The independence of the parallels postulate, verified in Isabelle 2009–2

T. J. M. Makarios

February 29, 2012

Contents

1 Metric spaces

theory *Metric*
imports *Euclidean-Space*
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* :: $'p \Rightarrow 'p \Rightarrow real$
  assumes [*simp*]: $dist\ x\ y = 0 \longleftrightarrow x = y$
  and *triangle*: $dist\ x\ z \leq dist\ y\ x + dist\ y\ z$

```
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 add: diff-minus)
  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 Complex-Main
  Metric
  Vec1
begin
```

lemma *unordered-pair-element-equality*:
  assumes $\{p, q\} = \{r, s\}$ **and** $p = r$
  shows $q = s$
proof *cases*
  assume $p = q$
  with $\langle\{p, q\} = \{r, s\}\rangle$ **have** $\{r, s\} = \{q\}$ **by** *simp*
  thus $q = s$ **by** *simp*
next
  assume $p \neq q$
  with $\langle\{p, q\} = \{r, s\}\rangle$ **have** $\{r, s\} - \{p\} = \{q\}$ **by** *auto*
  moreover
    from $\langle p = r\rangle$ **have** $\{r, s\} - \{p\} \subseteq \{s\}$ **by** *auto*
  ultimately have $\{q\} \subseteq \{s\}$ **by** *simp*
  thus $q = s$ **by** *simp*
qed

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

lemma *square-expand*: $(x::real)^2 = x * x$
proof $-$
  have $2 = Suc\ 1$ **by** *simp*
  with *power-Suc* [*of x 1*] **and** *power-one-right* [*of x*] **show** *?thesis* **by** *arith*
qed

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

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

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

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

lemma *real2-eq*:

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

definition *rotate2* :: *real*^2 ⇒ *real*^2 **where**
rotate2 x ≜ *vector* [−*x*\$2, *x*\$*1*]

declare *vector-2* [*simp*]

lemma *rotate2* [*simp*]:
(*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-vector-def*
by (*simp add*: *setsum-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 $u \cdot x = 0$ **and** $x\$1 \neq 0$
 shows $u = (u\$2\ /\ x\$1) *_R (rotate2\ x)$ (**is** $u = ?u'$)
proof −
 from ⟨$u \cdot x = 0$⟩ **have** $u\$1 * x\$1 = -(u\$2) * (x\$2)$
  unfolding *inner-vector-def*
  by (*simp add*: *setsum-2*)
 hence $u\$1 * x\$1\ /\ x\$1 = -u\$2\ /\ x\$1 * x\$2$ **by** *simp*
 with ⟨$x\$1 \neq 0$⟩ **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 $u\ v\ x :: real\hat{}('n::finite)$
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-vector-def*
by *simp*
also from *setsum-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-vector-def*
by *simp*
finally show *?thesis* **..**
qed

lemma *dot-right-diff-distrib*:
fixes $u\ v\ x :: real\hat{}('n::finite)$
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 *right-distrib* $[of\ a + b\ a\ b]$ **and** *left-distrib* $[of\ a\ b]$ **have**
$4 * a * b \leq (a + b) * (a + b)$
and $4 * a * b = (a + b) * (a + b) \longleftrightarrow a = b$ **by** *simp+*
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*: *mult-ac*)+
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* $(A \cap B)$ $(r \cap B \times B)$
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* $(A \times A)$
unfolding *sym-def*
by *simp*

lemma *sym-restrict*:
assumes *sym* r
shows *sym* $(r \cap A \times A)$
proof −
from ‹*sym* r› **and** *sym-allrel* **and** *sym-Int*
show *?thesis* **by** *auto*
qed

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

lemma *trans-restrict*:
assumes *trans* r
shows *trans* $(r \cap A \times A)$
proof −
from ‹*trans* r› **and** *trans-allrel* **and** *trans-Int*
show *?thesis* **by** *auto*
qed

lemma *equiv-Int*:
assumes *equiv* A r **and** *equiv* B s
shows *equiv* $(A \cap B)$ $(r \cap s)$
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* $(A \times A)$
unfolding *equiv-def*
by (*simp add*: *refl-on-allrel sym-allrel trans-allrel*)

lemma *equiv-restrict*:
assumes *equiv A r*
shows *equiv* $(A \cap B)$ $(r \cap B \times B)$
proof −
from ⟨*equiv A r*⟩ **and** *equiv-allrel* [*of B*] **and** *equiv-Int*
show *?thesis* **by** *auto*
qed

lemma *scalar-vector-matrix-assoc*:
fixes *k* :: *real* **and** *x* :: *real*^(′*n*::*finite*) **and** *A* :: *real*^(′*m*::*finite*)^′*n*
shows $(k *_R x)$ *v*∗ $A = k *_R$ (*x v*∗ *A*)
proof −
{ **fix** *i*
from *setsum-right-distrib* [*of k* λ*j*. *x*\$*j* ∗ *A*\$*j*\$*i UNIV*]
have $(\sum j{\in}UNIV.\ k * (x\$j * A\$j\$i)) = k * (\sum j{\in}UNIV.\ x\$j * A\$j\$i)$ **..** }
thus $(k *_R x)$ *v*∗ $A = k *_R$ (*x v*∗ *A*)
unfolding *vector-matrix-mult-def*
by (*simp add*: *Cart-eq algebra-simps*)
qed

lemma *vector-scalar-matrix-ac*:
fixes *k* :: *real* **and** *x* :: *real*^(′*n*::*finite*) **and** *A* :: *real*^(′*m*::*finite*)^′*n*
shows *x v*∗ $(k *_R A) = k *_R$ (*x v*∗ *A*)
proof −
have *x v*∗ $(k *_R A) = (k *_R x)$ *v*∗ *A*
unfolding *vector-matrix-mult-def*
by (*simp add*: *algebra-simps*)
with *scalar-vector-matrix-assoc*
show *x v*∗ $(k *_R A) = k *_R$ (*x v*∗ *A*)
by *auto*
qed

lemma *vector-matrix-left-distrib*:
fixes *x y* :: *real*^(′*n*::*finite*) **and** *A* :: *real*^(′*m*::*finite*)^′*n*
shows $(x + y)$ *v*∗ *A* = *x v*∗ *A* + *y v*∗ *A*
unfolding *vector-matrix-mult-def*
by (*simp add*: *algebra-simps setsum-addf Cart-eq*)

lemma *times-zero-vector* [*simp*]: *A* ∗*v 0* = *0*
unfolding *matrix-vector-mult-def*

by (*simp add*: *Cart-eq*)

lemma *invertible-times-eq-zero*:
fixes x :: *real*^('*n*::*finite*) **and** A :: *real*^'*n*^'*n*
assumes *invertible* A **and** $A *v\ x = 0$
shows $x = 0$
proof −
from ⟨*invertible* A⟩
and *someI-ex* [*of* $\lambda A'.\ A ** A' = mat\ 1 \wedge A' ** A = mat\ 1$]
have *matrix-inv* $A ** A = mat\ 1$
unfolding *invertible-def matrix-inv-def*
by *simp*
hence $x = (matrix\text{-}inv\ A ** A) *v\ x$ **by** (*simp add*: *matrix-vector-mul-lid*)
also have $\ldots = matrix\text{-}inv\ A *v\ (A *v\ x)$
by (*simp add*: *matrix-vector-mul-assoc*)
also from ⟨$A *v\ x = 0$⟩ **have** $\ldots = 0$ **by** *simp*
finally show $x = 0$ **.**
qed

lemma *vector-transpose-matrix* [*simp*]: $x\ v*\ transpose\ A = A *v\ x$
unfolding *transpose-def vector-matrix-mult-def matrix-vector-mult-def*
by *simp*

lemma *transpose-matrix-vector* [*simp*]: $transpose\ A *v\ x = x\ v*\ A$
unfolding *transpose-def vector-matrix-mult-def matrix-vector-mult-def*
by *simp*

lemma *transpose-invertible*:
fixes A :: *real*^('*n*::*finite*)^'*n*
assumes *invertible* A
shows *invertible* (*transpose* A)
proof −
from ⟨*invertible* A⟩ **obtain** A' **where** $A ** A' = mat\ 1$ **and** $A' ** A = mat\ 1$
unfolding *invertible-def*
by *auto*
with *matrix-transpose-mul* [*of* $A\ A'$] **and** *matrix-transpose-mul* [*of* $A'\ A$]
have $transpose\ A' ** transpose\ A = mat\ 1$ **and** $transpose\ A ** transpose\ A' = mat\ 1$
by (*simp add*: *transpose-mat*)+
thus *invertible* (*transpose* A)
unfolding *invertible-def*
by *auto*
qed

lemma *times-invertible-eq-zero*:
fixes x :: *real*^('*n*::*finite*) **and** A :: *real*^'*n*^'*n*
assumes *invertible* A **and** $x\ v*\ A = 0$
shows $x = 0$
proof −
from *transpose-invertible* **and** ⟨*invertible* A⟩ **have** *invertible* (*transpose* A) **by** *auto*

with *invertible-times-eq-zero* [*of transpose A x*] **and** ⟨*x v∗ A = 0*⟩
show $x = 0$ **by** *simp*
qed

lemma *matrix-id-invertible*:
invertible (*mat 1* :: ('*a*::*semiring-1*)^('*n*::*finite*)^'*n*)
proof −
from *matrix-mul-lid* [*of mat 1* :: '*a*^'*n*^'*n*]
show *invertible* (*mat 1* :: '*a*^'*n*^'*n*)
unfolding *invertible-def*
by *auto*
qed

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 $\exists\ x.\ x \in X$
proof −
from ⟨$X \in A//r$⟩ **obtain** x **where** $x \in A$ **and** $X = r``\{x\}$
unfolding *quotient-def*
by *auto*
with *equiv-class-self* [*of A r x*] **and** ⟨*equiv A r*⟩ **show** $\exists\ x.\ x \in X$ **by** *auto*
qed

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

lemma *card-suc-ge-insert*:
fixes A **and** x
shows $card\ A + 1 \geq card\ (insert\ x\ A)$
proof *cases*
assume *finite A*
with *card-insert-if* [*of A x*] **show** $card\ A + 1 \geq card\ (insert\ x\ A)$ **by** *simp*
next
assume *infinite A*
thus $card\ A + 1 \geq card\ (insert\ x\ A)$ **by** *simp*
qed

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

by (*simp add*: *card-mono*)

lemma *setsum-forall-cong*:
assumes $\forall\ x{\in}A.\ f\ x = g\ x$
shows $(\sum\ x{\in}A.\ f\ x) = (\sum\ x{\in}A.\ g\ x)$
proof −
from $\langle\forall\ x{\in}A.\ f\ x = g\ x\rangle$ **have** $\bigwedge\ x.\ x \in A \Longrightarrow f\ x = g\ x$ **..**
with *setsum-cong* **show** $(\sum\ x{\in}A.\ f\ x) = (\sum\ x{\in}A.\ g\ x)$ **by** *simp*
qed

lemma *partition-Image-element*:
assumes *equiv A r* **and** $X \in A//r$ **and** $x \in X$
shows $r``\{x\} = X$
proof −
from *Union-quotient* **and** *assms* **have** $x \in A$ **by** *auto*
with *quotientI* [*of x A r*] **have** $r``\{x\} \in A//r$ **by** *simp*

from *equiv-class-self* **and** ⟨*equiv A r*⟩ **and** $\langle x \in A\rangle$ **have** $x \in r``\{x\}$ **by** *simp*

from ⟨*equiv A r*⟩ **and** $\langle x \in A\rangle$ **have** $(x, x) \in r$
unfolding *equiv-def* **and** *refl-on-def*
by *simp*

with *quotient-eqI* [*of A r X* $r``\{x\}$ *x x*]
and *assms* **and** $\langle Image\ r\ \{x\} \in A//r\rangle$ **and** $\langle x \in Image\ r\ \{x\}\rangle$
show $r``\{x\} = X$ **by** *simp*
qed

lemma *card-insert-ge*: $card\ (insert\ x\ A) \geq card\ A$
proof *cases*
assume *finite A*
with *card-insert-le* [*of A x*] **show** $card\ (insert\ x\ A) \geq card\ A$ **by** *simp*
next
assume *infinite A*
hence $card\ A = 0$ **by** *simp*
thus $card\ (insert\ x\ A) \geq card\ A$ **by** *simp*
qed

lemma *choose-1*:
assumes $card\ S = 1$
shows $\exists\ x.\ S = \{x\}$
using $\langle card\ S = 1\rangle$ **and** *card-eq-SucD* [*of S 0*]
by *simp*

lemma *choose-2*:
assumes $card\ S = 2$
shows $\exists\ x\ y.\ S = \{x,y\}$
proof −
from $\langle card\ S = 2\rangle$ **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
proof cases
  assume x = 0
  with ‹sgn x = sgn y› have y = 0 by (simp add: sgn-0-0)
  with ‹x = 0› show sgn (x + y) = sgn x by (simp add: sgn-0-0)
next
  assume x ≠ 0
  show sgn (x + y) = sgn x
  proof cases
    assume x > 0
    with ‹sgn x = sgn y› and sgn-1-pos [where ?'a = 'a] have y > 0 by simp
    with ‹x > 0› and sgn-1-pos [where ?'a = 'a]
    show sgn (x + y) = sgn x by simp
  next
    assume ¬ x > 0
    with ‹x ≠ 0› have x < 0 by simp
    with ‹sgn x = sgn y› and sgn-1-neg [where ?'a = 'a] have y < 0 by auto
    with ‹x < 0› and sgn-1-neg [where ?'a = 'a]
    show sgn (x + y) = sgn x by simp
  qed
qed

lemma sgn-div:
  fixes x y :: 'a::linordered-field-inverse-zero
  assumes y ≠ 0 and sgn x = sgn y
  shows x / y > 0
proof cases
  assume y > 0
  with ‹sgn x = sgn y› and sgn-1-pos [where ?'a = 'a] have x > 0 by simp
  with ‹y > 0› show x / y > 0 by (simp add: zero-less-divide-iff)
next
  assume ¬ y > 0
  with ‹y ≠ 0› have y < 0 by simp
  with ‹sgn x = sgn y› and sgn-1-neg [where ?'a = 'a] have x < 0 by simp
  with ‹y < 0› show x / y > 0 by (simp add: zero-less-divide-iff)
qed

lemma abs-plus:
  fixes x y :: 'a::linordered-idom
  assumes sgn x = sgn y
  shows |x + y| = |x| + |y|
```

proof −
  from ⟨$sgn\ x = sgn\ y$⟩ **have** $sgn\ (x + y) = sgn\ x$ **by** (*rule sgn-plus*)
  hence $|x + y| = (x + y) * sgn\ x$ **by** (*simp add*: *abs-sgn*)
  also from ⟨$sgn\ x = sgn\ y$⟩
  have $\ldots = x * sgn\ x + y * sgn\ y$ **by** (*simp add*: *algebra-simps*)
  finally show $|x + y| = |x| + |y|$ **by** (*simp add*: *abs-sgn*)
qed

lemma *sgn-plus-abs*:
  fixes $x\ y$ :: $'a$::*linordered-idom*
  assumes $|x| > |y|$
  shows $sgn\ (x + y) = sgn\ x$
proof *cases*
  assume $x > 0$
  with ⟨$|x| > |y|$⟩ **have** $x + y > 0$ **by** *simp*
  with ⟨$x > 0$⟩ **show** $sgn\ (x + y) = sgn\ x$ **by** *simp*
next
  assume $\neg\ x > 0$

  from ⟨$|x| > |y|$⟩ **have** $x \neq 0$ **by** *simp*
  with ⟨$\neg\ x > 0$⟩ **have** $x < 0$ **by** *simp*
  with ⟨$|x| > |y|$⟩ **have** $x + y < 0$ **by** *simp*
  with ⟨$x < 0$⟩ **show** $sgn\ (x + y) = sgn\ x$ **by** *simp*
qed

lemma *sqrt-4* [*simp*]: $sqrt\ 4 = 2$
proof −
  have $sqrt\ 4 = sqrt\ (2 * 2)$ **by** *simp*
  thus $sqrt\ 4 = 2$ **by** (*unfold real-sqrt-abs2*) *simp*
qed

end

3 Tarski's geometry

theory *Tarski*
imports *Complex-Main Miscellany Euclidean-Space Metric*
begin

3.1 The axioms

locale *tarski-first3* =
  fixes C :: $'p \Rightarrow 'p \Rightarrow 'p \Rightarrow 'p \Rightarrow bool$ (- - ≡ - - [*99,99,99,99*] *50*)
  assumes *A1*: $\forall\, a\ b.\ a\ b \equiv b\ a$
  and *A2*: $\forall\, a\ b\ p\ q\ r\ s.\ a\ b \equiv p\ q \land a\ b \equiv r\ s \longrightarrow p\ q \equiv r\ s$
  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** $\langle c\ d \equiv a\ b\rangle$ **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* :: $'p\ set \Rightarrow bool$ **where**
$\textit{is-segment}\ X \triangleq \exists\, x\ y.\ X = \{x, y\}$

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

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

```
definition SC-rel :: ('p set × 'p set) set where
SC-rel = {(X, Y) | X Y. SC X Y}

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

lemma C-SC-equiv: a b ≡ c d = SC {a, b} {c, d}
proof
  assume a b ≡ 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 ≡ y z
      by blast
  from left-segment-congruence [of a b w x y z] and
      ‹{a, b} = {w, x}› and
      ‹w x ≡ y z›
    have a b ≡ 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
end

sublocale tarski-first3 ⊆ equiv segments SC-rel
 by (simp add: equiv-def SC-rel-refl SC-rel-sym SC-rel-trans)
```

3.4 Some consequences of the first five axioms

```
context tarski-first5
```

```
begin
 lemma A4′: ∃x. B q a x ∧ a x ≡ b c
  by (simp add: A4 [simplified])

 theorem th2-8: a a ≡ b b
 proof −
  from A4′ [of - a b b] obtain x where a x ≡ b b by auto
  with A3′ [of a x b] have x = a by simp
  with ‹a x ≡ b b› show ?thesis by simp
 qed

 definition OFS :: ['p,'p,'p,'p,'p,'p,'p,'p] ⇒ bool where
  OFS a b c d a′ b′ c′ d′ ≜
    B a b c ∧ B a′ b′ c′ ∧ a b ≡ a′ b′ ∧ b c ≡ b′ c′ ∧ a d ≡ a′ d′ ∧ b d ≡ b′ d′

 lemma A5′: ⟦OFS a b c d a′ b′ c′ d′; a ≠ b⟧ ⟹ c d ≡ c′ d′
 proof −
  assume OFS a b c d a′ b′ c′ d′ and a ≠ 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 ≡ a′ b′
    b c ≡ b′ c′
  shows a c ≡ a′ c′
 proof cases
  assume a = b
  with ‹a b ≡ a′ b′› have a′ = b′ by (simp add: A3-reversed)
  with ‹b c ≡ b′ c′› and ‹a = b› show ?thesis by simp
 next
  assume a ≠ 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 ≠ a and B q a x and a x ≡ b c
    and B q a x' and a x' ≡ b c
    shows x = x'
  proof -
    from SC-sym' and SC-trans and C-SC-equiv and ⟨a x' ≡ b c⟩ and ⟨a x ≡ b c⟩
      have a x ≡ 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 ≡ 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 ≡ 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 ≠ a⟩ have x x ≡ x x' by simp
    thus x = x' by (rule A3-reversed)
  qed

  theorem th2-12:
    assumes q ≠ a
    shows ∃!x. B q a x ∧ a x ≡ b c
    using ⟨q ≠ 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 ≡ b b by auto
  from A3 [rule-format, of b x b] and ⟨b x ≡ 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 ∃x. B p x b ∧ 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 ∀ x y. x ∈ X ∧ y ∈ Y ⟶ B a x y
  shows ∃ b. ∀ x y. x ∈ X ∧ y ∈ Y ⟶ B x b y
proof –
  from assms have ∃ a. ∀ x y. x ∈ X ∧ y ∈ Y ⟶ B a x y by (rule exI)
  thus ∃ b. ∀ x y. x ∈ X ∧ y ∈ Y ⟶ 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 ‹B a c d› and ‹B a b c› 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 ⟨$B\ a\ b\ c$⟩ **and** ⟨$B\ a\ c\ x$⟩ **have** $B\ b\ c\ x$ **by** (*rule th3-6-1*)
have $c\ d \equiv c\ d$ **by** (*rule th2-1*)
with ⟨$b \neq c$⟩ **and** ⟨$B\ b\ c\ x$⟩ **and** ⟨$c\ x \equiv c\ d$⟩ **and** ⟨$B\ b\ c\ d$⟩
have $x = d$ **by** (*rule A4-unique*)
with ⟨$B\ a\ c\ x$⟩ **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 ⟨$B\ b\ c\ d$⟩ **and** ⟨$B\ a\ b\ c$⟩ **and** *th3-2* **have** $B\ d\ c\ b$ **and** $B\ c\ b\ a$ **by** *fast+*
with ⟨$b \neq c$⟩ **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**
$Col\ a\ b\ c \triangleq B\ a\ b\ c \lor B\ b\ c\ a \lor B\ c\ a\ b$

end

4 Real Euclidean space and Tarski's axioms

theory *Euclid-Tarski*
imports *Tarski SupInf*
begin

4.1 Real Euclidean space satisfies the first five axioms

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

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

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 $\exists x.\ B_{\mathbb{R}}\ q\ a\ x \wedge a\ x \equiv_{\mathbb{R}} b\ c$
 proof *cases*
  assume $q = a$
  let $?x = a + c - b$
  have $B_{\mathbb{R}}\ q\ a\ ?x$
  proof $-$
   let $?l = 0$ :: *real*
   note *real-euclid-B-def* [*of q a ?x*]
   moreover
    have $?l \geq 0$ **and** $?l \leq 1$ **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 $a\ ?x \equiv_{\mathbb{R}} b\ c$ **by** *simp*
  ultimately show *?thesis* **by** *blast*
 next
  assume $q \neq a$
  hence *norm-dist* $q\ a > 0$ **by** *simp*
  let $?k =$ *norm-dist* $b\ c$ / *norm-dist* $q\ a$
  from ⟨*norm-dist* $q\ a > 0$⟩
   and *divide-nonneg-pos* [*of norm-dist b c norm-dist q a*]
  have $?k \geq 0$ **by** *simp*
  let $?x = a + ?k *_R (a - q)$
  have $B_{\mathbb{R}}\ q\ a\ ?x$
  proof $-$
   let $?l = 1\ /\ (1 + ?k)$
   from ⟨$?k \geq 0$⟩ **have** $?l > 0$ **by** *simp*
   note *real-euclid-B-def* [*of q a ?x*]
   moreover
    from ⟨$?k \geq 0$⟩ **have** $?l \geq 0$ **and** $?l \leq 1$ **by** *auto*
   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 $B_{\mathbb{R}}\ q\ a\ ?x$ **by** *blast*
  qed
  moreover
   have $a\ ?x \equiv_{\mathbb{R}} b\ c$

```
    proof −
      from norm-scaleR [of ?k a − q] have
        norm-dist a ?x = |?k| ∗ norm (a − q) by simp
      also from ⟨?k ≥ 0⟩ have
        . . . = ?k ∗ norm (a − q) by arith
      also from norm-metric.symm [of q a] have
        . . . = ?k ∗ norm-dist q a by simp
      finally have
        norm-dist a ?x = norm-dist b c / norm-dist q a ∗ norm-dist q a .
      with ⟨norm-dist q a > 0⟩ show a ?x ≡ℝ b c by auto
    qed
  ultimately show ?thesis by blast
 qed }
thus ∀ q a b c. ∃ x. Bℝ q a x ∧ a x ≡ℝ b c by auto
{ fix a b c d a′ b′ c′ d′
 assume a ≠ b and
   Bℝ a b c and
   Bℝ a′ b′ c′ and
   a b ≡ℝ a′ b′ and
   b c ≡ℝ b′ c′ and
   a d ≡ℝ a′ d′ and
   b d ≡ℝ b′ d′
 have c d ≡ℝ c′ d′
 proof −
   { fix m
     fix p q r :: real^('n::finite)
     assume 0 ≤ m and
       m ≤ 1 and
       p ≠ q and
       q − p = m ∗R (r − p)
     from ⟨p ≠ q⟩ and ⟨q − p = m ∗R (r − p)⟩ have m ≠ 0
     proof −
       { assume m = 0
         with ⟨q − p = m ∗R (r − p)⟩ have q − p = 0 by simp
         with ⟨p ≠ q⟩ have False by simp }
       thus ?thesis ..
     qed
     with ⟨m ≥ 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 $|1 - m| * \mathit{norm}\ (p - q) = |m| * \mathit{norm}\ (q - r)$ **by** *simp*
with ‹$m > 0$› **and** ‹$m \leq 1$›
have $\mathit{norm}\ (q - r) = (1 - m)\ /\ m * \mathit{norm}\ (p - q)$ **by** *simp*
moreover from ‹$p \neq q$› **have** $\mathit{norm}\ (p - q) \neq 0$ **by** *simp*
ultimately
have $\mathit{norm}\ (q - r)\ /\ \mathit{norm}\ (p - q) = (1 - m)\ /\ m$ **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 $(\textit{norm-dist}\ r\ s)^2 = (\textit{norm-dist}\ r\ p)^2 + (\textit{norm-dist}\ p\ s)^2 +$
$2 * (((1/m) *_R (q - p)) \cdot (p - s))$
by *simp*
also from *inner.scaleR-left* [*of* $1/m$ $q - p$ $p - s$]
have $\ldots =$
$(\textit{norm-dist}\ r\ p)^2 + (\textit{norm-dist}\ p\ s)^2 + 2/m * ((q - p) \cdot (p - s))$

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*: *square-expand*)
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 $w\ y\ z :: real\hat{}('n)$ **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** $\langle B_{\mathbb{R}}\ a\ p\ c\rangle$
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 $\langle p - a = i *_R (c - a)\rangle$ **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** ‹$1 - i \geq 0$› **and** *mult-nonneg-nonneg*
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 $\langle ?l *_R a + (1 - ?l) *_R (j *_R c + (1 - j) *_R b) =$
$?k *_R b + (1 - ?k) *_R (i *_R c + (1 - i) *_R a)\rangle$ **and**
$\langle p - a = i *_R (c - a)\rangle$ **and**
$\langle q - b = j *_R (c - b)\rangle$ **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 $\langle ?x = ?k *_R b + (1 - ?k) *_R p\rangle$ **and**
rearrange-real-euclid-B [*of* $?x\ p\ ?k\ b$]
have $?x - p = ?k *_R (b - p)$ **by** *simp*
from $\langle i + j * (1 - i) > 0\rangle$ **and**
$\langle j * (1 - i) \geq 0\rangle$ **and**
zero-le-divide-iff [*of* $j * (1 - i)\ i + j * (1 - i)$]
have $?l \geq 0$ **by** *simp*
from $\langle i + j * (1 - i) > 0\rangle$ **and**
$\langle i \geq 0\rangle$ **and**
zero-le-divide-iff [*of* $i\ i + j * (1 - i)$] **and**
$\langle 1 - ?l = i\ /\ (i + j * (1 - i))\rangle$
have $1 - ?l \geq 0$ **by** *simp*
hence $?l \leq 1$ **by** *simp*
with $\langle ?l \geq 0\rangle$ **and**
$\langle ?x - q = ?l *_R (a - q)\rangle$ **and**
real-euclid-B-def [*of* $q\ ?x\ a$]
have $B_{\mathbb{R}}\ q\ ?x\ a$ **by** *auto*
from $\langle j \leq 1\rangle$ **have** $1 - j \geq 0$ **by** *simp*
with $\langle 1 - ?l \geq 0\rangle$ **and**
$\langle (1 - ?l) * (1 - j) = ?k\rangle$ **and**
zero-le-mult-iff [*of* $1 - ?l\ 1 - j$]
have $?k \geq 0$ **by** *simp*
from $\langle j \geq 0\rangle$ **have** $1 - j \leq 1$ **by** *simp*
from $\langle ?l \geq 0\rangle$ **have** $1 - ?l \leq 1$ **by** *simp*
with $\langle 1 - j \leq 1\rangle$ **and**
$\langle 1 - j \geq 0\rangle$ **and**
mult-mono [*of* $1 - ?l\ 1\ 1 - j\ 1$] **and**
$\langle (1 - ?l) * (1 - j) = ?k\rangle$
have $?k \leq 1$ **by** *simp*
with $\langle ?k \geq 0\rangle$ **and**
$\langle ?x - p = ?k *_R (b - p)\rangle$ **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 ‹x − a = l *R (d − a)› and
    ‹d − a = jd *R (c − a)› and
    scaleR-scaleR
   have x − a = (l * jd) *R (c − a) by simp
  hence ∃ i. x − a = i *R (c − a) by auto }
note xlemma = this
let ?S = {j. j ≥ 1 ∧ (∃ y∈Y. y − a = j *R (c − a))}
from ‹d ∈ Y› and ‹jd ≥ 1› and ‹d − a = jd *R (c − a)›
  have ?S ≠ {} 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 ∈ X and y ∈ Y
  from xlemma [of x] and ‹x ∈ X›
    obtain i where x − a = i *R (c − a) by auto
  from ylemma [of y] and ‹y ∈ Y›
    obtain j where j ≥ 1 and y − a = j *R (c − a) by auto
  with ‹y ∈ Y› have j ∈ ?S by auto
  with Inf-lower have ?k ≤ j by auto
  { fix h
    assume h ∈ ?S
    hence h ≥ 1 by simp
    from ‹h ∈ ?S›
      obtain z where z ∈ Y and z − a = h *R (c − a) by auto
    from ‹∀ x y. x ∈ X ∧ y ∈ Y ⟶ Bℝ a x y› and ‹x ∈ X› and ‹z ∈ Y›
      have Bℝ a x z by simp
    with real-euclid-B-def [of a x z]
      obtain l where l ≤ 1 and x − a = l *R (z − a) by auto
    with ‹z − a = h *R (c − a)› and scaleR-scaleR
      have x − a = (l * h) *R (c − a) by simp
    with ‹x − a = i *R (c − a)›
      have i *R (c − a) = (l * h) *R (c − a) by auto
    with scaleR-cancel-right and ‹c − a ≠ 0› have i = l * h by blast
    with ‹l ≤ 1› and ‹h ≥ 1› have i ≤ h by simp }
  with ‹?S ≠ {}› and Inf-greatest [of ?S] have i ≤ ?k by simp
  have y − x = (y − a) − (x − a) by simp
  with ‹y − a = j *R (c − a)› and ‹x − a = i *R (c − a)›
    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 ‹?b − a = ?k *R (c − a)› and ‹x − a = i *R (c − a)›
    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 Bℝ 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 $\langle ?b - x = (?k - i) *_R (c - a)\rangle$ **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 $i \neq j$
with $\langle i \leq ?k\rangle$ **and** $\langle ?k \leq j\rangle$ **have** $j - i > 0$ **by** *simp*
with $\langle y - x = (j - i) *_R (c - a)\rangle$ **and** *scaleR-scaleR*
have $c - a = (1\ /\ (j - i)) *_R (y - x)$ **by** *simp*
with $\langle ?b - x = (?k - i) *_R (c - a)\rangle$ **and** *scaleR-scaleR*
have $?b - x = ((?k - i)\ /\ (j - i)) *_R (y - x)$ **by** *simp*
let $?l = (?k - i)\ /\ (j - i)$
from $\langle ?k \leq j\rangle$ **have** $?k - i \leq j - i$ **by** *simp*
with $\langle j - i > 0\rangle$ **have** $?l \leq 1$ **by** *simp*
from $\langle i \leq ?k\rangle$ **and** $\langle j - i > 0\rangle$ **and** *pos-le-divide-eq* [*of* $j - i\ 0\ ?k - i$]
have $?l \geq 0$ **by** *simp*
with *real-euclid-B-def* [*of* x $?b$ y] **and**
$\langle ?l \leq 1\rangle$ **and**
$\langle ?b - x = ?l *_R (y - x)\rangle$
show $B_{\mathbb{R}}\ x\ ?b\ y$ **by** *auto*
qed }
thus $\exists b.\ \forall x\ y.\ x \in X \wedge y \in Y \longrightarrow B_{\mathbb{R}}\ x\ b\ y$ **by** *auto*
qed }
thus $\forall X\ Y.\ (\exists a.\ \forall x\ y.\ x \in X \wedge y \in Y \longrightarrow B_{\mathbb{R}}\ a\ x\ y) \longrightarrow$
$(\exists b.\ \forall x\ y.\ x \in X \wedge y \in Y \longrightarrow B_{\mathbb{R}}\ x\ b\ y)$
by *auto*
qed

4.3 Real Euclidean space satisfies the Euclidean axiom

lemma *rearrange-real-euclid-B-2*:
fixes $a\ b\ c$ :: *real*^('*n*::*finite*)
assumes $l \neq 0$
shows $b - a = l *_R (c - a) \longleftrightarrow c = (1/l) *_R b + (1 - 1/l) *_R a$
proof
from *scaleR-right-diff-distrib* [*of* $1/l\ b\ a$]
have $(1/l) *_R (b - a) = c - a \longleftrightarrow (1/l) *_R b - (1/l) *_R a + a = c$ **by** *auto*
also with *scaleR-left-diff-distrib* [*of* $1\ 1/l\ a$]
have $\ldots \longleftrightarrow c = (1/l) *_R b + (1 - 1/l) *_R a$ **by** *auto*
finally have *eq*:
$(1/l) *_R (b - a) = c - a \longleftrightarrow c = (1/l) *_R b + (1 - 1/l) *_R a$ **.**
{ **assume** $b - a = l *_R (c - a)$
with $\langle l \neq 0\rangle$ **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 $\langle l \neq 0\rangle$ **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 $c - a = 1 *_R (c - a)$ **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**
    $c - a = (l - 1) *_R (a - b)$ **by** *simp*
  moreover from *scaleR-minus-left* [*of 1 a* $-$ *b*] **have**
    $b - a = (-1) *_R (a - b)$ **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 $b - a = 1 *_R (b - c) + (-l) *_R (b - c)$ **by** *simp*
  with *scaleR-left-distrib* [*of 1* $-l$ *b* $-$ *c*] **have**
    $b - a = (1 + (-l)) *_R (b - c)$ **by** *simp*
  with $\langle c - a = (-l) *_R (b - c)\rangle$ **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 $s = 0$
    with $\langle c - a = s *_R w\rangle$ **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 $s \neq 0$
    with $\langle c - a = s *_R w\rangle$ **have** $w = (1/s) *_R (c - a)$ **by** *simp*
    with $\langle b - a = r *_R w\rangle$ **have** $b - a = (r/s) *_R (c - a)$ **by** *simp*
    have $r/s < 0 \vee (r/s \geq 0 \wedge r/s \leq 1) \vee r/s > 1$ **by** *arith*
    moreover
    { **assume** $r/s \geq 0 \wedge r/s \leq 1$
      with *real-euclid-B-def* **and** $\langle b - a = (r/s) *_R (c - a)\rangle$ **have** $B_{\mathbb{R}}$ *a b c*
        by *auto*
      hence *?thesis* **by** *simp* }
    moreover
    { **assume** $r/s > 1$
      with $\langle b - a = (r/s) *_R (c - a)\rangle$ **have** $c - a = (s/r) *_R (b - a)$ **by** *auto*

from ⟨$r/s > 1$⟩ **and** *le-imp-inverse-le* [*of 1 r/s*] **have**
$s/r \leq 1$ **by** *simp*
from ⟨$r/s > 1$⟩ **and** *inverse-positive-iff-positive* [*of r/s*] **have**
$s/r \geq 0$ **by** *simp*
with *real-euclid-B-def*
and ⟨$c - a = (s/r) *_R (b - a)$⟩
and ⟨$s/r \leq 1$⟩
have $B_{\mathbb{R}}$ *a c b* **by** *auto*
with *real-euclid.th3-2* **have** $B_{\mathbb{R}}$ *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** *simp*
let $?l = 1 \ / \ (-(r/s) + 1)$
from ⟨$-(r/s) + 1 > 1$⟩ **and** *le-imp-inverse-le* [*of* 1 $-(r/s) + 1$] **have**
$?l \leq 1$ **by** *simp*
from ⟨$-(r/s) + 1 > 1$⟩
and *inverse-positive-iff-positive* [*of* $-(r/s) + 1$]
have
$?l \geq 0$ **by** *simp*
with *real-euclid-B-def* **and** ⟨$?l \leq 1$⟩ **and** ⟨$a - c = ?l *_R (b - c)$⟩ **have**
$B_{\mathbb{R}}$ *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 $w\$1 \neq 0$ **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
    { 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
      ... = ((norm (d − q))^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
      ... = (d − ?m) · (?m − q) by simp
    also from inner.minus-right [of d − ?m ?m − p]
      and ‹?m − q = −(?m − p)›
    have
      ... = −((d − ?m) · (?m − p)) by simp
    finally have (d − ?m) · (?m − p) = −((d − ?m) · (?m − p)) .
    thus (d − ?m) · (?m − p) = 0 by arith
  qed }
  note m-lemma = this
  with ‹a p ≡ℝ a q› have (a − ?m) · (?m − p) = 0 by simp
  { fix d
    assume d p ≡ℝ d q
    with m-lemma have (d − ?m) · (?m − p) = 0 by simp
    with dot-left-diff-distrib [of d − ?m a − ?m ?m − p]
      and ‹(a − ?m) · (?m − p) = 0›
    have (d − a) · (?m − p) = 0 by simp }
  with ‹b p ≡ℝ b q› and ‹c p ≡ℝ c q› have
    (b − a) · (?m − p) = 0 and (c − a) · (?m − p) = 0 by simp+
  with real2-orthogonal-dep2 and ‹?m − p ≠ 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 Bℝ a b c ∨ Bℝ b c a ∨ Bℝ c a b by auto }
 thus ∀ p q a b c :: real^2.
      p ≠ q ∧ a p ≡ℝ a q ∧ b p ≡ℝ b q ∧ c p ≡ℝ c q ⟶
        Bℝ a b c ∨ Bℝ b c a ∨ Bℝ c a b
  by blast
qed
```

4.5 Special cases of theorems of Tarski's geometry

```
lemma real-euclid-B-disjunction:
 assumes l ≥ 0 and b − a = l *R (c − a)
 shows Bℝ a b c ∨ Bℝ a c b
proof cases
 assume l ≤ 1
 with ‹l ≥ 0› and ‹b − a = l *R (c − a)›
 have Bℝ a b c by (unfold real-euclid-B-def) (simp add: exI [of - l])
 thus Bℝ a b c ∨ Bℝ a c b ..
next
 assume ¬ (l ≤ 1)
 hence 1/l ≤ 1 by simp
```

```
  from ‹l ≥ 0› have 1/l ≥ 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 ‹¬ (l ≤ 1)› have c − a = (1/l) *R (b − a) by simp
  with ‹1/l ≥ 0› and ‹1/l ≤ 1›
  have Bℝ a c b by (unfold real-euclid-B-def) (simp add: exI [of - 1/l])
  thus Bℝ a b c ∨ Bℝ 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 ≠ b and Bℝ a b c and Bℝ a b d
  shows Bℝ a c d ∨ Bℝ a d c
proof −
  from ‹Bℝ a b c› and ‹Bℝ a b d›
  obtain l and m where l ≥ 0 and b − a = l *R (c − a)
    and m ≥ 0 and b − a = m *R (d − a)
    by (unfold real-euclid-B-def) auto
  from ‹b − a = m *R (d − a)› and ‹a ≠ b› have m ≠ 0 by auto

  from ‹l ≥ 0› and ‹m ≥ 0› have l/m ≥ 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 ≠ 0› have d − a = (l/m) *R (c − a) by simp
  with ‹l/m ≥ 0› and real-euclid-B-disjunction
  show Bℝ a c d ∨ Bℝ a d c by auto
qed

theorem real-euclid-th5-3:
  assumes Bℝ a b d and Bℝ a c d
  shows Bℝ a b c ∨ Bℝ a c b
proof −
  from ‹Bℝ a b d› and ‹Bℝ a c d›
  obtain l and m where l ≥ 0 and b − a = l *R (d − a)
    and m ≥ 0 and c − a = m *R (d − a)
    by (unfold real-euclid-B-def) auto

  show Bℝ a b c ∨ Bℝ 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 Bℝ a b c by (unfold real-euclid-B-def) (simp add: exI [of - l])
    thus Bℝ a b c ∨ Bℝ 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-Algebra*
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-basis :: (real^('n::finite)) set ⇒ bool where
  is-basis S ≜ independent S ∧ 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^('n::finite)) set
  shows independent B ∧ card B = CARD('n) ⟷ is-basis B
proof
  assume independent B ∧ card B = CARD('n)
  hence independent B and card B = CARD('n) by simp+
  from card-finite [of B, where 'n = 'n] and ⟨card B = CARD('n)⟩
  have finite B by simp
  from dim-univ [where 'n = 'n] and ⟨card B = CARD('n)⟩
  have card B = dim (UNIV :: ((real^'n) set))
    by simp
  with card-eq-dim [of B UNIV] and ⟨finite B⟩ and ⟨independent B⟩
  have span B = UNIV by auto
  with ⟨independent B⟩ show is-basis B unfolding is-basis-def ..
next
  assume is-basis B
  hence independent B unfolding is-basis-def ..
  moreover have card B = CARD('n)
  proof −
    have B ⊆ UNIV by simp
    moreover
    { from ⟨is-basis B⟩ have UNIV ⊆ span B and independent B
        unfolding is-basis-def
        by simp+ }
    ultimately have card B = dim (UNIV::((real^'n) set))
      using basis-card-eq-dim [of B UNIV]
      by simp
    with dim-univ [where 'n = 'n] show card B = CARD('n) by simp
  qed
  ultimately show independent B ∧ card B = CARD('n) ..
qed

lemma basis-finite:
  fixes B :: (real^('n::finite)) 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 $\exists c.\ v = (\sum w{\in}B.\ (c\ w) *_R w)$
proof −
  from ⟨*is-basis B*⟩ **have** $v \in$ *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** ⟨$v \in$ *span B*⟩
  show $\exists c.\ v = (\sum w{\in}B.\ (c\ w) *_R w)$ **by** (*simp add*: *scalar-equiv*) *auto*
qed

lemma *not-span-independent-insert*:
  fixes *v* :: (′*a*::*real-vector*)^′*n*
  assumes *independent S* **and** $v \notin$ *span S*
  shows *independent* (*insert v S*)
proof −
  from *span-superset* **and** ⟨$v \notin$ *span S*⟩ **have** $v \notin S$ **by** *auto*
  with *independent-insert* [*of v S*] **and** ⟨*independent S*⟩ **and** ⟨$v \notin$ *span S*⟩
  show *independent* (*insert v S*) **by** *simp*
qed

lemma *in-span-eq*:
  fixes *v* :: (′*a*::*real-vector*)^′*b*
  assumes $v \in$ *span S*
  shows *span* (*insert v S*) = *span S*
proof
  { **fix** *w*
    assume $w \in$ *span* (*insert v S*)
    with ⟨$v \in$ *span S*⟩ **have** $w \in$ *span S* **by** (*rule span-trans*) }
  thus *span* (*insert v S*) ⊆ *span S* **..**

  have *S* ⊆ *insert v S* **by** (*rule subset-insertI*)
  thus *span S* ⊆ *span* (*insert v S*) **by** (*rule span-mono*)
qed

lemma *dot-setsum-right-distrib*:
  fixes *v* :: *real*^′*n*
  shows $v \cdot (\sum j{\in}S.\ w\ j) = (\sum j{\in}S.\ v \cdot (w\ j))$
proof −
  have $v \cdot (\sum j{\in}S.\ w\ j) = (\sum i{\in}UNIV.\ v\$i * (\sum j{\in}S.\ (w\ j)\$i))$
    unfolding *inner-vector-def*
    by *simp*
  also from *setsum-right-distrib* [**where** *?A* = *S* **and** *?′b* = *real*]
  have $\ldots = (\sum i{\in}UNIV.\ \sum j{\in}S.\ v\$i * (w\ j)\$i)$ **by** *simp*

also from *setsum-commute* [*of* λ *i j. v*\$*i* $*$ (*w j*)\$*i S UNIV*]
have ... = ($\sum$ *j*$\in$*S*. $\sum$ *i*$\in$*UNIV*. *v*\$*i* $*$ (*w j*)\$*i*) **by** *simp*
finally show *v* $\cdot$ ($\sum$ *j*$\in$*S*. *w j*) = ($\sum$ *j*$\in$*S*. *v* $\cdot$ (*w j*))
unfolding *inner-vector-def*
by *simp*
qed

lemma *orthogonal-setsum*:
fixes *v* :: *real*^'*n*
assumes $\forall$ *w*$\in$*S*. *orthogonal v w*
shows *orthogonal v* ($\sum$ *w*$\in$*S*. *c w* $*$*s w*)
proof −
from *dot-setsum-right-distrib* [*of v*]
have *v* $\cdot$ ($\sum$ *w*$\in$*S*. *c w* $*$*s w*) = ($\sum$ *w*$\in$*S*. *v* $\cdot$ (*c w* $*$*s w*)) **by** *auto*
with *inner.scaleR-right* [*of v*]
have *v* $\cdot$ ($\sum$ *w*$\in$*S*. *c w* $*$*s w*) = ($\sum$ *w*$\in$*S*. *c w* $*$ (*v* $\cdot$ *w*))
by (*simp add*: *scalar-equiv*)
with ⟨$\forall$ *w*$\in$*S*. *orthogonal v w*⟩ **show** *orthogonal v* ($\sum$ *w*$\in$*S*. *c w* $*$*s w*)
unfolding *orthogonal-def*
by *simp*
qed

lemma *orthogonal-self-eq-0*:
fixes *v* :: ('*a*::*real-inner*)^('*n*::*finite*)
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*::*finite*)
assumes *v* $\in$ *span S* **and** $\forall$ *w*$\in$*S*. *orthogonal v w*
shows *v* = *0*
proof −
from *span-explicit* [*of S*] **and** ⟨*v* $\in$ *span S*⟩
obtain *T* **and** *u* **where** *T* $\subseteq$ *S* **and** *v* = ($\sum$ *w*$\in$*T*. *u w* $*_R$ *w*) **by** *auto*
from ⟨$\forall$ *w*$\in$*S*. *orthogonal v w*⟩ **and** ⟨*T* $\subseteq$ *S*⟩ **have** $\forall$ *w*$\in$*T*. *orthogonal v w* **by** *auto*
with *orthogonal-setsum* [*of T v u*] **and** ⟨*v* = ($\sum$ *w*$\in$*T*. *u w* $*_R$ *w*)⟩
have *orthogonal v v* **by** (*auto simp add*: *scalar-equiv*)
with *orthogonal-self-eq-0* **show** *v* = *0* **by** *auto*
qed

lemma *orthogonal-independent*:
fixes *v* :: *real*^('*n*::*finite*)
assumes *independent S* **and** *v* $\neq$ *0* **and** $\forall$ *w*$\in$*S*. *orthogonal v w*
shows *independent* (*insert v S*)
proof −
from *orthogonal-in-span-eq-0* **and** ⟨*v* $\neq$ *0*⟩ **and** ⟨$\forall$ *w*$\in$*S*. *orthogonal v w*⟩

have $v \notin span\ S$ **by** *auto*
with *not-span-independent-insert* **and** ⟨*independent S*⟩
show *independent* (*insert v S*) **by** *auto*
qed

lemma *card-ge-dim*:
fixes S :: (*real*^(′*n*::*finite*)) *set*
assumes *finite S*
shows $card\ S \geq dim\ S$
proof −
from *span-inc* **have** $S \subseteq span\ S$ **by** *auto*
with *span-card-ge-dim* [*of S span S*] **and** ⟨*finite S*⟩
have $card\ S \geq dim\ (span\ S)$ **by** *simp*
with *dim-span* [*of S*] **show** $card\ S \geq dim\ S$ **by** *simp*
qed

lemma *dot-scaleR-mult*:
shows $(k *_R a) \cdot b = k * (a \cdot b)$ **and** $a \cdot (k *_R b) = k * (a \cdot b)$
unfolding *inner-vector-def*
by (*simp-all add*: *algebra-simps setsum-right-distrib*)

lemma *dependent-explicit-finite*:
fixes S :: ((′*a*::{*real-vector*,*field*})^′*n*) *set*
assumes *finite S*
shows *dependent S* $\longleftrightarrow (\exists\ u.\ (\exists\ v{\in}S.\ u\ v \neq 0) \wedge (\sum\ v{\in}S.\ u\ v *_R v) = 0)$
proof
assume *dependent S*
with *dependent-explicit* [*of S*]
obtain S' **and** u **where**
$S' \subseteq S$ **and** $\exists\ v{\in}S'.\ u\ v \neq 0$ **and** $(\sum\ v{\in}S'.\ u\ v *_R v) = 0$
by *auto*
let $?u' = \lambda\ v.\ if\ v \in S'\ then\ u\ v\ else\ 0$
from ⟨$S' \subseteq S$⟩ **and** ⟨$\exists\ v{\in}S'.\ u\ v \neq 0$⟩ **have** $\exists\ v{\in}S.\ ?u'\ v \neq 0$ **by** *auto*
moreover from *setsum-mono-zero-cong-right* [*of S* S' $\lambda\ v.\ ?u'\ v *_R v$]
and ⟨$S' \subseteq S$⟩ **and** ⟨$(\sum\ v{\in}S'.\ u\ v *_R v) = 0$⟩ **and** ⟨*finite S*⟩
have $(\sum\ v{\in}S.\ ?u'\ v *_R v) = 0$ **by** *simp*
ultimately show $(\exists\ u.\ (\exists\ v{\in}S.\ u\ v \neq 0) \wedge (\sum\ v{\in}S.\ u\ v *_R v) = 0)$ **by** *auto*
next
assume $(\exists\ u.\ (\exists\ v{\in}S.\ u\ v \neq 0) \wedge (\sum\ v{\in}S.\ u\ v *_R v) = 0)$
with *dependent-explicit* [*of S*] **and** ⟨*finite S*⟩
show *dependent S* **by** *auto*
qed

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 ≠ w⟩
    show ∃ i j. (i ≠ 0 ∨ j ≠ 0) ∧ i *R v + j *R w = 0 by auto }

  { assume ∃ i j. (i ≠ 0 ∨ j ≠ 0) ∧ i *R v + j *R w = 0
    then obtain i and j where i ≠ 0 ∨ j ≠ 0 and i *R v + j *R w = 0 by auto
    let ?u = λ x. if x = v then i else j
    from ⟨i ≠ 0 ∨ j ≠ 0⟩ and ⟨v ≠ w⟩ have ∃ x∈?S. ?u x ≠ 0 by simp
    from ⟨i *R v + j *R w = 0⟩ and ⟨v ≠ w⟩
    have (∑ x∈?S. ?u x *R x) = 0 by simp
    with dependent-explicit-finite [of ?S]
      and ⟨finite ?S⟩ and ⟨∃ x∈?S. ?u x ≠ 0⟩
    show dependent ?S by best }
qed
```

5.1 Matrices

```
lemma zero-times:
  0 ** A = (0::real^('n::finite)^'n)
  unfolding matrix-matrix-mult-def and vector-zero-def
  by simp

lemma zero-not-invertible:
  ¬ (invertible (0::real^('n::finite)^'n))
proof −
  let ?Λ = 0::real^'n^'n
  let ?I = mat 1::real^'n^'n
  let ?k = undefined :: 'n
  have ?I $ ?k $ ?k ≠ ?Λ $ ?k $ ?k
    unfolding mat-def
    by simp
  hence ?Λ ≠ ?I by auto
  from zero-times have ∀ A. ?Λ ** A = ?Λ by auto
  with ⟨?Λ ≠ ?I⟩ show ¬ (invertible ?Λ)
    unfolding invertible-def
    by simp
qed
```

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 = (∑ i∈UNIV. (x$i) *s (A$i))
  unfolding vector-matrix-mult-def
  by (simp add: Cart-eq mult-commute)

lemma invertible-mult:
```

```
  fixes A B :: real^('n::finite)^'n
  assumes invertible A and invertible B
  shows invertible (A ** B)
proof -
  from ‹invertible A› and ‹invertible B›
  obtain A' and B' where A ** A' = mat 1 and A' ** A = mat 1
    and B ** B' = mat 1 and B' ** B = mat 1
    unfolding invertible-def
    by auto
  have (A ** B) ** (B' ** A') = A ** (B ** B') ** A'
    by (simp add: matrix-mul-assoc)
  with ‹A ** A' = mat 1› and ‹B ** B' = mat 1›
  have (A ** B) ** (B' ** A') = mat 1 by (auto simp add: matrix-mul-rid)
  with matrix-left-right-inverse have (B' ** A') ** (A ** B) = mat 1 by auto
  with ‹(A ** B) ** (B' ** A') = mat 1›
  show invertible (A ** B)
    unfolding invertible-def
    by auto
qed

lemma scalar-matrix-assoc:
  fixes A :: real^'m^'n
  shows k *R (A ** B) = (k *R A) ** B
proof -
  have ∀ i j. (k *R (A ** B))$i$j = ((k *R A) ** B)$i$j
  proof default+
    fix i j
    have (k *R (A ** B))$i$j = k * (∑ l∈UNIV. A$i$l * B$l$j)
      unfolding matrix-matrix-mult-def
      by simp
    also from scaleR-right.setsum [of k λ l. A$i$l * B$l$j UNIV]
    have ... = (∑ l∈UNIV. k * A$i$l * B$l$j) by (simp add: algebra-simps)
    finally show (k *R (A ** B))$i$j = ((k *R A) ** B)$i$j
      unfolding matrix-matrix-mult-def
      by simp
  qed
  thus k *R (A ** B) = (k *R A) ** B by (simp add: Cart-eq)
qed

lemma transpose-scalar: transpose (k *R A) = k *R transpose A
  unfolding transpose-def
  by (simp add: Cart-eq)

lemma transpose-iff [iff]: transpose A = transpose B ⟷ A = B
proof
  assume transpose A = transpose B
  with transpose-transpose [of A] have A = transpose (transpose B) by simp
  with transpose-transpose [of B] show A = B by simp
next
```

assume $A = B$
thus *transpose* $A =$ *transpose* B **by** *simp*
qed

lemma *matrix-scalar-ac*:
fixes A :: *real^'m^'n*
shows $A ** (k *_R B) = k *_R A ** B$
proof −
from *matrix-transpose-mul* [*of* A $k *_R B$] **and** *transpose-scalar* [*of* k B]
have *transpose* $(A ** (k *_R B)) = k *_R$ *transpose* $B **$ *transpose* A
by *simp*
also from *matrix-transpose-mul* [*of* A B] **and** *transpose-scalar* [*of* k $A ** B$]
have $\ldots =$ *transpose* $(k *_R A ** B)$ **by** (*simp add*: *scalar-matrix-assoc*)
finally show $A ** (k *_R B) = k *_R A ** B$ **by** *simp*
qed

lemma *scalar-invertible*:
fixes A :: *real^'m^'n*
assumes $k \neq 0$ **and** *invertible* A
shows *invertible* $(k *_R A)$
proof −
from ⟨*invertible* A⟩
obtain A' **where** $A ** A' =$ *mat 1* **and** $A' ** A =$ *mat 1*
unfolding *invertible-def*
by *auto*
with ⟨$k \neq 0$⟩
have $(k *_R A) ** ((1/k) *_R A') =$ *mat 1*
and $((1/k) *_R A') ** (k *_R A) =$ *mat 1*
by (*simp-all add*: *matrix-scalar-ac*)
thus *invertible* $(k *_R A)$
unfolding *invertible-def*
by *auto*
qed

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* λ N. $M ** N =$ *mat 1* $\wedge$ $N ** M =$ *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 *vector-matrix-mul-rid*:
fixes v :: $('a::semiring\text{-}1)^\wedge('n::finite)$
shows $v\ v{*}\ mat\ 1 = v$
proof $-$
have $v\ v{*}\ mat\ 1 = transpose\ (mat\ 1)\ {*}v\ v$ **by** *simp*
thus $v\ v{*}\ mat\ 1 = v$ **by** (*simp only*: *transpose-mat matrix-vector-mul-lid*)
qed

lemma *vector-matrix-mul-assoc*:
fixes v :: $('a::comm\text{-}semiring\text{-}1)^\wedge'n$
shows $(v\ v{*}\ M)\ v{*}\ N = v\ v{*}\ (M\ {*}{*}\ N)$
proof $-$
from *matrix-vector-mul-assoc*
have $transpose\ N\ {*}v\ (transpose\ M\ {*}v\ v) = (transpose\ N\ {*}{*}\ transpose\ M)\ {*}v\ v$ **by** *fast*
thus $(v\ v{*}\ M)\ v{*}\ N = v\ v{*}\ (M\ {*}{*}\ N)$
by (*simp add*: *matrix-transpose-mul* [*symmetric*])
qed

lemma *matrix-scalar-vector-ac*:
fixes A :: $real^\wedge('m::finite)^\wedge('n::finite)$
shows $A\ {*}v\ (k\ {*}_R\ v) = k\ {*}_R\ A\ {*}v\ v$
proof $-$
have $A\ {*}v\ (k\ {*}_R\ v) = k\ {*}_R\ (v\ v{*}\ transpose\ A)$
by (*subst scalar-vector-matrix-assoc* [*symmetric*]) *simp*
also have $\ldots = v\ v{*}\ k\ {*}_R\ transpose\ A$
by (*subst vector-scalar-matrix-ac*) *simp*
also have $\ldots = v\ v{*}\ transpose\ (k\ {*}_R\ A)$ **by** (*subst transpose-scalar*) *simp*
also have $\ldots = k\ {*}_R\ A\ {*}v\ v$ **by** *simp*
finally show $A\ {*}v\ (k\ {*}_R\ v) = k\ {*}_R\ A\ {*}v\ v$ **.**
qed

lemma *scalar-matrix-vector-assoc*:
fixes A :: $real^\wedge('m::finite)^\wedge('n::finite)$
shows $k\ {*}_R\ (A\ {*}v\ v) = k\ {*}_R\ A\ {*}v\ v$
proof $-$
have $k\ {*}_R\ (A\ {*}v\ v) = k\ {*}_R\ (v\ v{*}\ transpose\ A)$ **by** *simp*
also have $\ldots = v\ v{*}\ k\ {*}_R\ transpose\ A$
by (*rule vector-scalar-matrix-ac* [*symmetric*])
also have $\ldots = v\ v{*}\ transpose\ (k\ {*}_R\ A)$ **apply** (*subst transpose-scalar*) **..**
finally show $k\ {*}_R\ (A\ {*}v\ v) = k\ {*}_R\ A\ {*}v\ v$ **by** *simp*
qed

lemma *invertible-times-non-zero*:
fixes M :: $real^\wedge'n^\wedge('n::finite)$
assumes *invertible* M **and** $v \neq 0$
shows $M\ {*}v\ v \neq 0$
using ⟨*invertible* M⟩ **and** ⟨$v \neq 0$⟩ **and** *invertible-times-eq-zero* [*of* $M\ v$]
by *auto*

```
lemma matrix-right-invertible-ker:
  fixes M :: real^('m::finite)^('n::finite)
  shows (∃ M′. M ** M′ = mat 1) ⟷ (∀ x. x v* M = 0 ⟶ x = 0)
proof
  assume ∃ M′. M ** M′ = mat 1
  then obtain M′ where M ** M′ = mat 1 ..
  have transpose (M ** M′) = transpose (mat 1) apply (subst ‹M ** M′ = mat 1›) ..
  hence transpose M′ ** transpose M = mat 1
    by (simp add: matrix-transpose-mul transpose-mat)
  hence ∃ M″. M″ ** transpose M = mat 1 ..
  with matrix-left-invertible-ker [of transpose M]
  have ∀ x. transpose M *v x = 0 ⟶ x = 0 by simp
  thus ∀ x. x v* M = 0 ⟶ x = 0 by simp
next
  assume ∀ x. x v* M = 0 ⟶ x = 0
  hence ∀ x. transpose M *v x = 0 ⟶ x = 0 by simp
  with matrix-left-invertible-ker [of transpose M]
  obtain M″ where M″ ** transpose M = mat 1 by auto
  hence transpose (M″ ** transpose M) = transpose (mat 1) by simp
  hence M ** transpose M″ = mat 1
    by (simp add: matrix-transpose-mul transpose-transpose transpose-mat)
  thus ∃ M′. M ** M′ = mat 1 ..
qed

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

lemma right-invertible-iff-invertible:
  fixes M :: real^('n::finite)^'n
  shows (∃ N. M ** N = mat 1) ⟷ invertible M
  using left-invertible-iff-invertible
  by (subst matrix-left-right-inverse) auto

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 ** transpose M ** transpose N
    by (simp add: matrix-transpose-mul transpose-transpose matrix-mul-assoc)
  with ‹symmatrix M›
```

show *symmatrix* $(N ** M ** transpose\ N)$
unfolding *symmatrix-def*
by *simp*
qed

lemma *matrix-vector-right-distrib*:
fixes $v\ w :: real\hat{}('n::finite)$ **and** $M :: real\hat{}'n\hat{}('m::finite)$
shows $M *v (v + w) = M *v\ v + M *v\ w$
proof $-$
have $M *v (v + w) = (v + w)\ v*\ transpose\ M$ **by** *simp*
also have $\ldots = v\ v*\ transpose\ M + w\ v*\ transpose\ M$
by (*rule vector-matrix-left-distrib* [*of v w transpose M*])
finally show $M *v (v + w) = M *v\ v + M *v\ w$ **by** *simp*
qed

lemma *non-zero-mult-invertible-non-zero*:
fixes $M :: real\hat{}'n\hat{}'n$
assumes $v \neq 0$ **and** *invertible M*
shows $v\ v*\ M \neq 0$
using ⟨$v \neq 0$⟩ **and** ⟨*invertible M*⟩ **and** *times-invertible-eq-zero*
by *auto*

end

6 Group Actions

theory *Action*
imports *Group*
begin

locale *action* = *group* +
fixes *act* :: $'b \Rightarrow 'a \Rightarrow 'b$ (**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-Algebra*
Euclid-Tarski
Group
Action
begin

7.1 Proportionality on non-zero vectors

context *vector-space*
begin

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

definition *non-zero-vectors* :: $'b\ set$ **where**

```
  non-zero-vectors ≜ {x. x ≠ 0}

lemma proportionality-refl-on: refl-on non-zero-vectors proportionality
proof -
  have proportionality ⊆ non-zero-vectors × non-zero-vectors
    unfolding proportionality-def non-zero-vectors-def
    by auto
  moreover have ∀ x∈non-zero-vectors. (x, x) ∈ proportionality
  proof
    fix x
    assume x ∈ non-zero-vectors
    hence x ≠ 0 unfolding non-zero-vectors-def ..
    moreover have x = scale 1 x by simp
    ultimately show (x, x) ∈ proportionality
      unfolding proportionality-def
      by blast
  qed
  ultimately show refl-on non-zero-vectors proportionality
    unfolding refl-on-def ..
qed

lemma proportionality-sym: sym proportionality
proof -
  { fix x y
    assume (x, y) ∈ proportionality
    hence x ≠ 0 and y ≠ 0 and ∃ k. x = scale k y
      unfolding proportionality-def
      by simp+
    from ‹∃ k. x = scale k y› obtain k where x = scale k y by auto
    with ‹x ≠ 0› have k ≠ 0 by simp
    with ‹x = scale k y› have y = scale (1/k) x by simp
    with ‹x ≠ 0› and ‹y ≠ 0› have (y, x) ∈ proportionality
      unfolding proportionality-def
      by auto
  }
  thus sym proportionality
    unfolding sym-def
    by blast
qed

lemma proportionality-trans: trans proportionality
proof -
  { fix x y z
    assume (x, y) ∈ proportionality and (y, z) ∈ proportionality
    hence x ≠ 0 and z ≠ 0 and ∃ j. x = scale j y and ∃ k. y = scale k z
      unfolding proportionality-def
      by simp+
    from ‹∃ j. x = scale j y› and ‹∃ k. y = scale k z›
    obtain j and k where x = scale j y and y = scale k z by auto+
    hence x = scale (j * k) z by simp
```

```
    with ‹x ≠ 0› and ‹z ≠ 0› have (x, z) ∈ proportionality
      unfolding proportionality-def
      by auto
  }
  thus trans proportionality
    unfolding trans-def
    by blast
qed

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

end

sublocale vector-space < equiv non-zero-vectors proportionality
  using proportionality-equiv .

definition invertible-proportionality ::
  ((real^('n::finite)^'n) × (real^'n^'n)) set where
  invertible-proportionality ≜
  real-vector.proportionality ∩ (Collect invertible × Collect invertible)

lemma invertible-proportionality-equiv:
  equiv (Collect invertible :: (real^('n::finite)^'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
```

from *basis-nonzero*
have $(basis\ 1 :: real\hat{}3) \in$ *real-vector.non-zero-vectors*
unfolding *real-vector.non-zero-vectors-def* **..**
thus *real-vector.proportionality* `` $\{basis\ 1\} \in$
$($*real-vector.non-zero-vectors* :: $(real\hat{}3)$ *set*$)$//*real-vector.proportionality*
unfolding *quotient-def*
by *auto*
qed

definition *proj2-rep* :: *proj2* $\Rightarrow$ *real^3* **where**
proj2-rep x $\triangleq \epsilon\ v.\ v \in$ *Rep-proj2 x*

definition *proj2-abs* :: *real^3* $\Rightarrow$ *proj2* **where**
proj2-abs v $\triangleq$ *Abs-proj2* $($*real-vector.proportionality* `` $\{v\})$

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

lemma *proj2-rep-non-zero*: *proj2-rep x* $\neq 0$
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* $\in$ *real-vector.non-zero-vectors*
unfolding *quotient-def* **and** *proj2-def*
by *auto*
thus *proj2-rep x* $\neq 0$
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 ∈ real-vector.non-zero-vectors›
  have real-vector.proportionality `` {v} ∈ proj2
    unfolding proj2-def
      and 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) ∈ real-vector.proportionality `` {v} by auto
  thus (v, proj2-rep (proj2-abs v)) ∈ 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
    unfolding proj2-def
    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 ≠ 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 ≠ 0
  with ‹c ≠ 0›
  have (c *R v, v) ∈ real-vector.proportionality
    and c *R v ∈ real-vector.non-zero-vectors
    and v ∈ 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 ‹$\mathit{proj2\text{-}abs}\ v = \mathit{proj2\text{-}abs}\ w$›
  have $\mathit{proj2\text{-}rep}\ (\mathit{proj2\text{-}abs}\ v) = \mathit{proj2\text{-}rep}\ (\mathit{proj2\text{-}abs}\ w)$ **by** *simp*
  with *proj2-rep-abs2* **and** ‹$w \neq 0$›
  obtain k **where** $\mathit{proj2\text{-}rep}\ (\mathit{proj2\text{-}abs}\ v) = k *_R w$ **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** $i \neq 0 \vee j \neq 0$ **and** $i *_R p + j *_R q = 0$
  shows $\mathit{proj2\text{-}abs}\ p = \mathit{proj2\text{-}abs}\ q$
proof −
  have $i \neq 0$
  proof
    assume $i = 0$
    with ‹$i \neq 0 \vee j \neq 0$› **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 $\mathit{proj2\text{-}abs}\ p = \mathit{proj2\text{-}abs}\ (i *_R p)$ **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 $\ldots = \mathit{proj2\text{-}abs}\ (j *_R q)$ **by** (*simp add*: *algebra-simps* [*symmetric*])
  also from ‹$j \neq 0$› **have** $\ldots = \mathit{proj2\text{-}abs}\ q$ **by** (*rule proj2-abs-mult*)
  finally show $\mathit{proj2\text{-}abs}\ p = \mathit{proj2\text{-}abs}\ q$ **.**
qed

lemma *proj2-rep-dependent*:
  assumes $i *_R \mathit{proj2\text{-}rep}\ v + j *_R \mathit{proj2\text{-}rep}\ w = 0$
  (**is** $i *_R\ ?p + j *_R\ ?q = 0$)
  and $i \neq 0 \vee j \neq 0$
  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 $\mathit{proj2\text{-}abs}\ ?p = \mathit{proj2\text{-}abs}\ ?q$ **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* $\{\mathit{proj2\text{-}rep}\ p, \mathit{proj2\text{-}rep}\ q\}$

proof
let $?p' = proj2\text{-}rep\ p$
let $?q' = proj2\text{-}rep\ q$
let $?S = \{?p', ?q'\}$
assume *dependent ?S*
from *proj2-rep-inj* **and** $\langle p \neq q \rangle$ **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 $\langle p \neq q \rangle$ **show** *False* **..**
qed

7.3 Lines of the real projective plane

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

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 $\langle p \neq 0 \rangle$ **and** *proj2-rep-abs2*
obtain i' **where** $i' \neq 0$ **and** $proj2\text{-}rep\ ?pp = i' *_R p$ (**is** $?rp = \text{-}$) **by** *auto*
from $\langle q \neq 0 \rangle$ **and** *proj2-rep-abs2*
obtain j' **where** $j' \neq 0$ **and** $proj2\text{-}rep\ ?pq = j' *_R q$ (**is** $?rq = \text{-}$) **by** *auto*
from $\langle r \neq 0 \rangle$ **and** *proj2-rep-abs2*
obtain k' **where** $k' \neq 0$ **and** $proj2\text{-}rep\ ?pr = k' *_R r$ (**is** $?rr = \text{-}$) **by** *auto*
with $\langle i *_R p + j *_R q + k *_R r = 0 \rangle$
and $\langle i' \neq 0 \rangle$ **and** $\langle proj2\text{-}rep\ ?pp = i' *_R p \rangle$
and $\langle j' \neq 0 \rangle$ **and** $\langle proj2\text{-}rep\ ?pq = j' *_R q \rangle$
have $(i/i') *_R ?rp + (j/j') *_R ?rq + (k/k') *_R ?rr = 0$ **by** *simp*

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 $i/i' \neq 0 \vee j/j' \neq 0 \vee k/k' \neq 0$ **by** *simp*
with $\langle (i/i') *_R ?rp + (j/j') *_R ?rq + (k/k') *_R ?rr = 0 \rangle$
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 $?a' = proj2\text{-}rep\ a$
  let $?b' = proj2\text{-}rep\ b$
  let $?c' = proj2\text{-}rep\ c$
  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*: *add-ac*)
  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 $1 *_R proj2\text{-}rep\ a + (-1) *_R proj2\text{-}rep\ a + 0 *_R proj2\text{-}rep\ c = 0$
    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 $proj2\text{-}Col\ a\ r\ t \longleftrightarrow$
  $t = a \vee (\exists\ i.\ t = proj2\text{-}abs\ (i *_R (proj2\text{-}rep\ a) + (proj2\text{-}rep\ r)))$
proof
  let $?a' = proj2\text{-}rep\ a$
  let $?r' = proj2\text{-}rep\ r$
  let $?t' = proj2\text{-}rep\ t$

  { **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 $t = a \vee (\exists\ i.\ t = proj2\text{-}abs\ (i *_R ?a' + ?r'))$

```
  proof cases
    assume j = 0
    with ‹h ≠ 0 ∨ j ≠ 0 ∨ k ≠ 0› have h ≠ 0 ∨ k ≠ 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 ∨ (∃ i. t = proj2-abs (i *R ?a' + ?r')) ..
  next
    assume j ≠ 0
    have k ≠ 0
    proof (rule ccontr)
      assume ¬ k ≠ 0
      with proj2-rep-dependent
        and ‹h *R ?a' + j *R ?r' + k *R ?t' = 0›
        and ‹j ≠ 0›
      have a = r by simp
      with ‹a ≠ 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 ≠ 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 ≠ 0›
    have proj2-abs ((h/j) *R ?a' + ?r') = t
      by (simp add: scaleR-right-distrib)
    hence ∃ i. t = proj2-abs (i *R ?a' + ?r') by auto
    thus t = a ∨ (∃ i. t = proj2-abs (i *R ?a' + ?r')) ..
  qed
}

{ assume t = a ∨ (∃ i. t = 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 ≠ a
    with ‹t = a ∨ (∃ i. t = proj2-abs (i *R ?a' + ?r'))›
    obtain i where t = proj2-abs (i *R ?a' + ?r') by auto
    from proj2-rep-dependent [of i a 1 r] and ‹a ≠ r›
    have i *R ?a' + ?r' ≠ 0 by auto
    with proj2-rep-abs2 and ‹t = proj2-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 *default+*
from $\langle(j * i) *_R ?a' + j *_R ?r' + (-1) *_R ?t' = 0\rangle$
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* $\Rightarrow$ *proj2* $\Rightarrow$ *proj2* $\Rightarrow$ *real* **where**
proj2-Col-coeff a r t $\triangleq \epsilon\ i.\ t =$ *proj2-abs* $(i *_R$ *proj2-rep a* $+$ *proj2-rep r*$)$

lemma *proj2-Col-coeff*:
assumes *proj2-Col a r t* **and** $a \neq r$ **and** $t \neq a$
shows $t =$ *proj2-abs* $(($*proj2-Col-coeff a r t*$) *_R$ *proj2-rep a* $+$ *proj2-rep r*$)$
proof $-$
from $\langle a \neq r\rangle$ **and** $\langle$*proj2-Col a r t*$\rangle$ **and** $\langle t \neq a\rangle$ **and** *proj2-Col-iff*
have $\exists\ i.\ t =$ *proj2-abs* $(i *_R$ *proj2-rep a* $+$ *proj2-rep r*$)$ **by** *simp*
thus $t =$ *proj2-abs* $(($*proj2-Col-coeff a r t*$) *_R$ *proj2-rep a* $+$ *proj2-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-abs a* $\neq$ *proj2-abs r*
and *proj2-abs* $(i *_R a + r) =$ *proj2-abs* $(j *_R a + r)$
shows $i = j$
proof $-$
from $\langle a \neq 0\rangle$ **and** $\langle r \neq 0\rangle$ **and** $\langle$*proj2-abs a* $\neq$ *proj2-abs r*$\rangle$
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-rep* (*proj2-abs* $(i *_R a + r)) = k *_R (i *_R a + r)$
and *proj2-rep* (*proj2-abs* $(j *_R a + r)) = l *_R (j *_R a + r)$
by *auto*
with $\langle$*proj2-abs* $(i *_R a + r) =$ *proj2-abs* $(j *_R a + r)\rangle$
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 Cart-eq*)
with $\langle a \neq 0\rangle$ **and** $\langle r \neq 0\rangle$ **and** $\langle$*proj2-abs a* $\neq$ *proj2-abs r*$\rangle$
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 ‹$k \neq 0$› **show** $i = j$ **by** *simp*
qed

lemma *proj2-Col-coeff-unique*:
assumes $a \neq r$
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 $?a' \neq 0$ **and** $?r' \neq 0$ **by** $($*rule proj2-rep-non-zero*$)+$

from ‹$a \neq r$› **have** *proj2-abs* $?a' \neq$ *proj2-abs* $?r'$ **by** $($*simp add*: *proj2-abs-rep*$)$
with ‹$?a' \neq 0$› **and** ‹$?r' \neq 0$›
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* $\Rightarrow$ *proj2* **where**
L2P l $\triangleq$ *case l of P2L p* $\Rightarrow$ *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* $= l \longleftrightarrow p =$ *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* $\Rightarrow$ *proj2-line* **where**
proj2-line-abs v $\triangleq$ *P2L* (*proj2-abs v*)

definition *proj2-line-rep* :: *proj2-line* $\Rightarrow$ *real^3* **where**
proj2-line-rep l $\triangleq$ *proj2-rep* (*L2P l*)

lemma *proj2-line-rep-abs*:
assumes $v \neq 0$
shows $\exists$ *k*. $k \neq 0 \wedge$ *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** ‹$v \neq 0$›
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* $\neq 0$
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 $i \neq 0 \vee j \neq 0$
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 $k \neq 0$
shows *proj2-line-abs* ($k *_R v$) = *proj2-line-abs v*
unfolding *proj2-line-abs-def*
using ‹$k \neq 0$›
by (*subst proj2-abs-mult*) *simp-all*

lemma *proj2-line-abs-abs-mult*:
assumes *proj2-line-abs v* = *proj2-line-abs w* **and** $w \neq 0$
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 *card-ge-dim* [*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-inc* [*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 $\triangleq$ ϵ *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 $\forall$ *w*$\in$*?A. orthogonal ?m′ w* **and** $\forall$ *w*$\in$*?A. orthogonal ?l′ w*

```
    unfolding proj2-incident-def and orthogonal-def
    by (simp-all add: inner-commute)
  from proj2-rep-independent and ‹p ≠ q› have independent ?A by simp
  from proj2-line-rep-non-zero have ?m' ≠ 0 by simp
  with orthogonal-independent
    and ‹independent ?A› and ‹∀ w∈?A. orthogonal ?m' w›
  have independent ?B by auto

  from proj2-rep-inj and ‹p ≠ q› have ?p' ≠ ?q'
    unfolding inj-on-def
    by auto
  hence card ?A = 2 by simp
  moreover have ?m' ∉ ?A
  proof
    assume ?m' ∈ ?A
    with span-inc [of ?A] have ?m' ∈ span ?A by auto
    with orthogonal-in-span-eq-0 and ‹∀ w∈?A. orthogonal ?m' w›
    have ?m' = 0 by auto
    with ‹?m' ≠ 0› show False ..
  qed
  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' = (∑ v∈?B. c v *R v) by auto
  let ?l'' = ?l' − c ?m' *R ?m'
  from ‹?l' = (∑ v∈?B. c v *R v)› and ‹?m' ∉ ?A›
  have ?l'' = (∑ v∈?A. c v *R v) by simp
  with orthogonal-setsum [of ?A]
    and ‹∀ w∈?A. orthogonal ?l' w› and ‹∀ w∈?A. orthogonal ?m' w›
  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 ∨ l = m
proof cases
  assume p = q
  thus p = q ∨ 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) ⟷ v · w = 0
    unfolding proj2-incident-def
```

```
    by (simp add: dot-scaleR-mult)
qed

lemma proj2-incident-left-abs:
  assumes v ≠ 0
  shows proj2-incident (proj2-abs v) l ⟷ v · (proj2-line-rep l) = 0
proof −
  have proj2-line-rep l ≠ 0 by (rule proj2-line-rep-non-zero)
  with ⟨v ≠ 0⟩ and proj2-incident-abs [of v proj2-line-rep l]
  show proj2-incident (proj2-abs v) l ⟷ v · (proj2-line-rep l) = 0 by simp
qed

lemma proj2-incident-right-abs:
  assumes v ≠ 0
  shows proj2-incident p (proj2-line-abs v) ⟷ (proj2-rep p) · v = 0
proof −
  have proj2-rep p ≠ 0 by (rule proj2-rep-non-zero)
  with ⟨v ≠ 0⟩ and proj2-incident-abs [of proj2-rep p v]
  show proj2-incident p (proj2-line-abs v) ⟷ (proj2-rep p) · v = 0
    by (simp add: proj2-abs-rep)
qed

definition proj2-set-Col :: proj2 set ⇒ bool where
  proj2-set-Col S ≜ ∃ l. ∀ p∈S. proj2-incident p l

lemma proj2-subset-Col:
  assumes T ⊆ S and proj2-set-Col S
  shows proj2-set-Col T
  using ⟨T ⊆ S⟩ and ⟨proj2-set-Col S⟩
  by (unfold proj2-set-Col-def) auto

definition proj2-no-3-Col :: proj2 set ⇒ bool where
  proj2-no-3-Col S ≜ card S = 4 ∧ (∀ p∈S. ¬ proj2-set-Col (S − {p}))

lemma proj2-Col-iff-not-invertible:
  proj2-Col p q r
  ⟷ ¬ invertible (vector [proj2-rep p, proj2-rep q, proj2-rep r] :: real^3^3)
  (is - ⟷ ¬ invertible (vector [?u, ?v, ?w]))
proof −
  let ?M = vector [?u,?v,?w] :: real^3^3
  have proj2-Col p q r ⟷ (∃ x. x ≠ 0 ∧ x v∗ ?M = 0)
  proof
    assume proj2-Col p q r
    then obtain i and j and k
      where i ≠ 0 ∨ j ≠ 0 ∨ k ≠ 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: Cart-eq 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: setsum-3 Cart-eq 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: Cart-eq 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 setsum-3 and vector-def
        by (simp add: Cart-eq 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-vector-def*
and *setsum-3*
by (*simp add*: *Cart-eq 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-vector-def*
and *setsum-3*
and *proj2-incident-def*
by (*simp add*: *Cart-eq*) }
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 ∈ rows ?M
    then obtain i :: 3  where x = ?M $ i
      unfolding rows-def and row-def
      by (auto simp add: Cart-nth-inverse)
    with exhaust-3 have x = ?u ∨ x = ?v ∨ x = ?w
      unfolding vector-def
      by auto
    hence x ∈ {?u, ?v, ?w} by simp }
  thus rows ?M ⊆ {?u, ?v, ?w} ..
  { fix x
    assume x ∈ {?u, ?v, ?w}
    hence x = ?u ∨ x = ?v ∨ x = ?w by simp
    hence x = ?M $ 1 ∨ x = ?M $ 2 ∨ x = ?M $ 3
      unfolding vector-def
      by simp
    hence x ∈ rows ?M
      unfolding rows-def and row-def
      by (auto simp add: Cart-nth-inverse) }
  thus {?u, ?v, ?w} ⊆ rows ?M ..
 qed
 with ‹S = {p,q,r}›
 have rows ?M = proj2-rep ‘ S
  unfolding image-def
  by auto
 with ‹¬ proj2-set-Col S ⟷ span (rows ?M) = UNIV›
 show ¬ proj2-set-Col S ⟷ span (proj2-rep ‘ S) = UNIV by simp
qed

lemma proj2-no-3-Col-span:
 assumes proj2-no-3-Col S and p ∈ 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 ∈ 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 ∈ S›
 have ¬ 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 ¬ proj2-Col p q r
 shows ∃ s. proj2-no-3-Col {s,r,p,q}
proof −
 from ‹¬ proj2-Col p q r› and proj2-Col-coincide have p ≠ q by auto
```

hence *card* $\{p,q\} = 2$ **by** *simp*

from ⟨$\neg$ *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*: *setsum-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*:
      setsum-diff1' [*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** ⟨$?s \in \{r,p,q\}$⟩
  have ... = $-j *_R$ *proj2-rep ?s* + ($\sum$ $t \in \{r,p,q\}$. *proj2-rep t*)

  by (*simp only*:
    setsum-diff1′ [*of* {*r*,*p*,*q*} *?s* λ *t*. *proj2-rep t*,*symmetric*])
  also from ⟨($\sum$ *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 ($\sum$ *t*∈{*r*,*p*,*q*}. *proj2-rep t*)
      = *proj2-rep u* + ($\sum$ *t*∈{*r*,*p*,*q*}−{*u*}. *proj2-rep t*)
      by (*simp add*: *setsum-diff1′*)
    with ⟨($\sum$ *t*∈{*r*,*p*,*q*}. *proj2-rep t*) = *j* $*_R$ *proj2-rep ?s*⟩
    have *proj2-rep u*
      = *j* $*_R$ *proj2-rep ?s* − ($\sum$ *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* + ($\sum$ *t*∈{*r*,*p*,*q*}−{*u*}. *?d t* $*_R$ *proj2-rep t*)
      by (*simp add*: *setsum-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: setsum-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 (simp add: span-setsum)

    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-inc [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** $\{p,q,r\} \subseteq S$ **and** $p \neq q$ **and** $r \neq p$
  shows $\exists$ *k*. $r =$ *proj2-abs* ($k *_R$ *proj2-rep p* + *proj2-rep q*)
proof −
  from ⟨*proj2-set-Col S*⟩
  obtain *l* **where** $\forall$ $t \in S$. *proj2-incident t l* **unfolding** *proj2-set-Col-def* **..**
  with ⟨$\{p,q,r\} \subseteq S$⟩ **and** ⟨$p \neq q$⟩ **and** ⟨$r \neq p$⟩ **and** *proj2-incident-iff* [*of p q l r*]
  show $\exists$ *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^3^3*) *set*)//*invertible-proportionality*
proof
  from *matrix-id-invertible* **have** (*mat 1* :: *real^3^3*) ∈ *Collect invertible*
    by *simp*
  thus *invertible-proportionality* '' {*mat 1*} ∈
    (*Collect invertible* :: (*real^3^3*) *set*)//*invertible-proportionality*
    unfolding *quotient-def*
    by *auto*
qed

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

definition *cltn2-abs* :: *real^3^3* ⇒ *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*
    unfolding *cltn2-def*
    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 and cltn2-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} ∈ cltn2
  unfolding cltn2-def
   and 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*
unfolding *cltn2-def*
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*: *scalar-vector-matrix-assoc vector-scalar-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-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-rep p* $v*$ *cltn2-rep* $A \neq 0$
using *proj2-rep-non-zero*
by (*rule non-zero-mult-rep-non-zero*)

definition *cltn2-image* :: *proj2 set* ⇒ *cltn2* ⇒ *proj2 set* **where**
cltn2-image P A ≜ {*apply-cltn2 p A* | *p*. $p \in P$}

7.4.1 As a group

definition *cltn2-id* :: *cltn2* **where**
cltn2-id ≜ *cltn2-abs* (*mat 1*)

definition *cltn2-compose* :: *cltn2* ⇒ *cltn2* ⇒ *cltn2* **where**
cltn2-compose A B ≜ *cltn2-abs* (*cltn2-rep A* $**$ *cltn2-rep B*)

definition *cltn2-inverse* :: *cltn2* ⇒ *cltn2* **where**
cltn2-inverse A ≜ *cltn2-abs* (*matrix-inv* (*cltn2-rep A*))

lemma *cltn2-compose-abs*:
assumes *invertible M* **and** *invertible N*
shows *cltn2-compose* (*cltn2-abs M*) (*cltn2-abs N*) = *cltn2-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-rep* (*cltn2-abs M*) = $j *_R M$
and *cltn2-rep* (*cltn2-abs N*) = $k *_R N$
by *blast*

from ⟨$j \neq 0$⟩ **and** ⟨$k \neq 0$⟩ **have** $j * k \neq 0$ **by** *simp*

from ⟨*cltn2-rep* (*cltn2-abs M*) = $j *_R M$⟩ **and** ⟨*cltn2-rep* (*cltn2-abs N*) = $k *_R N$⟩
have *cltn2-rep* (*cltn2-abs M*) $**$ *cltn2-rep* (*cltn2-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-compose* (*cltn2-abs M*) (*cltn2-abs N*) = *cltn2-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*]:
$inv_{(|carrier = UNIV,\ mult = cltn2\text{-}compose,\ one = cltn2\text{-}id|)}\ A$
= *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 *auto*
thus *apply-cltn2 p cltn2-id* = *p*
by (*simp add*: *vector-matrix-mul-rid 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 $A \in$ *carrier ?G* $\wedge$ $B \in$ *carrier ?G*
$\longrightarrow$ *apply-cltn2* (*apply-cltn2 p A*) *B* = *apply-cltn2 p* ($A \otimes_{?G} B$)
..
qed

definition *cltn2-transpose* :: *cltn2* $\Rightarrow$ *cltn2* **where**
cltn2-transpose A $\triangleq$ *cltn2-abs* (*transpose* (*cltn2-rep A*))

definition *apply-cltn2-line* :: *proj2-line* $\Rightarrow$ *cltn2* $\Rightarrow$ *proj2-line* **where**
apply-cltn2-line l A
$\triangleq$ *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** $k \neq 0$ **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 ⟨$k \neq 0$⟩ **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* $\neq$ *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*)
$\longleftrightarrow$ *proj2-incident* (*apply-cltn2 p* (*cltn2-inverse A*)) *l*
proof $-$
have *proj2-rep p v$*$ cltn2-rep* (*cltn2-inverse A*) $\neq$ *0*
by (*rule rep-mult-rep-non-zero*)
with *proj2-rep-abs2*
obtain *j* **where** *j* $\neq$ *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* $\neq$ *0* **by** (*rule rep-mult-line-rep-non-zero*)
with *proj2-line-rep-abs* [*of ?v*]
obtain *k* **where** *k* $\neq$ *0*
and *proj2-line-rep* (*proj2-line-abs ?v*) $= k *_R$ *?v*
by *auto*
hence *proj2-incident p* (*apply-cltn2-line l A*)
$\longleftrightarrow$ *proj2-rep p* $\cdot$ (*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
$\ldots \longleftrightarrow$ (*proj2-rep p v$*$ cltn2-rep* (*cltn2-inverse A*)) $\cdot$ *proj2-line-rep l* $=$ *0*
by *simp*
also from ⟨*j* $\neq$ *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 $\ldots \longleftrightarrow$ *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]

```
lemma statement52-existence:
  fixes a :: proj2^3 and a3 :: proj2
  assumes proj2-no-3-Col (insert a3 (range (op $ a)))
  shows ∃ A. apply-cltn2 (proj2-abs (vector [1,1,1])) A = a3 ∧
  (∀ j. apply-cltn2 (proj2-abs (basis j)) A = a$j)
proof -
  let ?v = proj2-rep a3
  let ?B = proj2-rep ` range (op $ a)
```

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

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

from card-suc-ge-insert [of a3 range (op $ a)]
  and ‹card (insert a3 (range (op $ a))) = 4›
  and ‹card (range (op $ a)) ≤ 3›
have card (range (op $ a)) = 3 by simp
with card-image [of proj2-rep range (op $ 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 (∑ w ∈ ?B. (c w) *R w) = ?v by (auto simp add: scalar-equiv)
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 (op $ 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 (op $ a) − {a$i})

  have a$i ∈ insert a3 (range (op $ 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$i)} by simp
  with setsum-diff1 [of ?B λ w. (c w) *R w]
    and ‹finite ?B›
    and ‹proj2-rep (a$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 (op $ a)› have a3 ≠ a$i by auto
  hence insert a3 (range (op $ a)) − {a$i} =
    insert a3 (range (op $ a) − {a$i}) by auto
  hence proj2-rep ‘ (insert a3 (range (op $ a)) − {a$i}) = insert ?v ?Bi
    by simp
  moreover from ‹proj2-no-3-Col (insert a3 (range (op $ a)))›
    and ‹a$i ∈ insert a3 (range (op $ a))›
  have span (proj2-rep ‘ (insert a3 (range (op $ 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 card-ge-dim [of ?Bi] have dim ?Bi ≤ 2 by simp
  hence dim (span ?Bi) ≤ 2 by (subst dim-span)
  with dim-univ [where ′n = 3] have span ?Bi ≠ UNIV by auto
  with ‹span (insert ?v ?Bi) = UNIV› and in-span-eq
  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$i)›
      and ‹(∑ w ∈ ?B. (c w) *R w) = ?v›
    have ?v = (∑ w ∈ ?Bi. (c w) *R w)
      by simp
    with span-finite [of ?Bi] and ‹finite ?Bi›
    have ?v ∈ span ?Bi by (simp add: scalar-equiv) auto
    with ‹?v ∉ span ?Bi› have False .. }
  thus c (proj2-rep (a$i)) ≠ 0 ..
qed
hence ∀ w∈?B. c w ≠ 0
  unfolding image-def
  by auto

from Cart-nth-inverse
have rows ?C = (λ w. (c w) *R w) ‘ ?B
  unfolding rows-def
    and row-def
    and image-def
  by auto

have ∀ x. x ∈ span (rows ?C)
proof
  fix x :: real^3
  from ‹finite ?B› and span-finite [of ?B] and ‹span ?B = UNIV›
```

```
  obtain ub where (∑ w∈?B. (ub w) *_R w) = x by (auto simp add: scalar-equiv)
  have ∀ w∈?B. (ub w) *_R w ∈ span (rows ?C)
  proof
    fix w
    assume w ∈ ?B
    with span-inc [of rows ?C] and ‹rows ?C = image (λ w. (c w) *_R w) ?B›
    have (c w) *_R w ∈ 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) ∈ span (rows ?C)
      by (simp add: scalar-equiv)
    with ‹∀ w∈?B. c w ≠ 0› and ‹w ∈ ?B›
    show (ub w) *_R w ∈ span (rows ?C) by auto
  qed
  with span-setsum [of ?B λ w. (ub w) *_R w] and ‹finite ?B›
  have (∑ w∈?B. (ub w) *_R w) ∈ span (rows ?C) by simp
  with ‹(∑ w∈?B. (ub w) *_R w) = x› show x ∈ span (rows ?C) by simp
qed
hence span (rows ?C) = UNIV by auto
with matrix-left-invertible-span-rows [of ?C]
have ∃ C'. C' ** ?C = mat 1 ..
with left-invertible-iff-invertible
have invertible ?C ..

have (vector [1,1,1] :: real^3) ≠ 0
  unfolding vector-def
  by (simp add: Cart-eq 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 op $ a] and ‹card (range (op $ a)) = 3›
have inj (op $ a) by simp
from exhaust-3 have ∀ i::3. (vector [1::real,1,1])$i = 1
  unfolding vector-def
  by auto
with vector-matrix-row [of vector [1,1,1] ?C]
have (vector [1,1,1]) v* ?C =
  (∑ i∈UNIV. (c (proj2-rep (a$i))) *_R (proj2-rep (a$i)))
  by simp
also from setsum-reindex
[of op $ a UNIV λ x. (c (proj2-rep x)) *_R (proj2-rep x)]
  and ‹inj (op $ a)›
have ... = (∑ x∈(range (op $ a)). (c (proj2-rep x)) *_R (proj2-rep x))
  by simp
also from setsum-reindex
[of proj2-rep range (op $ a) λ w. (c w) *_R w]
  and proj2-rep-inj and subset-inj-on [of proj2-rep UNIV range (op $ 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 (basis j)) ?A = a$j
  proof
    fix j
    have ((basis j)::real^3) ≠ 0 by (simp add: Cart-eq)
    with apply-cltn2-abs and ‹invertible ?C›
    have apply-cltn2 (proj2-abs (basis j)) ?A = proj2-abs (basis j v* ?C)
      by simp

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

lemma statement53-existence:
  fixes p :: proj2^4^2
  assumes ∀ i. proj2-no-3-Col (range (op $ (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$3
    ∧ (∀ j′. apply-cltn2 (proj2-abs (basis j′)) D = ?q$i$j′)
  have ∀ i. apply-cltn2 (proj2-abs (vector [1,1,1])) (?D$i) = p$i$3
    ∧ (∀ j′. apply-cltn2 (proj2-abs (basis j′)) (?D$i) = ?q$i$j′)
  proof
    fix i
    have range (op $ (p$i)) = insert (p$i$3) (range (op $ (?q$i)))
    proof
      show range (op $ (p$i)) ⊇ insert (p$i$3) (range (op $ (?q$i))) by auto
      show range (op $ (p$i)) ⊆ insert (p$i$3) (range (op $ (?q$i)))
```

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

and $p\$1\$j = apply\text{-}cltn2\ (proj2\text{-}abs\ (basis\ j'))\ (?D\$1)$
by *simp-all*
from ⟨$p\$0\$j = apply\text{-}cltn2\ (proj2\text{-}abs\ (basis\ j'))\ (?D\$0)$⟩
and *cltn2.act-inv-iff*
have $apply\text{-}cltn2\ (p\$0\$j)\ (cltn2\text{-}inverse\ (?D\$0)) = proj2\text{-}abs\ (basis\ j')$
by *simp*
with ⟨$p\$1\$j = apply\text{-}cltn2\ (proj2\text{-}abs\ (basis\ j'))\ (?D\$1)$⟩
and *cltn2.act-act* [*of cltn2-inverse* (*?D$0*) *?D$1 p0j*]
show $apply\text{-}cltn2\ (p\$0\$j)\ ?C = p\$1\$j$ **by** *simp*
qed
qed
thus $\exists\ C.\ \forall\ j.\ apply\text{-}cltn2\ (p\$0\$j)\ C = p\$1\$j$ **by** (*rule exI* [*of - ?C*])
qed

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

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

lemma *apply-cltn2-imp-mult*:
assumes $apply\text{-}cltn2\ p\ C = q$
shows $\exists\ k.\ k \neq 0 \land proj2\text{-}rep\ p\ v{*}\ cltn2\text{-}rep\ C = k *_R proj2\text{-}rep\ q$
proof −
have $proj2\text{-}rep\ p\ v{*}\ cltn2\text{-}rep\ C \neq 0$ **by** (*rule rep-mult-rep-non-zero*)

from ⟨$apply\text{-}cltn2\ p\ C = q$⟩
have $proj2\text{-}abs\ (proj2\text{-}rep\ p\ v{*}\ cltn2\text{-}rep\ C) = q$ **by** (*unfold apply-cltn2-def*)
hence $proj2\text{-}rep\ (proj2\text{-}abs\ (proj2\text{-}rep\ p\ v{*}\ cltn2\text{-}rep\ C)) = proj2\text{-}rep\ q$
by *simp*
with ⟨$proj2\text{-}rep\ p\ v{*}\ cltn2\text{-}rep\ C \neq 0$⟩ **and** *proj2-rep-abs2* [*of proj2-rep p v∗ cltn2-rep C*]
have $\exists\ j.\ j \neq 0 \land proj2\text{-}rep\ q = j *_R (proj2\text{-}rep\ p\ v{*}\ cltn2\text{-}rep\ C)$ **by** *simp*
then obtain j **where** $j \neq 0$
and $proj2\text{-}rep\ q = j *_R (proj2\text{-}rep\ p\ v{*}\ cltn2\text{-}rep\ C)$ **by** *auto*
hence $proj2\text{-}rep\ p\ v{*}\ cltn2\text{-}rep\ C = (1/j) *_R proj2\text{-}rep\ q$

by (*simp add*: *field-simps*)
with ⟨$j \neq 0$⟩
show $\exists\ k.\ k \neq 0 \wedge$ *proj2-rep p v∗ cltn2-rep C* $=$ *k* $*_R$ *proj2-rep q*
by (*simp add*: *exI* [*of* - $1/j$])
qed

lemma *statement55*:
assumes $p \neq q$
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 $r \neq p$

from ⟨*apply-cltn2 p C* $= q$⟩ **and** *apply-cltn2-imp-mult* [*of p C q*]
obtain *i* **where** $i \neq 0$ **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** $j \neq 0$ **and** *proj2-rep q v∗ cltn2-rep C* $=$ *j* $*_R$ *proj2-rep p*
by *auto*

from ⟨$p \neq q$⟩
and ⟨*proj2-incident p l*⟩
and ⟨*proj2-incident q l*⟩
and ⟨*proj2-incident r l*⟩
and *proj2-incident-iff*
have $r = p \vee (\exists\ k.$ *r* $=$ *proj2-abs* (*k* $*_R$ *proj2-rep p* $+$ *proj2-rep q*))
by *fast*
with ⟨$r \neq p$⟩
obtain *k* **where** *r* $=$ *proj2-abs* (*k* $*_R$ *proj2-rep p* $+$ *proj2-rep q*) **by** *auto*

from ⟨$p \neq q$⟩ **and** *proj2-rep-dependent* [*of k p 1 q*]
have *k* $*_R$ *proj2-rep p* $+$ *proj2-rep q* $\neq 0$ **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 $\mathit{proj2\text{-}rep}\ ?r = k *_R ((i/j) *_R \mathit{proj2\text{-}rep}\ p + \mathit{proj2\text{-}rep}\ q)$
by *auto*
hence $(k*i/j) *_R \mathit{proj2\text{-}rep}\ p + k *_R \mathit{proj2\text{-}rep}\ q - \mathit{proj2\text{-}rep}\ ?r = 0$
by (*simp add*: *scaleR-right-distrib*)
hence $\exists\ l.\ (k*i/j) *_R \mathit{proj2\text{-}rep}\ p + k *_R \mathit{proj2\text{-}rep}\ q + l *_R \mathit{proj2\text{-}rep}\ ?r = 0$
$\wedge\ (k*i/j \neq 0 \vee k \neq 0 \vee l \neq 0)$
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 ⟨$(k*i/j) *_R \mathit{proj2\text{-}rep}\ p + k *_R \mathit{proj2\text{-}rep}\ q - \mathit{proj2\text{-}rep}\ ?r = 0$⟩
have $(k*i/j - 1) *_R \mathit{proj2\text{-}rep}\ p + k *_R \mathit{proj2\text{-}rep}\ q = 0$
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 $?r = \mathit{proj2\text{-}abs}\ (\mathit{proj2\text{-}Col\text{-}coeff}\ p\ q\ ?r *_R \mathit{proj2\text{-}rep}\ p + \mathit{proj2\text{-}rep}\ q)$
by (*rule proj2-Col-coeff*)
with ⟨$p \neq q$⟩ **and** ⟨$?r = \mathit{proj2\text{-}abs}\ ((i/j) *_R \mathit{proj2\text{-}rep}\ p + \mathit{proj2\text{-}rep}\ q)$⟩
and *proj2-Col-coeff-unique*
show $\mathit{proj2\text{-}Col\text{-}coeff}\ p\ q\ ?r = i/j$ **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 $r = \mathit{proj2\text{-}abs}\ (\mathit{proj2\text{-}Col\text{-}coeff}\ p\ q\ r *_R \mathit{proj2\text{-}rep}\ p + \mathit{proj2\text{-}rep}\ q)$
(**is** $r = \mathit{proj2\text{-}abs}\ (?i *_R ?u + ?v)$)
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 $r = \mathit{proj2\text{-}abs}\ (?i *_R ?u + ?v)$ **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** $\mathit{proj2\text{-}abs}\ u \neq \mathit{proj2\text{-}abs}\ v$
and $j \neq 0$ **and** $l \neq 0$
shows *cross-ratio* (*proj2-abs u*) (*proj2-abs v*)
$(\mathit{proj2\text{-}abs}\ (i *_R u + j *_R v))$
$(\mathit{proj2\text{-}abs}\ (k *_R u + l *_R v))$
$= j * k\ /\ (i * l)$
(**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* ∈ {*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 $?pC \neq ?qC$ **and** $?rC \neq ?pC$ **and** $?sC \neq ?pC$ **and** $?rC \neq ?qC$ **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 $?uC \neq 0$ **and** $?vC \neq 0$ **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** $?pC \neq ?qC$ **by** *fast*

from ⟨$p \neq q$⟩ **and** *proj2-rep-dependent* [*of - p 1 q*]
have $?i *_R ?u + ?v \neq 0$ **and** $?j *_R ?u + ?v \neq 0$ **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* ≠ *q* **and** *r* ≠ *p* **and** *r* ≠ *q* **and** *s* ≠ *p* **and** *t* ≠ *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* ≠ *q*⟩ **and** ⟨*r* ≠ *p*⟩ **and** ⟨*s* ≠ *p*⟩ **and** ⟨*t* ≠ *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* ≠ *q*⟩ **and** ⟨*r* = *proj2-abs* (*?i* $*_R$ *?u* + *?v*)⟩
  have *?i* ≠ *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* ≠ *q* **and** *r* ≠ *p* **and** *r* ≠ *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* ≠ *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** $p \neq q$ **and** $r \neq p$ **and** $r \neq q$
shows *proj2-Col-coeff p q r* $\neq 0$
proof
assume *proj2-Col-coeff p q r* $= 0$

from ⟨*proj2-Col p q r*⟩ **and** ⟨$p \neq q$⟩ **and** ⟨$r \neq p$⟩
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** $r \neq s$
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** $p \neq q$ **and** $r \neq p$ **and** $s \neq p$ **and** $r \neq q$
by (*unfold cross-ratio-correct-def*, *simp-all*)

have *proj2-rep p* $\neq 0$ (**is** $?u \neq 0$) **and** *proj2-rep q* $\neq 0$ (**is** $?v \neq 0$)
by (*rule proj2-rep-non-zero*)+

have $p =$ *proj2-abs ?u* **and** $q =$ *proj2-abs ?v*
by (*simp-all add*: *proj2-abs-rep*)
with ⟨$p \neq q$⟩ **have** *proj2-abs ?u* $\neq$ *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** ⟨$p \neq q$⟩ **and** ⟨$r \neq p$⟩ **and** ⟨$s \neq p$⟩
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 ⟨$r \neq s$⟩ **have** $?i \neq ?j$ **by** *auto*

from ⟨$?u \neq 0$⟩ **and** ⟨$?v \neq 0$⟩ **and** ⟨*proj2-abs ?u* $\neq$ *proj2-abs ?v*⟩
and *dependent-proj2-abs* [*of ?u ?v - 1*]
have $?w \neq 0$ **and** $?x \neq 0$ **by** *auto*

from ⟨$r =$ *proj2-abs* ($?i *_R ?u + ?v$)⟩ **and** ⟨$r \neq q$⟩
have $?i \neq 0$ **by** (*auto simp add*: *proj2-abs-rep*)

have $?w - ?x = (?i - ?j) *_R ?u$ **by** (*simp add*: *algebra-simps*)
with ⟨$?i \neq ?j$⟩
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 ⟨$?i \neq ?j$⟩
have $q =$ *proj2-abs* ($?j *_R ?w - ?i *_R ?x$) **by** (*simp add*: *proj2-abs-mult-rep*)
with ⟨$?w \neq 0$⟩ **and** ⟨$?x \neq 0$⟩ **and** ⟨$r \neq s$⟩ **and** ⟨$?i \neq 0$⟩ **and** ⟨$r =$ *proj2-abs ?w*⟩
and ⟨$s =$ *proj2-abs ?x*⟩ **and** ⟨$p =$ *proj2-abs* ($?w - ?x$)⟩
and *cross-ratio-abs* [*of ?w ?x* -1 $-?i$ *1* $?j$]
have *cross-ratio r s p q* $= ?j$ / $?i$ **by** (*simp add*: *algebra-simps*)
thus *cross-ratio r s p q* $=$ *cross-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-ratio q p r s* = *1* / (*cross-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* ≠ *q* **and** *r* ≠ *p* **and** *r* ≠ *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* ≠ *q*⟩ **and** ⟨*r* ≠ *p*⟩ **and** ⟨*r* ≠ *q*⟩ **and** ⟨*r* = *s*⟩
have *cross-ratio p q r s* = *1* **and** *cross-ratio q p r s* = *1*
by (*simp-all add*: *cross-ratio-equal-1*)
thus *cross-ratio q p r s* = *1* / (*cross-ratio p q r s*) **by** *simp*
next
assume *r* ≠ *s*
with ⟨*cross-ratio-correct q p r s*⟩
have *cross-ratio q p r s* = *cross-ratio r s q p*
by (*simp add*: *cross-ratio-swap-13-24*)
also have ... = *1* / (*cross-ratio r s p q*) **by** (*rule cross-ratio-swap-34*)
also from ⟨*cross-ratio-correct p q r s*⟩ **and** ⟨*r* ≠ *s*⟩
have ... = *1* / (*cross-ratio p q r s*) **by** (*simp add*: *cross-ratio-swap-13-24*)
finally show *cross-ratio q p r s* = *1* / (*cross-ratio p q r s*) **.**
qed

7.6 Cartesian subspace of the real projective plane

definition *vector2-append1* :: *real*^*2* ⇒ *real*^*3* **where**
vector2-append1 v = *vector* [*v*$*1*, *v*$*2*, *1*]

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

definition *proj2-pt* :: *real*^*2* ⇒ *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 $\triangleq$ (*proj2-rep p*)\$3 $\neq$ *0*

definition *cart2-pt* :: *proj2* $\Rightarrow$ *real*^2 **where**
cart2-pt p $\triangleq$
vector [(*proj2-rep p*)\$1 / (*proj2-rep p*)\$3, (*proj2-rep p*)\$2 / (*proj2-rep p*)\$3]

definition *cart2-append1* :: *proj2* $\Rightarrow$ *real*^3 **where**
cart2-append1 p $\triangleq$ (*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* $\neq$ *0*
proof −
from ⟨*z-non-zero p*⟩ **have** (*cart2-append1 p*)\$3 = *1* **by** (*rule cart2-append1-z*)
thus *cart2-append1 p* $\neq$ *0* **by** (*simp add*: *Cart-eq 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*: *Cart-eq 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*: *Cart-eq 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: Cart-eq 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 ≠ 0 and ?a″ = i *R ?a′
    and j ≠ 0 and ?b″ = j *R ?b′
    and k ≠ 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'≠0 ∨ j'≠0 ∨ k'≠0
    unfolding proj2-Col-def
    by auto

  let ?i'' = i * i'
  let ?j'' = j * j'
  let ?k'' = k * k'
  from ‹i≠0› and ‹j≠0› and ‹k≠0› and ‹i'≠0 ∨ j'≠0 ∨ k'≠0›
  have ?i''≠0 ∨ ?j''≠0 ∨ ?k''≠0 by simp

  from ‹i' *R ?a'' + j' *R ?b'' + k' *R ?c'' = 0›
    and ‹?a'' = i *R ?a'›
    and ‹?b'' = j *R ?b'›
    and ‹?c'' = k *R ?c'›
  have ?i'' *R ?a' + ?j'' *R ?b' + ?k'' *R ?c' = 0
    by (simp add: mult-ac)
  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 ‹?i'' *R ?a' + ?j'' *R ?b' + ?k'' *R ?c' = 0›
  have ?i'' *R a + ?j'' *R b + ?k'' *R c = 0
    by (simp add: Cart-eq forall-2)

  have dep2 (b − a) (c − a)
  proof cases
    assume ?k'' = 0
    with ‹?i'' + ?j'' + ?k'' = 0› have ?j'' = −?i'' by simp
    with ‹?i''≠0 ∨ ?j''≠0 ∨ ?k''≠0› and ‹?k'' = 0› have ?i'' ≠ 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'' ≠ 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 ∃ x t s. b − a = t *R x ∧ c − a = s *R x
      by blast
    thus dep2 (b − a) (c − a) unfolding dep2-def .
```

```
  next
    assume ?k'' ≠ 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'' ≠ 0› have c − a = (−?j'' / ?k'') *R (b − a) by simp
    moreover have b − a = 1 *R (b − a) by simp
    ultimately have ∃ x t s. b − a = t *R x ∧ 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 ≠ 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: Cart-eq)
    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: Cart-eq 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 $\exists$ *l. proj2-incident p l* $\wedge$ *proj2-incident q l* $\wedge$ *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 $\exists$ *l. proj2-incident p l* $\wedge$ *proj2-incident q l* $\wedge$ *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*)
$\longleftrightarrow (\exists\ k \geq 0.\ k \leq 1$
$\wedge$ *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 $\forall$ *k. ?cq* $-$ *?cp* $= k *_R$ (*?cr* $-$ *?cp*) $\longleftrightarrow$ *?cq* $= k *_R$ *?cr* $+ (1 - k) *_R$ *?cp*
by (*simp add*: *algebra-simps*)
hence $\forall$ *k. ?cq* $-$ *?cp* $= k *_R$ (*?cr* $-$ *?cp*)
$\longleftrightarrow$ *?cq1* $= k *_R$ *?cr1* $+ (1 - k) *_R$ *?cp1*
unfolding *vector2-append1-def* **and** *vector-def*
by (*simp add*: *Cart-eq forall-2 forall-3*)
with ⟨*?cp1* = *cart2-append1 p*⟩
and ⟨*?cq1* = *cart2-append1 q*⟩
and ⟨*?cr1* = *cart2-append1 r*⟩
have $\forall$ *k. ?cq* $-$ *?cp* $= k *_R$ (*?cr* $-$ *?cp*)
$\longleftrightarrow$ *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*)
$\longleftrightarrow (\exists\ k \geq 0.\ k \leq 1$
$\wedge$ *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** $q \neq r$
shows $\exists\ k \geq 0.\ k < 1$
$\wedge$ *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** $k \geq 0$ **and** $k \leq 1$
and *cart2-append1 q* $=$ $k *_R$ *cart2-append1 r* $+$ $(1 - k) *_R$ *cart2-append1 p*
by *auto*

have $k \neq 1$
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 ⟨$q \neq r$⟩ **show** *False* **..**
qed
with ⟨$k \leq 1$⟩ **have** $k < 1$ **by** *simp*
with ⟨$k \geq 0$⟩
and ⟨*cart2-append1 q* $=$ $k *_R$ *cart2-append1 r* $+$ $(1 - k) *_R$ *cart2-append1 p*⟩
show $\exists\ k \geq 0.\ k < 1$
$\wedge$ *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** $q \neq p$ **and** $q \neq r$
shows $\exists\ k > 0.\ k < 1$
$\wedge$ *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** ⟨$q \neq r$⟩
and *cart2-append1-between-right-strict* [*of p q r*]
obtain k **where** $k \geq 0$ **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-append1 q* $=$ $k *_R$ *cart2-append1 r* $+$ $(1 - k) *_R$ *cart2-append1 p*⟩
have *cart2-append1 q* $=$ *cart2-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-append1* $q = k *_R$ *cart2-append1* $r + (1 - k) *_R$ *cart2-append1* p⟩
show $\exists\ k{>}0.\ k < 1$
$\wedge$ *cart2-append1* $q = k *_R$ *cart2-append1* $r + (1 - k) *_R$ *cart2-append1* p
by (*simp add*: *exI* [*of* - *k*])
qed

end

8 Roots of real quadratics

theory *Quadratic-Discriminant*
imports *Miscellany*
begin

definition *discrim* :: [*real*,*real*,*real*] $\Rightarrow$ *real* **where**
discrim a b c $\triangleq b^2 - 4 * a * c$

lemma *complete-square*:
fixes *a b c x* :: *real*
assumes $a \neq 0$
shows $a * x^2 + b * x + c = 0 \longleftrightarrow (2 * a * x + b)^2 =$ *discrim a b c*
proof $-$
have $4 * a^2 * x^2 + 4 * a * b * x + 4 * a * c = 4 * a * (a * x^2 + b * x + c)$
by (*simp add*: *algebra-simps square-expand*)
with ⟨$a \neq 0$⟩
have $a * x^2 + b * x + c = 0 \longleftrightarrow 4 * a^2 * x^2 + 4 * a * b * x + 4 * a * c = 0$
by *simp*
thus $a * x^2 + b * x + c = 0 \longleftrightarrow (2 * a * x + b)^2 =$ *discrim a b c*
unfolding *discrim-def*
by (*simp add*: *square-expand algebra-simps*)
qed

lemma *discriminant-negative*:
fixes *a b c x* :: *real*
assumes $a \neq 0$
and *discrim a b c* < 0
shows $a * x^2 + b * x + c \neq 0$
proof $-$
have $(2 * a * x + b)^2 \geq 0$ **by** *simp*
with ⟨*discrim a b c* < 0⟩ **have** $(2 * a * x + b)^2 \neq$ *discrim a b c* **by** *arith*
with *complete-square* **and** ⟨$a \neq 0$⟩ **show** $a * x^2 + b * x + c \neq 0$ **by** *simp*
qed

lemma *plus-or-minus-sqrt*:
fixes *x y* :: *real*
assumes $y \geq 0$

shows $x^2 = y \longleftrightarrow x = \mathit{sqrt}\ y \vee x = -\ \mathit{sqrt}\ y$
proof
assume $x^2 = y$
hence $\mathit{sqrt}\ (x^2) = \mathit{sqrt}\ y$ **by** *simp*
hence $\mathit{sqrt}\ y = |x|$ **by** *simp*
thus $x = \mathit{sqrt}\ y \vee x = -\ \mathit{sqrt}\ y$ **by** *auto*
next
assume $x = \mathit{sqrt}\ y \vee x = -\ \mathit{sqrt}\ y$
hence $x^2 = (\mathit{sqrt}\ y)^2 \vee x^2 = (-\ \mathit{sqrt}\ y)^2$ **by** *auto*
with $\langle y \geq 0 \rangle$ **show** $x^2 = y$ **by** *simp*
qed

lemma *divide-non-zero*:
fixes $x\ y\ z :: \mathit{real}$
assumes $x \neq 0$
shows $x * y = z \longleftrightarrow y = z\ /\ x$
proof
assume $x * y = z$
with $\langle x \neq 0 \rangle$ **show** $y = z\ /\ x$ **by** (*simp add*: *field-simps*)
next
assume $y = z\ /\ x$
with $\langle x \neq 0 \rangle$ **show** $x * y = z$ **by** *simp*
qed

lemma *discriminant-nonneg*:
fixes $a\ b\ c\ x :: \mathit{real}$
assumes $a \neq 0$
and $\mathit{discrim}\ a\ b\ c \geq 0$
shows $a * x^2 + b * x + c = 0 \longleftrightarrow$
$x = (-b + \mathit{sqrt}\ (\mathit{discrim}\ a\ b\ c))\ /\ (2 * a) \vee$
$x = (-b - \mathit{sqrt}\ (\mathit{discrim}\ a\ b\ c))\ /\ (2 * a)$
proof $-$
from *complete-square* **and** *plus-or-minus-sqrt* **and** *assms*
have $a * x^2 + b * x + c = 0 \longleftrightarrow$
$(2 * a) * x + b = \mathit{sqrt}\ (\mathit{discrim}\ a\ b\ c) \vee$
$(2 * a) * x + b = -\ \mathit{sqrt}\ (\mathit{discrim}\ a\ b\ c)$
by *simp*
also have $\ldots \longleftrightarrow (2 * a) * x = (-b + \mathit{sqrt}\ (\mathit{discrim}\ a\ b\ c)) \vee$
$(2 * a) * x = (-b - \mathit{sqrt}\ (\mathit{discrim}\ a\ b\ c))$
by *auto*
also from $\langle a \neq 0 \rangle$ **and** *divide-non-zero* [*of* $2 * a\ x$]
have $\ldots \longleftrightarrow x = (-b + \mathit{sqrt}\ (\mathit{discrim}\ a\ b\ c))\ /\ (2 * a) \vee$
$x = (-b - \mathit{sqrt}\ (\mathit{discrim}\ a\ b\ c))\ /\ (2 * a)$
by *simp*
finally show $a * x^2 + b * x + c = 0 \longleftrightarrow$
$x = (-b + \mathit{sqrt}\ (\mathit{discrim}\ a\ b\ c))\ /\ (2 * a) \vee$
$x = (-b - \mathit{sqrt}\ (\mathit{discrim}\ a\ b\ c))\ /\ (2 * a)$ **.**
qed

lemma *discriminant-zero*:
fixes $a\ b\ c\ x :: real$
assumes $a \neq 0$
and $discrim\ a\ b\ c = 0$
shows $a * x^2 + b * x + c = 0 \longleftrightarrow x = -b\ /\ (2 * a)$
using *discriminant-nonneg* **and** *assms*
by *simp*

theorem *discriminant-iff*:
fixes $a\ b\ c\ x :: real$
assumes $a \neq 0$
shows $a * x^2 + b * x + c = 0 \longleftrightarrow$
$discrim\ a\ b\ c \geq 0\ \wedge$
$(x = (-b + sqrt\ (discrim\ a\ b\ c))\ /\ (2 * a)\ \vee$
$x = (-b - sqrt\ (discrim\ a\ b\ c))\ /\ (2 * a))$
proof
assume $a * x^2 + b * x + c = 0$
with *discriminant-negative* **and** $\langle a \neq 0 \rangle$ **have** $\neg(discrim\ a\ b\ c < 0)$ **by** *auto*
hence $discrim\ a\ b\ c \geq 0$ **by** *simp*
with *discriminant-nonneg* **and** $\langle a * x^2 + b * x + c = 0 \rangle$ **and** $\langle a \neq 0 \rangle$
have $x = (-b + sqrt\ (discrim\ a\ b\ c))\ /\ (2 * a)\ \vee$
$x = (-b - sqrt\ (discrim\ a\ b\ c))\ /\ (2 * a)$
by *simp*
with $\langle discrim\ a\ b\ c \geq 0 \rangle$
show $discrim\ a\ b\ c \geq 0\ \wedge$
$(x = (-b + sqrt\ (discrim\ a\ b\ c))\ /\ (2 * a)\ \vee$
$x = (-b - sqrt\ (discrim\ a\ b\ c))\ /\ (2 * a))$ **..**
next
assume $discrim\ a\ b\ c \geq 0\ \wedge$
$(x = (-b + sqrt\ (discrim\ a\ b\ c))\ /\ (2 * a)\ \vee$
$x = (-b - sqrt\ (discrim\ a\ b\ c))\ /\ (2 * a))$
hence $discrim\ a\ b\ c \geq 0$ **and**
$x = (-b + sqrt\ (discrim\ a\ b\ c))\ /\ (2 * a)\ \vee$
$x = (-b - sqrt\ (discrim\ a\ b\ c))\ /\ (2 * a)$
by *simp-all*
with *discriminant-nonneg* **and** $\langle a \neq 0 \rangle$ **show** $a * x^2 + b * x + c = 0$ **by** *simp*
qed

lemma *discriminant-nonneg-ex*:
fixes $a\ b\ c :: real$
assumes $a \neq 0$
and $discrim\ a\ b\ c \geq 0$
shows $\exists\ x.\ a * x^2 + b * x + c = 0$
using *discriminant-nonneg* **and** *assms*
by *auto*

lemma *discriminant-pos-ex*:
fixes $a\ b\ c :: real$
assumes $a \neq 0$

```
  and discrim a b c > 0
  shows ∃ x y. x ≠ y ∧ a * x^2 + b * x + c = 0 ∧ a * y^2 + b * y + c = 0
proof −
  let ?x = (−b + sqrt (discrim a b c)) / (2 * a)
  let ?y = (−b − sqrt (discrim a b c)) / (2 * a)
  from ‹discrim a b c > 0› have sqrt (discrim a b c) ≠ 0 by simp
  hence sqrt (discrim a b c) ≠ − sqrt (discrim a b c) by arith
  with ‹a ≠ 0› have ?x ≠ ?y by simp
  moreover
  from discriminant-nonneg [of a b c ?x]
    and discriminant-nonneg [of a b c ?y]
    and assms
  have a * ?x^2 + b * ?x + c = 0 and a * ?y^2 + b * ?y + c = 0 by simp-all
  ultimately
  show ∃ x y. x ≠ y ∧ a * x^2 + b * x + c = 0 ∧ a * y^2 + b * y + c = 0 by blast
qed

lemma discriminant-pos-distinct:
  fixes a b c x :: real
  assumes a ≠ 0 and discrim a b c > 0
  shows ∃ y. x ≠ y ∧ a * y^2 + b * y + c = 0
proof −
  from discriminant-pos-ex and ‹a ≠ 0› and ‹discrim a b c > 0›
  obtain w and z where w ≠ z
    and a * w^2 + b * w + c = 0 and a * z^2 + b * z + c = 0
    by blast
  show ∃ y. x ≠ y ∧ a * y^2 + b * y + c = 0
  proof cases
    assume x = w
    with ‹w ≠ z› have x ≠ z by simp
    with ‹a * z^2 + b * z + c = 0›
    show ∃ y. x ≠ y ∧ a * y^2 + b * y + c = 0 by auto
  next
    assume x ≠ w
    with ‹a * w^2 + b * w + c = 0›
    show ∃ y. x ≠ y ∧ a * y^2 + b * y + c = 0 by auto
  qed
qed

end
```

9 The hyperbolic plane and Tarski's axioms

```
theory Hyperbolic-Tarski
imports Euclid-Tarski
  Projective
  Quadratic-Discriminant
```

begin

9.1 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*: *Cart-eq 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*: *setsum-3 Cart-eq forall-3*)

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

definition *polar* :: *proj2* $\Rightarrow$ *proj2-line* **where**
polar p $\triangleq$ *proj2-line-abs* (*M* $*v$ *proj2-rep p*)

definition *pole* :: *proj2-line* $\Rightarrow$ *proj2* **where**
pole l $\triangleq$ *proj2-abs* (*M* $*v$ *proj2-line-rep l*)

lemma *polar-abs*:
assumes $v \neq 0$
shows *polar* (*proj2-abs v*) = *proj2-line-abs* (*M* $*v$ *v*)
proof $-$
from ‹$v \neq 0$› **and** *proj2-rep-abs2*
obtain *k* **where** $k \neq 0$ **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-scalar-vector-ac scalar-matrix-vector-assoc*)
with ‹$k \neq 0$› **and** *proj2-line-abs-mult*
show *polar* (*proj2-abs v*) = *proj2-line-abs* (*M* $*v$ *v*) **by** *simp*
qed

lemma *pole-abs*:
assumes $v \neq 0$
shows *pole* (*proj2-line-abs v*) = *proj2-abs* (*M* $*v$ *v*)
proof $-$
from ‹$v \neq 0$› **and** *proj2-line-rep-abs*
obtain *k* **where** $k \neq 0$ **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-scalar-vector-ac scalar-matrix-vector-assoc*)
with ⟨$k \neq 0$⟩ **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* $p \neq 0$
proof −
have *proj2-rep* $p \neq 0$ **by** (*rule proj2-rep-non-zero*)
with *M-invertible*
show $M *v$ *proj2-rep* $p \neq 0$ **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 matrix-vector-mul-lid*)
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**
  $conic\text{-}sgn\ p \triangleq sgn\ (proj2\text{-}rep\ p \cdot (M *v\ proj2\text{-}rep\ p))$

lemma *conic-sgn-abs*:
  assumes $v \neq 0$
  shows $conic\text{-}sgn\ (proj2\text{-}abs\ v) = sgn\ (v \cdot (M *v\ v))$
proof −
  from ⟨$v \neq 0$⟩ **and** *proj2-rep-abs2*
  obtain *j* **where** $j \neq 0$ **and** $proj2\text{-}rep\ (proj2\text{-}abs\ v) = j *_R v$ **by** *auto*
  from ⟨$j \neq 0$⟩ **have** $j^2 > 0$ **by** *simp*

  from ⟨$proj2\text{-}rep\ (proj2\text{-}abs\ v) = j *_R v$⟩
  have $conic\text{-}sgn\ (proj2\text{-}abs\ v) = sgn\ (j^2 * (v \cdot (M *v\ v)))$
    unfolding *conic-sgn-def*
    by (*simp add*:
      matrix-scalar-vector-ac
      scalar-matrix-vector-assoc [*symmetric*]
      dot-scaleR-mult
      square-expand
      algebra-simps)
  also have $\ldots = sgn\ (j^2) * sgn\ (v \cdot (M *v\ v))$ **by** (*rule sgn-times*)
  also from ⟨$j^2 > 0$⟩ **have** $\ldots = sgn\ (v \cdot (M *v\ v))$ **by** *simp*
  finally show $conic\text{-}sgn\ (proj2\text{-}abs\ v) = sgn\ (v \cdot (M *v\ v))$ **.**
qed

lemma *sgn-conic-sgn*: $sgn\ (conic\text{-}sgn\ p) = conic\text{-}sgn\ p$
  by (*unfold conic-sgn-def*) *simp*

definition *S* :: *proj2 set* **where**
  $S \triangleq \{p.\ conic\text{-}sgn\ p = 0\}$

definition *K2* :: *proj2 set* **where**
  $K2 \triangleq \{p.\ conic\text{-}sgn\ p < 0\}$

lemma *S-K2-empty*: $S \cap K2 = \{\}$
  unfolding *S-def* **and** *K2-def*
  by *auto*

lemma *K2-abs*:
  assumes $v \neq 0$
  shows $proj2\text{-}abs\ v \in K2 \longleftrightarrow v \cdot (M *v\ v) < 0$
proof −
  have $proj2\text{-}abs\ v \in K2 \longleftrightarrow conic\text{-}sgn\ (proj2\text{-}abs\ v) < 0$
    by (*simp add*: *K2-def*)

with ⟨$v \neq 0$⟩ **and** *conic-sgn-abs*
show *proj2-abs* $v \in K2 \longleftrightarrow v \cdot (M *v\ v) < 0$ **by** *simp*
qed

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

lemma *K2-centre-non-zero*: *vector* $[0,0,1] \neq (0 :: real\hat{}3)$
by (*unfold vector-def*) (*simp add*: *Cart-eq forall-3*)

lemma *K2-centre-in-K2*: *K2-centre* $\in$ *K2*
proof −
from *K2-centre-non-zero* **and** *proj2-rep-abs2*
obtain *k* **where** $k \neq 0$ **and** *proj2-rep K2-centre* = $k *_R$ *vector* [0,0,1]
by (*unfold K2-centre-def*) *auto*
from ⟨$k \neq 0$⟩ **have** $0 < k^2$ **by** *simp*
with ⟨*proj2-rep K2-centre* = $k *_R$ *vector* [0,0,1]⟩
show *K2-centre* $\in$ *K2*
unfolding *K2-def*
and *conic-sgn-def*
and *M-def*
and *matrix-vector-mult-def*
and *inner-vector-def*
and *vector-def*
by (*simp add*: *Cart-eq setsum-3 square-expand*)
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-vector-def setsum-3 square-expand*)

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-vector-def setsum-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 … $\longleftrightarrow$ *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*: $p \in S \longleftrightarrow$ *proj2-rep* $p \cdot (M *v$ *proj2-rep* $p) = 0$
proof −
 have *proj2-rep* $p \neq 0$ **by** (*rule proj2-rep-non-zero*)
 hence *proj2-abs* (*proj2-rep* p) $\in S \longleftrightarrow$ *proj2-rep* $p \cdot (M *v$ *proj2-rep* $p) = 0$
  by (*rule S-abs*)
 thus $p \in S \longleftrightarrow$ *proj2-rep* $p \cdot (M *v$ *proj2-rep* $p) = 0$
  by (*simp add*: *proj2-abs-rep*)
qed

lemma *incident-polar*:
 proj2-incident p (*polar* q) $\longleftrightarrow$ *proj2-rep* $p \cdot (M *v$ *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) $\longleftrightarrow p \in S$
 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* $\cdot$ (*M* $*v$ *proj2-rep q*) $= 0$ **by** (*unfold incident-polar*)
hence *proj2-rep q* $\cdot$ (*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 $\triangleq$ *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-vector-def* **and** *vector-def*
by (*simp add*: *setsum-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*) $\neq 0$
unfolding *vector-def*
by (*simp add*: *Cart-eq*)
with ‹*proj2-incident p z-zero*›
have *?v* $\cdot$ *vector* [*0,0,1*] $= 0$
unfolding *z-zero-def*
by (*simp add*: *proj2-incident-right-abs*)
hence *?v*\$*3* $= 0$
unfolding *inner-vector-def* **and** *vector-def*
by (*simp add*: *setsum-3*)
hence *?v* $\cdot$ (*M* $*v$ *?v*) = (*?v*\$*1*)^2 + (*?v*\$2)^2
unfolding *inner-vector-def*
and *square-expand*
and *matrix-vector-mult-def*

and *M-def*
and *vector-def*
and *setsum-3*
by *simp*

have $?v \neq 0$ **by** (*rule proj2-rep-non-zero*)
with ⟨$?v\$3 = 0$⟩ **have** $?v\$1 \neq 0 \lor ?v\$2 \neq 0$ **by** (*simp add*: *Cart-eq 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* $p \neq 1$
shows *z-non-zero p*
proof −
from ⟨*conic-sgn* $p \neq 1$⟩
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 $p \notin S$
proof −
from ⟨*proj2-incident p z-zero*⟩ **have** *conic-sgn* $p = 1$
by (*rule z-zero-conic-sgn-1*)
thus $p \notin S$
unfolding *S-def*
by *simp*
qed

lemma *line-incident-point-not-in-S*: $\exists\ p.\ p \notin S \land$ *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** $?p \notin S$ **by** (*rule z-zero-not-in-S*)
with ⟨*proj2-incident ?p l*⟩
show $\exists\ p.\ p \notin S \land$ *proj2-incident p l* **by** *auto*
qed

lemma *apply-cltn2-abs-abs-in-S*:
assumes $v \neq 0$ **and** *invertible J*

shows *apply-cltn2* (*proj2-abs* v) (*cltn2-abs* J) $\in S$
$\longleftrightarrow v \cdot (J ** M ** \mathit{transpose}\ J *v\ v) = 0$
proof −
from ⟨$v \neq 0$⟩ **and** ⟨*invertible J*⟩
have $v\ v*\ J \neq 0$ **by** (*rule non-zero-mult-invertible-non-zero*)

from ⟨$v \neq 0$⟩ **and** ⟨*invertible J*⟩
have *apply-cltn2* (*proj2-abs* v) (*cltn2-abs* J) = *proj2-abs* ($v\ v*\ J$)
by (*rule apply-cltn2-abs*)
also from ⟨$v\ v*\ J \neq 0$⟩
have $\ldots \in S \longleftrightarrow (v\ v*\ J) \cdot (M *v\ (v\ v*\ J)) = 0$ **by** (*rule S-abs*)
finally show *apply-cltn2* (*proj2-abs* v) (*cltn2-abs* J) $\in S$
$\longleftrightarrow v \cdot (J ** M ** \mathit{transpose}\ J *v\ v) = 0$
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) $\in S$
$\longleftrightarrow$ (*proj2-rep p*) $\cdot$ ($J ** M ** \mathit{transpose}\ J *v$ (*proj2-rep p*)) $= 0$
proof −
have *proj2-rep p* $\neq 0$ **by** (*rule proj2-rep-non-zero*)
with ⟨*invertible J*⟩
have *apply-cltn2* (*proj2-abs* (*proj2-rep p*)) (*cltn2-abs* J) $\in S$
$\longleftrightarrow$ *proj2-rep p* $\cdot$ ($J ** M ** \mathit{transpose}\ J *v$ *proj2-rep p*) $= 0$
by (*simp add*: *apply-cltn2-abs-abs-in-S*)
thus *apply-cltn2 p* (*cltn2-abs* J) $\in S$
$\longleftrightarrow$ *proj2-rep p* $\cdot$ ($J ** M ** \mathit{transpose}\ J *v$ *proj2-rep p*) $= 0$
by (*simp add*: *proj2-abs-rep*)
qed

lemma *apply-cltn2-abs-in-S*:
assumes $v \neq 0$
shows *apply-cltn2* (*proj2-abs* v) $C \in S$
$\longleftrightarrow v \cdot$ (*cltn2-rep* $C ** M **$ *transpose* (*cltn2-rep* C) $*v\ v) = 0$
proof −
have *invertible* (*cltn2-rep* C) **by** (*rule cltn2-rep-invertible*)
with ⟨$v \neq 0$⟩
have *apply-cltn2* (*proj2-abs* v) (*cltn2-abs* (*cltn2-rep* C)) $\in S$
$\longleftrightarrow v \cdot$ (*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) $C \in S$
$\longleftrightarrow v \cdot$ (*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 $C \in S$
$\longleftrightarrow$ *proj2-rep p* $\cdot$ (*cltn2-rep* $C ** M **$ *transpose* (*cltn2-rep* C) $*v$ *proj2-rep p*)

$= 0$
proof −
have *proj2-rep p* $\neq$ *0* **by** (*rule proj2-rep-non-zero*)
hence *apply-cltn2* (*proj2-abs* (*proj2-rep p*)) *C* $\in$ *S*
$\longleftrightarrow$ *proj2-rep p* $\cdot$ (*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* $\in$ *S*
$\longleftrightarrow$ *proj2-rep p* $\cdot$ (*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*) $\cdot$ (*M* $*v$ *vector2-append1 v*) = (*norm v*)$^2 - 1$
proof −
have (*norm v*)^2 = (*v*\$*1*)^2 + (*v*\$*2*)^2
unfolding *norm-vector-def*
and *setL2-def*
by (*simp add*: *setsum-2*)
thus (*vector2-append1 v*) $\cdot$ (*M* $*v$ *vector2-append1 v*) = (*norm v*)$^2 - 1$
unfolding *vector2-append1-def*
and *inner-vector-def*
and *matrix-vector-mult-def*
and *vector-def*
and *M-def*
and *power2-norm-eq-inner*
by (*simp add*: *setsum-3 square-expand*)
qed

9.2 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*] $\neq$ (*0* :: *real^3*)
and *vector* [*−1,0,1*] $\neq$ (*0* :: *real^3*)
and *vector* [*0,1,1*] $\neq$ (*0* :: *real^3*)
and *vector* [*0,−1,1*] $\neq$ (*0* :: *real^3*)
and *vector* [*0,1,0*] $\neq$ (*0* :: *real^3*)
and *vector* [*1,0,0*] $\neq$ (*0* :: *real^3*)
unfolding *vector-def*
by (*simp-all add*: *Cart-eq 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: Cart-eq 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: Cart-eq forall-3)
qed

lemma north-not-east-or-west: north ∉ {east, west}
proof
  assume north ∈ {east, west}
  hence east = north ∨ 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^3) = k *R vector [0,1,1]
    ∨ (vector [−1,0,1] :: real^3) = k *R vector [0,1,1]
    unfolding compass-defs
    by auto
  thus False
    unfolding vector-def
    by (simp add: Cart-eq forall-3)
qed

lemma compass-in-S:
  shows east ∈ S and west ∈ S and north ∈ S and south ∈ S
  using compass-non-zero and S-abs
  unfolding compass-defs
    and M-def
    and inner-vector-def
    and matrix-vector-mult-def
    and vector-def
```

by (*simp-all add*: *setsum-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*: *Cart-eq setsum-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-vector-def*
by (*simp-all add*: *setsum-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*: *Cart-eq 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-vector-def vector-def setsum-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*: *Cart-eq 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-vector-def vector-def setsum-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-vector-def vector-def setsum-3*)

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

```
definition rep-equator-reflect ≜ vector [
  vector [1, 0,0],
  vector [0,−1,0],
  vector [0, 0,1]] :: real^3^3
definition rep-meridian-reflect ≜ vector [
  vector [−1,0,0],
  vector [ 0,1,0],
  vector [ 0,0,1]] :: real^3^3
definition equator-reflect ≜ cltn2-abs rep-equator-reflect
definition meridian-reflect ≜ 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: Cart-eq forall-3 setsum-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*: *Cart-eq forall-3 vector-3 setsum-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-vector-def vector-def*) (*simp add*: *setsum-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*: *Cart-eq 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-vector-def vector-def*) (*simp add*: *setsum-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*: *Cart-eq 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

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

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

definition *hyp2-rep* :: *hyp2* ⇒ *real^2* **where**
hyp2-rep p ≜ *cart2-pt* (*Rep-hyp2 p*)

definition *hyp2-abs* :: *real^2* ⇒ *hyp2* **where**
hyp2-abs v = Abs-hyp2 (*proj2-pt v*)

lemma *norm-lt-1-iff-in-hyp2*:
shows *norm v < 1* ⟷ *proj2-pt v* ∈ *hyp2*
proof −
let *?v′ = vector2-append1 v*
have *?v′* ≠ *0* **by** (*rule vector2-append1-non-zero*)

from *real-less-rsqrt* [*of norm v 1*]
and *less-one-imp-sqr-less-one* [*of norm v*]
have *norm v < 1* ⟷ $(norm\ v)^2 < 1$ **by** *auto*
hence *norm v < 1* ⟷ *?v′* · (*M* ∗*v ?v′*) *< 0* **by** (*simp add*: *norm-M*)
with ⟨*?v′* ≠ *0*⟩ **have** *norm v < 1* ⟷ *proj2-abs ?v′* ∈ *K2* **by** (*subst K2-abs*)
thus *norm v < 1* ⟷ *proj2-pt v* ∈ *hyp2* **by** (*unfold proj2-pt-def hyp2-def*)
qed

lemma *norm-eq-1-iff-in-S*:
shows *norm v = 1* ⟷ *proj2-pt v* ∈ *S*
proof −
let *?v′ = vector2-append1 v*
have *?v′* ≠ *0* **by** (*rule vector2-append1-non-zero*)

from *real-sqrt-unique* [*of norm v 1*]
have *norm v = 1* ⟷ $(norm\ v)^2 = 1$ **by** *auto*
hence *norm v = 1* ⟷ *?v′* · (*M* ∗*v ?v′*) *= 0* **by** (*simp add*: *norm-M*)

```
  with ‹?v' ≠ 0› have norm v = 1 ⟷ proj2-abs ?v' ∈ S by (subst S-abs)
  thus norm v = 1 ⟷ proj2-pt v ∈ S by (unfold proj2-pt-def)
qed

lemma norm-le-1-iff-in-hyp2-S:
  norm v ≤ 1 ⟷ proj2-pt v ∈ hyp2 ∪ 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 ≠ 0 by (rule proj2-rep-non-zero)

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

  have ?p' ∈ hyp2 by (rule Rep-hyp2)
  hence ?p' ∈ K2 by (unfold hyp2-def)
  with ‹?v ≠ 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 $p \in hyp2 \cup S$
  shows *z-non-zero p*
proof −
  from ⟨$p \in hyp2 \cup S$⟩
  have *conic-sgn* $p \leq 0$ **by** (*unfold hyp2-def K2-def S-def*) *auto*
  hence *conic-sgn* $p \neq 1$ **by** *simp*
  thus *z-non-zero p* **by** (*rule conic-sgn-not-1-z-non-zero*)
qed

lemma *hyp2-S-not-equal*:
  assumes $a \in hyp2$ **and** $p \in S$
  shows $a \neq p$
  using *assms* **and** *S-K2-empty*
  by (*unfold hyp2-def*) *auto*

lemma *hyp2-S-cart2-inj*:
  assumes $p \in hyp2 \cup S$ **and** $q \in hyp2 \cup S$ **and** *cart2-pt p* = *cart2-pt q*
  shows $p = q$
proof −
  from ⟨$p \in hyp2 \cup S$⟩ **and** ⟨$q \in hyp2 \cup S$⟩
  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 $a \in hyp2$ **and** *proj2-incident a equator*
  shows $\exists\ x.\ |x| < 1 \wedge a =$ *proj2-abs* (*vector* [*x*,*0*,*1*])
proof −
  from ⟨$a \in hyp2$⟩ **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 $?v \neq 0$ **by** (*simp add*: *Cart-eq forall-3 vector-3*)
  with ⟨$a \in hyp2$⟩ **and** ⟨*a* = *proj2-abs ?v*⟩
  have $?v \cdot (M * v\ ?v) < 0$ **by** (*unfold hyp2-def*) (*simp add*: *K2-abs*)
  hence $x^2 < 1$
    unfolding *M-def matrix-vector-mult-def inner-vector-def*
    by (*simp add*: *setsum-3 vector-3 square-expand*)
  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: Cart-eq forall-3 vector-3)
  with ⟨a ∈ hyp2⟩ and ⟨a = proj2-abs ?v⟩
  have ?v · (M *v ?v) < 0 by (unfold hyp2-def) (simp add: K2-abs)
  hence y² < 1
    unfolding M-def matrix-vector-mult-def inner-vector-def
    by (simp add: setsum-3 vector-3 square-expand)
  with real-sqrt-abs [of y] and real-sqrt-less-iff [of y² 1]
  have |y| < 1 by simp
  with ⟨a = proj2-abs ?v⟩
  show ∃ y. |y| < 1 ∧ a = proj2-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 ≠ 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 ≠ 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: scalar-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 ≠ 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
      Cart-eq forall-3 setsum-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
      Cart-eq forall-3 setsum-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*) ∈ *S*
⟷ *apply-cltn2* (*apply-cltn2 p* (*cltn2-inverse J*)) *J* ∈ *S*
unfolding *is-K2-isometry-def*
by *simp*
hence *apply-cltn2 p* (*cltn2-inverse J*) ∈ *S* ⟷ *p* ∈ *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*)

9.4 *K*-isometries map the interior of the conic to itself

lemma *collinear-quadratic*:

```
  assumes t = i *R a + r
  shows t · (M *v t) =
  (a · (M *v a)) * i² + 2 * (a · (M *v r)) * i + r · (M *v r)
proof -
  from M-reverse have i * (a · (M *v r)) = i * (r · (M *v a)) by simp
  with ‹t = i *R a + r›
  show t · (M *v t) =
    (a · (M *v a)) * i² + 2 * (a · (M *v r)) * i + r · (M *v r)
    by (simp add:
      inner.add-left
      matrix-vector-right-distrib
      inner.add-right
      matrix-scalar-vector-ac
      inner.scaleR-right
      scalar-matrix-vector-assoc [symmetric]
      M-reverse
      square-expand
      algebra-simps)
qed

lemma S-quadratic':
  assumes p ≠ 0 and q ≠ 0 and proj2-abs p ≠ proj2-abs q
  shows proj2-abs (k *R p + q) ∈ S
  ⟷ p · (M *v p) * k² + p · (M *v q) * 2 * k + q · (M *v q) = 0
proof -
  let ?r = k *R p + q
  from ‹p ≠ 0› and ‹q ≠ 0› and ‹proj2-abs p ≠ proj2-abs q›
    and dependent-proj2-abs [of p q k 1]
  have ?r ≠ 0 by auto
  hence proj2-abs ?r ∈ S ⟷ ?r · (M *v ?r) = 0 by (rule S-abs)
  with collinear-quadratic [of ?r k p q]
  show proj2-abs ?r ∈ S
    ⟷ p · (M *v p) * k² + p · (M *v q) * 2 * k + q · (M *v q) = 0
    by (simp add: dot-lmul-matrix [symmetric] algebra-simps)
qed

lemma S-quadratic:
  assumes p ≠ q and r = proj2-abs (k *R proj2-rep p + proj2-rep q)
  shows r ∈ S
  ⟷ proj2-rep p · (M *v proj2-rep p) * k²
     + proj2-rep p · (M *v proj2-rep q) * 2 * k
     + proj2-rep q · (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 ≠ 0 and ?v ≠ 0 by (rule proj2-rep-non-zero)+
```

from $\langle p \neq q \rangle$ **have** $\mathit{proj2\text{-}abs}\ ?u \neq \mathit{proj2\text{-}abs}\ ?v$ **by** (*simp add*: *proj2-abs-rep*)
with $\langle ?u \neq 0 \rangle$ **and** $\langle ?v \neq 0 \rangle$ **and** $\langle r = \mathit{proj2\text{-}abs}\ ?w \rangle$
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* :: $\mathit{real}\hat{\ }3 \Rightarrow \mathit{real}\hat{\ }3 \Rightarrow \mathit{real}$ **where**
$\mathit{quarter\text{-}discrim}\ p\ q \triangleq (p \cdot (M *v\ q))^2 - p \cdot (M *v\ p) * (q \cdot (M *v\ q))$

lemma *quarter-discrim-invariant*:
assumes $t = i *_R a + r$
shows $\mathit{quarter\text{-}discrim}\ a\ t = \mathit{quarter\text{-}discrim}\ a\ r$
proof −
from $\langle t = i *_R a + r \rangle$
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-scalar-vector-ac
scalar-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*: *square-expand algebra-simps*)
moreover from *collinear-quadratic* **and** $\langle t = i *_R a + r \rangle$
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*: *square-expand algebra-simps*)
ultimately show $\mathit{quarter\text{-}discrim}\ a\ t = \mathit{quarter\text{-}discrim}\ a\ r$
by (*unfold quarter-discrim-def*, *simp*)
qed

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

from $\langle p \neq 0 \rangle$ **and** $\langle ?pp \in K2 \rangle$
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-vector-def setsum-3 square-expand*)

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*: *Cart-eq 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 $b \neq 0$
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*: *square-expand*
matrix-scalar-vector-ac
scalar-matrix-vector-assoc [*symmetric*])
qed

definition *S-intersection-coeff1* :: *real^3* ⇒ *real^3* ⇒ *real* **where**
S-intersection-coeff1 p q
$\triangleq (-p \cdot (M *v\ q) + sqrt\ (quarter\text{-}discrim\ p\ q))\ /\ (p \cdot (M *v\ p))$

definition *S-intersection-coeff2* :: *real^3* ⇒ *real^3* ⇒ *real* **where**
S-intersection-coeff2 p q
$\triangleq (-p \cdot (M *v\ q) - sqrt\ (quarter\text{-}discrim\ p\ q))\ /\ (p \cdot (M *v\ p))$

definition *S-intersection1-rep* :: *real^3* ⇒ *real^3* ⇒ *real^3* **where**
S-intersection1-rep p q $\triangleq$ (*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 $?a = p \cdot (M *v\ p)$
let $?b = 2 * (p \cdot (M *v\ q))$
let $?c = q \cdot (M *v\ q)$

from ‹$p \neq 0$› **and** ‹$?pp \in K2$› **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*: *square-expand*)
with *times-divide-times-eq* [*of*
2 2 *sqrt* (*quarter-discrim p q*) $- p \cdot (M *v\ q)$ *?a*]
and *times-divide-times-eq* [*of*
2 2 $-p \cdot (M *v\ q) -$ *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* $(p \cdot (M *v\ p))\ (2 * (p \cdot (M *v\ q)))\ (q \cdot (M *v\ q)) > 0$
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 $p \cdot (M *v\ p) * ?j^2 + 2 * (p \cdot (M *v\ q)) * ?j + q \cdot (M *v\ q) = 0$
and $p \cdot (M *v\ p) * ?k^2 + 2 * (p \cdot (M *v\ q)) * ?k + q \cdot (M *v\ q) = 0$
by (*unfold S-intersection-coeffs-defs, auto*)
with ‹$p \neq 0$› **and** ‹$q \neq 0$› **and** ‹$?pp \neq ?pq$› **and** *S-quadratic′*
show *S-intersection1 p q* $\in S$ **and** *S-intersection2 p q* $\in S$
by (*unfold S-intersections-defs, simp-all*)
qed

lemma *S-intersections-Col*:
assumes $p \neq 0$ **and** $q \neq 0$
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** $?pp \neq ?pq$
with ‹$p \neq 0$› **and** ‹$q \neq 0$› **and** *dependent-proj2-abs* [*of p q* - 1]
have *S-intersection1-rep p q* $\neq 0$ (**is** $?r \neq 0$)
and *S-intersection2-rep p q* $\neq 0$ (**is** $?s \neq 0$)

```
    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 ∈ K2 and a ≠ r
 shows ∃ s u. s ≠ u ∧ s ∈ S ∧ u ∈ S ∧ proj2-Col a r s ∧ proj2-Col a r u
proof −
 let ?a' = proj2-rep a
 let ?r' = proj2-rep r
 from proj2-rep-non-zero have ?a' ≠ 0 and ?r' ≠ 0 by simp-all

 from ‹?a' ≠ 0› and K2-imp-M-neg and proj2-abs-rep and ‹a ∈ K2›
 have ?a' · (M *v ?a') < 0 by simp

 from ‹a ≠ r› have proj2-abs ?a' ≠ proj2-abs ?r' by (simp add: proj2-abs-rep)

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

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

 from ‹?a' ≠ 0› and ‹?r' ≠ 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 ≠ ?u› and ‹?s ∈ S› and ‹?u ∈ S›
  show ∃ s u. s ≠ u ∧ s ∈ S ∧ u ∈ S ∧ proj2-Col a r s ∧ proj2-Col a r u
    by auto
qed

lemma point-in-S-polar-is-tangent:
  assumes p ∈ S and q ∈ S and proj2-incident q (polar p)
  shows q = p
proof −
  from ‹p ∈ 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 ∉ S and proj2-incident r (polar p) by auto
  let ?u = proj2-rep r
  let ?v = proj2-rep p
  from ‹r ∉ S› and ‹p ∈ S› and ‹q ∈ S› have r ≠ p and q ≠ 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-abs (k *R ?u + ?v) by auto
  with ‹r ≠ p› and ‹q ∈ S› and S-quadratic
  have ?u · (M *v ?u) * k^2 + ?u · (M *v ?v) * 2 * k + ?v · (M *v ?v) = 0
    by simp
  moreover from ‹p ∈ S› have ?v · (M *v ?v) = 0 by (unfold S-alt-def)
  moreover from ‹proj2-incident r (polar p)›
  have ?u · (M *v ?v) = 0 by (unfold incident-polar)
  moreover from ‹r ∉ S› have ?u · (M *v ?u) ≠ 0 by (unfold S-alt-def)
  ultimately have k = 0 by simp
  with ‹q = proj2-abs (k *R ?u + ?v)›
  show q = p by (simp add: proj2-abs-rep)
qed

lemma line-through-K2-intersect-S-twice:
  assumes p ∈ K2 and proj2-incident p l
  shows ∃ q r. q ≠ r ∧ q ∈ S ∧ r ∈ S ∧ proj2-incident q l ∧ proj2-incident r l
proof −
  from proj2-another-point-on-line
  obtain s where s ≠ p and proj2-incident s l by auto
  from ‹p ∈ K2› and ‹s ≠ p› and K2-line-intersect-twice [of p s]
  obtain q and r where q ≠ r and q ∈ S and r ∈ S
    and proj2-Col p s q and proj2-Col p s r
    by auto
  with ‹s ≠ 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 ≠ 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

lemma line-through-K2-intersect-S-again:
  assumes p ∈ K2 and proj2-incident p l
  shows ∃ r. r ≠ q ∧ r ∈ S ∧ proj2-incident r l
proof −
  from ‹p ∈ K2› and ‹proj2-incident p l›
    and line-through-K2-intersect-S-twice [of p l]
  obtain s and t where s ≠ t and s ∈ S and t ∈ S
    and proj2-incident s l and proj2-incident t l
    by auto
  show ∃ r. r ≠ q ∧ r ∈ S ∧ proj2-incident r l
  proof cases
    assume t = q
    with ‹s ≠ t› and ‹s ∈ S› and ‹proj2-incident s l›
    have s ≠ q ∧ s ∈ S ∧ proj2-incident s l by simp
    thus ∃ r. r ≠ q ∧ r ∈ S ∧ 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 * i² + ?b * i + ?c = 0 by simp
      with ‹?a ≠ 0› and discriminant-iff have i = ?j ∨ i = ?k by simp
      with ‹r = proj2-abs (i *R ?v + ?w)› have r = ?p ∨ r = ?q by auto }
    thus proj2-incident r l ⟶ r = ?p ∨ r = ?q ..
  qed
  thus ∃ p q. ∀ r∈S. proj2-incident r l ⟶ r = p ∨ r = q by auto
qed

lemma card-line-intersect-S:
  assumes T ⊆ S and proj2-set-Col T
  shows card T ≤ 2
proof −
  from ‹proj2-set-Col T›
  obtain l where ∀ p∈T. proj2-incident p l unfolding proj2-set-Col-def ..
  from line-intersect-S-at-most-twice [of l]
  obtain b and c where ∀ a∈S. proj2-incident a l ⟶ a = b ∨ a = c by auto
  with ‹∀ p∈T. proj2-incident p l› and ‹T ⊆ S›
  have T ⊆ {b,c} by auto
  hence card T ≤ card {b,c} by (simp add: card-mono)
  also from card-suc-ge-insert [of b {c}] have . . . ≤ 2 by simp
  finally show card T ≤ 2 .
qed

lemma line-S-two-intersections-only:
  assumes p ≠ q and p ∈ S and q ∈ S and r ∈ S
  and proj2-incident p l and proj2-incident q l and proj2-incident r l
  shows r = p ∨ r = q
proof −
  from ‹p ≠ 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 $\exists$ *q r*. $q \neq r \wedge q \in S \wedge r \in S \wedge$ *proj2-incident q l* $\wedge$ *proj2-incident r l*
$\wedge$ ($\forall$ $s \in S$. *proj2-incident s l* $\longrightarrow s = q \vee s = r$)
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 $\exists$ *q r*. $q \neq r \wedge q \in S \wedge r \in S \wedge$ *proj2-incident q l* $\wedge$ *proj2-incident r l*
  $\wedge$ ($\forall$ $s \in S$. *proj2-incident s l* $\longrightarrow s = q \vee s = r$)
  by *blast*
qed

lemma *tangent-not-through-K2*:
assumes $p \in S$ **and** $q \in K2$
shows $\neg$ *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 $\exists$ *l*. *proj2-incident p l* $\wedge$ ($\forall$ *q*. *proj2-incident q l* $\longrightarrow q \notin S$)

```
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) ∗ k² ≥ 0
     by (simp add: mult-nonneg-nonneg)
```

with ⟨*proj2-rep p* $\cdot$ (*M* $*v$ *proj2-rep ?r*) $= 0$⟩
and ⟨*proj2-rep ?r* $\cdot$ (*M* $*v$ *proj2-rep ?r*) > 0⟩
have *proj2-rep p* $\cdot$ (*M* $*v$ *proj2-rep p*) $* k^2$
$+$ *proj2-rep p* $\cdot$ (*M* $*v$ *proj2-rep ?r*) $* 2 * k$
$+$ *proj2-rep ?r* $\cdot$ (*M* $*v$ *proj2-rep ?r*)
> 0
by *simp*
with ⟨$p \neq ?r$⟩ **and** ⟨$q =$ *proj2-abs* ($k *_R$ *proj2-rep p* $+$ *proj2-rep ?r*)⟩
show $q \notin S$ **by** (*simp add*: *S-quadratic*)
qed
qed
qed
with ⟨*proj2-incident p ?l*⟩
show $\exists$ *l*. *proj2-incident p l* $\wedge$ ($\forall$ *q*. *proj2-incident q l* $\longrightarrow q \notin S$)
by (*simp add*: *exI* [*of* - *?l*])
qed

lemma *lines-through-intersect-S-twice-in-K2*:
assumes $\forall$ *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*)
shows $p \in K2$
proof (*rule ccontr*)
assume $p \notin K2$
hence *conic-sgn p* ≥ 0 **by** (*unfold K2-def*) *simp*

have $\neg$ ($\forall$ *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 *cases*
assume *conic-sgn p* $= 0$
hence $p \in S$ **unfolding** *S-def* **..**
hence *proj2-incident p* (*polar p*) **by** (*simp add*: *incident-own-polar-in-S*)
let *?l* $=$ *polar p*
have $\neg$ ($\exists$ *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 $\exists$ *q r*.
$q \neq r \wedge q \in S \wedge r \in S \wedge$ *proj2-incident q ?l* $\wedge$ *proj2-incident r ?l*
then 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*
from ⟨$p \in S$⟩ **and** ⟨$q \in S$⟩ **and** ⟨*proj2-incident q ?l*⟩
and ⟨$r \in S$⟩ **and** ⟨*proj2-incident r ?l*⟩
have $q = p$ **and** $r = p$ **by** (*simp add*: *point-in-S-polar-is-tangent*)+
with ⟨$q \neq r$⟩ **show** *False* **by** *simp*
qed
with ⟨*proj2-incident p ?l*⟩
show $\neg$ ($\forall$ *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*))
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 ≠ r ∧ q ∈ S ∧ r ∈ S ∧ proj2-incident q l ∧ proj2-incident r l))
      by auto
  qed
  with ⟨∀ l. proj2-incident p l ⟶ (∃ q r.
    q ≠ r ∧ q ∈ S ∧ r ∈ S ∧ proj2-incident q l ∧ proj2-incident r l)⟩
  show False by simp
qed

lemma line-through-hyp2-pole-not-in-hyp2:
  assumes a ∈ hyp2 and proj2-incident a l
  shows pole l ∉ hyp2
proof −
  from assms and line-through-K2-intersect-S
  obtain p where p ∈ S and proj2-incident p l by (unfold hyp2-def) auto

  from ⟨proj2-incident p l⟩
  have proj2-incident (pole l) (polar p) by (rule incident-pole-polar)
  with ⟨p ∈ S⟩
  show pole l ∉ hyp2
    by (unfold hyp2-def) (auto simp add: tangent-not-through-K2)
qed

lemma statement60-one-way:
  assumes is-K2-isometry J and p ∈ K2
  shows apply-cltn2 p J ∈ K2 (is ?p′ ∈ K2)
proof −
  let ?J′ = cltn2-inverse J

  have ∀ l′. proj2-incident ?p′ l′ ⟶ (∃ q′ r′.
```

```
    q′ ≠ r′ ∧ q′ ∈ S ∧ r′ ∈ S ∧ proj2-incident q′ l′ ∧ proj2-incident r′ l′)
  proof
    fix l′
    let ?l = apply-cltn2-line l′ ?J′
    show proj2-incident ?p′ l′ ⟶ (∃ q′ r′.
      q′ ≠ r′ ∧ q′ ∈ S ∧ r′ ∈ S ∧ proj2-incident q′ l′ ∧ 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 ∈ K2› and line-through-K2-intersect-S-twice [of p ?l]
      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
      let ?q′ = apply-cltn2 q J
      let ?r′ = apply-cltn2 r J
      from ‹q ≠ r› and apply-cltn2-injective [of q J r] have ?q′ ≠ ?r′ by auto

      from ‹q ∈ S› and ‹r ∈ 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
  hence p ∈ K2 by (unfold hyp2-def)
  with ‹is-K2-isometry J›
  have apply-cltn2 p J ∈ hyp2 by (unfold hyp2-def) (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* $\in$ *hyp2* $\cup$ *S* **..**
qed

lemma *is-K2-isometry-z-non-zero*:
assumes *p* $\in$ *hyp2* $\cup$ *S* **and** *is-K2-isometry J*
shows *z-non-zero* (*apply-cltn2 p J*)
proof −
from ⟨*p* $\in$ *hyp2* $\cup$ *S*⟩ **and** ⟨*is-K2-isometry J*⟩
have *apply-cltn2 p J* $\in$ *hyp2* $\cup$ *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* $\in$ *hyp2* $\cup$ *S* **and** *is-K2-isometry J*
shows $\exists$ *k*. *k* $\neq$ *0*
$\wedge$ *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*: *scalar-vector-matrix-assoc*)

from ⟨*p* $\in$ *hyp2* $\cup$ *S*⟩ **have** (*proj2-rep p*)\$*3* $\neq$ *0* **by** (*rule hyp2-S-z-non-zero*)

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

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*)
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* $\neq$ *0*⟩ **and** ⟨*j* $\neq$ *0*⟩
and ⟨(*proj2-rep* (*apply-cltn2 p J*))\$*3* $\neq$ *0*⟩
have *?k* $\neq$ *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

9.5 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 *Rep-hyp2* [*of p*] **have** *Rep-hyp2 p* ∈ *K2* **by** (*unfold hyp2-def*)
with ⟨*is-K2-isometry J*⟩
have *apply-cltn2* (*Rep-hyp2 p*) *J* ∈ *K2* **by** (*rule statement60-one-way*)
thus *apply-cltn2* (*Rep-hyp2 p*) *J* ∈ *hyp2* **by** (*unfold hyp2-def*)
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 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 ⊗_?G J)
    by (simp add: hyp2-cltn2-compose)
qed
```

9.6 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 default+
    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 (op $ (?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 (op $ (?t$1)))
  by (simp add: three-in-S-tangent-intersection-no-3-Col)
 moreover have range (op $ (?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 (op $ (?t$2))) by simp
  ultimately have ∀ i. proj2-no-3-Col (range (op $ (?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: Cart-eq)

  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-vector-def and matrix-vector-mult-def
    by (simp add: setsum-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^3) · (?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-vector-def and matrix-vector-mult-def
    by (simp add: setsum-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^3) · (?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-vector-def* **and** *matrix-vector-mult-def*
by (*simp add*: *setsum-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-vector-def* **and** *matrix-vector-mult-def*
by (*simp-all add*: *setsum-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: Cart-eq 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$i)
proof default+
  fix i :: 2
  fix u :: proj2
  assume proj2-incident u (?m$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 (unfold hyp2-def) 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** $p \neq q$
  and $p \in S$ **and** $q \in S$
  and *proj2-incident p ?l* **and** *proj2-incident q ?l*
  and $\forall$ *r*$\in$*S*. *proj2-incident r ?l* $\longrightarrow$ $r = p \vee r = q$
  by *auto*
from ⟨$a \in K2$⟩ **and** ⟨$b \in K2$⟩ **and** ⟨$p \in S$⟩ **and** ⟨$q \in S$⟩
  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 ⟨$a \in K2$⟩ **and** ⟨$p \in S$⟩ **have** $a \neq p$
  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 ⟨$q \in S$⟩ **and** ⟨*is-K2-isometry J*⟩
have *apply-cltn2 q J* $\in$ *S* **by** (*unfold is-K2-isometry-def*) *simp*
with ⟨*proj2-incident* (*apply-cltn2 q J*) *?l*⟩
  and ⟨$\forall$ *r*$\in$*S*. *proj2-incident r ?l* $\longrightarrow$ $r = p \vee r = q$⟩
have *apply-cltn2 q J* = *p* $\vee$ *apply-cltn2 q J* = *q* **by** *simp*

have *apply-cltn2 q J* $\neq$ *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 ⟨$p \neq q$⟩ **show** *False* **..**
qed
with ⟨*apply-cltn2 q J* = *p* $\vee$ *apply-cltn2 q J* = *q*⟩
have *apply-cltn2 q J* = *p* **by** *simp*
with ⟨$p \neq q$⟩
  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 $\exists$ *J*. *is-K2-isometry J* $\wedge$ *apply-cltn2 a J* = *b* $\wedge$ *apply-cltn2 b J* = *a*
by (*simp add*: *exI* [*of* - *J*])
qed

theorem *hyp2-axiom1*: $\forall$ *a b*. *a b* $\equiv_K$ *b a*
proof *default*+
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 *default*+
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*: ∀ *a b c*. *a b* $\equiv_K$ *c c* ⟶ *a* = *b*
proof *default*+
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* ∈ *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*

9.7 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_ℝ ?cp ?cq ?cr ∨ B_ℝ ?cp ?cr ?cq
proof cases
  assume B_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 ≥ 0› and ‹k ≤ 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 ≥ 0› and mult-mono [of k k norm ?cq 1]
  have k * norm ?cq ≤ k by simp

  from ‹norm ?cr ≤ 1› and ‹k ≤ 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 ∈ S and r ∈ S by (simp-all add: norm-eq-1-iff-in-S)
      with ‹p ∈ 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 ∨ q = r ∨ r = p
      proof (rule ccontr)
        assume ¬ (p = q ∨ q = r ∨ r = p)
        hence p ≠ q and q ≠ r and r ≠ p by simp-all
        from ‹q ≠ r› have card {q,r} = 2 by simp
        with ‹p ≠ q› and ‹r ≠ 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_ℝ ?cp ?cq ?cr ∨ B_ℝ ?cp ?cr ?cq
      by (auto simp add: real-euclid.th3-1 real-euclid.th3-2)
  next
    assume ¬ B_ℝ ?cr ?cp ?cq
    with ‹real-euclid.Col ?cp ?cq ?cr›
    show B_ℝ ?cp ?cq ?cr ∨ B_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_ℝ (cart2-pt p) (cart2-pt r) (cart2-pt q) (is B_ℝ ?cp ?cr ?cq)
proof (rule ccontr)
  assume ¬ B_ℝ ?cp ?cr ?cq
  hence ¬ B_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_ℝ ?cp ?cq ?cr ∨ B_ℝ ?cp ?cr ?cq by (simp add: S-at-edge)
  with ‹¬ B_ℝ ?cp ?cr ?cq› have B_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_ℝ ?cq ?cp ?cr ∨ B_ℝ ?cq ?cr ?cp by (simp add: S-at-edge)
  with ‹¬ B_ℝ ?cq ?cr ?cp› have B_ℝ ?cq ?cp ?cr by simp
  with ‹B_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 ⟨$p \neq q$⟩ **show** *False* **..**
qed

lemma *hyp2-incident-in-middle*:
assumes $p \neq q$ **and** $p \in S$ **and** $q \in S$ **and** $a \in hyp2 \cup S$
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 ⟨$p \in S$⟩ **and** ⟨$q \in S$⟩ **and** ⟨$a \in hyp2 \cup S$⟩ **and** *this* **and** ⟨$p \neq q$⟩
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 ∃ *r*∈*S*. $B_{\mathbb{R}}$ (*cart2-pt p*) (*cart2-pt q*) (*cart2-pt r*)
(**is** ∃ *r*∈*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** ∃ *r*∈*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** (*unfold hyp2-def*) *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 ∈ S› and ‹t ∈ S› and ‹p ∈ hyp2 ∪ S› and ‹q ∈ hyp2 ∪ S›
  have Bℝ ?cs ?cp ?cq ∨ Bℝ ?cs ?cq ?cp and Bℝ ?ct ?cp ?cq ∨ Bℝ ?ct ?cq ?cp
    by (simp-all add: S-at-edge)
  with real-euclid.th3-2
  have Bℝ ?cq ?cp ?cs ∨ Bℝ ?cp ?cq ?cs and Bℝ ?cq ?cp ?ct ∨ Bℝ ?cp ?cq ?ct
    by fast+

  from ‹s ∈ S› and ‹t ∈ S› and ‹q ∈ hyp2 ∪ S› and ‹proj2-Col s t q› and ‹s ≠ t›
  have Bℝ ?cs ?cq ?ct by (rule hyp2-in-middle)
  hence Bℝ ?ct ?cq ?cs by (rule real-euclid.th3-2)

  have Bℝ ?cp ?cq ?cs ∨ Bℝ ?cp ?cq ?ct
  proof (rule ccontr)
    assume ¬ (Bℝ ?cp ?cq ?cs ∨ Bℝ ?cp ?cq ?ct)
    hence ¬ Bℝ ?cp ?cq ?cs and ¬ Bℝ ?cp ?cq ?ct by simp-all
    with ‹Bℝ ?cq ?cp ?cs ∨ Bℝ ?cp ?cq ?cs›
      and ‹Bℝ ?cq ?cp ?ct ∨ Bℝ ?cp ?cq ?ct›
    have Bℝ ?cq ?cp ?cs and Bℝ ?cq ?cp ?ct by simp-all
    from ‹¬ Bℝ ?cp ?cq ?cs› and ‹Bℝ ?cq ?cp ?cs› have ?cp ≠ ?cq by auto
    with ‹Bℝ ?cq ?cp ?cs› and ‹Bℝ ?cq ?cp ?ct›
    have Bℝ ?cq ?cs ?ct ∨ Bℝ ?cq ?ct ?cs
      by (simp add: real-euclid-th5-1 [of ?cq ?cp ?cs ?ct])
    with ‹Bℝ ?cs ?cq ?ct› and ‹Bℝ ?ct ?cq ?cs›
    have ?cq = ?cs ∨ ?cq = ?ct by (auto simp add: real-euclid.th3-4)
    with ‹q ∈ hyp2 ∪ S› and ‹s ∈ S› and ‹t ∈ S›
    have q = s ∨ q = t by (auto simp add: hyp2-S-cart2-inj)
    with ‹s ∈ S› and ‹t ∈ S› have q ∈ S by auto
    with ‹q ∉ S› show False ..
  qed
  with ‹s ∈ S› and ‹t ∈ S› show ∃ r∈S. Bℝ ?cp ?cq (cart2-pt r) by auto
qed

definition endpoint-in-S :: proj2 ⇒ proj2 ⇒ proj2 where
  endpoint-in-S a b
  ≜ ϵ p. p∈S ∧ Bℝ (cart2-pt a) (cart2-pt b) (cart2-pt p)

lemma endpoint-in-S:
  assumes a ∈ hyp2 ∪ S and b ∈ hyp2 ∪ S
  shows endpoint-in-S a b ∈ S (is ?p ∈ S)
  and Bℝ (cart2-pt a) (cart2-pt b) (cart2-pt (endpoint-in-S a b))
  (is Bℝ ?ca ?cb ?cp)
proof −
  from ‹a ∈ hyp2 ∪ S› and ‹b ∈ hyp2 ∪ S› and extend-to-S
  have ∃ p. p ∈ S ∧ Bℝ ?ca ?cb (cart2-pt p) by auto
  hence ?p ∈ S ∧ Bℝ ?ca ?cb ?cp
    by (unfold endpoint-in-S-def) (rule someI-ex)
  thus ?p ∈ S and Bℝ ?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 ‹$a \neq b$› **and** ‹$a \in hyp2 \cup S$› **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*)

  assume $?p = ?q$
  with ‹$B_{\mathbb{R}}\ ?cb\ ?ca\ ?cq$› **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 ⟨$a \in hyp2 \cup S$⟩ **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*)
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 ⟨$a \neq b$⟩ **and** ⟨$a \in hyp2 \cup S$⟩ **and** ⟨$b \in hyp2 \cup S$⟩ **and** ⟨$p \in S$⟩
have $p = ?q \vee p =$ *endpoint-in-S b a* (**is** $p = ?q \vee p = ?r$)
by (*rule endpoints-in-S-incident-unique*)

assume $p \neq ?q$
with ⟨$p = ?q \vee p = ?r$⟩ **have** $p = ?r$ **by** *simp*
with ⟨$b \in hyp2 \cup S$⟩ **and** ⟨$a \in hyp2 \cup S$⟩
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 ⟨$a \in hyp2 \cup S$⟩ **and** ⟨$b \in hyp2 \cup S$⟩ **have** $a = b$ **by** (*rule hyp2-S-cart2-inj*)
with ⟨$a \neq b$⟩ **show** *False* **..**
qed

lemma *between-hyp2-S*:
assumes $p \in hyp2 \cup S$ **and** $r \in hyp2 \cup S$ **and** $k \geq 0$ **and** $k \leq 1$
shows *proj2-pt* ($k *_R$ (*cart2-pt r*) $+ (1 - k) *_R$ (*cart2-pt p*)) $\in hyp2 \cup S$
(**is** *proj2-pt ?cq* $\in$ -)
proof $-$
let $?cp =$ *cart2-pt p*
let $?cr =$ *cart2-pt r*
let $?q =$ *proj2-pt ?cq*
from ⟨$p \in hyp2 \cup S$⟩ **and** ⟨$r \in hyp2 \cup S$⟩
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 ⟨$p \in hyp2 \cup S$⟩ **and** ⟨$r \in hyp2 \cup S$⟩
have *norm ?cp* ≤ 1 **and** *norm ?cr* ≤ 1

by (*simp-all add*: *norm-le-1-iff-in-hyp2-S*)

from ⟨$k \geq 0$⟩ **and** ⟨$k \leq 1$⟩
and *norm-triangle-ineq* [*of* $k *_R$ *?cr* $(1 - k) *_R$ *?cp*]
have *norm ?cq* $\leq k *$ *norm ?cr* $+ (1 - k) *$ *norm ?cp* **by** *simp*

from ⟨$k \geq 0$⟩ **and** ⟨*norm ?cr* ≤ 1⟩ **and** *mult-mono* [*of k k norm ?cr 1*]
have $k *$ *norm ?cr* $\leq k$ **by** *simp*

from ⟨$k \leq 1$⟩ **and** ⟨*norm ?cp* ≤ 1⟩
and *mult-mono* [*of* $1 - k$ $1 - k$ *norm ?cp 1*]
have $(1 - k) *$ *norm ?cp* $\leq 1 - k$ **by** *simp*
with ⟨*norm ?cq* $\leq k *$ *norm ?cr* $+ (1 - k) *$ *norm ?cp*⟩ **and** ⟨$k *$ *norm ?cr* $\leq k$⟩
have *norm ?cq* ≤ 1 **by** *simp*
thus *?q* $\in$ *hyp2* $\cup$ *S* **by** (*simp add*: *norm-le-1-iff-in-hyp2-S*)
qed

9.8 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 ∈ 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 *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: Cart-eq)
  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
      scalar-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: real-sgn-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: real-sgn-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 ∈ hyp2 ∪ S and q ∈ hyp2 ∪ S and r ∈ hyp2 ∪ S
  and is-K2-isometry J and Bℝ (cart2-pt p) (cart2-pt q) (cart2-pt r)
  shows Bℝ
  (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 ⟨Bℝ ?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 $\langle r \in hyp2 \cup S\rangle$ **and** $\langle p \in hyp2 \cup S\rangle$ **and** $\langle q \in hyp2 \cup S\rangle$
and ⟨*is-K2-isometry J*⟩ **and** $\langle ?cq = k *_R ?cr + (1 - k) *_R ?cp\rangle$
have $?eq = k * ?er + (1 - k) * ?ep$ **by** (*rule expansion-factor-linear*)
with $\langle ?eq \neq 0\rangle$ **have** $1 - ?c = (1 - k) * ?ep\ /\ ?eq$ **by** (*simp add*: *field-simps*)
with $\langle k \leq 1\rangle$ **and** $\langle ?ep\ /\ ?eq > 0\rangle$
and *mult-nonneg-nonneg* [*of* $1 - k\ ?ep\ /\ ?eq$]
have $?c \leq 1$ **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 $\langle p \in hyp2 \cup S\rangle$ **and** $\langle q \in hyp2 \cup S\rangle$ **and** $\langle r \in hyp2 \cup S\rangle$
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 $\langle r \in hyp2 \cup S\rangle$ **and** $\langle p \in hyp2 \cup S\rangle$ **and** $\langle q \in hyp2 \cup S\rangle$
and ⟨*is-K2-isometry J*⟩ **and** $\langle ?cq = k *_R ?cr + (1 - k) *_R ?cp\rangle$
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 $\langle 1 - ?c = (1 - k) * ?ep\ /\ ?eq\rangle$ **and** $\langle ?eq \neq 0\rangle$
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*: *Cart-eq forall-2 forall-3*)
with $\langle ?c \geq 0\rangle$ **and** $\langle ?c \leq 1\rangle$
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* = ϵ *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* λ *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* = ϵ *r*. *r* $\in$ *S* $\wedge$ $B_{\mathbb{R}}$ *?cq ?ca* (*cart2-pt r*)
let *?cr* = *cart2-pt ?pr*
from ⟨*?pq* $\in$ *hyp2*⟩ **and** ⟨*?pa* $\in$ *hyp2*⟩ **and** *extend-to-S* [*of ?pq ?pa*]
  and *someI-ex* [*of* λ *r*. *r* $\in$ *S* $\wedge$ $B_{\mathbb{R}}$ *?cq ?ca* (*cart2-pt r*)]
have *?pr* $\in$ *S* **and** $B_{\mathbb{R}}$ *?cq ?ca ?cr* **by** *auto*

from ⟨*?pb* $\in$ *hyp2*⟩ **and** ⟨*?pa* $\in$ *hyp2*⟩ **and** ⟨*?pp* $\in$ *S*⟩ **and** ⟨*?pr* $\in$ *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 (*unfold hyp2-def*) *auto*
let *?px* = *apply-cltn2 ?pc J*
let *?cx* = *cart2-pt ?px*
let *?x* = *Abs-hyp2 ?px*
from ⟨*is-K2-isometry J*⟩ **and** ⟨*?pc* $\in$ *hyp2*⟩
have *?px* $\in$ *hyp2* **by** (*unfold hyp2-def*) (*rule statement60-one-way*)
hence *Rep-hyp2 ?x* = *?px* **by** (*rule Abs-hyp2-inverse*)

from ⟨*?pb* $\in$ *hyp2*⟩ **and** ⟨*?pc* $\in$ *hyp2*⟩ **and** ⟨*?pp* $\in$ *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 ≡K a ?x
    by (unfold real-hyp2-C-def) (simp add: exI [of - J])
  hence a ?x ≡K b c by (rule hyp2.th2-2)
  with ⟨BK q a ?x⟩
  show ∃ x. BK q a x ∧ a x ≡K b c by (simp add: exI [of - ?x])
qed
```

9.9 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⟩ have a ∈ K2 by (unfold hyp2-def)
  with ⟨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 ‹$q \neq p$› **show** *False* **..**
qed
with ‹$q \neq p$› **and** ‹$a \neq p$› **and** ‹$a \neq q$› **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 $a \neq b$ **and** $a \in hyp2 \cup S$ **and** $b \in hyp2 \cup S$ **and** *is-K2-isometry J*
shows *apply-cltn2* (*endpoint-in-S a b*) *J*
= *endpoint-in-S* (*apply-cltn2 a J*) (*apply-cltn2 b J*)
(**is** $?pJ =$ *endpoint-in-S ?aJ ?bJ*)
proof −
let $?p =$ *endpoint-in-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-pt a*
let $?cb =$ *cart2-pt b*
let $?cp =$ *cart2-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-pt ?aJ*
let $?cbJ =$ *cart2-pt ?bJ*
let $?cpJ =$ *cart2-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-in-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-pt a*) (*cart2-pt b*) (*cart2-pt c*) (**is** $B_{\mathbb{R}}$ *?ca ?cb ?cc*)
shows *endpoint-in-S a b* = *endpoint-in-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 $Rep\text{-}hyp2\ a \neq Rep\text{-}hyp2\ b$ (**is** $?pa \neq ?pb$)
and $Rep\text{-}hyp2\ b \neq Rep\text{-}hyp2\ c$ (**is** $?pb \neq ?pc$)
and $Rep\text{-}hyp2\ b \neq Rep\text{-}hyp2\ d$ (**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 $?pp = endpoint\text{-}in\text{-}S\ ?pb\ ?pc$
let $?ca = cart2\text{-}pt\ ?pa$
let $?cb = cart2\text{-}pt\ ?pb$
let $?cc = cart2\text{-}pt\ ?pc$
let $?cd = cart2\text{-}pt\ ?pd$
let $?cp = cart2\text{-}pt\ ?pp$
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** (*unfold hyp2-def*, *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 (*unfold hyp2-def*, *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 $B_{\mathbb{R}}$

$(cart2\text{-}pt\ (endpoint\text{-}in\text{-}S\ a\ b))\ (cart2\text{-}pt\ a)\ (cart2\text{-}pt\ (endpoint\text{-}in\text{-}S\ b\ a))$
(**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* ≠ *?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* ≠ *?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 ≠ b and a ∈ hyp2 ∪ S and b ∈ hyp2 ∪ 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 ≠ b› and ‹a ∈ hyp2 ∪ S› and ‹b ∈ hyp2 ∪ 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 ≠ b and a ∈ hyp2 and b ∈ 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 ≠ 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 and ?q ∈ 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 ≠ ?q› and ‹?p ∈ S› and ‹?q ∈ S› and ‹a ∈ hyp2› and ‹b ∈ 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 ∈ S and a ∈ hyp2 ∪ S and p ≠ a
  shows endpoint-in-S p a ≠ p
proof
  from ‹p ≠ a› and ‹p ∈ S› and ‹a ∈ hyp2 ∪ S›
  have B_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_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 ∈ S› and ‹a ∈ hyp2 ∪ S› have p = a by (simp add: hyp2-S-cart2-inj)
  with ‹p ≠ a› show False ..
qed

lemma endpoint-in-S-S-strict-hyp2-distinct:
  assumes p ∈ S and a ∈ hyp2
```

```
  shows endpoint-in-S p a ≠ p
proof −
  from ‹a ∈ hyp2› and ‹p ∈ S›
  have p ≠ a by (rule hyp2-S-not-equal [symmetric])
  with assms
  show endpoint-in-S p a ≠ p by (simp add: endpoint-in-S-S-hyp2-distinct)
qed

lemma end-and-opposite-are-endpoints-in-S:
  assumes a ∈ hyp2 and b ∈ hyp2 and p ∈ 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 ∈ S› and ‹b ∈ hyp2›
  have p ≠ ?q by (rule endpoint-in-S-S-strict-hyp2-distinct [symmetric])

  from ‹p ∈ S› and ‹b ∈ hyp2› have ?q ∈ S by (simp add: endpoint-in-S)

  from ‹b ∈ hyp2› and ‹p ∈ S›
  have p ≠ b by (rule hyp2-S-not-equal [symmetric])
  with ‹p ∈ S› and ‹b ∈ 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 ≠ ?q› and ‹p ∈ S› and ‹?q ∈ S› and ‹a ∈ hyp2› and ‹b ∈ 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_ℝ (hyp2-rep a) (hyp2-rep b) (hyp2-rep c) by (unfold real-hyp2-B-def)
  with ‹is-K2-isometry J›
  have B_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 ∈ hyp2
    and apply-cltn2 (Rep-hyp2 b) J ∈ hyp2
    and apply-cltn2 (Rep-hyp2 c) J ∈ 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*])

9.10 Perpendicularity

definition *M-perp* :: *proj2-line* ⇒ *proj2-line* ⇒ *bool* **where**
M-perp l m ≜ *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 $\exists$ *J*. *is-K2-isometry J*
$\wedge$ *apply-cltn2-line equator J* $= l \wedge$ *apply-cltn2-line meridian J* $= m$
proof −
from ⟨$a \in hyp2$⟩ **and** ⟨$b \in hyp2$⟩ **have** $a \in K2$ **and** $b \in K2$ **by** (*unfold hyp2-def*)

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 $\exists$ *r*. $r \in S \wedge r \notin \{p,q\} \wedge$ *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 ≠ 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 ∈ S by (simp add: incident-own-polar-in-S)
    with ⟨p ∈ S⟩ and ⟨q ∈ 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 ≠ q⟩ show False by simp
  qed
  with ⟨r ∈ S⟩ and ⟨proj2-incident r m⟩
  show ∃ r. r ∈ S ∧ r ∉ {p,q} ∧ proj2-incident r m
    by (simp add: exI [of - r])
next
  assume ¬ proj2-incident p m

  from ⟨b ∈ K2⟩ and ⟨proj2-incident b m⟩
    and line-through-K2-intersect-S-again [of b m]
  obtain r where r ∈ S and r ≠ q and proj2-incident r m by auto

  from ⟨¬ proj2-incident p m⟩ and ⟨proj2-incident r m⟩ have r ≠ p by auto
  with ⟨r ∈ S⟩ and ⟨r ≠ q⟩ and ⟨proj2-incident r m⟩
  show ∃ r. r ∈ S ∧ r ∉ {p,q} ∧ proj2-incident r m
    by (simp add: exI [of - r])
qed
then obtain r where r ∈ S and r ∉ {p,q} and proj2-incident r m by auto

from ⟨p ∈ S⟩ and ⟨q ∈ S⟩ and ⟨r ∈ S⟩ and ⟨p ≠ q⟩ and ⟨r ∉ {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 $\exists$ *J*. *is-K2-isometry J*
$\wedge$ *apply-cltn2-line equator J* = *l* $\wedge$ *apply-cltn2-line meridian J* = *m*
by (*simp add*: *exI* [*of - J*])
qed

definition *drop-perp* :: *proj2* $\Rightarrow$ *proj2-line* $\Rightarrow$ *proj2-line* **where**
drop-perp p l $\triangleq$ *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* $\Rightarrow$ *proj2-line* $\Rightarrow$ *proj2* **where**
perp-foot p l $\triangleq$ *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* $\in$ *hyp2* **and** *proj2-incident a l* **and** *b* $\in$ *hyp2*
and *proj2-incident b m* **and** *proj2-incident c l* **and** *proj2-incident c m*
shows *c* $\in$ *hyp2*
proof −
from ⟨*M-perp l m*⟩ **and** ⟨*a* $\in$ *hyp2*⟩ **and** ⟨*proj2-incident a l*⟩ **and** ⟨*b* $\in$ *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* $\in$ *hyp2*
by (*unfold hyp2-def*) (*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* $\in$ *hyp2*⟩ show *c* $\in$ *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* $\Rightarrow$ *proj2-line* $\Rightarrow$ *proj2* where
perp-up a l
$\triangleq$ *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 $\exists$ *p*. *p* $\in$ *S* $\wedge$ *proj2-incident p* (*drop-perp a l*)
by (*unfold hyp2-def*) (*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* ∈ *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 (*unfold hyp2-def*) (*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* ∈ *hyp2* **and** *b* ∈ *hyp2* **and** *c* ∈ *hyp2* **and** *a* ≠ *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* ∈ *hyp2*› **and** ‹*b* ∈ *hyp2*› **have** *?p* ∈ *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* ≠ *b*› **and** ‹*a* ∈ *hyp2*› **and** ‹*b* ∈ *hyp2*›
have *proj2-incident ?p ?l*
by (*simp-all add*: *endpoint-in-S-incident*)

from ‹*a* ∈ *hyp2*› **and** ‹*proj2-incident a ?l*› **and** ‹*c* ∈ *hyp2*›
have *?d* ∈ *hyp2* **by** (*rule perp-foot-hyp2*)
with ‹*?p* ∈ *S*› **have** *?q* ≠ *?p* **by** (*rule endpoint-in-S-S-strict-hyp2-distinct*)

from ‹*?p* ∈ *S*› **and** ‹*?d* ∈ *hyp2*› **have** *?q* ∈ *S* **by** (*simp add*: *endpoint-in-S*)

from ‹*?d* ∈ *hyp2*› **and** ‹*?p* ∈ *S*›
have *?p* ≠ *?d* **by** (*rule hyp2-S-not-equal* [*symmetric*])
with ‹*?p* ∈ *S*› **and** ‹*?d* ∈ *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* ≠ *b*› **and** ‹*a* ∈ *hyp2*› **and** ‹*b* ∈ *hyp2*› **and** ‹*?q* ∈ *S*›
and ‹*proj2-incident a ?l*› **and** ‹*proj2-incident b ?l*›
have *?q* = *?p* ∨ *?q* = *endpoint-in-S b a*
by (*simp add*: *endpoints-in-S-incident-unique*)
with ‹*?q* ≠ *?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 −
  def $p \triangleq$ *endpoint-in-S a b*
    and $q \triangleq$ *endpoint-in-S b a*
    and $d \triangleq$ *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
  $\triangleq p \in S \wedge q \in S \wedge a \in hyp2$
  $\wedge$ *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 −
  def $c \triangleq$ *perp-foot a l*
  def $q \triangleq$ *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 ∈ S⟩ and ⟨q ∈ S⟩ have c ≠ p and c ≠ q
    by (simp-all add: hyp2-S-not-equal)

  from ⟨c ≠ 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)

  def m ≜ drop-perp a l
  from ⟨a ∈ hyp2⟩ and ⟨b ∈ hyp2⟩ and ⟨proj2-incident b l⟩
  have proj2-incident q m by (unfold q-def m-def) (rule perp-up-incident)
  with ⟨c ≠ 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 ∈ S⟩ and ⟨q ∈ S⟩ and ⟨c ∈ hyp2⟩ and drop-perp-perp [of l a]
    and ⟨l = proj2-line-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 ∈ hyp2 and b ∈ 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 ∈ hyp2⟩ and ⟨proj2-incident a l⟩
  have pole l ∉ hyp2 by (rule line-through-hyp2-pole-not-in-hyp2)
  with ⟨b ∈ hyp2⟩ have b ≠ 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 ∈ hyp2 and b ∈ 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 ∈ hyp2⟩ and ⟨proj2-incident a l⟩ and ⟨b ∈ hyp2⟩
  have perp-foot b l ∈ 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 ∈ hyp2⟩ and ⟨perp-foot b l ∈ 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 ∃ J. is-K2-isometry J ∧ apply-cltn2 p J = east
  ∧ apply-cltn2 a J = K2-centre ∧ apply-cltn2 q J = north
proof −
  from ⟨right-angle p a q⟩
  have p ∈ S and q ∈ S and a ∈ 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 ∈ 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 ∈ S⟩ and ⟨q ∈ S⟩
  have apply-cltn2 p ?J'' ∈ S and apply-cltn2 q ?J'' ∈ 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 ∨ apply-cltn2 p ?J'' = west
```

and *apply-cltn2 q ?J′′ = north ∨ apply-cltn2 q ?J′′ = south*
by (*simp-all add*: *line-S-two-intersections-only*)

have ∃ *J′. is-K2-isometry J′ ∧ apply-cltn2 p J′ = east*
∧ *apply-cltn2 a J′ = K2-centre*
∧ (*apply-cltn2 q J′ = north ∨ 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 ∨ apply-cltn2 q ?J′′ = south*⟩
show ∃ *J′. is-K2-isometry J′ ∧ apply-cltn2 p J′ = east*
∧ *apply-cltn2 a J′ = K2-centre*
∧ (*apply-cltn2 q J′ = north ∨ apply-cltn2 q J′ = south*)
by (*simp add*: *exI* [*of* - *?J′′*])
next
assume *apply-cltn2 p ?J′′ ≠ east*
with ⟨*apply-cltn2 p ?J′′ = east ∨ 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 ∨ 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 ∨ apply-cltn2 q ?J′ = south*
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 ∨ 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 ∨ apply-cltn2 q J′ = south*
by *auto*

show ∃ *J. is-K2-isometry J ∧ apply-cltn2 p J = east*
∧ *apply-cltn2 a J = K2-centre ∧ 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 ∃ *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′*])

```
  next
    assume apply-cltn2 q J′ ≠ north
    with ⟨apply-cltn2 q J′ = north ∨ 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

9.11 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** (*unfold hyp2-def*, *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 ≠ 0⟩ and ⟨b ≠ 0⟩ and ⟨?pa ≠ ?pb⟩ and ⟨?pa ∈ hyp2⟩
  have ?Si1 ∈ S and ?Si2 ∈ S
    by (unfold hyp2-def, simp-all add: S-intersections-in-S)
  with ⟨?pa ∈ hyp2⟩ and ⟨?pb ∈ hyp2⟩
  have ?Si1 ≠ ?pa and ?Si2 ≠ ?pa and ?Si1 ≠ ?pb and ?Si2 ≠ ?pb
    by (simp-all add: hyp2-S-not-equal [symmetric])
  with ⟨proj2-set-Col {?pa,?pb,?Si1,?Si2}⟩ and ⟨?pa ≠ ?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 ≠ 0⟩ and ⟨b ≠ 0⟩ and ⟨?pa ≠ ?pb⟩ and ⟨?pa ∈ hyp2⟩
  have ?Si1 ≠ ?Si2 by (unfold hyp2-def, 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 ≠ 0⟩ and ⟨proj2-abs a ∈ hyp2⟩
  have a · (M *v a) < 0 by (unfold hyp2-def, subst K2-abs [symmetric])
  with ⟨a ≠ 0⟩ and ⟨b ≠ 0⟩ and ⟨?pa ≠ ?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
  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* ≠ *0*⟩ **and** ⟨*b* ≠ *0*⟩ **and** ⟨*?pa* ≠ *?pb*⟩ **and** ⟨*?pa* ∈ *hyp2*⟩ **and** ⟨*?pb* ∈ *hyp2*⟩
have (*?Si1* = *endpoint-in-S ?pa ?pb* ∧ *?Si2* = *endpoint-in-S ?pb ?pa*)
∨ (*?Si2* = *endpoint-in-S ?pa ?pb* ∧ *?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* ≠ *?pb*⟩
show *?e2d* = (*?aMb* + *?sqd*) / (*?aMb* − *?sqd*)
∨ *?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** (*unfold hyp2-def*, *simp add*: *quarter-discrim-positive*)
thus *?qd* ≥ *0* **by** *simp*
qed
with *real-sqrt-pow2* [*of ?qd*] **have** $?sqd^2$ = *?qd* **by** *simp*
hence (*?aMb* + *?sqd*) ∗ (*?aMb* − *?sqd*) = *?aMa* ∗ *?bMb*
by (*unfold quarter-discrim-def*, *simp add*: *algebra-simps square-expand*)

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*: *square-expand*)
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*: *square-expand*)
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 ‹$a \neq 0$› **and** ‹$b \neq 0$› **and** ‹$?pa \in hyp2$› **and** ‹$?pb \in hyp2$›
have $?aMa < 0$ **and** $?bMb < 0$
by (*unfold hyp2-def*, *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 $?aMb + ?sqd \neq 0$ **and** $?aMb - ?sqd \neq 0$ **by** *auto*
hence *sgn* $(?aMb + ?sqd) \in \{-1,1\}$ **and** *sgn* $(?aMb - ?sqd) \in \{-1,1\}$
by (*simp-all add*: *real-sgn-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* $(?aMb + ?sqd) \in \{-1,1\}$› **and** ‹*sgn* $(?aMb - ?sqd) \in \{-1,1\}$›
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] ≠ (0::real^3) (is ?a ≠ 0)
    and vector [0,y,1] ≠ (0::real^3) (is ?b ≠ 0)
    by (unfold vector-def, simp-all add: Cart-eq forall-3)

  have ?a · (M ∗v ?a) = x^2 − 1 (is ?aMa = x^2 − 1)
    and ?b · (M ∗v ?b) = y^2 − 1 (is ?bMb = y^2 − 1)
    unfolding vector-def and M-def and inner-vector-def
      and matrix-vector-mult-def
    by (simp-all add: setsum-3 square-expand)
  with ⟨|x| < 1⟩ and ⟨|y| < 1⟩
  have ?aMa < 0 and ?bMb < 0 by (simp-all add: less-one-imp-sqr-less-one)
  hence ?pa ∈ hyp2 and ?pb ∈ hyp2
    by (unfold hyp2-def, simp-all add: M-neg-imp-K2)
  with ⟨?a ≠ 0⟩ and ⟨?b ≠ 0⟩
  have cosh-dist ?pa ?pb = |?a · (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 ... = |?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-vector-def
      and matrix-vector-mult-def
    by (simp add: setsum-3 algebra-simps)

  let ?o = vector [0,0,1]
  let ?oMa = ?o · (M ∗v ?a)
  let ?oMb = ?o · (M ∗v ?b)
  let ?oMo = ?o · (M ∗v ?o)
  from K2-centre-non-zero and ⟨?a ≠ 0⟩ and ⟨?b ≠ 0⟩
    and K2-centre-in-K2 and ⟨?pa ∈ hyp2⟩ and ⟨?pb ∈ 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 hyp2-def 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-vector-def
      and matrix-vector-mult-def
    by (simp-all add: setsum-3 square-expand)
  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 ∈ hyp2 and b ∈ hyp2 and is-K2-isometry J and a ≠ 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 ≠ b› and ‹?aJ ≠ ?bJ›
  show exp-2dist ?aJ ?bJ = exp-2dist a b by (unfold exp-2dist-def) simp
qed

lemma K2-isometry-cosh-dist:
  assumes a ∈ hyp2 and b ∈ 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 ∈ hyp2 and b ∈ hyp2 and c ∈ 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 ∈ 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 ∈ hyp2› and ‹c ∈ hyp2›
  have ?aJi ∈ hyp2 and ?cJi ∈ hyp2
```

```
    by (unfold hyp2-def) (simp-all add: statement60-one-way)

  from ‹?aJi ∈ 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 ∈ 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 ∈ hyp2› and ‹b ∈ hyp2› and ‹c ∈ 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 Bℝ (cart2-pt a) (cart2-pt b) (cart2-pt p) (is Bℝ ?ca ?cb ?cp)
  shows cross-ratio p q a b ≥ 1
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 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 ∈ hyp2› and ‹p ∈ S› have b ≠ p by (rule hyp2-S-not-equal)
   with ‹z-non-zero a› and ‹z-non-zero b› and ‹z-non-zero p›
     and ‹Bℝ ?ca ?cb ?cp› and cart2-append1-between-right-strict [of a b p]
   obtain j where j ≥ 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 add*: *mult-pos-pos*)

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* ≥ 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* $\in$ *S*⟩ **and** ⟨*a* $\in$ *hyp2*⟩ **and** ⟨*b* $\in$ *hyp2*⟩
have $B_{\mathbb{R}}$ *?cp ?ca ?cb* $\vee$ $B_{\mathbb{R}}$ *?cp ?cb ?ca* **by** (*simp add*: *S-at-edge*)
with ⟨$\neg$ $B_{\mathbb{R}}$ *?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* $\geq$ *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* $\neq$ *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 *a* $\neq$ *p* **and** *a* $\neq$ *q* **and** *b* $\neq$ *p* **and** *b* $\neq$ *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 ≠ 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)
  hence cross-ratio ?p ?q a b > 0 by (rule cross-ratio-S-S-hyp2-hyp2-positive)
  with ⟨a ≠ b⟩ show exp-2dist a b > 0 by (unfold exp-2dist-def) simp
qed

lemma cosh-dist-at-least-1:
  assumes a ∈ hyp2 and b ∈ 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 ∈ hyp2 and b ∈ 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 ∈ hyp2 and b ∈ hyp2 and c ∈ 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 ∈ hyp2⟩ and ⟨a ∈ 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** $?a' \neq ?b'$ **and** $?c' \neq ?d'$ **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 ⟨$?a' \neq ?b'$⟩ **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 ⟨$?a' \neq ?b'$⟩
have *proj2-set-Col* {*?p*,*?q*,*?a′*,*?b′*}
by (*simp add*: *Rep-hyp2 proj2-set-Col-endpoints-in-S*)

from ⟨$?a' \neq ?b'$⟩ **have** $?p \neq ?q$ **by** (*simp add*: *Rep-hyp2 endpoint-in-S-swap*)

have $?p \in S$ **by** (*simp add*: *Rep-hyp2 endpoint-in-S*)
hence $?a' \neq ?p$ **and** $?b' \neq ?p$ **by** (*simp-all add*: *Rep-hyp2 hyp2-S-not-equal*)
with ⟨*proj2-set-Col* {*?p*,*?q*,*?a′*,*?b′*}⟩ **and** ⟨$?p \neq ?q$⟩
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*] *square-expand minus-mult-left*)
  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-divide-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 square-expand)

  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: square-expand 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 $p \neq q$ **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 $a \neq p$ **and** $b \neq p$ **and** $a \neq q$ **by** (*simp-all add*: *hyp2-S-not-equal*)
with ⟨*proj2-set-Col* $\{p,q,a,b\}$⟩ **and** ⟨$p \neq q$⟩
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 ∈ hyp2› and ‹b ∈ hyp2› and ‹c ∈ hyp2› and ‹p ∈ 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_ℝ ?ca ?cb ?cp› and ‹B_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 ∈ hyp2› and ‹p ∈ S› have a ≠ 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 ∈ hyp2› and ‹b ∈ hyp2› and ‹c ∈ hyp2› and ‹p ∈ 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_ℝ ?ca ?cb ?cp› and ‹B_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 ∈ hyp2 and b ∈ 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 ∈ hyp2› and ‹a ∈ 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** $exp\text{-}2dist\ a\ b = 1$
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 $exp\text{-}2dist\ a\ b = cross\text{-}ratio\ ?p\ ?q\ a\ b$ **by** (*unfold exp-2dist-def*) *simp*
with ⟨$exp\text{-}2dist\ a\ b = 1$⟩ **have** $cross\text{-}ratio\ ?p\ ?q\ a\ b = 1$ **by** *simp*
ultimately have $a = b$ **by** (*rule cross-ratio-1-equal*)
with ⟨$a \neq b$⟩ **show** *False* **..**
qed

9.11.1 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*
$= (cosh\text{-}dist\ b\ c * sqrt\ (cross\text{-}ratio\ p\ q\ a\ b) - cosh\text{-}dist\ a\ c)$
$/\ (cosh\text{-}dist\ a\ c * cross\text{-}ratio\ p\ q\ a\ b$
$- cosh\text{-}dist\ b\ c * sqrt\ (cross\text{-}ratio\ p\ q\ a\ b))$
(**is** $?pqda = (?bc * sqrt\ ?pqab - ?ac)\ /\ (?ac * ?pqab - ?bc * sqrt\ ?pqab)$)
proof −
let $?da = cosh\text{-}dist\ d\ a$
let $?db = cosh\text{-}dist\ d\ b$
let $?dc = cosh\text{-}dist\ d\ c$
let $?pqdb = cross\text{-}ratio\ p\ q\ d\ b$

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 ⟨$p \neq q$⟩ **and** ⟨$p \in S$⟩ **and** ⟨$q \in S$⟩ **and** ⟨$d \in hyp2$⟩ **and** ⟨$a \in hyp2$⟩
and ⟨$b \in hyp2$⟩
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 ⟨$a \in hyp2$⟩ **and** ⟨$b \in hyp2$⟩ **and** ⟨$p \in S$⟩ **and** ⟨$q \in S$⟩
have $a \neq p$ **and** $a \neq q$ **and** $b \neq p$ **by** (*simp-all add*: *hyp2-S-not-equal*)

from ⟨*proj2-Col p q a*⟩ **and** ⟨$p \neq q$⟩ **and** ⟨$a \neq p$⟩ **and** ⟨$a \neq q$⟩
have *?pqdb* = *?pqda* ∗ *?pqab* **by** (*rule cross-ratio-product* [*symmetric*])

from ⟨*M-perp l m*⟩ **and** ⟨$a \in hyp2$⟩ **and** ⟨$b \in hyp2$⟩ **and** ⟨$c \in hyp2$⟩ **and** ⟨$d \in hyp2$⟩
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 square-expand)
  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: mult-pos-pos 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 ‹$a \neq b$› **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*)

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* **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*
$= (?bc * sqrt\ (cross\text{-}ratio\ ?p\ ?q\ a\ b) - ?ac)$
$/\ (?ac * (cross\text{-}ratio\ ?p\ ?q\ a\ b) - ?bc * sqrt\ (cross\text{-}ratio\ ?p\ ?q\ a\ b))$
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

9.12 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 $\exists$ *J. is-K2-isometry J*
$\wedge$ *hyp2-cltn2 a J* $= a'$ $\wedge$ *hyp2-cltn2 b J* $= b'$ $\wedge$ *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 $\exists$ *J. is-K2-isometry J*
$\wedge$ *hyp2-cltn2 a J* $= a'$ $\wedge$ *hyp2-cltn2 b J* $= b'$ $\wedge$ *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′
def 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′
def 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 $\langle pp' \neq pq' \rangle$ **and** $\langle pp' \in S \rangle$ **and** $\langle pq' \in S \rangle$ **and** $\langle pd' \in hyp2 \rangle$
and $\langle ?paJ \in hyp2 \rangle$ **and** $\langle ?pbJ \in hyp2 \rangle$
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 $\langle ?ppJ = pp' \rangle$ **and** $\langle ?pqJ = pq' \rangle$
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** $\langle pp \neq pq \rangle$ **and** $\langle ?pa \neq pp \rangle$
and $\langle ?pb \neq pp \rangle$
have $\ldots =$ *cross-ratio pp pq ?pa ?pb* **by** (*rule cross-ratio-cltn2*)
also from $\langle a \neq b \rangle$ **and** $\langle a\ b \equiv_K a'\ b' \rangle$
have $\ldots =$ *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 ⟨$pd \neq ps$⟩ **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** ⟨$pd' \in hyp2$⟩ **and** ⟨$?pcJ \in hyp2$⟩
  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 ⟨$pd' \in hyp2$⟩ **and** ⟨$?pcJ \in hyp2$⟩ **and** ⟨$?pc' \in hyp2$⟩ **and** ⟨$ps' \in S$⟩
  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*:
  $\forall\ a\ b\ c\ d\ a'\ b'\ c'\ d'.$
  $a \neq b \wedge B_K\ a\ b\ c \wedge B_K\ a'\ b'\ c' \wedge a\ b \equiv_K a'\ b' \wedge b\ c \equiv_K b'\ c'$
    $\wedge\ a\ d \equiv_K a'\ d' \wedge b\ d \equiv_K b'\ d'$
  $\longrightarrow c\ d \equiv_K c'\ d'$
proof *default+*
  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 $\langle a\ b \equiv_K a'\ b'\rangle$ **and** $\langle b\ d \equiv_K b'\ d'\rangle$ **and** $\langle a\ d \equiv_K a'\ d'\rangle$ **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 $\langle a \neq b\rangle$ **and** $\langle a\ b \equiv_K a'\ b'\rangle$
  have $a' \neq b'$ **by** (*auto simp add*: *hyp2.A3'*)

  from ⟨*is-K2-isometry J*⟩ **and** $\langle B_K\ a\ b\ c\rangle$
  have $B_K\ ?aJ\ ?bJ\ ?cJ$ **by** (*rule real-hyp2-B-hyp2-cltn2*)
  hence $B_K\ a'\ b'\ ?cJ$ **by** (*unfold* $\langle ?aJ = a'\rangle$ $\langle ?bJ = b'\rangle$)

  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* $\langle ?bJ = b'\rangle$)
  from *this* **and** $\langle b\ c \equiv_K b'\ c'\rangle$ **have** $b'\ ?cJ \equiv_K b'\ c'$ **by** (*rule hyp2.A2'*)
  with $\langle a' \neq b'\rangle$ **and** $\langle B_K\ a'\ b'\ ?cJ\rangle$ **and** $\langle B_K\ a'\ b'\ c'\rangle$
  have $?cJ = c'$ **by** (*rule hyp2-extend-segment-unique*)
  from ⟨*is-K2-isometry J*⟩
  show $c\ d \equiv_K c'\ d'$
    unfolding $\langle ?cJ = c'\rangle$ [*symmetric*] **and** $\langle ?dJ = d'\rangle$ [*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*

9.13 The Klein–Beltrami model satisfies axioms 6, 7, and 11

```
theorem hyp2-axiom6: ∀ a b. B_K a b a ⟶ a = b
proof default+
  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_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_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_ℝ ?u v ?w⟩
  obtain l where l ≥ 0 and l ≤ 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 ≥ 0⟩ and ⟨l ≤ 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 ≠ 0
    with ⟨norm ?w < 1⟩ and ⟨l ≥ 0⟩ have l *_R norm ?w < l by simp

    with ⟨norm ?u < 1⟩ and ⟨l ≤ 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 $?cX = hyp2\text{-}rep\ `\ X$
and $?cY = hyp2\text{-}rep\ `\ Y$
and $?ca = hyp2\text{-}rep\ a$
and $?cw = hyp2\text{-}rep\ w$
and $?cz = hyp2\text{-}rep\ z$

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 $\langle w \in X\rangle$ **and** $\langle z \in Y\rangle$ **have** $B_{\mathbb{R}}\ ?cw\ cb\ ?cz$ **by** *simp*
hence $hyp2\text{-}rep\ (hyp2\text{-}abs\ cb) = cb$ (**is** $hyp2\text{-}rep\ ?b = cb$)
by (*rule between-inverse*)
with $\langle\forall\ cx\ cy.\ cx \in ?cX \wedge cy \in ?cY \longrightarrow B_{\mathbb{R}}\ cx\ cb\ cy\rangle$
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*

9.14 The Klein–Beltrami model satisfies the dimension-specific axioms

lemma *hyp2-rep-abs-examples*:
shows $hyp2\text{-}rep\ (hyp2\text{-}abs\ 0) = 0$ (**is** $hyp2\text{-}rep\ ?a = ?ca$)
and $hyp2\text{-}rep\ (hyp2\text{-}abs\ (vector\ [1/2,0])) = vector\ [1/2,0]$
(**is** $hyp2\text{-}rep\ ?b = ?cb$)
and $hyp2\text{-}rep\ (hyp2\text{-}abs\ (vector\ [0,1/2])) = vector\ [0,1/2]$
(**is** $hyp2\text{-}rep\ ?c = ?cc$)
and $hyp2\text{-}rep\ (hyp2\text{-}abs\ (vector\ [1/4,1/4])) = vector\ [1/4,1/4]$
(**is** $hyp2\text{-}rep\ ?d = ?cd$)
and $hyp2\text{-}rep\ (hyp2\text{-}abs\ (vector\ [1/2,1/2])) = vector\ [1/2,1/2]$
(**is** $hyp2\text{-}rep\ ?t = ?ct$)
proof −
have $norm\ ?ca < 1$ **and** $norm\ ?cb < 1$ **and** $norm\ ?cc < 1$ **and** $norm\ ?cd < 1$
and $norm\ ?ct < 1$
by (*unfold norm-vector-def setL2-def*) (*simp-all add*: *setsum-2 square-expand*)
thus $hyp2\text{-}rep\ ?a = ?ca$ **and** $hyp2\text{-}rep\ ?b = ?cb$ **and** $hyp2\text{-}rep\ ?c = ?cc$
and $hyp2\text{-}rep\ ?d = ?cd$ **and** $hyp2\text{-}rep\ ?t = ?ct$
by (*simp-all add*: *hyp2-rep-abs*)
qed

```
theorem hyp2-axiom8: ∃ a b c. ¬ B_K a b c ∧ ¬ B_K b c a ∧ ¬ 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 ∃ 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:
  ∀ p q a b c. p ≠ q ∧ a p ≡_K a q ∧ b p ≡_K b q ∧ c p ≡_K c q
  ⟶ 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 ≠ q ∧ a p ≡_K a q ∧ b p ≡_K b q ∧ c p ≡_K c q
    ⟶ B_K a b c ∨ B_K b c a ∨ B_K c a b
  proof
    assume p ≠ q ∧ a p ≡_K a q ∧ b p ≡_K b q ∧ c p ≡_K c q
    hence p ≠ q and a p ≡_K a q and b p ≡_K b q and c p ≡_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
    def l ≜ proj2-line-through ?pp ?pq
    def 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 ≠ 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 ∀ 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

  def 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

lemma True ..
```

9.15 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*: *Cart-eq*)
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*: *Cart-eq 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*: *Cart-eq 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 $?cx\$1 \leq |?cx\$1|$ **by** *simp*
also have $\ldots \leq$ *norm ?cx* **by** (*rule component-le-norm*)
also have $\ldots < 1$ **by** (*rule norm-hyp2-rep-lt-1*)
finally have $?cx\$1 < 1$ **.**
with $\langle l \leq 1 \rangle$ **and** *mult-less-cancel-left* [*of* $1 - l$ *?cx$1 1*]
have $(1 - l) * ?cx\$1 \leq 1 - l$ **by** *auto*
with $\langle (1 - l) * (?cx\$1) = 1/2 \rangle$ **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*)
also have $\ldots < 1$ **by** (*rule norm-hyp2-rep-lt-1*)
finally have $?cy\$2 < 1$ **.**
with $\langle l \geq 0 \rangle$ **and** *mult-less-cancel-left* [*of l ?cy$2 1*]
have $l * ?cy\$2 \leq l$ **by** *auto*
with $\langle l * (?cy\$2) = 1/2 \rangle$ **have** $l \geq 1/2$ **by** *simp*
with $\langle l \leq 1/2 \rangle$ **have** $l = 1/2$ **by** *simp*
with $\langle l * (?cy\$2) = 1/2 \rangle$ **have** $?cy\$2 = 1$ **by** *simp*
with $\langle ?cy\$2 < 1 \rangle$ **show** *False* **by** *simp*
qed

theorem *hyp2-not-tarski*: $\neg$ (*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.

For some reason, because I extract the LaTeX source for the above theorem, I must write the following before the end, in order for the outline to typeset.

lemma *True* **..**

end

References

[1] Borsuk, K., and Szmielew, W. *Foundations of Geometry: Euclidean and Bolyai-Lobachevskian Geometry; Projective Geometry*. North-Holland Publishing Company, 1960. Translated from Polish by Erwin Marquit.

[2] Schwabhäuser, W., Szmielew, W., and Tarski, A. *Metamathematische Methoden in der Geometrie*. Springer-Verlag, 1983.