proj2-rep p v* cltn2-rep A*) *v* cltn2-rep B*)
unfolding *apply-cltn2-def* [*of p A*]
by *simp*
hence *apply-cltn2* (*apply-cltn2 p A*) *B*
= *proj2-abs* (*proj2-rep p v** (*cltn2-rep A ** cltn2-rep B*))
by (*simp add*: *cltn2-abs-rep vector-matrix-mul-assoc*)

from *cltn2-rep-invertible* **and** *invertible-mult*
have *invertible* (*cltn2-rep A ** cltn2-rep B*) **by** *auto*

with *apply-cltn2-right-abs*
have *apply-cltn2 p* (*cltn2-compose A B*)
= *proj2-abs* (*proj2-rep p v*∗ (*cltn2-rep A* ∗∗ *cltn2-rep B*))
unfolding *cltn2-compose-def*
by *simp*
with ‹*apply-cltn2* (*apply-cltn2 p A*) *B*
= *proj2-abs* (*proj2-rep p v*∗ (*cltn2-rep A* ∗∗ *cltn2-rep B*))›
show *apply-cltn2* (*apply-cltn2 p A*) *B* = *apply-cltn2 p* (*cltn2-compose A B*)
by *simp*
qed

interpretation *cltn2*:
action (|*carrier* = *UNIV*, *mult* = *cltn2-compose*, *one* = *cltn2-id*|) *apply-cltn2*
proof
let *?G* = (|*carrier* = *UNIV*, *mult* = *cltn2-compose*, *one* = *cltn2-id*|)
fix *p*
show *apply-cltn2 p* $\mathbf{1}_{?G}$ = *p* **by** *simp*
fix *A B*
have *apply-cltn2* (*apply-cltn2 p A*) *B* = *apply-cltn2 p* ($A \otimes_{?G} B$)
by *simp* (*rule apply-cltn2-compose*)
thus *A* ∈ *carrier ?G* ∧ *B* ∈ *carrier ?G*
⟶ *apply-cltn2* (*apply-cltn2 p A*) *B* = *apply-cltn2 p* ($A \otimes_{?G} B$)
..
qed

definition *cltn2-transpose* :: *cltn2* ⇒ *cltn2* **where**
cltn2-transpose A ≜ *cltn2-abs* (*transpose* (*cltn2-rep A*))

definition *apply-cltn2-line* :: *proj2-line* ⇒ *cltn2* ⇒ *proj2-line* **where**
apply-cltn2-line l A
≜ *P2L* (*apply-cltn2* (*L2P l*) (*cltn2-transpose* (*cltn2-inverse A*)))

lemma *cltn2-transpose-abs*:
assumes *invertible M*
shows *cltn2-transpose* (*cltn2-abs M*) = *cltn2-abs* (*transpose M*)
proof −
from ‹*invertible M*› **and** *transpose-invertible* **have** *invertible* (*transpose M*) **by** *auto*

from ‹*invertible M*› **and** *cltn2-rep-abs2*
obtain *k* **where** *k* ≠ *0* **and** *cltn2-rep* (*cltn2-abs M*) = $k *_R M$ **by** *auto*

from ‹*cltn2-rep* (*cltn2-abs M*) = $k *_R M$›
have *transpose* (*cltn2-rep* (*cltn2-abs M*)) = $k *_R$ *transpose M*
by (*simp add*: *transpose-scalar*)
with ‹*k* ≠ *0*› **and** ‹*invertible* (*transpose M*)›
show *cltn2-transpose* (*cltn2-abs M*) = *cltn2-abs* (*transpose M*)
unfolding *cltn2-transpose-def*
by (*simp add*: *cltn2-abs-mult*)

qed

lemma *cltn2-transpose-compose*:
  cltn2-transpose (*cltn2-compose A B*)
  = *cltn2-compose* (*cltn2-transpose B*) (*cltn2-transpose A*)
proof –
  from *cltn2-rep-invertible*
  have *invertible* (*cltn2-rep A*) **and** *invertible* (*cltn2-rep B*)
    by *simp-all*
  with *transpose-invertible*
  have *invertible* (*transpose* (*cltn2-rep A*))
    and *invertible* (*transpose* (*cltn2-rep B*))
    by *auto*

  from ‹*invertible* (*cltn2-rep A*)› **and** ‹*invertible* (*cltn2-rep B*)›
    and *invertible-mult*
  have *invertible* (*cltn2-rep A* ∗∗ *cltn2-rep B*) **by** *auto*
  with ‹*invertible* (*cltn2-rep A* ∗∗ *cltn2-rep B*)› **and** *cltn2-transpose-abs*
  have *cltn2-transpose* (*cltn2-compose A B*)
    = *cltn2-abs* (*transpose* (*cltn2-rep A* ∗∗ *cltn2-rep B*))
    unfolding *cltn2-compose-def*
    by *simp*
  also have . . . = *cltn2-abs* (*transpose* (*cltn2-rep B*) ∗∗ *transpose* (*cltn2-rep A*))
    by (*simp add*: *matrix-transpose-mul*)
  also from ‹*invertible* (*transpose* (*cltn2-rep B*))›
    and ‹*invertible* (*transpose* (*cltn2-rep A*))›
    and *cltn2-compose-abs*
  have . . . = *cltn2-compose* (*cltn2-transpose B*) (*cltn2-transpose A*)
    unfolding *cltn2-transpose-def*
    by *simp*
  finally show *cltn2-transpose* (*cltn2-compose A B*)
    = *cltn2-compose* (*cltn2-transpose B*) (*cltn2-transpose A*) **.**
qed

lemma *cltn2-transpose-transpose*: *cltn2-transpose* (*cltn2-transpose A*) = *A*
proof –
  from *cltn2-rep-invertible* **have** *invertible* (*cltn2-rep A*) **by** *simp*
  with *transpose-invertible* **have** *invertible* (*transpose* (*cltn2-rep A*)) **by** *auto*
  with *cltn2-transpose-abs* [*of transpose* (*cltn2-rep A*)]
  have
   cltn2-transpose (*cltn2-transpose A*) = *cltn2-abs* (*transpose* (*transpose* (*cltn2-rep A*)))
    unfolding *cltn2-transpose-def* [*of A*]
    by *simp*
  with *cltn2-abs-rep* **and** *transpose-transpose* [*of cltn2-rep A*]
  show *cltn2-transpose* (*cltn2-transpose A*) = *A* **by** *simp*
qed

lemma *cltn2-transpose-id* [*simp*]: *cltn2-transpose cltn2-id* = *cltn2-id*

using *cltn2-transpose-abs*
unfolding *cltn2-id-def*
by (*simp add*: *transpose-mat matrix-id-invertible*)

lemma *apply-cltn2-line-id* [*simp*]: *apply-cltn2-line l cltn2-id* = *l*
unfolding *apply-cltn2-line-def*
by *simp*

lemma *apply-cltn2-line-compose*:
apply-cltn2-line (*apply-cltn2-line l A*) *B*
= *apply-cltn2-line l* (*cltn2-compose A B*)
proof −
have *cltn2-compose*
(*cltn2-transpose* (*cltn2-inverse A*)) (*cltn2-transpose* (*cltn2-inverse B*))
= *cltn2-transpose* (*cltn2-inverse* (*cltn2-compose A B*))
by (*simp add*: *cltn2-transpose-compose cltn2-inverse-compose*)
thus *apply-cltn2-line* (*apply-cltn2-line l A*) *B*
= *apply-cltn2-line l* (*cltn2-compose A B*)
unfolding *apply-cltn2-line-def*
by (*simp add*: *apply-cltn2-compose*)
qed

interpretation *cltn2-line*:
action
(|*carrier* = *UNIV*, *mult* = *cltn2-compose*, *one* = *cltn2-id*|)
apply-cltn2-line
proof
let *?G* = (|*carrier* = *UNIV*, *mult* = *cltn2-compose*, *one* = *cltn2-id*|)
fix *l*
show *apply-cltn2-line l* $\mathbf{1}_{?G}$ = *l* **by** *simp*
fix *A B*
have *apply-cltn2-line* (*apply-cltn2-line l A*) *B*
= *apply-cltn2-line l* ($A \otimes_{?G} B$)
by *simp* (*rule apply-cltn2-line-compose*)
thus $A \in$ *carrier ?G* $\wedge$ $B \in$ *carrier ?G*
$\longrightarrow$ *apply-cltn2-line* (*apply-cltn2-line l A*) *B*
= *apply-cltn2-line l* ($A \otimes_{?G} B$)
..
qed

lemmas *apply-cltn2-inv* [*simp*] = *cltn2.act-act-inv* [*simplified*]
lemmas *apply-cltn2-line-inv* [*simp*] = *cltn2-line.act-act-inv* [*simplified*]

lemma *apply-cltn2-line-alt-def*:
apply-cltn2-line l A
= *proj2-line-abs* (*cltn2-rep* (*cltn2-inverse A*) *∗v proj2-line-rep l*)
proof −
have *invertible* (*cltn2-rep* (*cltn2-inverse A*)) **by** (*rule cltn2-rep-invertible*)
hence *invertible* (*transpose* (*cltn2-rep* (*cltn2-inverse A*)))

by (*rule transpose-invertible*)
hence
apply-cltn2 (*L2P l*) (*cltn2-transpose* (*cltn2-inverse A*))
= *proj2-abs* (*proj2-rep* (*L2P l*) *v*∗ *transpose* (*cltn2-rep* (*cltn2-inverse A*)))
unfolding *cltn2-transpose-def*
by (*rule apply-cltn2-right-abs*)
hence *apply-cltn2* (*L2P l*) (*cltn2-transpose* (*cltn2-inverse A*))
= *proj2-abs* (*cltn2-rep* (*cltn2-inverse A*) ∗*v proj2-line-rep l*)
unfolding *proj2-line-rep-def*
by *simp*
thus *apply-cltn2-line l A*
= *proj2-line-abs* (*cltn2-rep* (*cltn2-inverse A*) ∗*v proj2-line-rep l*)
unfolding *apply-cltn2-line-def* **and** *proj2-line-abs-def* **..**
qed

lemma *rep-mult-line-rep-non-zero*: *cltn2-rep A* ∗*v proj2-line-rep l* ≠ *0*
using *proj2-line-rep-non-zero* **and** *cltn2-rep-invertible*
and *invertible-times-eq-zero*
by *auto*

lemma *apply-cltn2-incident*:
proj2-incident p (*apply-cltn2-line l A*)
⟷ *proj2-incident* (*apply-cltn2 p* (*cltn2-inverse A*)) *l*
proof −
have *proj2-rep p v*∗ *cltn2-rep* (*cltn2-inverse A*) ≠ *0*
by (*rule rep-mult-rep-non-zero*)
with *proj2-rep-abs2*
obtain *j* **where** *j* ≠ *0*
and *proj2-rep* (*proj2-abs* (*proj2-rep p v*∗ *cltn2-rep* (*cltn2-inverse A*)))
= *j* $*_R$ (*proj2-rep p v*∗ *cltn2-rep* (*cltn2-inverse A*))
by *auto*

let *?v* = *cltn2-rep* (*cltn2-inverse A*) ∗*v proj2-line-rep l*
have *?v* ≠ *0* **by** (*rule rep-mult-line-rep-non-zero*)
with *proj2-line-rep-abs* [*of ?v*]
obtain *k* **where** *k* ≠ *0*
and *proj2-line-rep* (*proj2-line-abs ?v*) = *k* $*_R$ *?v*
by *auto*
hence *proj2-incident p* (*apply-cltn2-line l A*)
⟷ *proj2-rep p* · (*cltn2-rep* (*cltn2-inverse A*) ∗*v proj2-line-rep l*) = *0*
unfolding *proj2-incident-def* **and** *apply-cltn2-line-alt-def*
by (*simp add*: *dot-scaleR-mult*)
also from *dot-lmul-matrix* [*of proj2-rep p cltn2-rep* (*cltn2-inverse A*)]
have
... ⟷ (*proj2-rep p v*∗ *cltn2-rep* (*cltn2-inverse A*)) · *proj2-line-rep l* = *0*
by *simp*
also from ⟨*j* ≠ *0*⟩
and ⟨*proj2-rep* (*proj2-abs* (*proj2-rep p v*∗ *cltn2-rep* (*cltn2-inverse A*)))
= *j* $*_R$ (*proj2-rep p v*∗ *cltn2-rep* (*cltn2-inverse A*))⟩

have ... $\longleftrightarrow$ *proj2-incident* (*apply-cltn2 p* (*cltn2-inverse A*)) *l*
unfolding *proj2-incident-def* **and** *apply-cltn2-def*
by (*simp add*: *dot-scaleR-mult*)
finally show *?thesis* **.**
qed

lemma *apply-cltn2-preserve-incident* [*iff*]:
proj2-incident (*apply-cltn2 p A*) (*apply-cltn2-line l A*)
$\longleftrightarrow$ *proj2-incident p l*
by (*simp add*: *apply-cltn2-incident*)

lemma *apply-cltn2-preserve-set-Col*:
assumes *proj2-set-Col S*
shows *proj2-set-Col* {*apply-cltn2 p C* | *p*. *p* ∈ *S*}
proof −
from ‹*proj2-set-Col S*›
obtain *l* **where** ∀ *p*∈*S*. *proj2-incident p l* **unfolding** *proj2-set-Col-def* **..**
hence ∀ *q* ∈ {*apply-cltn2 p C* | *p*. *p* ∈ *S*}.
proj2-incident q (*apply-cltn2-line l C*)
by *auto*
thus *proj2-set-Col* {*apply-cltn2 p C* | *p*. *p* ∈ *S*}
unfolding *proj2-set-Col-def* **..**
qed

lemma *apply-cltn2-injective*:
assumes *apply-cltn2 p C* = *apply-cltn2 q C*
shows *p* = *q*
proof −
from ‹*apply-cltn2 p C* = *apply-cltn2 q C*›
have *apply-cltn2* (*apply-cltn2 p C*) (*cltn2-inverse C*)
= *apply-cltn2* (*apply-cltn2 q C*) (*cltn2-inverse C*)
by *simp*
thus *p* = *q* **by** *simp*
qed

lemma *apply-cltn2-line-injective*:
assumes *apply-cltn2-line l C* = *apply-cltn2-line m C*
shows *l* = *m*
proof −
from ‹*apply-cltn2-line l C* = *apply-cltn2-line m C*›
have *apply-cltn2-line* (*apply-cltn2-line l C*) (*cltn2-inverse C*)
= *apply-cltn2-line* (*apply-cltn2-line m C*) (*cltn2-inverse C*)
by *simp*
thus *l* = *m* **by** *simp*
qed

lemma *apply-cltn2-line-unique*:
assumes *p* ≠ *q* **and** *proj2-incident p l* **and** *proj2-incident q l*
and *proj2-incident* (*apply-cltn2 p C*) *m*

and *proj2-incident* (*apply-cltn2 q C*) *m*
shows *apply-cltn2-line l C* = *m*
proof −
from ‹*proj2-incident p l*›
have *proj2-incident* (*apply-cltn2 p C*) (*apply-cltn2-line l C*) **by** *simp*

from ‹*proj2-incident q l*›
have *proj2-incident* (*apply-cltn2 q C*) (*apply-cltn2-line l C*) **by** *simp*

from ‹*p* ≠ *q*› **and** *apply-cltn2-injective* [*of p C q*]
have *apply-cltn2 p C* ≠ *apply-cltn2 q C* **by** *auto*
with ‹*proj2-incident* (*apply-cltn2 p C*) (*apply-cltn2-line l C*)›
and ‹*proj2-incident* (*apply-cltn2 q C*) (*apply-cltn2-line l C*)›
and ‹*proj2-incident* (*apply-cltn2 p C*) *m*›
and ‹*proj2-incident* (*apply-cltn2 q C*) *m*›
and *proj2-incident-unique*
show *apply-cltn2-line l C* = *m* **by** *fast*
qed

lemma *apply-cltn2-unique*:
assumes *l* ≠ *m* **and** *proj2-incident p l* **and** *proj2-incident p m*
and *proj2-incident q* (*apply-cltn2-line l C*)
and *proj2-incident q* (*apply-cltn2-line m C*)
shows *apply-cltn2 p C* = *q*
proof −
from ‹*proj2-incident p l*›
have *proj2-incident* (*apply-cltn2 p C*) (*apply-cltn2-line l C*) **by** *simp*

from ‹*proj2-incident p m*›
have *proj2-incident* (*apply-cltn2 p C*) (*apply-cltn2-line m C*) **by** *simp*

from ‹*l* ≠ *m*› **and** *apply-cltn2-line-injective* [*of l C m*]
have *apply-cltn2-line l C* ≠ *apply-cltn2-line m C* **by** *auto*
with ‹*proj2-incident* (*apply-cltn2 p C*) (*apply-cltn2-line l C*)›
and ‹*proj2-incident* (*apply-cltn2 p C*) (*apply-cltn2-line m C*)›
and ‹*proj2-incident q* (*apply-cltn2-line l C*)›
and ‹*proj2-incident q* (*apply-cltn2-line m C*)›
and *proj2-incident-unique*
show *apply-cltn2 p C* = *q* **by** *fast*
qed

7.4.3 Parts of some Statements from [1]

All theorems with names beginning with *statement* are based on corresponding theorems in [1].

lemma *statement52-existence*:
fixes *a* :: *proj2^3* **and** *a3* :: *proj2*
assumes *proj2-no-3-Col* (*insert a3* (*range* (($) *a*)))
shows ∃ *A*. *apply-cltn2* (*proj2-abs* (*vector* [*1*,*1*,*1*])) *A* = *a3* ∧

(∀ *j*. *apply-cltn2* (*proj2-abs* (*axis j 1*)) *A* = *a*$*j*)
proof −
let *?v* = *proj2-rep a3*
let *?B* = *proj2-rep* ‘ *range* (($) *a*)

from ‹*proj2-no-3-Col* (*insert a3* (*range* (($) *a*)))›
have *card* (*insert a3* (*range* (($) *a*))) = *4* **unfolding** *proj2-no-3-Col-def* **..**

from *card-image-le* [*of UNIV* ($) *a*]
have *card* (*range* (($) *a*)) ≤ *3* **by** *simp*
with *card-insert-if* [*of range* (($) *a*) *a3*]
and ‹*card* (*insert a3* (*range* (($) *a*))) = *4*›
have *a3* ∉ *range* (($) *a*) **by** *auto*
hence (*insert a3* (*range* (($) *a*))) − {*a3*} = *range* (($) *a*) **by** *simp*
with ‹*proj2-no-3-Col* (*insert a3* (*range* (($) *a*)))›
and *proj2-no-3-Col-span* [*of insert a3* (*range* (($) *a*)) *a3*]
have *span ?B* = *UNIV* **by** *simp*

from *card-suc-ge-insert* [*of a3 range* (($) *a*)]
and ‹*card* (*insert a3* (*range* (($) *a*))) = *4*›
and ‹*card* (*range* (($) *a*)) ≤ *3*›
have *card* (*range* (($) *a*)) = *3* **by** *simp*
with *card-image* [*of proj2-rep range* (($) *a*)]
and *proj2-rep-inj*
and *subset-inj-on*
have *card ?B* = *3* **by** *auto*
hence *finite ?B* **by** *simp*
with ‹*span ?B* = *UNIV*› **and** *span-finite* [*of ?B*]
obtain *c* **where** ($\sum$ *w* ∈ *?B*. (*c w*) $*_R$ *w*) = *?v*
by (*auto simp add*: *scalar-equiv*) (*metis* (*no-types*, *lifting*) *UNIV-I rangeE*)
let *?C* = χ *i*. *c* (*proj2-rep* (*a*$*i*)) $*_R$ (*proj2-rep* (*a*$*i*))
let *?A* = *cltn2-abs ?C*

from *proj2-rep-inj* **and** ‹*a3* ∉ *range* (($) *a*)› **have** *?v* ∉ *?B*
unfolding *inj-on-def*
by *auto*

have ∀ *i*. *c* (*proj2-rep* (*a*$*i*)) ≠ *0*
proof
fix *i*
let *?Bi* = *proj2-rep* ‘ (*range* (($) *a*) − {*a*$*i*})

have *a*$*i* ∈ *insert a3* (*range* (($) *a*)) **by** *simp*

have *proj2-rep* (*a*$*i*) ∈ *?B* **by** *auto*

from *image-set-diff* [*of proj2-rep*] **and** *proj2-rep-inj*
have *?Bi* = *?B* − {*proj2-rep* (*a*$*i*)} **by** *simp*
with *sum-diff1* [*of ?B* λ *w*. (*c w*) $*_R$ *w*]

  and ‹*finite ?B*›
  and ‹*proj2-rep* (*a*$*i*) ∈ *?B*›
have (∑ *w* ∈ *?Bi*. (*c w*) $*_R$ *w*) =
  (∑ *w* ∈ *?B*. (*c w*) $*_R$ *w*) − *c* (*proj2-rep* (*a*$*i*)) $*_R$ *proj2-rep* (*a*$*i*)
  by *simp*

from ‹*a3* ∉ *range* (($) *a*)› **have** *a3* ≠ *a*$*i* **by** *auto*
hence *insert a3* (*range* (($) *a*)) − {*a*$*i*} =
  insert a3 (*range* (($) *a*) − {*a*$*i*}) **by** *auto*
hence *proj2-rep* ‘ (*insert a3* (*range* (($) *a*)) − {*a*$*i*}) = *insert ?v ?Bi*
  by *simp*
moreover from ‹*proj2-no-3-Col* (*insert a3* (*range* (($) *a*)))›
  and ‹*a*$*i* ∈ *insert a3* (*range* (($) *a*))›
have *span* (*proj2-rep* ‘ (*insert a3* (*range* (($) *a*)) − {*a*$*i*})) = *UNIV*
  by (*rule proj2-no-3-Col-span*)
ultimately have *span* (*insert ?v ?Bi*) = *UNIV* **by** *simp*

from ‹*?Bi* = *?B* − {*proj2-rep* (*a*$*i*)}›
  and ‹*proj2-rep* (*a*$*i*) ∈ *?B*›
  and ‹*card ?B* = *3*›
have *card ?Bi* = *2* **by** (*simp add*: *card-gt-0-diff-singleton*)
hence *finite ?Bi* **by** *simp*
with ‹*card ?Bi* = *2*› **and** *dim-le-card′* [*of ?Bi*] **have** *dim ?Bi* ≤ *2* **by** *simp*
hence *dim* (*span ?Bi*) ≤ *2* **by** (*subst dim-span*)
then have *span ?Bi* ≠ *UNIV*
  by *clarify* (*auto simp*: *dim-UNIV*)
with ‹*span* (*insert ?v ?Bi*) = *UNIV*› **and** *span-redundant*
have *?v* ∉ *span ?Bi* **by** *auto*

{ assume *c* (*proj2-rep* (*a*$*i*)) = *0*
  with ‹(∑ *w* ∈ *?Bi*. (*c w*) $*_R$ *w*) =
    (∑ *w* ∈ *?B*. (*c w*) $*_R$ *w*) − *c* (*proj2-rep* (*a*$*i*)) $*_R$ *proj2-rep* (*a*$*i*)›
    and ‹(∑ *w* ∈ *?B*. (*c w*) $*_R$ *w*) = *?v*›
  have *?v* = (∑ *w* ∈ *?Bi*. (*c w*) $*_R$ *w*)
    by *simp*
  with *span-finite* [*of ?Bi*] **and** ‹*finite ?Bi*›
  have *?v* ∈ *span ?Bi* **by** (*simp add*: *scalar-equiv*)
  with ‹*?v* ∉ *span ?Bi*› **have** *False* **.. }**
thus *c* (*proj2-rep* (*a*$*i*)) ≠ *0* **..**
qed
hence ∀ *w*∈*?B*. *c w* ≠ *0*
  unfolding *image-def*
  by *auto*

have *rows ?C* = (λ *w*. (*c w*) $*_R$ *w*) ‘ *?B*
  unfolding *rows-def*
    and *row-def*
    and *image-def*
  by (*auto simp*: *vec-lambda-eta*)

have $\forall\ x.\ x \in span\ (rows\ ?C)$
proof
  fix $x :: real\hat{}3$
  from ‹*finite ?B*› **and** *span-finite* [*of ?B*] **and** ‹*span ?B = UNIV*›
  obtain *ub* **where** $(\sum\ w{\in}?B.\ (ub\ w) *_R w) = x$
    by (*auto simp add*: *scalar-equiv*) (*metis* (*no-types*, *lifting*) *UNIV-I rangeE*)
  have $\forall\ w{\in}?B.\ (ub\ w) *_R w \in span\ (rows\ ?C)$
  proof
    fix *w*
    assume $w \in ?B$
    with *span-superset* [*of rows ?C*] **and** ‹$rows\ ?C = image\ (\lambda\ w.\ (c\ w) *_R w)$
?B›
    have $(c\ w) *_R w \in span\ (rows\ ?C)$ **by** *auto*
    with *span-mul* [*of* $(c\ w) *_R w$ *rows ?C* $(ub\ w)/(c\ w)$]
    have $((ub\ w)/(c\ w)) *_R ((c\ w) *_R w) \in span\ (rows\ ?C)$
      by (*simp add*: *scalar-equiv*)
    with ‹$\forall\ w{\in}?B.\ c\ w \neq 0$› **and** ‹$w \in ?B$›
    show $(ub\ w) *_R w \in span\ (rows\ ?C)$ **by** *auto*
  qed
  with *span-sum* [*of ?B* $\lambda\ w.\ (ub\ w) *_R w$] **and** ‹*finite ?B*›
  have $(\sum\ w{\in}?B.\ (ub\ w) *_R w) \in span\ (rows\ ?C)$ **by** *blast*
  with ‹$(\sum\ w{\in}?B.\ (ub\ w) *_R w) = x$› **show** $x \in span\ (rows\ ?C)$ **by** *simp*
qed
hence $span\ (rows\ ?C) = UNIV$ **by** *auto*
with *matrix-left-invertible-span-rows* [*of ?C*]
have $\exists\ C'.\ C' ** ?C = mat\ 1$ **..**
with *left-invertible-iff-invertible*
have *invertible ?C* **..**

have $(vector\ [1,1,1] :: real\hat{}3) \neq 0$
  unfolding *vector-def*
  by (*simp add*: *vec-eq-iff forall-3*)
with *apply-cltn2-abs* **and** ‹*invertible ?C*›
have *apply-cltn2* (*proj2-abs* (*vector* [*1*,*1*,*1*])) *?A* =
  proj2-abs (*vector* [*1*,*1*,*1*] $v*$ *?C*)
  by *simp*
from *inj-on-iff-eq-card* [*of UNIV* (\$) *a*] **and** ‹*card* (*range* ((\$) *a*)) = *3*›
have *inj* ((\$) *a*) **by** *simp*
from *exhaust-3* **have** $\forall\ i{::}3.$ (*vector* [*1::real*,*1*,*1*])\$*i* = *1*
  unfolding *vector-def*
  by *auto*
with *vector-matrix-row* [*of vector* [*1*,*1*,*1*] *?C*]
have (*vector* [*1*,*1*,*1*]) $v*$ *?C* =
  $(\sum\ i{\in}UNIV.\ (c\ (proj2\text{-}rep\ (a\$i))) *_R (proj2\text{-}rep\ (a\$i)))$
  by *simp*
also from *sum.reindex*
[*of* (\$) *a UNIV* $\lambda\ x.\ (c\ (proj2\text{-}rep\ x)) *_R (proj2\text{-}rep\ x)$]
  and ‹*inj* ((\$) *a*)›

have . . . = (∑ *x*∈(*range* (($) *a*)). (*c* (*proj2-rep x*)) ∗*R* (*proj2-rep x*))
by *simp*
also from *sum.reindex*
[*of proj2-rep range* (($) *a*) λ *w*. (*c w*) ∗*R* *w*]
and *proj2-rep-inj* **and** *subset-inj-on* [*of proj2-rep UNIV range* (($) *a*)]
have . . . = (∑ *w*∈*?B*. (*c w*) ∗*R* *w*) **by** *simp*
also from ‹(∑ *w* ∈ *?B*. (*c w*) ∗*R* *w*) = *?v*› **have** . . . = *?v* **by** *simp*
finally have (*vector* [*1*,*1*,*1*]) *v*∗ *?C* = *?v* **.**
with ‹*apply-cltn2* (*proj2-abs* (*vector* [*1*,*1*,*1*])) *?A* =
proj2-abs (*vector* [*1*,*1*,*1*] *v*∗ *?C*)›
have *apply-cltn2* (*proj2-abs* (*vector* [*1*,*1*,*1*])) *?A* = *proj2-abs ?v* **by** *simp*
with *proj2-abs-rep* **have** *apply-cltn2* (*proj2-abs* (*vector* [*1*,*1*,*1*])) *?A* = *a3*
by *simp*
have ∀ *j*. *apply-cltn2* (*proj2-abs* (*axis j 1*)) *?A* = *a*$*j*
proof
fix *j* :: *3*
have ((*axis j 1*)::*real*ˆ*3*) ≠ *0* **by** (*simp add*: *vec-eq-iff axis-def*)
with *apply-cltn2-abs* **and** ‹*invertible ?C*›
have *apply-cltn2* (*proj2-abs* (*axis j 1*)) *?A* = *proj2-abs* (*axis j 1 v*∗ *?C*)
by *simp*

have ∀ *i*∈(*UNIV*−{*j*}).
((*axis j 1*)$*i* ∗ *c* (*proj2-rep* (*a*$*i*))) ∗*R* (*proj2-rep* (*a*$*i*)) = *0*
by (*simp add*: *axis-def*)
with *sum.mono-neutral-left* [*of UNIV* {*j*}
λ *i*. ((*axis j 1*)$*i* ∗ *c* (*proj2-rep* (*a*$*i*))) ∗*R* (*proj2-rep* (*a*$*i*))]
and *vector-matrix-row* [*of axis j 1 ?C*]
have (*axis j 1*) *v*∗ *?C* = *?C*$*j* **by** (*simp add*: *scalar-equiv*)
hence (*axis j 1*) *v*∗ *?C* = *c* (*proj2-rep* (*a*$*j*)) ∗*R* (*proj2-rep* (*a*$*j*)) **by** *simp*
with *proj2-abs-mult-rep* **and** ‹∀ *i*. *c* (*proj2-rep* (*a*$*i*)) ≠ *0*›
and ‹*apply-cltn2* (*proj2-abs* (*axis j 1*)) *?A* = *proj2-abs* (*axis j 1 v*∗ *?C*)›
show *apply-cltn2* (*proj2-abs* (*axis j 1*)) *?A* = *a*$*j*
by *simp*
qed
with ‹*apply-cltn2* (*proj2-abs* (*vector* [*1*,*1*,*1*])) *?A* = *a3*›
show ∃ *A*. *apply-cltn2* (*proj2-abs* (*vector* [*1*,*1*,*1*])) *A* = *a3* ∧
(∀ *j*. *apply-cltn2* (*proj2-abs* (*axis j 1*)) *A* = *a*$*j*)
by *auto*
qed

lemma *statement53-existence*:
fixes *p* :: *proj2*ˆ*4*ˆ*2*
assumes ∀ *i*. *proj2-no-3-Col* (*range* (($) (*p*$*i*)))
shows ∃ *C*. ∀ *j*. *apply-cltn2* (*p*$*0*$*j*) *C* = *p*$*1*$*j*
proof −
let *?q* = χ *i*. χ *j*::*3*. *p*$*i* $ (*of-int* (*Rep-bit1 j*))
let *?D* = χ *i*. ϵ *D*. *apply-cltn2* (*proj2-abs* (*vector* [*1*,*1*,*1*])) *D* = *p*$*i*$*3*
∧ (∀ *j*′. *apply-cltn2* (*proj2-abs* (*axis j*′ *1*)) *D* = *?q*$*i*$*j*′)
have ∀ *i*. *apply-cltn2* (*proj2-abs* (*vector* [*1*,*1*,*1*])) (*?D*$*i*) = *p*$*i*$*3*

```
    ∧ (∀ j'. apply-cltn2 (proj2-abs (axis j' 1)) (?D$i) = ?q$i$j')
  proof
    fix i
    have range (($) (p$i)) = insert (p$i$3) (range (($) (?q$i)))
    proof
      show range (($) (p$i)) ⊇ insert (p$i$3) (range (($) (?q$i))) by auto
      show range (($) (p$i)) ⊆ insert (p$i$3) (range (($) (?q$i)))
      proof
        fix r
        assume r ∈ range (($) (p$i))
        then obtain j where r = p$i$j by auto
        with eq-3-or-of-3 [of j]
        show r ∈ insert (p$i$3) (range (($) (?q$i))) by auto
      qed
    qed
    moreover from ‹∀ i. proj2-no-3-Col (range (($) (p$i)))›
    have proj2-no-3-Col (range (($) (p$i))) ..
    ultimately have proj2-no-3-Col (insert (p$i$3) (range (($) (?q$i))))
      by simp
    hence ∃ D. apply-cltn2 (proj2-abs (vector [1,1,1])) D = p$i$3
      ∧ (∀ j'. apply-cltn2 (proj2-abs (axis j' 1)) D = ?q$i$j')
      by (rule statement52-existence)
    with someI-ex [of λ D. apply-cltn2 (proj2-abs (vector [1,1,1])) D = p$i$3
      ∧ (∀ j'. apply-cltn2 (proj2-abs (axis j' 1)) D = ?q$i$j')]
    show apply-cltn2 (proj2-abs (vector [1,1,1])) (?D$i) = p$i$3
      ∧ (∀ j'. apply-cltn2 (proj2-abs (axis j' 1)) (?D$i) = ?q$i$j')
      by simp
  qed
  hence apply-cltn2 (proj2-abs (vector [1,1,1])) (?D$0) = p$0$3
    and apply-cltn2 (proj2-abs (vector [1,1,1])) (?D$1) = p$1$3
    and ∀ j'. apply-cltn2 (proj2-abs (axis j' 1)) (?D$0) = ?q$0$j'
    and ∀ j'. apply-cltn2 (proj2-abs (axis j' 1)) (?D$1) = ?q$1$j'
    by simp-all

  let ?C = cltn2-compose (cltn2-inverse (?D$0)) (?D$1)
  have ∀ j. apply-cltn2 (p$0$j) ?C = p$1$j
  proof
    fix j
    show apply-cltn2 (p$0$j) ?C = p$1$j
    proof cases
      assume j = 3
      with ‹apply-cltn2 (proj2-abs (vector [1,1,1])) (?D$0) = p$0$3›
        and  cltn2.act-inv-iff
      have
        apply-cltn2 (p$0$j) (cltn2-inverse (?D$0)) = proj2-abs (vector [1,1,1])
        by simp
      with ‹apply-cltn2 (proj2-abs (vector [1,1,1])) (?D$1) = p$1$3›
        and ‹j = 3›
        and cltn2.act-act [of cltn2-inverse (?D$0) ?D$1 p$0$j]
```

show *apply-cltn2* (*p*\$*0*\$*j*) *?C* = *p*\$*1*\$*j* **by** *simp*
next
assume *j* ≠ *3*
with *eq-3-or-of-3* **obtain** *j′* :: *3* **where** *j* = *of-int* (*Rep-bit1 j′*)
by *metis*
with ⟨∀ *j′*. *apply-cltn2* (*proj2-abs* (*axis j′ 1*)) (*?D*\$*0*) = *?q*\$*0*\$*j′*⟩
and ⟨∀ *j′*. *apply-cltn2* (*proj2-abs* (*axis j′ 1*)) (*?D*\$*1*) = *?q*\$*1*\$*j′*⟩
have *p*\$*0*\$*j* = *apply-cltn2* (*proj2-abs* (*axis j′ 1*)) (*?D*\$*0*)
and *p*\$*1*\$*j* = *apply-cltn2* (*proj2-abs* (*axis j′ 1*)) (*?D*\$*1*)
by *simp-all*
from ⟨*p*\$*0*\$*j* = *apply-cltn2* (*proj2-abs* (*axis j′ 1*)) (*?D*\$*0*)⟩
and *cltn2.act-inv-iff*
have *apply-cltn2* (*p*\$*0*\$*j*) (*cltn2-inverse* (*?D*\$*0*)) = *proj2-abs* (*axis j′ 1*)
by *simp*
with ⟨*p*\$*1*\$*j* = *apply-cltn2* (*proj2-abs* (*axis j′ 1*)) (*?D*\$*1*)⟩
and *cltn2.act-act* [*of cltn2-inverse* (*?D*\$*0*) *?D*\$*1 p*\$*0*\$*j*]
show *apply-cltn2* (*p*\$*0*\$*j*) *?C* = *p*\$*1*\$*j* **by** *simp*
qed
qed
thus ∃ *C*. ∀ *j*. *apply-cltn2* (*p*\$*0*\$*j*) *C* = *p*\$*1*\$*j* **by** (*rule exI* [*of* - *?C*])
qed

lemma *apply-cltn2-linear*:
assumes *j* $*_R$ *v* + *k* $*_R$ *w* ≠ *0*
shows *j* $*_R$ (*v v*∗ *cltn2-rep C*) + *k* $*_R$ (*w v*∗ *cltn2-rep C*) ≠ *0*
(**is** *?u* ≠ *0*)
and *apply-cltn2* (*proj2-abs* (*j* $*_R$ *v* + *k* $*_R$ *w*)) *C*
= *proj2-abs* (*j* $*_R$ (*v v*∗ *cltn2-rep C*) + *k* $*_R$ (*w v*∗ *cltn2-rep C*))
proof −
have *?u* = (*j* $*_R$ *v* + *k* $*_R$ *w*) *v*∗ *cltn2-rep C*
by (*simp only*: *vector-matrix-left-distrib scaleR-vector-matrix-assoc*)
with ⟨*j* $*_R$ *v* + *k* $*_R$ *w* ≠ *0*⟩ **and** *non-zero-mult-rep-non-zero*
show *?u* ≠ *0* **by** *simp*

from ⟨*?u* = (*j* $*_R$ *v* + *k* $*_R$ *w*) *v*∗ *cltn2-rep C*⟩
and ⟨*j* $*_R$ *v* + *k* $*_R$ *w* ≠ *0*⟩
and *apply-cltn2-left-abs*
show *apply-cltn2* (*proj2-abs* (*j* $*_R$ *v* + *k* $*_R$ *w*)) *C* = *proj2-abs ?u*
by *simp*
qed

lemma *apply-cltn2-imp-mult*:
assumes *apply-cltn2 p C* = *q*
shows ∃ *k*. *k* ≠ *0* ∧ *proj2-rep p v*∗ *cltn2-rep C* = *k* $*_R$ *proj2-rep q*
proof −
have *proj2-rep p v*∗ *cltn2-rep C* ≠ *0* **by** (*rule rep-mult-rep-non-zero*)

from ⟨*apply-cltn2 p C* = *q*⟩
have *proj2-abs* (*proj2-rep p v*∗ *cltn2-rep C*) = *q* **by** (*unfold apply-cltn2-def*)

hence *proj2-rep* (*proj2-abs* (*proj2-rep p v∗ cltn2-rep C*)) = *proj2-rep q*
  by *simp*
 with ‹*proj2-rep p v∗ cltn2-rep C ≠ 0*› **and** *proj2-rep-abs2* [*of proj2-rep p v∗ cltn2-rep C*]
have ∃ *j*. *j ≠ 0* ∧ *proj2-rep q* = *j* $*_R$ (*proj2-rep p v∗ cltn2-rep C*) **by** *simp*
then obtain *j* **where** *j ≠ 0*
  and *proj2-rep q* = *j* $*_R$ (*proj2-rep p v∗ cltn2-rep C*) **by** *auto*
hence *proj2-rep p v∗ cltn2-rep C* = (*1*/*j*) $*_R$ *proj2-rep q*
  by (*simp add*: *field-simps*)
with ‹*j ≠ 0*›
show ∃ *k*. *k ≠ 0* ∧ *proj2-rep p v∗ cltn2-rep C* = *k* $*_R$ *proj2-rep q*
  by (*simp add*: *exI* [*of - 1*/*j*])
qed

lemma *statement55*:
  assumes *p ≠ q*
  and *apply-cltn2 p C* = *q*
  and *apply-cltn2 q C* = *p*
  and *proj2-incident p l*
  and *proj2-incident q l*
  and *proj2-incident r l*
  shows *apply-cltn2* (*apply-cltn2 r C*) *C* = *r*
proof *cases*
  assume *r* = *p*
  with ‹*apply-cltn2 p C* = *q*› **and** ‹*apply-cltn2 q C* = *p*›
  show *apply-cltn2* (*apply-cltn2 r C*) *C* = *r* **by** *simp*
next
  assume *r ≠ p*

  from ‹*apply-cltn2 p C* = *q*› **and** *apply-cltn2-imp-mult* [*of p C q*]
  obtain *i* **where** *i ≠ 0* **and** *proj2-rep p v∗ cltn2-rep C* = *i* $*_R$ *proj2-rep q*
    by *auto*

  from ‹*apply-cltn2 q C* = *p*› **and** *apply-cltn2-imp-mult* [*of q C p*]
  obtain *j* **where** *j ≠ 0* **and** *proj2-rep q v∗ cltn2-rep C* = *j* $*_R$ *proj2-rep p*
    by *auto*

  from ‹*p ≠ q*›
    and ‹*proj2-incident p l*›
    and ‹*proj2-incident q l*›
    and ‹*proj2-incident r l*›
    and *proj2-incident-iff*
  have *r* = *p* ∨ (∃ *k*. *r* = *proj2-abs* (*k* $*_R$ *proj2-rep p* + *proj2-rep q*))
    by *fast*
  with ‹*r ≠ p*›
  obtain *k* **where** *r* = *proj2-abs* (*k* $*_R$ *proj2-rep p* + *proj2-rep q*) **by** *auto*

  from ‹*p ≠ q*› **and** *proj2-rep-dependent* [*of k p 1 q*]
  have *k* $*_R$ *proj2-rep p* + *proj2-rep q ≠ 0* **by** *auto*

with ‹r = *proj2-abs* ($k *_R$ *proj2-rep* p + *proj2-rep* q)›
and *apply-cltn2-linear* [*of* k *proj2-rep* p *1* *proj2-rep* q]
have $k *_R$ (*proj2-rep* p $v*$ *cltn2-rep* C) + *proj2-rep* q $v*$ *cltn2-rep* $C \neq 0$
and *apply-cltn2* r C
= *proj2-abs*
($k *_R$ (*proj2-rep* p $v*$ *cltn2-rep* C) + *proj2-rep* q $v*$ *cltn2-rep* C)
by *simp-all*
with ‹*proj2-rep* p $v*$ *cltn2-rep* C = $i *_R$ *proj2-rep* q›
and ‹*proj2-rep* q $v*$ *cltn2-rep* C = $j *_R$ *proj2-rep* p›
have ($k * i$) $*_R$ *proj2-rep* q + $j *_R$ *proj2-rep* $p \neq 0$
and *apply-cltn2* r C
= *proj2-abs* (($k * i$) $*_R$ *proj2-rep* q + $j *_R$ *proj2-rep* p)
by *simp-all*
with *apply-cltn2-linear*
have *apply-cltn2* (*apply-cltn2* r C) C
= *proj2-abs*
(($k * i$) $*_R$ (*proj2-rep* q $v*$ *cltn2-rep* C)
+ $j *_R$ (*proj2-rep* p $v*$ *cltn2-rep* C))
by *simp*
with ‹*proj2-rep* p $v*$ *cltn2-rep* C = $i *_R$ *proj2-rep* q›
and ‹*proj2-rep* q $v*$ *cltn2-rep* C = $j *_R$ *proj2-rep* p›
have *apply-cltn2* (*apply-cltn2* r C) C
= *proj2-abs* (($k * i * j$) $*_R$ *proj2-rep* p + ($j * i$) $*_R$ *proj2-rep* q)
by *simp*
also have . . . = *proj2-abs* (($i * j$) $*_R$ ($k *_R$ *proj2-rep* p + *proj2-rep* q))
by (*simp add*: *algebra-simps*)
also from ‹$i \neq 0$› **and** ‹$j \neq 0$› **and** *proj2-abs-mult*
have . . . = *proj2-abs* ($k *_R$ *proj2-rep* p + *proj2-rep* q) **by** *simp*
also from ‹r = *proj2-abs* ($k *_R$ *proj2-rep* p + *proj2-rep* q)›
have . . . = r **by** *simp*
finally show *apply-cltn2* (*apply-cltn2* r C) C = r **.**
qed

7.5 Cross ratios

definition *cross-ratio* :: *proj2* ⇒ *proj2* ⇒ *proj2* ⇒ *proj2* ⇒ *real* **where**
cross-ratio p q r s ≜ *proj2-Col-coeff* p q s / *proj2-Col-coeff* p q r

definition *cross-ratio-correct* :: *proj2* ⇒ *proj2* ⇒ *proj2* ⇒ *proj2* ⇒ *bool* **where**
cross-ratio-correct p q r s ≜
proj2-set-Col $\{p,q,r,s\} \wedge p \neq q \wedge r \neq p \wedge s \neq p \wedge r \neq q$

lemma *proj2-Col-coeff-abs*:
assumes $p \neq q$ **and** $j \neq 0$
shows *proj2-Col-coeff* p q (*proj2-abs* ($i *_R$ *proj2-rep* p + $j *_R$ *proj2-rep* q))
= i/j
(**is** *proj2-Col-coeff* p q ?r = i/j)
proof −
from ‹$j \neq 0$›

and *proj2-abs-mult* [*of* $1/j$ $i *_R$ *proj2-rep p* + $j *_R$ *proj2-rep q*]
have *?r* = *proj2-abs* ((i/j) $*_R$ *proj2-rep p* + *proj2-rep q*)
by (*simp add*: *scaleR-right-distrib*)

from ‹$p \neq q$› **and** *proj2-rep-dependent* [*of - p 1 q*]
have (i/j) $*_R$ *proj2-rep p* + *proj2-rep q* $\neq 0$ **by** *auto*
with ‹*?r* = *proj2-abs* ((i/j) $*_R$ *proj2-rep p* + *proj2-rep q*)›
and *proj2-rep-abs2*
obtain k **where** $k \neq 0$
and *proj2-rep ?r* = $k *_R$ ((i/j) $*_R$ *proj2-rep p* + *proj2-rep q*)
by *auto*
hence ($k*i/j$) $*_R$ *proj2-rep p* + $k *_R$ *proj2-rep q* − *proj2-rep ?r* = 0
by (*simp add*: *scaleR-right-distrib*)
hence $\exists$ l. ($k*i/j$) $*_R$ *proj2-rep p* + $k *_R$ *proj2-rep q* + $l *_R$ *proj2-rep ?r* = 0
$\wedge$ ($k*i/j \neq 0 \vee k \neq 0 \vee l \neq 0$)
by (*simp add*: *exI* [*of* - -1])
hence *proj2-Col p q ?r* **by** (*unfold proj2-Col-def*) *auto*

have *?r* $\neq$ *p*
proof
assume *?r* = *p*
with ‹($k*i/j$) $*_R$ *proj2-rep p* + $k *_R$ *proj2-rep q* − *proj2-rep ?r* = 0›
have ($k*i/j - 1$) $*_R$ *proj2-rep p* + $k *_R$ *proj2-rep q* = 0
by (*simp add*: *algebra-simps*)
with ‹$k \neq 0$› **and** *proj2-rep-dependent* **have** $p = q$ **by** *simp*
with ‹$p \neq q$› **show** *False* **..**
qed
with ‹*proj2-Col p q ?r*› **and** ‹$p \neq q$›
have *?r* = *proj2-abs* (*proj2-Col-coeff p q ?r* $*_R$ *proj2-rep p* + *proj2-rep q*)
by (*rule proj2-Col-coeff*)
with ‹$p \neq q$› **and** ‹*?r* = *proj2-abs* ((i/j) $*_R$ *proj2-rep p* + *proj2-rep q*)›
and *proj2-Col-coeff-unique*
show *proj2-Col-coeff p q ?r* = i/j **by** *simp*
qed

lemma *proj2-set-Col-coeff*:
assumes *proj2-set-Col S* **and** $\{p,q,r\} \subseteq S$ **and** $p \neq q$ **and** $r \neq p$
shows r = *proj2-abs* (*proj2-Col-coeff p q r* $*_R$ *proj2-rep p* + *proj2-rep q*)
(**is** r = *proj2-abs* (*?i* $*_R$ *?u* + *?v*))
proof −
from ‹$\{p,q,r\} \subseteq S$› **and** ‹*proj2-set-Col S*›
have *proj2-set-Col* $\{p,q,r\}$ **by** (*rule proj2-subset-Col*)
hence *proj2-Col p q r* **by** (*subst proj2-Col-iff-set-Col*)
with ‹$p \neq q$› **and** ‹$r \neq p$› **and** *proj2-Col-coeff*
show r = *proj2-abs* (*?i* $*_R$ *?u* + *?v*) **by** *simp*
qed

lemma *cross-ratio-abs*:
fixes u v :: *real^3* **and** i j k l :: *real*

assumes $u \neq 0$ **and** $v \neq 0$ **and** *proj2-abs* $u \neq$ *proj2-abs* v
and $j \neq 0$ **and** $l \neq 0$
shows *cross-ratio* (*proj2-abs* u) (*proj2-abs* v)
(*proj2-abs* $(i *_R u + j *_R v)$)
(*proj2-abs* $(k *_R u + l *_R v)$)
$= j * k / (i * l)$
(**is** *cross-ratio ?p ?q ?r ?s* = -)
proof −
from ‹$u \neq 0$› **and** *proj2-rep-abs2*
obtain g **where** $g \neq 0$ **and** *proj2-rep ?p* $= g *_R u$ **by** *auto*

from ‹$v \neq 0$› **and** *proj2-rep-abs2*
obtain h **where** $h \neq 0$ **and** *proj2-rep ?q* $= h *_R v$ **by** *auto*
with ‹$g \neq 0$› **and** ‹*proj2-rep ?p* $= g *_R u$›
have *?r* = *proj2-abs* $((i/g) *_R$ *proj2-rep ?p* $+ (j/h) *_R$ *proj2-rep ?q*)
and *?s* = *proj2-abs* $((k/g) *_R$ *proj2-rep ?p* $+ (l/h) *_R$ *proj2-rep ?q*)
by (*simp-all add*: *field-simps*)
with ‹*?p* ≠ *?q*› **and** ‹$h \neq 0$› **and** ‹$j \neq 0$› **and** ‹$l \neq 0$› **and** *proj2-Col-coeff-abs*
have *proj2-Col-coeff ?p ?q ?r* $= h*i/(g*j)$
and *proj2-Col-coeff ?p ?q ?s* $= h*k/(g*l)$
by *simp-all*
with ‹$g \neq 0$› **and** ‹$h \neq 0$›
show *cross-ratio ?p ?q ?r ?s* $= j*k/(i*l)$
by (*unfold cross-ratio-def*) (*simp add*: *field-simps*)
qed

lemma *cross-ratio-abs2*:
assumes $p \neq q$
shows *cross-ratio* p q
(*proj2-abs* $(i *_R$ *proj2-rep* p + *proj2-rep* q))
(*proj2-abs* $(j *_R$ *proj2-rep* p + *proj2-rep* q))
$= j/i$
(**is** *cross-ratio p q ?r ?s* = -)
proof −
let *?u* = *proj2-rep* p
let *?v* = *proj2-rep* q
have *?u* ≠ *0* **and** *?v* ≠ *0* **by** (*rule proj2-rep-non-zero*)+

have *proj2-abs ?u* = p **and** *proj2-abs ?v* = q **by** (*rule proj2-abs-rep*)+
with ‹*?u* ≠ *0*› **and** ‹*?v* ≠ *0*› **and** ‹$p \neq q$› **and** *cross-ratio-abs* [*of ?u ?v 1 1 i j*]
show *cross-ratio p q ?r ?s* $= j/i$ **by** *simp*
qed

lemma *cross-ratio-correct-cltn2*:
assumes *cross-ratio-correct p q r s*
shows *cross-ratio-correct* (*apply-cltn2 p C*) (*apply-cltn2 q C*)
(*apply-cltn2 r C*) (*apply-cltn2 s C*)
(**is** *cross-ratio-correct ?pC ?qC ?rC ?sC*)
proof −

from ‹*cross-ratio-correct p q r s*›
have *proj2-set-Col* {*p*,*q*,*r*,*s*}
and $p \neq q$ **and** $r \neq p$ **and** $s \neq p$ **and** $r \neq q$
by (*unfold cross-ratio-correct-def*) *simp-all*

have {*apply-cltn2 t C* | *t*. $t \in$ {*p*,*q*,*r*,*s*}} = {*?pC*,*?qC*,*?rC*,*?sC*} **by** *auto*
with ‹*proj2-set-Col* {*p*,*q*,*r*,*s*}›
and *apply-cltn2-preserve-set-Col* [*of* {*p*,*q*,*r*,*s*} *C*]
have *proj2-set-Col* {*?pC*,*?qC*,*?rC*,*?sC*} **by** *simp*

from ‹$p \neq q$› **and** ‹$r \neq p$› **and** ‹$s \neq p$› **and** ‹$r \neq q$› **and** *apply-cltn2-injective*
have *?pC* ≠ *?qC* **and** *?rC* ≠ *?pC* **and** *?sC* ≠ *?pC* **and** *?rC* ≠ *?qC* **by** *fast*+
with ‹*proj2-set-Col* {*?pC*,*?qC*,*?rC*,*?sC*}›
show *cross-ratio-correct ?pC ?qC ?rC ?sC*
by (*unfold cross-ratio-correct-def*) *simp*
qed

lemma *cross-ratio-cltn2*:
assumes *proj2-set-Col* {*p*,*q*,*r*,*s*} **and** $p \neq q$ **and** $r \neq p$ **and** $s \neq p$
shows *cross-ratio* (*apply-cltn2 p C*) (*apply-cltn2 q C*)
(*apply-cltn2 r C*) (*apply-cltn2 s C*)
= *cross-ratio p q r s*
(**is** *cross-ratio ?pC ?qC ?rC ?sC* = -)
proof −
let *?u* = *proj2-rep p*
let *?v* = *proj2-rep q*
let *?i* = *proj2-Col-coeff p q r*
let *?j* = *proj2-Col-coeff p q s*
from ‹*proj2-set-Col* {*p*,*q*,*r*,*s*}› **and** ‹$p \neq q$› **and** ‹$r \neq p$› **and** ‹$s \neq p$›
and *proj2-set-Col-coeff*
have *r* = *proj2-abs* (*?i* $*_R$ *?u* + *?v*) **and** *s* = *proj2-abs* (*?j* $*_R$ *?u* + *?v*)
by *simp-all*

let *?uC* = *?u v** *cltn2-rep C*
let *?vC* = *?v v** *cltn2-rep C*
have *?uC* ≠ *0* **and** *?vC* ≠ *0* **by** (*rule rep-mult-rep-non-zero*)+

have *proj2-abs ?uC* = *?pC* **and** *proj2-abs ?vC* = *?qC*
by (*unfold apply-cltn2-def*) *simp-all*

from ‹$p \neq q$› **and** *apply-cltn2-injective* **have** *?pC* ≠ *?qC* **by** *fast*

from ‹$p \neq q$› **and** *proj2-rep-dependent* [*of* - *p 1 q*]
have *?i* $*_R$ *?u* + *?v* ≠ *0* **and** *?j* $*_R$ *?u* + *?v* ≠ *0* **by** *auto*
with ‹*r* = *proj2-abs* (*?i* $*_R$ *?u* + *?v*)› **and** ‹*s* = *proj2-abs* (*?j* $*_R$ *?u* + *?v*)›
and *apply-cltn2-linear* [*of ?i ?u 1 ?v*]
and *apply-cltn2-linear* [*of ?j ?u 1 ?v*]
have *?rC* = *proj2-abs* (*?i* $*_R$ *?uC* + *?vC*)
and *?sC* = *proj2-abs* (*?j* $*_R$ *?uC* + *?vC*)

by *simp-all*
with ‹$?uC \neq 0$› **and** ‹$?vC \neq 0$› **and** ‹*proj2-abs* $?uC = ?pC$›
and ‹*proj2-abs* $?vC = ?qC$› **and** ‹$?pC \neq ?qC$›
and *cross-ratio-abs* [*of* $?uC$ $?vC$ 1 1 $?i$ $?j$]
have *cross-ratio* $?pC$ $?qC$ $?rC$ $?sC = ?j / ?i$ **by** *simp*
thus *cross-ratio* $?pC$ $?qC$ $?rC$ $?sC$ = *cross-ratio* p q r s
unfolding *cross-ratio-def* [*of* p q r s] **.**
qed

lemma *cross-ratio-unique*:
assumes *cross-ratio-correct* p q r s **and** *cross-ratio-correct* p q r t
and *cross-ratio* p q r s = *cross-ratio* p q r t
shows $s = t$
proof −
from ‹*cross-ratio-correct* p q r s› **and** ‹*cross-ratio-correct* p q r t›
have *proj2-set-Col* $\{p,q,r,s\}$ **and** *proj2-set-Col* $\{p,q,r,t\}$
and $p \neq q$ **and** $r \neq p$ **and** $r \neq q$ **and** $s \neq p$ **and** $t \neq p$
by (*unfold cross-ratio-correct-def*) *simp-all*

let $?u$ = *proj2-rep* p
let $?v$ = *proj2-rep* q
let $?i$ = *proj2-Col-coeff* p q r
let $?j$ = *proj2-Col-coeff* p q s
let $?k$ = *proj2-Col-coeff* p q t
from ‹*proj2-set-Col* $\{p,q,r,s\}$› **and** ‹*proj2-set-Col* $\{p,q,r,t\}$›
and ‹$p \neq q$› **and** ‹$r \neq p$› **and** ‹$s \neq p$› **and** ‹$t \neq p$› **and** *proj2-set-Col-coeff*
have r = *proj2-abs* ($?i *_R ?u + ?v$)
and s = *proj2-abs* ($?j *_R ?u + ?v$)
and t = *proj2-abs* ($?k *_R ?u + ?v$)
by *simp-all*

from ‹$r \neq q$› **and** ‹r = *proj2-abs* ($?i *_R ?u + ?v$)›
have $?i \neq 0$ **by** (*auto simp add*: *proj2-abs-rep*)
with ‹*cross-ratio* p q r s = *cross-ratio* p q r t›
have $?j = ?k$ **by** (*unfold cross-ratio-def*) *simp*
with ‹s = *proj2-abs* ($?j *_R ?u + ?v$)› **and** ‹t = *proj2-abs* ($?k *_R ?u + ?v$)›
show $s = t$ **by** *simp*
qed

lemma *cltn2-three-point-line*:
assumes $p \neq q$ **and** $r \neq p$ **and** $r \neq q$
and *proj2-incident* p l **and** *proj2-incident* q l **and** *proj2-incident* r l
and *apply-cltn2* p $C = p$ **and** *apply-cltn2* q $C = q$ **and** *apply-cltn2* r $C = r$
and *proj2-incident* s l
shows *apply-cltn2* s $C = s$ (**is** $?sC = s$)
proof *cases*
assume $s = p$
with ‹*apply-cltn2* p $C = p$› **show** $?sC = s$ **by** *simp*
next

assume $s \neq p$

let *?pC* = *apply-cltn2 p C*
let *?qC* = *apply-cltn2 q C*
let *?rC* = *apply-cltn2 r C*

from ‹*proj2-incident p l*› **and** ‹*proj2-incident q l*› **and** ‹*proj2-incident r l*›
and ‹*proj2-incident s l*›
have *proj2-set-Col* {*p*,*q*,*r*,*s*} **by** (*unfold proj2-set-Col-def*) *auto*
with ‹$p \neq q$› **and** ‹$r \neq p$› **and** ‹$s \neq p$› **and** ‹$r \neq q$›
have *cross-ratio-correct p q r s* **by** (*unfold cross-ratio-correct-def*) *simp*
hence *cross-ratio-correct ?pC ?qC ?rC ?sC*
by (*rule cross-ratio-correct-cltn2*)
with ‹*?pC* = *p*› **and** ‹*?qC* = *q*› **and** ‹*?rC* = *r*›
have *cross-ratio-correct p q r ?sC* **by** *simp*

from ‹*proj2-set-Col* {*p*,*q*,*r*,*s*}› **and** ‹$p \neq q$› **and** ‹$r \neq p$› **and** ‹$s \neq p$›
have *cross-ratio ?pC ?qC ?rC ?sC* = *cross-ratio p q r s*
by (*rule cross-ratio-cltn2*)
with ‹*?pC* = *p*› **and** ‹*?qC* = *q*› **and** ‹*?rC* = *r*›
have *cross-ratio p q r ?sC* = *cross-ratio p q r s* **by** *simp*
with ‹*cross-ratio-correct p q r ?sC*› **and** ‹*cross-ratio-correct p q r s*›
show *?sC* = *s* **by** (*rule cross-ratio-unique*)
qed

lemma *cross-ratio-equal-cltn2*:
assumes *cross-ratio-correct p q r s*
and *cross-ratio-correct* (*apply-cltn2 p C*) (*apply-cltn2 q C*)
(*apply-cltn2 r C*) *t*
(**is** *cross-ratio-correct ?pC ?qC ?rC t*)
and *cross-ratio* (*apply-cltn2 p C*) (*apply-cltn2 q C*) (*apply-cltn2 r C*) *t*
= *cross-ratio p q r s*
shows *t* = *apply-cltn2 s C* (**is** *t* = *?sC*)
proof −
from ‹*cross-ratio-correct p q r s*›
have *cross-ratio-correct ?pC ?qC ?rC ?sC* **by** (*rule cross-ratio-correct-cltn2*)

from ‹*cross-ratio-correct p q r s*›
have *proj2-set-Col* {*p*,*q*,*r*,*s*} **and** $p \neq q$ **and** $r \neq p$ **and** $s \neq p$
by (*unfold cross-ratio-correct-def*) *simp-all*
hence *cross-ratio ?pC ?qC ?rC ?sC* = *cross-ratio p q r s*
by (*rule cross-ratio-cltn2*)
with ‹*cross-ratio ?pC ?qC ?rC t* = *cross-ratio p q r s*›
have *cross-ratio ?pC ?qC ?rC t* = *cross-ratio ?pC ?qC ?rC ?sC* **by** *simp*
with ‹*cross-ratio-correct ?pC ?qC ?rC t*›
and ‹*cross-ratio-correct ?pC ?qC ?rC ?sC*›
show *t* = *?sC* **by** (*rule cross-ratio-unique*)
qed

lemma *proj2-Col-distinct-coeff-non-zero*:
  assumes *proj2-Col p q r* **and** *p ≠ q* **and** *r ≠ p* **and** *r ≠ q*
  shows *proj2-Col-coeff p q r ≠ 0*
proof
  assume *proj2-Col-coeff p q r = 0*

  from ⟨*proj2-Col p q r*⟩ **and** ⟨*p ≠ q*⟩ **and** ⟨*r ≠ p*⟩
  have *r = proj2-abs* ((*proj2-Col-coeff p q r*) $*_R$ *proj2-rep p + proj2-rep q*)
    by (*rule proj2-Col-coeff*)
  with ⟨*proj2-Col-coeff p q r = 0*⟩ **have** *r = q* **by** (*simp add*: *proj2-abs-rep*)
  with ⟨*r ≠ q*⟩ **show** *False* **..**
qed

lemma *cross-ratio-product*:
  assumes *proj2-Col p q s* **and** *p ≠ q* **and** *s ≠ p* **and** *s ≠ q*
  shows *cross-ratio p q r s * cross-ratio p q s t = cross-ratio p q r t*
proof −
  from ⟨*proj2-Col p q s*⟩ **and** ⟨*p ≠ q*⟩ **and** ⟨*s ≠ p*⟩ **and** ⟨*s ≠ q*⟩
  have *proj2-Col-coeff p q s ≠ 0* **by** (*rule proj2-Col-distinct-coeff-non-zero*)
  thus *cross-ratio p q r s * cross-ratio p q s t = cross-ratio p q r t*
    by (*unfold cross-ratio-def*) *simp*
qed

lemma *cross-ratio-equal-1*:
  assumes *proj2-Col p q r* **and** *p ≠ q* **and** *r ≠ p* **and** *r ≠ q*
  shows *cross-ratio p q r r = 1*
proof −
  from ⟨*proj2-Col p q r*⟩ **and** ⟨*p ≠ q*⟩ **and** ⟨*r ≠ p*⟩ **and** ⟨*r ≠ q*⟩
  have *proj2-Col-coeff p q r ≠ 0* **by** (*rule proj2-Col-distinct-coeff-non-zero*)
  thus *cross-ratio p q r r = 1* **by** (*unfold cross-ratio-def*) *simp*
qed

lemma *cross-ratio-1-equal*:
  assumes *cross-ratio-correct p q r s* **and** *cross-ratio p q r s = 1*
  shows *r = s*
proof −
  from ⟨*cross-ratio-correct p q r s*⟩
  have *proj2-set-Col* {*p*,*q*,*r*,*s*} **and** *p ≠ q* **and** *r ≠ p* **and** *r ≠ q*
    by (*unfold cross-ratio-correct-def*) *simp-all*

  from ⟨*proj2-set-Col* {*p*,*q*,*r*,*s*}⟩
  have *proj2-set-Col* {*p*,*q*,*r*}
    by (*simp add*: *proj2-subset-Col* [*of* {*p*,*q*,*r*} {*p*,*q*,*r*,*s*}])
  with ⟨*p ≠ q*⟩ **and** ⟨*r ≠ p*⟩ **and** ⟨*r ≠ q*⟩
  have *cross-ratio-correct p q r r* **by** (*unfold cross-ratio-correct-def*) *simp*

  from ⟨*proj2-set-Col* {*p*,*q*,*r*}⟩
  have *proj2-Col p q r* **by** (*subst proj2-Col-iff-set-Col*)
  with ⟨*p ≠ q*⟩ **and** ⟨*r ≠ p*⟩ **and** ⟨*r ≠ q*⟩

```
  have cross-ratio p q r r = 1 by (simp add: cross-ratio-equal-1)
  with ‹cross-ratio p q r s = 1›
  have cross-ratio p q r r = cross-ratio p q r s by simp
  with ‹cross-ratio-correct p q r r› and ‹cross-ratio-correct p q r s›
  show r = s by (rule cross-ratio-unique)
qed

lemma cross-ratio-swap-34:
  shows cross-ratio p q s r = 1 / (cross-ratio p q r s)
  by (unfold cross-ratio-def) simp

lemma cross-ratio-swap-13-24:
  assumes cross-ratio-correct p q r s and r ≠ s
  shows cross-ratio r s p q = cross-ratio p q r s
proof −
  from ‹cross-ratio-correct p q r s›
  have proj2-set-Col {p,q,r,s} and p ≠ q and r ≠ p and s ≠ p and r ≠ q
    by (unfold cross-ratio-correct-def, simp-all)

  have proj2-rep p ≠ 0 (is ?u ≠ 0) and proj2-rep q ≠ 0 (is ?v ≠ 0)
    by (rule proj2-rep-non-zero)+

  have p = proj2-abs ?u and q = proj2-abs ?v
    by (simp-all add: proj2-abs-rep)
  with ‹p ≠ q› have proj2-abs ?u ≠ proj2-abs ?v by simp

  let ?i = proj2-Col-coeff p q r
  let ?j = proj2-Col-coeff p q s
  from ‹proj2-set-Col {p,q,r,s}› and ‹p ≠ q› and ‹r ≠ p› and ‹s ≠ p›
  have r = proj2-abs (?i *R ?u + ?v) (is r = proj2-abs ?w)
    and s = proj2-abs (?j *R ?u + ?v) (is s = proj2-abs ?x)
    by (simp-all add: proj2-set-Col-coeff)
  with ‹r ≠ s› have ?i ≠ ?j by auto

  from ‹?u ≠ 0› and ‹?v ≠ 0› and ‹proj2-abs ?u ≠ proj2-abs ?v›
    and dependent-proj2-abs [of ?u ?v - 1]
  have ?w ≠ 0 and ?x ≠ 0 by auto

  from ‹r = proj2-abs (?i *R ?u + ?v)› and ‹r ≠ q›
  have ?i ≠ 0 by (auto simp add: proj2-abs-rep)

  have ?w − ?x = (?i − ?j) *R ?u by (simp add: algebra-simps)
  with ‹?i ≠ ?j›
  have p = proj2-abs (?w − ?x) by (simp add: proj2-abs-mult-rep)

  have ?j *R ?w − ?i *R ?x = (?j − ?i) *R ?v by (simp add: algebra-simps)
  with ‹?i ≠ ?j›
  have q = proj2-abs (?j *R ?w − ?i *R ?x) by (simp add: proj2-abs-mult-rep)
  with ‹?w ≠ 0› and ‹?x ≠ 0› and ‹r ≠ s› and ‹?i ≠ 0› and ‹r = proj2-abs ?w›
```

and ‹$s = proj2\text{-}abs\ ?x$› **and** ‹$p = proj2\text{-}abs\ (?w - ?x)$›
and *cross-ratio-abs* [*of ?w ?x −1 −?i 1 ?j*]
have $cross\text{-}ratio\ r\ s\ p\ q = ?j\ /\ ?i$ **by** (*simp add*: *algebra-simps*)
thus $cross\text{-}ratio\ r\ s\ p\ q = cross\text{-}ratio\ p\ q\ r\ s$
by (*unfold cross-ratio-def* [*of p q r s*], *simp*)
qed

lemma *cross-ratio-swap-12*:
assumes *cross-ratio-correct p q r s* **and** *cross-ratio-correct q p r s*
shows $cross\text{-}ratio\ q\ p\ r\ s = 1\ /\ (cross\text{-}ratio\ p\ q\ r\ s)$
proof *cases*
assume $r = s$

from ‹*cross-ratio-correct p q r s*›
have *proj2-set-Col* $\{p,q,r,s\}$ **and** $p \neq q$ **and** $r \neq p$ **and** $r \neq q$
by (*unfold cross-ratio-correct-def*) *simp-all*

from ‹*proj2-set-Col* $\{p,q,r,s\}$› **and** ‹$r = s$›
have *proj2-Col p q r* **by** (*simp-all add*: *proj2-Col-iff-set-Col*)
hence *proj2-Col q p r* **by** (*rule proj2-Col-permute*)
with ‹*proj2-Col p q r*› **and** ‹$p \neq q$› **and** ‹$r \neq p$› **and** ‹$r \neq q$› **and** ‹$r = s$›
have $cross\text{-}ratio\ p\ q\ r\ s = 1$ **and** $cross\text{-}ratio\ q\ p\ r\ s = 1$
by (*simp-all add*: *cross-ratio-equal-1*)
thus $cross\text{-}ratio\ q\ p\ r\ s = 1\ /\ (cross\text{-}ratio\ p\ q\ r\ s)$ **by** *simp*
next
assume $r \neq s$
with ‹*cross-ratio-correct q p r s*›
have $cross\text{-}ratio\ q\ p\ r\ s = cross\text{-}ratio\ r\ s\ q\ p$
by (*simp add*: *cross-ratio-swap-13-24*)
also have $\ldots = 1\ /\ (cross\text{-}ratio\ r\ s\ p\ q)$ **by** (*rule cross-ratio-swap-34*)
also from ‹*cross-ratio-correct p q r s*› **and** ‹$r \neq s$›
have $\ldots = 1\ /\ (cross\text{-}ratio\ p\ q\ r\ s)$ **by** (*simp add*: *cross-ratio-swap-13-24*)
finally show $cross\text{-}ratio\ q\ p\ r\ s = 1\ /\ (cross\text{-}ratio\ p\ q\ r\ s)$ **.**
qed

7.6 Cartesian subspace of the real projective plane

definition *vector2-append1* :: $real\hat{\ }2 \Rightarrow real\hat{\ }3$ **where**
$vector2\text{-}append1\ v = vector\ [v\$1,\ v\$2,\ 1]$

lemma *vector2-append1-non-zero*: $vector2\text{-}append1\ v \neq 0$
proof −
have $(vector2\text{-}append1\ v)\$3 \neq 0\$3$
unfolding *vector2-append1-def* **and** *vector-def*
by *simp*
thus $vector2\text{-}append1\ v \neq 0$ **by** *auto*
qed

definition *proj2-pt* :: $real\hat{\ }2 \Rightarrow proj2$ **where**

proj2-pt v ≜ *proj2-abs* (*vector2-append1 v*)

lemma *proj2-pt-scalar*:
∃ *c*. *c* ≠ *0* ∧ *proj2-rep* (*proj2-pt v*) = *c* $*_R$ *vector2-append1 v*
unfolding *proj2-pt-def*
by (*simp add*: *proj2-rep-abs2 vector2-append1-non-zero*)

abbreviation *z-non-zero* :: *proj2* ⇒ *bool* **where**
z-non-zero p ≜ (*proj2-rep p*)$*3* ≠ *0*

definition *cart2-pt* :: *proj2* ⇒ *real*ˆ*2* **where**
cart2-pt p ≜
vector [(*proj2-rep p*)$*1* / (*proj2-rep p*)$*3*, (*proj2-rep p*)$*2* / (*proj2-rep p*)$*3*]

definition *cart2-append1* :: *proj2* ⇒ *real*ˆ*3* **where**
cart2-append1 p ≜ (*1* / ((*proj2-rep p*)$*3*)) $*_R$ *proj2-rep p*

lemma *cart2-append1-z*:
assumes *z-non-zero p*
shows (*cart2-append1 p*)$*3* = *1*
using ‹*z-non-zero p*›
by (*unfold cart2-append1-def*) *simp*

lemma *cart2-append1-non-zero*:
assumes *z-non-zero p*
shows *cart2-append1 p* ≠ *0*
proof −
from ‹*z-non-zero p*› **have** (*cart2-append1 p*)$*3* = *1* **by** (*rule cart2-append1-z*)
thus *cart2-append1 p* ≠ *0* **by** (*simp add*: *vec-eq-iff exI* [*of* - *3*])
qed

lemma *proj2-rep-cart2-append1*:
assumes *z-non-zero p*
shows *proj2-rep p* = ((*proj2-rep p*)$*3*) $*_R$ *cart2-append1 p*
using ‹*z-non-zero p*›
by (*unfold cart2-append1-def*) *simp*

lemma *proj2-abs-cart2-append1*:
assumes *z-non-zero p*
shows *proj2-abs* (*cart2-append1 p*) = *p*
proof −
from ‹*z-non-zero p*›
have *proj2-abs* (*cart2-append1 p*) = *proj2-abs* (*proj2-rep p*)
by (*unfold cart2-append1-def*) (*simp add*: *proj2-abs-mult*)
thus *proj2-abs* (*cart2-append1 p*) = *p* **by** (*simp add*: *proj2-abs-rep*)
qed

lemma *cart2-append1-inj*:
assumes *z-non-zero p* **and** *cart2-append1 p* = *cart2-append1 q*

shows $p = q$
proof −
from ‹*z-non-zero p*› **have** (*cart2-append1 p*)$*3* = *1* **by** (*rule cart2-append1-z*)
with ‹*cart2-append1 p* = *cart2-append1 q*›
have (*cart2-append1 q*)$*3* = *1* **by** *simp*
hence *z-non-zero q* **by** (*unfold cart2-append1-def*) *auto*

from ‹*cart2-append1 p* = *cart2-append1 q*›
have *proj2-abs* (*cart2-append1 p*) = *proj2-abs* (*cart2-append1 q*) **by** *simp*
with ‹*z-non-zero p*› **and** ‹*z-non-zero q*›
show $p = q$ **by** (*simp add*: *proj2-abs-cart2-append1*)
qed

lemma *cart2-append1*:
assumes *z-non-zero p*
shows *vector2-append1* (*cart2-pt p*) = *cart2-append1 p*
using ‹*z-non-zero p*›
unfolding *vector2-append1-def*
and *cart2-append1-def*
and *cart2-pt-def*
and *vector-def*
by (*simp add*: *vec-eq-iff forall-3*)

lemma *cart2-proj2*: *cart2-pt* (*proj2-pt v*) = *v*
proof −
let *?v′* = *vector2-append1 v*
let *?p* = *proj2-pt v*
from *proj2-pt-scalar*
obtain *c* **where** $c \neq 0$ **and** *proj2-rep ?p* = $c *_R$ *?v′* **by** *auto*
hence (*cart2-pt ?p*)$*1* = *v*$*1* **and** (*cart2-pt ?p*)$*2* = *v*$*2*
unfolding *cart2-pt-def* **and** *vector2-append1-def* **and** *vector-def*
by *simp+*
thus *cart2-pt ?p* = *v* **by** (*simp add*: *vec-eq-iff forall-2*)
qed

lemma *z-non-zero-proj2-pt*: *z-non-zero* (*proj2-pt v*)
proof −
from *proj2-pt-scalar*
obtain *c* **where** $c \neq 0$ **and** *proj2-rep* (*proj2-pt v*) = $c *_R$ (*vector2-append1 v*)
by *auto*
from ‹*proj2-rep* (*proj2-pt v*) = $c *_R$ (*vector2-append1 v*)›
have (*proj2-rep* (*proj2-pt v*))$*3* = *c*
unfolding *vector2-append1-def* **and** *vector-def*
by *simp*
with ‹$c \neq 0$› **show** *z-non-zero* (*proj2-pt v*) **by** *simp*
qed

lemma *cart2-append1-proj2*: *cart2-append1* (*proj2-pt v*) = *vector2-append1 v*
proof −

from *z-non-zero-proj2-pt*
have *cart2-append1* (*proj2-pt v*) = *vector2-append1* (*cart2-pt* (*proj2-pt v*))
by (*simp add*: *cart2-append1*)
thus *cart2-append1* (*proj2-pt v*) = *vector2-append1 v*
by (*simp add*: *cart2-proj2*)
qed

lemma *proj2-pt-inj*: *inj proj2-pt*
by (*simp add*: *inj-on-inverseI* [*of UNIV cart2-pt proj2-pt*] *cart2-proj2*)

lemma *proj2-cart2*:
assumes *z-non-zero p*
shows *proj2-pt* (*cart2-pt p*) = *p*
proof −
from ‹*z-non-zero p*›
have (*proj2-rep p*)\$*3* $*_R$ *vector2-append1* (*cart2-pt p*) = *proj2-rep p*
unfolding *vector2-append1-def* **and** *cart2-pt-def* **and** *vector-def*
by (*simp add*: *vec-eq-iff forall-3*)
with ‹*z-non-zero p*›
and *proj2-abs-mult* [*of* (*proj2-rep p*)\$*3 vector2-append1* (*cart2-pt p*)]
have *proj2-abs* (*vector2-append1* (*cart2-pt p*)) = *proj2-abs* (*proj2-rep p*)
by *simp*
thus *proj2-pt* (*cart2-pt p*) = *p*
by (*unfold proj2-pt-def*) (*simp add*: *proj2-abs-rep*)
qed

lemma *cart2-injective*:
assumes *z-non-zero p* **and** *z-non-zero q* **and** *cart2-pt p* = *cart2-pt q*
shows *p* = *q*
proof −
from ‹*z-non-zero p*› **and** ‹*z-non-zero q*›
have *proj2-pt* (*cart2-pt p*) = *p* **and** *proj2-pt* (*cart2-pt q*) = *q*
by (*simp-all add*: *proj2-cart2*)

from ‹*proj2-pt* (*cart2-pt p*) = *p*› **and** ‹*cart2-pt p* = *cart2-pt q*›
have *proj2-pt* (*cart2-pt q*) = *p* **by** *simp*
with ‹*proj2-pt* (*cart2-pt q*) = *q*› **show** *p* = *q* **by** *simp*
qed

lemma *proj2-Col-iff-euclid*:
proj2-Col (*proj2-pt a*) (*proj2-pt b*) (*proj2-pt c*) ⟷ *real-euclid.Col a b c*
(**is** *proj2-Col ?p ?q ?r* ⟷ -)
proof
let *?a′* = *vector2-append1 a*
let *?b′* = *vector2-append1 b*
let *?c′* = *vector2-append1 c*
let *?a″* = *proj2-rep ?p*
let *?b″* = *proj2-rep ?q*
let *?c″* = *proj2-rep ?r*

from *proj2-pt-scalar* **obtain** i **and** j **and** k **where**
$i \neq 0$ **and** $?a'' = i *_R\ ?a'$
and $j \neq 0$ **and** $?b'' = j *_R\ ?b'$
and $k \neq 0$ **and** $?c'' = k *_R\ ?c'$
by *metis*
hence $?a' = (1/i) *_R\ ?a''$
and $?b' = (1/j) *_R\ ?b''$
and $?c' = (1/k) *_R\ ?c''$
by *simp-all*

{ assume *proj2-Col* $?p\ ?q\ ?r$
then obtain i' **and** j' **and** k' **where**
$i' *_R\ ?a'' + j' *_R\ ?b'' + k' *_R\ ?c'' = 0$ **and** $i'{\neq}0 \vee j'{\neq}0 \vee k'{\neq}0$
unfolding *proj2-Col-def*
by *auto*

let $?i'' = i * i'$
let $?j'' = j * j'$
let $?k'' = k * k'$
from $\langle i{\neq}0\rangle$ **and** $\langle j{\neq}0\rangle$ **and** $\langle k{\neq}0\rangle$ **and** $\langle i'{\neq}0 \vee j'{\neq}0 \vee k'{\neq}0\rangle$
have $?i''{\neq}0 \vee ?j''{\neq}0 \vee ?k''{\neq}0$ **by** *simp*

from $\langle i' *_R\ ?a'' + j' *_R\ ?b'' + k' *_R\ ?c'' = 0\rangle$
and $\langle ?a'' = i *_R\ ?a'\rangle$
and $\langle ?b'' = j *_R\ ?b'\rangle$
and $\langle ?c'' = k *_R\ ?c'\rangle$
have $?i'' *_R\ ?a' + ?j'' *_R\ ?b' + ?k'' *_R\ ?c' = 0$
by (*simp add*: *ac-simps*)
hence $(?i'' *_R\ ?a' + ?j'' *_R\ ?b' + ?k'' *_R\ ?c')\$3 = 0$
by *simp*
hence $?i'' + ?j'' + ?k'' = 0$
unfolding *vector2-append1-def* **and** *vector-def*
by *simp*

have $(?i'' *_R\ ?a' + ?j'' *_R\ ?b' + ?k'' *_R\ ?c')\$1 =$
$(?i'' *_R\ a + ?j'' *_R\ b + ?k'' *_R\ c)\$1$
and $(?i'' *_R\ ?a' + ?j'' *_R\ ?b' + ?k'' *_R\ ?c')\$2 =$
$(?i'' *_R\ a + ?j'' *_R\ b + ?k'' *_R\ c)\$2$
unfolding *vector2-append1-def* **and** *vector-def*
by *simp+*
with $\langle ?i'' *_R\ ?a' + ?j'' *_R\ ?b' + ?k'' *_R\ ?c' = 0\rangle$
have $?i'' *_R\ a + ?j'' *_R\ b + ?k'' *_R\ c = 0$
by (*simp add*: *vec-eq-iff forall-2*)

have *dep2* $(b - a)\ (c - a)$
proof *cases*
assume $?k'' = 0$
with $\langle ?i'' + ?j'' + ?k'' = 0\rangle$ **have** $?j'' = -?i''$ **by** *simp*
with $\langle ?i''{\neq}0 \vee ?j''{\neq}0 \vee ?k''{\neq}0\rangle$ **and** $\langle ?k'' = 0\rangle$ **have** $?i'' \neq 0$ **by** *simp*

from ‹$?i'' *_R\ a + ?j'' *_R\ b + ?k'' *_R\ c = 0$›
  and ‹$?k'' = 0$› **and** ‹$?j'' = -?i''$›
have $?i'' *_R\ a + (-?i'' *_R\ b) = 0$ **by** *simp*
with ‹$?i'' \neq 0$› **have** $a = b$ **by** (*simp add*: *algebra-simps*)
hence $b - a = 0 *_R (c - a)$ **by** *simp*
moreover have $c - a = 1 *_R (c - a)$ **by** *simp*
ultimately have $\exists\ x\ t\ s.\ b - a = t *_R\ x \wedge c - a = s *_R\ x$
  by *blast*
thus *dep2* $(b - a)$ $(c - a)$ **unfolding** *dep2-def* **.**
next
  assume $?k'' \neq 0$
  from ‹$?i'' + ?j'' + ?k'' = 0$› **have** $?i'' = -(?j'' + ?k'')$ **by** *simp*
  with ‹$?i'' *_R\ a + ?j'' *_R\ b + ?k'' *_R\ c = 0$›
  have $-(?j'' + ?k'') *_R\ a + ?j'' *_R\ b + ?k'' *_R\ c = 0$ **by** *simp*
  hence $?k'' *_R (c - a) = -\ ?j'' *_R (b - a)$
    by (*simp add*: *scaleR-left-distrib*
      scaleR-right-diff-distrib
      scaleR-left-diff-distrib
      algebra-simps)
  hence $(1 / ?k'') *_R\ ?k'' *_R (c - a) = (-?j'' / ?k'') *_R (b - a)$
    by *simp*
  with ‹$?k'' \neq 0$› **have** $c - a = (-?j'' / ?k'') *_R (b - a)$ **by** *simp*
  moreover have $b - a = 1 *_R (b - a)$ **by** *simp*
  ultimately have $\exists\ x\ t\ s.\ b - a = t *_R\ x \wedge c - a = s *_R\ x$ **by** *blast*
  thus *dep2* $(b - a)$ $(c - a)$ **unfolding** *dep2-def* **.**
qed
with *Col-dep2* **show** *real-euclid.Col a b c* **by** *auto*
}

{ **assume** *real-euclid.Col a b c*
with *Col-dep2* **have** *dep2* $(b - a)$ $(c - a)$ **by** *auto*
then obtain x **and** t **and** s **where** $b - a = t *_R\ x$ **and** $c - a = s *_R\ x$
  unfolding *dep2-def*
  by *auto*

show *proj2-Col ?p ?q ?r*
proof *cases*
  assume $t = 0$
  with ‹$b - a = t *_R\ x$› **have** $a = b$ **by** *simp*
  with *proj2-Col-coincide* **show** *proj2-Col ?p ?q ?r* **by** *simp*
next
  assume $t \neq 0$

  from ‹$b - a = t *_R\ x$› **and** ‹$c - a = s *_R\ x$›
  have $s *_R (b - a) = t *_R (c - a)$ **by** *simp*
  hence $(s - t) *_R\ a + (-s) *_R\ b + t *_R\ c = 0$
    by (*simp add*: *scaleR-right-diff-distrib*
      scaleR-left-diff-distrib

algebra-simps)
hence $((s - t) *_R\ ?a' + (-s) *_R\ ?b' + t *_R\ ?c')\$1 = 0$
and $((s - t) *_R\ ?a' + (-s) *_R\ ?b' + t *_R\ ?c')\$2 = 0$
unfolding *vector2-append1-def* **and** *vector-def*
by (*simp-all add*: *vec-eq-iff*)
moreover have $((s - t) *_R\ ?a' + (-s) *_R\ ?b' + t *_R\ ?c')\$3 = 0$
unfolding *vector2-append1-def* **and** *vector-def*
by *simp*
ultimately have $(s - t) *_R\ ?a' + (-s) *_R\ ?b' + t *_R\ ?c' = 0$
by (*simp add*: *vec-eq-iff forall-3*)
with ‹$?a' = (1/i) *_R\ ?a''$›
and ‹$?b' = (1/j) *_R\ ?b''$›
and ‹$?c' = (1/k) *_R\ ?c''$›
have $((s - t)/i) *_R\ ?a'' + (-s/j) *_R\ ?b'' + (t/k) *_R\ ?c'' = 0$
by *simp*
moreover from ‹$t \neq 0$› **and** ‹$k \neq 0$› **have** $t/k \neq 0$ **by** *simp*
ultimately show *proj2-Col ?p ?q ?r*
unfolding *proj2-Col-def*
by *blast*
qed
}
qed

lemma *proj2-Col-iff-euclid-cart2*:
assumes *z-non-zero p* **and** *z-non-zero q* **and** *z-non-zero r*
shows
proj2-Col p q r ⟷ *real-euclid.Col* (*cart2-pt p*) (*cart2-pt q*) (*cart2-pt r*)
(**is** - ⟷ *real-euclid.Col ?a ?b ?c*)
proof −
from ‹*z-non-zero p*› **and** ‹*z-non-zero q*› **and** ‹*z-non-zero r*›
have *proj2-pt ?a* = *p* **and** *proj2-pt ?b* = *q* **and** *proj2-pt ?c* = *r*
by (*simp-all add*: *proj2-cart2*)
with *proj2-Col-iff-euclid* [*of ?a ?b ?c*]
show *proj2-Col p q r* ⟷ *real-euclid.Col ?a ?b ?c* **by** *simp*
qed

lemma *euclid-Col-cart2-incident*:
assumes *z-non-zero p* **and** *z-non-zero q* **and** *z-non-zero r* **and** $p \neq q$
and *proj2-incident p l* **and** *proj2-incident q l*
and *real-euclid.Col* (*cart2-pt p*) (*cart2-pt q*) (*cart2-pt r*)
(**is** *real-euclid.Col ?cp ?cq ?cr*)
shows *proj2-incident r l*
proof −
from ‹*z-non-zero p*› **and** ‹*z-non-zero q*› **and** ‹*z-non-zero r*›
and ‹*real-euclid.Col ?cp ?cq ?cr*›
have *proj2-Col p q r* **by** (*subst proj2-Col-iff-euclid-cart2*, *simp-all*)
hence *proj2-set-Col* {*p*,*q*,*r*} **by** (*simp add*: *proj2-Col-iff-set-Col*)
then obtain *m* **where**
proj2-incident p m **and** *proj2-incident q m* **and** *proj2-incident r m*

by (*unfold proj2-set-Col-def*, *auto*)

from ‹$p \neq q$› **and** ‹*proj2-incident p l*› **and** ‹*proj2-incident q l*›
and ‹*proj2-incident p m*› **and** ‹*proj2-incident q m*› **and** *proj2-incident-unique*
have $l = m$ **by** *auto*
with ‹*proj2-incident r m*› **show** *proj2-incident r l* **by** *simp*
qed

lemma *euclid-B-cart2-common-line*:
assumes *z-non-zero p* **and** *z-non-zero q* **and** *z-non-zero r*
and $B_{\mathbb{R}}$ (*cart2-pt p*) (*cart2-pt q*) (*cart2-pt r*)
(**is** $B_{\mathbb{R}}$ *?cp ?cq ?cr*)
shows ∃ *l. proj2-incident p l* ∧ *proj2-incident q l* ∧ *proj2-incident r l*
proof −
from ‹*z-non-zero p*› **and** ‹*z-non-zero q*› **and** ‹*z-non-zero r*›
and ‹$B_{\mathbb{R}}$ *?cp ?cq ?cr*› **and** *proj2-Col-iff-euclid-cart2*
have *proj2-Col p q r* **by** (*unfold real-euclid.Col-def*) *simp*
hence *proj2-set-Col* {*p*,*q*,*r*} **by** (*simp add*: *proj2-Col-iff-set-Col*)
thus ∃ *l. proj2-incident p l* ∧ *proj2-incident q l* ∧ *proj2-incident r l*
by (*unfold proj2-set-Col-def*) *simp*
qed

lemma *cart2-append1-between*:
assumes *z-non-zero p* **and** *z-non-zero q* **and** *z-non-zero r*
shows $B_{\mathbb{R}}$ (*cart2-pt p*) (*cart2-pt q*) (*cart2-pt r*)
⟷ (∃ *k*≥*0*. $k \leq 1$
∧ *cart2-append1 q* = $k *_R$ *cart2-append1 r* + (*1* − *k*) $*_R$ *cart2-append1 p*)
proof −
let *?cp* = *cart2-pt p*
let *?cq* = *cart2-pt q*
let *?cr* = *cart2-pt r*
let *?cp1* = *vector2-append1 ?cp*
let *?cq1* = *vector2-append1 ?cq*
let *?cr1* = *vector2-append1 ?cr*
from ‹*z-non-zero p*› **and** ‹*z-non-zero q*› **and** ‹*z-non-zero r*›
have *?cp1* = *cart2-append1 p*
and *?cq1* = *cart2-append1 q*
and *?cr1* = *cart2-append1 r*
by (*simp-all add*: *cart2-append1*)

have ∀ *k. ?cq* − *?cp* = $k *_R$ (*?cr* − *?cp*) ⟷ *?cq* = $k *_R$ *?cr* + (*1* − *k*) $*_R$
?cp
by (*simp add*: *algebra-simps*)
hence ∀ *k. ?cq* − *?cp* = $k *_R$ (*?cr* − *?cp*)
⟷ *?cq1* = $k *_R$ *?cr1* + (*1* − *k*) $*_R$ *?cp1*
unfolding *vector2-append1-def* **and** *vector-def*
by (*simp add*: *vec-eq-iff forall-2 forall-3*)
with ‹*?cp1* = *cart2-append1 p*›
and ‹*?cq1* = *cart2-append1 q*›

and ‹?cr1 = cart2-append1 r›
have ∀ k. ?cq − ?cp = k $*_R$ (?cr − ?cp)
⟷ cart2-append1 q = k $*_R$ cart2-append1 r + (1 − k) $*_R$ cart2-append1 p
by *simp*
thus $B_{\mathbb{R}}$ (cart2-pt p) (cart2-pt q) (cart2-pt r)
⟷ (∃ k≥0. k ≤ 1
∧ cart2-append1 q = k $*_R$ cart2-append1 r + (1 − k) $*_R$ cart2-append1 p)
by (*unfold real-euclid-B-def*) *simp*
qed

lemma *cart2-append1-between-right-strict*:
assumes *z-non-zero p* **and** *z-non-zero q* **and** *z-non-zero r*
and $B_{\mathbb{R}}$ (cart2-pt p) (cart2-pt q) (cart2-pt r) **and** q ≠ r
shows ∃ k≥0. k < 1
∧ cart2-append1 q = k $*_R$ cart2-append1 r + (1 − k) $*_R$ cart2-append1 p
proof −
from ‹z-non-zero p› **and** ‹z-non-zero q› **and** ‹z-non-zero r›
and ‹$B_{\mathbb{R}}$ (cart2-pt p) (cart2-pt q) (cart2-pt r)› **and** *cart2-append1-between*
obtain k **where** k ≥ 0 **and** k ≤ 1
and cart2-append1 q = k $*_R$ cart2-append1 r + (1 − k) $*_R$ cart2-append1 p
by *auto*

have k ≠ 1
proof
assume k = 1
with ‹cart2-append1 q = k $*_R$ cart2-append1 r + (1 − k) $*_R$ cart2-append1 p›
have cart2-append1 q = cart2-append1 r **by** *simp*
with ‹z-non-zero q› **have** q = r **by** (*rule cart2-append1-inj*)
with ‹q ≠ r› **show** *False* **..**
qed
with ‹k ≤ 1› **have** k < 1 **by** *simp*
with ‹k ≥ 0›
and ‹cart2-append1 q = k $*_R$ cart2-append1 r + (1 − k) $*_R$ cart2-append1 p›
show ∃ k≥0. k < 1
∧ cart2-append1 q = k $*_R$ cart2-append1 r + (1 − k) $*_R$ cart2-append1 p
by (*simp add*: *exI* [*of* - k])
qed

lemma *cart2-append1-between-strict*:
assumes *z-non-zero p* **and** *z-non-zero q* **and** *z-non-zero r*
and $B_{\mathbb{R}}$ (cart2-pt p) (cart2-pt q) (cart2-pt r) **and** q ≠ p **and** q ≠ r
shows ∃ k>0. k < 1
∧ cart2-append1 q = k $*_R$ cart2-append1 r + (1 − k) $*_R$ cart2-append1 p
proof −
from ‹z-non-zero p› **and** ‹z-non-zero q› **and** ‹z-non-zero r›
and ‹$B_{\mathbb{R}}$ (cart2-pt p) (cart2-pt q) (cart2-pt r)› **and** ‹q ≠ r›
and *cart2-append1-between-right-strict* [*of* p q r]
obtain k **where** k ≥ 0 **and** k < 1
and cart2-append1 q = k $*_R$ cart2-append1 r + (1 − k) $*_R$ cart2-append1 p

by *auto*

have $k \neq 0$
proof
assume $k = 0$
with ‹$cart2\text{-}append1\ q = k *_R cart2\text{-}append1\ r + (1 - k) *_R cart2\text{-}append1\ p$›
have $cart2\text{-}append1\ q = cart2\text{-}append1\ p$ **by** *simp*
with ‹*z-non-zero q*› **have** $q = p$ **by** (*rule cart2-append1-inj*)
with ‹$q \neq p$› **show** *False* **..**
qed
with ‹$k \geq 0$› **have** $k > 0$ **by** *simp*
with ‹$k < 1$›
and ‹$cart2\text{-}append1\ q = k *_R cart2\text{-}append1\ r + (1 - k) *_R cart2\text{-}append1\ p$›
show $\exists\ k{>}0.\ k < 1$
$\wedge\ cart2\text{-}append1\ q = k *_R cart2\text{-}append1\ r + (1 - k) *_R cart2\text{-}append1\ p$
by (*simp add*: *exI* [*of* - *k*])
qed

end

8 The hyperbolic plane and Tarski's axioms

theory *Hyperbolic-Tarski*
imports *Euclid-Tarski*
Projective
HOL−Library.Quadratic-Discriminant
begin

8.1 Characterizing a specific conic in the projective plane

definition *M* :: *real^3^3* **where**
$M \triangleq vector$ [
vector [*1*, *0*, *0*],
vector [*0*, *1*, *0*],
vector [*0*, *0*, *−1*]]

lemma *M-symmatrix*: *symmatrix M*
unfolding *symmatrix-def* **and** *transpose-def* **and** *M-def*
by (*simp add*: *vec-eq-iff forall-3 vector-3*)

lemma *M-self-inverse*: $M ** M = mat\ 1$
unfolding *M-def* **and** *matrix-matrix-mult-def* **and** *mat-def* **and** *vector-def*
by (*simp add*: *sum-3 vec-eq-iff forall-3*)

lemma *M-invertible*: *invertible M*
unfolding *invertible-def*
using *M-self-inverse*
by *auto*

definition *polar* :: *proj2* ⇒ *proj2-line* **where**
polar p ≜ *proj2-line-abs* (*M* ∗*v proj2-rep p*)

definition *pole* :: *proj2-line* ⇒ *proj2* **where**
pole l ≜ *proj2-abs* (*M* ∗*v proj2-line-rep l*)

lemma *polar-abs*:
assumes *v* ≠ *0*
shows *polar* (*proj2-abs v*) = *proj2-line-abs* (*M* ∗*v v*)
proof −
from ‹*v* ≠ *0*› **and** *proj2-rep-abs2*
obtain *k* **where** *k* ≠ *0* **and** *proj2-rep* (*proj2-abs v*) = *k* $*_R$ *v* **by** *auto*
from ‹*proj2-rep* (*proj2-abs v*) = *k* $*_R$ *v*›
have *polar* (*proj2-abs v*) = *proj2-line-abs* (*k* $*_R$ (*M* ∗*v v*))
unfolding *polar-def*
by (*simp add*: *matrix-scaleR-vector-ac scaleR-matrix-vector-assoc*)
with ‹*k* ≠ *0*› **and** *proj2-line-abs-mult*
show *polar* (*proj2-abs v*) = *proj2-line-abs* (*M* ∗*v v*) **by** *simp*
qed

lemma *pole-abs*:
assumes *v* ≠ *0*
shows *pole* (*proj2-line-abs v*) = *proj2-abs* (*M* ∗*v v*)
proof −
from ‹*v* ≠ *0*› **and** *proj2-line-rep-abs*
obtain *k* **where** *k* ≠ *0* **and** *proj2-line-rep* (*proj2-line-abs v*) = *k* $*_R$ *v*
by *auto*
from ‹*proj2-line-rep* (*proj2-line-abs v*) = *k* $*_R$ *v*›
have *pole* (*proj2-line-abs v*) = *proj2-abs* (*k* $*_R$ (*M* ∗*v v*))
unfolding *pole-def*
by (*simp add*: *matrix-scaleR-vector-ac scaleR-matrix-vector-assoc*)
with ‹*k* ≠ *0*› **and** *proj2-abs-mult*
show *pole* (*proj2-line-abs v*) = *proj2-abs* (*M* ∗*v v*) **by** *simp*
qed

lemma *polar-rep-non-zero*: *M* ∗*v proj2-rep p* ≠ *0*
proof −
have *proj2-rep p* ≠ *0* **by** (*rule proj2-rep-non-zero*)
with *M-invertible*
show *M* ∗*v proj2-rep p* ≠ *0* **by** (*rule invertible-times-non-zero*)
qed

lemma *pole-polar*: *pole* (*polar p*) = *p*
proof −
from *polar-rep-non-zero*
have *pole* (*polar p*) = *proj2-abs* (*M* ∗*v* (*M* ∗*v proj2-rep p*))
unfolding *polar-def*
by (*rule pole-abs*)
with *M-self-inverse*

show *pole* (*polar* *p*) = *p*
  by (*simp add*: *matrix-vector-mul-assoc proj2-abs-rep*)
qed

lemma *pole-rep-non-zero*: $M *v$ *proj2-line-rep* $l \neq 0$
proof –
  have *proj2-line-rep* $l \neq 0$ **by** (*rule proj2-line-rep-non-zero*)
  with *M-invertible*
  show $M *v$ *proj2-line-rep* $l \neq 0$ **by** (*rule invertible-times-non-zero*)
qed

lemma *polar-pole*: *polar* (*pole* *l*) = *l*
proof –
  from *pole-rep-non-zero*
  have *polar* (*pole* *l*) = *proj2-line-abs* ($M *v$ ($M *v$ *proj2-line-rep* *l*))
    unfolding *pole-def*
    by (*rule polar-abs*)
  with *M-self-inverse*
  show *polar* (*pole* *l*) = *l*
    by (*simp add*: *matrix-vector-mul-assoc proj2-line-abs-rep*
      matrix-vector-mul-lid)
qed

lemma *polar-inj*:
  assumes *polar* *p* = *polar* *q*
  shows *p* = *q*
proof –
  from ‹*polar* *p* = *polar* *q*› **have** *pole* (*polar* *p*) = *pole* (*polar* *q*) **by** *simp*
  thus *p* = *q* **by** (*simp add*: *pole-polar*)
qed

definition *conic-sgn* :: *proj2* ⇒ *real* **where**
  conic-sgn *p* $\triangleq$ *sgn* (*proj2-rep* *p* · ($M *v$ *proj2-rep* *p*))

lemma *conic-sgn-abs*:
  assumes $v \neq 0$
  shows *conic-sgn* (*proj2-abs* *v*) = *sgn* (*v* · ($M *v$ *v*))
proof –
  from ‹$v \neq 0$› **and** *proj2-rep-abs2*
  obtain *j* **where** $j \neq 0$ **and** *proj2-rep* (*proj2-abs* *v*) = $j *_R v$ **by** *auto*

  from ‹*proj2-rep* (*proj2-abs* *v*) = $j *_R v$›
  have *conic-sgn* (*proj2-abs* *v*) = *sgn* ($j^2 *$ (*v* · ($M *v$ *v*)))
    unfolding *conic-sgn-def*
    by (*simp add*:
      matrix-scaleR-vector-ac
      scaleR-matrix-vector-assoc [*symmetric*]
      dot-scaleR-mult
      power2-eq-square

algebra-simps)
also have ... = *sgn* (j^2) ∗ *sgn* (*v* · (*M* ∗*v* *v*)) **by** (*rule sgn-mult*)
also from ‹*j* ≠ *0*› **have** ... = *sgn* (*v* · (*M* ∗*v* *v*))
by (*simp add*: *power2-eq-square sgn-mult*)
finally show *conic-sgn* (*proj2-abs v*) = *sgn* (*v* · (*M* ∗*v* *v*)) **.**
qed

lemma *sgn-conic-sgn*: *sgn* (*conic-sgn p*) = *conic-sgn p*
by (*unfold conic-sgn-def*) *simp*

definition *S* :: *proj2 set* **where**
S ≜ {*p*. *conic-sgn p* = *0*}

definition *K2* :: *proj2 set* **where**
K2 ≜ {*p*. *conic-sgn p* < *0*}

lemma *S-K2-empty*: *S* ∩ *K2* = {}
unfolding *S-def* **and** *K2-def*
by *auto*

lemma *K2-abs*:
assumes *v* ≠ *0*
shows *proj2-abs v* ∈ *K2* ⟷ *v* · (*M* ∗*v* *v*) < *0*
proof −
have *proj2-abs v* ∈ *K2* ⟷ *conic-sgn* (*proj2-abs v*) < *0*
by (*simp add*: *K2-def*)
with ‹*v* ≠ *0*› **and** *conic-sgn-abs*
show *proj2-abs v* ∈ *K2* ⟷ *v* · (*M* ∗*v* *v*) < *0* **by** *simp*
qed

definition *K2-centre* = *proj2-abs* (*vector* [*0*,*0*,*1*])

lemma *K2-centre-non-zero*: *vector* [*0*,*0*,*1*] ≠ (*0* :: *real^3*)
by (*unfold vector-def*) (*simp add*: *vec-eq-iff forall-3*)

lemma *K2-centre-in-K2*: *K2-centre* ∈ *K2*
proof −
from *K2-centre-non-zero* **and** *proj2-rep-abs2*
obtain *k* **where** *k* ≠ *0* **and** *proj2-rep K2-centre* = *k* $*_R$ *vector* [*0*,*0*,*1*]
by (*unfold K2-centre-def*) *auto*
from ‹*k* ≠ *0*› **have** *0* < k^2 **by** *simp*
with ‹*proj2-rep K2-centre* = *k* $*_R$ *vector* [*0*,*0*,*1*]›
show *K2-centre* ∈ *K2*
unfolding *K2-def*
and *conic-sgn-def*
and *M-def*
and *matrix-vector-mult-def*
and *inner-vec-def*
and *vector-def*

by (*simp add*: *vec-eq-iff sum-3 power2-eq-square*)
qed

lemma *K2-imp-M-neg*:
assumes $v \neq 0$ **and** *proj2-abs* $v \in K2$
shows $v \cdot (M *v\ v) < 0$
using *assms*
by (*simp add*: *K2-abs*)

lemma *M-neg-imp-z-squared-big*:
assumes $v \cdot (M *v\ v) < 0$
shows $(v\$3)^2 > (v\$1)^2 + (v\$2)^2$
using ‹$v \cdot (M *v\ v) < 0$›
unfolding *matrix-vector-mult-def* **and** *M-def* **and** *vector-def*
by (*simp add*: *inner-vec-def sum-3 power2-eq-square*)

lemma *M-neg-imp-z-non-zero*:
assumes $v \cdot (M *v\ v) < 0$
shows $v\$3 \neq 0$
proof –
have $(v\$1)^2 + (v\$2)^2 \geq 0$ **by** *simp*
with *M-neg-imp-z-squared-big* [*of v*] **and** ‹$v \cdot (M *v\ v) < 0$›
have $(v\$3)^2 > 0$ **by** *arith*
thus $v\$3 \neq 0$ **by** *simp*
qed

lemma *M-neg-imp-K2*:
assumes $v \cdot (M *v\ v) < 0$
shows *proj2-abs* $v \in K2$
proof –
from ‹$v \cdot (M *v\ v) < 0$› **have** $v\$3 \neq 0$ **by** (*rule M-neg-imp-z-non-zero*)
hence $v \neq 0$ **by** *auto*
with ‹$v \cdot (M *v\ v) < 0$› **and** *K2-abs* **show** *proj2-abs* $v \in K2$ **by** *simp*
qed

lemma *M-reverse*: $a \cdot (M *v\ b) = b \cdot (M *v\ a)$
unfolding *matrix-vector-mult-def* **and** *M-def* **and** *vector-def*
by (*simp add*: *inner-vec-def sum-3*)

lemma *S-abs*:
assumes $v \neq 0$
shows *proj2-abs* $v \in S \longleftrightarrow v \cdot (M *v\ v) = 0$
proof –
have *proj2-abs* $v \in S \longleftrightarrow$ *conic-sgn* (*proj2-abs* v) $= 0$
unfolding *S-def*
by *simp*
also from ‹$v \neq 0$› **and** *conic-sgn-abs*
have $\ldots \longleftrightarrow$ *sgn* $(v \cdot (M *v\ v)) = 0$ **by** *simp*
finally show *proj2-abs* $v \in S \longleftrightarrow v \cdot (M *v\ v) = 0$ **by** (*simp add*: *sgn-0-0*)

qed

lemma *S-alt-def*: *p* ∈ *S* ⟷ *proj2-rep p* · (*M* ∗*v proj2-rep p*) = *0*
proof −
  have *proj2-rep p* ≠ *0* **by** (*rule proj2-rep-non-zero*)
  hence *proj2-abs* (*proj2-rep p*) ∈ *S* ⟷ *proj2-rep p* · (*M* ∗*v proj2-rep p*) = *0*
    by (*rule S-abs*)
  thus *p* ∈ *S* ⟷ *proj2-rep p* · (*M* ∗*v proj2-rep p*) = *0*
    by (*simp add*: *proj2-abs-rep*)
qed

lemma *incident-polar*:
  proj2-incident p (*polar q*) ⟷ *proj2-rep p* · (*M* ∗*v proj2-rep q*) = *0*
  using *polar-rep-non-zero*
  unfolding *polar-def*
  by (*rule proj2-incident-right-abs*)

lemma *incident-own-polar-in-S*: *proj2-incident p* (*polar p*) ⟷ *p* ∈ *S*
  using *incident-polar* **and** *S-alt-def*
  by *simp*

lemma *incident-polar-swap*:
  assumes *proj2-incident p* (*polar q*)
  shows *proj2-incident q* (*polar p*)
proof −
  from ‹*proj2-incident p* (*polar q*)›
  have *proj2-rep p* · (*M* ∗*v proj2-rep q*) = *0* **by** (*unfold incident-polar*)
  hence *proj2-rep q* · (*M* ∗*v proj2-rep p*) = *0* **by** (*simp add*: *M-reverse*)
  thus *proj2-incident q* (*polar p*) **by** (*unfold incident-polar*)
qed

lemma *incident-pole-polar*:
  assumes *proj2-incident p l*
  shows *proj2-incident* (*pole l*) (*polar p*)
proof −
  from ‹*proj2-incident p l*›
  have *proj2-incident p* (*polar* (*pole l*)) **by** (*subst polar-pole*)
  thus *proj2-incident* (*pole l*) (*polar p*) **by** (*rule incident-polar-swap*)
qed

definition *z-zero* :: *proj2-line* **where**
  z-zero ≜ *proj2-line-abs* (*vector* [*0*,*0*,*1*])

lemma *z-zero*:
  assumes (*proj2-rep p*)$*3* = *0*
  shows *proj2-incident p z-zero*
proof −
  from *K2-centre-non-zero* **and** *proj2-line-rep-abs*
  obtain *k* **where** *proj2-line-rep z-zero* = *k* $*_R$ *vector* [*0*,*0*,*1*]

by (*unfold z-zero-def*) *auto*
with ⟨(*proj2-rep p*)\$*3* = *0*⟩
show *proj2-incident p z-zero*
unfolding *proj2-incident-def* **and** *inner-vec-def* **and** *vector-def*
by (*simp add*: *sum-3*)
qed

lemma *z-zero-conic-sgn-1*:
assumes *proj2-incident p z-zero*
shows *conic-sgn p* = *1*
proof −
let *?v* = *proj2-rep p*
have (*vector* [*0*,*0*,*1*] :: *real^3*) ≠ *0*
unfolding *vector-def*
by (*simp add*: *vec-eq-iff*)
with ⟨*proj2-incident p z-zero*⟩
have *?v* · *vector* [*0*,*0*,*1*] = *0*
unfolding *z-zero-def*
by (*simp add*: *proj2-incident-right-abs*)
hence *?v*\$*3* = *0*
unfolding *inner-vec-def* **and** *vector-def*
by (*simp add*: *sum-3*)
hence $?v \cdot (M *v\ ?v) = (?v\$1)^2 + (?v\$2)^2$
unfolding *inner-vec-def*
and *power2-eq-square*
and *matrix-vector-mult-def*
and *M-def*
and *vector-def*
and *sum-3*
by *simp*

have *?v* ≠ *0* **by** (*rule proj2-rep-non-zero*)
with ⟨*?v*\$*3* = *0*⟩ **have** *?v*\$*1* ≠ *0* ∨ *?v*\$*2* ≠ *0* **by** (*simp add*: *vec-eq-iff forall-3*)
hence $(?v\$1)^2 > 0 \lor (?v\$2)^2 > 0$ **by** *simp*
with *add-sign-intros* [*of* $(?v\$1)^2$ $(?v\$2)^2$]
have $(?v\$1)^2 + (?v\$2)^2 > 0$ **by** *auto*
with ⟨$?v \cdot (M *v\ ?v) = (?v\$1)^2 + (?v\$2)^2$⟩
have $?v \cdot (M *v\ ?v) > 0$ **by** *simp*
thus *conic-sgn p* = *1*
unfolding *conic-sgn-def*
by *simp*
qed

lemma *conic-sgn-not-1-z-non-zero*:
assumes *conic-sgn p* ≠ *1*
shows *z-non-zero p*
proof −
from ⟨*conic-sgn p* ≠ *1*⟩
have ¬ *proj2-incident p z-zero* **by** (*auto simp add*: *z-zero-conic-sgn-1*)

```
  thus z-non-zero p by (auto simp add: z-zero)
qed

lemma z-zero-not-in-S:
  assumes proj2-incident p z-zero
  shows p ∉ S
proof −
  from ‹proj2-incident p z-zero› have conic-sgn p = 1
    by (rule z-zero-conic-sgn-1)
  thus p ∉ S
    unfolding S-def
    by simp
qed

lemma line-incident-point-not-in-S: ∃ p. p ∉ S ∧ proj2-incident p l
proof −
  let ?p = proj2-intersection l z-zero
  have proj2-incident ?p l and proj2-incident ?p z-zero
    by (rule proj2-intersection-incident)+
  from ‹proj2-incident ?p z-zero› have ?p ∉ S by (rule z-zero-not-in-S)
  with ‹proj2-incident ?p l›
  show ∃ p. p ∉ S ∧ proj2-incident p l by auto
qed

lemma apply-cltn2-abs-abs-in-S:
  assumes v ≠ 0 and invertible J
  shows apply-cltn2 (proj2-abs v) (cltn2-abs J) ∈ S
  ⟷ v · (J ** M ** transpose J *v v) = 0
proof −
  from ‹v ≠ 0› and ‹invertible J›
  have v v* J ≠ 0 by (rule non-zero-mult-invertible-non-zero)

  from ‹v ≠ 0› and ‹invertible J›
  have apply-cltn2 (proj2-abs v) (cltn2-abs J) = proj2-abs (v v* J)
    by (rule apply-cltn2-abs)
  also from ‹v v* J ≠ 0›
  have ... ∈ S ⟷ (v v* J) · (M *v (v v* J)) = 0 by (rule S-abs)
  finally show apply-cltn2 (proj2-abs v) (cltn2-abs J) ∈ S
    ⟷ v · (J ** M ** transpose J *v v) = 0
    by (simp add: dot-lmul-matrix matrix-vector-mul-assoc [symmetric])
qed

lemma apply-cltn2-right-abs-in-S:
  assumes invertible J
  shows apply-cltn2 p (cltn2-abs J) ∈ S
  ⟷ (proj2-rep p) · (J ** M ** transpose J *v (proj2-rep p)) = 0
proof −
  have proj2-rep p ≠ 0 by (rule proj2-rep-non-zero)
  with ‹invertible J›
```

have *apply-cltn2* (*proj2-abs* (*proj2-rep p*)) (*cltn2-abs J*) ∈ *S*
⟷ *proj2-rep p* · (*J* ** *M* ** *transpose J* *v *proj2-rep p*) = *0*
by (*simp add*: *apply-cltn2-abs-abs-in-S*)
thus *apply-cltn2 p* (*cltn2-abs J*) ∈ *S*
⟷ *proj2-rep p* · (*J* ** *M* ** *transpose J* *v *proj2-rep p*) = *0*
by (*simp add*: *proj2-abs-rep*)
qed

lemma *apply-cltn2-abs-in-S*:
assumes *v* ≠ *0*
shows *apply-cltn2* (*proj2-abs v*) *C* ∈ *S*
⟷ *v* · (*cltn2-rep C* ** *M* ** *transpose* (*cltn2-rep C*) *v *v*) = *0*
proof −
have *invertible* (*cltn2-rep C*) **by** (*rule cltn2-rep-invertible*)
with ‹*v* ≠ *0*›
have *apply-cltn2* (*proj2-abs v*) (*cltn2-abs* (*cltn2-rep C*)) ∈ *S*
⟷ *v* · (*cltn2-rep C* ** *M* ** *transpose* (*cltn2-rep C*) *v *v*) = *0*
by (*rule apply-cltn2-abs-abs-in-S*)
thus *apply-cltn2* (*proj2-abs v*) *C* ∈ *S*
⟷ *v* · (*cltn2-rep C* ** *M* ** *transpose* (*cltn2-rep C*) *v *v*) = *0*
by (*simp add*: *cltn2-abs-rep*)
qed

lemma *apply-cltn2-in-S*:
apply-cltn2 p C ∈ *S*
⟷ *proj2-rep p* · (*cltn2-rep C* ** *M* ** *transpose* (*cltn2-rep C*) *v *proj2-rep p*)
= *0*
proof −
have *proj2-rep p* ≠ *0* **by** (*rule proj2-rep-non-zero*)
hence *apply-cltn2* (*proj2-abs* (*proj2-rep p*)) *C* ∈ *S*
⟷ *proj2-rep p* · (*cltn2-rep C* ** *M* ** *transpose* (*cltn2-rep C*) *v *proj2-rep p*)
= *0*
by (*rule apply-cltn2-abs-in-S*)
thus *apply-cltn2 p C* ∈ *S*
⟷ *proj2-rep p* · (*cltn2-rep C* ** *M* ** *transpose* (*cltn2-rep C*) *v *proj2-rep p*)
= *0*
by (*simp add*: *proj2-abs-rep*)
qed

lemma *norm-M*: (*vector2-append1 v*) · (*M* *v *vector2-append1 v*) = (*norm v*)^2 −
1
proof −
have (*norm v*)^2 = (*v*$*1*)^2 + (*v*$*2*)^2
unfolding *norm-vec-def*
and *L2-set-def*
by (*simp add*: *sum-2*)
thus (*vector2-append1 v*) · (*M* *v *vector2-append1 v*) = (*norm v*)^2 − *1*
unfolding *vector2-append1-def*
and *inner-vec-def*

    and *matrix-vector-mult-def*
    and *vector-def*
    and *M-def*
    and *power2-norm-eq-inner*
  by (*simp add*: *sum-3 power2-eq-square*)
qed

8.2 Some specific points and lines of the projective plane

definition *east* = *proj2-abs* (*vector* [*1*,*0*,*1*])
definition *west* = *proj2-abs* (*vector* [−*1*,*0*,*1*])
definition *north* = *proj2-abs* (*vector* [*0*,*1*,*1*])
definition *south* = *proj2-abs* (*vector* [*0*,−*1*,*1*])
definition *far-north* = *proj2-abs* (*vector* [*0*,*1*,*0*])

lemmas *compass-defs* = *east-def west-def north-def south-def*

lemma *compass-non-zero*:
  shows *vector* [*1*,*0*,*1*] ≠ (*0* :: *real^3*)
  and *vector* [−*1*,*0*,*1*] ≠ (*0* :: *real^3*)
  and *vector* [*0*,*1*,*1*] ≠ (*0* :: *real^3*)
  and *vector* [*0*,−*1*,*1*] ≠ (*0* :: *real^3*)
  and *vector* [*0*,*1*,*0*] ≠ (*0* :: *real^3*)
  and *vector* [*1*,*0*,*0*] ≠ (*0* :: *real^3*)
  unfolding *vector-def*
  by (*simp-all add*: *vec-eq-iff forall-3*)

lemma *east-west-distinct*: *east* ≠ *west*
proof
  assume *east* = *west*
  with *compass-non-zero*
    and *proj2-abs-abs-mult* [*of vector* [*1*,*0*,*1*] *vector* [−*1*,*0*,*1*]]
  obtain *k* **where** (*vector* [*1*,*0*,*1*] :: *real^3*) = $k *_R$ *vector* [−*1*,*0*,*1*]
    unfolding *compass-defs*
    by *auto*
  thus *False*
    unfolding *vector-def*
    by (*auto simp add*: *vec-eq-iff forall-3*)
qed

lemma *north-south-distinct*: *north* ≠ *south*
proof
  assume *north* = *south*
  with *compass-non-zero*
    and *proj2-abs-abs-mult* [*of vector* [*0*,*1*,*1*] *vector* [*0*,−*1*,*1*]]
  obtain *k* **where** (*vector* [*0*,*1*,*1*] :: *real^3*) = $k *_R$ *vector* [*0*,−*1*,*1*]
    unfolding *compass-defs*
    by *auto*
  thus *False*

  unfolding *vector-def*
  by (*auto simp add*: *vec-eq-iff forall-3*)
qed

lemma *north-not-east-or-west*: $north \notin \{east,\ west\}$
proof
 assume $north \in \{east,\ west\}$
 hence $east = north \vee west = north$ **by** *auto*
 with *compass-non-zero*
  and *proj2-abs-abs-mult* [*of* - *vector* [*0*,*1*,*1*]]
 obtain k **where** ($vector\ [1,0,1] :: real\hat{}3) = k *_R vector\ [0,1,1]$
  $\vee$ ($vector\ [-1,0,1] :: real\hat{}3) = k *_R vector\ [0,1,1]$
  unfolding *compass-defs*
  by *auto*
 thus *False*
  unfolding *vector-def*
  by (*simp add*: *vec-eq-iff forall-3*)
qed

lemma *compass-in-S*:
 shows $east \in S$ **and** $west \in S$ **and** $north \in S$ **and** $south \in S$
 using *compass-non-zero* **and** *S-abs*
 unfolding *compass-defs*
  and *M-def*
  and *inner-vec-def*
  and *matrix-vector-mult-def*
  and *vector-def*
 by (*simp-all add*: *sum-3*)

lemma *east-west-tangents*:
 shows *polar east* = *proj2-line-abs* (*vector* [-1,*0*,*1*])
 and *polar west* = *proj2-line-abs* (*vector* [*1*,*0*,*1*])
proof –
 have $M *v\ vector\ [1,0,1] = (-1) *_R vector\ [-1,0,1]$
  and $M *v\ vector\ [-1,0,1] = (-1) *_R vector\ [1,0,1]$
  unfolding *M-def* **and** *matrix-vector-mult-def* **and** *vector-def*
  by (*simp-all add*: *vec-eq-iff sum-3*)
 with *compass-non-zero* **and** *polar-abs*
 have *polar east* = *proj2-line-abs* ((-1) $*_R$ *vector* [-1,*0*,*1*])
  and *polar west* = *proj2-line-abs* ((-1) $*_R$ *vector* [*1*,*0*,*1*])
  unfolding *compass-defs*
  by *simp-all*
 with *proj2-line-abs-mult* [*of* -1]
 show *polar east* = *proj2-line-abs* (*vector* [-1,*0*,*1*])
  and *polar west* = *proj2-line-abs* (*vector* [*1*,*0*,*1*])
  by *simp-all*
qed

lemma *east-west-tangents-distinct*: *polar east* $\neq$ *polar west*

proof
  assume *polar east* = *polar west*
  hence *east* = *west* **by** (*rule polar-inj*)
  with *east-west-distinct* **show** *False* **..**
qed

lemma *east-west-tangents-incident-far-north*:
  shows *proj2-incident far-north* (*polar east*)
  and *proj2-incident far-north* (*polar west*)
  using *compass-non-zero* **and** *proj2-incident-abs*
  unfolding *far-north-def* **and** *east-west-tangents* **and** *inner-vec-def*
  by (*simp-all add*: *sum-3 vector-3*)

lemma *east-west-tangents-far-north*:
  proj2-intersection (*polar east*) (*polar west*) = *far-north*
  using *east-west-tangents-distinct* **and** *east-west-tangents-incident-far-north*
  by (*rule proj2-intersection-unique* [*symmetric*])

instantiation *proj2* :: *zero*
begin
definition *proj2-zero-def*: *0* = *proj2-pt 0*
instance ..
end

definition *equator* $\triangleq$ *proj2-line-abs* (*vector* [*0*,*1*,*0*])
definition *meridian* $\triangleq$ *proj2-line-abs* (*vector* [*1*,*0*,*0*])

lemma *equator-meridian-distinct*: *equator* $\neq$ *meridian*
proof
  assume *equator* = *meridian*
  with *compass-non-zero*
    and *proj2-line-abs-abs-mult* [*of vector* [*0*,*1*,*0*] *vector* [*1*,*0*,*0*]]
  obtain *k* **where** (*vector* [*0*,*1*,*0*] :: *real^3*) = $k *_R$ *vector* [*1*,*0*,*0*]
    by (*unfold equator-def meridian-def*) *auto*
  thus *False* **by** (*unfold vector-def*) (*auto simp add*: *vec-eq-iff forall-3*)
qed

lemma *east-west-on-equator*:
  shows *proj2-incident east equator* **and** *proj2-incident west equator*
  unfolding *east-def* **and** *west-def* **and** *equator-def*
  using *compass-non-zero*
  by (*simp-all add*: *proj2-incident-abs inner-vec-def vector-def sum-3*)

lemma *north-far-north-distinct*: *north* $\neq$ *far-north*
proof
  assume *north* = *far-north*
  with *compass-non-zero*
    and *proj2-abs-abs-mult* [*of vector* [*0*,*1*,*1*] *vector* [*0*,*1*,*0*]]
  obtain *k* **where** (*vector* [*0*,*1*,*1*] :: *real^3*) = $k *_R$ *vector* [*0*,*1*,*0*]

    by (*unfold north-def far-north-def*) *auto*
  thus *False*
    unfolding *vector-def*
    by (*auto simp add*: *vec-eq-iff forall-3*)
qed

lemma *north-south-far-north-on-meridian*:
  shows *proj2-incident north meridian* **and** *proj2-incident south meridian*
  and *proj2-incident far-north meridian*
  unfolding *compass-defs* **and** *far-north-def* **and** *meridian-def*
  using *compass-non-zero*
  by (*simp-all add*: *proj2-incident-abs inner-vec-def vector-def sum-3*)

lemma *K2-centre-on-equator-meridian*:
  shows *proj2-incident K2-centre equator*
  and *proj2-incident K2-centre meridian*
  unfolding *K2-centre-def* **and** *equator-def* **and** *meridian-def*
  using *K2-centre-non-zero* **and** *compass-non-zero*
  by (*simp-all add*: *proj2-incident-abs inner-vec-def vector-def sum-3*)

lemma *on-equator-meridian-is-K2-centre*:
  assumes *proj2-incident a equator* **and** *proj2-incident a meridian*
  shows $a = K2\text{-}centre$
  using *assms* **and** *K2-centre-on-equator-meridian* **and** *equator-meridian-distinct*
    and *proj2-incident-unique*
  by *auto*

definition *rep-equator-reflect* $\triangleq$ *vector* [
  vector [*1*, *0*,*0*],
  vector [*0*,−*1*,*0*],
  vector [*0*, *0*,*1*]] :: *real^3^3*
definition *rep-meridian-reflect* $\triangleq$ *vector* [
  vector [−*1*,*0*,*0*],
  vector [*0*,*1*,*0*],
  vector [*0*,*0*,*1*]] :: *real^3^3*
definition *equator-reflect* $\triangleq$ *cltn2-abs rep-equator-reflect*
definition *meridian-reflect* $\triangleq$ *cltn2-abs rep-meridian-reflect*

lemmas *compass-reflect-defs* = *equator-reflect-def meridian-reflect-def*
  rep-equator-reflect-def rep-meridian-reflect-def

lemma *compass-reflect-self-inverse*:
  shows *rep-equator-reflect* ∗∗ *rep-equator-reflect* = *mat 1*
  and *rep-meridian-reflect* ∗∗ *rep-meridian-reflect* = *mat 1*
  unfolding *compass-reflect-defs matrix-matrix-mult-def mat-def*
  by (*simp-all add*: *vec-eq-iff forall-3 sum-3 vector-3*)

lemma *compass-reflect-invertible*:
  shows *invertible rep-equator-reflect* **and** *invertible rep-meridian-reflect*

unfolding *invertible-def*
using *compass-reflect-self-inverse*
by *auto*

lemma *compass-reflect-compass*:
shows *apply-cltn2 east meridian-reflect* = *west*
and *apply-cltn2 west meridian-reflect* = *east*
and *apply-cltn2 north meridian-reflect* = *north*
and *apply-cltn2 south meridian-reflect* = *south*
and *apply-cltn2 K2-centre meridian-reflect* = *K2-centre*
and *apply-cltn2 east equator-reflect* = *east*
and *apply-cltn2 west equator-reflect* = *west*
and *apply-cltn2 north equator-reflect* = *south*
and *apply-cltn2 south equator-reflect* = *north*
and *apply-cltn2 K2-centre equator-reflect* = *K2-centre*
proof −
have (*vector* [*1*,*0*,*1*] :: *real^3*) *v*∗ *rep-meridian-reflect* = *vector* [−*1*,*0*,*1*]
and (*vector* [−*1*,*0*,*1*] :: *real^3*) *v*∗ *rep-meridian-reflect* = *vector* [*1*,*0*,*1*]
and (*vector* [*0*,*1*,*1*] :: *real^3*) *v*∗ *rep-meridian-reflect* = *vector* [*0*,*1*,*1*]
and (*vector* [*0*,−*1*,*1*] :: *real^3*) *v*∗ *rep-meridian-reflect* = *vector* [*0*,−*1*,*1*]
and (*vector* [*0*,*0*,*1*] :: *real^3*) *v*∗ *rep-meridian-reflect* = *vector* [*0*,*0*,*1*]
and (*vector* [*1*,*0*,*1*] :: *real^3*) *v*∗ *rep-equator-reflect* = *vector* [*1*,*0*,*1*]
and (*vector* [−*1*,*0*,*1*] :: *real^3*) *v*∗ *rep-equator-reflect* = *vector* [−*1*,*0*,*1*]
and (*vector* [*0*,*1*,*1*] :: *real^3*) *v*∗ *rep-equator-reflect* = *vector* [*0*,−*1*,*1*]
and (*vector* [*0*,−*1*,*1*] :: *real^3*) *v*∗ *rep-equator-reflect* = *vector* [*0*,*1*,*1*]
and (*vector* [*0*,*0*,*1*] :: *real^3*) *v*∗ *rep-equator-reflect* = *vector* [*0*,*0*,*1*]
unfolding *rep-meridian-reflect-def* **and** *rep-equator-reflect-def*
and *vector-matrix-mult-def*
by (*simp-all add*: *vec-eq-iff forall-3 vector-3 sum-3*)
with *compass-reflect-invertible* **and** *compass-non-zero* **and** *K2-centre-non-zero*
show *apply-cltn2 east meridian-reflect* = *west*
and *apply-cltn2 west meridian-reflect* = *east*
and *apply-cltn2 north meridian-reflect* = *north*
and *apply-cltn2 south meridian-reflect* = *south*
and *apply-cltn2 K2-centre meridian-reflect* = *K2-centre*
and *apply-cltn2 east equator-reflect* = *east*
and *apply-cltn2 west equator-reflect* = *west*
and *apply-cltn2 north equator-reflect* = *south*
and *apply-cltn2 south equator-reflect* = *north*
and *apply-cltn2 K2-centre equator-reflect* = *K2-centre*
unfolding *compass-defs* **and** *K2-centre-def*
and *meridian-reflect-def* **and** *equator-reflect-def*
by (*simp-all add*: *apply-cltn2-abs*)
qed

lemma *on-equator-rep*:
assumes *z-non-zero a* **and** *proj2-incident a equator*
shows ∃ *x*. *a* = *proj2-abs* (*vector* [*x*,*0*,*1*])
proof −

let *?ra* = *proj2-rep a*
let *?ca1* = *cart2-append1 a*
let *?x* = *?ca1$1*
from *compass-non-zero* **and** ‹*proj2-incident a equator*›
have *?ra* · *vector* [*0*,*1*,*0*] = *0*
by (*unfold equator-def*) (*simp add*: *proj2-incident-right-abs*)
hence *?ra$2* = *0* **by** (*unfold inner-vec-def vector-def*) (*simp add*: *sum-3*)
hence *?ca1$2* = *0* **by** (*unfold cart2-append1-def*) *simp*
moreover
from ‹*z-non-zero a*› **have** *?ca1$3* = *1* **by** (*rule cart2-append1-z*)
ultimately
have *?ca1* = *vector* [*?x*,*0*,*1*]
by (*unfold vector-def*) (*simp add*: *vec-eq-iff forall-3*)
with ‹*z-non-zero a*›
have *proj2-abs* (*vector* [*?x*,*0*,*1*]) = *a* **by** (*simp add*: *proj2-abs-cart2-append1*)
thus ∃ *x*. *a* = *proj2-abs* (*vector* [*x*,*0*,*1*]) **by** (*simp add*: *exI* [*of - ?x*])
qed

lemma *on-meridian-rep*:
assumes *z-non-zero a* **and** *proj2-incident a meridian*
shows ∃ *y*. *a* = *proj2-abs* (*vector* [*0*,*y*,*1*])
proof −
let *?ra* = *proj2-rep a*
let *?ca1* = *cart2-append1 a*
let *?y* = *?ca1$2*
from *compass-non-zero* **and** ‹*proj2-incident a meridian*›
have *?ra* · *vector* [*1*,*0*,*0*] = *0*
by (*unfold meridian-def*) (*simp add*: *proj2-incident-right-abs*)
hence *?ra$1* = *0* **by** (*unfold inner-vec-def vector-def*) (*simp add*: *sum-3*)
hence *?ca1$1* = *0* **by** (*unfold cart2-append1-def*) *simp*
moreover
from ‹*z-non-zero a*› **have** *?ca1$3* = *1* **by** (*rule cart2-append1-z*)
ultimately
have *?ca1* = *vector* [*0*,*?y*,*1*]
by (*unfold vector-def*) (*simp add*: *vec-eq-iff forall-3*)
with ‹*z-non-zero a*›
have *proj2-abs* (*vector* [*0*,*?y*,*1*]) = *a* **by** (*simp add*: *proj2-abs-cart2-append1*)
thus ∃ *y*. *a* = *proj2-abs* (*vector* [*0*,*y*,*1*]) **by** (*simp add*: *exI* [*of - ?y*])
qed

8.3 Definition of the Klein–Beltrami model of the hyperbolic plane

abbreviation *hyp2* == *K2*

typedef *hyp2* = *K2*
using *K2-centre-in-K2*
by *auto*

definition *hyp2-rep* :: *hyp2* $\Rightarrow$ *real*ˆ*2* **where**
hyp2-rep p $\triangleq$ *cart2-pt* (*Rep-hyp2 p*)

definition *hyp2-abs* :: *real*ˆ*2* $\Rightarrow$ *hyp2* **where**
hyp2-abs v = *Abs-hyp2* (*proj2-pt v*)

lemma *norm-lt-1-iff-in-hyp2*:
shows *norm v* < *1* $\longleftrightarrow$ *proj2-pt v* $\in$ *hyp2*
proof −
let *?v'* = *vector2-append1 v*
have *?v'* $\neq$ *0* **by** (*rule vector2-append1-non-zero*)

from *real-less-rsqrt* [*of norm v 1*]
and *abs-square-less-1* [*of norm v*]
have *norm v* < *1* $\longleftrightarrow$ (*norm v*)^2 < *1* **by** *auto*
hence *norm v* < *1* $\longleftrightarrow$ *?v'* · (*M* ∗*v* *?v'*) < *0* **by** (*simp add*: *norm-M*)
with ‹*?v'* $\neq$ *0*› **have** *norm v* < *1* $\longleftrightarrow$ *proj2-abs ?v'* $\in$ *K2* **by** (*subst K2-abs*)
thus *norm v* < *1* $\longleftrightarrow$ *proj2-pt v* $\in$ *hyp2* **by** (*unfold proj2-pt-def*)
qed

lemma *norm-eq-1-iff-in-S*:
shows *norm v* = *1* $\longleftrightarrow$ *proj2-pt v* $\in$ *S*
proof −
let *?v'* = *vector2-append1 v*
have *?v'* $\neq$ *0* **by** (*rule vector2-append1-non-zero*)

from *real-sqrt-unique* [*of norm v 1*]
have *norm v* = *1* $\longleftrightarrow$ (*norm v*)^2 = *1* **by** *auto*
hence *norm v* = *1* $\longleftrightarrow$ *?v'* · (*M* ∗*v* *?v'*) = *0* **by** (*simp add*: *norm-M*)
with ‹*?v'* $\neq$ *0*› **have** *norm v* = *1* $\longleftrightarrow$ *proj2-abs ?v'* $\in$ *S* **by** (*subst S-abs*)
thus *norm v* = *1* $\longleftrightarrow$ *proj2-pt v* $\in$ *S* **by** (*unfold proj2-pt-def*)
qed

lemma *norm-le-1-iff-in-hyp2-S*:
norm v $\leq$ *1* $\longleftrightarrow$ *proj2-pt v* $\in$ *hyp2* $\cup$ *S*
using *norm-lt-1-iff-in-hyp2* [*of v*] **and** *norm-eq-1-iff-in-S* [*of v*]
by *auto*

lemma *proj2-pt-hyp2-rep*: *proj2-pt* (*hyp2-rep p*) = *Rep-hyp2 p*
proof −
let *?p'* = *Rep-hyp2 p*
let *?v* = *proj2-rep ?p'*
have *?v* $\neq$ *0* **by** (*rule proj2-rep-non-zero*)

have *proj2-abs ?v* = *?p'* **by** (*rule proj2-abs-rep*)

have *?p'* $\in$ *hyp2* **by** (*rule Rep-hyp2*)
with ‹*?v* $\neq$ *0*› **and** ‹*proj2-abs ?v* = *?p'*›
have *?v* · (*M* ∗*v* *?v*) < *0* **by** (*simp add*: *K2-imp-M-neg*)

hence *?v$3 ≠ 0* **by** (*rule M-neg-imp-z-non-zero*)
hence *proj2-pt* (*cart2-pt ?p′*) = *?p′* **by** (*rule proj2-cart2*)
thus *proj2-pt* (*hyp2-rep p*) = *?p′* **by** (*unfold hyp2-rep-def*)
qed

lemma *hyp2-rep-abs*:
assumes *norm v < 1*
shows *hyp2-rep* (*hyp2-abs v*) = *v*
proof −
from ‹*norm v < 1*›
have *proj2-pt v ∈ hyp2* **by** (*simp add*: *norm-lt-1-iff-in-hyp2*)
hence *Rep-hyp2* (*Abs-hyp2* (*proj2-pt v*)) = *proj2-pt v*
by (*simp add*: *Abs-hyp2-inverse*)
hence *hyp2-rep* (*hyp2-abs v*) = *cart2-pt* (*proj2-pt v*)
by (*unfold hyp2-rep-def hyp2-abs-def*) *simp*
thus *hyp2-rep* (*hyp2-abs v*) = *v* **by** (*simp add*: *cart2-proj2*)
qed

lemma *hyp2-abs-rep*: *hyp2-abs* (*hyp2-rep p*) = *p*
by (*unfold hyp2-abs-def*) (*simp add*: *proj2-pt-hyp2-rep Rep-hyp2-inverse*)

lemma *norm-hyp2-rep-lt-1*: *norm* (*hyp2-rep p*) < *1*
proof −
have *proj2-pt* (*hyp2-rep p*) = *Rep-hyp2 p* **by** (*rule proj2-pt-hyp2-rep*)
hence *proj2-pt* (*hyp2-rep p*) ∈ *hyp2* **by** (*simp add*: *Rep-hyp2*)
thus *norm* (*hyp2-rep p*) < *1* **by** (*simp add*: *norm-lt-1-iff-in-hyp2*)
qed

lemma *hyp2-S-z-non-zero*:
assumes *p ∈ hyp2 ∪ S*
shows *z-non-zero p*
proof −
from ‹*p ∈ hyp2 ∪ S*›
have *conic-sgn p ≤ 0* **by** (*unfold K2-def S-def*) *auto*
hence *conic-sgn p ≠ 1* **by** *simp*
thus *z-non-zero p* **by** (*rule conic-sgn-not-1-z-non-zero*)
qed

lemma *hyp2-S-not-equal*:
assumes *a ∈ hyp2* **and** *p ∈ S*
shows *a ≠ p*
using *assms* **and** *S-K2-empty*
by *auto*

lemma *hyp2-S-cart2-inj*:
assumes *p ∈ hyp2 ∪ S* **and** *q ∈ hyp2 ∪ S* **and** *cart2-pt p = cart2-pt q*
shows *p = q*
proof −
from ‹*p ∈ hyp2 ∪ S*› **and** ‹*q ∈ hyp2 ∪ S*›

have *z-non-zero p* **and** *z-non-zero q* **by** (*simp-all add*: *hyp2-S-z-non-zero*)
hence *proj2-pt* (*cart2-pt p*) = *p* **and** *proj2-pt* (*cart2-pt q*) = *q*
by (*simp-all add*: *proj2-cart2*)

from ‹*cart2-pt p* = *cart2-pt q*›
have *proj2-pt* (*cart2-pt p*) = *proj2-pt* (*cart2-pt q*) **by** *simp*
with ‹*proj2-pt* (*cart2-pt p*) = *p*› [*symmetric*] **and** ‹*proj2-pt* (*cart2-pt q*) = *q*›
show *p* = *q* **by** *simp*
qed

lemma *on-equator-in-hyp2-rep*:
assumes *a* ∈ *hyp2* **and** *proj2-incident a equator*
shows ∃ *x*. |*x*| < *1* ∧ *a* = *proj2-abs* (*vector* [*x*,*0*,*1*])
proof −
from ‹*a* ∈ *hyp2*› **have** *z-non-zero a* **by** (*simp add*: *hyp2-S-z-non-zero*)
with ‹*proj2-incident a equator*› **and** *on-equator-rep*
obtain *x* **where** *a* = *proj2-abs* (*vector* [*x*,*0*,*1*]) (**is** *a* = *proj2-abs ?v*)
by *auto*

have *?v* ≠ *0* **by** (*simp add*: *vec-eq-iff forall-3 vector-3*)
with ‹*a* ∈ *hyp2*› **and** ‹*a* = *proj2-abs ?v*›
have *?v* · (*M* ∗*v ?v*) < *0* **by** (*simp add*: *K2-abs*)
hence x^2 < *1*
unfolding *M-def matrix-vector-mult-def inner-vec-def*
by (*simp add*: *sum-3 vector-3 power2-eq-square*)
with *real-sqrt-abs* [*of x*] **and** *real-sqrt-less-iff* [*of* x^2 *1*]
have |*x*| < *1* **by** *simp*
with ‹*a* = *proj2-abs ?v*›
show ∃ *x*. |*x*| < *1* ∧ *a* = *proj2-abs* (*vector* [*x*,*0*,*1*])
by (*simp add*: *exI* [*of - x*])
qed

lemma *on-meridian-in-hyp2-rep*:
assumes *a* ∈ *hyp2* **and** *proj2-incident a meridian*
shows ∃ *y*. |*y*| < *1* ∧ *a* = *proj2-abs* (*vector* [*0*,*y*,*1*])
proof −
from ‹*a* ∈ *hyp2*› **have** *z-non-zero a* **by** (*simp add*: *hyp2-S-z-non-zero*)
with ‹*proj2-incident a meridian*› **and** *on-meridian-rep*
obtain *y* **where** *a* = *proj2-abs* (*vector* [*0*,*y*,*1*]) (**is** *a* = *proj2-abs ?v*)
by *auto*

have *?v* ≠ *0* **by** (*simp add*: *vec-eq-iff forall-3 vector-3*)
with ‹*a* ∈ *hyp2*› **and** ‹*a* = *proj2-abs ?v*›
have *?v* · (*M* ∗*v ?v*) < *0* **by** (*simp add*: *K2-abs*)
hence y^2 < *1*
unfolding *M-def matrix-vector-mult-def inner-vec-def*
by (*simp add*: *sum-3 vector-3 power2-eq-square*)
with *real-sqrt-abs* [*of y*] **and** *real-sqrt-less-iff* [*of* y^2 *1*]
have |*y*| < *1* **by** *simp*

with ‹$a = proj2\text{-}abs\ ?v$›
show $\exists\ y.\ |y| < 1 \wedge a = proj2\text{-}abs\ (vector\ [0,y,1])$
by (*simp add*: *exI* [*of* - *y*])
qed

definition *hyp2-cltn2* :: *hyp2* ⇒ *cltn2* ⇒ *hyp2* **where**
hyp2-cltn2 p A ≜ *Abs-hyp2* (*apply-cltn2* (*Rep-hyp2 p*) *A*)

definition *is-K2-isometry* :: *cltn2* ⇒ *bool* **where**
is-K2-isometry J ≜ (∀ *p*. *apply-cltn2 p J* ∈ *S* ⟷ *p* ∈ *S*)

lemma *cltn2-id-is-K2-isometry*: *is-K2-isometry cltn2-id*
unfolding *is-K2-isometry-def*
by *simp*

lemma *J-M-J-transpose-K2-isometry*:
assumes $k \neq 0$
and *repJ* ∗∗ *M* ∗∗ *transpose repJ* = $k *_R M$ (**is** *?N* = -)
shows *is-K2-isometry* (*cltn2-abs repJ*) (**is** *is-K2-isometry ?J*)
proof −
from ‹$?N = k *_R M$›
have *?N* ∗∗ $((1/k) *_R M)$ = *mat 1*
by (*simp add*: *matrix-scalar-ac* ‹$k \neq 0$› *M-self-inverse*)
with *right-invertible-iff-invertible* [*of repJ*]
have *invertible repJ*
by (*simp add*: *matrix-mul-assoc*
exI [*of* - *M* ∗∗ *transpose repJ* ∗∗ $((1/k) *_R M)$])

have ∀ *t*. *apply-cltn2 t ?J* ∈ *S* ⟷ *t* ∈ *S*
proof
fix *t* :: *proj2*
have *proj2-rep t* · $((k *_R M)$ ∗*v proj2-rep t*)
= *k* ∗ (*proj2-rep t* · (*M* ∗*v proj2-rep t*))
by (*simp add*: *scaleR-matrix-vector-assoc* [*symmetric*] *dot-scaleR-mult*)
with ‹$?N = k *_R M$›
have *proj2-rep t* · (*?N* ∗*v proj2-rep t*)
= *k* ∗ (*proj2-rep t* · (*M* ∗*v proj2-rep t*))
by *simp*
hence *proj2-rep t* · (*?N* ∗*v proj2-rep t*) = *0*
⟷ *k* ∗ (*proj2-rep t* · (*M* ∗*v proj2-rep t*)) = *0*
by *simp*
with ‹$k \neq 0$›
have *proj2-rep t* · (*?N* ∗*v proj2-rep t*) = *0*
⟷ *proj2-rep t* · (*M* ∗*v proj2-rep t*) = *0*
by *simp*
with ‹*invertible repJ*›
have *apply-cltn2 t ?J* ∈ *S* ⟷ *proj2-rep t* · (*M* ∗*v proj2-rep t*) = *0*
by (*simp add*: *apply-cltn2-right-abs-in-S*)
thus *apply-cltn2 t ?J* ∈ *S* ⟷ *t* ∈ *S* **by** (*unfold S-alt-def*)

qed
thus *is-K2-isometry ?J* **by** (*unfold is-K2-isometry-def*)
qed

lemma *equator-reflect-K2-isometry*:
shows *is-K2-isometry equator-reflect*
unfolding *compass-reflect-defs*
by (*rule J-M-J-transpose-K2-isometry* [*of 1*])
(*simp-all add*: *M-def matrix-matrix-mult-def transpose-def*
vec-eq-iff forall-3 sum-3 vector-3)

lemma *meridian-reflect-K2-isometry*:
shows *is-K2-isometry meridian-reflect*
unfolding *compass-reflect-defs*
by (*rule J-M-J-transpose-K2-isometry* [*of 1*])
(*simp-all add*: *M-def matrix-matrix-mult-def transpose-def*
vec-eq-iff forall-3 sum-3 vector-3)

lemma *cltn2-compose-is-K2-isometry*:
assumes *is-K2-isometry H* **and** *is-K2-isometry J*
shows *is-K2-isometry* (*cltn2-compose H J*)
using ‹*is-K2-isometry H*› **and** ‹*is-K2-isometry J*›
unfolding *is-K2-isometry-def*
by (*simp add*: *cltn2.act-act* [*simplified*, *symmetric*])

lemma *cltn2-inverse-is-K2-isometry*:
assumes *is-K2-isometry J*
shows *is-K2-isometry* (*cltn2-inverse J*)
proof −
{ fix *p*
from ‹*is-K2-isometry J*›
have *apply-cltn2 p* (*cltn2-inverse J*) $\in S$
$\longleftrightarrow$ *apply-cltn2* (*apply-cltn2 p* (*cltn2-inverse J*)) *J* $\in S$
unfolding *is-K2-isometry-def*
by *simp*
hence *apply-cltn2 p* (*cltn2-inverse J*) $\in S \longleftrightarrow p \in S$
by (*simp add*: *cltn2.act-inv-act* [*simplified*]) **}**
thus *is-K2-isometry* (*cltn2-inverse J*)
unfolding *is-K2-isometry-def* **..**
qed

interpretation *K2-isometry-subgroup*: *subgroup*
Collect is-K2-isometry
(|*carrier* = *UNIV*, *mult* = *cltn2-compose*, *one* = *cltn2-id*|)
unfolding *subgroup-def*
by (*simp add*:
cltn2-id-is-K2-isometry
cltn2-compose-is-K2-isometry
cltn2-inverse-is-K2-isometry)

interpretation *K2-isometry*: *group*
(|*carrier* = *Collect is-K2-isometry*, *mult* = *cltn2-compose*, *one* = *cltn2-id*|)
using *cltn2.is-group* **and** *K2-isometry-subgroup.subgroup-is-group*
by *simp*

lemma *K2-isometry-inverse-inv* [*simp*]:
assumes *is-K2-isometry J*
shows $inv_{(|carrier\ =\ Collect\ is\text{-}K2\text{-}isometry,\ mult\ =\ cltn2\text{-}compose,\ one\ =\ cltn2\text{-}id|)}$
J
= *cltn2-inverse J*
using *cltn2-left-inverse*
and ‹*is-K2-isometry J*›
and *cltn2-inverse-is-K2-isometry*
and *K2-isometry.inv-equality*
by *simp*

definition *real-hyp2-C* :: [*hyp2*, *hyp2*, *hyp2*, *hyp2*] ⇒ *bool*
(- - $\equiv_K$ - - [*99*,*99*,*99*,*99*] *50*) **where**
p q $\equiv_K$ *r s* ≜
(∃ *A*. *is-K2-isometry A* ∧ *hyp2-cltn2 p A* = *r* ∧ *hyp2-cltn2 q A* = *s*)

definition *real-hyp2-B* :: [*hyp2*, *hyp2*, *hyp2*] ⇒ *bool*
(B_K - - - [*99*,*99*,*99*] *50*) **where**
B_K *p q r* ≜ $B_{\mathbb{R}}$ (*hyp2-rep p*) (*hyp2-rep q*) (*hyp2-rep r*)

8.4 K-isometries map the interior of the conic to itself

lemma *collinear-quadratic*:
assumes $t = i *_R a + r$
shows $t \cdot (M *v\ t) =$
$(a \cdot (M *v\ a)) * i^2 + 2 * (a \cdot (M *v\ r)) * i + r \cdot (M *v\ r)$
proof −
from *M-reverse* **have** $i * (a \cdot (M *v\ r)) = i * (r \cdot (M *v\ a))$ **by** *simp*
with ‹$t = i *_R a + r$›
show $t \cdot (M *v\ t) =$
$(a \cdot (M *v\ a)) * i^2 + 2 * (a \cdot (M *v\ r)) * i + r \cdot (M *v\ r)$
by (*simp add*:
inner-add-left
matrix-vector-right-distrib
inner-add-right
matrix-scaleR-vector-ac
inner-scaleR-right
scaleR-matrix-vector-assoc [*symmetric*]
M-reverse
power2-eq-square
algebra-simps)
qed

lemma *S-quadratic'*:
  assumes $p \neq 0$ **and** $q \neq 0$ **and** *proj2-abs* $p \neq$ *proj2-abs* q
  shows *proj2-abs* $(k *_R p + q) \in S$
  $\longleftrightarrow p \cdot (M *v\ p) * k^2 + p \cdot (M *v\ q) * 2 * k + q \cdot (M *v\ q) = 0$
proof −
  let $?r = k *_R p + q$
  from ‹$p \neq 0$› **and** ‹$q \neq 0$› **and** ‹*proj2-abs* $p \neq$ *proj2-abs* q›
    and *dependent-proj2-abs* [*of p q k 1*]
  have $?r \neq 0$ **by** *auto*
  hence *proj2-abs* $?r \in S \longleftrightarrow ?r \cdot (M *v\ ?r) = 0$ **by** (*rule S-abs*)
  with *collinear-quadratic* [*of ?r k p q*]
  show *proj2-abs* $?r \in S$
    $\longleftrightarrow p \cdot (M *v\ p) * k^2 + p \cdot (M *v\ q) * 2 * k + q \cdot (M *v\ q) = 0$
    by (*simp add*: *dot-lmul-matrix* [*symmetric*] *algebra-simps*)
qed

lemma *S-quadratic*:
  assumes $p \neq q$ **and** $r =$ *proj2-abs* $(k *_R$ *proj2-rep* p + *proj2-rep* $q)$
  shows $r \in S$
  $\longleftrightarrow$ *proj2-rep* $p \cdot (M *v$ *proj2-rep* $p) * k^2$
      + *proj2-rep* $p \cdot (M *v$ *proj2-rep* $q) * 2 * k$
      + *proj2-rep* $q \cdot (M *v$ *proj2-rep* $q)$
    $= 0$
proof −
  let $?u =$ *proj2-rep* p
  let $?v =$ *proj2-rep* q
  let $?w = k *_R ?u + ?v$
  have $?u \neq 0$ **and** $?v \neq 0$ **by** (*rule proj2-rep-non-zero*)+

  from ‹$p \neq q$› **have** *proj2-abs* $?u \neq$ *proj2-abs* $?v$ **by** (*simp add*: *proj2-abs-rep*)
  with ‹$?u \neq 0$› **and** ‹$?v \neq 0$› **and** ‹$r =$ *proj2-abs* $?w$›
  show $r \in S$
    $\longleftrightarrow ?u \cdot (M *v\ ?u) * k^2 + ?u \cdot (M *v\ ?v) * 2 * k + ?v \cdot (M *v\ ?v) = 0$
    by (*simp add*: *S-quadratic'*)
qed

definition *quarter-discrim* :: *real^3* $\Rightarrow$ *real^3* $\Rightarrow$ *real* **where**
  quarter-discrim p q $\triangleq (p \cdot (M *v\ q))^2 - p \cdot (M *v\ p) * (q \cdot (M *v\ q))$

lemma *quarter-discrim-invariant*:
  assumes $t = i *_R a + r$
  shows *quarter-discrim a t* = *quarter-discrim a r*
proof −
  from ‹$t = i *_R a + r$›
  have $a \cdot (M *v\ t) = i * (a \cdot (M *v\ a)) + a \cdot (M *v\ r)$
    by (*simp add*:
      matrix-vector-right-distrib
      inner-add-right
      matrix-scaleR-vector-ac

scaleR-matrix-vector-assoc [*symmetric*])
hence $(a \cdot (M *v\ t))^2 =$
$(a \cdot (M *v\ a))^2 * i^2 +$
$2 * (a \cdot (M *v\ a)) * (a \cdot (M *v\ r)) * i +$
$(a \cdot (M *v\ r))^2$
by (*simp add*: *power2-eq-square algebra-simps*)
moreover from *collinear-quadratic* **and** ‹$t = i *_R a + r$›
have $a \cdot (M *v\ a) * (t \cdot (M *v\ t)) =$
$(a \cdot (M *v\ a))^2 * i^2 +$
$2 * (a \cdot (M *v\ a)) * (a \cdot (M *v\ r)) * i +$
$a \cdot (M *v\ a) * (r \cdot (M *v\ r))$
by (*simp add*: *power2-eq-square algebra-simps*)
ultimately show *quarter-discrim a t* = *quarter-discrim a r*
by (*unfold quarter-discrim-def*, *simp*)
qed

lemma *quarter-discrim-positive*:
assumes $p \neq 0$ **and** $q \neq 0$ **and** *proj2-abs p* $\neq$ *proj2-abs q* (**is** *?pp* $\neq$ *?pq*)
and *proj2-abs p* $\in$ *K2*
shows *quarter-discrim p q* > 0
proof −
let $?i = -q\$3/p\$3$
let $?t = ?i *_R p + q$

from ‹$p \neq 0$› **and** ‹$?pp \in K2$›
have $p \cdot (M *v\ p) < 0$ **by** (*subst K2-abs* [*symmetric*])
hence $p\$3 \neq 0$ **by** (*rule M-neg-imp-z-non-zero*)
hence $?t\$3 = 0$ **by** *simp*
hence $?t \cdot (M *v\ ?t) = (?t\$1)^2 + (?t\$2)^2$
unfolding *matrix-vector-mult-def* **and** *M-def* **and** *vector-def*
by (*simp add*: *inner-vec-def sum-3 power2-eq-square*)

from ‹$p\$3 \neq 0$› **have** $p \neq 0$ **by** *auto*
with ‹$q \neq 0$› **and** ‹$?pp \neq ?pq$› **and** *dependent-proj2-abs* [*of p q ?i 1*]
have $?t \neq 0$ **by** *auto*
with ‹$?t\$3 = 0$› **have** $?t\$1 \neq 0 \lor ?t\$2 \neq 0$ **by** (*simp add*: *vec-eq-iff forall-3*)
hence $(?t\$1)^2 > 0 \lor (?t\$2)^2 > 0$ **by** *simp*
moreover have $(?t\$2)^2 \geq 0$ **and** $(?t\$1)^2 \geq 0$ **by** *simp-all*
ultimately have $(?t\$1)^2 + (?t\$2)^2 > 0$ **by** *arith*
with ‹$?t \cdot (M *v\ ?t) = (?t\$1)^2 + (?t\$2)^2$› **have** $?t \cdot (M *v\ ?t) > 0$ **by** *simp*
with *mult-neg-pos* [*of* $p \cdot (M *v\ p)$] **and** ‹$p \cdot (M *v\ p) < 0$›
have $p \cdot (M *v\ p) * (?t \cdot (M *v\ ?t)) < 0$ **by** *simp*
moreover have $(p \cdot (M *v\ ?t))^2 \geq 0$ **by** *simp*
ultimately
have $(p \cdot (M *v\ ?t))^2 - p \cdot (M *v\ p) * (?t \cdot (M *v\ ?t)) > 0$ **by** *arith*
with *quarter-discrim-invariant* [*of ?t ?i p q*]
show *quarter-discrim p q* > 0 **by** (*unfold quarter-discrim-def*, *simp*)
qed

lemma *quarter-discrim-self-zero*:
  assumes *proj2-abs a = proj2-abs b*
  shows *quarter-discrim a b = 0*
proof *cases*
  assume *b = 0*
  thus *quarter-discrim a b = 0* **by** (*unfold quarter-discrim-def*, *simp*)
next
  assume *b ≠ 0*
  with ⟨*proj2-abs a = proj2-abs b*⟩ **and** *proj2-abs-abs-mult*
  obtain *k* **where** $a = k *_R b$ **by** *auto*
  thus *quarter-discrim a b = 0*
    unfolding *quarter-discrim-def*
    by (*simp add*: *power2-eq-square*
      matrix-scaleR-vector-ac
      scaleR-matrix-vector-assoc [*symmetric*])
qed

definition *S-intersection-coeff1* :: *real^3 ⇒ real^3 ⇒ real* **where**
  S-intersection-coeff1 p q
  ≜ (−*p* · (*M* ∗*v q*) + *sqrt* (*quarter-discrim p q*)) / (*p* · (*M* ∗*v p*))

definition *S-intersection-coeff2* :: *real^3 ⇒ real^3 ⇒ real* **where**
  S-intersection-coeff2 p q
  ≜ (−*p* · (*M* ∗*v q*) − *sqrt* (*quarter-discrim p q*)) / (*p* · (*M* ∗*v p*))

definition *S-intersection1-rep* :: *real^3 ⇒ real^3 ⇒ real^3* **where**
  S-intersection1-rep p q ≜ (*S-intersection-coeff1 p q*) $*_R$ *p* + *q*

definition *S-intersection2-rep* :: *real^3 ⇒ real^3 ⇒ real^3* **where**
  S-intersection2-rep p q ≜ (*S-intersection-coeff2 p q*) $*_R$ *p* + *q*

definition *S-intersection1* :: *real^3 ⇒ real^3 ⇒ proj2* **where**
  S-intersection1 p q ≜ *proj2-abs* (*S-intersection1-rep p q*)

definition *S-intersection2* :: *real^3 ⇒ real^3 ⇒ proj2* **where**
  S-intersection2 p q ≜ *proj2-abs* (*S-intersection2-rep p q*)

lemmas *S-intersection-coeffs-defs* =
  S-intersection-coeff1-def S-intersection-coeff2-def

lemmas *S-intersections-defs* =
  S-intersection1-def S-intersection2-def
  S-intersection1-rep-def S-intersection2-rep-def

lemma *S-intersection-coeffs-distinct*:
  assumes *p ≠ 0* **and** *q ≠ 0* **and** *proj2-abs p ≠ proj2-abs q* (**is** *?pp ≠ ?pq*)
  and *proj2-abs p ∈ K2*
  shows *S-intersection-coeff1 p q ≠ S-intersection-coeff2 p q*
proof −

```
  from ‹p ≠ 0› and ‹?pp ∈ K2›
  have p · (M *v p) < 0 by (subst K2-abs [symmetric])

  from assms have quarter-discrim p q > 0 by (rule quarter-discrim-positive)
  with ‹p · (M *v p) < 0›
  show S-intersection-coeff1 p q ≠ S-intersection-coeff2 p q
    by (unfold S-intersection-coeffs-defs, simp)
qed

lemma S-intersections-distinct:
  assumes p ≠ 0 and q ≠ 0 and proj2-abs p ≠ proj2-abs q (is ?pp ≠ ?pq)
  and proj2-abs p ∈ K2
  shows S-intersection1 p q ≠ S-intersection2 p q
proof−
  from ‹p ≠ 0› and ‹q ≠ 0› and ‹?pp ≠ ?pq› and ‹?pp ∈ K2›
  have S-intersection-coeff1 p q ≠ S-intersection-coeff2 p q
    by (rule S-intersection-coeffs-distinct)
  with ‹p ≠ 0› and ‹q ≠ 0› and ‹?pp ≠ ?pq› and proj2-Col-coeff-unique'
  show S-intersection1 p q ≠ S-intersection2 p q
    by (unfold S-intersections-defs, auto)
qed

lemma S-intersections-in-S:
  assumes p ≠ 0  and q ≠ 0 and proj2-abs p ≠ proj2-abs q (is ?pp ≠ ?pq)
  and proj2-abs p ∈ K2
  shows S-intersection1 p q ∈ S and S-intersection2 p q ∈ S
proof −
  let ?j = S-intersection-coeff1 p q
  let ?k = S-intersection-coeff2 p q
  let ?a = p · (M *v p)
  let ?b = 2 * (p · (M *v q))
  let ?c = q · (M *v q)

  from ‹p ≠ 0› and ‹?pp ∈ K2› have ?a < 0 by (subst K2-abs [symmetric])

  have qd: discrim ?a ?b ?c = 4 * quarter-discrim p q
    unfolding discrim-def quarter-discrim-def
    by (simp add: power2-eq-square)
  with times-divide-times-eq [of
    2 2 sqrt (quarter-discrim p q) − p · (M *v q) ?a]
    and times-divide-times-eq [of
    2 2 −p · (M *v q) − sqrt (quarter-discrim p q) ?a]
    and real-sqrt-mult and real-sqrt-abs [of 2]
  have ?j = (−?b + sqrt (discrim ?a ?b ?c)) / (2 * ?a)
    and ?k = (−?b − sqrt (discrim ?a ?b ?c)) / (2 * ?a)
    by (unfold S-intersection-coeffs-defs, simp-all add: algebra-simps)

  from assms have quarter-discrim p q > 0 by (rule quarter-discrim-positive)
  with qd
```

```
  have discrim (p · (M *v p)) (2 * (p · (M *v q))) (q · (M *v q)) > 0
    by simp
  with ‹?j = (-?b + sqrt (discrim ?a ?b ?c)) / (2 * ?a)›
    and ‹?k = (-?b - sqrt (discrim ?a ?b ?c)) / (2 * ?a)›
    and ‹?a < 0› and discriminant-nonneg [of ?a ?b ?c ?j]
    and discriminant-nonneg [of ?a ?b ?c ?k]
  have p · (M *v p) * ?j^2 + 2 * (p · (M *v q)) * ?j + q · (M *v q) = 0
    and p · (M *v p) * ?k^2 + 2 * (p · (M *v q)) * ?k + q · (M *v q) = 0
    by (unfold S-intersection-coeffs-defs, auto)
  with ‹p ≠ 0› and ‹q ≠ 0› and ‹?pp ≠ ?pq› and S-quadratic'
  show S-intersection1 p q ∈ S and S-intersection2 p q ∈ S
    by (unfold S-intersections-defs, simp-all)
qed

lemma S-intersections-Col:
  assumes p ≠ 0 and q ≠ 0
  shows proj2-Col (proj2-abs p) (proj2-abs q) (S-intersection1 p q)
  (is proj2-Col ?pp ?pq ?pr)
    and proj2-Col (proj2-abs p) (proj2-abs q) (S-intersection2 p q)
  (is proj2-Col ?pp ?pq ?ps)
proof -
  { assume ?pp = ?pq
    hence proj2-Col ?pp ?pq ?pr and proj2-Col ?pp ?pq ?ps
      by (simp-all add: proj2-Col-coincide) }
  moreover
  { assume ?pp ≠ ?pq
    with ‹p ≠ 0› and ‹q ≠ 0› and dependent-proj2-abs [of p q - 1]
    have S-intersection1-rep p q ≠ 0 (is ?r ≠ 0)
      and S-intersection2-rep p q ≠ 0 (is ?s ≠ 0)
      by (unfold S-intersection1-rep-def S-intersection2-rep-def, auto)
    with ‹p ≠ 0› and ‹q ≠ 0›
      and proj2-Col-abs [of p q ?r S-intersection-coeff1 p q 1 -1]
      and proj2-Col-abs [of p q ?s S-intersection-coeff2 p q 1 -1]
    have proj2-Col ?pp ?pq ?pr and proj2-Col ?pp ?pq ?ps
      by (unfold S-intersections-defs, simp-all) }
  ultimately show proj2-Col ?pp ?pq ?pr and proj2-Col ?pp ?pq ?ps by fast+
qed

lemma S-intersections-incident:
  assumes p ≠ 0 and q ≠ 0 and proj2-abs p ≠ proj2-abs q (is ?pp ≠ ?pq)
  and proj2-incident (proj2-abs p) l and proj2-incident (proj2-abs q) l
  shows proj2-incident (S-intersection1 p q) l (is proj2-incident ?pr l)
  and proj2-incident (S-intersection2 p q) l (is proj2-incident ?ps l)
proof -
  from ‹p ≠ 0› and ‹q ≠ 0›
  have proj2-Col ?pp ?pq ?pr and proj2-Col ?pp ?pq ?ps
    by (rule S-intersections-Col)+
  with ‹?pp ≠ ?pq› and ‹proj2-incident ?pp l› and ‹proj2-incident ?pq l›
    and proj2-incident-iff-Col
```

show *proj2-incident ?pr l* **and** *proj2-incident ?ps l* **by** *fast+*
qed

lemma *K2-line-intersect-twice*:
assumes $a \in K2$ **and** $a \neq r$
shows $\exists\ s\ u.\ s \neq u \wedge s \in S \wedge u \in S \wedge$ *proj2-Col a r s* $\wedge$ *proj2-Col a r u*
proof −
let $?a' =$ *proj2-rep a*
let $?r' =$ *proj2-rep r*
from *proj2-rep-non-zero* **have** $?a' \neq 0$ **and** $?r' \neq 0$ **by** *simp-all*

from ‹$?a' \neq 0$› **and** *K2-imp-M-neg* **and** *proj2-abs-rep* **and** ‹$a \in K2$›
have $?a' \cdot (M *v\ ?a') < 0$ **by** *simp*

from ‹$a \neq r$› **have** *proj2-abs ?a'* $\neq$ *proj2-abs ?r'* **by** (*simp add*: *proj2-abs-rep*)

from ‹$a \in K2$› **have** *proj2-abs ?a'* $\in K2$ **by** (*simp add*: *proj2-abs-rep*)
with ‹$?a' \neq 0$› **and** ‹$?r' \neq 0$› **and** ‹*proj2-abs ?a'* $\neq$ *proj2-abs ?r'*›
have *S-intersection1 ?a' ?r'* $\neq$ *S-intersection2 ?a' ?r'* (**is** $?s \neq ?u$)
by (*rule S-intersections-distinct*)

from ‹$?a' \neq 0$› **and** ‹$?r' \neq 0$› **and** ‹*proj2-abs ?a'* $\neq$ *proj2-abs ?r'*›
and ‹*proj2-abs ?a'* $\in K2$›
have $?s \in S$ **and** $?u \in S$ **by** (*rule S-intersections-in-S*)+

from ‹$?a' \neq 0$› **and** ‹$?r' \neq 0$›
have *proj2-Col* (*proj2-abs ?a'*) (*proj2-abs ?r'*) *?s*
and *proj2-Col* (*proj2-abs ?a'*) (*proj2-abs ?r'*) *?u*
by (*rule S-intersections-Col*)+
hence *proj2-Col a r ?s* **and** *proj2-Col a r ?u*
by (*simp-all add*: *proj2-abs-rep*)
with ‹$?s \neq ?u$› **and** ‹$?s \in S$› **and** ‹$?u \in S$›
show $\exists\ s\ u.\ s \neq u \wedge s \in S \wedge u \in S \wedge$ *proj2-Col a r s* $\wedge$ *proj2-Col a r u*
by *auto*
qed

lemma *point-in-S-polar-is-tangent*:
assumes $p \in S$ **and** $q \in S$ **and** *proj2-incident q* (*polar p*)
shows $q = p$
proof −
from ‹$p \in S$› **have** *proj2-incident p* (*polar p*)
by (*subst incident-own-polar-in-S*)

from *line-incident-point-not-in-S*
obtain *r* **where** $r \notin S$ **and** *proj2-incident r* (*polar p*) **by** *auto*
let $?u =$ *proj2-rep r*
let $?v =$ *proj2-rep p*
from ‹$r \notin S$› **and** ‹$p \in S$› **and** ‹$q \in S$› **have** $r \neq p$ **and** $q \neq r$ **by** *auto*
with ‹*proj2-incident p* (*polar p*)›

and ‹*proj2-incident q* (*polar p*)›
and ‹*proj2-incident r* (*polar p*)›
and *proj2-incident-iff* [*of r p polar p q*]
obtain k **where** $q = proj2\text{-}abs\ (k *_R\ ?u + ?v)$ **by** *auto*
with ‹$r \neq p$› **and** ‹$q \in S$› **and** *S-quadratic*
have $?u \cdot (M *v\ ?u) * k^2 + ?u \cdot (M *v\ ?v) * 2 * k + ?v \cdot (M *v\ ?v) = 0$
by *simp*
moreover from ‹$p \in S$› **have** $?v \cdot (M *v\ ?v) = 0$ **by** (*unfold S-alt-def*)
moreover from ‹*proj2-incident r* (*polar p*)›
have $?u \cdot (M *v\ ?v) = 0$ **by** (*unfold incident-polar*)
moreover from ‹$r \notin S$› **have** $?u \cdot (M *v\ ?u) \neq 0$ **by** (*unfold S-alt-def*)
ultimately have $k = 0$ **by** *simp*
with ‹$q = proj2\text{-}abs\ (k *_R\ ?u + ?v)$›
show $q = p$ **by** (*simp add*: *proj2-abs-rep*)
qed

lemma *line-through-K2-intersect-S-twice*:
assumes $p \in K2$ **and** *proj2-incident p l*
shows $\exists\ q\ r.\ q \neq r \wedge q \in S \wedge r \in S \wedge$ *proj2-incident q l* $\wedge$ *proj2-incident r l*
proof −
from *proj2-another-point-on-line*
obtain s **where** $s \neq p$ **and** *proj2-incident s l* **by** *auto*
from ‹$p \in K2$› **and** ‹$s \neq p$› **and** *K2-line-intersect-twice* [*of p s*]
obtain q **and** r **where** $q \neq r$ **and** $q \in S$ **and** $r \in S$
and *proj2-Col p s q* **and** *proj2-Col p s r*
by *auto*
with ‹$s \neq p$› **and** ‹*proj2-incident p l*› **and** ‹*proj2-incident s l*›
and *proj2-incident-iff-Col* [*of p s*]
have *proj2-incident q l* **and** *proj2-incident r l* **by** *fast+*
with ‹$q \neq r$› **and** ‹$q \in S$› **and** ‹$r \in S$›
show $\exists\ q\ r.\ q \neq r \wedge q \in S \wedge r \in S \wedge$ *proj2-incident q l* $\wedge$ *proj2-incident r l*
by *auto*
qed

lemma *line-through-K2-intersect-S-again*:
assumes $p \in K2$ **and** *proj2-incident p l*
shows $\exists\ r.\ r \neq q \wedge r \in S \wedge$ *proj2-incident r l*
proof −
from ‹$p \in K2$› **and** ‹*proj2-incident p l*›
and *line-through-K2-intersect-S-twice* [*of p l*]
obtain s **and** t **where** $s \neq t$ **and** $s \in S$ **and** $t \in S$
and *proj2-incident s l* **and** *proj2-incident t l*
by *auto*
show $\exists\ r.\ r \neq q \wedge r \in S \wedge$ *proj2-incident r l*
proof *cases*
assume $t = q$
with ‹$s \neq t$› **and** ‹$s \in S$› **and** ‹*proj2-incident s l*›
have $s \neq q \wedge s \in S \wedge$ *proj2-incident s l* **by** *simp*
thus $\exists\ r.\ r \neq q \wedge r \in S \wedge$ *proj2-incident r l* **..**

```
  next
    assume t ≠ q
    with ‹t ∈ S› and ‹proj2-incident t l›
    have t ≠ q ∧ t ∈ S ∧ proj2-incident t l by simp
    thus ∃ r. r ≠ q ∧ r ∈ S ∧ proj2-incident r l ..
  qed
qed

lemma line-through-K2-intersect-S:
  assumes p ∈ K2 and proj2-incident p l
  shows ∃ r. r ∈ S ∧ proj2-incident r l
proof −
  from assms
  have ∃ r. r ≠ p ∧ r ∈ S ∧ proj2-incident r l
    by (rule line-through-K2-intersect-S-again)
  thus ∃ r. r ∈ S ∧ proj2-incident r l by auto
qed

lemma line-intersect-S-at-most-twice:
  ∃ p q. ∀ r∈S. proj2-incident r l ⟶ r = p ∨ r = q
proof −
  from line-incident-point-not-in-S
  obtain s where s ∉ S and proj2-incident s l by auto
  let ?v = proj2-rep s
  from proj2-another-point-on-line
  obtain t where t ≠ s and proj2-incident t l by auto
  let ?w = proj2-rep t
  have ?v ≠ 0 and ?w ≠ 0 by (rule proj2-rep-non-zero)+

  let ?a = ?v · (M *v ?v)
  let ?b = 2 * (?v · (M *v ?w))
  let ?c = ?w · (M *v ?w)
  from ‹s ∉ S› have ?a ≠ 0
    unfolding S-def and conic-sgn-def
    by auto
  let ?j = (−?b + sqrt (discrim ?a ?b ?c)) / (2 * ?a)
  let ?k = (−?b − sqrt (discrim ?a ?b ?c)) / (2 * ?a)
  let ?p = proj2-abs (?j *R ?v + ?w)
  let ?q = proj2-abs (?k *R ?v + ?w)
  have ∀ r∈S. proj2-incident r l ⟶ r = ?p ∨ r = ?q
  proof
    fix r
    assume r ∈ S
    with ‹s ∉ S› have r ≠ s by auto
    { assume proj2-incident r l
      with ‹t ≠ s› and ‹r ≠ s› and ‹proj2-incident s l› and ‹proj2-incident t l›
        and proj2-incident-iff [of s t l r]
      obtain i where r = proj2-abs (i *R ?v + ?w) by auto
      with ‹r ∈ S› and ‹t ≠ s› and S-quadratic
```

    have $?a * i^2 + ?b * i + ?c = 0$ **by** *simp*
    with ‹$?a \neq 0$› **and** *discriminant-iff* **have** $i = ?j \vee i = ?k$ **by** *simp*
    with ‹$r = proj2\text{-}abs\ (i *_R ?v + ?w)$› **have** $r = ?p \vee r = ?q$ **by** *auto* **}**
  thus $proj2\text{-}incident\ r\ l \longrightarrow r = ?p \vee r = ?q$ **..**
 qed
 thus $\exists\ p\ q.\ \forall\ r{\in}S.\ proj2\text{-}incident\ r\ l \longrightarrow r = p \vee r = q$ **by** *auto*
qed

lemma *card-line-intersect-S*:
 assumes $T \subseteq S$ **and** *proj2-set-Col T*
 shows $card\ T \leq 2$
proof −
 from ‹*proj2-set-Col T*›
 obtain *l* **where** $\forall\ p{\in}T.\ proj2\text{-}incident\ p\ l$ **unfolding** *proj2-set-Col-def* **..**
 from *line-intersect-S-at-most-twice* [*of l*]
 obtain *b* **and** *c* **where** $\forall\ a{\in}S.\ proj2\text{-}incident\ a\ l \longrightarrow a = b \vee a = c$ **by** *auto*
 with ‹$\forall\ p{\in}T.\ proj2\text{-}incident\ p\ l$› **and** ‹$T \subseteq S$›
 have $T \subseteq \{b,c\}$ **by** *auto*
 hence $card\ T \leq card\ \{b,c\}$ **by** (*simp add*: *card-mono*)
 also from *card-suc-ge-insert* [*of b* $\{c\}$] **have** $\ldots \leq 2$ **by** *simp*
 finally show $card\ T \leq 2$ **.**
qed

lemma *line-S-two-intersections-only*:
 assumes $p \neq q$ **and** $p \in S$ **and** $q \in S$ **and** $r \in S$
 and *proj2-incident p l* **and** *proj2-incident q l* **and** *proj2-incident r l*
 shows $r = p \vee r = q$
proof −
 from ‹$p \neq q$› **have** $card\ \{p,q\} = 2$ **by** *simp*

 from ‹$p \in S$› **and** ‹$q \in S$› **and** ‹$r \in S$› **have** $\{r,p,q\} \subseteq S$ **by** *simp-all*

 from ‹*proj2-incident p l*› **and** ‹*proj2-incident q l*› **and** ‹*proj2-incident r l*›
 have *proj2-set-Col* $\{r,p,q\}$
  by (*unfold proj2-set-Col-def*) (*simp add*: *exI* [*of - l*])
 with ‹$\{r,p,q\} \subseteq S$› **have** $card\ \{r,p,q\} \leq 2$ **by** (*rule card-line-intersect-S*)

 show $r = p \vee r = q$
 proof (*rule ccontr*)
  assume $\neg\ (r = p \vee r = q)$
  hence $r \notin \{p,q\}$ **by** *simp*
  with ‹$card\ \{p,q\} = 2$› **and** *card-insert-disjoint* [*of* $\{p,q\}$ *r*]
  have $card\ \{r,p,q\} = 3$ **by** *simp*
  with ‹$card\ \{r,p,q\} \leq 2$› **show** *False* **by** *simp*
 qed
qed

lemma *line-through-K2-intersect-S-exactly-twice*:
 assumes $p \in K2$ **and** *proj2-incident p l*

shows ∃ *q r*. $q \neq r \wedge q \in S \wedge r \in S \wedge$ *proj2-incident q l* ∧ *proj2-incident r l* ∧ (∀ *s*∈*S*. *proj2-incident s l* ⟶ $s = q \vee s = r$)
proof −
from ‹$p \in K2$› **and** ‹*proj2-incident p l*›
and *line-through-K2-intersect-S-twice* [*of p l*]
obtain *q* **and** *r* **where** $q \neq r$ **and** $q \in S$ **and** $r \in S$
and *proj2-incident q l* **and** *proj2-incident r l*
by *auto*
with *line-S-two-intersections-only*
show ∃ *q r*. $q \neq r \wedge q \in S \wedge r \in S \wedge$ *proj2-incident q l* ∧ *proj2-incident r l*
∧ (∀ *s*∈*S*. *proj2-incident s l* ⟶ $s = q \vee s = r$)
by *blast*
qed

lemma *tangent-not-through-K2*:
assumes $p \in S$ **and** $q \in K2$
shows ¬ *proj2-incident q* (*polar p*)
proof
assume *proj2-incident q* (*polar p*)
with ‹$q \in K2$› **and** *line-through-K2-intersect-S-again* [*of q polar p p*]
obtain *r* **where** $r \neq p$ **and** $r \in S$ **and** *proj2-incident r* (*polar p*) **by** *auto*
from ‹$p \in S$› **and** ‹$r \in S$› **and** ‹*proj2-incident r* (*polar p*)›
have $r = p$ **by** (*rule point-in-S-polar-is-tangent*)
with ‹$r \neq p$› **show** *False* **..**
qed

lemma *outside-exists-line-not-intersect-S*:
assumes *conic-sgn p* = *1*
shows ∃ *l*. *proj2-incident p l* ∧ (∀ *q*. *proj2-incident q l* ⟶ $q \notin S$)
proof −
let *?r* = *proj2-intersection* (*polar p*) *z-zero*
have *proj2-incident ?r* (*polar p*) **and** *proj2-incident ?r z-zero*
by (*rule proj2-intersection-incident*)+
from ‹*proj2-incident ?r z-zero*›
have *conic-sgn ?r* = *1* **by** (*rule z-zero-conic-sgn-1*)
with ‹*conic-sgn p* = *1*›
have *proj2-rep p* · (*M* ∗*v proj2-rep p*) > *0*
and *proj2-rep ?r* · (*M* ∗*v proj2-rep ?r*) > *0*
by (*unfold conic-sgn-def*) (*simp-all add*: *sgn-1-pos*)

from ‹*proj2-incident ?r* (*polar p*)›
have *proj2-incident p* (*polar ?r*) **by** (*rule incident-polar-swap*)
hence *proj2-rep p* · (*M* ∗*v proj2-rep ?r*) = *0* **by** (*simp add*: *incident-polar*)

have *p* ≠ *?r*
proof
assume *p* = *?r*
with ‹*proj2-incident ?r* (*polar p*)› **have** *proj2-incident p* (*polar p*) **by** *simp*
hence *proj2-rep p* · (*M* ∗*v proj2-rep p*) = *0* **by** (*simp add*: *incident-polar*)

    with ‹*proj2-rep p* · (*M* ∗*v proj2-rep p*) > *0*› **show** *False* **by** *simp*
  qed

  let *?l* = *proj2-line-through p ?r*
  have *proj2-incident p ?l* **and** *proj2-incident ?r ?l*
    by (*rule proj2-line-through-incident*)+

  have ∀ *q*. *proj2-incident q ?l* ⟶ *q* ∉ *S*
  proof
    fix *q*
    show *proj2-incident q ?l* ⟶ *q* ∉ *S*
    proof
      assume *proj2-incident q ?l*
      with ‹*p* ≠ *?r*› **and** ‹*proj2-incident p ?l*› **and** ‹*proj2-incident ?r ?l*›
      have *q* = *p* ∨ (∃ *k*. *q* = *proj2-abs* (*k* $*_R$ *proj2-rep p* + *proj2-rep ?r*))
        by (*simp add*: *proj2-incident-iff* [*of p ?r ?l q*])

      show *q* ∉ *S*
      proof *cases*
        assume *q* = *p*
        with ‹*conic-sgn p* = *1*› **show** *q* ∉ *S* **by** (*unfold S-def*) *simp*
      next
        assume *q* ≠ *p*
        with ‹*q* = *p* ∨ (∃ *k*. *q* = *proj2-abs* (*k* $*_R$ *proj2-rep p* + *proj2-rep ?r*))›
        obtain *k* **where** *q* = *proj2-abs* (*k* $*_R$ *proj2-rep p* + *proj2-rep ?r*)
          by *auto*
        from ‹*proj2-rep p* · (*M* ∗*v proj2-rep p*) > *0*›
        have *proj2-rep p* · (*M* ∗*v proj2-rep p*) ∗ k^2 ≥ *0*
          by *simp*
        with ‹*proj2-rep p* · (*M* ∗*v proj2-rep ?r*) = *0*›
          and ‹*proj2-rep ?r* · (*M* ∗*v proj2-rep ?r*) > *0*›
        have *proj2-rep p* · (*M* ∗*v proj2-rep p*) ∗ k^2
          + *proj2-rep p* · (*M* ∗*v proj2-rep ?r*) ∗ *2* ∗ *k*
          + *proj2-rep ?r* · (*M* ∗*v proj2-rep ?r*)
          > *0*
          by *simp*
        with ‹*p* ≠ *?r*› **and** ‹*q* = *proj2-abs* (*k* $*_R$ *proj2-rep p* + *proj2-rep ?r*)›
        show *q* ∉ *S* **by** (*simp add*: *S-quadratic*)
      qed
    qed
  qed
  with ‹*proj2-incident p ?l*›
  show ∃ *l*. *proj2-incident p l* ∧ (∀ *q*. *proj2-incident q l* ⟶ *q* ∉ *S*)
    by (*simp add*: *exI* [*of* - *?l*])
qed

lemma *lines-through-intersect-S-twice-in-K2*:
  assumes ∀ *l*. *proj2-incident p l*
  ⟶ (∃ *q r*. *q* ≠ *r* ∧ *q* ∈ *S* ∧ *r* ∈ *S* ∧ *proj2-incident q l* ∧ *proj2-incident r l*)

```
  shows p ∈ K2
proof (rule ccontr)
  assume p ∉ K2
  hence conic-sgn p ≥ 0 by (unfold K2-def) simp

  have ¬ (∀ l. proj2-incident p l ⟶ (∃ q r.
    q ≠ r ∧ q ∈ S ∧ r ∈ S ∧ proj2-incident q l ∧ proj2-incident r l))
  proof cases
    assume conic-sgn p = 0
    hence p ∈ S unfolding S-def ..
    hence proj2-incident p (polar p) by (simp add: incident-own-polar-in-S)
    let ?l = polar p
    have ¬ (∃ q r.
      q ≠ r ∧ q ∈ S ∧ r ∈ S ∧ proj2-incident q ?l ∧ proj2-incident r ?l)
    proof
      assume ∃ q r.
        q ≠ r ∧ q ∈ S ∧ r ∈ S ∧ proj2-incident q ?l ∧ proj2-incident r ?l
      then obtain q and r where q ≠ r and q ∈ S and r ∈ S
        and proj2-incident q ?l and proj2-incident r ?l
        by auto
      from ‹p ∈ S› and ‹q ∈ S› and ‹proj2-incident q ?l›
        and ‹r ∈ S› and ‹proj2-incident r ?l›
      have q = p and r = p by (simp add: point-in-S-polar-is-tangent)+
      with ‹q ≠ r› show False by simp
    qed
    with ‹proj2-incident p ?l›
    show ¬ (∀ l. proj2-incident p l ⟶ (∃ q r.
      q ≠ r ∧ q ∈ S ∧ r ∈ S ∧ proj2-incident q l ∧ proj2-incident r l))
      by auto
  next
    assume conic-sgn p ≠ 0
    with ‹conic-sgn p ≥ 0› have conic-sgn p > 0 by simp
    hence sgn (conic-sgn p) = 1 by simp
    hence conic-sgn p = 1 by (simp add: sgn-conic-sgn)
    with outside-exists-line-not-intersect-S
    obtain l where proj2-incident p l and ∀ q. proj2-incident q l ⟶ q ∉ S
      by auto
    have ¬ (∃ q r.
      q ≠ r ∧ q ∈ S ∧ r ∈ S ∧ proj2-incident q l ∧ proj2-incident r l)
    proof
      assume ∃ q r.
        q ≠ r ∧ q ∈ S ∧ r ∈ S ∧ proj2-incident q l ∧ proj2-incident r l
      then obtain q where q ∈ S and proj2-incident q l by auto
      from ‹proj2-incident q l› and ‹∀ q. proj2-incident q l ⟶ q ∉ S›
      have q ∉ S by simp
      with ‹q ∈ S› show False by simp
    qed
    with ‹proj2-incident p l›
    show ¬ (∀ l. proj2-incident p l ⟶ (∃ q r.
```

$q \neq r \wedge q \in S \wedge r \in S \wedge$ *proj2-incident* $q\ l \wedge$ *proj2-incident* $r\ l))$
by *auto*
qed
with ‹$\forall\ l.$ *proj2-incident* $p\ l \longrightarrow (\exists\ q\ r.$
$q \neq r \wedge q \in S \wedge r \in S \wedge$ *proj2-incident* $q\ l \wedge$ *proj2-incident* $r\ l)$›
show *False* **by** *simp*
qed

lemma *line-through-hyp2-pole-not-in-hyp2*:
assumes $a \in$ *hyp2* **and** *proj2-incident* $a\ l$
shows *pole* $l \notin$ *hyp2*
proof −
from *assms* **and** *line-through-K2-intersect-S*
obtain p **where** $p \in S$ **and** *proj2-incident* $p\ l$ **by** *auto*

from ‹*proj2-incident* $p\ l$›
have *proj2-incident* (*pole* l) (*polar* p) **by** (*rule incident-pole-polar*)
with ‹$p \in S$›
show *pole* $l \notin$ *hyp2*
by (*auto simp add*: *tangent-not-through-K2*)
qed

lemma *statement60-one-way*:
assumes *is-K2-isometry* J **and** $p \in$ *K2*
shows *apply-cltn2* $p\ J \in$ *K2* (**is** $?p' \in$ *K2*)
proof −
let $?J'$ = *cltn2-inverse* J

have $\forall\ l'.$ *proj2-incident* $?p'\ l' \longrightarrow (\exists\ q'\ r'.$
$q' \neq r' \wedge q' \in S \wedge r' \in S \wedge$ *proj2-incident* $q'\ l' \wedge$ *proj2-incident* $r'\ l')$
proof
fix l'
let $?l$ = *apply-cltn2-line* $l'\ ?J'$
show *proj2-incident* $?p'\ l' \longrightarrow (\exists\ q'\ r'.$
$q' \neq r' \wedge q' \in S \wedge r' \in S \wedge$ *proj2-incident* $q'\ l' \wedge$ *proj2-incident* $r'\ l')$
proof
assume *proj2-incident* $?p'\ l'$
hence *proj2-incident* $p\ ?l$
by (*simp add*: *apply-cltn2-incident* [*of* $p\ l'\ ?J'$]
cltn2.inv-inv [*simplified*])
with ‹$p \in$ *K2*› **and** *line-through-K2-intersect-S-twice* [*of* $p\ ?l$]
obtain q **and** r **where** $q \neq r$ **and** $q \in S$ **and** $r \in S$
and *proj2-incident* $q\ ?l$ **and** *proj2-incident* $r\ ?l$
by *auto*
let $?q'$ = *apply-cltn2* $q\ J$
let $?r'$ = *apply-cltn2* $r\ J$
from ‹$q \neq r$› **and** *apply-cltn2-injective* [*of* $q\ J\ r$] **have** $?q' \neq ?r'$ **by** *auto*

from ‹$q \in S$› **and** ‹$r \in S$› **and** ‹*is-K2-isometry* J›

have *?q′* ∈ *S* **and** *?r′* ∈ *S* **by** (*unfold is-K2-isometry-def*) *simp-all*

from ⟨*proj2-incident q ?l*⟩ **and** ⟨*proj2-incident r ?l*⟩
have *proj2-incident ?q′ l′* **and** *proj2-incident ?r′ l′*
by (*simp-all add*: *apply-cltn2-incident* [*of - l′ ?J′*]
cltn2.inv-inv [*simplified*])
with ⟨*?q′* ≠ *?r′*⟩ **and** ⟨*?q′* ∈ *S*⟩ **and** ⟨*?r′* ∈ *S*⟩
show ∃ *q′ r′*.
q′ ≠ *r′* ∧ *q′* ∈ *S* ∧ *r′* ∈ *S* ∧ *proj2-incident q′ l′* ∧ *proj2-incident r′ l′*
by *auto*
qed
qed
thus *?p′* ∈ *K2* **by** (*rule lines-through-intersect-S-twice-in-K2*)
qed

lemma *is-K2-isometry-hyp2-S*:
assumes *p* ∈ *hyp2* ∪ *S* **and** *is-K2-isometry J*
shows *apply-cltn2 p J* ∈ *hyp2* ∪ *S*
proof *cases*
assume *p* ∈ *hyp2*
with ⟨*is-K2-isometry J*⟩
have *apply-cltn2 p J* ∈ *hyp2* **by** (*rule statement60-one-way*)
thus *apply-cltn2 p J* ∈ *hyp2* ∪ *S* **..**
next
assume *p* ∉ *hyp2*
with ⟨*p* ∈ *hyp2* ∪ *S*⟩ **have** *p* ∈ *S* **by** *simp*
with ⟨*is-K2-isometry J*⟩
have *apply-cltn2 p J* ∈ *S* **by** (*unfold is-K2-isometry-def*) *simp*
thus *apply-cltn2 p J* ∈ *hyp2* ∪ *S* **..**
qed

lemma *is-K2-isometry-z-non-zero*:
assumes *p* ∈ *hyp2* ∪ *S* **and** *is-K2-isometry J*
shows *z-non-zero* (*apply-cltn2 p J*)
proof −
from ⟨*p* ∈ *hyp2* ∪ *S*⟩ **and** ⟨*is-K2-isometry J*⟩
have *apply-cltn2 p J* ∈ *hyp2* ∪ *S* **by** (*rule is-K2-isometry-hyp2-S*)
thus *z-non-zero* (*apply-cltn2 p J*) **by** (*rule hyp2-S-z-non-zero*)
qed

lemma *cart2-append1-apply-cltn2*:
assumes *p* ∈ *hyp2* ∪ *S* **and** *is-K2-isometry J*
shows ∃ *k*. *k* ≠ *0*
∧ *cart2-append1 p v*∗ *cltn2-rep J* = *k* $*_R$ *cart2-append1* (*apply-cltn2 p J*)
proof −
have *cart2-append1 p v*∗ *cltn2-rep J*
= (*1* / (*proj2-rep p*)$*3*) $*_R$ (*proj2-rep p v*∗ *cltn2-rep J*)
by (*unfold cart2-append1-def*) (*simp add*: *scaleR-vector-matrix-assoc*)

from ‹p ∈ hyp2 ∪ S› **have** (proj2-rep p)$3 ≠ 0 **by** (rule hyp2-S-z-non-zero)

from apply-cltn2-imp-mult [of p J]
obtain j **where** j ≠ 0
  and proj2-rep p v* cltn2-rep J = j *R proj2-rep (apply-cltn2 p J)
  by auto

from ‹p ∈ hyp2 ∪ S› **and** ‹is-K2-isometry J›
have z-non-zero (apply-cltn2 p J) **by** (rule is-K2-isometry-z-non-zero)
hence proj2-rep (apply-cltn2 p J)
  = (proj2-rep (apply-cltn2 p J))$3 *R cart2-append1 (apply-cltn2 p J)
  by (rule proj2-rep-cart2-append1)

let ?k = 1 / (proj2-rep p)$3 * j * (proj2-rep (apply-cltn2 p J))$3
from ‹(proj2-rep p)$3 ≠ 0› **and** ‹j ≠ 0›
  and ‹(proj2-rep (apply-cltn2 p J))$3 ≠ 0›
have ?k ≠ 0 **by** simp

from ‹cart2-append1 p v* cltn2-rep J
  = (1 / (proj2-rep p)$3) *R (proj2-rep p v* cltn2-rep J)›
  and ‹proj2-rep p v* cltn2-rep J = j *R proj2-rep (apply-cltn2 p J)›
have cart2-append1 p v* cltn2-rep J
  = (1 / (proj2-rep p)$ 3 * j) *R proj2-rep (apply-cltn2 p J)
  by simp

from ‹proj2-rep (apply-cltn2 p J)
  = (proj2-rep (apply-cltn2 p J))$3 *R cart2-append1 (apply-cltn2 p J)›
have (1 / (proj2-rep p)$3 * j) *R proj2-rep (apply-cltn2 p J)
  = (1 / (proj2-rep p)$3 * j) *R ((proj2-rep (apply-cltn2 p J))$3
  *R cart2-append1 (apply-cltn2 p J))
  by simp
with ‹cart2-append1 p v* cltn2-rep J
  = (1 / (proj2-rep p)$ 3 * j) *R proj2-rep (apply-cltn2 p J)›
have cart2-append1 p v* cltn2-rep J = ?k *R cart2-append1 (apply-cltn2 p J)
  by simp
with ‹?k ≠ 0›
show ∃ k. k ≠ 0
  ∧ cart2-append1 p v* cltn2-rep J = k *R cart2-append1 (apply-cltn2 p J)
  by (simp add: exI [of - ?k])
qed

8.5 The K-isometries form a group action

lemma hyp2-cltn2-id [simp]: hyp2-cltn2 p cltn2-id = p
  by (unfold hyp2-cltn2-def) (simp add: Rep-hyp2-inverse)

lemma apply-cltn2-Rep-hyp2:
  assumes is-K2-isometry J
  shows apply-cltn2 (Rep-hyp2 p) J ∈ hyp2

proof –
  from ‹*is-K2-isometry J*› **and** *Rep-hyp2* [*of p*]
  show *apply-cltn2* (*Rep-hyp2 p*) *J* ∈ *K2* **by** (*rule statement60-one-way*)
qed

lemma *Rep-hyp2-cltn2*:
  assumes *is-K2-isometry J*
  shows *Rep-hyp2* (*hyp2-cltn2 p J*) = *apply-cltn2* (*Rep-hyp2 p*) *J*
proof –
  from ‹*is-K2-isometry J*›
  have *apply-cltn2* (*Rep-hyp2 p*) *J* ∈ *hyp2* **by** (*rule apply-cltn2-Rep-hyp2*)
  thus *Rep-hyp2* (*hyp2-cltn2 p J*) = *apply-cltn2* (*Rep-hyp2 p*) *J*
    by (*unfold hyp2-cltn2-def*) (*rule Abs-hyp2-inverse*)
qed

lemma *hyp2-cltn2-compose*:
  assumes *is-K2-isometry H*
  shows *hyp2-cltn2* (*hyp2-cltn2 p H*) *J* = *hyp2-cltn2 p* (*cltn2-compose H J*)
proof –
  from ‹*is-K2-isometry H*›
  have *apply-cltn2* (*Rep-hyp2 p*) *H* ∈ *hyp2* **by** (*rule apply-cltn2-Rep-hyp2*)
  thus *hyp2-cltn2* (*hyp2-cltn2 p H*) *J* = *hyp2-cltn2 p* (*cltn2-compose H J*)
    by (*unfold hyp2-cltn2-def*) (*simp add*: *Abs-hyp2-inverse apply-cltn2-compose*)
qed

interpretation *K2-isometry*: *action*
  (|*carrier* = *Collect is-K2-isometry*, *mult* = *cltn2-compose*, *one* = *cltn2-id*|)
  hyp2-cltn2
proof
  let *?G* =
    (|*carrier* = *Collect is-K2-isometry*, *mult* = *cltn2-compose*, *one* = *cltn2-id*|)
  fix *p*
  show *hyp2-cltn2 p* $\mathbf{1}_{?G}$ = *p*
    by (*unfold hyp2-cltn2-def*) (*simp add*: *Rep-hyp2-inverse*)
  fix *H J*
  show *H* ∈ *carrier ?G* ∧ *J* ∈ *carrier ?G*
    ⟶ *hyp2-cltn2* (*hyp2-cltn2 p H*) *J* = *hyp2-cltn2 p* ($H \otimes_{?G} J$)
    by (*simp add*: *hyp2-cltn2-compose*)
qed

8.6 The Klein–Beltrami model satisfies Tarski's first three axioms

lemma *three-in-S-tangent-intersection-no-3-Col*:
  assumes *p* ∈ *S* **and** *q* ∈ *S* **and** *r* ∈ *S*
  and *p* ≠ *q* **and** *r* ∉ {*p*,*q*}
  shows *proj2-no-3-Col* {*proj2-intersection* (*polar p*) (*polar q*),*r*,*p*,*q*}
  (**is** *proj2-no-3-Col* {*?s*,*r*,*p*,*q*})
proof –

```
let ?T = {?s,r,p,q}

from ‹p ≠ q› have card {p,q} = 2 by simp
with ‹r ∉ {p,q}› have card {r,p,q} = 3 by simp

from ‹p ∈ S› and ‹q ∈ S› and ‹r ∈ S› have {r,p,q} ⊆ S by simp

have proj2-incident ?s (polar p) and proj2-incident ?s (polar q)
  by (rule proj2-intersection-incident)+

have ?s ∉ S
proof
  assume ?s ∈ S
  with ‹p ∈ S› and ‹proj2-incident ?s (polar p)›
    and ‹q ∈ S› and ‹proj2-incident ?s (polar q)›
  have ?s = p and ?s = q by (simp-all add: point-in-S-polar-is-tangent)
  hence p = q by simp
  with ‹p ≠ q› show False ..
qed
with ‹{r,p,q} ⊆ S› have ?s ∉ {r,p,q} by auto
with ‹card {r,p,q} = 3› have card {?s,r,p,q} = 4 by simp

have ∀ t∈?T. ¬ proj2-set-Col (?T − {t})
proof standard+
  fix t
  assume t ∈ ?T
  assume proj2-set-Col (?T − {t})
  then obtain l where ∀ a ∈ (?T − {t}). proj2-incident a l
    unfolding proj2-set-Col-def ..

  from ‹proj2-set-Col (?T − {t})›
  have proj2-set-Col (S ∩ (?T − {t}))
    by (simp add: proj2-subset-Col [of (S ∩ (?T − {t})) ?T − {t}])
  hence card (S ∩ (?T − {t})) ≤ 2 by (simp add: card-line-intersect-S)

  show False
  proof cases
    assume t = ?s
    with ‹?s ∉ {r,p,q}› have ?T − {t} = {r,p,q} by simp
    with ‹{r,p,q} ⊆ S› have S ∩ (?T − {t}) = {r,p,q} by simp
     with ‹card {r,p,q} = 3› and ‹card (S ∩ (?T − {t})) ≤ 2› show False by
simp
  next
    assume t ≠ ?s
    hence ?s ∈ ?T − {t} by simp
    with ‹∀ a ∈ (?T − {t}). proj2-incident a l› have proj2-incident ?s l ..

    from ‹p ≠ q› have {p,q} ∩ ?T − {t} ≠ {} by auto
    then obtain d where d ∈ {p,q} and d ∈ ?T − {t} by auto
```

```
      from ‹d ∈ ?T − {t}› and ‹∀ a ∈ (?T − {t}). proj2-incident a l›
      have proj2-incident d l by simp

      from ‹d ∈ {p,q}›
        and ‹proj2-incident ?s (polar p)›
        and ‹proj2-incident ?s (polar q)›
      have proj2-incident ?s (polar d) by auto

      from ‹d ∈ {p,q}› and ‹{r,p,q} ⊆ S› have d ∈ S by auto
      hence proj2-incident d (polar d) by (unfold incident-own-polar-in-S)

      from ‹d ∈ S› and ‹?s ∉ S› have d ≠ ?s by auto
      with ‹proj2-incident ?s l›
        and ‹proj2-incident d l›
        and ‹proj2-incident ?s (polar d)›
        and ‹proj2-incident d (polar d)›
        and proj2-incident-unique
      have l = polar d by auto
      with ‹d ∈ S› and point-in-S-polar-is-tangent
      have ∀ a∈S. proj2-incident a l ⟶ a = d by simp
      with ‹∀ a ∈ (?T − {t}). proj2-incident a l›
      have S ∩ (?T − {t}) ⊆ {d} by auto
      with card-mono [of {d}] have card (S ∩ (?T − {t})) ≤ 1 by simp
      hence card ((S ∩ ?T) − {t}) ≤ 1 by (simp add: Int-Diff)

      have S ∩ ?T ⊆ insert t ((S ∩ ?T) − {t}) by auto
      with card-suc-ge-insert [of t (S ∩ ?T) − {t}]
        and card-mono [of insert t ((S ∩ ?T) − {t}) S ∩ ?T]
      have card (S ∩ ?T) ≤ card ((S ∩ ?T) − {t}) + 1 by simp
      with ‹card ((S ∩ ?T) − {t}) ≤ 1› have card (S ∩ ?T) ≤ 2 by simp

      from ‹{r,p,q} ⊆ S› have {r,p,q} ⊆ S ∩ ?T by simp
      with ‹card {r,p,q} = 3› and card-mono [of S ∩ ?T {r,p,q}]
      have card (S ∩ ?T) ≥ 3 by simp
      with ‹card (S ∩ ?T) ≤ 2› show False by simp
    qed
  qed
  with ‹card ?T = 4› show proj2-no-3-Col ?T unfolding proj2-no-3-Col-def ..
qed

lemma statement65-special-case:
  assumes p ∈ S and q ∈ S and r ∈ S and p ≠ q and r ∉ {p,q}
  shows ∃ J. is-K2-isometry J
  ∧ apply-cltn2 east J = p
  ∧ apply-cltn2 west J = q
  ∧ apply-cltn2 north J = r
  ∧ apply-cltn2 far-north J = proj2-intersection (polar p) (polar q)
proof −
  let ?s = proj2-intersection (polar p) (polar q)
```

```
let ?t = vector [vector [?s,r,p,q], vector [far-north, north, east, west]]
  :: proj2^4^2
have range (($) (?t$1)) = {?s, r, p, q}
  unfolding image-def
  by (auto simp add: UNIV-4 vector-4)
with ‹p ∈ S› and ‹q ∈ S› and ‹r ∈ S› and ‹p ≠ q› and ‹r ∉ {p,q}›
have proj2-no-3-Col (range (($) (?t$1)))
  by (simp add: three-in-S-tangent-intersection-no-3-Col)
moreover have range (($) (?t$2)) = {far-north, north, east, west}
  unfolding image-def
  by (auto simp add: UNIV-4 vector-4)
 with compass-in-S and east-west-distinct and north-not-east-or-west
   and east-west-tangents-far-north
   and three-in-S-tangent-intersection-no-3-Col [of east west north]
 have proj2-no-3-Col (range (($) (?t$2))) by simp
 ultimately have ∀ i. proj2-no-3-Col (range (($) (?t$i)))
   by (simp add: forall-2)
 hence ∃ J. ∀ j. apply-cltn2 (?t$0$j) J = ?t$1$j
   by (rule statement53-existence)
 moreover have 0 = (2::2) by simp
 ultimately obtain J where ∀ j. apply-cltn2 (?t$2$j) J = ?t$1$j by auto
 hence apply-cltn2 (?t$2$1) J = ?t$1$1
   and apply-cltn2 (?t$2$2) J = ?t$1$2
   and apply-cltn2 (?t$2$3) J = ?t$1$3
   and apply-cltn2 (?t$2$4) J = ?t$1$4
   by simp-all
 hence apply-cltn2 east J = p
   and apply-cltn2 west J = q
   and apply-cltn2 north J = r
   and apply-cltn2 far-north J = ?s
   by (simp-all add: vector-2 vector-4)
 with compass-non-zero
 have p = proj2-abs (vector [1,0,1] v* cltn2-rep J)
   and q = proj2-abs (vector [−1,0,1] v* cltn2-rep J)
   and r = proj2-abs (vector [0,1,1] v* cltn2-rep J)
   and ?s = proj2-abs (vector [0,1,0] v* cltn2-rep J)
   unfolding compass-defs and far-north-def
   by (simp-all add: apply-cltn2-left-abs)

 let ?N = cltn2-rep J ** M ** transpose (cltn2-rep J)
 from M-symmatrix have symmatrix ?N by (rule symmatrix-preserve)
 hence ?N$2$1 = ?N$1$2 and ?N$3$1 = ?N$1$3 and ?N$3$2 = ?N$2$3
   unfolding symmatrix-def and transpose-def
   by (simp-all add: vec-eq-iff)

 from compass-non-zero and ‹apply-cltn2 east J = p› and ‹p ∈ S›
   and apply-cltn2-abs-in-S [of vector [1,0,1] J]
 have (vector [1,0,1] :: real^3) · (?N *v vector [1,0,1]) = 0
   unfolding east-def
```

by *simp*
hence ?N$1$1 + ?N$1$3 + ?N$3$1 + ?N$3$3 = 0
unfolding *inner-vec-def* **and** *matrix-vector-mult-def*
by (*simp add*: *sum-3 vector-3*)
with ⟨?N$3$1 = ?N$1$3⟩ **have** ?N$1$1 + 2 ∗ (?N$1$3) + ?N$3$3 = 0 **by** *simp*

from *compass-non-zero* **and** ⟨*apply-cltn2 west J* = *q*⟩ **and** ⟨*q* ∈ *S*⟩
and *apply-cltn2-abs-in-S* [*of vector* [−1,0,1] *J*]
have (*vector* [−1,0,1] :: *real*ˆ3) · (?N ∗*v vector* [−1,0,1]) = 0
unfolding *west-def*
by *simp*
hence ?N$1$1 − ?N$1$3 − ?N$3$1 + ?N$3$3 = 0
unfolding *inner-vec-def* **and** *matrix-vector-mult-def*
by (*simp add*: *sum-3 vector-3*)
with ⟨?N$3$1 = ?N$1$3⟩ **have** ?N$1$1 − 2 ∗ (?N$1$3) + ?N$3$3 = 0 **by** *simp*
with ⟨?N$1$1 + 2 ∗ (?N$1$3) + ?N$3$3 = 0⟩
have ?N$1$1 + 2 ∗ (?N$1$3) + ?N$3$3 = ?N$1$1 − 2 ∗ (?N$1$3) + ?N$3$3
by *simp*
hence ?N$1$3 = 0 **by** *simp*
with ⟨?N$1$1 + 2 ∗ (?N$1$3) + ?N$3$3 = 0⟩ **have** ?N$3$3 = − (?N$1$1) **by** *simp*

from *compass-non-zero* **and** ⟨*apply-cltn2 north J* = *r*⟩ **and** ⟨*r* ∈ *S*⟩
and *apply-cltn2-abs-in-S* [*of vector* [0,1,1] *J*]
have (*vector* [0,1,1] :: *real*ˆ3) · (?N ∗*v vector* [0,1,1]) = 0
unfolding *north-def*
by *simp*
hence ?N$2$2 + ?N$2$3 + ?N$3$2 + ?N$3$3 = 0
unfolding *inner-vec-def* **and** *matrix-vector-mult-def*
by (*simp add*: *sum-3 vector-3*)
with ⟨?N$3$2 = ?N$2$3⟩ **have** ?N$2$2 + 2 ∗ (?N$2$3) + ?N$3$3 = 0 **by** *simp*

have *proj2-incident ?s* (*polar p*) **and** *proj2-incident ?s* (*polar q*)
by (*rule proj2-intersection-incident*)+

from *compass-non-zero*
have *vector* [1,0,1] *v*∗ *cltn2-rep J* ≠ 0
and *vector* [−1,0,1] *v*∗ *cltn2-rep J* ≠ 0
and *vector* [0,1,0] *v*∗ *cltn2-rep J* ≠ 0
by (*simp-all add*: *non-zero-mult-rep-non-zero*)
from ⟨*vector* [1,0,1] *v*∗ *cltn2-rep J* ≠ 0⟩
and ⟨*vector* [−1,0,1] *v*∗ *cltn2-rep J* ≠ 0⟩
and ⟨*p* = *proj2-abs* (*vector* [1,0,1] *v*∗ *cltn2-rep J*)⟩
and ⟨*q* = *proj2-abs* (*vector* [−1,0,1] *v*∗ *cltn2-rep J*)⟩
have *polar p* = *proj2-line-abs* (*M* ∗*v* (*vector* [1,0,1] *v*∗ *cltn2-rep J*))

and *polar q* = *proj2-line-abs* (*M* ∗*v* (*vector* [−*1*,*0*,*1*] *v*∗ *cltn2-rep J*))
by (*simp-all add*: *polar-abs*)

from ‹*vector* [*1*,*0*,*1*] *v*∗ *cltn2-rep J* ≠ *0*›
and ‹*vector* [−*1*,*0*,*1*] *v*∗ *cltn2-rep J* ≠ *0*›
and *M-invertible*
have *M* ∗*v* (*vector* [*1*,*0*,*1*] *v*∗ *cltn2-rep J*) ≠ *0*
and *M* ∗*v* (*vector* [−*1*,*0*,*1*] *v*∗ *cltn2-rep J*) ≠ *0*
by (*simp-all add*: *invertible-times-non-zero*)
with ‹*vector* [*0*,*1*,*0*] *v*∗ *cltn2-rep J* ≠ *0*›
and ‹*polar p* = *proj2-line-abs* (*M* ∗*v* (*vector* [*1*,*0*,*1*] *v*∗ *cltn2-rep J*))›
and ‹*polar q* = *proj2-line-abs* (*M* ∗*v* (*vector* [−*1*,*0*,*1*] *v*∗ *cltn2-rep J*))›
and ‹*?s* = *proj2-abs* (*vector* [*0*,*1*,*0*] *v*∗ *cltn2-rep J*)›
have *proj2-incident ?s* (*polar p*)
⟷ (*vector* [*0*,*1*,*0*] *v*∗ *cltn2-rep J*)
· (*M* ∗*v* (*vector* [*1*,*0*,*1*] *v*∗ *cltn2-rep J*)) = *0*
and *proj2-incident ?s* (*polar q*)
⟷ (*vector* [*0*,*1*,*0*] *v*∗ *cltn2-rep J*)
· (*M* ∗*v* (*vector* [−*1*,*0*,*1*] *v*∗ *cltn2-rep J*)) = *0*
by (*simp-all add*: *proj2-incident-abs*)
with ‹*proj2-incident ?s* (*polar p*)› **and** ‹*proj2-incident ?s* (*polar q*)›
have (*vector* [*0*,*1*,*0*] *v*∗ *cltn2-rep J*)
· (*M* ∗*v* (*vector* [*1*,*0*,*1*] *v*∗ *cltn2-rep J*)) = *0*
and (*vector* [*0*,*1*,*0*] *v*∗ *cltn2-rep J*)
· (*M* ∗*v* (*vector* [−*1*,*0*,*1*] *v*∗ *cltn2-rep J*)) = *0*
by *simp-all*
hence *vector* [*0*,*1*,*0*] · (*?N* ∗*v vector* [*1*,*0*,*1*]) = *0*
and *vector* [*0*,*1*,*0*] · (*?N* ∗*v vector* [−*1*,*0*,*1*]) = *0*
by (*simp-all add*: *dot-lmul-matrix matrix-vector-mul-assoc* [*symmetric*])
hence *?N*\$*2*\$*1* + *?N*\$*2*\$*3* = *0* **and** −(*?N*\$*2*\$*1*) + *?N*\$*2*\$*3* = *0*
unfolding *inner-vec-def* **and** *matrix-vector-mult-def*
by (*simp-all add*: *sum-3 vector-3*)
hence *?N*\$*2*\$*1* + *?N*\$*2*\$*3* = −(*?N*\$*2*\$*1*) + *?N*\$*2*\$*3* **by** *simp*
hence *?N*\$*2*\$*1* = *0* **by** *simp*
with ‹*?N*\$*2*\$*1* + *?N*\$*2*\$*3* = *0*› **have** *?N*\$*2*\$*3* = *0* **by** *simp*
with ‹*?N*\$*2*\$*2* + *2* ∗ (*?N*\$*2*\$*3*) + *?N*\$*3*\$*3* = *0*› **and** ‹*?N*\$*3*\$*3* = −(*?N*\$*1*\$*1*)›
have *?N*\$*2*\$*2* = *?N*\$*1*\$*1* **by** *simp*
with ‹*?N*\$*1*\$*3* = *0*› **and** ‹*?N*\$*2*\$*1* = *?N*\$*1*\$*2*› **and** ‹*?N*\$*1*\$*3* = *0*›
and ‹*?N*\$*2*\$*1* = *0*› **and** ‹*?N*\$*2*\$*2* = *?N*\$*1*\$*1*› **and** ‹*?N*\$*2*\$*3* = *0*›
and ‹*?N*\$*3*\$*1* = *?N*\$*1*\$*3*› **and** ‹*?N*\$*3*\$*2* = *?N*\$*2*\$*3*› **and** ‹*?N*\$*3*\$*3* = −(*?N*\$*1*\$*1*)›
have *?N* = (*?N*\$*1*\$*1*) $*_R$ *M*
unfolding *M-def*
by (*simp add*: *vec-eq-iff vector-3 forall-3*)

have *invertible* (*cltn2-rep J*) **by** (*rule cltn2-rep-invertible*)
with *M-invertible*
have *invertible ?N* **by** (*simp add*: *invertible-mult transpose-invertible*)
hence *?N* ≠ *0* **by** (*auto simp add*: *zero-not-invertible*)

```
    with ‹?N = (?N$1$1) *R M› have ?N$1$1 ≠ 0 by auto
    with ‹?N = (?N$1$1) *R M›
    have is-K2-isometry (cltn2-abs (cltn2-rep J))
      by (simp add: J-M-J-transpose-K2-isometry)
    hence is-K2-isometry J by (simp add: cltn2-abs-rep)
    with ‹apply-cltn2 east J = p›
      and ‹apply-cltn2 west J = q›
      and ‹apply-cltn2 north J = r›
      and ‹apply-cltn2 far-north J = ?s›
    show ∃ J. is-K2-isometry J
      ∧ apply-cltn2 east J = p
      ∧ apply-cltn2 west J = q
      ∧ apply-cltn2 north J = r
      ∧ apply-cltn2 far-north J = ?s
      by auto
qed

lemma statement66-existence:
  assumes a1 ∈ K2 and a2 ∈ K2 and p1 ∈ S and p2 ∈ S
  shows ∃ J. is-K2-isometry J ∧ apply-cltn2 a1 J = a2 ∧ apply-cltn2 p1 J = p2
proof −
  let ?a = vector [a1,a2] :: proj2^2
  from ‹a1 ∈ K2› and ‹a2 ∈ K2› have ∀ i. ?a$i ∈ K2 by (simp add: forall-2)

  let ?p = vector [p1,p2] :: proj2^2
  from ‹p1 ∈ S› and ‹p2 ∈ S› have ∀ i. ?p$i ∈ S by (simp add: forall-2)

  let ?l = χ i. proj2-line-through (?a$i) (?p$i)
  have ∀ i. proj2-incident (?a$i) (?l$i)
    by (simp add: proj2-line-through-incident)
  hence proj2-incident (?a$1) (?l$1) and proj2-incident (?a$2) (?l$2)
    by fast+

  have ∀ i. proj2-incident (?p$i) (?l$i)
    by (simp add: proj2-line-through-incident)
  hence proj2-incident (?p$1) (?l$1) and proj2-incident (?p$2) (?l$2)
    by fast+

  let ?q = χ i. ϵ qi. qi ≠ ?p$i ∧ qi ∈ S ∧ proj2-incident qi (?l$i)
  have ∀ i. ?q$i ≠ ?p$i ∧ ?q$i ∈ S ∧ proj2-incident (?q$i) (?l$i)
  proof
    fix i
    from ‹∀ i. ?a$i ∈ K2› have ?a$i ∈ K2 ..

    from ‹∀ i. proj2-incident (?a$i) (?l$i)›
    have proj2-incident (?a$i) (?l$i) ..
    with ‹?a$i ∈ K2›
    have ∃ qi. qi ≠ ?p$i ∧ qi ∈ S ∧ proj2-incident qi (?l$i)
      by (rule line-through-K2-intersect-S-again)
```

```
  with someI-ex [of λ qi. qi ≠ ?p$i ∧ qi ∈ S ∧ proj2-incident qi (?l$i)]
  show ?q$i ≠ ?p$i ∧ ?q$i ∈ S ∧ proj2-incident (?q$i) (?l$i) by simp
qed
hence ?q$1 ≠ ?p$1 and proj2-incident (?q$1) (?l$1)
  and proj2-incident (?q$2) (?l$2)
  by fast+

let ?r = χ i. proj2-intersection (polar (?q$i)) (polar (?p$i))
let ?m = χ i. proj2-line-through (?a$i) (?r$i)
have ∀ i. proj2-incident (?a$i) (?m$i)
  by (simp add: proj2-line-through-incident)
hence proj2-incident (?a$1) (?m$1) and proj2-incident (?a$2) (?m$2)
  by fast+

have ∀ i. proj2-incident (?r$i) (?m$i)
  by (simp add: proj2-line-through-incident)
hence proj2-incident (?r$1) (?m$1) and proj2-incident (?r$2) (?m$2)
  by fast+

let ?s = χ i. ϵ si. si ≠ ?r$i ∧ si ∈ S ∧ proj2-incident si (?m$i)
have ∀ i. ?s$i ≠ ?r$i ∧ ?s$i ∈ S ∧ proj2-incident (?s$i) (?m$i)
proof
  fix i
  from ‹∀ i. ?a$i ∈ K2› have ?a$i ∈ K2 ..

  from ‹∀ i. proj2-incident (?a$i) (?m$i)›
  have proj2-incident (?a$i) (?m$i) ..
  with ‹?a$i ∈ K2›
  have ∃ si. si ≠ ?r$i ∧ si ∈ S ∧ proj2-incident si (?m$i)
    by (rule line-through-K2-intersect-S-again)
  with someI-ex [of λ si. si ≠ ?r$i ∧ si ∈ S ∧ proj2-incident si (?m$i)]
  show ?s$i ≠ ?r$i ∧ ?s$i ∈ S ∧ proj2-incident (?s$i) (?m$i) by simp
qed
hence ?s$1 ≠ ?r$1 and proj2-incident (?s$1) (?m$1)
  and proj2-incident (?s$2) (?m$2)
  by fast+

have ∀ i . ∀ u. proj2-incident u (?m$i) ⟶ ¬ (u = ?p$i ∨ u = ?q$i)
proof standard+
  fix i :: 2
  fix u :: proj2
  assume proj2-incident u (?m$i)
  assume u = ?p$i ∨ u = ?q$i

  from ‹∀ i. ?p$i ∈ S› have ?p$i ∈ S ..

  from ‹∀ i. ?q$i ≠ ?p$i ∧ ?q$i ∈ S ∧ proj2-incident (?q$i) (?l$i)›
  have ?q$i ≠ ?p$i and ?q$i ∈ S
    by simp-all
```

```
  from ‹?p$i ∈ S› and ‹?q$i ∈ S› and ‹u = ?p$i ∨ u = ?q$i›
  have u ∈ S by auto
  hence proj2-incident u (polar u)
    by (simp add: incident-own-polar-in-S)

  have proj2-incident (?r$i) (polar (?p$i))
    and proj2-incident (?r$i) (polar (?q$i))
    by (simp-all add: proj2-intersection-incident)
  with ‹u = ?p$i ∨ u = ?q$i›
  have proj2-incident (?r$i) (polar u) by auto

  from ‹∀ i. proj2-incident (?r$i) (?m$i)›
  have proj2-incident (?r$i) (?m$i) ..

  from ‹∀ i. proj2-incident (?a$i) (?m$i)›
  have proj2-incident (?a$i) (?m$i) ..

  from ‹∀ i. ?a$i ∈ K2› have ?a$i ∈ K2 ..

  have u ≠ ?r$i
  proof
    assume u = ?r$i
    with ‹proj2-incident (?r$i) (polar (?p$i))›
      and ‹proj2-incident (?r$i) (polar (?q$i))›
    have proj2-incident u (polar (?p$i))
      and proj2-incident u (polar (?q$i))
      by simp-all
    with ‹u ∈ S› and ‹?p$i ∈ S› and ‹?q$i ∈ S›
    have u = ?p$i and u = ?q$i
      by (simp-all add: point-in-S-polar-is-tangent)
    with ‹?q$i ≠ ?p$i› show False by simp
  qed
  with ‹proj2-incident (u) (polar u)›
    and ‹proj2-incident (?r$i) (polar u)›
    and ‹proj2-incident u (?m$i)›
    and ‹proj2-incident (?r$i) (?m$i)›
    and proj2-incident-unique
  have ?m$i = polar u by auto
  with ‹proj2-incident (?a$i) (?m$i)›
  have proj2-incident (?a$i) (polar u) by simp
  with ‹u ∈ S› and ‹?a$i ∈ K2› and tangent-not-through-K2
  show False by simp
qed

let ?H = χ i. ϵ Hi. is-K2-isometry Hi
  ∧ apply-cltn2 east Hi = ?q$i
  ∧ apply-cltn2 west Hi = ?p$i
  ∧ apply-cltn2 north Hi = ?s$i
```

```
  ∧ apply-cltn2 far-north Hi = ?r$i
have ∀ i. is-K2-isometry (?H$i)
  ∧ apply-cltn2 east (?H$i) = ?q$i
  ∧ apply-cltn2 west (?H$i) = ?p$i
  ∧ apply-cltn2 north (?H$i) = ?s$i
  ∧ apply-cltn2 far-north (?H$i) = ?r$i
proof
  fix i :: 2
  from ‹∀ i. ?p$i ∈ S› have ?p$i ∈ S ..

  from ‹∀ i. ?q$i ≠ ?p$i ∧ ?q$i ∈ S ∧ proj2-incident (?q$i) (?l$i)›
  have ?q$i ≠ ?p$i and ?q$i ∈ S
    by simp-all

  from ‹∀ i. ?s$i ≠ ?r$i ∧ ?s$i ∈ S ∧ proj2-incident (?s$i) (?m$i)›
  have ?s$i ∈ S and proj2-incident (?s$i) (?m$i) by simp-all
  from ‹proj2-incident (?s$i) (?m$i)›
    and ‹∀ i. ∀ u. proj2-incident u (?m$i) ⟶ ¬ (u = ?p$i ∨ u = ?q$i)›
  have ?s$i ∉ {?q$i, ?p$i} by fast
  with ‹?q$i ∈ S› and ‹?p$i ∈ S› and ‹?s$i ∈ S› and ‹?q$i ≠ ?p$i›
  have ∃ Hi. is-K2-isometry Hi
    ∧ apply-cltn2 east Hi = ?q$i
    ∧ apply-cltn2 west Hi = ?p$i
    ∧ apply-cltn2 north Hi = ?s$i
    ∧ apply-cltn2 far-north Hi = ?r$i
    by (simp add: statement65-special-case)
  with someI-ex [of λ Hi. is-K2-isometry Hi
    ∧ apply-cltn2 east Hi = ?q$i
    ∧ apply-cltn2 west Hi = ?p$i
    ∧ apply-cltn2 north Hi = ?s$i
    ∧ apply-cltn2 far-north Hi = ?r$i]
  show is-K2-isometry (?H$i)
    ∧ apply-cltn2 east (?H$i) = ?q$i
    ∧ apply-cltn2 west (?H$i) = ?p$i
    ∧ apply-cltn2 north (?H$i) = ?s$i
    ∧ apply-cltn2 far-north (?H$i) = ?r$i
    by simp
qed
hence is-K2-isometry (?H$1)
  and apply-cltn2 east (?H$1) = ?q$1
  and apply-cltn2 west (?H$1) = ?p$1
  and apply-cltn2 north (?H$1) = ?s$1
  and apply-cltn2 far-north (?H$1) = ?r$1
  and  is-K2-isometry (?H$2)
  and apply-cltn2 east (?H$2) = ?q$2
  and apply-cltn2 west (?H$2) = ?p$2
  and apply-cltn2 north (?H$2) = ?s$2
  and apply-cltn2 far-north (?H$2) = ?r$2
  by fast+
```

let *?J* = *cltn2-compose* (*cltn2-inverse* (*?H$1*)) (*?H$2*)
from ‹*is-K2-isometry* (*?H$1*)› **and** ‹*is-K2-isometry* (*?H$2*)›
have *is-K2-isometry ?J*
  by (*simp only*: *cltn2-inverse-is-K2-isometry cltn2-compose-is-K2-isometry*)

from ‹*apply-cltn2 west* (*?H$1*) = *?p$1*›
have *apply-cltn2 p1* (*cltn2-inverse* (*?H$1*)) = *west*
  by (*simp add*: *cltn2.act-inv-iff* [*simplified*])
with ‹*apply-cltn2 west* (*?H$2*) = *?p$2*›
have *apply-cltn2 p1 ?J* = *p2*
  by (*simp add*: *cltn2.act-act* [*simplified*, *symmetric*])

from ‹*apply-cltn2 east* (*?H$1*) = *?q$1*›
have *apply-cltn2* (*?q$1*) (*cltn2-inverse* (*?H$1*)) = *east*
  by (*simp add*: *cltn2.act-inv-iff* [*simplified*])
with ‹*apply-cltn2 east* (*?H$2*) = *?q$2*›
have *apply-cltn2* (*?q$1*) *?J* = *?q$2*
  by (*simp add*: *cltn2.act-act* [*simplified*, *symmetric*])
with ‹*?q$1* ≠ *?p$1*› **and** ‹*apply-cltn2 p1 ?J* = *p2*›
  and ‹*proj2-incident* (*?p$1*) (*?l$1*)›
  and ‹*proj2-incident* (*?q$1*) (*?l$1*)›
  and ‹*proj2-incident* (*?p$2*) (*?l$2*)›
  and ‹*proj2-incident* (*?q$2*) (*?l$2*)›
have *apply-cltn2-line* (*?l$1*) *?J* = (*?l$2*)
  by (*simp add*: *apply-cltn2-line-unique*)
moreover from ‹*proj2-incident* (*?a$1*) (*?l$1*)›
have *proj2-incident* (*apply-cltn2* (*?a$1*) *?J*) (*apply-cltn2-line* (*?l$1*) *?J*)
  by *simp*
ultimately have *proj2-incident* (*apply-cltn2* (*?a$1*) *?J*) (*?l$2*) **by** *simp*

from ‹*apply-cltn2 north* (*?H$1*) = *?s$1*›
have *apply-cltn2* (*?s$1*) (*cltn2-inverse* (*?H$1*)) = *north*
  by (*simp add*: *cltn2.act-inv-iff* [*simplified*])
with ‹*apply-cltn2 north* (*?H$2*) = *?s$2*›
have *apply-cltn2* (*?s$1*) *?J* = *?s$2*
  by (*simp add*: *cltn2.act-act* [*simplified*, *symmetric*])

from ‹*apply-cltn2 far-north* (*?H$1*) = *?r$1*›
have *apply-cltn2* (*?r$1*) (*cltn2-inverse* (*?H$1*)) = *far-north*
  by (*simp add*: *cltn2.act-inv-iff* [*simplified*])
with ‹*apply-cltn2 far-north* (*?H$2*) = *?r$2*›
have *apply-cltn2* (*?r$1*) *?J* = *?r$2*
  by (*simp add*: *cltn2.act-act* [*simplified*, *symmetric*])
with ‹*?s$1* ≠ *?r$1*› **and** ‹*apply-cltn2* (*?s$1*) *?J* = (*?s$2*)›
  and ‹*proj2-incident* (*?r$1*) (*?m$1*)›
  and ‹*proj2-incident* (*?s$1*) (*?m$1*)›
  and ‹*proj2-incident* (*?r$2*) (*?m$2*)›
  and ‹*proj2-incident* (*?s$2*) (*?m$2*)›

```
  have apply-cltn2-line (?m$1) ?J = (?m$2)
    by (simp add: apply-cltn2-line-unique)
  moreover from ‹proj2-incident (?a$1) (?m$1)›
  have proj2-incident (apply-cltn2 (?a$1) ?J) (apply-cltn2-line (?m$1) ?J)
    by simp
  ultimately have proj2-incident (apply-cltn2 (?a$1) ?J) (?m$2) by simp

  from ‹∀ i. ∀ u. proj2-incident u (?m$i) ⟶ ¬ (u = ?p$i ∨ u = ?q$i)›
  have ¬ proj2-incident (?p$2) (?m$2) by fast
  with ‹proj2-incident (?p$2) (?l$2)› have ?m$2 ≠ ?l$2 by auto
  with ‹proj2-incident (?a$2) (?l$2)›
    and ‹proj2-incident (?a$2) (?m$2)›
    and ‹proj2-incident (apply-cltn2 (?a$1) ?J) (?l$2)›
    and ‹proj2-incident (apply-cltn2 (?a$1) ?J) (?m$2)›
    and proj2-incident-unique
  have apply-cltn2 a1 ?J = a2 by auto
  with ‹is-K2-isometry ?J› and ‹apply-cltn2 p1 ?J = p2›
  show ∃ J. is-K2-isometry J ∧ apply-cltn2 a1 J = a2 ∧ apply-cltn2 p1 J = p2
    by auto
qed

lemma K2-isometry-swap:
  assumes a ∈ hyp2 and b ∈ hyp2
  shows ∃ J. is-K2-isometry J ∧ apply-cltn2 a J = b ∧ apply-cltn2 b J = a
proof −
  from ‹a ∈ hyp2› and ‹b ∈ hyp2›
  have a ∈ K2 and b ∈ K2 by simp-all

  let ?l = proj2-line-through a b
  have proj2-incident a ?l and proj2-incident b ?l
    by (rule proj2-line-through-incident)+
  from ‹a ∈ K2› and ‹proj2-incident a ?l›
    and line-through-K2-intersect-S-exactly-twice [of a ?l]
  obtain p and q where p ≠ q
    and p ∈ S and q ∈ S
    and proj2-incident p ?l and proj2-incident q ?l
    and ∀ r∈S. proj2-incident r ?l ⟶ r = p ∨ r = q
    by auto
  from ‹a ∈ K2› and ‹b ∈ K2› and ‹p ∈ S› and ‹q ∈ S›
    and statement66-existence [of a b p q]
  obtain J where is-K2-isometry J and apply-cltn2 a J = b
    and apply-cltn2 p J = q
    by auto
  from ‹apply-cltn2 a J = b› and ‹apply-cltn2 p J = q›
    and ‹proj2-incident b ?l› and ‹proj2-incident q ?l›
  have proj2-incident (apply-cltn2 a J) ?l
    and proj2-incident (apply-cltn2 p J) ?l
    by simp-all
```

```
  from ‹a ∈ K2› and ‹p ∈ S› have a ≠ p
    unfolding S-def and K2-def
    by auto
  with ‹proj2-incident a ?l›
    and ‹proj2-incident p ?l›
    and ‹proj2-incident (apply-cltn2 a J) ?l›
    and ‹proj2-incident (apply-cltn2 p J) ?l›
  have apply-cltn2-line ?l J = ?l by (simp add: apply-cltn2-line-unique)
  with ‹proj2-incident q ?l› and apply-cltn2-preserve-incident [of q J ?l]
  have proj2-incident (apply-cltn2 q J) ?l by simp

  from ‹q ∈ S› and ‹is-K2-isometry J›
  have apply-cltn2 q J ∈ S by (unfold is-K2-isometry-def) simp
  with ‹proj2-incident (apply-cltn2 q J) ?l›
    and ‹∀ r∈S. proj2-incident r ?l ⟶ r = p ∨ r = q›
  have apply-cltn2 q J = p ∨ apply-cltn2 q J = q by simp

  have apply-cltn2 q J ≠ q
  proof
    assume apply-cltn2 q J = q
    with ‹apply-cltn2 p J = q›
    have apply-cltn2 p J = apply-cltn2 q J by simp
    hence p = q by (rule apply-cltn2-injective [of p J q])
    with ‹p ≠ q› show False ..
  qed
  with ‹apply-cltn2 q J = p ∨ apply-cltn2 q J = q›
  have apply-cltn2 q J = p by simp
  with ‹p ≠ q›
    and ‹apply-cltn2 p J = q›
    and ‹proj2-incident p ?l›
    and ‹proj2-incident q ?l›
    and ‹proj2-incident a ?l›
    and statement55
  have apply-cltn2 (apply-cltn2 a J) J = a by simp
  with ‹apply-cltn2 a J = b› have apply-cltn2 b J = a by simp
  with ‹is-K2-isometry J› and ‹apply-cltn2 a J = b›
  show ∃ J. is-K2-isometry J ∧ apply-cltn2 a J = b ∧ apply-cltn2 b J = a
    by (simp add: exI [of - J])
qed

theorem hyp2-axiom1: ∀ a b. a b ≡K b a
proof standard+
  fix a b
  let ?a' = Rep-hyp2 a
  let ?b' = Rep-hyp2 b
  from Rep-hyp2 and K2-isometry-swap [of ?a' ?b']
  obtain J where is-K2-isometry J and apply-cltn2 ?a' J = ?b'
    and apply-cltn2 ?b' J = ?a'
    by auto
```

from ‹*apply-cltn2 ?a' J = ?b'*› **and** ‹*apply-cltn2 ?b' J = ?a'*›
have *hyp2-cltn2 a J = b* **and** *hyp2-cltn2 b J = a*
unfolding *hyp2-cltn2-def* **by** (*simp-all add*: *Rep-hyp2-inverse*)
with ‹*is-K2-isometry J*›
show $a\ b \equiv_K b\ a$
by (*unfold real-hyp2-C-def*) (*simp add*: *exI* [*of - J*])
qed

theorem *hyp2-axiom2*: $\forall\ a\ b\ p\ q\ r\ s.\ a\ b \equiv_K p\ q \wedge a\ b \equiv_K r\ s \longrightarrow p\ q \equiv_K r\ s$
proof *standard+*
fix *a b p q r s*
assume $a\ b \equiv_K p\ q \wedge a\ b \equiv_K r\ s$
then obtain *G* **and** *H* **where** *is-K2-isometry G* **and** *is-K2-isometry H*
and *hyp2-cltn2 a G = p* **and** *hyp2-cltn2 b G = q*
and *hyp2-cltn2 a H = r* **and** *hyp2-cltn2 b H = s*
by (*unfold real-hyp2-C-def*) *auto*
let *?J = cltn2-compose* (*cltn2-inverse G*) *H*
from ‹*is-K2-isometry G*› **have** *is-K2-isometry* (*cltn2-inverse G*)
by (*rule cltn2-inverse-is-K2-isometry*)
with ‹*is-K2-isometry H*›
have *is-K2-isometry ?J* **by** (*simp only*: *cltn2-compose-is-K2-isometry*)

from ‹*is-K2-isometry G*› **and** ‹*hyp2-cltn2 a G = p*› **and** ‹*hyp2-cltn2 b G = q*›
and *K2-isometry.act-inv-iff*
have *hyp2-cltn2 p* (*cltn2-inverse G*) = *a*
and *hyp2-cltn2 q* (*cltn2-inverse G*) = *b*
by *simp-all*
with ‹*hyp2-cltn2 a H = r*› **and** ‹*hyp2-cltn2 b H = s*›
and ‹*is-K2-isometry* (*cltn2-inverse G*)› **and** ‹*is-K2-isometry H*›
and *K2-isometry.act-act* [*symmetric*]
have *hyp2-cltn2 p ?J = r* **and** *hyp2-cltn2 q ?J = s* **by** *simp-all*
with ‹*is-K2-isometry ?J*›
show $p\ q \equiv_K r\ s$
by (*unfold real-hyp2-C-def*) (*simp add*: *exI* [*of - ?J*])
qed

theorem *hyp2-axiom3*: $\forall\ a\ b\ c.\ a\ b \equiv_K c\ c \longrightarrow a = b$
proof *standard+*
fix *a b c*
assume $a\ b \equiv_K c\ c$
then obtain *J* **where** *is-K2-isometry J*
and *hyp2-cltn2 a J = c* **and** *hyp2-cltn2 b J = c*
by (*unfold real-hyp2-C-def*) *auto*
from ‹*hyp2-cltn2 a J = c*› **and** ‹*hyp2-cltn2 b J = c*›
have *hyp2-cltn2 a J = hyp2-cltn2 b J* **by** *simp*

from ‹*is-K2-isometry J*›
have *apply-cltn2* (*Rep-hyp2 a*) *J* $\in$ *hyp2*

and *apply-cltn2* (*Rep-hyp2 b*) *J* ∈ *hyp2*
by (*rule apply-cltn2-Rep-hyp2*)+
with ⟨*hyp2-cltn2 a J* = *hyp2-cltn2 b J*⟩
have *apply-cltn2* (*Rep-hyp2 a*) *J* = *apply-cltn2* (*Rep-hyp2 b*) *J*
by (*unfold hyp2-cltn2-def*) (*simp add*: *Abs-hyp2-inject*)
hence *Rep-hyp2 a* = *Rep-hyp2 b* **by** (*rule apply-cltn2-injective*)
thus *a* = *b* **by** (*simp add*: *Rep-hyp2-inject*)
qed

interpretation *hyp2*: *tarski-first3 real-hyp2-C*
using *hyp2-axiom1* **and** *hyp2-axiom2* **and** *hyp2-axiom3*
by *unfold-locales*

8.7 Some lemmas about betweenness

lemma *S-at-edge*:
assumes *p* ∈ *S* **and** *q* ∈ *hyp2* ∪ *S* **and** *r* ∈ *hyp2* ∪ *S* **and** *proj2-Col p q r*
shows $B_{\mathbb{R}}$ (*cart2-pt p*) (*cart2-pt q*) (*cart2-pt r*)
∨ $B_{\mathbb{R}}$ (*cart2-pt p*) (*cart2-pt r*) (*cart2-pt q*)
(**is** $B_{\mathbb{R}}$ *?cp ?cq ?cr* ∨ -)
proof −
from ⟨*p* ∈ *S*⟩ **and** ⟨*q* ∈ *hyp2* ∪ *S*⟩ **and** ⟨*r* ∈ *hyp2* ∪ *S*⟩
have *z-non-zero p* **and** *z-non-zero q* **and** *z-non-zero r*
by (*simp-all add*: *hyp2-S-z-non-zero*)
with ⟨*proj2-Col p q r*⟩
have *real-euclid.Col ?cp ?cq ?cr* **by** (*simp add*: *proj2-Col-iff-euclid-cart2*)

with ⟨*z-non-zero p*⟩ **and** ⟨*z-non-zero q*⟩ **and** ⟨*z-non-zero r*⟩
have *proj2-pt ?cp* = *p* **and** *proj2-pt ?cq* = *q* **and** *proj2-pt ?cr* = *r*
by (*simp-all add*: *proj2-cart2*)
from ⟨*proj2-pt ?cp* = *p*⟩ **and** ⟨*p* ∈ *S*⟩
have *norm ?cp* = *1* **by** (*simp add*: *norm-eq-1-iff-in-S*)

from ⟨*proj2-pt ?cq* = *q*⟩ **and** ⟨*proj2-pt ?cr* = *r*⟩
and ⟨*q* ∈ *hyp2* ∪ *S*⟩ **and** ⟨*r* ∈ *hyp2* ∪ *S*⟩
have *norm ?cq* ≤ *1* **and** *norm ?cr* ≤ *1*
by (*simp-all add*: *norm-le-1-iff-in-hyp2-S*)

show $B_{\mathbb{R}}$ *?cp ?cq ?cr* ∨ $B_{\mathbb{R}}$ *?cp ?cr ?cq*
proof *cases*
assume $B_{\mathbb{R}}$ *?cr ?cp ?cq*
then obtain *k* **where** *k* ≥ *0* **and** *k* ≤ *1*
and *?cp* − *?cr* = *k* $*_R$ (*?cq* − *?cr*)
by (*unfold real-euclid-B-def*) *auto*
from ⟨*?cp* − *?cr* = *k* $*_R$ (*?cq* − *?cr*)⟩
have *?cp* = *k* $*_R$ *?cq* + (*1* − *k*) $*_R$ *?cr* **by** (*simp add*: *algebra-simps*)
with ⟨*norm ?cp* = *1*⟩ **have** *norm* (*k* $*_R$ *?cq* + (*1* − *k*) $*_R$ *?cr*) = *1* **by** *simp*
with *norm-triangle-ineq* [*of k* $*_R$ *?cq* (*1* − *k*) $*_R$ *?cr*]
have *norm* (*k* $*_R$ *?cq*) + *norm* ((*1* − *k*) $*_R$ *?cr*) ≥ *1* **by** *simp*

from ‹$k \geq 0$› **and** ‹$k \leq 1$›
have *norm* ($k *_R$ *?cq*) + *norm* ((*1* − k) $*_R$ *?cr*)
  = k ∗ *norm ?cq* + (*1* − k) ∗ *norm ?cr*
  by *simp*
with ‹*norm* ($k *_R$ *?cq*) + *norm* ((*1* − k) $*_R$ *?cr*) ≥ *1*›
have k ∗ *norm ?cq* + (*1* − k) ∗ *norm ?cr* ≥ *1* **by** *simp*

from ‹*norm ?cq* ≤ *1*› **and** ‹$k \geq 0$› **and** *mult-mono* [*of k k norm ?cq 1*]
have k ∗ *norm ?cq* ≤ k **by** *simp*

from ‹*norm ?cr* ≤ *1*› **and** ‹$k \leq 1$›
  and *mult-mono* [*of 1 − k 1 − k norm ?cr 1*]
have (*1* − k) ∗ *norm ?cr* ≤ *1* − k **by** *simp*
with ‹k ∗ *norm ?cq* ≤ k›
have k ∗ *norm ?cq* + (*1* − k) ∗ *norm ?cr* ≤ *1* **by** *simp*
with ‹k ∗ *norm ?cq* + (*1* − k) ∗ *norm ?cr* ≥ *1*›
have k ∗ *norm ?cq* + (*1* − k) ∗ *norm ?cr* = *1* **by** *simp*
with ‹k ∗ *norm ?cq* ≤ k› **have** (*1* − k) ∗ *norm ?cr* ≥ *1* − k **by** *simp*
with ‹(*1* − k) ∗ *norm ?cr* ≤ *1* − k› **have** (*1* − k) ∗ *norm ?cr* = *1* − k **by** *simp*
with ‹k ∗ *norm ?cq* + (*1* − k) ∗ *norm ?cr* = *1*› **have** k ∗ *norm ?cq* = k **by** *simp*

have *?cp* = *?cq* ∨ *?cq* = *?cr* ∨ *?cr* = *?cp*
proof *cases*
  assume k = *0* ∨ k = *1*
  with ‹*?cp* = $k *_R$ *?cq* + (*1* − k) $*_R$ *?cr*›
  show *?cp* = *?cq* ∨ *?cq* = *?cr* ∨ *?cr* = *?cp* **by** *auto*
next
  assume ¬ (k = *0* ∨ k = *1*)
  hence k ≠ *0* **and** k ≠ *1* **by** *simp-all*
  with ‹k ∗ *norm ?cq* = k› **and** ‹(*1* − k) ∗ *norm ?cr* = *1* − k›
  have *norm ?cq* = *1* **and** *norm ?cr* = *1* **by** *simp-all*
  with ‹*proj2-pt ?cq* = q› **and** ‹*proj2-pt ?cr* = r›
  have $q \in S$ **and** $r \in S$ **by** (*simp-all add*: *norm-eq-1-iff-in-S*)
  with ‹$p \in S$› **have** {p,q,r} ⊆ S **by** *simp*

  from ‹*proj2-Col p q r*›
  have *proj2-set-Col* {p,q,r} **by** (*simp add*: *proj2-Col-iff-set-Col*)
  with ‹{p,q,r} ⊆ S› **have** *card* {p,q,r} ≤ *2* **by** (*rule card-line-intersect-S*)

  have $p = q \vee q = r \vee r = p$
  proof (*rule ccontr*)
    assume ¬ ($p = q \vee q = r \vee r = p$)
    hence $p \neq q$ **and** $q \neq r$ **and** $r \neq p$ **by** *simp-all*
    from ‹$q \neq r$› **have** *card* {q,r} = *2* **by** *simp*
    with ‹$p \neq q$› **and** ‹$r \neq p$› **have** *card* {p,q,r} = *3* **by** *simp*
    with ‹*card* {p,q,r} ≤ *2*› **show** *False* **by** *simp*

    qed
    thus *?cp* = *?cq* ∨ *?cq* = *?cr* ∨ *?cr* = *?cp* **by** *auto*
  qed
  thus $B_{\mathbb{R}}$ *?cp ?cq ?cr* ∨ $B_{\mathbb{R}}$ *?cp ?cr ?cq*
    by (*auto simp add*: *real-euclid.th3-1 real-euclid.th3-2*)
next
  assume ¬ $B_{\mathbb{R}}$ *?cr ?cp ?cq*
  with ‹*real-euclid.Col ?cp ?cq ?cr*›
  show $B_{\mathbb{R}}$ *?cp ?cq ?cr* ∨ $B_{\mathbb{R}}$ *?cp ?cr ?cq*
    unfolding *real-euclid.Col-def*
    by (*auto simp add*: *real-euclid.th3-1 real-euclid.th3-2*)
  qed
qed

lemma *hyp2-in-middle*:
  assumes *p* ∈ *S* **and** *q* ∈ *S* **and** *r* ∈ *hyp2* ∪ *S* **and** *proj2-Col p q r*
  and *p* ≠ *q*
  shows $B_{\mathbb{R}}$ (*cart2-pt p*) (*cart2-pt r*) (*cart2-pt q*) (**is** $B_{\mathbb{R}}$ *?cp ?cr ?cq*)
proof (*rule ccontr*)
  assume ¬ $B_{\mathbb{R}}$ *?cp ?cr ?cq*
  hence ¬ $B_{\mathbb{R}}$ *?cq ?cr ?cp*
    by (*auto simp add*: *real-euclid.th3-2* [*of ?cq ?cr ?cp*])

  from ‹*p* ∈ *S*› **and** ‹*q* ∈ *S*› **and** ‹*r* ∈ *hyp2* ∪ *S*› **and** ‹*proj2-Col p q r*›
  have $B_{\mathbb{R}}$ *?cp ?cq ?cr* ∨ $B_{\mathbb{R}}$ *?cp ?cr ?cq* **by** (*simp add*: *S-at-edge*)
  with ‹¬ $B_{\mathbb{R}}$ *?cp ?cr ?cq*› **have** $B_{\mathbb{R}}$ *?cp ?cq ?cr* **by** *simp*

  from ‹*proj2-Col p q r*› **and** *proj2-Col-permute* **have** *proj2-Col q p r* **by** *fast*
  with ‹*q* ∈ *S*› **and** ‹*p* ∈ *S*› **and** ‹*r* ∈ *hyp2* ∪ *S*›
  have $B_{\mathbb{R}}$ *?cq ?cp ?cr* ∨ $B_{\mathbb{R}}$ *?cq ?cr ?cp* **by** (*simp add*: *S-at-edge*)
  with ‹¬ $B_{\mathbb{R}}$ *?cq ?cr ?cp*› **have** $B_{\mathbb{R}}$ *?cq ?cp ?cr* **by** *simp*
  with ‹$B_{\mathbb{R}}$ *?cp ?cq ?cr*› **have** *?cp* = *?cq* **by** (*rule real-euclid.th3-4*)
  hence *proj2-pt ?cp* = *proj2-pt ?cq* **by** *simp*

  from ‹*p* ∈ *S*› **and** ‹*q* ∈ *S*›
  have *z-non-zero p* **and** *z-non-zero q* **by** (*simp-all add*: *hyp2-S-z-non-zero*)
  hence *proj2-pt ?cp* = *p* **and** *proj2-pt ?cq* = *q* **by** (*simp-all add*: *proj2-cart2*)
  with ‹*proj2-pt ?cp* = *proj2-pt ?cq*› **have** *p* = *q* **by** *simp*
  with ‹*p* ≠ *q*› **show** *False* **..**
qed

lemma *hyp2-incident-in-middle*:
  assumes *p* ≠ *q* **and** *p* ∈ *S* **and** *q* ∈ *S* **and** *a* ∈ *hyp2* ∪ *S*
  and *proj2-incident p l* **and** *proj2-incident q l* **and** *proj2-incident a l*
  shows $B_{\mathbb{R}}$ (*cart2-pt p*) (*cart2-pt a*) (*cart2-pt q*)
proof −
  from ‹*proj2-incident p l*› **and** ‹*proj2-incident q l*› **and** ‹*proj2-incident a l*›
  have *proj2-Col p q a* **by** (*rule proj2-incident-Col*)
  from ‹*p* ∈ *S*› **and** ‹*q* ∈ *S*› **and** ‹*a* ∈ *hyp2* ∪ *S*› **and** *this* **and** ‹*p* ≠ *q*›

show $B_{\mathbb{R}}$ (*cart2-pt p*) (*cart2-pt a*) (*cart2-pt q*)
by (*rule hyp2-in-middle*)
qed

lemma *extend-to-S*:
assumes $p \in$ *hyp2* $\cup$ S **and** $q \in$ *hyp2* $\cup$ S
shows $\exists$ $r{\in}S$. $B_{\mathbb{R}}$ (*cart2-pt p*) (*cart2-pt q*) (*cart2-pt r*)
(**is** $\exists$ $r{\in}S$. $B_{\mathbb{R}}$ *?cp ?cq* (*cart2-pt r*))
proof *cases*
assume $q \in S$

have $B_{\mathbb{R}}$ *?cp ?cq ?cq* **by** (*rule real-euclid.th3-1*)
with ⟨$q \in S$⟩ **show** $\exists$ $r{\in}S$. $B_{\mathbb{R}}$ *?cp ?cq* (*cart2-pt r*) **by** *auto*
next
assume $q \notin S$
with ⟨$q \in$ *hyp2* $\cup$ S⟩ **have** $q \in$ *K2* **by** *simp*

let *?l* = *proj2-line-through p q*
have *proj2-incident p ?l* **and** *proj2-incident q ?l*
by (*rule proj2-line-through-incident*)+
from ⟨$q \in$ *K2*⟩ **and** ⟨*proj2-incident q ?l*⟩
and *line-through-K2-intersect-S-twice* [*of q ?l*]
obtain *s* **and** *t* **where** $s \neq t$ **and** $s \in S$ **and** $t \in S$
and *proj2-incident s ?l* **and** *proj2-incident t ?l*
by *auto*
let *?cs* = *cart2-pt s*
let *?ct* = *cart2-pt t*

from ⟨*proj2-incident s ?l*⟩
and ⟨*proj2-incident t ?l*⟩
and ⟨*proj2-incident p ?l*⟩
and ⟨*proj2-incident q ?l*⟩
have *proj2-Col s p q* **and** *proj2-Col t p q* **and** *proj2-Col s t q*
by (*simp-all add*: *proj2-incident-Col*)
from ⟨*proj2-Col s p q*⟩ **and** ⟨*proj2-Col t p q*⟩
and ⟨$s \in S$⟩ **and** ⟨$t \in S$⟩ **and** ⟨$p \in$ *hyp2* $\cup$ S⟩ **and** ⟨$q \in$ *hyp2* $\cup$ S⟩
have $B_{\mathbb{R}}$ *?cs ?cp ?cq* $\vee$ $B_{\mathbb{R}}$ *?cs ?cq ?cp* **and** $B_{\mathbb{R}}$ *?ct ?cp ?cq* $\vee$ $B_{\mathbb{R}}$ *?ct ?cq ?cp*
by (*simp-all add*: *S-at-edge*)
with *real-euclid.th3-2*
have $B_{\mathbb{R}}$ *?cq ?cp ?cs* $\vee$ $B_{\mathbb{R}}$ *?cp ?cq ?cs* **and** $B_{\mathbb{R}}$ *?cq ?cp ?ct* $\vee$ $B_{\mathbb{R}}$ *?cp ?cq ?ct*
by *fast*+

from ⟨$s \in S$⟩ **and** ⟨$t \in S$⟩ **and** ⟨$q \in$ *hyp2* $\cup$ S⟩ **and** ⟨*proj2-Col s t q*⟩ **and** ⟨$s \neq t$⟩
have $B_{\mathbb{R}}$ *?cs ?cq ?ct* **by** (*rule hyp2-in-middle*)
hence $B_{\mathbb{R}}$ *?ct ?cq ?cs* **by** (*rule real-euclid.th3-2*)

have $B_{\mathbb{R}}$ *?cp ?cq ?cs* $\vee$ $B_{\mathbb{R}}$ *?cp ?cq ?ct*
proof (*rule ccontr*)
assume $\neg$ ($B_{\mathbb{R}}$ *?cp ?cq ?cs* $\vee$ $B_{\mathbb{R}}$ *?cp ?cq ?ct*)

hence $\neg\ B_{\mathbb{R}}\ ?cp\ ?cq\ ?cs$ **and** $\neg\ B_{\mathbb{R}}\ ?cp\ ?cq\ ?ct$ **by** *simp-all*
with $\langle B_{\mathbb{R}}\ ?cq\ ?cp\ ?cs \vee B_{\mathbb{R}}\ ?cp\ ?cq\ ?cs\rangle$
and $\langle B_{\mathbb{R}}\ ?cq\ ?cp\ ?ct \vee B_{\mathbb{R}}\ ?cp\ ?cq\ ?ct\rangle$
have $B_{\mathbb{R}}\ ?cq\ ?cp\ ?cs$ **and** $B_{\mathbb{R}}\ ?cq\ ?cp\ ?ct$ **by** *simp-all*
from $\langle\neg\ B_{\mathbb{R}}\ ?cp\ ?cq\ ?cs\rangle$ **and** $\langle B_{\mathbb{R}}\ ?cq\ ?cp\ ?cs\rangle$ **have** $?cp \neq ?cq$ **by** *auto*
with $\langle B_{\mathbb{R}}\ ?cq\ ?cp\ ?cs\rangle$ **and** $\langle B_{\mathbb{R}}\ ?cq\ ?cp\ ?ct\rangle$
have $B_{\mathbb{R}}\ ?cq\ ?cs\ ?ct \vee B_{\mathbb{R}}\ ?cq\ ?ct\ ?cs$
by (*simp add*: *real-euclid-th5-1* [*of* $?cq\ ?cp\ ?cs\ ?ct$])
with $\langle B_{\mathbb{R}}\ ?cs\ ?cq\ ?ct\rangle$ **and** $\langle B_{\mathbb{R}}\ ?ct\ ?cq\ ?cs\rangle$
have $?cq = ?cs \vee ?cq = ?ct$ **by** (*auto simp add*: *real-euclid.th3-4*)
with $\langle q \in hyp2 \cup S\rangle$ **and** $\langle s \in S\rangle$ **and** $\langle t \in S\rangle$
have $q = s \vee q = t$ **by** (*auto simp add*: *hyp2-S-cart2-inj*)
with $\langle s \in S\rangle$ **and** $\langle t \in S\rangle$ **have** $q \in S$ **by** *auto*
with $\langle q \notin S\rangle$ **show** *False* **..**
qed
with $\langle s \in S\rangle$ **and** $\langle t \in S\rangle$ **show** $\exists\ r{\in}S.\ B_{\mathbb{R}}\ ?cp\ ?cq\ (cart2\text{-}pt\ r)$ **by** *auto*
qed

definition *endpoint-in-S* :: $proj2 \Rightarrow proj2 \Rightarrow proj2$ **where**
endpoint-in-S a b
$\triangleq \epsilon\ p.\ p{\in}S \wedge B_{\mathbb{R}}\ (cart2\text{-}pt\ a)\ (cart2\text{-}pt\ b)\ (cart2\text{-}pt\ p)$

lemma *endpoint-in-S*:
assumes $a \in hyp2 \cup S$ **and** $b \in hyp2 \cup S$
shows *endpoint-in-S a b* $\in S$ (**is** $?p \in S$)
and $B_{\mathbb{R}}\ (cart2\text{-}pt\ a)\ (cart2\text{-}pt\ b)\ (cart2\text{-}pt\ (endpoint\text{-}in\text{-}S\ a\ b))$
(**is** $B_{\mathbb{R}}\ ?ca\ ?cb\ ?cp$)
proof −
from $\langle a \in hyp2 \cup S\rangle$ **and** $\langle b \in hyp2 \cup S\rangle$ **and** *extend-to-S*
have $\exists\ p.\ p \in S \wedge B_{\mathbb{R}}\ ?ca\ ?cb\ (cart2\text{-}pt\ p)$ **by** *auto*
hence $?p \in S \wedge B_{\mathbb{R}}\ ?ca\ ?cb\ ?cp$
by (*unfold endpoint-in-S-def*) (*rule someI-ex*)
thus $?p \in S$ **and** $B_{\mathbb{R}}\ ?ca\ ?cb\ ?cp$ **by** *simp-all*
qed

lemma *endpoint-in-S-swap*:
assumes $a \neq b$ **and** $a \in hyp2 \cup S$ **and** $b \in hyp2 \cup S$
shows *endpoint-in-S a b* $\neq$ *endpoint-in-S b a* (**is** $?p \neq ?q$)
proof
let $?ca = cart2\text{-}pt\ a$
let $?cb = cart2\text{-}pt\ b$
let $?cp = cart2\text{-}pt\ ?p$
let $?cq = cart2\text{-}pt\ ?q$
from $\langle a \neq b\rangle$ **and** $\langle a \in hyp2 \cup S\rangle$ **and** $\langle b \in hyp2 \cup S\rangle$
have $B_{\mathbb{R}}\ ?ca\ ?cb\ ?cp$ **and** $B_{\mathbb{R}}\ ?cb\ ?ca\ ?cq$
by (*simp-all add*: *endpoint-in-S*)

assume $?p = ?q$
with $\langle B_{\mathbb{R}}\ ?cb\ ?ca\ ?cq\rangle$ **have** $B_{\mathbb{R}}\ ?cb\ ?ca\ ?cp$ **by** *simp*

with ‹$B_{\mathbb{R}}$ *?ca ?cb ?cp*› **have** *?ca* = *?cb* **by** (*rule real-euclid.th3-4*)
with ‹$a \in$ *hyp2* $\cup S$› **and** ‹$b \in$ *hyp2* $\cup S$› **have** $a = b$ **by** (*rule hyp2-S-cart2-inj*)
with ‹$a \neq b$› **show** *False* **..**
qed

lemma *endpoint-in-S-incident*:
assumes $a \neq b$ **and** $a \in$ *hyp2* $\cup S$ **and** $b \in$ *hyp2* $\cup S$
and *proj2-incident a l* **and** *proj2-incident b l*
shows *proj2-incident* (*endpoint-in-S a b*) *l* (**is** *proj2-incident ?p l*)
proof –
from ‹$a \in$ *hyp2* $\cup S$› **and** ‹$b \in$ *hyp2* $\cup S$›
have *?p* $\in S$ **and** $B_{\mathbb{R}}$ (*cart2-pt a*) (*cart2-pt b*) (*cart2-pt ?p*)
(**is** $B_{\mathbb{R}}$ *?ca ?cb ?cp*)
by (*rule endpoint-in-S*)+

from ‹$a \in$ *hyp2* $\cup S$› **and** ‹$b \in$ *hyp2* $\cup S$› **and** ‹*?p* $\in S$›
have *z-non-zero a* **and** *z-non-zero b* **and** *z-non-zero ?p*
by (*simp-all add*: *hyp2-S-z-non-zero*)

from ‹$B_{\mathbb{R}}$ *?ca ?cb ?cp*›
have *real-euclid.Col ?ca ?cb ?cp* **unfolding** *real-euclid.Col-def* **..**
with ‹*z-non-zero a*› **and** ‹*z-non-zero b*› **and** ‹*z-non-zero ?p*› **and** ‹$a \neq b$›
and ‹*proj2-incident a l*› **and** ‹*proj2-incident b l*›
show *proj2-incident ?p l* **by** (*rule euclid-Col-cart2-incident*)
qed

lemma *endpoints-in-S-incident-unique*:
assumes $a \neq b$ **and** $a \in$ *hyp2* $\cup S$ **and** $b \in$ *hyp2* $\cup S$ **and** $p \in S$
and *proj2-incident a l* **and** *proj2-incident b l* **and** *proj2-incident p l*
shows p = *endpoint-in-S a b* $\vee$ p = *endpoint-in-S b a*
(**is** p = *?q* $\vee$ p = *?r*)
proof –
from ‹$a \neq b$› **and** ‹$a \in$ *hyp2* $\cup S$› **and** ‹$b \in$ *hyp2* $\cup S$›
have *?q* $\neq$ *?r* **by** (*rule endpoint-in-S-swap*)

from ‹$a \in$ *hyp2* $\cup S$› **and** ‹$b \in$ *hyp2* $\cup S$›
have *?q* $\in S$ **and** *?r* $\in S$ **by** (*simp-all add*: *endpoint-in-S*)

from ‹$a \neq b$› **and** ‹$a \in$ *hyp2* $\cup S$› **and** ‹$b \in$ *hyp2* $\cup S$›
and ‹*proj2-incident a l*› **and** ‹*proj2-incident b l*›
have *proj2-incident ?q l* **and** *proj2-incident ?r l*
by (*simp-all add*: *endpoint-in-S-incident*)
with ‹*?q* $\neq$ *?r*› **and** ‹*?q* $\in S$› **and** ‹*?r* $\in S$› **and** ‹$p \in S$› **and** ‹*proj2-incident p l*›
show p = *?q* $\vee$ p = *?r* **by** (*simp add*: *line-S-two-intersections-only*)
qed

lemma *endpoint-in-S-unique*:
assumes $a \neq b$ **and** $a \in$ *hyp2* $\cup S$ **and** $b \in$ *hyp2* $\cup S$ **and** $p \in S$
and $B_{\mathbb{R}}$ (*cart2-pt a*) (*cart2-pt b*) (*cart2-pt p*) (**is** $B_{\mathbb{R}}$ *?ca ?cb ?cp*)

shows *p* = *endpoint-in-S a b* (**is** *p* = *?q*)
proof (*rule ccontr*)
from ‹*a* ∈ *hyp2* ∪ *S*› **and** ‹*b* ∈ *hyp2* ∪ *S*› **and** ‹*p* ∈ *S*›
have *z-non-zero a* **and** *z-non-zero b* **and** *z-non-zero p*
by (*simp-all add*: *hyp2-S-z-non-zero*)
with ‹$B_{\mathbb{R}}$ *?ca ?cb ?cp*› **and** *euclid-B-cart2-common-line* [*of a b p*]
obtain *l* **where**
proj2-incident a l **and** *proj2-incident b l* **and** *proj2-incident p l*
by *auto*
with ‹*a* ≠ *b*› **and** ‹*a* ∈ *hyp2* ∪ *S*› **and** ‹*b* ∈ *hyp2* ∪ *S*› **and** ‹*p* ∈ *S*›
have *p* = *?q* ∨ *p* = *endpoint-in-S b a* (**is** *p* = *?q* ∨ *p* = *?r*)
by (*rule endpoints-in-S-incident-unique*)

assume *p* ≠ *?q*
with ‹*p* = *?q* ∨ *p* = *?r*› **have** *p* = *?r* **by** *simp*
with ‹*b* ∈ *hyp2* ∪ *S*› **and** ‹*a* ∈ *hyp2* ∪ *S*›
have $B_{\mathbb{R}}$ *?cb ?ca ?cp* **by** (*simp add*: *endpoint-in-S*)
with ‹$B_{\mathbb{R}}$ *?ca ?cb ?cp*› **have** *?ca* = *?cb* **by** (*rule real-euclid.th3-4*)
with ‹*a* ∈ *hyp2* ∪ *S*› **and** ‹*b* ∈ *hyp2* ∪ *S*› **have** *a* = *b* **by** (*rule hyp2-S-cart2-inj*)
with ‹*a* ≠ *b*› **show** *False* **..**
qed

lemma *between-hyp2-S*:
assumes *p* ∈ *hyp2* ∪ *S* **and** *r* ∈ *hyp2* ∪ *S* **and** *k* ≥ *0* **and** *k* ≤ *1*
shows *proj2-pt* (*k* $*_R$ (*cart2-pt r*) + (*1* − *k*) $*_R$ (*cart2-pt p*)) ∈ *hyp2* ∪ *S*
(**is** *proj2-pt ?cq* ∈ -)
proof −
let *?cp* = *cart2-pt p*
let *?cr* = *cart2-pt r*
let *?q* = *proj2-pt ?cq*
from ‹*p* ∈ *hyp2* ∪ *S*› **and** ‹*r* ∈ *hyp2* ∪ *S*›
have *z-non-zero p* **and** *z-non-zero r* **by** (*simp-all add*: *hyp2-S-z-non-zero*)
hence *proj2-pt ?cp* = *p* **and** *proj2-pt ?cr* = *r* **by** (*simp-all add*: *proj2-cart2*)
with ‹*p* ∈ *hyp2* ∪ *S*› **and** ‹*r* ∈ *hyp2* ∪ *S*›
have *norm ?cp* ≤ *1* **and** *norm ?cr* ≤ *1*
by (*simp-all add*: *norm-le-1-iff-in-hyp2-S*)

from ‹*k* ≥ *0*› **and** ‹*k* ≤ *1*›
and *norm-triangle-ineq* [*of k* $*_R$ *?cr* (*1* − *k*) $*_R$ *?cp*]
have *norm ?cq* ≤ *k* ∗ *norm ?cr* + (*1* − *k*) ∗ *norm ?cp* **by** *simp*

from ‹*k* ≥ *0*› **and** ‹*norm ?cr* ≤ *1*› **and** *mult-mono* [*of k k norm ?cr 1*]
have *k* ∗ *norm ?cr* ≤ *k* **by** *simp*

from ‹*k* ≤ *1*› **and** ‹*norm ?cp* ≤ *1*›
and *mult-mono* [*of 1* − *k 1* − *k norm ?cp 1*]
have (*1* − *k*) ∗ *norm ?cp* ≤ *1* − *k* **by** *simp*
with ‹*norm ?cq* ≤ *k* ∗ *norm ?cr* + (*1* − *k*) ∗ *norm ?cp*› **and** ‹*k* ∗ *norm ?cr* ≤ *k*›

have *norm ?cq* $\leq$ *1* **by** *simp*
thus *?q* $\in$ *hyp2* $\cup$ *S* **by** (*simp add*: *norm-le-1-iff-in-hyp2-S*)
qed

8.8 The Klein–Beltrami model satisfies axiom 4

definition *expansion-factor* :: *proj2* $\Rightarrow$ *cltn2* $\Rightarrow$ *real* **where**
expansion-factor p J $\triangleq$ (*cart2-append1 p v*∗ *cltn2-rep J*)\$*3*

lemma *expansion-factor*:
assumes *p* $\in$ *hyp2* $\cup$ *S* **and** *is-K2-isometry J*
shows *expansion-factor p J* $\neq$ *0*
and *cart2-append1 p v*∗ *cltn2-rep J*
= *expansion-factor p J* $*_R$ *cart2-append1* (*apply-cltn2 p J*)
proof −
from ⟨*p* $\in$ *hyp2* $\cup$ *S*⟩ **and** ⟨*is-K2-isometry J*⟩
have *z-non-zero* (*apply-cltn2 p J*) **by** (*rule is-K2-isometry-z-non-zero*)

from ⟨*p* $\in$ *hyp2* $\cup$ *S*⟩ **and** ⟨*is-K2-isometry J*⟩
and *cart2-append1-apply-cltn2*
obtain *k* **where** *k* $\neq$ *0*
and *cart2-append1 p v*∗ *cltn2-rep J* = *k* $*_R$ *cart2-append1* (*apply-cltn2 p J*)
by *auto*
from ⟨*cart2-append1 p v*∗ *cltn2-rep J* = *k* $*_R$ *cart2-append1* (*apply-cltn2 p J*)⟩
and ⟨*z-non-zero* (*apply-cltn2 p J*)⟩
have *expansion-factor p J* = *k*
by (*unfold expansion-factor-def*) (*simp add*: *cart2-append1-z*)
with ⟨*k* $\neq$ *0*⟩
and ⟨*cart2-append1 p v*∗ *cltn2-rep J* = *k* $*_R$ *cart2-append1* (*apply-cltn2 p J*)⟩
show *expansion-factor p J* $\neq$ *0*
and *cart2-append1 p v*∗ *cltn2-rep J*
= *expansion-factor p J* $*_R$ *cart2-append1* (*apply-cltn2 p J*)
by *simp-all*
qed

lemma *expansion-factor-linear-apply-cltn2*:
assumes *p* $\in$ *hyp2* $\cup$ *S* **and** *q* $\in$ *hyp2* $\cup$ *S* **and** *r* $\in$ *hyp2* $\cup$ *S*
and *is-K2-isometry J*
and *cart2-pt r* = *k* $*_R$ *cart2-pt p* + (*1* − *k*) $*_R$ *cart2-pt q*
shows *expansion-factor r J* $*_R$ *cart2-append1* (*apply-cltn2 r J*)
= (*k* ∗ *expansion-factor p J*) $*_R$ *cart2-append1* (*apply-cltn2 p J*)
+ ((*1* − *k*) ∗ *expansion-factor q J*) $*_R$ *cart2-append1* (*apply-cltn2 q J*)
(**is** *?er* $*_R$ - = (*k* ∗ *?ep*) $*_R$ - + ((*1* − *k*) ∗ *?eq*) $*_R$ -)
proof −
let *?cp* = *cart2-pt p*
let *?cq* = *cart2-pt q*
let *?cr* = *cart2-pt r*
let *?cp1* = *cart2-append1 p*
let *?cq1* = *cart2-append1 q*

let *?cr1* = *cart2-append1 r*
let *?repJ* = *cltn2-rep J*
from ‹*p* ∈ *hyp2* ∪ *S*› **and** ‹*q* ∈ *hyp2* ∪ *S*› **and** ‹*r* ∈ *hyp2* ∪ *S*›
have *z-non-zero p* **and** *z-non-zero q* **and** *z-non-zero r*
by (*simp-all add*: *hyp2-S-z-non-zero*)

from ‹*?cr* = $k *_R$ *?cp* + (*1* − *k*) $*_R$ *?cq*›
have *vector2-append1 ?cr*
= $k *_R$ *vector2-append1 ?cp* + (*1* − *k*) $*_R$ *vector2-append1 ?cq*
by (*unfold vector2-append1-def vector-def*) (*simp add*: *vec-eq-iff*)
with ‹*z-non-zero p*› **and** ‹*z-non-zero q*› **and** ‹*z-non-zero r*›
have *?cr1* = $k *_R$ *?cp1* + (*1* − *k*) $*_R$ *?cq1* **by** (*simp add*: *cart2-append1*)
hence *?cr1 v*∗ *?repJ* = $k *_R$ (*?cp1 v*∗ *?repJ*) + (*1* − *k*) $*_R$ (*?cq1 v*∗ *?repJ*)
by (*simp add*: *vector-matrix-left-distrib scaleR-vector-matrix-assoc* [*symmetric*])
with ‹*p* ∈ *hyp2* ∪ *S*› **and** ‹*q* ∈ *hyp2* ∪ *S*› **and** ‹*r* ∈ *hyp2* ∪ *S*›
and ‹*is-K2-isometry J*›
show *?er* $*_R$ *cart2-append1* (*apply-cltn2 r J*)
= (*k* ∗ *?ep*) $*_R$ *cart2-append1* (*apply-cltn2 p J*)
+ ((*1* − *k*) ∗ *?eq*) $*_R$ *cart2-append1* (*apply-cltn2 q J*)
by (*simp add*: *expansion-factor*)
qed

lemma *expansion-factor-linear*:
assumes *p* ∈ *hyp2* ∪ *S* **and** *q* ∈ *hyp2* ∪ *S* **and** *r* ∈ *hyp2* ∪ *S*
and *is-K2-isometry J*
and *cart2-pt r* = $k *_R$ *cart2-pt p* + (*1* − *k*) $*_R$ *cart2-pt q*
shows *expansion-factor r J*
= *k* ∗ *expansion-factor p J* + (*1* − *k*) ∗ *expansion-factor q J*
(**is** *?er* = *k* ∗ *?ep* + (*1* − *k*) ∗ *?eq*)
proof −
from ‹*p* ∈ *hyp2* ∪ *S*› **and** ‹*q* ∈ *hyp2* ∪ *S*› **and** ‹*r* ∈ *hyp2* ∪ *S*›
and ‹*is-K2-isometry J*›
have *z-non-zero* (*apply-cltn2 p J*)
and *z-non-zero* (*apply-cltn2 q J*)
and *z-non-zero* (*apply-cltn2 r J*)
by (*simp-all add*: *is-K2-isometry-z-non-zero*)

from ‹*p* ∈ *hyp2* ∪ *S*› **and** ‹*q* ∈ *hyp2* ∪ *S*› **and** ‹*r* ∈ *hyp2* ∪ *S*›
and ‹*is-K2-isometry J*›
and ‹*cart2-pt r* = $k *_R$ *cart2-pt p* + (*1* − *k*) $*_R$ *cart2-pt q*›
have *?er* $*_R$ *cart2-append1* (*apply-cltn2 r J*)
= (*k* ∗ *?ep*) $*_R$ *cart2-append1* (*apply-cltn2 p J*)
+ ((*1* − *k*) ∗ *?eq*) $*_R$ *cart2-append1* (*apply-cltn2 q J*)
by (*rule expansion-factor-linear-apply-cltn2*)
hence (*?er* $*_R$ *cart2-append1* (*apply-cltn2 r J*))$*3*
= ((*k* ∗ *?ep*) $*_R$ *cart2-append1* (*apply-cltn2 p J*)
+ ((*1* − *k*) ∗ *?eq*) $*_R$ *cart2-append1* (*apply-cltn2 q J*))$*3*
by *simp*
with ‹*z-non-zero* (*apply-cltn2 p J*)›

and ‹*z-non-zero* (*apply-cltn2 q J*)›
and ‹*z-non-zero* (*apply-cltn2 r J*)›
show *?er* = *k* ∗ *?ep* + (*1* − *k*) ∗ *?eq* **by** (*simp add*: *cart2-append1-z*)
qed

lemma *expansion-factor-sgn-invariant*:
assumes *p* ∈ *hyp2* ∪ *S* **and** *q* ∈ *hyp2* ∪ *S* **and** *is-K2-isometry J*
shows *sgn* (*expansion-factor p J*) = *sgn* (*expansion-factor q J*)
(**is** *sgn ?ep* = *sgn ?eq*)
proof (*rule ccontr*)
assume *sgn ?ep* ≠ *sgn ?eq*

from ‹*p* ∈ *hyp2* ∪ *S*› **and** ‹*q* ∈ *hyp2* ∪ *S*› **and** ‹*is-K2-isometry J*›
have *?ep* ≠ *0* **and** *?eq* ≠ *0* **by** (*simp-all add*: *expansion-factor*)
hence *sgn ?ep* ∈ {−*1*,*1*} **and** *sgn ?eq* ∈ {−*1*,*1*}
by (*simp-all add*: *sgn-real-def*)
with ‹*sgn ?ep* ≠ *sgn ?eq*› **have** *sgn ?ep* = − *sgn ?eq* **by** *auto*
hence *sgn ?ep* = *sgn* (−*?eq*) **by** (*subst sgn-minus*)
with *sgn-plus* [*of ?ep* −*?eq*]
have *sgn* (*?ep* − *?eq*) = *sgn ?ep* **by** (*simp add*: *algebra-simps*)
with ‹*sgn ?ep* ∈ {−*1*,*1*}› **have** *?ep* − *?eq* ≠ *0* **by** (*auto simp add*: *sgn-real-def*)

let *?k* = −*?eq* / (*?ep* − *?eq*)
from ‹*sgn* (*?ep* − *?eq*) = *sgn ?ep*› **and** ‹*sgn ?ep* = *sgn* (−*?eq*)›
have *sgn* (*?ep* − *?eq*) = *sgn* (−*?eq*) **by** *simp*
with ‹*?ep* − *?eq* ≠ *0*› **and** *sgn-div* [*of ?ep* − *?eq* −*?eq*]
have *?k* > *0* **by** *simp*

from ‹*?ep* − *?eq* ≠ *0*›
have *1* − *?k* = *?ep* / (*?ep* − *?eq*) **by** (*simp add*: *field-simps*)
with ‹*sgn* (*?ep* − *?eq*) = *sgn ?ep*› **and** ‹*?ep* − *?eq* ≠ *0*›
have *1* − *?k* > *0* **by** (*simp add*: *sgn-div*)
hence *?k* < *1* **by** *simp*

let *?cp* = *cart2-pt p*
let *?cq* = *cart2-pt q*
let *?cr* = *?k* $*_R$ *?cp* + (*1* − *?k*) $*_R$ *?cq*
let *?r* = *proj2-pt ?cr*
let *?er* = *expansion-factor ?r J*
have *cart2-pt ?r* = *?cr* **by** (*rule cart2-proj2*)

from ‹*p* ∈ *hyp2* ∪ *S*› **and** ‹*q* ∈ *hyp2* ∪ *S*› **and** ‹*?k* > *0*› **and** ‹*?k* < *1*›
and *between-hyp2-S* [*of q p ?k*]
have *?r* ∈ *hyp2* ∪ *S* **by** *simp*
with ‹*p* ∈ *hyp2* ∪ *S*› **and** ‹*q* ∈ *hyp2* ∪ *S*› **and** ‹*is-K2-isometry J*›
and ‹*cart2-pt ?r* = *?cr*›
and *expansion-factor-linear* [*of p q ?r J ?k*]
have *?er* = *?k* ∗ *?ep* + (*1* − *?k*) ∗ *?eq* **by** *simp*
with ‹*?ep* − *?eq* ≠ *0*› **have** *?er* = *0* **by** (*simp add*: *field-simps*)

with ‹*?r* ∈ *hyp2* ∪ *S*› **and** ‹*is-K2-isometry J*›
show *False* **by** (*simp add*: *expansion-factor*)
qed

lemma *statement-63*:
assumes $p \in hyp2 \cup S$ **and** $q \in hyp2 \cup S$ **and** $r \in hyp2 \cup S$
and *is-K2-isometry J* **and** $B_{\mathbb{R}}$ (*cart2-pt p*) (*cart2-pt q*) (*cart2-pt r*)
shows $B_{\mathbb{R}}$
(*cart2-pt* (*apply-cltn2 p J*))
(*cart2-pt* (*apply-cltn2 q J*))
(*cart2-pt* (*apply-cltn2 r J*))
proof −
let *?cp* = *cart2-pt p*
let *?cq* = *cart2-pt q*
let *?cr* = *cart2-pt r*
let *?ep* = *expansion-factor p J*
let *?eq* = *expansion-factor q J*
let *?er* = *expansion-factor r J*
from ‹*q* ∈ *hyp2* ∪ *S*› **and** ‹*is-K2-isometry J*›
have *?eq* ≠ *0* **by** (*rule expansion-factor*)

from ‹*p* ∈ *hyp2* ∪ *S*› **and** ‹*q* ∈ *hyp2* ∪ *S*› **and** ‹*r* ∈ *hyp2* ∪ *S*›
and ‹*is-K2-isometry J*› **and** *expansion-factor-sgn-invariant*
have *sgn ?ep* = *sgn ?eq* **and** *sgn ?er* = *sgn ?eq* **by** *fast+*
with ‹*?eq* ≠ *0*›
have *?ep* / *?eq* > *0* **and** *?er* / *?eq* > *0* **by** (*simp-all add*: *sgn-div*)

from ‹$B_{\mathbb{R}}$ *?cp ?cq ?cr*›
obtain *k* **where** *k* ≥ *0* **and** *k* ≤ *1* **and** *?cq* = *k* $*_R$ *?cr* + (*1* − *k*) $*_R$ *?cp*
by (*unfold real-euclid-B-def*) (*auto simp add*: *algebra-simps*)

let *?c* = *k* ∗ *?er* / *?eq*
from ‹*k* ≥ *0*› **and** ‹*?er* / *?eq* > *0*› **and** *mult-nonneg-nonneg* [*of k ?er* / *?eq*]
have *?c* ≥ *0* **by** *simp*

from ‹*r* ∈ *hyp2* ∪ *S*› **and** ‹*p* ∈ *hyp2* ∪ *S*› **and** ‹*q* ∈ *hyp2* ∪ *S*›
and ‹*is-K2-isometry J*› **and** ‹*?cq* = *k* $*_R$ *?cr* + (*1* − *k*) $*_R$ *?cp*›
have *?eq* = *k* ∗ *?er* + (*1* − *k*) ∗ *?ep* **by** (*rule expansion-factor-linear*)
with ‹*?eq* ≠ *0*› **have** *1* − *?c* = (*1* − *k*) ∗ *?ep* / *?eq* **by** (*simp add*: *field-simps*)
with ‹*k* ≤ *1*› **and** ‹*?ep* / *?eq* > *0*›
and *mult-nonneg-nonneg* [*of 1* − *k ?ep* / *?eq*]
have *?c* ≤ *1* **by** *simp*

let *?pJ* = *apply-cltn2 p J*
let *?qJ* = *apply-cltn2 q J*
let *?rJ* = *apply-cltn2 r J*
let *?cpJ* = *cart2-pt ?pJ*
let *?cqJ* = *cart2-pt ?qJ*
let *?crJ* = *cart2-pt ?rJ*

let *?cpJ1* = *cart2-append1 ?pJ*
let *?cqJ1* = *cart2-append1 ?qJ*
let *?crJ1* = *cart2-append1 ?rJ*
from ‹$p \in hyp2 \cup S$› **and** ‹$q \in hyp2 \cup S$› **and** ‹$r \in hyp2 \cup S$›
and ‹*is-K2-isometry J*›
have *z-non-zero ?pJ* **and** *z-non-zero ?qJ* **and** *z-non-zero ?rJ*
by (*simp-all add*: *is-K2-isometry-z-non-zero*)

from ‹$r \in hyp2 \cup S$› **and** ‹$p \in hyp2 \cup S$› **and** ‹$q \in hyp2 \cup S$›
and ‹*is-K2-isometry J*› **and** ‹$?cq = k *_R ?cr + (1 - k) *_R ?cp$›
have $?eq *_R ?cqJ1 = (k * ?er) *_R ?crJ1 + ((1 - k) * ?ep) *_R ?cpJ1$
by (*rule expansion-factor-linear-apply-cltn2*)
hence $(1 / ?eq) *_R (?eq *_R ?cqJ1)$
$= (1 / ?eq) *_R ((k * ?er) *_R ?crJ1 + ((1 - k) * ?ep) *_R ?cpJ1)$ **by** *simp*
with ‹$1 - ?c = (1 - k) * ?ep / ?eq$› **and** ‹$?eq \neq 0$›
have $?cqJ1 = ?c *_R ?crJ1 + (1 - ?c) *_R ?cpJ1$
by (*simp add*: *scaleR-right-distrib*)
with ‹*z-non-zero ?pJ*› **and** ‹*z-non-zero ?qJ*› **and** ‹*z-non-zero ?rJ*›
have *vector2-append1 ?cqJ*
$= ?c *_R$ *vector2-append1 ?crJ* $+ (1 - ?c) *_R$ *vector2-append1 ?cpJ*
by (*simp add*: *cart2-append1*)
hence $?cqJ = ?c *_R ?crJ + (1 - ?c) *_R ?cpJ$
unfolding *vector2-append1-def* **and** *vector-def*
by (*simp add*: *vec-eq-iff forall-2 forall-3*)
with ‹$?c \geq 0$› **and** ‹$?c \leq 1$›
show $B_{\mathbb{R}}$ *?cpJ ?cqJ ?crJ*
by (*unfold real-euclid-B-def*) (*simp add*: *algebra-simps exI* [*of* - *?c*])
qed

theorem *hyp2-axiom4*: $\forall\ q\ a\ b\ c.\ \exists\ x.\ B_K\ q\ a\ x \wedge a\ x \equiv_K b\ c$
proof (*rule allI*)+
fix *q a b c* :: *hyp2*
let *?pq* = *Rep-hyp2 q*
let *?pa* = *Rep-hyp2 a*
let *?pb* = *Rep-hyp2 b*
let *?pc* = *Rep-hyp2 c*
have $?pq \in hyp2$ **and** $?pa \in hyp2$ **and** $?pb \in hyp2$ **and** $?pc \in hyp2$
by (*rule Rep-hyp2*)+
let *?cq* = *cart2-pt ?pq*
let *?ca* = *cart2-pt ?pa*
let *?cb* = *cart2-pt ?pb*
let *?cc* = *cart2-pt ?pc*
let $?pp = \epsilon\ p.\ p \in S \wedge B_{\mathbb{R}}$ *?cb ?cc* (*cart2-pt p*)
let *?cp* = *cart2-pt ?pp*
from ‹$?pb \in hyp2$› **and** ‹$?pc \in hyp2$› **and** *extend-to-S* [*of ?pb ?pc*]
and *someI-ex* [*of* $\lambda\ p.\ p \in S \wedge B_{\mathbb{R}}$ *?cb ?cc* (*cart2-pt p*)]
have $?pp \in S$ **and** $B_{\mathbb{R}}$ *?cb ?cc ?cp* **by** *auto*

let $?pr = \epsilon\ r.\ r \in S \wedge B_{\mathbb{R}}$ *?cq ?ca* (*cart2-pt r*)

let *?cr* = *cart2-pt ?pr*
from ‹*?pq* ∈ *hyp2*› **and** ‹*?pa* ∈ *hyp2*› **and** *extend-to-S* [*of ?pq ?pa*]
and *someI-ex* [*of* λ *r*. *r* ∈ *S* ∧ $B_{\mathbb{R}}$ *?cq ?ca* (*cart2-pt r*)]
have *?pr* ∈ *S* **and** $B_{\mathbb{R}}$ *?cq ?ca ?cr* **by** *auto*

from ‹*?pb* ∈ *hyp2*› **and** ‹*?pa* ∈ *hyp2*› **and** ‹*?pp* ∈ *S*› **and** ‹*?pr* ∈ *S*›
and *statement66-existence* [*of ?pb ?pa ?pp ?pr*]
obtain *J* **where** *is-K2-isometry J*
and *apply-cltn2 ?pb J* = *?pa* **and** *apply-cltn2 ?pp J* = *?pr*
by *auto*
let *?px* = *apply-cltn2 ?pc J*
let *?cx* = *cart2-pt ?px*
let *?x* = *Abs-hyp2 ?px*
from ‹*is-K2-isometry J*› **and** ‹*?pc* ∈ *hyp2*›
have *?px* ∈ *hyp2* **by** (*rule statement60-one-way*)
hence *Rep-hyp2 ?x* = *?px* **by** (*rule Abs-hyp2-inverse*)

from ‹*?pb* ∈ *hyp2*› **and** ‹*?pc* ∈ *hyp2*› **and** ‹*?pp* ∈ *S*› **and** ‹*is-K2-isometry J*›
and ‹$B_{\mathbb{R}}$ *?cb ?cc ?cp*› **and** *statement-63*
have $B_{\mathbb{R}}$ (*cart2-pt* (*apply-cltn2 ?pb J*)) *?cx* (*cart2-pt* (*apply-cltn2 ?pp J*))
by *simp*
with ‹*apply-cltn2 ?pb J* = *?pa*› **and** ‹*apply-cltn2 ?pp J* = *?pr*›
have $B_{\mathbb{R}}$ *?ca ?cx ?cr* **by** *simp*
with ‹$B_{\mathbb{R}}$ *?cq ?ca ?cr*› **have** $B_{\mathbb{R}}$ *?cq ?ca ?cx* **by** (*rule real-euclid.th3-5-1*)
with ‹*Rep-hyp2 ?x* = *?px*›
have B_K *q a ?x*
unfolding *real-hyp2-B-def* **and** *hyp2-rep-def*
by *simp*

have *Abs-hyp2 ?pa* = *a* **by** (*rule Rep-hyp2-inverse*)
with ‹*apply-cltn2 ?pb J* = *?pa*›
have *hyp2-cltn2 b J* = *a* **by** (*unfold hyp2-cltn2-def*) *simp*

have *hyp2-cltn2 c J* = *?x* **unfolding** *hyp2-cltn2-def* **..**
with ‹*is-K2-isometry J*› **and** ‹*hyp2-cltn2 b J* = *a*›
have *b c* $\equiv_K$ *a ?x*
by (*unfold real-hyp2-C-def*) (*simp add*: *exI* [*of* - *J*])
hence *a ?x* $\equiv_K$ *b c* **by** (*rule hyp2.th2-2*)
with ‹B_K *q a ?x*›
show ∃ *x*. B_K *q a x* ∧ *a x* $\equiv_K$ *b c* **by** (*simp add*: *exI* [*of* - *?x*])
qed

8.9 More betweenness theorems

lemma *hyp2-S-points-fix-line*:
assumes *a* ∈ *hyp2* **and** *p* ∈ *S* **and** *is-K2-isometry J*
and *apply-cltn2 a J* = *a* (**is** *?aJ* = *a*)
and *apply-cltn2 p J* = *p* (**is** *?pJ* = *p*)
and *proj2-incident a l* **and** *proj2-incident p l* **and** *proj2-incident b l*

shows *apply-cltn2 b J = b* (**is** *?bJ = b*)
proof −
let *?lJ = apply-cltn2-line l J*
from ‹*proj2-incident a l*› **and** ‹*proj2-incident p l*›
have *proj2-incident ?aJ ?lJ* **and** *proj2-incident ?pJ ?lJ* **by** *simp-all*
with ‹*?aJ = a*› **and** ‹*?pJ = p*›
have *proj2-incident a ?lJ* **and** *proj2-incident p ?lJ* **by** *simp-all*

from ‹*a ∈ hyp2*› ‹*proj2-incident a l*› **and** *line-through-K2-intersect-S-again* [*of a l*]
obtain *q* **where** *q ≠ p* **and** *q ∈ S* **and** *proj2-incident q l* **by** *auto*
let *?qJ = apply-cltn2 q J*

from ‹*a ∈ hyp2*› **and** ‹*p ∈ S*› **and** ‹*q ∈ S*›
have *a ≠ p* **and** *a ≠ q* **by** (*simp-all add*: *hyp2-S-not-equal*)

from ‹*a ≠ p*› **and** ‹*proj2-incident a l*› **and** ‹*proj2-incident p l*›
and ‹*proj2-incident a ?lJ*› **and** ‹*proj2-incident p ?lJ*›
and *proj2-incident-unique*
have *?lJ = l* **by** *auto*

from ‹*proj2-incident q l*› **have** *proj2-incident ?qJ ?lJ* **by** *simp*
with ‹*?lJ = l*› **have** *proj2-incident ?qJ l* **by** *simp*

from ‹*q ∈ S*› **and** ‹*is-K2-isometry J*›
have *?qJ ∈ S* **by** (*unfold is-K2-isometry-def*) *simp*
with ‹*q ≠ p*› **and** ‹*p ∈ S*› **and** ‹*q ∈ S*› **and** ‹*proj2-incident p l*›
and ‹*proj2-incident q l*› **and** ‹*proj2-incident ?qJ l*›
and *line-S-two-intersections-only*
have *?qJ = p ∨ ?qJ = q* **by** *simp*

have *?qJ = q*
proof (*rule ccontr*)
assume *?qJ ≠ q*
with ‹*?qJ = p ∨ ?qJ = q*› **have** *?qJ = p* **by** *simp*
with ‹*?pJ = p*› **have** *?qJ = ?pJ* **by** *simp*
with *apply-cltn2-injective* **have** *q = p* **by** *fast*
with ‹*q ≠ p*› **show** *False* **..**
qed
with ‹*q ≠ p*› **and** ‹*a ≠ p*› **and** ‹*a ≠ q*› **and** ‹*proj2-incident p l*›
and ‹*proj2-incident q l*› **and** ‹*proj2-incident a l*›
and ‹*?pJ = p*› **and** ‹*?aJ = a*› **and** ‹*proj2-incident b l*›
and *cltn2-three-point-line* [*of p q a l J b*]
show *?bJ = b* **by** *simp*
qed

lemma *K2-isometry-endpoint-in-S*:
assumes *a ≠ b* **and** *a ∈ hyp2 ∪ S* **and** *b ∈ hyp2 ∪ S* **and** *is-K2-isometry J*
shows *apply-cltn2* (*endpoint-in-S a b*) *J*

$= endpoint\text{-}in\text{-}S\ (apply\text{-}cltn2\ a\ J)\ (apply\text{-}cltn2\ b\ J)$
(**is** $?pJ = endpoint\text{-}in\text{-}S\ ?aJ\ ?bJ$)
proof −
let $?p = endpoint\text{-}in\text{-}S\ a\ b$

from ‹$a \neq b$› **and** *apply-cltn2-injective* **have** $?aJ \neq ?bJ$ **by** *fast*

from ‹$a \in hyp2 \cup S$› **and** ‹$b \in hyp2 \cup S$› **and** ‹*is-K2-isometry J*›
and *is-K2-isometry-hyp2-S*
have $?aJ \in hyp2 \cup S$ **and** $?bJ \in hyp2 \cup S$ **by** *simp-all*

let $?ca = cart2\text{-}pt\ a$
let $?cb = cart2\text{-}pt\ b$
let $?cp = cart2\text{-}pt\ ?p$
from ‹$a \in hyp2 \cup S$› **and** ‹$b \in hyp2 \cup S$›
have $?p \in S$ **and** $B_{\mathbb{R}}\ ?ca\ ?cb\ ?cp$ **by** (*rule endpoint-in-S*)+

from ‹$?p \in S$› **and** ‹*is-K2-isometry J*›
have $?pJ \in S$ **by** (*unfold is-K2-isometry-def*) *simp*

let $?caJ = cart2\text{-}pt\ ?aJ$
let $?cbJ = cart2\text{-}pt\ ?bJ$
let $?cpJ = cart2\text{-}pt\ ?pJ$
from ‹$a \in hyp2 \cup S$› **and** ‹$b \in hyp2 \cup S$› **and** ‹$?p \in S$› **and** ‹*is-K2-isometry J*›
and ‹$B_{\mathbb{R}}\ ?ca\ ?cb\ ?cp$› **and** *statement-63*
have $B_{\mathbb{R}}\ ?caJ\ ?cbJ\ ?cpJ$ **by** *simp*
with ‹$?aJ \neq ?bJ$› **and** ‹$?aJ \in hyp2 \cup S$› **and** ‹$?bJ \in hyp2 \cup S$› **and** ‹$?pJ \in S$›
show $?pJ = endpoint\text{-}in\text{-}S\ ?aJ\ ?bJ$ **by** (*rule endpoint-in-S-unique*)
qed

lemma *between-endpoint-in-S*:
assumes $a \neq b$ **and** $b \neq c$
and $a \in hyp2 \cup S$ **and** $b \in hyp2 \cup S$ **and** $c \in hyp2 \cup S$
and $B_{\mathbb{R}}\ (cart2\text{-}pt\ a)\ (cart2\text{-}pt\ b)\ (cart2\text{-}pt\ c)$ (**is** $B_{\mathbb{R}}\ ?ca\ ?cb\ ?cc$)
shows $endpoint\text{-}in\text{-}S\ a\ b = endpoint\text{-}in\text{-}S\ b\ c$ (**is** $?p = ?q$)
proof −
from ‹$b \neq c$› **and** ‹$b \in hyp2 \cup S$› **and** ‹$c \in hyp2 \cup S$› **and** *hyp2-S-cart2-inj*
have $?cb \neq ?cc$ **by** *auto*

let $?cq = cart2\text{-}pt\ ?q$
from ‹$b \in hyp2 \cup S$› **and** ‹$c \in hyp2 \cup S$›
have $?q \in S$ **and** $B_{\mathbb{R}}\ ?cb\ ?cc\ ?cq$ **by** (*rule endpoint-in-S*)+

from ‹$?cb \neq ?cc$› **and** ‹$B_{\mathbb{R}}\ ?ca\ ?cb\ ?cc$› **and** ‹$B_{\mathbb{R}}\ ?cb\ ?cc\ ?cq$›
have $B_{\mathbb{R}}\ ?ca\ ?cb\ ?cq$ **by** (*rule real-euclid.th3-7-2*)
with ‹$a \neq b$› **and** ‹$a \in hyp2 \cup S$› **and** ‹$b \in hyp2 \cup S$› **and** ‹$?q \in S$›
have $?q = ?p$ **by** (*rule endpoint-in-S-unique*)
thus $?p = ?q$ **..**
qed

lemma *hyp2-extend-segment-unique*:
  assumes $a \neq b$ **and** $B_K\ a\ b\ c$ **and** $B_K\ a\ b\ d$ **and** $b\ c \equiv_K b\ d$
  shows $c = d$
proof *cases*
  assume $b = c$
  with ‹$b\ c \equiv_K b\ d$› **show** $c = d$ **by** (*simp add*: *hyp2.A3-reversed*)
next
  assume $b \neq c$

  have $b \neq d$
  proof (*rule ccontr*)
    assume $\neg\ b \neq d$
    hence $b = d$ **by** *simp*
    with ‹$b\ c \equiv_K b\ d$› **have** $b\ c \equiv_K b\ b$ **by** *simp*
    hence $b = c$ **by** (*rule hyp2.A3'*)
    with ‹$b \neq c$› **show** *False* **..**
  qed
  with ‹$a \neq b$› **and** ‹$b \neq c$›
  have *Rep-hyp2 a* $\neq$ *Rep-hyp2 b* (**is** *?pa* $\neq$ *?pb*)
    and *Rep-hyp2 b* $\neq$ *Rep-hyp2 c* (**is** *?pb* $\neq$ *?pc*)
    and *Rep-hyp2 b* $\neq$ *Rep-hyp2 d* (**is** *?pb* $\neq$ *?pd*)
    by (*simp-all add*: *Rep-hyp2-inject*)

  have *?pa* $\in$ *hyp2* **and** *?pb* $\in$ *hyp2* **and** *?pc* $\in$ *hyp2* **and** *?pd* $\in$ *hyp2*
    by (*rule Rep-hyp2*)+

  let *?pp* = *endpoint-in-S ?pb ?pc*
  let *?ca* = *cart2-pt ?pa*
  let *?cb* = *cart2-pt ?pb*
  let *?cc* = *cart2-pt ?pc*
  let *?cd* = *cart2-pt ?pd*
  let *?cp* = *cart2-pt ?pp*
  from ‹*?pb* $\in$ *hyp2*› **and** ‹*?pc* $\in$ *hyp2*›
  have *?pp* $\in$ *S* **and** $B_{\mathbb{R}}$ *?cb ?cc ?cp* **by** (*simp-all add*: *endpoint-in-S*)

  from ‹$b\ c \equiv_K b\ d$›
  obtain *J* **where** *is-K2-isometry J*
    and *hyp2-cltn2 b J* = *b* **and** *hyp2-cltn2 c J* = *d*
    by (*unfold real-hyp2-C-def*) *auto*

  from ‹*hyp2-cltn2 b J* = *b*› **and** ‹*hyp2-cltn2 c J* = *d*›
  have *Rep-hyp2* (*hyp2-cltn2 b J*) = *?pb*
    and *Rep-hyp2* (*hyp2-cltn2 c J*) = *?pd*
    by *simp-all*
  with ‹*is-K2-isometry J*›
  have *apply-cltn2 ?pb J* = *?pb* **and** *apply-cltn2 ?pc J* = *?pd*
    by (*simp-all add*: *Rep-hyp2-cltn2*)

from ‹B_K *a b c*› **and** ‹B_K *a b d*›
have $B_{\mathbb{R}}$ *?ca ?cb ?cc* **and** $B_{\mathbb{R}}$ *?ca ?cb ?cd*
unfolding *real-hyp2-B-def* **and** *hyp2-rep-def* **.**

from ‹*?pb* $\neq$ *?pc*› **and** ‹*?pb* $\in$ *hyp2*› **and** ‹*?pc* $\in$ *hyp2*› **and** ‹*is-K2-isometry J*›
have *apply-cltn2 ?pp J*
= *endpoint-in-S* (*apply-cltn2 ?pb J*) (*apply-cltn2 ?pc J*)
by (*simp add*: *K2-isometry-endpoint-in-S*)
also from ‹*apply-cltn2 ?pb J* = *?pb*› **and** ‹*apply-cltn2 ?pc J* = *?pd*›
have . . . = *endpoint-in-S ?pb ?pd* **by** *simp*
also from ‹*?pa* $\neq$ *?pb*› **and** ‹*?pb* $\neq$ *?pd*›
and ‹*?pa* $\in$ *hyp2*› **and** ‹*?pb* $\in$ *hyp2*› **and** ‹*?pd* $\in$ *hyp2*› **and** ‹$B_{\mathbb{R}}$ *?ca ?cb ?cd*›
have . . . = *endpoint-in-S ?pa ?pb* **by** (*simp add*: *between-endpoint-in-S*)
also from ‹*?pa* $\neq$ *?pb*› **and** ‹*?pb* $\neq$ *?pc*›
and ‹*?pa* $\in$ *hyp2*› **and** ‹*?pb* $\in$ *hyp2*› **and** ‹*?pc* $\in$ *hyp2*› **and** ‹$B_{\mathbb{R}}$ *?ca ?cb ?cc*›
have . . . = *endpoint-in-S ?pb ?pc* **by** (*simp add*: *between-endpoint-in-S*)
finally have *apply-cltn2 ?pp J* = *?pp* **.**

from ‹*?pb* $\in$ *hyp2*› **and** ‹*?pc* $\in$ *hyp2*› **and** ‹*?pp* $\in$ *S*›
have *z-non-zero ?pb* **and** *z-non-zero ?pc* **and** *z-non-zero ?pp*
by (*simp-all add*: *hyp2-S-z-non-zero*)
with ‹$B_{\mathbb{R}}$ *?cb ?cc ?cp*› **and** *euclid-B-cart2-common-line* [*of ?pb ?pc ?pp*]
obtain *l* **where** *proj2-incident ?pb l* **and** *proj2-incident ?pp l*
and *proj2-incident ?pc l*
by *auto*
with ‹*?pb* $\in$ *hyp2*› **and** ‹*?pp* $\in$ *S*› **and** ‹*is-K2-isometry J*›
and ‹*apply-cltn2 ?pb J* = *?pb*› **and** ‹*apply-cltn2 ?pp J* = *?pp*›
have *apply-cltn2 ?pc J* = *?pc* **by** (*rule hyp2-S-points-fix-line*)
with ‹*apply-cltn2 ?pc J* = *?pd*› **have** *?pc* = *?pd* **by** *simp*
thus *c* = *d* **by** (*subst Rep-hyp2-inject* [*symmetric*])
qed

lemma *line-S-match-intersections*:
assumes $p \neq q$ **and** $r \neq s$ **and** $p \in S$ **and** $q \in S$ **and** $r \in S$ **and** $s \in S$
and *proj2-set-Col* $\{p,q,r,s\}$
shows $(p = r \wedge q = s) \vee (q = r \wedge p = s)$
proof −
from ‹*proj2-set-Col* $\{p,q,r,s\}$›
obtain *l* **where** *proj2-incident p l* **and** *proj2-incident q l*
and *proj2-incident r l* **and** *proj2-incident s l*
by (*unfold proj2-set-Col-def*) *auto*
with ‹$r \neq s$› **and** ‹$p \in S$› **and** ‹$q \in S$› **and** ‹$r \in S$› **and** ‹$s \in S$›
have $p = r \vee p = s$ **and** $q = r \vee q = s$
by (*simp-all add*: *line-S-two-intersections-only*)

show $(p = r \wedge q = s) \vee (q = r \wedge p = s)$
proof *cases*
assume $p = r$
with ‹$p \neq q$› **and** ‹$q = r \vee q = s$›

show $(p = r \wedge q = s) \vee (q = r \wedge p = s)$ **by** *simp*
next
assume $p \neq r$
with ‹$p = r \vee p = s$› **have** $p = s$ **by** *simp*
with ‹$p \neq q$› **and** ‹$q = r \vee q = s$›
show $(p = r \wedge q = s) \vee (q = r \wedge p = s)$ **by** *simp*
qed
qed

definition *are-endpoints-in-S* :: [*proj2*, *proj2*, *proj2*, *proj2*] $\Rightarrow$ *bool* **where**
are-endpoints-in-S p q a b
$\triangleq p \neq q \wedge p \in S \wedge q \in S \wedge a \in hyp2 \wedge b \in hyp2 \wedge$ *proj2-set-Col* $\{p,q,a,b\}$

lemma *are-endpoints-in-S'*:
assumes $p \neq q$ **and** $a \neq b$ **and** $p \in S$ **and** $q \in S$ **and** $a \in hyp2 \cup S$
and $b \in hyp2 \cup S$ **and** *proj2-set-Col* $\{p,q,a,b\}$
shows (p = *endpoint-in-S a b* $\wedge$ q = *endpoint-in-S b a*)
$\vee$ (q = *endpoint-in-S a b* $\wedge$ p = *endpoint-in-S b a*)
(**is** $(p = ?r \wedge q = ?s) \vee (q = ?r \wedge p = ?s)$)
proof −
from ‹$a \neq b$› **and** ‹$a \in hyp2 \cup S$› **and** ‹$b \in hyp2 \cup S$›
have $?r \neq ?s$ **by** (*simp add*: *endpoint-in-S-swap*)

from ‹$a \in hyp2 \cup S$› **and** ‹$b \in hyp2 \cup S$›
have $?r \in S$ **and** $?s \in S$ **by** (*simp-all add*: *endpoint-in-S*)

from ‹*proj2-set-Col* $\{p,q,a,b\}$›
obtain l **where** *proj2-incident p l* **and** *proj2-incident q l*
and *proj2-incident a l* **and** *proj2-incident b l*
by (*unfold proj2-set-Col-def*) *auto*

from ‹$a \neq b$› **and** ‹$a \in hyp2 \cup S$› **and** ‹$b \in hyp2 \cup S$› **and** ‹*proj2-incident a l*›
and ‹*proj2-incident b l*›
have *proj2-incident ?r l* **and** *proj2-incident ?s l*
by (*simp-all add*: *endpoint-in-S-incident*)
with ‹*proj2-incident p l*› **and** ‹*proj2-incident q l*›
have *proj2-set-Col* $\{p,q,?r,?s\}$
by (*unfold proj2-set-Col-def*) (*simp add*: *exI* [*of - l*])
with ‹$p \neq q$› **and** ‹$?r \neq ?s$› **and** ‹$p \in S$› **and** ‹$q \in S$› **and** ‹$?r \in S$› **and** ‹$?s \in S$›
show $(p = ?r \wedge q = ?s) \vee (q = ?r \wedge p = ?s)$
by (*rule line-S-match-intersections*)
qed

lemma *are-endpoints-in-S*:
assumes $a \neq b$ **and** *are-endpoints-in-S p q a b*
shows (p = *endpoint-in-S a b* $\wedge$ q = *endpoint-in-S b a*)
$\vee$ (q = *endpoint-in-S a b* $\wedge$ p = *endpoint-in-S b a*)
using *assms*

by (*unfold are-endpoints-in-S-def*) (*simp add*: *are-endpoints-in-S'*)

lemma *S-intersections-endpoints-in-S*:
assumes $a \neq 0$ **and** $b \neq 0$ **and** *proj2-abs* $a \neq$ *proj2-abs* b (**is** *?pa* $\neq$ *?pb*)
and *proj2-abs* $a \in$ *hyp2* **and** *proj2-abs* $b \in$ *hyp2* $\cup$ S
shows (*S-intersection1 a b* = *endpoint-in-S ?pa ?pb*
$\wedge$ *S-intersection2 a b* = *endpoint-in-S ?pb ?pa*)
$\vee$ (*S-intersection2 a b* = *endpoint-in-S ?pa ?pb*
$\wedge$ *S-intersection1 a b* = *endpoint-in-S ?pb ?pa*)
(**is** (*?pp* = *?pr* $\wedge$ *?pq* = *?ps*) $\vee$ (*?pq* = *?pr* $\wedge$ *?pp* = *?ps*))
proof −
from ‹$a \neq 0$› **and** ‹$b \neq 0$› **and** ‹*?pa* $\neq$ *?pb*› **and** ‹*?pa* $\in$ *hyp2*›
have *?pp* $\neq$ *?pq* **by** (*simp add*: *S-intersections-distinct*)

from ‹$a \neq 0$› **and** ‹$b \neq 0$› **and** ‹*?pa* $\neq$ *?pb*› **and** ‹*proj2-abs a* $\in$ *hyp2*›
have *?pp* $\in S$ **and** *?pq* $\in S$
by (*simp-all add*: *S-intersections-in-S*)

let *?l* = *proj2-line-through ?pa ?pb*
have *proj2-incident ?pa ?l* **and** *proj2-incident ?pb ?l*
by (*rule proj2-line-through-incident*)+
with ‹$a \neq 0$› **and** ‹$b \neq 0$› **and** ‹*?pa* $\neq$ *?pb*›
have *proj2-incident ?pp ?l* **and** *proj2-incident ?pq ?l*
by (*rule S-intersections-incident*)+
with ‹*proj2-incident ?pa ?l*› **and** ‹*proj2-incident ?pb ?l*›
have *proj2-set-Col* {*?pp*,*?pq*,*?pa*,*?pb*}
by (*unfold proj2-set-Col-def*) (*simp add*: *exI* [*of* - *?l*])
with ‹*?pp* $\neq$ *?pq*› **and** ‹*?pa* $\neq$ *?pb*› **and** ‹*?pp* $\in S$› **and** ‹*?pq* $\in S$› **and** ‹*?pa* $\in$
hyp2›
and ‹*?pb* $\in$ *hyp2* $\cup$ S›
show (*?pp* = *?pr* $\wedge$ *?pq* = *?ps*) $\vee$ (*?pq* = *?pr* $\wedge$ *?pp* = *?ps*)
by (*simp add*: *are-endpoints-in-S'*)
qed

lemma *between-endpoints-in-S*:
assumes $a \neq b$ **and** $a \in$ *hyp2* $\cup$ S **and** $b \in$ *hyp2* $\cup$ S
shows $B_{\mathbb{R}}$
(*cart2-pt* (*endpoint-in-S a b*)) (*cart2-pt a*) (*cart2-pt* (*endpoint-in-S b a*))
(**is** $B_{\mathbb{R}}$ *?cp ?ca ?cq*)
proof −
let *?cb* = *cart2-pt b*
from ‹$b \in$ *hyp2* $\cup$ S› **and** ‹$a \in$ *hyp2* $\cup$ S› **and** ‹$a \neq b$›
have *?cb* $\neq$ *?ca* **by** (*auto simp add*: *hyp2-S-cart2-inj*)

from ‹$a \in$ *hyp2* $\cup$ S› **and** ‹$b \in$ *hyp2* $\cup$ S›
have $B_{\mathbb{R}}$ *?ca ?cb ?cp* **and** $B_{\mathbb{R}}$ *?cb ?ca ?cq* **by** (*simp-all add*: *endpoint-in-S*)

from ‹$B_{\mathbb{R}}$ *?ca ?cb ?cp*› **have** $B_{\mathbb{R}}$ *?cp ?cb ?ca* **by** (*rule real-euclid.th3-2*)
with ‹*?cb* $\neq$ *?ca*› **and** ‹$B_{\mathbb{R}}$ *?cb ?ca ?cq*›

show $B_{\mathbb{R}}$ *?cp ?ca ?cq* **by** (*simp add*: *real-euclid.th3-7-1*)
qed

lemma *S-hyp2-S-cart2-append1*:
assumes $p \neq q$ **and** $p \in S$ **and** $q \in S$ **and** $a \in hyp2$
and *proj2-incident p l* **and** *proj2-incident q l* **and** *proj2-incident a l*
shows $\exists\ k.\ k > 0 \wedge k < 1$
$\wedge$ *cart2-append1 a* $=$ $k *_R$ *cart2-append1 q* $+$ $(1 - k) *_R$ *cart2-append1 p*
proof –
from ‹$p \in S$› **and** ‹$q \in S$› **and** ‹$a \in hyp2$›
have *z-non-zero p* **and** *z-non-zero q* **and** *z-non-zero a*
by (*simp-all add*: *hyp2-S-z-non-zero*)

from *assms*
have $B_{\mathbb{R}}$ (*cart2-pt p*) (*cart2-pt a*) (*cart2-pt q*) (**is** $B_{\mathbb{R}}$ *?cp ?ca ?cq*)
by (*simp add*: *hyp2-incident-in-middle*)

from ‹$p \in S$› **and** ‹$q \in S$› **and** ‹$a \in hyp2$›
have $a \neq p$ **and** $a \neq q$ **by** (*simp-all add*: *hyp2-S-not-equal*)

with ‹*z-non-zero p*› **and** ‹*z-non-zero a*› **and** ‹*z-non-zero q*›
and ‹$B_{\mathbb{R}}$ *?cp ?ca ?cq*›
show $\exists\ k.\ k > 0 \wedge k < 1$
$\wedge$ *cart2-append1 a* $=$ $k *_R$ *cart2-append1 q* $+$ $(1 - k) *_R$ *cart2-append1 p*
by (*rule cart2-append1-between-strict*)
qed

lemma *are-endpoints-in-S-swap-34*:
assumes *are-endpoints-in-S p q a b*
shows *are-endpoints-in-S p q b a*
proof –
have $\{p,q,b,a\} = \{p,q,a,b\}$ **by** *auto*
with ‹*are-endpoints-in-S p q a b*›
show *are-endpoints-in-S p q b a* **by** (*unfold are-endpoints-in-S-def*) *simp*
qed

lemma *proj2-set-Col-endpoints-in-S*:
assumes $a \neq b$ **and** $a \in hyp2 \cup S$ **and** $b \in hyp2 \cup S$
shows *proj2-set-Col* {*endpoint-in-S a b*, *endpoint-in-S b a*, *a*, *b*}
(**is** *proj2-set-Col* {*?p*,*?q*,*a*,*b*})
proof –
let *?l* = *proj2-line-through a b*
have *proj2-incident a ?l* **and** *proj2-incident b ?l*
by (*rule proj2-line-through-incident*)+
with ‹$a \neq b$› **and** ‹$a \in hyp2 \cup S$› **and** ‹$b \in hyp2 \cup S$›
have *proj2-incident ?p ?l* **and** *proj2-incident ?q ?l*
by (*simp-all add*: *endpoint-in-S-incident*)
with ‹*proj2-incident a ?l*› **and** ‹*proj2-incident b ?l*›
show *proj2-set-Col* {*?p*,*?q*,*a*,*b*}

by (*unfold proj2-set-Col-def*) (*simp add*: *exI* [*of* - *?l*])
qed

lemma *endpoints-in-S-are-endpoints-in-S*:
assumes $a \neq b$ **and** $a \in hyp2$ **and** $b \in hyp2$
shows *are-endpoints-in-S* (*endpoint-in-S a b*) (*endpoint-in-S b a*) *a b*
(**is** *are-endpoints-in-S ?p ?q a b*)
proof −
from ‹$a \neq b$› **and** ‹$a \in hyp2$› **and** ‹$b \in hyp2$›
have $?p \neq ?q$ **by** (*simp add*: *endpoint-in-S-swap*)

from ‹$a \in hyp2$› **and** ‹$b \in hyp2$›
have $?p \in S$ **and** $?q \in S$ **by** (*simp-all add*: *endpoint-in-S*)

from *assms*
have *proj2-set-Col* {*?p*,*?q*,*a*,*b*} **by** (*simp add*: *proj2-set-Col-endpoints-in-S*)
with ‹$?p \neq ?q$› **and** ‹$?p \in S$› **and** ‹$?q \in S$› **and** ‹$a \in hyp2$› **and** ‹$b \in hyp2$›
show *are-endpoints-in-S ?p ?q a b* **by** (*unfold are-endpoints-in-S-def*) *simp*
qed

lemma *endpoint-in-S-S-hyp2-distinct*:
assumes $p \in S$ **and** $a \in hyp2 \cup S$ **and** $p \neq a$
shows *endpoint-in-S p a* $\neq p$
proof
from ‹$p \neq a$› **and** ‹$p \in S$› **and** ‹$a \in hyp2 \cup S$›
have $B_{\mathbb{R}}$ (*cart2-pt p*) (*cart2-pt a*) (*cart2-pt* (*endpoint-in-S p a*))
by (*simp add*: *endpoint-in-S*)

assume *endpoint-in-S p a* $= p$
with ‹$B_{\mathbb{R}}$ (*cart2-pt p*) (*cart2-pt a*) (*cart2-pt* (*endpoint-in-S p a*))›
have *cart2-pt p* = *cart2-pt a* **by** (*simp add*: *real-euclid.A6'*)
with ‹$p \in S$› **and** ‹$a \in hyp2 \cup S$› **have** $p = a$ **by** (*simp add*: *hyp2-S-cart2-inj*)
with ‹$p \neq a$› **show** *False* **..**
qed

lemma *endpoint-in-S-S-strict-hyp2-distinct*:
assumes $p \in S$ **and** $a \in hyp2$
shows *endpoint-in-S p a* $\neq p$
proof −
from ‹$a \in hyp2$› **and** ‹$p \in S$›
have $p \neq a$ **by** (*rule hyp2-S-not-equal* [*symmetric*])
with *assms*
show *endpoint-in-S p a* $\neq p$ **by** (*simp add*: *endpoint-in-S-S-hyp2-distinct*)
qed

lemma *end-and-opposite-are-endpoints-in-S*:
assumes $a \in hyp2$ **and** $b \in hyp2$ **and** $p \in S$
and *proj2-incident a l* **and** *proj2-incident b l* **and** *proj2-incident p l*
shows *are-endpoints-in-S p* (*endpoint-in-S p b*) *a b*

(**is** *are-endpoints-in-S p ?q a b*)
proof −
from ‹$p \in S$› **and** ‹$b \in$ *hyp2*›
have $p \neq$ *?q* **by** (*rule endpoint-in-S-S-strict-hyp2-distinct* [*symmetric*])

from ‹$p \in S$› **and** ‹$b \in$ *hyp2*› **have** *?q* $\in S$ **by** (*simp add*: *endpoint-in-S*)

from ‹$b \in$ *hyp2*› **and** ‹$p \in S$›
have $p \neq b$ **by** (*rule hyp2-S-not-equal* [*symmetric*])
with ‹$p \in S$› **and** ‹$b \in$ *hyp2*› **and** ‹*proj2-incident p l*› **and** ‹*proj2-incident b l*›
have *proj2-incident ?q l* **by** (*simp add*: *endpoint-in-S-incident*)
with ‹*proj2-incident p l*› **and** ‹*proj2-incident a l*› **and** ‹*proj2-incident b l*›
have *proj2-set-Col* {*p*,*?q*,*a*,*b*}
by (*unfold proj2-set-Col-def*) (*simp add*: *exI* [*of - l*])
with ‹$p \neq$ *?q*› **and** ‹$p \in S$› **and** ‹*?q* $\in S$› **and** ‹$a \in$ *hyp2*› **and** ‹$b \in$ *hyp2*›
show *are-endpoints-in-S p ?q a b* **by** (*unfold are-endpoints-in-S-def*) *simp*
qed

lemma *real-hyp2-B-hyp2-cltn2*:
assumes *is-K2-isometry J* **and** B_K *a b c*
shows B_K (*hyp2-cltn2 a J*) (*hyp2-cltn2 b J*) (*hyp2-cltn2 c J*)
(**is** B_K *?aJ ?bJ ?cJ*)
proof −
from ‹B_K *a b c*›
have $B_{\mathbb{R}}$ (*hyp2-rep a*) (*hyp2-rep b*) (*hyp2-rep c*) **by** (*unfold real-hyp2-B-def*)
with ‹*is-K2-isometry J*›
have $B_{\mathbb{R}}$ (*cart2-pt* (*apply-cltn2* (*Rep-hyp2 a*) *J*))
(*cart2-pt* (*apply-cltn2* (*Rep-hyp2 b*) *J*))
(*cart2-pt* (*apply-cltn2* (*Rep-hyp2 c*) *J*))
by (*unfold hyp2-rep-def*) (*simp add*: *Rep-hyp2 statement-63*)
moreover from ‹*is-K2-isometry J*›
have *apply-cltn2* (*Rep-hyp2 a*) *J* $\in$ *hyp2*
and *apply-cltn2* (*Rep-hyp2 b*) *J* $\in$ *hyp2*
and *apply-cltn2* (*Rep-hyp2 c*) *J* $\in$ *hyp2*
by (*rule apply-cltn2-Rep-hyp2*)+
ultimately show B_K (*hyp2-cltn2 a J*) (*hyp2-cltn2 b J*) (*hyp2-cltn2 c J*)
unfolding *hyp2-cltn2-def* **and** *real-hyp2-B-def* **and** *hyp2-rep-def*
by (*simp add*: *Abs-hyp2-inverse*)
qed

lemma *real-hyp2-C-hyp2-cltn2*:
assumes *is-K2-isometry J*
shows *a b* $\equiv_K$ (*hyp2-cltn2 a J*) (*hyp2-cltn2 b J*) (**is** *a b* $\equiv_K$ *?aJ ?bJ*)
using *assms* **by** (*unfold real-hyp2-C-def*) (*simp add*: *exI* [*of - J*])

8.10 Perpendicularity

definition *M-perp* :: *proj2-line* $\Rightarrow$ *proj2-line* $\Rightarrow$ *bool* **where**
M-perp l m $\triangleq$ *proj2-incident* (*pole l*) *m*

lemma *M-perp-sym*:
  assumes *M-perp l m*
  shows *M-perp m l*
proof −
  from ⟨*M-perp l m*⟩ **have** *proj2-incident* (*pole l*) *m* **by** (*unfold M-perp-def*)
  hence *proj2-incident* (*pole m*) (*polar* (*pole l*)) **by** (*rule incident-pole-polar*)
  hence *proj2-incident* (*pole m*) *l* **by** (*simp add*: *polar-pole*)
  thus *M-perp m l* **by** (*unfold M-perp-def*)
qed

lemma *M-perp-to-compass*:
  assumes *M-perp l m* **and** $a \in$ *hyp2* **and** *proj2-incident a l*
  and $b \in$ *hyp2* **and** *proj2-incident b m*
  shows ∃ *J*. *is-K2-isometry J*
  ∧ *apply-cltn2-line equator J* = *l* ∧ *apply-cltn2-line meridian J* = *m*
proof −
  from ⟨$a \in K2$⟩ **and** ⟨*proj2-incident a l*⟩
    and *line-through-K2-intersect-S-twice* [*of a l*]
  obtain *p* **and** *q* **where** $p \neq q$ **and** $p \in S$ **and** $q \in S$
    and *proj2-incident p l* **and** *proj2-incident q l*
    by *auto*

  have ∃ *r*. $r \in S \wedge r \notin \{p,q\}$ ∧ *proj2-incident r m*
  proof *cases*
    assume *proj2-incident p m*

    from ⟨$b \in K2$⟩ **and** ⟨*proj2-incident b m*⟩
      and *line-through-K2-intersect-S-again* [*of b m*]
    obtain *r* **where** $r \in S$ **and** $r \neq p$ **and** *proj2-incident r m* **by** *auto*

    have $r \notin \{p,q\}$
    proof
      assume $r \in \{p,q\}$
      with ⟨$r \neq p$⟩ **have** $r = q$ **by** *simp*
      with ⟨*proj2-incident r m*⟩ **have** *proj2-incident q m* **by** *simp*
      with ⟨*proj2-incident p l*⟩ **and** ⟨*proj2-incident q l*⟩
        and ⟨*proj2-incident p m*⟩ **and** ⟨*proj2-incident q m*⟩ **and** ⟨$p \neq q$⟩
        and *proj2-incident-unique* [*of p l q m*]
      have $l = m$ **by** *simp*
      with ⟨*M-perp l m*⟩ **have** *M-perp l l* **by** *simp*
      hence *proj2-incident* (*pole l*) *l* (**is** *proj2-incident ?s l*)
        by (*unfold M-perp-def*)
      hence *proj2-incident ?s* (*polar ?s*) **by** (*subst polar-pole*)
      hence $?s \in S$ **by** (*simp add*: *incident-own-polar-in-S*)
      with ⟨$p \in S$⟩ **and** ⟨$q \in S$⟩ **and** ⟨*proj2-incident p l*⟩ **and** ⟨*proj2-incident q l*⟩
        and *point-in-S-polar-is-tangent* [*of ?s*]
      have $p = ?s$ **and** $q = ?s$ **by** (*auto simp add*: *polar-pole*)
      with ⟨$p \neq q$⟩ **show** *False* **by** *simp*

qed
with ‹$r \in S$› **and** ‹*proj2-incident r m*›
show $\exists\ r.\ r \in S \wedge r \notin \{p,q\} \wedge$ *proj2-incident r m*
by (*simp add*: *exI* [*of* - *r*])
next
assume $\neg$ *proj2-incident p m*

from ‹$b \in K2$› **and** ‹*proj2-incident b m*›
and *line-through-K2-intersect-S-again* [*of b m*]
obtain *r* **where** $r \in S$ **and** $r \neq q$ **and** *proj2-incident r m* **by** *auto*

from ‹$\neg$ *proj2-incident p m*› **and** ‹*proj2-incident r m*› **have** $r \neq p$ **by** *auto*
with ‹$r \in S$› **and** ‹$r \neq q$› **and** ‹*proj2-incident r m*›
show $\exists\ r.\ r \in S \wedge r \notin \{p,q\} \wedge$ *proj2-incident r m*
by (*simp add*: *exI* [*of* - *r*])
qed
then obtain *r* **where** $r \in S$ **and** $r \notin \{p,q\}$ **and** *proj2-incident r m* **by** *auto*

from ‹$p \in S$› **and** ‹$q \in S$› **and** ‹$r \in S$› **and** ‹$p \neq q$› **and** ‹$r \notin \{p,q\}$›
and *statement65-special-case* [*of p q r*]
obtain *J* **where** *is-K2-isometry J* **and** *apply-cltn2 east J* = *p*
and *apply-cltn2 west J* = *q* **and** *apply-cltn2 north J* = *r*
and *apply-cltn2 far-north J* = *proj2-intersection* (*polar p*) (*polar q*)
by *auto*

from ‹*apply-cltn2 east J* = *p*› **and** ‹*apply-cltn2 west J* = *q*›
and ‹*proj2-incident p l*› **and** ‹*proj2-incident q l*›
have *proj2-incident* (*apply-cltn2 east J*) *l*
and *proj2-incident* (*apply-cltn2 west J*) *l*
by *simp-all*
with *east-west-distinct* **and** *east-west-on-equator*
have *apply-cltn2-line equator J* = *l* **by** (*rule apply-cltn2-line-unique*)

from ‹*apply-cltn2 north J* = *r*› **and** ‹*proj2-incident r m*›
have *proj2-incident* (*apply-cltn2 north J*) *m* **by** *simp*

from ‹$p \neq q$› **and** *polar-inj* **have** *polar p* $\neq$ *polar q* **by** *fast*

from ‹*proj2-incident p l*› **and** ‹*proj2-incident q l*›
have *proj2-incident* (*pole l*) (*polar p*)
and *proj2-incident* (*pole l*) (*polar q*)
by (*simp-all add*: *incident-pole-polar*)
with ‹*polar p* $\neq$ *polar q*›
have *pole l* = *proj2-intersection* (*polar p*) (*polar q*)
by (*rule proj2-intersection-unique*)
with ‹*apply-cltn2 far-north J* = *proj2-intersection* (*polar p*) (*polar q*)›
have *apply-cltn2 far-north J* = *pole l* **by** *simp*
with ‹*M-perp l m*›
have *proj2-incident* (*apply-cltn2 far-north J*) *m* **by** (*unfold M-perp-def*) *simp*

with *north-far-north-distinct* **and** *north-south-far-north-on-meridian*
and ‹*proj2-incident* (*apply-cltn2 north J*) *m*›
have *apply-cltn2-line meridian J* = *m* **by** (*simp add*: *apply-cltn2-line-unique*)
with ‹*is-K2-isometry J*› **and** ‹*apply-cltn2-line equator J* = *l*›
show ∃ *J*. *is-K2-isometry J*
∧ *apply-cltn2-line equator J* = *l* ∧ *apply-cltn2-line meridian J* = *m*
by (*simp add*: *exI* [*of* - *J*])
qed

definition *drop-perp* :: *proj2* ⇒ *proj2-line* ⇒ *proj2-line* **where**
drop-perp p l ≜ *proj2-line-through p* (*pole l*)

lemma *drop-perp-incident*: *proj2-incident p* (*drop-perp p l*)
by (*unfold drop-perp-def*) (*rule proj2-line-through-incident*)

lemma *drop-perp-perp*: *M-perp l* (*drop-perp p l*)
by (*unfold drop-perp-def M-perp-def*) (*rule proj2-line-through-incident*)

definition *perp-foot* :: *proj2* ⇒ *proj2-line* ⇒ *proj2* **where**
perp-foot p l ≜ *proj2-intersection l* (*drop-perp p l*)

lemma *perp-foot-incident*:
shows *proj2-incident* (*perp-foot p l*) *l*
and *proj2-incident* (*perp-foot p l*) (*drop-perp p l*)
by (*unfold perp-foot-def*) (*rule proj2-intersection-incident*)+

lemma *M-perp-hyp2*:
assumes *M-perp l m* **and** *a* ∈ *hyp2* **and** *proj2-incident a l* **and** *b* ∈ *hyp2*
and *proj2-incident b m* **and** *proj2-incident c l* **and** *proj2-incident c m*
shows *c* ∈ *hyp2*
proof −
from ‹*M-perp l m*› **and** ‹*a* ∈ *hyp2*› **and** ‹*proj2-incident a l*› **and** ‹*b* ∈ *hyp2*›
and ‹*proj2-incident b m*› **and** *M-perp-to-compass* [*of l m a b*]
obtain *J* **where** *is-K2-isometry J* **and** *apply-cltn2-line equator J* = *l*
and *apply-cltn2-line meridian J* = *m*
by *auto*

from ‹*is-K2-isometry J*› **and** *K2-centre-in-K2*
have *apply-cltn2 K2-centre J* ∈ *hyp2*
by (*rule statement60-one-way*)

from ‹*proj2-incident c l*› **and** ‹*apply-cltn2-line equator J* = *l*›
and ‹*proj2-incident c m*› **and** ‹*apply-cltn2-line meridian J* = *m*›
have *proj2-incident c* (*apply-cltn2-line equator J*)
and *proj2-incident c* (*apply-cltn2-line meridian J*)
by *simp-all*
with *equator-meridian-distinct* **and** *K2-centre-on-equator-meridian*
have *apply-cltn2 K2-centre J* = *c* **by** (*rule apply-cltn2-unique*)
with ‹*apply-cltn2 K2-centre J* ∈ *hyp2*› **show** *c* ∈ *hyp2* **by** *simp*

qed

lemma *perp-foot-hyp2*:
  assumes $a \in hyp2$ **and** *proj2-incident a l* **and** $b \in hyp2$
  shows *perp-foot b l* $\in hyp2$
  using *drop-perp-perp* [*of l b*] **and** ‹$a \in hyp2$› **and** ‹*proj2-incident a l*›
    and ‹$b \in hyp2$› **and** *drop-perp-incident* [*of b l*]
    and *perp-foot-incident* [*of b l*]
  by (*rule M-perp-hyp2*)

definition *perp-up* :: *proj2* ⇒ *proj2-line* ⇒ *proj2* **where**
  perp-up a l
  ≜ *if proj2-incident a l then* ϵ *p*. $p \in S \wedge$ *proj2-incident p* (*drop-perp a l*)
  else endpoint-in-S (*perp-foot a l*) *a*

lemma *perp-up-degenerate-in-S-incident*:
  assumes $a \in hyp2$ **and** *proj2-incident a l*
  shows *perp-up a l* $\in S$ (**is** *?p* $\in S$)
  and *proj2-incident* (*perp-up a l*) (*drop-perp a l*)
proof −
  from ‹*proj2-incident a l*›
  have *?p* = (ϵ *p*. $p \in S \wedge$ *proj2-incident p* (*drop-perp a l*))
    by (*unfold perp-up-def*) *simp*

  from ‹$a \in hyp2$› **and** *drop-perp-incident* [*of a l*]
  have ∃ *p*. $p \in S \wedge$ *proj2-incident p* (*drop-perp a l*)
    by (*rule line-through-K2-intersect-S*)
  hence *?p* $\in S \wedge$ *proj2-incident ?p* (*drop-perp a l*)
    unfolding ‹*?p* = (ϵ *p*. $p \in S \wedge$ *proj2-incident p* (*drop-perp a l*))›
    by (*rule someI-ex*)
  thus *?p* $\in S$ **and** *proj2-incident ?p* (*drop-perp a l*) **by** *simp-all*
qed

lemma *perp-up-non-degenerate-in-S-at-end*:
  assumes $a \in hyp2$ **and** $b \in hyp2$ **and** *proj2-incident b l*
  and ¬ *proj2-incident a l*
  shows *perp-up a l* $\in S$
  and $B_{\mathbb{R}}$ (*cart2-pt* (*perp-foot a l*)) (*cart2-pt a*) (*cart2-pt* (*perp-up a l*))
proof −
  from ‹¬ *proj2-incident a l*›
  have *perp-up a l* = *endpoint-in-S* (*perp-foot a l*) *a*
    by (*unfold perp-up-def*) *simp*

  from ‹$b \in hyp2$› **and** ‹*proj2-incident b l*› **and** ‹$a \in hyp2$›
  have *perp-foot a l* $\in hyp2$ **by** (*rule perp-foot-hyp2*)
  with ‹$a \in hyp2$›
  show *perp-up a l* $\in S$
    and $B_{\mathbb{R}}$ (*cart2-pt* (*perp-foot a l*)) (*cart2-pt a*) (*cart2-pt* (*perp-up a l*))
    unfolding ‹*perp-up a l* = *endpoint-in-S* (*perp-foot a l*) *a*›

    by (*simp-all add*: *endpoint-in-S*)
qed

lemma *perp-up-in-S*:
  assumes $a \in hyp2$ **and** $b \in hyp2$ **and** *proj2-incident b l*
  shows *perp-up a l* $\in S$
proof *cases*
  assume *proj2-incident a l*
  with ‹$a \in hyp2$›
  show *perp-up a l* $\in S$ **by** (*rule perp-up-degenerate-in-S-incident*)
next
  assume ¬ *proj2-incident a l*
  with *assms*
  show *perp-up a l* $\in S$ **by** (*rule perp-up-non-degenerate-in-S-at-end*)
qed

lemma *perp-up-incident*:
  assumes $a \in hyp2$ **and** $b \in hyp2$ **and** *proj2-incident b l*
  shows *proj2-incident* (*perp-up a l*) (*drop-perp a l*)
  (**is** *proj2-incident ?p ?m*)
proof *cases*
  assume *proj2-incident a l*
  with ‹$a \in hyp2$›
  show *proj2-incident ?p ?m* **by** (*rule perp-up-degenerate-in-S-incident*)
next
  assume ¬ *proj2-incident a l*
  hence *?p* = *endpoint-in-S* (*perp-foot a l*) *a* (**is** *?p* = *endpoint-in-S ?c a*)
    by (*unfold perp-up-def*) *simp*

  from *perp-foot-incident* [*of a l*] **and** ‹¬ *proj2-incident a l*›
  have *?c* $\neq$ *a* **by** *auto*

  from ‹$b \in hyp2$› **and** ‹*proj2-incident b l*› **and** ‹$a \in hyp2$›
  have *?c* $\in hyp2$ **by** (*rule perp-foot-hyp2*)
  with ‹*?c* $\neq$ *a*› **and** ‹$a \in hyp2$› **and** *drop-perp-incident* [*of a l*]
    and *perp-foot-incident* [*of a l*]
  show *proj2-incident ?p ?m*
    by (*unfold* ‹*?p* = *endpoint-in-S ?c a*›) (*simp add*: *endpoint-in-S-incident*)
qed

lemma *drop-perp-same-line-pole-in-S*:
  assumes *drop-perp p l* = *l*
  shows *pole l* $\in S$
proof −
  from ‹*drop-perp p l* = *l*›
  have *l* = *proj2-line-through p* (*pole l*) **by** (*unfold drop-perp-def*) *simp*
  with *proj2-line-through-incident* [*of pole l p*]
  have *proj2-incident* (*pole l*) *l* **by** *simp*
  hence *proj2-incident* (*pole l*) (*polar* (*pole l*)) **by** (*subst polar-pole*)

thus *pole l* ∈ *S* **by** (*unfold incident-own-polar-in-S*)
qed

lemma *hyp2-drop-perp-not-same-line*:
assumes *a* ∈ *hyp2*
shows *drop-perp a l* ≠ *l*
proof
assume *drop-perp a l* = *l*
hence *pole l* ∈ *S* **by** (*rule drop-perp-same-line-pole-in-S*)
with ‹*a* ∈ *hyp2*›
have ¬ *proj2-incident a* (*polar* (*pole l*))
by (*simp add*: *tangent-not-through-K2*)
with ‹*drop-perp a l* = *l*›
have ¬ *proj2-incident a* (*drop-perp a l*) **by** (*simp add*: *polar-pole*)
with *drop-perp-incident* [*of a l*] **show** *False* **by** *simp*
qed

lemma *hyp2-incident-perp-foot-same-point*:
assumes *a* ∈ *hyp2* **and** *proj2-incident a l*
shows *perp-foot a l* = *a*
proof −
from ‹*a* ∈ *hyp2*›
have *drop-perp a l* ≠ *l* **by** (*rule hyp2-drop-perp-not-same-line*)
with *perp-foot-incident* [*of a l*] **and** ‹*proj2-incident a l*›
and *drop-perp-incident* [*of a l*] **and** *proj2-incident-unique*
show *perp-foot a l* = *a* **by** *fast*
qed

lemma *perp-up-at-end*:
assumes *a* ∈ *hyp2* **and** *b* ∈ *hyp2* **and** *proj2-incident b l*
shows $B_{\mathbb{R}}$ (*cart2-pt* (*perp-foot a l*)) (*cart2-pt a*) (*cart2-pt* (*perp-up a l*))
proof *cases*
assume *proj2-incident a l*
with ‹*a* ∈ *hyp2*›
have *perp-foot a l* = *a* **by** (*rule hyp2-incident-perp-foot-same-point*)
thus $B_{\mathbb{R}}$ (*cart2-pt* (*perp-foot a l*)) (*cart2-pt a*) (*cart2-pt* (*perp-up a l*))
by (*simp add*: *real-euclid.th3-1 real-euclid.th3-2*)
next
assume ¬ *proj2-incident a l*
with *assms*
show $B_{\mathbb{R}}$ (*cart2-pt* (*perp-foot a l*)) (*cart2-pt a*) (*cart2-pt* (*perp-up a l*))
by (*rule perp-up-non-degenerate-in-S-at-end*)
qed

definition *perp-down* :: *proj2* ⇒ *proj2-line* ⇒ *proj2* **where**
perp-down a l ≜ *endpoint-in-S* (*perp-up a l*) *a*

lemma *perp-down-in-S*:
assumes *a* ∈ *hyp2* **and** *b* ∈ *hyp2* **and** *proj2-incident b l*

shows *perp-down a l* ∈ *S*
proof −
from *assms* **have** *perp-up a l* ∈ *S* **by** (*rule perp-up-in-S*)
with ‹*a* ∈ *hyp2*›
show *perp-down a l* ∈ *S* **by** (*unfold perp-down-def*) (*simp add*: *endpoint-in-S*)
qed

lemma *perp-down-incident*:
assumes *a* ∈ *hyp2* **and** *b* ∈ *hyp2* **and** *proj2-incident b l*
shows *proj2-incident* (*perp-down a l*) (*drop-perp a l*)
proof −
from *assms* **have** *perp-up a l* ∈ *S* **by** (*rule perp-up-in-S*)
with ‹*a* ∈ *hyp2*› **have** *perp-up a l* ≠ *a* **by** (*rule hyp2-S-not-equal* [*symmetric*])

from *assms*
have *proj2-incident* (*perp-up a l*) (*drop-perp a l*) **by** (*rule perp-up-incident*)
with ‹*perp-up a l* ≠ *a*› **and** ‹*perp-up a l* ∈ *S*› **and** ‹*a* ∈ *hyp2*›
and *drop-perp-incident* [*of a l*]
show *proj2-incident* (*perp-down a l*) (*drop-perp a l*)
by (*unfold perp-down-def*) (*simp add*: *endpoint-in-S-incident*)
qed

lemma *perp-up-down-distinct*:
assumes *a* ∈ *hyp2* **and** *b* ∈ *hyp2* **and** *proj2-incident b l*
shows *perp-up a l* ≠ *perp-down a l*
proof −
from *assms* **have** *perp-up a l* ∈ *S* **by** (*rule perp-up-in-S*)
with ‹*a* ∈ *hyp2*›
show *perp-up a l* ≠ *perp-down a l*
unfolding *perp-down-def*
by (*simp add*: *endpoint-in-S-S-strict-hyp2-distinct* [*symmetric*])
qed

lemma *perp-up-down-foot-are-endpoints-in-S*:
assumes *a* ∈ *hyp2* **and** *b* ∈ *hyp2* **and** *proj2-incident b l*
shows *are-endpoints-in-S* (*perp-up a l*) (*perp-down a l*) (*perp-foot a l*) *a*
proof −
from ‹*b* ∈ *hyp2*› **and** ‹*proj2-incident b l*› **and** ‹*a* ∈ *hyp2*›
have *perp-foot a l* ∈ *hyp2* **by** (*rule perp-foot-hyp2*)

from *assms* **have** *perp-up a l* ∈ *S* **by** (*rule perp-up-in-S*)

from *assms*
have *proj2-incident* (*perp-up a l*) (*drop-perp a l*) **by** (*rule perp-up-incident*)
with ‹*perp-foot a l* ∈ *hyp2*› **and** ‹*a* ∈ *hyp2*› **and** ‹*perp-up a l* ∈ *S*›
and *perp-foot-incident*(*2*) [*of a l*] **and** *drop-perp-incident* [*of a l*]
show *are-endpoints-in-S* (*perp-up a l*) (*perp-down a l*) (*perp-foot a l*) *a*
by (*unfold perp-down-def*) (*rule end-and-opposite-are-endpoints-in-S*)
qed

lemma *perp-foot-opposite-endpoint-in-S*:
assumes $a \in hyp2$ **and** $b \in hyp2$ **and** $c \in hyp2$ **and** $a \neq b$
shows
endpoint-in-S (*endpoint-in-S* a b) (*perp-foot* c (*proj2-line-through* a b))
= *endpoint-in-S* b a
(**is** *endpoint-in-S* ?p ?d = *endpoint-in-S* b a)
proof −
let ?q = *endpoint-in-S* ?p ?d

from ‹$a \in hyp2$› **and** ‹$b \in hyp2$› **have** ?$p \in S$ **by** (*simp add*: *endpoint-in-S*)

let ?l = *proj2-line-through* a b
have *proj2-incident* a ?l **and** *proj2-incident* b ?l
by (*rule proj2-line-through-incident*)+
with ‹$a \neq b$› **and** ‹$a \in hyp2$› **and** ‹$b \in hyp2$›
have *proj2-incident* ?p ?l
by (*simp-all add*: *endpoint-in-S-incident*)

from ‹$a \in hyp2$› **and** ‹*proj2-incident* a ?l› **and** ‹$c \in hyp2$›
have ?$d \in hyp2$ **by** (*rule perp-foot-hyp2*)
with ‹?$p \in S$› **have** ?$q \neq$?p **by** (*rule endpoint-in-S-S-strict-hyp2-distinct*)

from ‹?$p \in S$› **and** ‹?$d \in hyp2$› **have** ?$q \in S$ **by** (*simp add*: *endpoint-in-S*)

from ‹?$d \in hyp2$› **and** ‹?$p \in S$›
have ?$p \neq$?d **by** (*rule hyp2-S-not-equal* [*symmetric*])
with ‹?$p \in S$› **and** ‹?$d \in hyp2$› **and** ‹*proj2-incident* ?p ?l›
and *perp-foot-incident*(*1*) [*of* c ?l]
have *proj2-incident* ?q ?l **by** (*simp add*: *endpoint-in-S-incident*)
with ‹$a \neq b$› **and** ‹$a \in hyp2$› **and** ‹$b \in hyp2$› **and** ‹?$q \in S$›
and ‹*proj2-incident* a ?l› **and** ‹*proj2-incident* b ?l›
have ?q = ?p $\vee$?q = *endpoint-in-S* b a
by (*simp add*: *endpoints-in-S-incident-unique*)
with ‹?$q \neq$?p› **show** ?q = *endpoint-in-S* b a **by** *simp*
qed

lemma *endpoints-in-S-perp-foot-are-endpoints-in-S*:
assumes $a \in hyp2$ **and** $b \in hyp2$ **and** $c \in hyp2$ **and** $a \neq b$
and *proj2-incident* a l **and** *proj2-incident* b l
shows *are-endpoints-in-S*
(*endpoint-in-S* a b) (*endpoint-in-S* b a) a (*perp-foot* c l)
proof −
define p q d
where p = *endpoint-in-S* a b
and q = *endpoint-in-S* b a
and d = *perp-foot* c l

from ‹$a \neq b$› **and** ‹$a \in hyp2$› **and** ‹$b \in hyp2$›

have $p \neq q$ **by** (*unfold p-def q-def*) (*simp add*: *endpoint-in-S-swap*)

from ‹$a \in hyp2$› **and** ‹$b \in hyp2$›
have $p \in S$ **and** $q \in S$ **by** (*unfold p-def q-def*) (*simp-all add*: *endpoint-in-S*)

from ‹$a \in hyp2$› **and** ‹*proj2-incident a l*› **and** ‹$c \in hyp2$›
have $d \in hyp2$ **by** (*unfold d-def*) (*rule perp-foot-hyp2*)

from ‹$a \neq b$› **and** ‹$a \in hyp2$› **and** ‹$b \in hyp2$› **and** ‹*proj2-incident a l*›
  and ‹*proj2-incident b l*›
have *proj2-incident p l* **and** *proj2-incident q l*
  by (*unfold p-def q-def*) (*simp-all add*: *endpoint-in-S-incident*)
with ‹*proj2-incident a l*› **and** *perp-foot-incident*(*1*) [*of c l*]
have *proj2-set-Col* $\{p,q,a,d\}$
  by (*unfold d-def proj2-set-Col-def*) (*simp add*: *exI* [*of - l*])
with ‹$p \neq q$› **and** ‹$p \in S$› **and** ‹$q \in S$› **and** ‹$a \in hyp2$› **and** ‹$d \in hyp2$›
show *are-endpoints-in-S p q a d* **by** (*unfold are-endpoints-in-S-def*) *simp*
qed

definition *right-angle* :: *proj2* ⇒ *proj2* ⇒ *proj2* ⇒ *bool* **where**
  right-angle p a q
  $\triangleq p \in S \wedge q \in S \wedge a \in hyp2$
  ∧ *M-perp* (*proj2-line-through p a*) (*proj2-line-through a q*)

lemma *perp-foot-up-right-angle*:
  assumes $p \in S$ **and** $a \in hyp2$ **and** $b \in hyp2$ **and** *proj2-incident p l*
  and *proj2-incident b l*
  shows *right-angle p* (*perp-foot a l*) (*perp-up a l*)
proof −
  define c **where** c = *perp-foot a l*
  define q **where** q = *perp-up a l*
  from ‹$a \in hyp2$› **and** ‹$b \in hyp2$› **and** ‹*proj2-incident b l*›
  have $q \in S$ **by** (*unfold q-def*) (*rule perp-up-in-S*)

  from ‹$b \in hyp2$› **and** ‹*proj2-incident b l*› **and** ‹$a \in hyp2$›
  have $c \in hyp2$ **by** (*unfold c-def*) (*rule perp-foot-hyp2*)
  with ‹$p \in S$› **and** ‹$q \in S$› **have** $c \neq p$ **and** $c \neq q$
    by (*simp-all add*: *hyp2-S-not-equal*)

  from ‹$c \neq p$› [*symmetric*] **and** ‹*proj2-incident p l*›
    and *perp-foot-incident*(*1*) [*of a l*]
  have l = *proj2-line-through p c*
    by (*unfold c-def*) (*rule proj2-line-through-unique*)

  define m **where** m = *drop-perp a l*
  from ‹$a \in hyp2$› **and** ‹$b \in hyp2$› **and** ‹*proj2-incident b l*›
  have *proj2-incident q m* **by** (*unfold q-def m-def*) (*rule perp-up-incident*)
  with ‹$c \neq q$› **and** *perp-foot-incident*(*2*) [*of a l*]
  have m = *proj2-line-through c q*

by (*unfold c-def m-def*) (*rule proj2-line-through-unique*)
with ‹$p \in S$› **and** ‹$q \in S$› **and** ‹$c \in hyp2$› **and** *drop-perp-perp* [*of l a*]
and ‹$l = proj2\text{-}line\text{-}through\ p\ c$›
show *right-angle p* (*perp-foot a l*) (*perp-up a l*)
by (*unfold right-angle-def q-def c-def m-def*) *simp*
qed

lemma *M-perp-unique*:
assumes $a \in hyp2$ **and** $b \in hyp2$ **and** *proj2-incident a l*
and *proj2-incident b m* **and** *proj2-incident b n* **and** *M-perp l m*
and *M-perp l n*
shows $m = n$
proof −
from ‹$a \in hyp2$› **and** ‹*proj2-incident a l*›
have $pole\ l \notin hyp2$ **by** (*rule line-through-hyp2-pole-not-in-hyp2*)
with ‹$b \in hyp2$› **have** $b \neq pole\ l$ **by** *auto*
with ‹*proj2-incident b m*› **and** ‹*M-perp l m*› **and** ‹*proj2-incident b n*›
and ‹*M-perp l n*› **and** *proj2-incident-unique*
show $m = n$ **by** (*unfold M-perp-def*) *auto*
qed

lemma *perp-foot-eq-implies-drop-perp-eq*:
assumes $a \in hyp2$ **and** $b \in hyp2$ **and** *proj2-incident a l*
and *perp-foot b l* = *perp-foot c l*
shows *drop-perp b l* = *drop-perp c l*
proof −
from ‹$a \in hyp2$› **and** ‹*proj2-incident a l*› **and** ‹$b \in hyp2$›
have $perp\text{-}foot\ b\ l \in hyp2$ **by** (*rule perp-foot-hyp2*)

from ‹*perp-foot b l* = *perp-foot c l*›
have *proj2-incident* (*perp-foot b l*) (*drop-perp c l*)
by (*simp add*: *perp-foot-incident*)
with ‹$a \in hyp2$› **and** ‹$perp\text{-}foot\ b\ l \in hyp2$› **and** ‹*proj2-incident a l*›
and *perp-foot-incident*(*2*) [*of b l*] **and** *drop-perp-perp* [*of l*]
show *drop-perp b l* = *drop-perp c l* **by** (*simp add*: *M-perp-unique*)
qed

lemma *right-angle-to-compass*:
assumes *right-angle p a q*
shows $\exists$ *J*. *is-K2-isometry J* $\wedge$ *apply-cltn2 p J* = *east*
$\wedge$ *apply-cltn2 a J* = *K2-centre* $\wedge$ *apply-cltn2 q J* = *north*
proof −
from ‹*right-angle p a q*›
have $p \in S$ **and** $q \in S$ **and** $a \in hyp2$
and *M-perp* (*proj2-line-through p a*) (*proj2-line-through a q*)
(**is** *M-perp ?l ?m*)
by (*unfold right-angle-def*) *simp-all*

have *proj2-incident p ?l* **and** *proj2-incident a ?l*

  and *proj2-incident q ?m* **and** *proj2-incident a ?m*
  by (*rule proj2-line-through-incident*)+

from ‹*M-perp ?l ?m*› **and** ‹$a \in$ *hyp2*› **and** ‹*proj2-incident a ?l*›
  and ‹*proj2-incident a ?m*› **and** *M-perp-to-compass* [*of ?l ?m a a*]
obtain $J''i$ **where** *is-K2-isometry* $J''i$
  and *apply-cltn2-line equator* $J''i$ = *?l*
  and *apply-cltn2-line meridian* $J''i$ = *?m*
  by *auto*
let *?*J'' = *cltn2-inverse* $J''i$

from ‹*apply-cltn2-line equator* $J''i$ = *?l*›
  and ‹*apply-cltn2-line meridian* $J''i$ = *?m*›
  and ‹*proj2-incident p ?l*› **and** ‹*proj2-incident a ?l*›
  and ‹*proj2-incident q ?m*› **and** ‹*proj2-incident a ?m*›
have *proj2-incident* (*apply-cltn2 p ?*J'') *equator*
  and *proj2-incident* (*apply-cltn2 a ?*J'') *equator*
  and *proj2-incident* (*apply-cltn2 q ?*J'') *meridian*
  and *proj2-incident* (*apply-cltn2 a ?*J'') *meridian*
  by (*simp-all add*: *apply-cltn2-incident* [*symmetric*])

from ‹*proj2-incident* (*apply-cltn2 a ?*J'') *equator*›
  and ‹*proj2-incident* (*apply-cltn2 a ?*J'') *meridian*›
have *apply-cltn2 a ?*J'' = *K2-centre*
  by (*rule on-equator-meridian-is-K2-centre*)

from ‹*is-K2-isometry* $J''i$›
have *is-K2-isometry ?*J'' **by** (*rule cltn2-inverse-is-K2-isometry*)
with ‹$p \in S$› **and** ‹$q \in S$›
have *apply-cltn2 p ?*$J'' \in S$ **and** *apply-cltn2 q ?*$J'' \in S$
  by (*unfold is-K2-isometry-def*) *simp-all*
with *east-west-distinct* **and** *north-south-distinct* **and** *compass-in-S*
  and *east-west-on-equator* **and** *north-south-far-north-on-meridian*
  and ‹*proj2-incident* (*apply-cltn2 p ?*J'') *equator*›
  and ‹*proj2-incident* (*apply-cltn2 q ?*J'') *meridian*›
have *apply-cltn2 p ?*J'' = *east* $\vee$ *apply-cltn2 p ?*J'' = *west*
  and *apply-cltn2 q ?*J'' = *north* $\vee$ *apply-cltn2 q ?*J'' = *south*
  by (*simp-all add*: *line-S-two-intersections-only*)

have $\exists$ J'. *is-K2-isometry* J' $\wedge$ *apply-cltn2 p* J' = *east*
  $\wedge$ *apply-cltn2 a* J' = *K2-centre*
  $\wedge$ (*apply-cltn2 q* J' = *north* $\vee$ *apply-cltn2 q* J' = *south*)
proof *cases*
  assume *apply-cltn2 p ?*J'' = *east*
  with ‹*is-K2-isometry ?*J''› **and** ‹*apply-cltn2 a ?*J'' = *K2-centre*›
    and ‹*apply-cltn2 q ?*J'' = *north* $\vee$ *apply-cltn2 q ?*J'' = *south*›
  show $\exists$ J'. *is-K2-isometry* J' $\wedge$ *apply-cltn2 p* J' = *east*
    $\wedge$ *apply-cltn2 a* J' = *K2-centre*
    $\wedge$ (*apply-cltn2 q* J' = *north* $\vee$ *apply-cltn2 q* J' = *south*)

by (*simp add*: *exI* [*of* - *?J''*])
next
assume *apply-cltn2 p ?J''* $\neq$ *east*
with ‹*apply-cltn2 p ?J''* = *east* $\vee$ *apply-cltn2 p ?J''* = *west*›
have *apply-cltn2 p ?J''* = *west* **by** *simp*

let *?J'* = *cltn2-compose ?J'' meridian-reflect*
from ‹*is-K2-isometry ?J''*› **and** *meridian-reflect-K2-isometry*
have *is-K2-isometry ?J'* **by** (*rule cltn2-compose-is-K2-isometry*)
moreover
from ‹*apply-cltn2 p ?J''* = *west*› **and** ‹*apply-cltn2 a ?J''* = *K2-centre*›
and ‹*apply-cltn2 q ?J''* = *north* $\vee$ *apply-cltn2 q ?J''* = *south*›
and *compass-reflect-compass*
have *apply-cltn2 p ?J'* = *east* **and** *apply-cltn2 a ?J'* = *K2-centre*
and *apply-cltn2 q ?J'* = *north* $\vee$ *apply-cltn2 q ?J'* = *south*
by (*auto simp add*: *cltn2.act-act* [*simplified*, *symmetric*])
ultimately
show $\exists$ *J'*. *is-K2-isometry J'* $\wedge$ *apply-cltn2 p J'* = *east*
$\wedge$ *apply-cltn2 a J'* = *K2-centre*
$\wedge$ (*apply-cltn2 q J'* = *north* $\vee$ *apply-cltn2 q J'* = *south*)
by (*simp add*: *exI* [*of* - *?J'*])
qed
then obtain *J'* **where** *is-K2-isometry J'* **and** *apply-cltn2 p J'* = *east*
and *apply-cltn2 a J'* = *K2-centre*
and *apply-cltn2 q J'* = *north* $\vee$ *apply-cltn2 q J'* = *south*
by *auto*

show $\exists$ *J*. *is-K2-isometry J* $\wedge$ *apply-cltn2 p J* = *east*
$\wedge$ *apply-cltn2 a J* = *K2-centre* $\wedge$ *apply-cltn2 q J* = *north*
proof *cases*
assume *apply-cltn2 q J'* = *north*
with ‹*is-K2-isometry J'*› **and** ‹*apply-cltn2 p J'* = *east*›
and ‹*apply-cltn2 a J'* = *K2-centre*›
show $\exists$ *J*. *is-K2-isometry J* $\wedge$ *apply-cltn2 p J* = *east*
$\wedge$ *apply-cltn2 a J* = *K2-centre* $\wedge$ *apply-cltn2 q J* = *north*
by (*simp add*: *exI* [*of* - *J'*])
next
assume *apply-cltn2 q J'* $\neq$ *north*
with ‹*apply-cltn2 q J'* = *north* $\vee$ *apply-cltn2 q J'* = *south*›
have *apply-cltn2 q J'* = *south* **by** *simp*

let *?J* = *cltn2-compose J' equator-reflect*
from ‹*is-K2-isometry J'*› **and** *equator-reflect-K2-isometry*
have *is-K2-isometry ?J* **by** (*rule cltn2-compose-is-K2-isometry*)
moreover
from ‹*apply-cltn2 p J'* = *east*› **and** ‹*apply-cltn2 a J'* = *K2-centre*›
and ‹*apply-cltn2 q J'* = *south*› **and** *compass-reflect-compass*
have *apply-cltn2 p ?J* = *east* **and** *apply-cltn2 a ?J* = *K2-centre*
and *apply-cltn2 q ?J* = *north*

by (*auto simp add*: *cltn2.act-act* [*simplified*, *symmetric*])
ultimately
show ∃ *J*. *is-K2-isometry J* ∧ *apply-cltn2 p J* = *east*
∧ *apply-cltn2 a J* = *K2-centre* ∧ *apply-cltn2 q J* = *north*
by (*simp add*: *exI* [*of* - *?J*])
qed
qed

lemma *right-angle-to-right-angle*:
assumes *right-angle p a q* **and** *right-angle r b s*
shows ∃ *J*. *is-K2-isometry J*
∧ *apply-cltn2 p J* = *r* ∧ *apply-cltn2 a J* = *b* ∧ *apply-cltn2 q J* = *s*
proof −
from ‹*right-angle p a q*› **and** *right-angle-to-compass* [*of p a q*]
obtain *H* **where** *is-K2-isometry H* **and** *apply-cltn2 p H* = *east*
and *apply-cltn2 a H* = *K2-centre* **and** *apply-cltn2 q H* = *north*
by *auto*

from ‹*right-angle r b s*› **and** *right-angle-to-compass* [*of r b s*]
obtain *K* **where** *is-K2-isometry K* **and** *apply-cltn2 r K* = *east*
and *apply-cltn2 b K* = *K2-centre* **and** *apply-cltn2 s K* = *north*
by *auto*

let *?Ki* = *cltn2-inverse K*
let *?J* = *cltn2-compose H ?Ki*
from ‹*is-K2-isometry H*› **and** ‹*is-K2-isometry K*›
have *is-K2-isometry ?J*
by (*simp add*: *cltn2-inverse-is-K2-isometry cltn2-compose-is-K2-isometry*)

from ‹*apply-cltn2 r K* = *east*› **and** ‹*apply-cltn2 b K* = *K2-centre*›
and ‹*apply-cltn2 s K* = *north*›
have *apply-cltn2 east ?Ki* = *r* **and** *apply-cltn2 K2-centre ?Ki* = *b*
and *apply-cltn2 north ?Ki* = *s*
by (*simp-all add*: *cltn2.act-inv-iff* [*simplified*])
with ‹*apply-cltn2 p H* = *east*› **and** ‹*apply-cltn2 a H* = *K2-centre*›
and ‹*apply-cltn2 q H* = *north*›
have *apply-cltn2 p ?J* = *r* **and** *apply-cltn2 a ?J* = *b*
and *apply-cltn2 q ?J* = *s*
by (*simp-all add*: *cltn2.act-act* [*simplified*,*symmetric*])
with ‹*is-K2-isometry ?J*›
show ∃ *J*. *is-K2-isometry J*
∧ *apply-cltn2 p J* = *r* ∧ *apply-cltn2 a J* = *b* ∧ *apply-cltn2 q J* = *s*
by (*simp add*: *exI* [*of* - *?J*])
qed

8.11 Functions of distance

definition *exp-2dist* :: *proj2* ⇒ *proj2* ⇒ *real* **where**
exp-2dist a b

```
  ≜ if a = b
  then 1
  else cross-ratio (endpoint-in-S a b) (endpoint-in-S b a) a b

definition cosh-dist :: proj2 ⇒ proj2 ⇒ real where
  cosh-dist a b ≜ (sqrt (exp-2dist a b) + sqrt (1 / (exp-2dist a b))) / 2

lemma exp-2dist-formula:
  assumes a ≠ 0 and b ≠ 0 and proj2-abs a ∈ hyp2 (is ?pa ∈ hyp2)
  and proj2-abs b ∈ hyp2 (is ?pb ∈ hyp2)
  shows exp-2dist (proj2-abs a) (proj2-abs b)
    = (a · (M *v b) + sqrt (quarter-discrim a b))
      / (a · (M *v b) − sqrt (quarter-discrim a b))
  ∨ exp-2dist (proj2-abs a) (proj2-abs b)
    = (a · (M *v b) − sqrt (quarter-discrim a b))
      / (a · (M *v b) + sqrt (quarter-discrim a b))
  (is ?e2d = (?aMb + ?sqd) / (?aMb − ?sqd)
    ∨ ?e2d = (?aMb − ?sqd) / (?aMb + ?sqd))
proof cases
  assume ?pa = ?pb
  hence ?e2d = 1 by (unfold exp-2dist-def, simp)

  from ‹?pa = ?pb›
  have quarter-discrim a b = 0 by (rule quarter-discrim-self-zero)
  hence ?sqd = 0 by simp

  from ‹proj2-abs a = proj2-abs b› and ‹b ≠ 0› and proj2-abs-abs-mult
  obtain k where a = k *R b by auto

  from ‹b ≠ 0› and ‹proj2-abs b ∈ hyp2›
  have b · (M *v b) < 0 by (subst K2-abs [symmetric])
  with ‹a ≠ 0› and ‹a = k *R b› have ?aMb ≠ 0 by simp
  with ‹?e2d = 1› and ‹?sqd = 0›
  show ?e2d = (?aMb + ?sqd) / (?aMb − ?sqd)
    ∨ ?e2d = (?aMb − ?sqd) / (?aMb + ?sqd)
    by simp
next
  assume ?pa ≠ ?pb
  let ?l = proj2-line-through ?pa ?pb
  have proj2-incident ?pa ?l and proj2-incident ?pb ?l
    by (rule proj2-line-through-incident)+
  with ‹a ≠ 0› and ‹b ≠ 0› and ‹?pa ≠ ?pb›
  have proj2-incident (S-intersection1 a b) ?l (is proj2-incident ?Si1 ?l)
    and proj2-incident (S-intersection2 a b) ?l (is proj2-incident ?Si2 ?l)
    by (rule S-intersections-incident)+
  with ‹proj2-incident ?pa ?l› and ‹proj2-incident ?pb ?l›
  have proj2-set-Col {?pa,?pb,?Si1,?Si2} by (unfold proj2-set-Col-def, auto)

  have {?pa,?pb,?Si2,?Si1} = {?pa,?pb,?Si1,?Si2} by auto
```

from ‹$a \neq 0$› **and** ‹$b \neq 0$› **and** ‹*?pa* $\neq$ *?pb*› **and** ‹*?pa* $\in$ *hyp2*›
have *?Si1* $\in$ *S* **and** *?Si2* $\in$ *S*
  by (*simp-all add*: *S-intersections-in-S*)
with ‹*?pa* $\in$ *hyp2*› **and** ‹*?pb* $\in$ *hyp2*›
have *?Si1* $\neq$ *?pa* **and** *?Si2* $\neq$ *?pa* **and** *?Si1* $\neq$ *?pb* **and** *?Si2* $\neq$ *?pb*
  by (*simp-all add*: *hyp2-S-not-equal* [*symmetric*])
with ‹*proj2-set-Col* {*?pa*,*?pb*,*?Si1*,*?Si2*}› **and** ‹*?pa* $\neq$ *?pb*›
have *cross-ratio-correct ?pa ?pb ?Si1 ?Si2*
  and *cross-ratio-correct ?pa ?pb ?Si2 ?Si1*
  unfolding *cross-ratio-correct-def*
  by (*simp-all add*: ‹{*?pa*,*?pb*,*?Si2*,*?Si1*} = {*?pa*,*?pb*,*?Si1*,*?Si2*}›)

from ‹$a \neq 0$› **and** ‹$b \neq 0$› **and** ‹*?pa* $\neq$ *?pb*› **and** ‹*?pa* $\in$ *hyp2*›
have *?Si1* $\neq$ *?Si2* **by** (*simp add*: *S-intersections-distinct*)
with ‹*cross-ratio-correct ?pa ?pb ?Si1 ?Si2*›
  and ‹*cross-ratio-correct ?pa ?pb ?Si2 ?Si1*›
have *cross-ratio ?Si1 ?Si2 ?pa ?pb* = *cross-ratio ?pa ?pb ?Si1 ?Si2*
  and *cross-ratio ?Si2 ?Si1 ?pa ?pb* = *cross-ratio ?pa ?pb ?Si2 ?Si1*
  by (*simp-all add*: *cross-ratio-swap-13-24*)

from ‹$a \neq 0$› **and** ‹*proj2-abs a* $\in$ *hyp2*›
have $a \cdot (M *v\ a) < 0$ **by** (*subst K2-abs* [*symmetric*])
with ‹$a \neq 0$› **and** ‹$b \neq 0$› **and** ‹*?pa* $\neq$ *?pb*› **and** *cross-ratio-abs* [*of a b 1 1*]
have *cross-ratio ?pa ?pb ?Si1 ?Si2* = (− *?aMb* − *?sqd*) / (− *?aMb* + *?sqd*)
  by (*unfold S-intersections-defs S-intersection-coeffs-defs*, *simp*)
with *times-divide-times-eq* [*of* −*1* −*1* −*?aMb* − *?sqd* −*?aMb* + *?sqd*]
have *cross-ratio ?pa ?pb ?Si1 ?Si2* = (*?aMb* + *?sqd*) / (*?aMb* − *?sqd*) **by** (*simp add*: *ac-simps*)
with ‹*cross-ratio ?Si1 ?Si2 ?pa ?pb* = *cross-ratio ?pa ?pb ?Si1 ?Si2*›
have *cross-ratio ?Si1 ?Si2 ?pa ?pb* = (*?aMb* + *?sqd*) / (*?aMb* − *?sqd*) **by** *simp*

from ‹*cross-ratio ?pa ?pb ?Si1 ?Si2* = (*?aMb* + *?sqd*) / (*?aMb* − *?sqd*)›
  and *cross-ratio-swap-34* [*of ?pa ?pb ?Si2 ?Si1*]
have *cross-ratio ?pa ?pb ?Si2 ?Si1* = (*?aMb* − *?sqd*) / (*?aMb* + *?sqd*) **by** *simp*
with ‹*cross-ratio ?Si2 ?Si1 ?pa ?pb* = *cross-ratio ?pa ?pb ?Si2 ?Si1*›
have *cross-ratio ?Si2 ?Si1 ?pa ?pb* = (*?aMb* − *?sqd*) / (*?aMb* + *?sqd*) **by** *simp*

from ‹$a \neq 0$› **and** ‹$b \neq 0$› **and** ‹*?pa* $\neq$ *?pb*› **and** ‹*?pa* $\in$ *hyp2*› **and** ‹*?pb* $\in$ *hyp2*›
have (*?Si1* = *endpoint-in-S ?pa ?pb* $\wedge$ *?Si2* = *endpoint-in-S ?pb ?pa*)
  $\vee$ (*?Si2* = *endpoint-in-S ?pa ?pb* $\wedge$ *?Si1* = *endpoint-in-S ?pb ?pa*)
  by (*simp add*: *S-intersections-endpoints-in-S*)
with ‹*cross-ratio ?Si1 ?Si2 ?pa ?pb* = (*?aMb* + *?sqd*) / (*?aMb* − *?sqd*)›
  and ‹*cross-ratio ?Si2 ?Si1 ?pa ?pb* = (*?aMb* − *?sqd*) / (*?aMb* + *?sqd*)›
  and ‹*?pa* $\neq$ *?pb*›
show *?e2d* = (*?aMb* + *?sqd*) / (*?aMb* − *?sqd*)
  $\vee$ *?e2d* = (*?aMb* − *?sqd*) / (*?aMb* + *?sqd*)
  by (*unfold exp-2dist-def*, *auto*)
qed

```
lemma cosh-dist-formula:
  assumes a ≠ 0 and b ≠ 0 and proj2-abs a ∈ hyp2 (is ?pa ∈ hyp2)
  and proj2-abs b ∈ hyp2 (is ?pb ∈ hyp2)
  shows cosh-dist (proj2-abs a) (proj2-abs b)
  = |a · (M *v b)| / sqrt (a · (M *v a) * (b · (M *v b)))
  (is cosh-dist ?pa ?pb = |?aMb| / sqrt (?aMa * ?bMb))
proof −
  let ?qd = quarter-discrim a b
  let ?sqd = sqrt ?qd
  let ?e2d = exp-2dist ?pa ?pb
  from assms
  have ?e2d = (?aMb + ?sqd) / (?aMb − ?sqd)
    ∨ ?e2d = (?aMb − ?sqd) / (?aMb + ?sqd)
    by (rule exp-2dist-formula)
  hence cosh-dist ?pa ?pb
    = (sqrt ((?aMb + ?sqd) / (?aMb − ?sqd))
    + sqrt ((?aMb − ?sqd) / (?aMb + ?sqd)))
    / 2
    by (unfold cosh-dist-def, auto)

  have ?qd ≥ 0
  proof cases
    assume ?pa = ?pb
    thus ?qd ≥ 0 by (simp add: quarter-discrim-self-zero)
  next
    assume ?pa ≠ ?pb
    with ‹a ≠ 0› and ‹b ≠ 0› and ‹?pa ∈ hyp2›
    have ?qd > 0 by (simp add: quarter-discrim-positive)
    thus ?qd ≥ 0 by simp
  qed
  with real-sqrt-pow2 [of ?qd] have ?sqd² = ?qd by simp
  hence (?aMb + ?sqd) * (?aMb − ?sqd) = ?aMa * ?bMb
    by (unfold quarter-discrim-def, simp add: algebra-simps power2-eq-square)

  from times-divide-times-eq [of
    ?aMb + ?sqd ?aMb + ?sqd ?aMb + ?sqd ?aMb − ?sqd]
  have (?aMb + ?sqd) / (?aMb − ?sqd)
    = (?aMb + ?sqd)² / ((?aMb + ?sqd) * (?aMb − ?sqd))
    by (simp add: power2-eq-square)
  with ‹(?aMb + ?sqd) * (?aMb − ?sqd) = ?aMa * ?bMb›
  have (?aMb + ?sqd) / (?aMb − ?sqd) = (?aMb + ?sqd)² / (?aMa * ?bMb) by
simp
  hence sqrt ((?aMb + ?sqd) / (?aMb − ?sqd))
    = |?aMb + ?sqd| / sqrt (?aMa * ?bMb)
    by (simp add: real-sqrt-divide)

  from times-divide-times-eq [of
    ?aMb + ?sqd ?aMb − ?sqd ?aMb − ?sqd ?aMb − ?sqd]
```

```
  have (?aMb − ?sqd) / (?aMb + ?sqd)
    = (?aMb − ?sqd)^2 / ((?aMb + ?sqd) ∗ (?aMb − ?sqd))
    by (simp add: power2-eq-square)
  with ‹(?aMb + ?sqd) ∗ (?aMb − ?sqd) = ?aMa ∗ ?bMb›
  have (?aMb − ?sqd) / (?aMb + ?sqd) = (?aMb − ?sqd)^2 / (?aMa ∗ ?bMb) by
simp
  hence sqrt ((?aMb − ?sqd) / (?aMb + ?sqd))
    = |?aMb − ?sqd| / sqrt (?aMa ∗ ?bMb)
    by (simp add: real-sqrt-divide)

  from ‹a ≠ 0› and ‹b ≠ 0› and ‹?pa ∈ hyp2› and ‹?pb ∈ hyp2›
  have ?aMa < 0 and ?bMb < 0
    by (simp-all add: K2-imp-M-neg)
  with ‹(?aMb + ?sqd) ∗ (?aMb − ?sqd) = ?aMa ∗ ?bMb›
  have (?aMb + ?sqd) ∗ (?aMb − ?sqd) > 0 by (simp add: mult-neg-neg)
  hence ?aMb + ?sqd ≠ 0 and ?aMb − ?sqd ≠ 0 by auto
  hence sgn (?aMb + ?sqd) ∈ {−1,1} and sgn (?aMb − ?sqd) ∈ {−1,1}
    by (simp-all add: sgn-real-def)

  from ‹(?aMb + ?sqd) ∗ (?aMb − ?sqd) > 0›
  have sgn ((?aMb + ?sqd) ∗ (?aMb − ?sqd)) = 1 by simp
  hence sgn (?aMb + ?sqd) ∗ sgn (?aMb − ?sqd) = 1 by (simp add: sgn-mult)
  with ‹sgn (?aMb + ?sqd) ∈ {−1,1}› and ‹sgn (?aMb − ?sqd) ∈ {−1,1}›
  have sgn (?aMb + ?sqd) = sgn (?aMb − ?sqd) by auto
  with abs-plus [of ?aMb + ?sqd ?aMb − ?sqd]
  have |?aMb + ?sqd| + |?aMb − ?sqd| = 2 ∗ |?aMb| by simp
  with ‹sqrt ((?aMb + ?sqd) / (?aMb − ?sqd))
    = |?aMb + ?sqd| / sqrt (?aMa ∗ ?bMb)›
    and ‹sqrt ((?aMb − ?sqd) / (?aMb + ?sqd))
    = |?aMb − ?sqd| / sqrt (?aMa ∗ ?bMb)›
    and add-divide-distrib [of
    |?aMb + ?sqd| |?aMb − ?sqd| sqrt (?aMa ∗ ?bMb)]
  have sqrt ((?aMb + ?sqd) / (?aMb − ?sqd))
    + sqrt ((?aMb − ?sqd) / (?aMb + ?sqd))
    = 2 ∗ |?aMb| / sqrt (?aMa ∗ ?bMb)
    by simp
  with ‹cosh-dist ?pa ?pb
    = (sqrt ((?aMb + ?sqd) / (?aMb − ?sqd))
    + sqrt ((?aMb − ?sqd) / (?aMb + ?sqd)))
    / 2›
  show cosh-dist ?pa ?pb = |?aMb| / sqrt (?aMa ∗ ?bMb) by simp
qed

lemma cosh-dist-perp-special-case:
  assumes |x| < 1 and |y| < 1
  shows cosh-dist (proj2-abs (vector [x,0,1])) (proj2-abs (vector [0,y,1]))
  = (cosh-dist K2-centre (proj2-abs (vector [x,0,1])))
  ∗ (cosh-dist K2-centre (proj2-abs (vector [0,y,1])))
  (is cosh-dist ?pa ?pb = (cosh-dist ?po ?pa) ∗ (cosh-dist ?po ?pb))
```

proof −
  have *vector* $[x,0,1] \neq (0::real\hat{}3)$ (**is** $?a \neq 0$)
    and *vector* $[0,y,1] \neq (0::real\hat{}3)$ (**is** $?b \neq 0$)
    by (*unfold vector-def*, *simp-all add*: *vec-eq-iff forall-3*)

  have $?a \cdot (M *v\ ?a) = x^2 - 1$ (**is** $?aMa = x^2 - 1$)
    and $?b \cdot (M *v\ ?b) = y^2 - 1$ (**is** $?bMb = y^2 - 1$)
    unfolding *vector-def* **and** *M-def* **and** *inner-vec-def*
      and *matrix-vector-mult-def*
    by (*simp-all add*: *sum-3 power2-eq-square*)
  with ‹$|x| < 1$› **and** ‹$|y| < 1$›
  have $?aMa < 0$ **and** $?bMb < 0$ **by** (*simp-all add*: *abs-square-less-1*)
  hence $?pa \in hyp2$ **and** $?pb \in hyp2$
    by (*simp-all add*: *M-neg-imp-K2*)
  with ‹$?a \neq 0$› **and** ‹$?b \neq 0$›
  have *cosh-dist* $?pa\ ?pb = |?a \cdot (M *v\ ?b)|\ /\ sqrt\ (?aMa * ?bMb)$
    (**is** *cosh-dist* $?pa\ ?pb = |?aMb|\ /\ sqrt\ (?aMa * ?bMb)$)
    by (*rule cosh-dist-formula*)
  also from ‹$?aMa = x^2 - 1$› **and** ‹$?bMb = y^2 - 1$›
  have $\ldots = |?aMb|\ /\ sqrt\ ((x^2 - 1) * (y^2 - 1))$ **by** *simp*
  finally have *cosh-dist* $?pa\ ?pb = 1\ /\ sqrt\ ((1 - x^2) * (1 - y^2))$
    unfolding *vector-def* **and** *M-def* **and** *inner-vec-def*
      and *matrix-vector-mult-def*
    by (*simp add*: *sum-3 algebra-simps*)

  let $?o = vector\ [0,0,1]$
  let $?oMa = ?o \cdot (M *v\ ?a)$
  let $?oMb = ?o \cdot (M *v\ ?b)$
  let $?oMo = ?o \cdot (M *v\ ?o)$
  from *K2-centre-non-zero* **and** ‹$?a \neq 0$› **and** ‹$?b \neq 0$›
    and *K2-centre-in-K2* **and** ‹$?pa \in hyp2$› **and** ‹$?pb \in hyp2$›
    and *cosh-dist-formula* [*of* $?o$]
  have *cosh-dist* $?po\ ?pa = |?oMa|\ /\ sqrt\ (?oMo * ?aMa)$
    and *cosh-dist* $?po\ ?pb = |?oMb|\ /\ sqrt\ (?oMo * ?bMb)$
    by (*unfold K2-centre-def*, *simp-all*)
  hence *cosh-dist* $?po\ ?pa = 1\ /\ sqrt\ (1 - x^2)$
    and *cosh-dist* $?po\ ?pb = 1\ /\ sqrt\ (1 - y^2)$
    unfolding *vector-def* **and** *M-def* **and** *inner-vec-def*
      and *matrix-vector-mult-def*
    by (*simp-all add*: *sum-3 power2-eq-square*)
  with ‹*cosh-dist* $?pa\ ?pb = 1\ /\ sqrt\ ((1 - x^2) * (1 - y^2))$›
  show *cosh-dist* $?pa\ ?pb =$ *cosh-dist* $?po\ ?pa *$ *cosh-dist* $?po\ ?pb$
    by (*simp add*: *real-sqrt-mult*)
qed

lemma *K2-isometry-cross-ratio-endpoints-in-S*:
  assumes $a \in hyp2$ **and** $b \in hyp2$ **and** *is-K2-isometry* J **and** $a \neq b$
  shows *cross-ratio* (*apply-cltn2* (*endpoint-in-S* a b) J)
  (*apply-cltn2* (*endpoint-in-S* b a) J) (*apply-cltn2* a J) (*apply-cltn2* b J)

```
  = cross-ratio (endpoint-in-S a b) (endpoint-in-S b a) a b
  (is cross-ratio ?pJ ?qJ ?aJ ?bJ = cross-ratio ?p ?q a b)
proof −
  let ?l = proj2-line-through a b
  have proj2-incident a ?l and proj2-incident b ?l
    by (rule proj2-line-through-incident)+
  with ‹a ≠ b› and ‹a ∈ hyp2› and ‹b ∈ hyp2›
  have proj2-incident ?p ?l and proj2-incident ?q ?l
    by (simp-all add: endpoint-in-S-incident)
  with ‹proj2-incident a ?l› and ‹proj2-incident b ?l›
  have proj2-set-Col {?p,?q,a,b}
    by (unfold proj2-set-Col-def) (simp add: exI [of - ?l])

  from ‹a ≠ b› and ‹a ∈ hyp2› and ‹b ∈ hyp2›
  have ?p ≠ ?q by (simp add: endpoint-in-S-swap)

  from ‹a ∈ hyp2› and ‹b ∈ hyp2› have ?p ∈ S by (simp add: endpoint-in-S)
  with ‹a ∈ hyp2› and ‹b ∈ hyp2›
  have a ≠ ?p and b ≠ ?p by (simp-all add: hyp2-S-not-equal)
  with ‹proj2-set-Col {?p,?q,a,b}› and ‹?p ≠ ?q›
  show cross-ratio ?pJ ?qJ ?aJ ?bJ = cross-ratio ?p ?q a b
    by (rule cross-ratio-cltn2)
qed

lemma K2-isometry-exp-2dist:
  assumes a ∈ hyp2 and b ∈ hyp2 and is-K2-isometry J
  shows exp-2dist (apply-cltn2 a J) (apply-cltn2 b J) = exp-2dist a b
  (is exp-2dist ?aJ ?bJ = -)
proof cases
  assume a = b
  thus exp-2dist ?aJ ?bJ = exp-2dist a b by (unfold exp-2dist-def) simp
next
  assume a ≠ b
  with apply-cltn2-injective have ?aJ ≠ ?bJ by fast

  let ?p = endpoint-in-S a b
  let ?q = endpoint-in-S b a
  let ?aJ = apply-cltn2 a J
    and ?bJ = apply-cltn2 b J
    and ?pJ = apply-cltn2 ?p J
    and ?qJ = apply-cltn2 ?q J
  from ‹a ≠ b› and ‹a ∈ hyp2› and ‹b ∈ hyp2› and ‹is-K2-isometry J›
  have endpoint-in-S ?aJ ?bJ = ?pJ and endpoint-in-S ?bJ ?aJ = ?qJ
    by (simp-all add: K2-isometry-endpoint-in-S)

  from assms and ‹a ≠ b›
  have cross-ratio ?pJ ?qJ ?aJ ?bJ = cross-ratio ?p ?q a b
    by (rule K2-isometry-cross-ratio-endpoints-in-S)
  with ‹endpoint-in-S ?aJ ?bJ = ?pJ› and ‹endpoint-in-S ?bJ ?aJ = ?qJ›
```

and ‹$a \neq b$› **and** ‹$?aJ \neq ?bJ$›
show *exp-2dist ?aJ ?bJ* = *exp-2dist a b* **by** (*unfold exp-2dist-def*) *simp*
qed

lemma *K2-isometry-cosh-dist*:
assumes $a \in hyp2$ **and** $b \in hyp2$ **and** *is-K2-isometry J*
shows *cosh-dist* (*apply-cltn2 a J*) (*apply-cltn2 b J*) = *cosh-dist a b*
using *assms*
by (*unfold cosh-dist-def*) (*simp add*: *K2-isometry-exp-2dist*)

lemma *cosh-dist-perp*:
assumes *M-perp l m* **and** $a \in hyp2$ **and** $b \in hyp2$ **and** $c \in hyp2$
and *proj2-incident a l* **and** *proj2-incident b l*
and *proj2-incident b m* **and** *proj2-incident c m*
shows *cosh-dist a c* = *cosh-dist b a* ∗ *cosh-dist b c*
proof −
from ‹*M-perp l m*› **and** ‹$b \in hyp2$› **and** ‹*proj2-incident b l*›
and ‹*proj2-incident b m*› **and** *M-perp-to-compass* [*of l m b b*]
obtain *J* **where** *is-K2-isometry J* **and** *apply-cltn2-line equator J* = *l*
and *apply-cltn2-line meridian J* = *m*
by *auto*

let *?Ji* = *cltn2-inverse J*
let *?aJi* = *apply-cltn2 a ?Ji*
let *?bJi* = *apply-cltn2 b ?Ji*
let *?cJi* = *apply-cltn2 c ?Ji*
from ‹*apply-cltn2-line equator J* = *l*› **and** ‹*apply-cltn2-line meridian J* = *m*›
and ‹*proj2-incident a l*› **and** ‹*proj2-incident b l*›
and ‹*proj2-incident b m*› **and** ‹*proj2-incident c m*›
have *proj2-incident ?aJi equator* **and** *proj2-incident ?bJi equator*
and *proj2-incident ?bJi meridian* **and** *proj2-incident ?cJi meridian*
by (*auto simp add*: *apply-cltn2-incident*)

from ‹*is-K2-isometry J*›
have *is-K2-isometry ?Ji* **by** (*rule cltn2-inverse-is-K2-isometry*)
with ‹$a \in hyp2$› **and** ‹$c \in hyp2$›
have $?aJi \in hyp2$ **and** $?cJi \in hyp2$
by (*simp-all add*: *statement60-one-way*)

from ‹$?aJi \in hyp2$› **and** ‹*proj2-incident ?aJi equator*›
and *on-equator-in-hyp2-rep*
obtain *x* **where** $|x| < 1$ **and** *?aJi* = *proj2-abs* (*vector* [x,*0*,*1*]) **by** *auto*
moreover
from ‹$?cJi \in hyp2$› **and** ‹*proj2-incident ?cJi meridian*›
and *on-meridian-in-hyp2-rep*
obtain *y* **where** $|y| < 1$ **and** *?cJi* = *proj2-abs* (*vector* [*0*,y,*1*]) **by** *auto*
moreover
from ‹*proj2-incident ?bJi equator*› **and** ‹*proj2-incident ?bJi meridian*›
have *?bJi* = *K2-centre* **by** (*rule on-equator-meridian-is-K2-centre*)

ultimately
have *cosh-dist ?aJi ?cJi* = *cosh-dist ?bJi ?aJi* $*$ *cosh-dist ?bJi ?cJi*
by (*simp add*: *cosh-dist-perp-special-case*)
with ‹$a \in$ *hyp2*› **and** ‹$b \in$ *hyp2*› **and** ‹$c \in$ *hyp2*› **and** ‹*is-K2-isometry ?Ji*›
show *cosh-dist a c* = *cosh-dist b a* $*$ *cosh-dist b c*
by (*simp add*: *K2-isometry-cosh-dist*)
qed

lemma *are-endpoints-in-S-ordered-cross-ratio*:
assumes *are-endpoints-in-S p q a b*
and $B_{\mathbb{R}}$ (*cart2-pt a*) (*cart2-pt b*) (*cart2-pt p*) (**is** $B_{\mathbb{R}}$ *?ca ?cb ?cp*)
shows *cross-ratio p q a b* $\geq$ *1*
proof −
from ‹*are-endpoints-in-S p q a b*›
have $p \neq q$ **and** $p \in S$ **and** $q \in S$ **and** $a \in$ *hyp2* **and** $b \in$ *hyp2*
and *proj2-set-Col* {*p*,*q*,*a*,*b*}
by (*unfold are-endpoints-in-S-def*) *simp-all*

from ‹$a \in$ *hyp2*› **and** ‹$b \in$ *hyp2*› **and** ‹$p \in S$› **and** ‹$q \in S$›
have *z-non-zero a* **and** *z-non-zero b* **and** *z-non-zero p* **and** *z-non-zero q*
by (*simp-all add*: *hyp2-S-z-non-zero*)
hence *proj2-abs* (*cart2-append1 p*) = *p* (**is** *proj2-abs ?cp1* = *p*)
and *proj2-abs* (*cart2-append1 q*) = *q* (**is** *proj2-abs ?cq1* = *q*)
and *proj2-abs* (*cart2-append1 a*) = *a* (**is** *proj2-abs ?ca1* = *a*)
and *proj2-abs* (*cart2-append1 b*) = *b* (**is** *proj2-abs ?cb1* = *b*)
by (*simp-all add*: *proj2-abs-cart2-append1*)

from ‹$b \in$ *hyp2*› **and** ‹$p \in S$› **have** $b \neq p$ **by** (*rule hyp2-S-not-equal*)
with ‹*z-non-zero a*› **and** ‹*z-non-zero b*› **and** ‹*z-non-zero p*›
and ‹$B_{\mathbb{R}}$ *?ca ?cb ?cp*› **and** *cart2-append1-between-right-strict* [*of a b p*]
obtain *j* **where** $j \geq 0$ **and** $j < 1$ **and** *?cb1* = $j *_R$ *?cp1* + $(1-j) *_R$ *?ca1*
by *auto*

from ‹*proj2-set-Col* {*p*,*q*,*a*,*b*}›
obtain *l* **where** *proj2-incident q l* **and** *proj2-incident p l*
and *proj2-incident a l*
by (*unfold proj2-set-Col-def*) *auto*
with ‹$p \neq q$› **and** ‹$q \in S$› **and** ‹$p \in S$› **and** ‹$a \in$ *hyp2*›
and *S-hyp2-S-cart2-append1* [*of q p a l*]
obtain *k* **where** $k > 0$ **and** $k < 1$ **and** *?ca1* = $k *_R$ *?cp1* + $(1-k) *_R$ *?cq1*
by *auto*

from ‹*z-non-zero p*› **and** ‹*z-non-zero q*›
have *?cp1* $\neq 0$ **and** *?cq1* $\neq 0$ **by** (*simp-all add*: *cart2-append1-non-zero*)

from ‹$p \neq q$› **and** ‹*proj2-abs ?cp1* = *p*› **and** ‹*proj2-abs ?cq1* = *q*›
have *proj2-abs ?cp1* $\neq$ *proj2-abs ?cq1* **by** *simp*

from ‹$k < 1$› **have** $1-k \neq 0$ **by** *simp*

with ‹$j < 1$› **have** $(1-j)*(1-k) \neq 0$ **by** *simp*

from ‹$j < 1$› **and** ‹$k > 0$› **have** $(1-j)*k > 0$ **by** *simp*

from ‹$?cb1 = j *_R ?cp1 + (1-j) *_R ?ca1$›
have $?cb1 = (j+(1-j)*k) *_R ?cp1 + ((1-j)*(1-k)) *_R ?cq1$
  by (*unfold* ‹$?ca1 = k *_R ?cp1 + (1-k) *_R ?cq1$›) (*simp add*: *algebra-simps*)
with ‹$?ca1 = k *_R ?cp1 + (1-k) *_R ?cq1$›
have *proj2-abs* $?ca1 =$ *proj2-abs* $(k *_R ?cp1 + (1-k) *_R ?cq1)$
  and *proj2-abs* $?cb1$
  $=$ *proj2-abs* $((j+(1-j)*k) *_R ?cp1 + ((1-j)*(1-k)) *_R ?cq1)$
  by *simp-all*
with ‹*proj2-abs* $?ca1 = a$› **and** ‹*proj2-abs* $?cb1 = b$›
have $a =$ *proj2-abs* $(k *_R ?cp1 + (1-k) *_R ?cq1)$
  and $b =$ *proj2-abs* $((j+(1-j)*k) *_R ?cp1 + ((1-j)*(1-k)) *_R ?cq1)$
  by *simp-all*
with ‹*proj2-abs* $?cp1 = p$› **and** ‹*proj2-abs* $?cq1 = q$›
have *cross-ratio* $p\ q\ a\ b$
  $=$ *cross-ratio* (*proj2-abs* $?cp1$) (*proj2-abs* $?cq1$)
  (*proj2-abs* $(k *_R ?cp1 + (1-k) *_R ?cq1)$)
  (*proj2-abs* $((j+(1-j)*k) *_R ?cp1 + ((1-j)*(1-k)) *_R ?cq1)$)
  by *simp*
also from ‹$?cp1 \neq 0$› **and** ‹$?cq1 \neq 0$› **and** ‹*proj2-abs* $?cp1 \neq$ *proj2-abs* $?cq1$›
  and ‹$1-k \neq 0$› **and** ‹$(1-j)*(1-k) \neq 0$›
have $\ldots = (1-k)*(j+(1-j)*k) / (k*((1-j)*(1-k)))$ **by** (*rule cross-ratio-abs*)
also from ‹$1-k \neq 0$› **have** $\ldots = (j+(1-j)*k) / ((1-j)*k)$ **by** *simp*
also from ‹$j \geq 0$› **and** ‹$(1-j)*k > 0$› **have** $\ldots \geq 1$ **by** *simp*
finally show *cross-ratio* $p\ q\ a\ b \geq 1$ **.**
qed

lemma *cross-ratio-S-S-hyp2-hyp2-positive*:
  assumes *are-endpoints-in-S* $p\ q\ a\ b$
  shows *cross-ratio* $p\ q\ a\ b > 0$
proof *cases*
  assume $B_{\mathbb{R}}$ (*cart2-pt* p) (*cart2-pt* b) (*cart2-pt* a)
  hence $B_{\mathbb{R}}$ (*cart2-pt* a) (*cart2-pt* b) (*cart2-pt* p)
    by (*rule real-euclid.th3-2*)
  with *assms* **have** *cross-ratio* $p\ q\ a\ b \geq 1$
    by (*rule are-endpoints-in-S-ordered-cross-ratio*)
  thus *cross-ratio* $p\ q\ a\ b > 0$ **by** *simp*
next
  assume $\neg\ B_{\mathbb{R}}$ (*cart2-pt* p) (*cart2-pt* b) (*cart2-pt* a) (**is** $\neg\ B_{\mathbb{R}}\ ?cp\ ?cb\ ?ca$)

  from ‹*are-endpoints-in-S* $p\ q\ a\ b$›
  have *are-endpoints-in-S* $p\ q\ b\ a$ **by** (*rule are-endpoints-in-S-swap-34*)

  from ‹*are-endpoints-in-S* $p\ q\ a\ b$›
  have $p \in S$ **and** $a \in$ *hyp2* **and** $b \in$ *hyp2* **and** *proj2-set-Col* $\{p,q,a,b\}$
    by (*unfold are-endpoints-in-S-def*) *simp-all*

from ‹*proj2-set-Col* {*p*,*q*,*a*,*b*}›
have *proj2-set-Col* {*p*,*a*,*b*}
by (*simp add*: *proj2-subset-Col* [*of* {*p*,*a*,*b*} {*p*,*q*,*a*,*b*}])
hence *proj2-Col p a b* **by** (*subst proj2-Col-iff-set-Col*)
with ‹*p* ∈ *S*› **and** ‹*a* ∈ *hyp2*› **and** ‹*b* ∈ *hyp2*›
have $B_{\mathbb{R}}$ *?cp ?ca ?cb* ∨ $B_{\mathbb{R}}$ *?cp ?cb ?ca* **by** (*simp add*: *S-at-edge*)
with ‹¬ $B_{\mathbb{R}}$ *?cp ?cb ?ca*› **have** $B_{\mathbb{R}}$ *?cp ?ca ?cb* **by** *simp*
hence $B_{\mathbb{R}}$ *?cb ?ca ?cp* **by** (*rule real-euclid.th3-2*)
with ‹*are-endpoints-in-S p q b a*›
have *cross-ratio p q b a* ≥ *1*
by (*rule are-endpoints-in-S-ordered-cross-ratio*)
thus *cross-ratio p q a b* > *0* **by** (*subst cross-ratio-swap-34*) *simp*
qed

lemma *cosh-dist-general*:
assumes *are-endpoints-in-S p q a b*
shows *cosh-dist a b*
= (*sqrt* (*cross-ratio p q a b*) + *1* / *sqrt* (*cross-ratio p q a b*)) / *2*
proof −
from ‹*are-endpoints-in-S p q a b*›
have *p* ≠ *q* **and** *p* ∈ *S* **and** *q* ∈ *S* **and** *a* ∈ *hyp2* **and** *b* ∈ *hyp2*
and *proj2-set-Col* {*p*,*q*,*a*,*b*}
by (*unfold are-endpoints-in-S-def*) *simp-all*

from ‹*a* ∈ *hyp2*› **and** ‹*b* ∈ *hyp2*› **and** ‹*p* ∈ *S*› **and** ‹*q* ∈ *S*›
have *a* ≠ *p* **and** *a* ≠ *q* **and** *b* ≠ *p* **and** *b* ≠ *q*
by (*simp-all add*: *hyp2-S-not-equal*)

show *cosh-dist a b*
= (*sqrt* (*cross-ratio p q a b*) + *1* / *sqrt* (*cross-ratio p q a b*)) / *2*
proof *cases*
assume *a* = *b*
hence *cosh-dist a b* = *1* **by** (*unfold cosh-dist-def exp-2dist-def*) *simp*

from ‹*proj2-set-Col* {*p*,*q*,*a*,*b*}›
have *proj2-Col p q a* **by** (*unfold* ‹*a* = *b*›) (*simp add*: *proj2-Col-iff-set-Col*)
with ‹*p* ≠ *q*› **and** ‹*a* ≠ *p*› **and** ‹*a* ≠ *q*›
have *cross-ratio p q a b* = *1* **by** (*simp add*: ‹*a* = *b*› *cross-ratio-equal-1*)
hence (*sqrt* (*cross-ratio p q a b*) + *1* / *sqrt* (*cross-ratio p q a b*)) / *2*
= *1*
by *simp*
with ‹*cosh-dist a b* = *1*›
show *cosh-dist a b*
= (*sqrt* (*cross-ratio p q a b*) + *1* / *sqrt* (*cross-ratio p q a b*)) / *2*
by *simp*
next
assume *a* ≠ *b*

```
    let ?r = endpoint-in-S a b
    let ?s = endpoint-in-S b a
    from ‹a ≠ b›
    have exp-2dist a b = cross-ratio ?r ?s a b by (unfold exp-2dist-def) simp

    from ‹a ≠ b› and ‹are-endpoints-in-S p q a b›
    have (p = ?r ∧ q = ?s) ∨ (q = ?r ∧ p = ?s) by (rule are-endpoints-in-S)

    show cosh-dist a b
      = (sqrt (cross-ratio p q a b) + 1 / sqrt (cross-ratio p q a b)) / 2
    proof cases
      assume p = ?r ∧ q = ?s
      with ‹exp-2dist a b = cross-ratio ?r ?s a b›
      have exp-2dist a b = cross-ratio p q a b by simp
      thus cosh-dist a b
        = (sqrt (cross-ratio p q a b) + 1 / sqrt (cross-ratio p q a b)) / 2
        by (unfold cosh-dist-def) (simp add: real-sqrt-divide)
    next
      assume ¬ (p = ?r ∧ q = ?s)
      with ‹(p = ?r ∧ q = ?s) ∨ (q = ?r ∧ p = ?s)›
      have q = ?r and p = ?s by simp-all
      with ‹exp-2dist a b = cross-ratio ?r ?s a b›
      have exp-2dist a b = cross-ratio q p a b by simp

      have {q,p,a,b} = {p,q,a,b} by auto
      with ‹proj2-set-Col {p,q,a,b}› and ‹p ≠ q› and ‹a ≠ p› and ‹b ≠ p›
        and ‹a ≠ q› and ‹b ≠ q›
      have cross-ratio-correct p q a b and cross-ratio-correct q p a b
        by (unfold cross-ratio-correct-def) simp-all
      hence cross-ratio q p a b = 1 / (cross-ratio p q a b)
        by (rule cross-ratio-swap-12)
      with ‹exp-2dist a b = cross-ratio q p a b›
      have exp-2dist a b = 1 / (cross-ratio p q a b) by simp
      thus cosh-dist a b
        = (sqrt (cross-ratio p q a b) + 1 / sqrt (cross-ratio p q a b)) / 2
        by (unfold cosh-dist-def) (simp add: real-sqrt-divide)
    qed
  qed
qed

lemma exp-2dist-positive:
  assumes a ∈ hyp2 and b ∈ hyp2
  shows exp-2dist a b > 0
proof cases
  assume a = b
  thus exp-2dist a b > 0 by (unfold exp-2dist-def) simp
next
  assume a ≠ b
```

let *?p* = *endpoint-in-S a b*
let *?q* = *endpoint-in-S b a*
from ‹$a \neq b$› **and** ‹$a \in$ *hyp2*› **and** ‹$b \in$ *hyp2*›
have *are-endpoints-in-S ?p ?q a b*
by (*rule endpoints-in-S-are-endpoints-in-S*)
hence *cross-ratio ?p ?q a b* > *0* **by** (*rule cross-ratio-S-S-hyp2-hyp2-positive*)
with ‹$a \neq b$› **show** *exp-2dist a b* > *0* **by** (*unfold exp-2dist-def*) *simp*
qed

lemma *cosh-dist-at-least-1*:
assumes $a \in$ *hyp2* **and** $b \in$ *hyp2*
shows *cosh-dist a b* ≥ *1*
proof −
from *assms* **have** *exp-2dist a b* > *0* **by** (*rule exp-2dist-positive*)
with *am-gm2*(*1*) [*of sqrt* (*exp-2dist a b*) *sqrt* (*1* / *exp-2dist a b*)]
show *cosh-dist a b* ≥ *1*
by (*unfold cosh-dist-def*) (*simp add*: *real-sqrt-mult* [*symmetric*])
qed

lemma *cosh-dist-positive*:
assumes $a \in$ *hyp2* **and** $b \in$ *hyp2*
shows *cosh-dist a b* > *0*
proof −
from *assms* **have** *cosh-dist a b* ≥ *1* **by** (*rule cosh-dist-at-least-1*)
thus *cosh-dist a b* > *0* **by** *simp*
qed

lemma *cosh-dist-perp-divide*:
assumes *M-perp l m* **and** $a \in$ *hyp2* **and** $b \in$ *hyp2* **and** $c \in$ *hyp2*
and *proj2-incident a l* **and** *proj2-incident b l* **and** *proj2-incident b m*
and *proj2-incident c m*
shows *cosh-dist b c* = *cosh-dist a c* / *cosh-dist b a*
proof −
from ‹$b \in$ *hyp2*› **and** ‹$a \in$ *hyp2*›
have *cosh-dist b a* > *0* **by** (*rule cosh-dist-positive*)

from *assms*
have *cosh-dist a c* = *cosh-dist b a* ∗ *cosh-dist b c* **by** (*rule cosh-dist-perp*)
with ‹*cosh-dist b a* > *0*›
show *cosh-dist b c* = *cosh-dist a c* / *cosh-dist b a* **by** *simp*
qed

lemma *real-hyp2-C-cross-ratio-endpoints-in-S*:
assumes $a \neq b$ **and** $a\ b \equiv_K c\ d$
shows *cross-ratio* (*endpoint-in-S* (*Rep-hyp2 a*) (*Rep-hyp2 b*))
(*endpoint-in-S* (*Rep-hyp2 b*) (*Rep-hyp2 a*)) (*Rep-hyp2 a*) (*Rep-hyp2 b*)
= *cross-ratio* (*endpoint-in-S* (*Rep-hyp2 c*) (*Rep-hyp2 d*))
(*endpoint-in-S* (*Rep-hyp2 d*) (*Rep-hyp2 c*)) (*Rep-hyp2 c*) (*Rep-hyp2 d*)
(**is** *cross-ratio ?p ?q ?a′ ?b′* = *cross-ratio ?r ?s ?c′ ?d′*)

proof −
from ‹$a \neq b$› **and** ‹$a\ b \equiv_K c\ d$› **have** $c \neq d$ **by** (*auto simp add*: *hyp2.A3'*)
with ‹$a \neq b$› **have** *?a'* ≠ *?b'* **and** *?c'* ≠ *?d'* **by** (*unfold Rep-hyp2-inject*)

from ‹$a\ b \equiv_K c\ d$›
obtain *J* **where** *is-K2-isometry J* **and** *hyp2-cltn2 a J* = *c*
and *hyp2-cltn2 b J* = *d*
by (*unfold real-hyp2-C-def*) *auto*
hence *apply-cltn2 ?a' J* = *?c'* **and** *apply-cltn2 ?b' J* = *?d'*
by (*simp-all add*: *Rep-hyp2-cltn2* [*symmetric*])
with ‹*?a'* ≠ *?b'*› **and** ‹*is-K2-isometry J*›
have *apply-cltn2 ?p J* = *?r* **and** *apply-cltn2 ?q J* = *?s*
by (*simp-all add*: *Rep-hyp2 K2-isometry-endpoint-in-S*)

from ‹*?a'* ≠ *?b'*›
have *proj2-set-Col* {*?p*,*?q*,*?a'*,*?b'*}
by (*simp add*: *Rep-hyp2 proj2-set-Col-endpoints-in-S*)

from ‹*?a'* ≠ *?b'*› **have** *?p* ≠ *?q* **by** (*simp add*: *Rep-hyp2 endpoint-in-S-swap*)

have *?p* ∈ *S* **by** (*simp add*: *Rep-hyp2 endpoint-in-S*)
hence *?a'* ≠ *?p* **and** *?b'* ≠ *?p* **by** (*simp-all add*: *Rep-hyp2 hyp2-S-not-equal*)
with ‹*proj2-set-Col* {*?p*,*?q*,*?a'*,*?b'*}› **and** ‹*?p* ≠ *?q*›
have *cross-ratio ?p ?q ?a' ?b'*
= *cross-ratio* (*apply-cltn2 ?p J*) (*apply-cltn2 ?q J*)
(*apply-cltn2 ?a' J*) (*apply-cltn2 ?b' J*)
by (*rule cross-ratio-cltn2* [*symmetric*])
with ‹*apply-cltn2 ?p J* = *?r*› **and** ‹*apply-cltn2 ?q J* = *?s*›
and ‹*apply-cltn2 ?a' J* = *?c'*› **and** ‹*apply-cltn2 ?b' J* = *?d'*›
show *cross-ratio ?p ?q ?a' ?b'* = *cross-ratio ?r ?s ?c' ?d'* **by** *simp*
qed

lemma *real-hyp2-C-exp-2dist*:
assumes $a\ b \equiv_K c\ d$
shows *exp-2dist* (*Rep-hyp2 a*) (*Rep-hyp2 b*)
= *exp-2dist* (*Rep-hyp2 c*) (*Rep-hyp2 d*)
(**is** *exp-2dist ?a' ?b'* = *exp-2dist ?c' ?d'*)
proof −
from ‹$a\ b \equiv_K c\ d$›
obtain *J* **where** *is-K2-isometry J* **and** *hyp2-cltn2 a J* = *c*
and *hyp2-cltn2 b J* = *d*
by (*unfold real-hyp2-C-def*) *auto*
hence *apply-cltn2 ?a' J* = *?c'* **and** *apply-cltn2 ?b' J* = *?d'*
by (*simp-all add*: *Rep-hyp2-cltn2* [*symmetric*])

from *Rep-hyp2* [*of a*] **and** *Rep-hyp2* [*of b*] **and** ‹*is-K2-isometry J*›
have *exp-2dist* (*apply-cltn2 ?a' J*) (*apply-cltn2 ?b' J*) = *exp-2dist ?a' ?b'*
by (*rule K2-isometry-exp-2dist*)
with ‹*apply-cltn2 ?a' J* = *?c'*› **and** ‹*apply-cltn2 ?b' J* = *?d'*›

show *exp-2dist ?a′ ?b′* = *exp-2dist ?c′ ?d′* **by** *simp*
qed

lemma *real-hyp2-C-cosh-dist*:
assumes *a b* $\equiv_K$ *c d*
shows *cosh-dist* (*Rep-hyp2 a*) (*Rep-hyp2 b*)
= *cosh-dist* (*Rep-hyp2 c*) (*Rep-hyp2 d*)
using *assms*
by (*unfold cosh-dist-def*) (*simp add*: *real-hyp2-C-exp-2dist*)

lemma *cross-ratio-in-terms-of-cosh-dist*:
assumes *are-endpoints-in-S p q a b*
and $B_{\mathbb{R}}$ (*cart2-pt a*) (*cart2-pt b*) (*cart2-pt p*)
shows *cross-ratio p q a b*
= *2* ∗ (*cosh-dist a b*)^2 + *2* ∗ *cosh-dist a b* ∗ *sqrt* ((*cosh-dist a b*)^2 − *1*) − *1*
(**is** *?pqab* = *2* ∗ *?ab*^{2} + *2* ∗ *?ab* ∗ *sqrt* (*?ab*^{2} − *1*) − *1*)
proof −
from ‹*are-endpoints-in-S p q a b*›
have *?ab* = (*sqrt ?pqab* + *1* / *sqrt ?pqab*) / *2* **by** (*rule cosh-dist-general*)
hence *sqrt ?pqab* − *2* ∗ *?ab* + *1* / *sqrt ?pqab* = *0* **by** *simp*
hence *sqrt ?pqab* ∗ (*sqrt ?pqab* − *2* ∗ *?ab* + *1* / *sqrt ?pqab*) = *0* **by** *simp*
moreover from *assms*
have *?pqab* ≥ *1* **by** (*rule are-endpoints-in-S-ordered-cross-ratio*)
ultimately have *?pqab* − *2* ∗ *?ab* ∗ (*sqrt ?pqab*) + *1* = *0*
by (*simp add*: *algebra-simps real-sqrt-mult* [*symmetric*])
with ‹*?pqab* ≥ *1*› **and** *discriminant-iff* [*of 1 sqrt ?pqab* − *2* ∗ *?ab 1*]
have *sqrt ?pqab* = (*2* ∗ *?ab* + *sqrt* (*4* ∗ *?ab*^{2} − *4*)) / *2*
∨ *sqrt ?pqab* = (*2* ∗ *?ab* − *sqrt* (*4* ∗ *?ab*^{2} − *4*)) / *2*
unfolding *discrim-def*
by (*simp add*: *real-sqrt-mult* [*symmetric*] *power2-eq-square*)
moreover have *sqrt* (*4* ∗ *?ab*^{2} − *4*) = *sqrt* (*4* ∗ (*?ab*^{2} − *1*)) **by** *simp*
hence *sqrt* (*4* ∗ *?ab*^{2} − *4*) = *2* ∗ *sqrt* (*?ab*^{2} − *1*)
by (*unfold real-sqrt-mult*) *simp*
ultimately have *sqrt ?pqab* = *2* ∗ (*?ab* + *sqrt* (*?ab*^{2} − *1*)) / *2*
∨ *sqrt ?pqab* = *2* ∗ (*?ab* − *sqrt* (*?ab*^{2} − *1*)) / *2*
by *simp*
hence *sqrt ?pqab* = *?ab* + *sqrt* (*?ab*^{2} − *1*)
∨ *sqrt ?pqab* = *?ab* − *sqrt* (*?ab*^{2} − *1*)
by (*simp only*: *nonzero-mult-div-cancel-left* [*of 2*])

from ‹*are-endpoints-in-S p q a b*›
have *a* ∈ *hyp2* **and** *b* ∈ *hyp2* **by** (*unfold are-endpoints-in-S-def*) *simp-all*
hence *?ab* ≥ *1* **by** (*rule cosh-dist-at-least-1*)
hence *?ab*^{2} ≥ *1* **by** *simp*
hence *sqrt* (*?ab*^{2} − *1*) ≥ *0* **by** *simp*
hence *sqrt* (*?ab*^{2} − *1*) ∗ *sqrt* (*?ab*^{2} − *1*) = *?ab*^{2} − *1*
by (*simp add*: *real-sqrt-mult* [*symmetric*])
hence (*?ab* + *sqrt* (*?ab*^{2} − *1*)) ∗ (*?ab* − *sqrt* (*?ab*^{2} − *1*)) = *1*
by (*simp add*: *algebra-simps power2-eq-square*)

have *?ab* − *sqrt* (*?ab*^{2} − *1*) ≤ *1*
proof (*rule ccontr*)
assume ¬ (*?ab* − *sqrt* (*?ab*^{2} − *1*) ≤ *1*)
hence *1* < *?ab* − *sqrt* (*?ab*^{2} − *1*) **by** *simp*
also from ‹*sqrt* (*?ab*^{2} − *1*) ≥ *0*›
have . . . ≤ *?ab* + *sqrt* (*?ab*^{2} − *1*) **by** *simp*
finally have *1* < *?ab* + *sqrt* (*?ab*^{2} − *1*) **by** *simp*
with ‹*1* < *?ab* − *sqrt* (*?ab*^{2} − *1*)›
and *mult-strict-mono′* [*of*
1 *?ab* + *sqrt* (*?ab*^{2} − *1*) *1* *?ab* − *sqrt* (*?ab*^{2} − *1*)]
have *1* < (*?ab* + *sqrt* (*?ab*^{2} − *1*)) ∗ (*?ab* − *sqrt* (*?ab*^{2} − *1*)) **by** *simp*
with ‹(*?ab* + *sqrt* (*?ab*^{2} − *1*)) ∗ (*?ab* − *sqrt* (*?ab*^{2} − *1*)) = *1*›
show *False* **by** *simp*
qed

have *sqrt* *?pqab* = *?ab* + *sqrt* (*?ab*^{2} − *1*)
proof (*rule ccontr*)
assume *sqrt* *?pqab* ≠ *?ab* + *sqrt* (*?ab*^{2} − *1*)
with ‹*sqrt* *?pqab* = *?ab* + *sqrt* (*?ab*^{2} − *1*)
∨ *sqrt* *?pqab* = *?ab* − *sqrt* (*?ab*^{2} − *1*)›
have *sqrt* *?pqab* = *?ab* − *sqrt* (*?ab*^{2} − *1*) **by** *simp*
with ‹*?ab* − *sqrt* (*?ab*^{2} − *1*) ≤ *1*› **have** *sqrt* *?pqab* ≤ *1* **by** *simp*
with ‹*?pqab* ≥ *1*› **have** *sqrt* *?pqab* = *1* **by** *simp*
with ‹*sqrt* *?pqab* = *?ab* − *sqrt* (*?ab*^{2} − *1*)›
and ‹(*?ab* + *sqrt* (*?ab*^{2} − *1*)) ∗ (*?ab* − *sqrt* (*?ab*^{2} − *1*)) = *1*›
have *?ab* + *sqrt* (*?ab*^{2} − *1*) = *1* **by** *simp*
with ‹*sqrt* *?pqab* = *1*› **have** *sqrt* *?pqab* = *?ab* + *sqrt* (*?ab*^{2} − *1*) **by** *simp*
with ‹*sqrt* *?pqab* ≠ *?ab* + *sqrt* (*?ab*^{2} − *1*)› **show** *False* **..**
qed
moreover from ‹*?pqab* ≥ *1*› **have** *?pqab* = (*sqrt* *?pqab*)^2 **by** *simp*
ultimately have *?pqab* = (*?ab* + *sqrt* (*?ab*^{2} − *1*))^2 **by** *simp*
with ‹*sqrt* (*?ab*^{2} − *1*) ∗ *sqrt* (*?ab*^{2} − *1*) = *?ab*^{2} − *1*›
show *?pqab* = *2* ∗ *?ab*^{2} + *2* ∗ *?ab* ∗ *sqrt* (*?ab*^{2} − *1*) − *1*
by (*simp add*: *power2-eq-square algebra-simps*)
qed

lemma *are-endpoints-in-S-cross-ratio-correct*:
assumes *are-endpoints-in-S p q a b*
shows *cross-ratio-correct p q a b*
proof −
from ‹*are-endpoints-in-S p q a b*›
have *p* ≠ *q* **and** *p* ∈ *S* **and** *q* ∈ *S* **and** *a* ∈ *hyp2* **and** *b* ∈ *hyp2*
and *proj2-set-Col* {*p*,*q*,*a*,*b*}
by (*unfold are-endpoints-in-S-def*) *simp-all*

from ‹*a* ∈ *hyp2*› **and** ‹*b* ∈ *hyp2*› **and** ‹*p* ∈ *S*› **and** ‹*q* ∈ *S*›
have *a* ≠ *p* **and** *b* ≠ *p* **and** *a* ≠ *q* **by** (*simp-all add*: *hyp2-S-not-equal*)
with ‹*proj2-set-Col* {*p*,*q*,*a*,*b*}› **and** ‹*p* ≠ *q*›

show *cross-ratio-correct p q a b* **by** (*unfold cross-ratio-correct-def*) *simp*
qed

lemma *endpoints-in-S-cross-ratio-correct*:
assumes $a \neq b$ **and** $a \in hyp2$ **and** $b \in hyp2$
shows *cross-ratio-correct* (*endpoint-in-S a b*) (*endpoint-in-S b a*) *a b*
proof −
from *assms*
have *are-endpoints-in-S* (*endpoint-in-S a b*) (*endpoint-in-S b a*) *a b*
by (*rule endpoints-in-S-are-endpoints-in-S*)
thus *cross-ratio-correct* (*endpoint-in-S a b*) (*endpoint-in-S b a*) *a b*
by (*rule are-endpoints-in-S-cross-ratio-correct*)
qed

lemma *endpoints-in-S-perp-foot-cross-ratio-correct*:
assumes $a \in hyp2$ **and** $b \in hyp2$ **and** $c \in hyp2$ **and** $a \neq b$
and *proj2-incident a l* **and** *proj2-incident b l*
shows *cross-ratio-correct*
(*endpoint-in-S a b*) (*endpoint-in-S b a*) *a* (*perp-foot c l*)
(**is** *cross-ratio-correct ?p ?q a ?d*)
proof −
from *assms*
have *are-endpoints-in-S ?p ?q a ?d*
by (*rule endpoints-in-S-perp-foot-are-endpoints-in-S*)
thus *cross-ratio-correct ?p ?q a ?d*
by (*rule are-endpoints-in-S-cross-ratio-correct*)
qed

lemma *cosh-dist-unique*:
assumes $a \in hyp2$ **and** $b \in hyp2$ **and** $c \in hyp2$ **and** $p \in S$
and $B_{\mathbb{R}}$ (*cart2-pt a*) (*cart2-pt b*) (*cart2-pt p*) (**is** $B_{\mathbb{R}}$ *?ca ?cb ?cp*)
and $B_{\mathbb{R}}$ (*cart2-pt a*) (*cart2-pt c*) (*cart2-pt p*) (**is** $B_{\mathbb{R}}$ *?ca ?cc ?cp*)
and *cosh-dist a b* = *cosh-dist a c* (**is** *?ab* = *?ac*)
shows $b = c$
proof −
let *?q* = *endpoint-in-S p a*

from ‹$a \in hyp2$› **and** ‹$b \in hyp2$› **and** ‹$c \in hyp2$› **and** ‹$p \in S$›
have *z-non-zero a* **and** *z-non-zero b* **and** *z-non-zero c* **and** *z-non-zero p*
by (*simp-all add*: *hyp2-S-z-non-zero*)
with ‹$B_{\mathbb{R}}$ *?ca ?cb ?cp*› **and** ‹$B_{\mathbb{R}}$ *?ca ?cc ?cp*›
have ∃ *l*. *proj2-incident a l* ∧ *proj2-incident b l* ∧ *proj2-incident p l*
and ∃ *m*. *proj2-incident a m* ∧ *proj2-incident c m* ∧ *proj2-incident p m*
by (*simp-all add*: *euclid-B-cart2-common-line*)
then obtain *l* **and** *m* **where**
proj2-incident a l **and** *proj2-incident b l* **and** *proj2-incident p l*
and *proj2-incident a m* **and** *proj2-incident c m* **and** *proj2-incident p m*
by *auto*

from ‹$a \in hyp2$› **and** ‹$p \in S$› **have** $a \neq p$ **by** (*rule hyp2-S-not-equal*)
with ‹*proj2-incident a l*› **and** ‹*proj2-incident p l*›
and ‹*proj2-incident a m*› **and** ‹*proj2-incident p m*› **and** *proj2-incident-unique*
have $l = m$ **by** *fast*
with ‹*proj2-incident c m*› **have** *proj2-incident c l* **by** *simp*
with ‹$a \in hyp2$› **and** ‹$b \in hyp2$› **and** ‹$c \in hyp2$› **and** ‹$p \in S$›
and ‹*proj2-incident a l*› **and** ‹*proj2-incident b l*› **and** ‹*proj2-incident p l*›
have *are-endpoints-in-S p ?q b a* **and** *are-endpoints-in-S p ?q c a*
by (*simp-all add*: *end-and-opposite-are-endpoints-in-S*)
with *are-endpoints-in-S-swap-34*
have *are-endpoints-in-S p ?q a b* **and** *are-endpoints-in-S p ?q a c* **by** *fast+*
hence *cross-ratio-correct p ?q a b* **and** *cross-ratio-correct p ?q a c*
by (*simp-all add*: *are-endpoints-in-S-cross-ratio-correct*)
moreover
from ‹*are-endpoints-in-S p ?q a b*› **and** ‹*are-endpoints-in-S p ?q a c*›
and ‹$B_{\mathbb{R}}$ *?ca ?cb ?cp*› **and** ‹$B_{\mathbb{R}}$ *?ca ?cc ?cp*›
have *cross-ratio p ?q a b* $= 2 * ?ab^2 + 2 * ?ab * sqrt\ (?ab^2 - 1) - 1$
and *cross-ratio p ?q a c* $= 2 * ?ac^2 + 2 * ?ac * sqrt\ (?ac^2 - 1) - 1$
by (*simp-all add*: *cross-ratio-in-terms-of-cosh-dist*)
with ‹$?ab = ?ac$› **have** *cross-ratio p ?q a b* = *cross-ratio p ?q a c* **by** *simp*
ultimately show $b = c$ **by** (*rule cross-ratio-unique*)
qed

lemma *cosh-dist-swap*:
assumes $a \in hyp2$ **and** $b \in hyp2$
shows *cosh-dist a b* = *cosh-dist b a*
proof −
from *assms* **and** *K2-isometry-swap*
obtain J **where** *is-K2-isometry J* **and** *apply-cltn2 a J* = b
and *apply-cltn2 b J* = a
by *auto*

from ‹$b \in hyp2$› **and** ‹$a \in hyp2$› **and** ‹*is-K2-isometry J*›
have *cosh-dist* (*apply-cltn2 b J*) (*apply-cltn2 a J*) = *cosh-dist b a*
by (*rule K2-isometry-cosh-dist*)
with ‹*apply-cltn2 a J* = b› **and** ‹*apply-cltn2 b J* = a›
show *cosh-dist a b* = *cosh-dist b a* **by** *simp*
qed

lemma *exp-2dist-1-equal*:
assumes $a \in hyp2$ **and** $b \in hyp2$ **and** *exp-2dist a b* = *1*
shows $a = b$
proof (*rule ccontr*)
assume $a \neq b$
with ‹$a \in hyp2$› **and** ‹$b \in hyp2$›
have *cross-ratio-correct* (*endpoint-in-S a b*) (*endpoint-in-S b a*) *a b*
(**is** *cross-ratio-correct ?p ?q a b*)
by (*simp add*: *endpoints-in-S-cross-ratio-correct*)
moreover

from ‹$a \neq b$›
have *exp-2dist a b* = *cross-ratio ?p ?q a b* **by** (*unfold exp-2dist-def*) *simp*
with ‹*exp-2dist a b* = *1*› **have** *cross-ratio ?p ?q a b* = *1* **by** *simp*
ultimately have $a = b$ **by** (*rule cross-ratio-1-equal*)
with ‹$a \neq b$› **show** *False* **..**
qed

8.11.1 A formula for a cross ratio involving a perpendicular foot

lemma *described-perp-foot-cross-ratio-formula*:
assumes $a \neq b$ **and** $c \in hyp2$ **and** *are-endpoints-in-S p q a b*
and *proj2-incident p l* **and** *proj2-incident q l* **and** *M-perp l m*
and *proj2-incident d l* **and** *proj2-incident d m* **and** *proj2-incident c m*
shows *cross-ratio p q d a*
= (*cosh-dist b c* ∗ *sqrt* (*cross-ratio p q a b*) − *cosh-dist a c*)
/ (*cosh-dist a c* ∗ *cross-ratio p q a b*
− *cosh-dist b c* ∗ *sqrt* (*cross-ratio p q a b*))
(**is** *?pqda* = (*?bc* ∗ *sqrt ?pqab* − *?ac*) / (*?ac* ∗ *?pqab* − *?bc* ∗ *sqrt ?pqab*))
proof −
let *?da* = *cosh-dist d a*
let *?db* = *cosh-dist d b*
let *?dc* = *cosh-dist d c*
let *?pqdb* = *cross-ratio p q d b*

from ‹*are-endpoints-in-S p q a b*›
have $p \neq q$ **and** $p \in S$ **and** $q \in S$ **and** $a \in hyp2$ **and** $b \in hyp2$
and *proj2-set-Col* {*p*,*q*,*a*,*b*}
by (*unfold are-endpoints-in-S-def*) *simp-all*

from ‹*proj2-set-Col* {*p*,*q*,*a*,*b*}›
obtain l' **where** *proj2-incident p* l' **and** *proj2-incident q* l'
and *proj2-incident a* l' **and** *proj2-incident b* l'
by (*unfold proj2-set-Col-def*) *auto*

from ‹$p \neq q$› **and** ‹*proj2-incident p* l'› **and** ‹*proj2-incident q* l'›
and ‹*proj2-incident p l*› **and** ‹*proj2-incident q l*› **and** *proj2-incident-unique*
have $l' = l$ **by** *fast*
with ‹*proj2-incident a* l'› **and** ‹*proj2-incident b* l'›
have *proj2-incident a l* **and** *proj2-incident b l* **by** *simp-all*

from ‹*M-perp l m*› **and** ‹$a \in hyp2$› **and** ‹*proj2-incident a l*› **and** ‹$c \in hyp2$›
and ‹*proj2-incident c m*› **and** ‹*proj2-incident d l*› **and** ‹*proj2-incident d m*›
have $d \in hyp2$ **by** (*rule M-perp-hyp2*)
with ‹$a \in hyp2$› **and** ‹$b \in hyp2$› **and** ‹$c \in hyp2$›
have *?bc* > *0* **and** *?da* > *0* **and** *?ac* > *0*
by (*simp-all add*: *cosh-dist-positive*)

from ‹*proj2-incident p l*› **and** ‹*proj2-incident q l*› **and** ‹*proj2-incident d l*›
and ‹*proj2-incident a l*› **and** ‹*proj2-incident b l*›

have *proj2-set-Col* {p,q,d,a} **and** *proj2-set-Col* {p,q,d,b}
  and *proj2-set-Col* {p,q,a,b}
  by (*unfold proj2-set-Col-def*) (*simp-all add*: *exI* [*of - l*])
with ⟨p ≠ q⟩ **and** ⟨p ∈ S⟩ **and** ⟨q ∈ S⟩ **and** ⟨d ∈ hyp2⟩ **and** ⟨a ∈ hyp2⟩
  and ⟨b ∈ hyp2⟩
have *are-endpoints-in-S p q d a* **and** *are-endpoints-in-S p q d b*
  and *are-endpoints-in-S p q a b*
  by (*unfold are-endpoints-in-S-def*) *simp-all*
hence ?pqda > 0 **and** ?pqdb > 0 **and** ?pqab > 0
  by (*simp-all add*: *cross-ratio-S-S-hyp2-hyp2-positive*)

from ⟨proj2-incident p l⟩ **and** ⟨proj2-incident q l⟩ **and** ⟨proj2-incident a l⟩
have *proj2-Col p q a* **by** (*rule proj2-incident-Col*)

from ⟨a ∈ hyp2⟩ **and** ⟨b ∈ hyp2⟩ **and** ⟨p ∈ S⟩ **and** ⟨q ∈ S⟩
have a ≠ p **and** a ≠ q **and** b ≠ p **by** (*simp-all add*: *hyp2-S-not-equal*)

from ⟨proj2-Col p q a⟩ **and** ⟨p ≠ q⟩ **and** ⟨a ≠ p⟩ **and** ⟨a ≠ q⟩
have ?pqdb = ?pqda * ?pqab **by** (*rule cross-ratio-product* [*symmetric*])

from ⟨M-perp l m⟩ **and** ⟨a ∈ hyp2⟩ **and** ⟨b ∈ hyp2⟩ **and** ⟨c ∈ hyp2⟩ **and** ⟨d ∈ hyp2⟩
  and ⟨proj2-incident a l⟩ **and** ⟨proj2-incident b l⟩ **and** ⟨proj2-incident d l⟩
  and ⟨proj2-incident d m⟩ **and** ⟨proj2-incident c m⟩
  and *cosh-dist-perp-divide* [*of l m - d c*]
have ?dc = ?ac / ?da **and** ?dc = ?bc / ?db **by** *fast+*
hence ?ac / ?da = ?bc / ?db **by** *simp*
with ⟨?bc > 0⟩ **and** ⟨?da > 0⟩
have ?ac / ?bc = ?da / ?db **by** (*simp add*: *field-simps*)
also from ⟨are-endpoints-in-S p q d a⟩ **and** ⟨are-endpoints-in-S p q d b⟩
have ...
  = 2 * (sqrt ?pqda + 1 / (sqrt ?pqda))
  / (2 * (sqrt ?pqdb + 1 / (sqrt ?pqdb)))
  by (*simp add*: *cosh-dist-general*)
also
have ... = (sqrt ?pqda + 1 / (sqrt ?pqda)) / (sqrt ?pqdb + 1 / (sqrt ?pqdb))
  by (*simp only*: *mult-divide-mult-cancel-left-if*) *simp*
also have ...
  = sqrt ?pqdb * (sqrt ?pqda + 1 / (sqrt ?pqda))
  / (sqrt ?pqdb * (sqrt ?pqdb + 1 / (sqrt ?pqdb)))
  by *simp*
also from ⟨?pqdb > 0⟩
have ... = (sqrt (?pqdb * ?pqda) + sqrt (?pqdb / ?pqda)) / (?pqdb + 1)
  by (*simp add*: *real-sqrt-mult* [*symmetric*] *real-sqrt-divide algebra-simps*)
also from ⟨?pqdb = ?pqda * ?pqab⟩ **and** ⟨?pqda > 0⟩ **and** *real-sqrt-pow2*
have ... = (?pqda * sqrt ?pqab + sqrt ?pqab) / (?pqda * ?pqab + 1)
  by (*simp add*: *real-sqrt-mult power2-eq-square*)
finally
have ?ac / ?bc = (?pqda * sqrt ?pqab + sqrt ?pqab) / (?pqda * ?pqab + 1) **.**

from ‹*?pqda* > *0*› **and** ‹*?pqab* > *0*›
have *?pqda* ∗ *?pqab* + *1* > *0* **by** (*simp add*: *add-pos-pos*)
with ‹*?bc* > *0*›
and ‹*?ac* / *?bc* = (*?pqda* ∗ *sqrt ?pqab* + *sqrt ?pqab*) / (*?pqda* ∗ *?pqab* + *1*)›
have *?ac* ∗ (*?pqda* ∗ *?pqab* + *1*) = *?bc* ∗ (*?pqda* ∗ *sqrt ?pqab* + *sqrt ?pqab*)
by (*simp add*: *field-simps*)
hence *?pqda* ∗ (*?ac* ∗ *?pqab* − *?bc* ∗ *sqrt ?pqab*) = *?bc* ∗ *sqrt ?pqab* − *?ac*
by (*simp add*: *algebra-simps*)

from ‹*proj2-set-Col* {*p*,*q*,*a*,*b*}› **and** ‹*p* ≠ *q*› **and** ‹*a* ≠ *p*› **and** ‹*a* ≠ *q*›
and ‹*b* ≠ *p*›
have *cross-ratio-correct p q a b* **by** (*unfold cross-ratio-correct-def*) *simp*

have *?ac* ∗ *?pqab* − *?bc* ∗ *sqrt ?pqab* ≠ *0*
proof
assume *?ac* ∗ *?pqab* − *?bc* ∗ *sqrt ?pqab* = *0*
with ‹*?pqda* ∗ (*?ac* ∗ *?pqab* − *?bc* ∗ *sqrt ?pqab*) = *?bc* ∗ *sqrt ?pqab* − *?ac*›
have *?bc* ∗ *sqrt ?pqab* − *?ac* = *0* **by** *simp*
with ‹*?ac* ∗ *?pqab* − *?bc* ∗ *sqrt ?pqab* = *0*› **and** ‹*?ac* > *0*›
have *?pqab* = *1* **by** *simp*
with ‹*cross-ratio-correct p q a b*›
have *a* = *b* **by** (*rule cross-ratio-1-equal*)
with ‹*a* ≠ *b*› **show** *False* **..**
qed
with ‹*?pqda* ∗ (*?ac* ∗ *?pqab* − *?bc* ∗ *sqrt ?pqab*) = *?bc* ∗ *sqrt ?pqab* − *?ac*›
show *?pqda* = (*?bc* ∗ *sqrt ?pqab* − *?ac*) / (*?ac* ∗ *?pqab* − *?bc* ∗ *sqrt ?pqab*)
by (*simp add*: *field-simps*)
qed

lemma *perp-foot-cross-ratio-formula*:
assumes *a* ∈ *hyp2* **and** *b* ∈ *hyp2* **and** *c* ∈ *hyp2* **and** *a* ≠ *b*
shows *cross-ratio* (*endpoint-in-S a b*) (*endpoint-in-S b a*)
(*perp-foot c* (*proj2-line-through a b*)) *a*
= (*cosh-dist b c* ∗ *sqrt* (*exp-2dist a b*) − *cosh-dist a c*)
/ (*cosh-dist a c* ∗ *exp-2dist a b* − *cosh-dist b c* ∗ *sqrt* (*exp-2dist a b*))
(**is** *cross-ratio ?p ?q ?d a*
= (*?bc* ∗ *sqrt ?pqab* − *?ac*) / (*?ac* ∗ *?pqab* − *?bc* ∗ *sqrt ?pqab*))
proof −
from ‹*a* ≠ *b*› **and** ‹*a* ∈ *hyp2*› **and** ‹*b* ∈ *hyp2*›
have *are-endpoints-in-S ?p ?q a b*
by (*rule endpoints-in-S-are-endpoints-in-S*)

let *?l* = *proj2-line-through a b*
have *proj2-incident a ?l* **and** *proj2-incident b ?l*
by (*rule proj2-line-through-incident*)+
with ‹*a* ≠ *b*› **and** ‹*a* ∈ *hyp2*› **and** ‹*b* ∈ *hyp2*›
have *proj2-incident ?p ?l* **and** *proj2-incident ?q ?l*
by (*simp-all add*: *endpoint-in-S-incident*)

let *?m* = *drop-perp c ?l*
have *M-perp ?l ?m* **by** (*rule drop-perp-perp*)

have *proj2-incident ?d ?l* **and** *proj2-incident ?d ?m*
by (*rule perp-foot-incident*)+

have *proj2-incident c ?m* **by** (*rule drop-perp-incident*)
with ‹$a \neq b$› **and** ‹$c \in$ *hyp2*› **and** ‹*are-endpoints-in-S ?p ?q a b*›
and ‹*proj2-incident ?p ?l*› **and** ‹*proj2-incident ?q ?l*› **and** ‹*M-perp ?l ?m*›
and ‹*proj2-incident ?d ?l*› **and** ‹*proj2-incident ?d ?m*›
have *cross-ratio ?p ?q ?d a*
= (*?bc* ∗ *sqrt* (*cross-ratio ?p ?q a b*) − *?ac*)
/ (*?ac* ∗ (*cross-ratio ?p ?q a b*) − *?bc* ∗ *sqrt* (*cross-ratio ?p ?q a b*))
by (*rule described-perp-foot-cross-ratio-formula*)
with ‹$a \neq b$›
show *cross-ratio ?p ?q ?d a*
= (*?bc* ∗ *sqrt ?pqab* − *?ac*) / (*?ac* ∗ *?pqab* − *?bc* ∗ *sqrt ?pqab*)
by (*unfold exp-2dist-def*) *simp*
qed

8.12 The Klein–Beltrami model satisfies axiom 5

lemma *statement69*:
assumes $a\ b \equiv_K a'\ b'$ **and** $b\ c \equiv_K b'\ c'$ **and** $a\ c \equiv_K a'\ c'$
shows ∃ *J*. *is-K2-isometry J*
∧ *hyp2-cltn2 a J* = a' ∧ *hyp2-cltn2 b J* = b' ∧ *hyp2-cltn2 c J* = c'
proof *cases*
assume $a = b$
with ‹$a\ b \equiv_K a'\ b'$› **have** $a' = b'$ **by** (*simp add*: *hyp2.A3-reversed*)
with ‹$a = b$› **and** ‹$b\ c \equiv_K b'\ c'$›
show ∃ *J*. *is-K2-isometry J*
∧ *hyp2-cltn2 a J* = a' ∧ *hyp2-cltn2 b J* = b' ∧ *hyp2-cltn2 c J* = c'
by (*unfold real-hyp2-C-def*) *simp*
next
assume $a \neq b$
with ‹$a\ b \equiv_K a'\ b'$›
have $a' \neq b'$ **by** (*auto simp add*: *hyp2.A3'*)

let *?pa* = *Rep-hyp2 a*
and *?pb* = *Rep-hyp2 b*
and *?pc* = *Rep-hyp2 c*
and *?pa'* = *Rep-hyp2 a'*
and *?pb'* = *Rep-hyp2 b'*
and *?pc'* = *Rep-hyp2 c'*
define *pp pq l pp' pq' l'*
where *pp* = *endpoint-in-S ?pa ?pb*
and *pq* = *endpoint-in-S ?pb ?pa*
and *l* = *proj2-line-through ?pa ?pb*

and *pp'* = *endpoint-in-S ?pa' ?pb'*
and *pq'* = *endpoint-in-S ?pb' ?pa'*
and *l'* = *proj2-line-through ?pa' ?pb'*
define *pd ps m pd' ps' m'*
where *pd* = *perp-foot ?pc l*
and *ps* = *perp-up ?pc l*
and *m* = *drop-perp ?pc l*
and *pd'* = *perp-foot ?pc' l'*
and *ps'* = *perp-up ?pc' l'*
and *m'* = *drop-perp ?pc' l'*

have *pp* ∈ *S* **and** *pp'* ∈ *S* **and** *pq* ∈ *S* **and** *pq'* ∈ *S*
unfolding *pp-def* **and** *pp'-def* **and** *pq-def* **and** *pq'-def*
by (*simp-all add*: *Rep-hyp2 endpoint-in-S*)

from ‹*a* ≠ *b*› **and** ‹*a'* ≠ *b'*›
have *?pa* ≠ *?pb* **and** *?pa'* ≠ *?pb'* **by** (*unfold Rep-hyp2-inject*)
moreover
have *proj2-incident ?pa l* **and** *proj2-incident ?pb l*
and *proj2-incident ?pa' l'* **and** *proj2-incident ?pb' l'*
by (*unfold l-def l'-def*) (*rule proj2-line-through-incident*)+
ultimately have *proj2-incident pp l* **and** *proj2-incident pp' l'*
and *proj2-incident pq l* **and** *proj2-incident pq' l'*
unfolding *pp-def* **and** *pp'-def* **and** *pq-def* **and** *pq'-def*
by (*simp-all add*: *Rep-hyp2 endpoint-in-S-incident*)

from ‹*pp* ∈ *S*› **and** ‹*pp'* ∈ *S*› **and** ‹*proj2-incident pp l*›
and ‹*proj2-incident pp' l'*› **and** ‹*proj2-incident ?pa l*›
and ‹*proj2-incident ?pa' l'*›
have *right-angle pp pd ps* **and** *right-angle pp' pd' ps'*
unfolding *pd-def* **and** *ps-def* **and** *pd'-def* **and** *ps'-def*
by (*simp-all add*: *Rep-hyp2*
perp-foot-up-right-angle [*of pp ?pc ?pa l*]
perp-foot-up-right-angle [*of pp' ?pc' ?pa' l'*])
with *right-angle-to-right-angle* [*of pp pd ps pp' pd' ps'*]
obtain *J* **where** *is-K2-isometry J* **and** *apply-cltn2 pp J* = *pp'*
and *apply-cltn2 pd J* = *pd'* **and** *apply-cltn2 ps J* = *ps'*
by *auto*

let *?paJ* = *apply-cltn2 ?pa J*
and *?pbJ* = *apply-cltn2 ?pb J*
and *?pcJ* = *apply-cltn2 ?pc J*
and *?pdJ* = *apply-cltn2 pd J*
and *?ppJ* = *apply-cltn2 pp J*
and *?pqJ* = *apply-cltn2 pq J*
and *?psJ* = *apply-cltn2 ps J*
and *?lJ* = *apply-cltn2-line l J*
and *?mJ* = *apply-cltn2-line m J*

have *proj2-incident pd l* **and** *proj2-incident pd′ l′*
  and *proj2-incident pd m* **and** *proj2-incident pd′ m′*
  by (*unfold pd-def pd′-def m-def m′-def*) (*rule perp-foot-incident*)+

from ‹*proj2-incident pp l*› **and** ‹*proj2-incident pq l*›
  and ‹*proj2-incident pd l*› **and** ‹*proj2-incident ?pa l*›
  and ‹*proj2-incident ?pb l*›
have *proj2-set-Col* {*pp*,*pq*,*pd*,*?pa*} **and** *proj2-set-Col* {*pp*,*pq*,*?pa*,*?pb*}
  by (*unfold pd-def proj2-set-Col-def*) (*simp-all add*: *exI* [*of* - *l*])

from ‹*?pa* ≠ *?pb*› **and** ‹*?pa′* ≠ *?pb′*›
have *pp* ≠ *pq* **and** *pp′* ≠ *pq′*
  unfolding *pp-def* **and** *pq-def* **and** *pp′-def* **and** *pq′-def*
  by (*simp-all add*: *Rep-hyp2 endpoint-in-S-swap*)

from ‹*proj2-incident ?pa l*› **and** ‹*proj2-incident ?pa′ l′*›
have *pd* ∈ *hyp2* **and** *pd′* ∈ *hyp2*
  unfolding *pd-def* **and** *pd′-def*
  by (*simp-all add*: *Rep-hyp2 perp-foot-hyp2* [*of ?pa l ?pc*]
    perp-foot-hyp2 [*of ?pa′ l′ ?pc′*])

from ‹*proj2-incident ?pa l*› **and** ‹*proj2-incident ?pa′ l′*›
have *ps* ∈ *S* **and** *ps′* ∈ *S*
  unfolding *ps-def* **and** *ps′-def*
  by (*simp-all add*: *Rep-hyp2 perp-up-in-S* [*of ?pc ?pa l*]
    perp-up-in-S [*of ?pc′ ?pa′ l′*])

from ‹*pd* ∈ *hyp2*› **and** ‹*pp* ∈ *S*› **and** ‹*ps* ∈ *S*›
have *pd* ≠ *pp* **and** *?pa* ≠ *pp* **and** *?pb* ≠ *pp* **and** *pd* ≠ *ps*
  by (*simp-all add*: *Rep-hyp2 hyp2-S-not-equal*)

from ‹*is-K2-isometry J*› **and** ‹*pq* ∈ *S*›
have *?pqJ* ∈ *S* **by** (*unfold is-K2-isometry-def*) *simp*

from ‹*pd* ≠ *pp*› **and** ‹*proj2-incident pd l*› **and** ‹*proj2-incident pp l*›
  and ‹*proj2-incident pd′ l′*› **and** ‹*proj2-incident pp′ l′*›
have *?lJ* = *l′*
  unfolding ‹*?pdJ* = *pd′*› [*symmetric*] **and** ‹*?ppJ* = *pp′*› [*symmetric*]
  by (*rule apply-cltn2-line-unique*)
from ‹*proj2-incident pq l*› **and** ‹*proj2-incident ?pa l*›
  and ‹*proj2-incident ?pb l*›
have *proj2-incident ?pqJ l′* **and** *proj2-incident ?paJ l′*
  and *proj2-incident ?pbJ l′*
  by (*unfold* ‹*?lJ* = *l′*› [*symmetric*]) *simp-all*

from ‹*?pa′* ≠ *?pb′*› **and** ‹*?pqJ* ∈ *S*› **and** ‹*proj2-incident ?pa′ l′*›
  and ‹*proj2-incident ?pb′ l′*› **and** ‹*proj2-incident ?pqJ l′*›
have *?pqJ* = *pp′* ∨ *?pqJ* = *pq′*
  unfolding *pp′-def* **and** *pq′-def*

by (*simp add*: *Rep-hyp2 endpoints-in-S-incident-unique*)
moreover
from ‹*pp* ≠ *pq*› **and** *apply-cltn2-injective*
have *pp*′ ≠ *?pqJ* **by** (*unfold* ‹*?ppJ* = *pp*′› [*symmetric*]) *fast*
ultimately have *?pqJ* = *pq*′ **by** *simp*

from ‹*?pa*′ ≠ *?pb*′›
have *cross-ratio pp*′ *pq*′ *pd*′ *?pa*′
= (*cosh-dist ?pb*′ *?pc*′ ∗ *sqrt* (*exp-2dist ?pa*′ *?pb*′) − *cosh-dist ?pa*′ *?pc*′)
/ (*cosh-dist ?pa*′ *?pc*′ ∗ *exp-2dist ?pa*′ *?pb*′
− *cosh-dist ?pb*′ *?pc*′ ∗ *sqrt* (*exp-2dist ?pa*′ *?pb*′))
unfolding *pp*′*-def* **and** *pq*′*-def* **and** *pd*′*-def* **and** *l*′*-def*
by (*simp add*: *Rep-hyp2 perp-foot-cross-ratio-formula*)
also from *assms*
have . . . = (*cosh-dist ?pb ?pc* ∗ *sqrt* (*exp-2dist ?pa ?pb*) − *cosh-dist ?pa ?pc*)
/ (*cosh-dist ?pa ?pc* ∗ *exp-2dist ?pa ?pb*
− *cosh-dist ?pb ?pc* ∗ *sqrt* (*exp-2dist ?pa ?pb*))
by (*simp add*: *real-hyp2-C-exp-2dist real-hyp2-C-cosh-dist*)
also from ‹*?pa* ≠ *?pb*›
have . . . = *cross-ratio pp pq pd ?pa*
unfolding *pp-def* **and** *pq-def* **and** *pd-def* **and** *l-def*
by (*simp add*: *Rep-hyp2 perp-foot-cross-ratio-formula*)
also from ‹*proj2-set-Col* {*pp*,*pq*,*pd*,*?pa*}› **and** ‹*pp* ≠ *pq*› **and** ‹*pd* ≠ *pp*›
and ‹*?pa* ≠ *pp*›
have . . . = *cross-ratio ?ppJ ?pqJ ?pdJ ?paJ* **by** (*simp add*: *cross-ratio-cltn2*)
also from ‹*?ppJ* = *pp*′› **and** ‹*?pqJ* = *pq*′› **and** ‹*?pdJ* = *pd*′›
have . . . = *cross-ratio pp*′ *pq*′ *pd*′ *?paJ* **by** *simp*
finally
have *cross-ratio pp*′ *pq*′ *pd*′ *?paJ* = *cross-ratio pp*′ *pq*′ *pd*′ *?pa*′ **by** *simp*

from ‹*is-K2-isometry J*›
have *?paJ* ∈ *hyp2* **and** *?pbJ* ∈ *hyp2* **and** *?pcJ* ∈ *hyp2*
by (*rule apply-cltn2-Rep-hyp2*)+

from ‹*proj2-incident pp*′ *l*′› **and** ‹*proj2-incident pq*′ *l*′›
and ‹*proj2-incident pd*′ *l*′› **and** ‹*proj2-incident ?paJ l*′›
and ‹*proj2-incident ?pa*′ *l*′› **and** ‹*proj2-incident ?pbJ l*′›
and ‹*proj2-incident ?pb*′ *l*′›
have *proj2-set-Col* {*pp*′,*pq*′,*pd*′,*?paJ*} **and** *proj2-set-Col* {*pp*′,*pq*′,*pd*′,*?pa*′}
and *proj2-set-Col* {*pp*′,*pq*′,*?pa*′,*?pbJ*}
and *proj2-set-Col* {*pp*′,*pq*′,*?pa*′,*?pb*′}
by (*unfold proj2-set-Col-def*) (*simp-all add*: *exI* [*of* - *l*′])
with ‹*pp*′ ≠ *pq*′› **and** ‹*pp*′ ∈ *S*› **and** ‹*pq*′ ∈ *S*› **and** ‹*pd*′ ∈ *hyp2*›
and ‹*?paJ* ∈ *hyp2*› **and** ‹*?pbJ* ∈ *hyp2*›
have *are-endpoints-in-S pp*′ *pq*′ *pd*′ *?paJ*
and *are-endpoints-in-S pp*′ *pq*′ *pd*′ *?pa*′
and *are-endpoints-in-S pp*′ *pq*′ *?pa*′ *?pbJ*
and *are-endpoints-in-S pp*′ *pq*′ *?pa*′ *?pb*′
by (*unfold are-endpoints-in-S-def*) (*simp-all add*: *Rep-hyp2*)

hence *cross-ratio-correct pp′ pq′ pd′ ?paJ*
and *cross-ratio-correct pp′ pq′ pd′ ?pa′*
and *cross-ratio-correct pp′ pq′ ?pa′ ?pbJ*
and *cross-ratio-correct pp′ pq′ ?pa′ ?pb′*
by (*simp-all add*: *are-endpoints-in-S-cross-ratio-correct*)

from ⟨*cross-ratio-correct pp′ pq′ pd′ ?paJ*⟩
and ⟨*cross-ratio-correct pp′ pq′ pd′ ?pa′*⟩
and ⟨*cross-ratio pp′ pq′ pd′ ?paJ* = *cross-ratio pp′ pq′ pd′ ?pa′*⟩
have *?paJ* = *?pa′* **by** (*simp add*: *cross-ratio-unique*)
with ⟨*?ppJ* = *pp′*⟩ **and** ⟨*?pqJ* = *pq′*⟩
have *cross-ratio pp′ pq′ ?pa′ ?pbJ* = *cross-ratio ?ppJ ?pqJ ?paJ ?pbJ* **by** *simp*
also from ⟨*proj2-set-Col* {*pp*,*pq*,*?pa*,*?pb*}⟩ **and** ⟨*pp* ≠ *pq*⟩ **and** ⟨*?pa* ≠ *pp*⟩
and ⟨*?pb* ≠ *pp*⟩
have . . . = *cross-ratio pp pq ?pa ?pb* **by** (*rule cross-ratio-cltn2*)
also from ⟨$a \neq b$⟩ **and** ⟨$a\ b \equiv_K a'\ b'$⟩
have . . . = *cross-ratio pp′ pq′ ?pa′ ?pb′*
unfolding *pp-def pq-def pp′-def pq′-def*
by (*rule real-hyp2-C-cross-ratio-endpoints-in-S*)
finally have *cross-ratio pp′ pq′ ?pa′ ?pbJ* = *cross-ratio pp′ pq′ ?pa′ ?pb′* **.**
with ⟨*cross-ratio-correct pp′ pq′ ?pa′ ?pbJ*⟩
and ⟨*cross-ratio-correct pp′ pq′ ?pa′ ?pb′*⟩
have *?pbJ* = *?pb′* **by** (*rule cross-ratio-unique*)

let *?cc* = *cart2-pt ?pc*
and *?cd* = *cart2-pt pd*
and *?cs* = *cart2-pt ps*
and *?cc′* = *cart2-pt ?pc′*
and *?cd′* = *cart2-pt pd′*
and *?cs′* = *cart2-pt ps′*
and *?ccJ* = *cart2-pt ?pcJ*
and *?cdJ* = *cart2-pt ?pdJ*
and *?csJ* = *cart2-pt ?psJ*

from ⟨*proj2-incident ?pa l*⟩ **and** ⟨*proj2-incident ?pa′ l′*⟩
have $B_{\mathbb{R}}$ *?cd ?cc ?cs* **and** $B_{\mathbb{R}}$ *?cd′ ?cc′ ?cs′*
unfolding *pd-def* **and** *ps-def* **and** *pd′-def* **and** *ps′-def*
by (*simp-all add*: *Rep-hyp2 perp-up-at-end* [*of ?pc ?pa l*]
perp-up-at-end [*of ?pc′ ?pa′ l′*])

from ⟨$pd \in hyp2$⟩ **and** ⟨$ps \in S$⟩ **and** ⟨*is-K2-isometry J*⟩
and ⟨$B_{\mathbb{R}}$ *?cd ?cc ?cs*⟩
have $B_{\mathbb{R}}$ *?cdJ ?ccJ ?csJ* **by** (*simp add*: *Rep-hyp2 statement-63*)
hence $B_{\mathbb{R}}$ *?cd′ ?ccJ ?cs′* **by** (*unfold* ⟨*?pdJ* = *pd′*⟩ ⟨*?psJ* = *ps′*⟩)

from ⟨*?paJ* = *?pa′*⟩ **have** *cosh-dist ?pa′ ?pcJ* = *cosh-dist ?paJ ?pcJ* **by** *simp*
also from ⟨*is-K2-isometry J*⟩
have . . . = *cosh-dist ?pa ?pc* **by** (*simp add*: *Rep-hyp2 K2-isometry-cosh-dist*)
also from ⟨$a\ c \equiv_K a'\ c'$⟩

have ... = *cosh-dist ?pa′ ?pc′* **by** (*rule real-hyp2-C-cosh-dist*)
finally have *cosh-dist ?pa′ ?pcJ* = *cosh-dist ?pa′ ?pc′* **.**

have *M-perp l′ m′* **by** (*unfold m′-def*) (*rule drop-perp-perp*)

have *proj2-incident ?pc m* **and** *proj2-incident ?pc′ m′*
by (*unfold m-def m′-def*) (*rule drop-perp-incident*)+

from ‹*proj2-incident ?pa l*› **and** ‹*proj2-incident ?pa′ l′*›
have *proj2-incident ps m* **and** *proj2-incident ps′ m′*
unfolding *ps-def* **and** *m-def* **and** *ps′-def* **and** *m′-def*
by (*simp-all add*: *Rep-hyp2 perp-up-incident* [*of ?pc ?pa l*]
perp-up-incident [*of ?pc′ ?pa′ l′*])
with ‹*pd* ≠ *ps*› **and** ‹*proj2-incident pd m*› **and** ‹*proj2-incident pd′ m′*›
have *?mJ* = *m′*
unfolding ‹*?pdJ* = *pd′*› [*symmetric*] **and** ‹*?psJ* = *ps′*› [*symmetric*]
by (*simp add*: *apply-cltn2-line-unique*)
from ‹*proj2-incident ?pc m*›
have *proj2-incident ?pcJ m′* **by** (*unfold* ‹*?mJ* = *m′*› [*symmetric*]) *simp*
with ‹*M-perp l′ m′*› **and** *Rep-hyp2* [*of a′*] **and** ‹*pd′* ∈ *hyp2*› **and** ‹*?pcJ* ∈ *hyp2*›
and *Rep-hyp2* [*of c′*] **and** ‹*proj2-incident ?pa′ l′*›
and ‹*proj2-incident pd′ l′*› **and** ‹*proj2-incident pd′ m′*›
and ‹*proj2-incident ?pc′ m′*›
have *cosh-dist pd′ ?pcJ* = *cosh-dist ?pa′ ?pcJ* / *cosh-dist pd′ ?pa′*
and *cosh-dist pd′ ?pc′* = *cosh-dist ?pa′ ?pc′* / *cosh-dist pd′ ?pa′*
by (*simp-all add*: *cosh-dist-perp-divide*)
with ‹*cosh-dist ?pa′ ?pcJ* = *cosh-dist ?pa′ ?pc′*›
have *cosh-dist pd′ ?pcJ* = *cosh-dist pd′ ?pc′* **by** *simp*
with ‹*pd′* ∈ *hyp2*› **and** ‹*?pcJ* ∈ *hyp2*› **and** ‹*?pc′* ∈ *hyp2*› **and** ‹*ps′* ∈ *S*›
and ‹$B_{\mathbb{R}}$ *?cd′ ?ccJ ?cs′*› **and** ‹$B_{\mathbb{R}}$ *?cd′ ?cc′ ?cs′*›
have *?pcJ* = *?pc′* **by** (*rule cosh-dist-unique*)
with ‹*?paJ* = *?pa′*› **and** ‹*?pbJ* = *?pb′*›
have *hyp2-cltn2 a J* = *a′* **and** *hyp2-cltn2 b J* = *b′* **and** *hyp2-cltn2 c J* = *c′*
by (*unfold hyp2-cltn2-def*) (*simp-all add*: *Rep-hyp2-inverse*)
with ‹*is-K2-isometry J*›
show ∃ *J*. *is-K2-isometry J*
∧ *hyp2-cltn2 a J* = *a′* ∧ *hyp2-cltn2 b J* = *b′* ∧ *hyp2-cltn2 c J* = *c′*
by (*simp add*: *exI* [*of - J*])
qed

theorem *hyp2-axiom5*:
∀ *a b c d a′ b′ c′ d′*.
$a \neq b \wedge B_K\ a\ b\ c \wedge B_K\ a'\ b'\ c' \wedge a\ b \equiv_K a'\ b' \wedge b\ c \equiv_K b'\ c'$
$\wedge\ a\ d \equiv_K a'\ d' \wedge b\ d \equiv_K b'\ d'$
$\longrightarrow c\ d \equiv_K c'\ d'$
proof *standard*+
fix *a b c d a′ b′ c′ d′*
assume $a \neq b \wedge B_K\ a\ b\ c \wedge B_K\ a'\ b'\ c' \wedge a\ b \equiv_K a'\ b' \wedge b\ c \equiv_K b'\ c'$
$\wedge\ a\ d \equiv_K a'\ d' \wedge b\ d \equiv_K b'\ d'$

hence $a \neq b$ **and** $B_K\ a\ b\ c$ **and** $B_K\ a'\ b'\ c'$ **and** $a\ b \equiv_K\ a'\ b'$
and $b\ c \equiv_K\ b'\ c'$ **and** $a\ d \equiv_K\ a'\ d'$ **and** $b\ d \equiv_K\ b'\ d'$
by *simp-all*

from ‹$a\ b \equiv_K\ a'\ b'$› **and** ‹$b\ d \equiv_K\ b'\ d'$› **and** ‹$a\ d \equiv_K\ a'\ d'$› **and** *statement69* [*of* $a\ b\ a'\ b'\ d\ d'$]
obtain J **where** *is-K2-isometry* J **and** *hyp2-cltn2* $a\ J = a'$
and *hyp2-cltn2* $b\ J = b'$ **and** *hyp2-cltn2* $d\ J = d'$
by *auto*

let *?aJ* = *hyp2-cltn2* $a\ J$
and *?bJ* = *hyp2-cltn2* $b\ J$
and *?cJ* = *hyp2-cltn2* $c\ J$
and *?dJ* = *hyp2-cltn2* $d\ J$

from ‹$a \neq b$› **and** ‹$a\ b \equiv_K\ a'\ b'$›
have $a' \neq b'$ **by** (*auto simp add*: *hyp2.A3'*)

from ‹*is-K2-isometry* J› **and** ‹$B_K\ a\ b\ c$›
have B_K *?aJ ?bJ ?cJ* **by** (*rule real-hyp2-B-hyp2-cltn2*)
hence $B_K\ a'\ b'$ *?cJ* **by** (*unfold* ‹*?aJ* = a'› ‹*?bJ* = b'›)

from ‹*is-K2-isometry* J›
have $b\ c \equiv_K$ *?bJ ?cJ* **by** (*rule real-hyp2-C-hyp2-cltn2*)
hence $b\ c \equiv_K\ b'$ *?cJ* **by** (*unfold* ‹*?bJ* = b'›)
from *this* **and** ‹$b\ c \equiv_K\ b'\ c'$› **have** b' *?cJ* $\equiv_K\ b'\ c'$ **by** (*rule hyp2.A2'*)
with ‹$a' \neq b'$› **and** ‹$B_K\ a'\ b'$ *?cJ*› **and** ‹$B_K\ a'\ b'\ c'$›
have *?cJ* = c' **by** (*rule hyp2-extend-segment-unique*)
from ‹*is-K2-isometry* J›
show $c\ d \equiv_K\ c'\ d'$
unfolding ‹*?cJ* = c'› [*symmetric*] **and** ‹*?dJ* = d'› [*symmetric*]
by (*rule real-hyp2-C-hyp2-cltn2*)
qed

interpretation *hyp2*: *tarski-first5 real-hyp2-C real-hyp2-B*
using *hyp2-axiom4* **and** *hyp2-axiom5*
by *unfold-locales*

8.13 The Klein–Beltrami model satisfies axioms 6, 7, and 11

theorem *hyp2-axiom6*: $\forall\ a\ b.\ B_K\ a\ b\ a \longrightarrow a = b$
proof *standard+*
fix $a\ b$
let *?ca* = *cart2-pt* (*Rep-hyp2* a)
and *?cb* = *cart2-pt* (*Rep-hyp2* b)
assume $B_K\ a\ b\ a$
hence $B_{\mathbb{R}}$ *?ca ?cb ?ca* **by** (*unfold real-hyp2-B-def hyp2-rep-def*)
hence *?ca* = *?cb* **by** (*rule real-euclid.A6'*)
hence *Rep-hyp2* a = *Rep-hyp2* b **by** (*simp add*: *Rep-hyp2 hyp2-S-cart2-inj*)

thus $a = b$ **by** (*unfold Rep-hyp2-inject*)
qed

lemma *between-inverse*:
assumes $B_{\mathbb{R}}$ (*hyp2-rep p*) *v* (*hyp2-rep q*)
shows *hyp2-rep* (*hyp2-abs v*) = *v*
proof −
let *?u* = *hyp2-rep p*
let *?w* = *hyp2-rep q*
have *norm ?u* < *1* **and** *norm ?w* < *1* **by** (*rule norm-hyp2-rep-lt-1*)+

from ‹$B_{\mathbb{R}}$ *?u v ?w*›
obtain *l* **where** $l \geq 0$ **and** $l \leq 1$ **and** *v* − *?u* = *l* $*_R$ (*?w* − *?u*)
by (*unfold real-euclid-B-def*) *auto*
from ‹*v* − *?u* = *l* $*_R$ (*?w* − *?u*)›
have *v* = *l* $*_R$ *?w* + (*1* − *l*) $*_R$ *?u* **by** (*simp add*: *algebra-simps*)
hence *norm v* ≤ *norm* (*l* $*_R$ *?w*) + *norm* ((*1* − *l*) $*_R$ *?u*)
by (*simp only*: *norm-triangle-ineq* [*of l* $*_R$ *?w* (*1* − *l*) $*_R$ *?u*])
with ‹$l \geq 0$› **and** ‹$l \leq 1$›
have *norm v* ≤ *l* $*_R$ *norm ?w* + (*1* − *l*) $*_R$ *norm ?u* **by** *simp*

have *norm v* < *1*
proof *cases*
assume *l* = *0*
with ‹*v* = *l* $*_R$ *?w* + (*1* − *l*) $*_R$ *?u*›
have *v* = *?u* **by** *simp*
with ‹*norm ?u* < *1*› **show** *norm v* < *1* **by** *simp*
next
assume $l \neq 0$
with ‹*norm ?w* < *1*› **and** ‹$l \geq 0$› **have** *l* $*_R$ *norm ?w* < *l* **by** *simp*

with ‹*norm ?u* < *1*› **and** ‹$l \leq 1$›
and *mult-mono* [*of 1* − *l 1* − *l norm ?u 1*]
have (*1* − *l*) $*_R$ *norm ?u* ≤ *1* − *l* **by** *simp*
with ‹*l* $*_R$ *norm ?w* < *l*›
have *l* $*_R$ *norm ?w* + (*1* − *l*) $*_R$ *norm ?u* < *1* **by** *simp*
with ‹*norm v* ≤ *l* $*_R$ *norm ?w* + (*1* − *l*) $*_R$ *norm ?u*›
show *norm v* < *1* **by** *simp*
qed
thus *hyp2-rep* (*hyp2-abs v*) = *v* **by** (*rule hyp2-rep-abs*)
qed

lemma *between-switch*:
assumes $B_{\mathbb{R}}$ (*hyp2-rep p*) *v* (*hyp2-rep q*)
shows B_K *p* (*hyp2-abs v*) *q*
proof −
from *assms* **have** *hyp2-rep* (*hyp2-abs v*) = *v* **by** (*rule between-inverse*)
with *assms* **show** B_K *p* (*hyp2-abs v*) *q* **by** (*unfold real-hyp2-B-def*) *simp*
qed

theorem *hyp2-axiom7*:
  $\forall\ a\ b\ c\ p\ q.\ B_K\ a\ p\ c \wedge B_K\ b\ q\ c \longrightarrow (\exists\ x.\ B_K\ p\ x\ b \wedge B_K\ q\ x\ a)$
proof *auto*
  fix *a b c p q*
  let *?ca* = *hyp2-rep a*
    and *?cb* = *hyp2-rep b*
    and *?cc* = *hyp2-rep c*
    and *?cp* = *hyp2-rep p*
    and *?cq* = *hyp2-rep q*
  assume $B_K\ a\ p\ c$ **and** $B_K\ b\ q\ c$
  hence $B_{\mathbb{R}}$ *?ca ?cp ?cc* **and** $B_{\mathbb{R}}$ *?cb ?cq ?cc* **by** (*unfold real-hyp2-B-def*)
  with *real-euclid.A7′* [*of ?ca ?cp ?cc ?cb ?cq*]
  obtain *cx* **where** $B_{\mathbb{R}}$ *?cp cx ?cb* **and** $B_{\mathbb{R}}$ *?cq cx ?ca* **by** *auto*
  hence B_K *p* (*hyp2-abs cx*) *b* **and** B_K *q* (*hyp2-abs cx*) *a*
    by (*simp-all add*: *between-switch*)
  thus $\exists\ x.\ B_K\ p\ x\ b \wedge B_K\ q\ x\ a$ **by** (*simp add*: *exI* [*of - hyp2-abs cx*])
qed

theorem *hyp2-axiom11*:
  $\forall\ X\ Y.\ (\exists\ a.\ \forall\ x\ y.\ x \in X \wedge y \in Y \longrightarrow B_K\ a\ x\ y)$
  $\longrightarrow (\exists\ b.\ \forall\ x\ y.\ x \in X \wedge y \in Y \longrightarrow B_K\ x\ b\ y)$
proof (*rule allI*)+
  fix *X Y* :: *hyp2 set*
  show $(\exists\ a.\ \forall\ x\ y.\ x \in X \wedge y \in Y \longrightarrow B_K\ a\ x\ y)$
    $\longrightarrow (\exists\ b.\ \forall\ x\ y.\ x \in X \wedge y \in Y \longrightarrow B_K\ x\ b\ y)$
  proof *cases*
    assume $X = \{\} \vee Y = \{\}$
    thus $(\exists\ a.\ \forall\ x\ y.\ x \in X \wedge y \in Y \longrightarrow B_K\ a\ x\ y)$
      $\longrightarrow (\exists\ b.\ \forall\ x\ y.\ x \in X \wedge y \in Y \longrightarrow B_K\ x\ b\ y)$ **by** *auto*
  next
    assume $\neg\ (X = \{\} \vee Y = \{\})$
    hence $X \neq \{\}$ **and** $Y \neq \{\}$ **by** *simp-all*
    then obtain *w* **and** *z* **where** $w \in X$ **and** $z \in Y$ **by** *auto*

    show $(\exists\ a.\ \forall\ x\ y.\ x \in X \wedge y \in Y \longrightarrow B_K\ a\ x\ y)$
      $\longrightarrow (\exists\ b.\ \forall\ x\ y.\ x \in X \wedge y \in Y \longrightarrow B_K\ x\ b\ y)$
    proof
      assume $\exists\ a.\ \forall\ x\ y.\ x \in X \wedge y \in Y \longrightarrow B_K\ a\ x\ y$
      then obtain *a* **where** $\forall\ x\ y.\ x \in X \wedge y \in Y \longrightarrow B_K\ a\ x\ y$ **..**

      let *?cX* = *hyp2-rep* ‘ *X*
        and *?cY* = *hyp2-rep* ‘ *Y*
        and *?ca* = *hyp2-rep a*
        and *?cw* = *hyp2-rep w*
        and *?cz* = *hyp2-rep z*

      from $\langle\forall\ x\ y.\ x \in X \wedge y \in Y \longrightarrow B_K\ a\ x\ y\rangle$
      have $\forall$ *cx cy*. *cx* $\in$ *?cX* $\wedge$ *cy* $\in$ *?cY* $\longrightarrow B_{\mathbb{R}}$ *?ca cx cy*

by (*unfold real-hyp2-B-def*) *auto*
with *real-euclid.A11′* [*of ?cX ?cY ?ca*]
obtain *cb* **where** $\forall$ *cx cy*. *cx* $\in$ *?cX* $\wedge$ *cy* $\in$ *?cY* $\longrightarrow$ $B_{\mathbb{R}}$ *cx cb cy* **by** *auto*
with ‹*w* $\in$ *X*› **and** ‹*z* $\in$ *Y*› **have** $B_{\mathbb{R}}$ *?cw cb ?cz* **by** *simp*
hence *hyp2-rep* (*hyp2-abs cb*) = *cb* (**is** *hyp2-rep ?b* = *cb*)
by (*rule between-inverse*)
with ‹$\forall$ *cx cy*. *cx* $\in$ *?cX* $\wedge$ *cy* $\in$ *?cY* $\longrightarrow$ $B_{\mathbb{R}}$ *cx cb cy*›
have $\forall$ *x y*. *x* $\in$ *X* $\wedge$ *y* $\in$ *Y* $\longrightarrow$ B_K *x ?b y*
by (*unfold real-hyp2-B-def*) *simp*
thus $\exists$ *b*. $\forall$ *x y*. *x* $\in$ *X* $\wedge$ *y* $\in$ *Y* $\longrightarrow$ B_K *x b y* **by** (*rule exI*)
qed
qed
qed

interpretation *tarski-absolute-space real-hyp2-C real-hyp2-B*
using *hyp2-axiom6* **and** *hyp2-axiom7* **and** *hyp2-axiom11*
by *unfold-locales*

8.14 The Klein–Beltrami model satisfies the dimension-specific axioms

lemma *hyp2-rep-abs-examples*:
shows *hyp2-rep* (*hyp2-abs 0*) = *0* (**is** *hyp2-rep ?a* = *?ca*)
and *hyp2-rep* (*hyp2-abs* (*vector* [*1/2,0*])) = *vector* [*1/2,0*]
(**is** *hyp2-rep ?b* = *?cb*)
and *hyp2-rep* (*hyp2-abs* (*vector* [*0,1/2*])) = *vector* [*0,1/2*]
(**is** *hyp2-rep ?c* = *?cc*)
and *hyp2-rep* (*hyp2-abs* (*vector* [*1/4,1/4*])) = *vector* [*1/4,1/4*]
(**is** *hyp2-rep ?d* = *?cd*)
and *hyp2-rep* (*hyp2-abs* (*vector* [*1/2,1/2*])) = *vector* [*1/2,1/2*]
(**is** *hyp2-rep ?t* = *?ct*)
proof −
have *norm ?ca* < *1* **and** *norm ?cb* < *1* **and** *norm ?cc* < *1* **and** *norm ?cd* < *1*
and *norm ?ct* < *1*
by (*unfold norm-vec-def L2-set-def*) (*simp-all add*: *sum-2 power2-eq-square*)
thus *hyp2-rep ?a* = *?ca* **and** *hyp2-rep ?b* = *?cb* **and** *hyp2-rep ?c* = *?cc*
and *hyp2-rep ?d* = *?cd* **and** *hyp2-rep ?t* = *?ct*
by (*simp-all add*: *hyp2-rep-abs*)
qed

theorem *hyp2-axiom8*: $\exists$ *a b c*. $\neg$ B_K *a b c* $\wedge$ $\neg$ B_K *b c a* $\wedge$ $\neg$ B_K *c a b*
proof −
let *?ca* = *0* :: *real^2*
and *?cb* = *vector* [*1/2,0*] :: *real^2*
and *?cc* = *vector* [*0,1/2*] :: *real^2*
let *?a* = *hyp2-abs ?ca*
and *?b* = *hyp2-abs ?cb*
and *?c* = *hyp2-abs ?cc*
from *hyp2-rep-abs-examples* **and** *non-Col-example*

have ¬ (*hyp2.Col ?a ?b ?c*)
  by (*unfold hyp2.Col-def real-euclid.Col-def real-hyp2-B-def*) *simp*
thus $\exists$ *a b c*. ¬ B_K *a b c* ∧ ¬ B_K *b c a* ∧ ¬ B_K *c a b*
  unfolding *hyp2.Col-def*
  by *simp* (*rule exI*)+
qed

theorem *hyp2-axiom9*:
$\forall$ *p q a b c*. $p \neq q$ ∧ *a p* $\equiv_K$ *a q* ∧ *b p* $\equiv_K$ *b q* ∧ *c p* $\equiv_K$ *c q*
$\longrightarrow$ B_K *a b c* ∨ B_K *b c a* ∨ B_K *c a b*
proof (*rule allI*)+
  fix *p q a b c*
  show $p \neq q$ ∧ *a p* $\equiv_K$ *a q* ∧ *b p* $\equiv_K$ *b q* ∧ *c p* $\equiv_K$ *c q*
    $\longrightarrow$ B_K *a b c* ∨ B_K *b c a* ∨ B_K *c a b*
  proof
    assume $p \neq q$ ∧ *a p* $\equiv_K$ *a q* ∧ *b p* $\equiv_K$ *b q* ∧ *c p* $\equiv_K$ *c q*
    hence $p \neq q$ **and** *a p* $\equiv_K$ *a q* **and** *b p* $\equiv_K$ *b q* **and** *c p* $\equiv_K$ *c q* **by** *simp-all*

    let *?pp* = *Rep-hyp2 p*
      and *?pq* = *Rep-hyp2 q*
      and *?pa* = *Rep-hyp2 a*
      and *?pb* = *Rep-hyp2 b*
      and *?pc* = *Rep-hyp2 c*
    define *l* **where** *l* = *proj2-line-through ?pp ?pq*
    define *m ps pt stpq*
      where *m* = *drop-perp ?pa l*
        and *ps* = *endpoint-in-S ?pp ?pq*
        and *pt* = *endpoint-in-S ?pq ?pp*
        and *stpq* = *exp-2dist ?pp ?pq*

    from ‹$p \neq q$› **have** *?pp* ≠ *?pq* **by** (*simp add*: *Rep-hyp2-inject*)

    from *Rep-hyp2*
    have *stpq* > *0* **by** (*unfold stpq-def*) (*simp add*: *exp-2dist-positive*)
    hence *sqrt stpq* ∗ *sqrt stpq* = *stpq*
      by (*simp add*: *real-sqrt-mult* [*symmetric*])

    from *Rep-hyp2* **and** ‹*?pp* ≠ *?pq*›
    have *stpq* ≠ *1* **by** (*unfold stpq-def*) (*auto simp add*: *exp-2dist-1-equal*)

    have *z-non-zero ?pa* **and** *z-non-zero ?pb* **and** *z-non-zero ?pc*
      by (*simp-all add*: *Rep-hyp2 hyp2-S-z-non-zero*)

    have $\forall$ *pd* ∈ {*?pa*,*?pb*,*?pc*}.
      cross-ratio ps pt (*perp-foot pd l*) *?pp* = *1* / (*sqrt stpq*)
    proof
      fix *pd*
      assume *pd* ∈ {*?pa*,*?pb*,*?pc*}
      with *Rep-hyp2* **have** *pd* ∈ *hyp2* **by** *auto*

```
  define pe x
    where pe = perp-foot pd l
      and x = cosh-dist ?pp pd

  from ‹pd ∈ {?pa,?pb,?pc}› and ‹a p ≡K a q› and ‹b p ≡K b q›
    and ‹c p ≡K c q›
  have cosh-dist pd ?pp = cosh-dist pd ?pq
    by (auto simp add: real-hyp2-C-cosh-dist)
  with ‹pd ∈ hyp2› and Rep-hyp2
  have x = cosh-dist ?pq pd by (unfold x-def) (simp add: cosh-dist-swap)

  from Rep-hyp2 [of p] and ‹pd ∈ hyp2› and cosh-dist-positive [of ?pp pd]
  have x ≠ 0 by (unfold x-def) simp

  from Rep-hyp2 and ‹pd ∈ hyp2› and ‹?pp ≠ ?pq›
  have cross-ratio ps pt pe ?pp
    = (cosh-dist ?pq pd * sqrt stpq − cosh-dist ?pp pd)
    / (cosh-dist ?pp pd *  stpq − cosh-dist ?pq pd * sqrt stpq)
    unfolding ps-def and pt-def and pe-def and l-def and stpq-def
    by (simp add: perp-foot-cross-ratio-formula)
  also from x-def and ‹x = cosh-dist ?pq pd›
  have ... = (x * sqrt stpq − x) / (x * stpq − x * sqrt stpq) by simp
  also from ‹sqrt stpq * sqrt stpq = stpq›
  have ... = (x * sqrt stpq − x) / ((x * sqrt stpq − x) * sqrt stpq)
    by (simp add: algebra-simps)
  also from ‹x ≠ 0› and ‹stpq ≠ 1› have ... = 1 / sqrt stpq by simp
  finally show cross-ratio ps pt pe ?pp = 1 / sqrt stpq .
qed
hence cross-ratio ps pt (perp-foot ?pa l) ?pp = 1 / sqrt stpq by simp

have ∀ pd ∈ {?pa,?pb,?pc}. proj2-incident pd m
proof
  fix pd
  assume pd ∈ {?pa,?pb,?pc}
  with Rep-hyp2 have pd ∈ hyp2 by auto
  with Rep-hyp2 and ‹?pp ≠ ?pq› and proj2-line-through-incident
  have cross-ratio-correct ps pt ?pp (perp-foot pd l)
    and cross-ratio-correct ps pt ?pp (perp-foot ?pa l)
    unfolding ps-def and pt-def and l-def
    by (simp-all add: endpoints-in-S-perp-foot-cross-ratio-correct)

  from ‹pd ∈ {?pa,?pb,?pc}›
    and ‹∀ pd ∈ {?pa,?pb,?pc}.
    cross-ratio ps pt (perp-foot pd l) ?pp = 1 / (sqrt stpq)›
  have cross-ratio ps pt (perp-foot pd l) ?pp = 1 / sqrt stpq by auto
  with ‹cross-ratio ps pt (perp-foot ?pa l) ?pp = 1 / sqrt stpq›
  have cross-ratio ps pt (perp-foot pd l) ?pp
    = cross-ratio ps pt (perp-foot ?pa l) ?pp
```

    by *simp*
  hence *cross-ratio ps pt ?pp* (*perp-foot pd l*)
    = *cross-ratio ps pt ?pp* (*perp-foot ?pa l*)
    by (*simp add*: *cross-ratio-swap-34* [*of ps pt - ?pp*])
  with ‹*cross-ratio-correct ps pt ?pp* (*perp-foot pd l*)›
    and ‹*cross-ratio-correct ps pt ?pp* (*perp-foot ?pa l*)›
  have *perp-foot pd l* = *perp-foot ?pa l* **by** (*rule cross-ratio-unique*)
  with *Rep-hyp2* [*of p*] **and** ‹*pd* ∈ *hyp2*›
    and *proj2-line-through-incident* [*of ?pp ?pq*]
    and *perp-foot-eq-implies-drop-perp-eq* [*of ?pp pd l ?pa*]
  have *drop-perp pd l* = *m* **by** (*unfold m-def l-def*) *simp*
  with *drop-perp-incident* [*of pd l*] **show** *proj2-incident pd m* **by** *simp*
qed
hence *proj2-set-Col* {*?pa*,*?pb*,*?pc*}
  by (*unfold proj2-set-Col-def*) (*simp add*: *exI* [*of - m*])
hence *proj2-Col ?pa ?pb ?pc* **by** (*simp add*: *proj2-Col-iff-set-Col*)
with ‹*z-non-zero ?pa*› **and** ‹*z-non-zero ?pb*› **and** ‹*z-non-zero ?pc*›
have *real-euclid.Col* (*hyp2-rep a*) (*hyp2-rep b*) (*hyp2-rep c*)
  by (*unfold hyp2-rep-def*) (*simp add*: *proj2-Col-iff-euclid-cart2*)
thus B_K *a b c* ∨ B_K *b c a* ∨ B_K *c a b*
  by (*unfold real-hyp2-B-def real-euclid.Col-def*)
qed
qed

interpretation *hyp2*: *tarski-absolute real-hyp2-C real-hyp2-B*
  using *hyp2-axiom8* **and** *hyp2-axiom9*
  by *unfold-locales*

8.15 The Klein–Beltrami model violates the Euclidean axiom

theorem *hyp2-axiom10-false*:
  shows ¬ (∀ *a b c d t*. B_K *a d t* ∧ B_K *b d c* ∧ *a* ≠ *d*
  ⟶ (∃ *x y*. B_K *a b x* ∧ B_K *a c y* ∧ B_K *x t y*))
proof
  assume ∀ *a b c d t*. B_K *a d t* ∧ B_K *b d c* ∧ *a* ≠ *d*
    ⟶ (∃ *x y*. B_K *a b x* ∧ B_K *a c y* ∧ B_K *x t y*)

  let *?ca* = *0* :: *real^2*
    and *?cb* = *vector* [*1/2*,*0*] :: *real^2*
    and *?cc* = *vector* [*0*,*1/2*] :: *real^2*
    and *?cd* = *vector* [*1/4*,*1/4*] :: *real^2*
    and *?ct* = *vector* [*1/2*,*1/2*] :: *real^2*
  let *?a* = *hyp2-abs ?ca*
    and *?b* = *hyp2-abs ?cb*
    and *?c* = *hyp2-abs ?cc*
    and *?d* = *hyp2-abs ?cd*
    and *?t* = *hyp2-abs ?ct*

have *?cd* = (*1*/*2*) $*_R$ *?ct* **and** *?cd* − *?cb* = (*1*/*2*) $*_R$ (*?cc* − *?cb*)
  by (*unfold vector-def*) (*simp-all add*: *vec-eq-iff*)
hence $B_{\mathbb{R}}$ *?ca ?cd ?ct* **and** $B_{\mathbb{R}}$ *?cb ?cd ?cc*
  by (*unfold real-euclid-B-def*) (*simp-all add*: *exI* [*of* - *1*/*2*])
hence B_K *?a ?d ?t* **and** B_K *?b ?d ?c*
  by (*unfold real-hyp2-B-def*) (*simp-all add*: *hyp2-rep-abs-examples*)

have *?a* ≠ *?d*
proof
  assume *?a* = *?d*
  hence *hyp2-rep ?a* = *hyp2-rep ?d* **by** *simp*
  hence *?ca* = *?cd* **by** (*simp add*: *hyp2-rep-abs-examples*)
  thus *False* **by** (*simp add*: *vec-eq-iff forall-2*)
qed
with ‹B_K *?a ?d ?t*› **and** ‹B_K *?b ?d ?c*›
  and ‹∀ *a b c d t*. B_K *a d t* ∧ B_K *b d c* ∧ *a* ≠ *d*
  ⟶ (∃ *x y*. B_K *a b x* ∧ B_K *a c y* ∧ B_K *x t y*)›
obtain *x* **and** *y* **where** B_K *?a ?b x* **and** B_K *?a ?c y* **and** B_K *x ?t y*
  by *blast*

let *?cx* = *hyp2-rep x*
  and *?cy* = *hyp2-rep y*
from ‹B_K *?a ?b x*› **and** ‹B_K *?a ?c y*› **and** ‹B_K *x ?t y*›
have $B_{\mathbb{R}}$ *?ca ?cb ?cx* **and** $B_{\mathbb{R}}$ *?ca ?cc ?cy* **and** $B_{\mathbb{R}}$ *?cx ?ct ?cy*
  by (*unfold real-hyp2-B-def*) (*simp-all add*: *hyp2-rep-abs-examples*)

from ‹$B_{\mathbb{R}}$ *?ca ?cb ?cx*› **and** ‹$B_{\mathbb{R}}$ *?ca ?cc ?cy*› **and** ‹$B_{\mathbb{R}}$ *?cx ?ct ?cy*›
obtain *j* **and** *k* **and** *l* **where** *?cb* − *?ca* = *j* $*_R$ (*?cx* − *?ca*)
  and *?cc* − *?ca* = *k* $*_R$ (*?cy* − *?ca*)
  and *l* ≥ *0* **and** *l* ≤ *1* **and** *?ct* − *?cx* = *l* $*_R$ (*?cy* − *?cx*)
  by (*unfold real-euclid-B-def*) *fast*

from ‹*?cb* − *?ca* = *j* $*_R$ (*?cx* − *?ca*)› **and** ‹*?cc* − *?ca* = *k* $*_R$ (*?cy* − *?ca*)›
have *j* ≠ *0* **and** *k* ≠ *0* **by** (*auto simp add*: *vec-eq-iff forall-2*)
with ‹*?cb* − *?ca* = *j* $*_R$ (*?cx* − *?ca*)› **and** ‹*?cc* − *?ca* = *k* $*_R$ (*?cy* − *?ca*)›
have *?cx* = (*1*/*j*) $*_R$ *?cb* **and** *?cy* = (*1*/*k*) $*_R$ *?cc* **by** *simp-all*
hence *?cx$2* = *0* **and** *?cy$1* = *0* **by** *simp-all*

from ‹*?ct* − *?cx* = *l* $*_R$ (*?cy* − *?cx*)›
have *?ct* = (*1* − *l*) $*_R$ *?cx* + *l* $*_R$ *?cy* **by** (*simp add*: *algebra-simps*)
with ‹*?cx$2* = *0*› **and** ‹*?cy$1* = *0*›
have *?ct$1* = (*1* − *l*) ∗ (*?cx$1*) **and** *?ct$2* = *l* ∗ (*?cy$2*) **by** *simp-all*
hence *l* ∗ (*?cy$2*) = *1*/*2* **and** (*1* − *l*) ∗ (*?cx$1*) = *1*/*2* **by** *simp-all*

have *?cx$1* ≤ |*?cx$1*| **by** *simp*
also have ... ≤ *norm ?cx* **by** (*rule component-le-norm-cart*)
also have ... < *1* **by** (*rule norm-hyp2-rep-lt-1*)
finally have *?cx$1* < *1* **.**
with ‹*l* ≤ *1*› **and** *mult-less-cancel-left* [*of 1* − *l ?cx$1 1*]

have $(1 - l) * ?cx\$1 \leq 1 - l$ **by** *auto*
with ‹$(1 - l) * (?cx\$1) = 1/2$› **have** $l \leq 1/2$ **by** *simp*

have $?cy\$2 \leq |?cy\$2|$ **by** *simp*
also have $\ldots \leq$ *norm ?cy* **by** (*rule component-le-norm-cart*)
also have $\ldots < 1$ **by** (*rule norm-hyp2-rep-lt-1*)
finally have $?cy\$2 < 1$ **.**
with ‹$l \geq 0$› **and** *mult-less-cancel-left* [*of l ?cy$2 1*]
have $l * ?cy\$2 \leq l$ **by** *auto*
with ‹$l * (?cy\$2) = 1/2$› **have** $l \geq 1/2$ **by** *simp*
with ‹$l \leq 1/2$› **have** $l = 1/2$ **by** *simp*
with ‹$l * (?cy\$2) = 1/2$› **have** $?cy\$2 = 1$ **by** *simp*
with ‹$?cy\$2 < 1$› **show** *False* **by** *simp*
qed

theorem *hyp2-not-tarski*: ¬ (*tarski real-hyp2-C real-hyp2-B*)
using *hyp2-axiom10-false*
by (*unfold tarski-def tarski-space-def tarski-space-axioms-def*) *simp*

Therefore axiom 10 is independent.

end

References

[1] K. Borsuk and W. Szmielew. *Foundations of Geometry: Euclidean and Bolyai-Lobachevskian Geometry; Projective Geometry*. North-Holland Publishing Company, 1960. Translated from Polish by Erwin Marquit.

[2] T. J. M. Makarios. A mechanical verification of the independence of Tarski's Euclidean axiom. Master's thesis, Victoria University of Wellington, New Zealand, 2012. http://researcharchive.vuw.ac.nz/handle/10063/2315.

[3] W. Schwabhäuser, W. Szmielew, and A. Tarski. *Metamathematische Methoden in der Geometrie*. Springer-Verlag, 1983.