# No Faster-Than-Light Observers (GenRel)

Mike Stannett*, Edward Higgins
University of Sheffield, UK

Hajnal Andréka, Judit Madarász, István Németi, Gergely Székely†
Alfréd Rényi Institute of Mathematics, Budapest, Hungary
(† Secondary affiliation: University of Public Service, Budapest, Hungary)

April 18, 2024

**Abstract**

We have previously verified, in the first order theory SpecRel of Special Relativity, that inertial observers cannot travel faster than light [1, 2]. We now prove the corresponding result for GenRel, the first-order theory of General Relativity. Specifically, we prove that whenever an observer $m$ encounters another observer $k$ (so that $m$ and $k$ are both present at some spacetime location $x$), $k$ will necessarily be observed by $m$ to be traveling at less than light speed.

## Contents



*Corresponding author: m.stannett@sheffield.ac.uk

# 1 Sorts

GenRel is a 2-sorted first-order logic. This theory introduces the two sorts and proves a number of basic arithmetical results. The two sorts are Bodies (things that can move) and Quantities (used to specify coordinates, masses, etc).

**theory** *Sorts*
  **imports** *Main*
**begin**

## 1.1 Bodies

There are two types of Body: photons and observers. We do not assume a priori that these sorts are disjoint.

**record** *Body* =
  *Ph* :: *bool*
  *Ob* :: *bool*

## 1.2 Quantities

The quantities are assumed to form a linearly ordered field. We may sometimes need to assume that the field is also Euclidean, i.e. that square roots exist, but this is not a general requirement so it will be added later using a separate axiom class, AxEField.

**class** *Quantities* = *linordered-field*
**begin**

**abbreviation** *inRangeOO* :: $'a \Rightarrow 'a \Rightarrow 'a \Rightarrow bool$ (- < - < -)
  **where** $(a < b < c) \equiv (a < b) \wedge (b < c)$

**abbreviation** *inRangeOC* :: $'a \Rightarrow 'a \Rightarrow 'a \Rightarrow bool$ (- < - ≤ -)
  **where** $(a < b \leq c) \equiv (a < b) \wedge (b \leq c)$

**abbreviation** *inRangeCO* :: $'a \Rightarrow 'a \Rightarrow 'a \Rightarrow bool$ (- ≤ - < -)
  **where** $(a \leq b < c) \equiv (a \leq b) \wedge (b < c)$

**abbreviation** *inRangeCC* :: $'a \Rightarrow 'a \Rightarrow 'a \Rightarrow bool$ (- ≤ - ≤ -)

**where** $(a \leq b \leq c) \equiv (a \leq b) \wedge (b \leq c)$

**lemma** *lemLEPlus*: $a \leq b + c \longrightarrow c \geq a - b$
**by** (*simp add*: *add-commute local.diff-le-eq*)

**lemma** *lemMultPosLT1*:
**assumes** $(a > 0) \wedge (b \geq 0) \wedge (b < 1)$
**shows** $(a * b) < a$
**using** *assms local.mult-less-cancel-left2 local.not-less* **by** *auto*

**lemma** *lemAbsRange*: $e > 0 \longrightarrow ((a-e) < b < (a+e)) \longleftrightarrow (abs\ (b-a) < e)$
**by** (*simp add*: *local.abs-diff-less-iff*)

**lemma** *lemAbsNeg*: $abs\ x = abs\ (-x)$
**by** *simp*

**lemma** *lemAbsNegNeg*: $abs\ (-a-b) = abs\ (a+b)$
**using** *add-commute local.abs-minus-commute* **by** *auto*

**lemma** *lemGENZGT*: $(x \geq 0) \wedge (x \neq 0) \longrightarrow x > 0$
**by** *auto*

**lemma** *lemLENZLT*: $(x \leq 0) \wedge (x \neq 0) \longrightarrow x < 0$
**by** *force*

**lemma** *lemSumOfNonNegAndPos*: $x \geq 0 \wedge y > 0 \longrightarrow x+y > 0$
**by** (*simp add*: *local.add-strict-increasing2*)

**lemma** *lemSumOfTwoHalves*: $x = x/2 + x/2$
**using** *mult-2*[*of* $x/2$] **by** *force*

**lemma** *lemDiffDiffAdd*: $(b-a)+(c-b) = (c-a)$
**by** (*auto simp add*: *field-simps*)

**lemma** *lemSumDiffCancelMiddle*: $(a - b) + (b - c) = (a - c)$
**by** (*auto simp add*: *field-simps*)

**lemma** *lemDiffSumCancelMiddle*: $(a - b) + (b + c) = (a + c)$
**by** (*auto simp add*: *field-simps*)

**lemma** *lemMultPosLT*: $((0 < a) \wedge (b < c)) \longrightarrow (a*b < a*c)$
**using** *mult-strict-left-mono* **by** *auto*

**lemma** *lemMultPosLE*: $((0 < a) \wedge (b \leq c)) \longrightarrow (a*b \leq a*c)$
**using** *mult-left-mono* **by** *auto*

**lemma** *lemNonNegLT*: $((0 \leq a) \wedge (b < c)) \longrightarrow (a{*}b \leq a{*}c)$
  **using** *mult-left-mono* **by** *auto*

**lemma** *lemMultNonNegLE*: $((0 \leq a) \wedge (b \leq c)) \longrightarrow (a{*}b \leq a{*}c)$
  **using** *mult-left-mono* **by** *auto*

**abbreviation** *sqr* :: $'a \Rightarrow 'a$
  **where** $sqr\ x \equiv\ x{*}x$

**abbreviation** *hasRoot* :: $'a \Rightarrow bool$
  **where** $hasRoot\ x \equiv \exists\ r\ .\ x = sqr\ r$

**abbreviation** *isNonNegRoot* :: $'a \Rightarrow 'a \Rightarrow bool$
  **where** $isNonNegRoot\ x\ r \equiv (r \geq 0) \wedge (x = sqr\ r)$

**abbreviation** *hasUniqueRoot* :: $'a \Rightarrow bool$
  **where** $hasUniqueRoot\ x \equiv \exists!\ r\ .\ isNonNegRoot\ x\ r$

**abbreviation** *sqrt* :: $'a \Rightarrow 'a$
  **where** $sqrt\ x \equiv\ THE\ r\ .\ isNonNegRoot\ x\ r$

**lemma** *lemAbsIsRootOfSquare*: $isNonNegRoot\ (sqr\ x)\ (abs\ x)$
  **by** *simp*

**lemma** *lemSqrt*:
  **assumes** *hasRoot x*
  **shows** *hasUniqueRoot x*
**proof** −
  **obtain** *r* **where** $x = sqr\ r$ **using** *assms(1)* **by** *auto*
  **define** *rt* **where** $rt = (if\ (r \geq 0)\ then\ r\ else\ (-r))$
  **hence** *rt*: $rt \geq 0 \wedge sqr\ rt = x$ **using** *rt-def* ‹$x = sqr\ r$› **by** *auto*
  **hence** *rtroot*: *isNonNegRoot x rt* **by** *auto*

  **{ fix** *y*
    **assume** *yprops*: *isNonNegRoot x y*
    **hence** $y = rt$
      **using** *local.square-eq-iff rt* **by** *auto*
    **hence** $((\ y \geq 0) \wedge (x = sqr\ \ y)) \longrightarrow (y = rt)$ **by** *auto*
  **}**
  **hence** *rtunique*: $\forall\ y\ .\ isNonNegRoot\ x\ y \longrightarrow (y = rt)$ **by** *auto*
  **thus** *?thesis* **using** *rtroot* **by** *auto*
**qed**

**lemma** *lemSqrMonoStrict*: **assumes** $(0 \leq u) \wedge (u < v)$

**shows** (*sqr u*) < (*sqr v*)
**proof** −
  **have** *1*: (*u*∗*u*) ≤ (*u*∗*v*)  **using** *assms comm-mult-left-mono* **by** *auto*
  **have** (*u*∗*v*) < (*v*∗*v*)
    **using** *assms mult-commute comm-mult-strict-left-mono* **by** *auto*
  **thus** *?thesis* **using** *1 le-less-trans* **by** *auto*
**qed**

**lemma** *lemSqrMono*: (*0* ≤ *u*) ∧ (*u* ≤ *v*) ⟶ (*sqr u*) ≤ (*sqr v*)
  **by** (*simp add*: *local.mult-mono′*)

**lemma** *lemSqrOrderedStrict*: (*v* > *0*) ∧ (*sqr u* < *sqr v*) ⟶ (*u* < *v*)
  **using** *mult-mono*[*of v u v u*] **by** *force*

**lemma** *lemSqrOrdered*: (*v* ≥ *0*) ∧ (*sqr u* ≤ *sqr v*) ⟶ (*u* ≤ *v*)
  **using** *mult-strict-mono*[*of v u v u*] **by** *force*

**lemma** *lemSquaredNegative*: *sqr x* = *sqr* (−*x*)
  **by** *auto*

**lemma** *lemSqrDiffSymmetrical*: *sqr* (*x* − *y*) = *sqr* (*y* − *x*)
  **using** *lemSquaredNegative*[*of y*−*x*] **by** *auto*

**lemma** *lemSquaresPositive*: *x* ≠ *0* ⟶ *sqr x* > *0*
  **by** (*simp add*:  *lemGENZGT*)

**lemma** *lemZeroRoot*: (*sqr x* = *0*) ⟷ (*x* = *0*)
  **by** *simp*

**lemma** *lemSqrMult*: *sqr* (*a* ∗ *b*) = (*sqr a*) ∗ (*sqr b*)
  **using** *mult-commute mult-assoc* **by** *simp*

**lemma** *lemEqualSquares*: *sqr u* = *sqr v* ⟶ *abs u* = *abs v*
 **by** (*metis local.abs-mult-less local.abs-mult-self-eq local.not-less-iff-gr-or-eq*)

**lemma** *lemSqrtOfSquare*:
  **assumes** *b* = *sqr a*
**shows**      *sqrt b* = *abs a*
**proof** −
  **have** *b* ≥ *0* **using** *assms* **by** *auto*
  **hence** *conj1*: *hasUniqueRoot b* **using** *lemSqrt*[*of b*] *assms* **by** *auto*
   **moreover have** *isNonNegRoot b* (*abs a*) **using** *lemAbsIsRootOf-Square assms* **by** *auto*
  **ultimately have** *sqrt b* = *abs a*
    **using** *theI*[*of* λ *r* . *0* ≤ *r* ∧ *b* = *sqr r abs a*] **by** *blast*
  **thus** *?thesis* **by** *auto*

**qed**

**lemma** *lemSquareOfSqrt*:
**assumes** *hasRoot b*
**and** $a = sqrt\ b$
**shows** $sqr\ a = b$
**proof** –
**obtain** *r* **where** *r*: *isNonNegRoot b r* **using** *assms(1) lemSqrt[of b]* **by** *auto*
**hence** $\forall x.\ 0 \leq x \wedge b = sqr\ x \longrightarrow x = r$ **using** *lemSqrt* **by** *blast*
**hence** $a = r$ **using** *r assms(2) the-equality[of isNonNegRoot b r]* **by** *blast*
**thus** *?thesis* **using** *r* **by** *auto*
**qed**

**lemma** *lemSqrt1*: $sqrt\ 1 = 1$
**proof** –
**have** *isNonNegRoot 1 1* **by** *auto*
**moreover have** $\forall\ r\ .\ isNonNegRoot\ 1\ r \longrightarrow r = 1$
**proof** –
**{ fix** *r* **assume** *isNonNegRoot 1 r*
**hence** *r*: $(r \geq 0) \wedge (1 = sqr\ r)$ **by** *simp*
**hence** $r = 1$ **using** *calculation lemSqrt* **by** *blast*
**}**
**thus** *?thesis* **by** *blast*
**qed**
**ultimately show** *?thesis* **using** *the-equality[of isNonNegRoot 1 1]* **by** *blast*
**qed**

**lemma** *lemSqrt0*: $sqrt\ 0 = 0$
**using** *lemZeroRoot local.mult-cancel-right1* **by** *blast*

**lemma** *lemSqrSum*: $sqr\ (x + y) = (x*x) + (2*x*y) + (y*y)$
**proof** –
**have** $x*y + y*x = x*y + x*y$ **using** *mult-commute* **by** *simp*
**also have** $... = (x+x)*y$ **using** *distrib-right* **by** *simp*
**finally have** *xy*: $x*y + y*x = 2*x*y$ **using** *mult-2* **by** *auto*

**have** $sqr\ (x+y) = x*(x+y) + y*(x+y)$ **using** *distrib-right* **by** *auto*
**also have** $... = x*x + x*y + y*x + y*y$ **using** *distrib-left add-assoc* **by** *auto*
**finally have** $sqr\ (x+y) = (sqr\ x) + x*y + y*x + (sqr\ y)$
**using** *distrib-left add-assoc* **by** *auto*

**thus** *?thesis* **using** *xy add-assoc* **by** *auto*
**qed**

**lemma** *lemQuadraticGEZero*:
**assumes** $\forall\ x.\ a{*}(sqr\ x) + b{*}x + c \geq 0$
**and** $a > 0$
**shows** $(sqr\ b) \leq 4{*}a{*}c$
**proof** −
**{ fix** $x :: 'a$
**have** $a * sqr\ (x + (b/(2{*}a))) = a * ((sqr\ x) + 2{*}(b/(2{*}a)){*}x + (sqr\ (b/(2{*}a))))$
**using** *lemSqrSum*[*of x* $(b/(2{*}a))$] *mult-assoc*
*mult-commute*[*of x* $(b/(2{*}a))$]
**by** *auto*
**hence** *1*: $a * sqr\ (x + (b/(2{*}a)))$
$= (a * (sqr\ x)) + (a{*}(2{*}(b/(2{*}a)){*}x)) + (a * sqr\ (b/(2{*}a)))$
**using** *distrib-left* **by** *auto*
**have** $a{*}(2{*}(b/(2{*}a)){*}x) = b{*}x$ **using** *mult-assoc assms*(*2*) **by** *simp*

**hence** *2*: $a * sqr\ (x + (b/(2{*}a))) = a{*}(sqr\ x) + (b{*}x) + (a * sqr\ (b/(2{*}a)))$
**using** *1* **by** *auto*

**have** $(a * sqr\ (b/(2{*}a))) = c + ((a * sqr\ (b/(2{*}a))) - c)$
**using** *add-commute diff-add-cancel* **by** *auto*
**hence** $(a * sqr\ (x + (b/(2{*}a))))$
$= (a{*}(sqr\ x) + (b{*}x) + c) + ((a * sqr\ (b/(2{*}a))) - c)$ **using** *2* *add-assoc* **by** *auto*
**hence** *3*: $(a * sqr\ (x + (b/(2{*}a)))) \geq ((a * sqr\ (b/(2{*}a))) - c)$
**using** *assms*(*1*) **by** *auto*
**}**
**hence** $\forall\ x\ .\ (a * sqr\ (x + (b/(2{*}a)))) \geq ((a * sqr\ (b/(2{*}a))) - c)$
**by** *auto*

**hence** $(a * sqr\ ((-b/(2{*}a)) + (b/(2{*}a)))) \geq ((a * sqr\ (b/(2{*}a))) - c)$ **by** *fast*
**hence** $((a * sqr\ (b/(2{*}a))) - c) \leq 0$ **by** *simp*
**hence** $4{*}a{*}((a * sqr\ (b/(2{*}a))) - c) \leq 0$
**using** *local.mult-le-0-iff assms*(*2*) **by** *auto*
**hence** $4{*}a{*}((a * sqr\ (b/(2{*}a)))) - 4{*}a{*}c \leq 0$
**using** *right-diff-distrib mult-assoc* **by** *auto*
**hence** *4*: $4{*}a{*}((a * sqr\ (b/(2{*}a)))) \leq 4{*}a{*}c$ **by** *simp*

**have** $sqr\ (b/(2{*}a)) = (sqr\ b)/(4{*}a{*}a)$
**using** *mult-assoc mult-commute* **by** *simp*
**hence** $4{*}a{*}((a * sqr\ (b/(2{*}a)))) = 4{*}a{*}((a * (sqr\ b)/(4{*}a{*}a)))$ **by**

```
auto
  hence 4∗a∗((a ∗ sqr (b/(2∗a)))) = (4∗a∗a)∗(sqr b)/(4∗a∗a)
    using mult-commute by auto
  hence 4∗a∗((a ∗ sqr (b/(2∗a)))) = (sqr b)
    using assms(2) by simp

  thus ?thesis using 4 by auto
qed


lemma lemSquareExistsAbove:
  shows ∃ x > 0 . (sqr x) > y
proof −
  have cases: (y ≤ 0) ∨ (y > 0) by auto
  have one: 1 ≥ 0 by simp
  have onestrict: 0 < 1 by simp

  { assume yle0: y ≤ 0
    hence y < sqr 1 using yle0 le-less-trans by simp
    hence ?thesis using onestrict by fast
  }
  hence case1: (y ≤ 0) ⟶ ?thesis by auto

  { assume ygt0: y > 0
    { assume ylt1: y < 1
     hence sqr y < y using ygt0 mult-strict-left-mono[of y 1] by auto
      hence sqr y < sqr 1 using ylt1 by simp
      hence y < 1 using ygt0 lemSqrOrderedStrict[of 1 y] by auto
      hence y < sqr 1 by simp
      hence ?thesis using onestrict by best
    }
    hence a: (y < 1) ⟶ ?thesis by auto
    { assume y = 1
      hence b1: y < sqr 2 by simp
      have 2 > 0 by simp
      hence ?thesis using b1 by fast
    }
    hence b: (y = 1) ⟶ ?thesis by auto
    { assume ygt1: y > 1
      hence yge1: y ≥ 1 by simp
      have yge0: y ≥ 0 using ygt0 by simp
      have y ≤ y by simp
      hence sqr y > y∗1 using lemMultPosLT ygt0 ygt1 by blast
      hence sqr y > y by simp
      hence ?thesis using ygt0 by bestsimp
    }
    hence (y > 1) ⟶ ?thesis by simp

    hence ((y<1)∨(y=1)∨(y>1)) ⟶ ?thesis using a b by auto
```

```
    hence ?thesis by fastforce
  }
  hence ypos: (y > 0) ⟶ ?thesis by auto

  thus ?thesis using cases case1 by auto
qed


lemma lemSmallSquares:
  assumes x > 0
  shows ∃ y > 0. (sqr y < x)
proof −
  have invpos: 1/x > 0 using assms(1) by auto
  then obtain z where z: (z > 0) ∧ ((sqr z) > (1/x))
    using lemSquareExistsAbove by auto
  define y where y: y = 1/z
  hence ypos: y > 0 using z by auto
  have 1: 1/(sqr z) < 1/(1/x) using z invpos
    by (meson local.divide-strict-left-mono
             local.mult-pos-pos local.zero-less-one)
  hence (sqr y) < x using z y by simp
  thus ?thesis using ypos by auto
qed


lemma lemSqrLT1:
  assumes 0 < x < 1
  shows 0 < (sqr x) < x
using assms lemMultPosLT1[of x x] by auto



lemma lemReducedBound:
  assumes x > 0
  shows ∃ y > 0 . (y < x) ∧ (sqr y < y) ∧ (y < 1)
proof −
  have x2: x > x/2
     using assms lemSumOfTwoHalves[of x]  add-strict-left-mono[of 0
x/2 x/2]
    by auto
  have x2pos: x/2 > 0 using assms by simp

  define y where y = min (x/2) (1/2)
  hence y: (y ≤ x/2) ∧ (y ≤ 1/2) ∧ (y > 0) using x2pos by auto

  have yltx: y < x using y x2 le-less-trans by auto
  have ylt1: y < 1 using y le-less-trans by auto

  hence sqr y < y using lemSqrLT1 y by simp
```

```
  thus ?thesis using yltx ylt1 y by auto
qed

end


end
```

# 2 Points

This theory defines (1+3)-dimensional spacetime points. The first coordinate is the time coordinate, and the remaining three coordinates give the spatial component.

```
theory Points
  imports Sorts
begin


record 'a Point =
  tval :: 'a
  xval :: 'a
  yval :: 'a
  zval :: 'a


record 'a Space =
  svalx :: 'a
  svaly :: 'a
  svalz :: 'a


abbreviation tComponent :: 'a Point ⇒ 'a where
  tComponent p ≡ tval p

abbreviation sComponent :: 'a Point ⇒ 'a Space where
  sComponent p ≡ (| svalx = xval p, svaly = yval p, svalz = zval p |)

abbreviation mkPoint :: 'a ⇒ 'a ⇒ 'a ⇒ 'a ⇒ 'a Point where
  mkPoint t x y z ≡ (| tval = t, xval = x, yval =y, zval = z |)

abbreviation stPoint :: 'a ⇒ 'a Space ⇒ 'a Point where
  stPoint t s ≡ mkPoint t (svalx s) (svaly s) (svalz s)

abbreviation mkSpace :: 'a ⇒ 'a ⇒ 'a ⇒ 'a Space where
  mkSpace x y z ≡ (| svalx = x, svaly =y, svalz = z |)
```

Points have coordinates in the field of quantities, and can be thought of as the end-points of vectors pinned to the origin. We can translate and scale them, define accumulation points, etc.

**class** *Points* = *Quantities*
**begin**

**abbreviation** *moveBy* :: $'a$ *Point* ⇒ $'a$ *Point* ⇒ $'a$ *Point* (- ⊕ -)
**where**
$(p \oplus q) \equiv (\!|\ tval = tval\ p + tval\ q,$
$\quad xval = xval\ p + xval\ q,$
$\quad yval = yval\ p + yval\ q,$
$\quad zval = zval\ p + zval\ q\ |\!)$

**abbreviation** *movebackBy* :: $'a$ *Point* ⇒ $'a$ *Point* ⇒ $'a$ *Point* (- ⊖ -)
**where**
$(p \ominus q) \equiv (\!|\ tval = tval\ p - tval\ q,$
$\quad xval = xval\ p - xval\ q,$
$\quad yval = yval\ p - yval\ q,$
$\quad zval = zval\ p - zval\ q\ |\!)$

**abbreviation** *sMoveBy* :: $'a$ *Space* ⇒ $'a$ *Space* ⇒ $'a$ *Space* (- ⊕*s* -)
**where**
$(p \oplus s\ q) \equiv (\!|\ svalx = svalx\ p + svalx\ q,$
$\quad svaly = svaly\ p + svaly\ q,$
$\quad svalz = svalz\ p + svalz\ q\ |\!)$

**abbreviation** *sMovebackBy* :: $'a$ *Space* ⇒ $'a$ *Space* ⇒ $'a$ *Space* (- ⊖*s* -) **where**
$(p \ominus s\ q) \equiv (\!|\ svalx = svalx\ p - svalx\ q,$
$\quad svaly = svaly\ p - svaly\ q,$
$\quad svalz = svalz\ p - svalz\ q\ |\!)$

**abbreviation** *scaleBy* :: $'a$ ⇒ $'a$ *Point* ⇒ $'a$ *Point* (- ⊗ -) **where**
$scaleBy\ a\ p \equiv (\!|\ tval = a{*}tval\ p,\ xval = a{*}xval\ p,$
$\quad yval = a{*}yval\ p,\ zval = a{*}zval\ p|\!)$

**abbreviation** *sScaleBy* :: $'a$ ⇒ $'a$ *Space* ⇒ $'a$ *Space* ( - ⊗*s* -) **where**
$sScaleBy\ a\ p \equiv (\!|\ svalx = a{*}svalx\ p,$
$\quad svaly = a{*}svaly\ p,$
$\quad svalz = a{*}svalz\ p|\!)$

**abbreviation** *sOrigin* :: $'a$ *Space* **where**

$sOrigin \equiv (\!|\ svalx = 0,\ svaly = 0,\ svalz = 0\ |\!)$

**abbreviation** $origin$ :: $'a\ Point$ **where**
$origin \equiv (\!|\ tval = 0,\ xval = 0,\ yval = 0,\ zval = 0\ |\!)$

**abbreviation** $tUnit$ :: $'a\ Point$ **where**
$tUnit \equiv (\!|\ tval = 1,\ xval = 0,\ yval = 0,\ zval = 0\ |\!)$

**abbreviation** $xUnit$ :: $'a\ Point$ **where**
$xUnit \equiv (\!|\ tval = 0,\ xval = 1,\ yval = 0,\ zval = 0\ |\!)$

**abbreviation** $yUnit$ :: $'a\ Point$ **where**
$yUnit \equiv (\!|\ tval = 0,\ xval = 0,\ yval = 1,\ zval = 0\ |\!)$

**abbreviation** $zUnit$ :: $'a\ Point$ **where**
$zUnit \equiv (\!|\ tval = 0,\ xval = 0,\ yval = 0,\ zval = 1\ |\!)$

**abbreviation** $timeAxis$ :: $'a\ Point\ set$ **where**
$timeAxis \equiv \{\ p\ .\ xval\ p = 0 \wedge yval\ p = 0 \wedge zval\ p = 0\ \}$

**abbreviation** $onTimeAxis$ :: $'a\ Point \Rightarrow bool$
**where** $onTimeAxis\ p \equiv (p \in timeAxis)$

## 2.1 Squared norms and separation functions

This theory defines squared norms and separations. We do not yet define unsquared norms because we are not assuming here that quantities necessarily have square roots.

**abbreviation** $norm2$ :: $'a\ Point \Rightarrow\ 'a$ **where**
$norm2\ p \equiv sqr\ (tval\ p) + sqr\ (xval\ p) + sqr\ (yval\ p) + sqr\ (zval\ p)$

**abbreviation** $sep2$ :: $'a\ Point \Rightarrow\ 'a\ Point \Rightarrow\ 'a$ **where**
$sep2\ p\ q \equiv norm2\ (p \ominus q)$

**abbreviation** $sNorm2$ :: $'a\ Space \Rightarrow\ 'a$ **where**
$sNorm2\ s \equiv sqr\ (svalx\ s)$
$\quad + sqr\ (svaly\ s)$
$\quad + sqr\ (svalz\ s)$

**abbreviation** $sSep2$ :: $'a\ Point \Rightarrow\ 'a\ Point \Rightarrow\ 'a$ **where**
$sSep2\ p\ q \equiv sqr\ (xval\ p - xval\ q)$
$\quad + sqr\ (yval\ p - yval\ q)$
$\quad + sqr\ (zval\ p - zval\ q)$

**abbreviation** $mNorm2$ :: $'a\ Point \Rightarrow\ 'a\ (\|\ \text{-}\ \|m)$
**where** $\|\ p\ \|m \equiv sqr\ (tval\ p) - sNorm2\ (sComponent\ p)$

## 2.2 Topological concepts

We will need to define topological concepts like continuity and affine approximation later, so here we define open balls and accumulation points.

**abbreviation** *inBall* :: $'a$ *Point* $\Rightarrow$ $'a$ $\Rightarrow$ $'a$ *Point* $\Rightarrow$ *bool*
(- *within* - *of* -)
**where** *inBall* $q\ \varepsilon\ p \equiv sep2\ q\ p < sqr\ \varepsilon$

**abbreviation** *ball* :: $'a$ *Point* $\Rightarrow$ $'a$ $\Rightarrow$ $'a$ *Point set*
**where** *ball* $q\ \varepsilon \equiv \{\ p\ .\ inBall\ q\ \varepsilon\ p\ \}$

**abbreviation** *accPoint* :: $'a$ *Point* $\Rightarrow$ $'a$ *Point set* $\Rightarrow$ *bool*
**where** *accPoint* $p\ s \equiv \forall\ \varepsilon > 0.\ \exists\ q \in s.\ (p \neq q) \land (inBall\ q\ \varepsilon\ p)$

## 2.3 Lines

A line is specified by giving a point on the line, and a point (thought of as a vector) giving its direction. For these purposes it doesn't matter whether the direction is "positive" or "negative".

**abbreviation** *line* :: $'a$ *Point* $\Rightarrow$ $'a$ *Point* $\Rightarrow$ $'a$ *Point set*
**where** *line base drtn* $\equiv \{\ p\ .\ \exists\ \alpha\ .\ p = (base \oplus (\alpha \otimes drtn))\ \}$

**abbreviation** *lineJoining* :: $'a$ *Point* $\Rightarrow$ $'a$ *Point* $\Rightarrow$ $'a$ *Point set*
**where** *lineJoining* $p\ q \equiv line\ p\ (q \ominus p)$

**abbreviation** *isLine* :: $'a$ *Point set* $\Rightarrow$ *bool*
**where** *isLine* $l \equiv \exists\ b\ d\ .\ (l = line\ b\ d)$

**abbreviation** *sameLine* :: $'a$ *Point set* $\Rightarrow$ $'a$ *Point set* $\Rightarrow$ *bool*
**where** *sameLine* $l1\ l2 \equiv ((isLine\ l1) \lor (isLine\ l2)) \land (l1 = l2)$

**abbreviation** *onLine* :: $'a$ *Point* $\Rightarrow$ $'a$ *Point set* $\Rightarrow$ *bool*
**where** *onLine* $p\ l \equiv (isLine\ l) \land (p \in l)$

## 2.4 Directions

Given any two distinct points on a line, the vector joining them can be used to specify the line's direction. The direction of a line is therefore a *set* of points/vectors. By lemDrtn these are all parallel

**fun** *drtn* :: $'a$ *Point set* $\Rightarrow$ $'a$ *Point set*
**where** *drtn* $l = \{\ d\ .\ \exists\ p\ q\ .\ (p \neq q) \land (onLine\ p\ l) \land (onLine\ q\ l) \land (d = (q \ominus p))\ \}$

**abbreviation** *parallelLines* :: $'a$ *Point set* $\Rightarrow$ $'a$ *Point set* $\Rightarrow$ *bool*
**where** *parallelLines* $l1\ l2 \equiv (drtn\ l1) \cap (drtn\ l2) \neq \{\}$

**abbreviation** *parallel* :: $'a$ *Point* $\Rightarrow$ $'a$ *Point* $\Rightarrow$ *bool* ( - $\|$ - )
**where** *parallel p q* $\equiv$ $(\exists\ \alpha \neq 0\ .\ p = (\alpha \otimes q))$

The "slope" of a line can be either finite or infinite. We will often need to consider these two cases separately.

**abbreviation** *slopeFinite* :: $'a$ *Point* $\Rightarrow$ $'a$ *Point* $\Rightarrow$ *bool*
**where** *slopeFinite p q* $\equiv$ (*tval p* $\neq$ *tval q*)

**abbreviation** *slopeInfinite* :: $'a$ *Point* $\Rightarrow$ $'a$ *Point* $\Rightarrow$ *bool*
**where** *slopeInfinite p q* $\equiv$ (*tval p* $=$ *tval q*)

**abbreviation** *lineSlopeFinite* :: $'a$ *Point set* $\Rightarrow$ *bool*
**where** *lineSlopeFinite l* $\equiv$ ($\exists$ *x y* . (*onLine x l*) $\wedge$ (*onLine y l*)
$\wedge$ ($x \neq y$) $\wedge$ (*slopeFinite x y*))

## 2.5 Slopes and slopers

We specify the slope of a line by giving the spatial component ("sloper") of the point on the line at time 1. This is defined if and only if the slope is finite. If the slope is infinite (the line is "horizontal") we return the spatial origin. This avoids using "option" but means we need to consider carefully whether a sloper with value sOrigin indicates a truly zero slope or an infinite one.

**fun** *sloper* :: $'a$ *Point* $\Rightarrow$ $'a$ *Point* $\Rightarrow$ $'a$ *Point*
**where** *sloper p q* $=$ (*if* (*slopeFinite p q*) *then* ((*1* / (*tval p* $-$ *tval q*)) $\otimes$ ($p \ominus q$))
*else origin*)

**fun** *velocityJoining* :: $'a$ *Point* $\Rightarrow$ $'a$ *Point* $\Rightarrow$ $'a$ *Space*
**where** *velocityJoining p q* $=$ *sComponent* (*sloper p q*)

**fun** *lineVelocity* :: $'a$ *Point set* $\Rightarrow$ $'a$ *Space set*
**where** *lineVelocity l* $=$ { *v* . $\exists$ *d* $\in$ *drtn l* . *v* $=$ *velocityJoining origin d* }

**lemma** *lemNorm2Decomposition*:
  **shows** *norm2 u* = *sqr* (*tval u*) + *sNorm2* (*sComponent u*)
  **by** (*simp add*: *add-commute local.add.left-commute*)

**lemma** *lemPointDecomposition*:
  **shows** $p = (((tval\ p) \otimes tUnit) \oplus (((xval\ p) \otimes xUnit)$
  $\oplus (((yval\ p) \otimes yUnit) \oplus ((zval\ p) \otimes zUnit))))$
  **by** *force*

**lemma** *lemScaleLeftSumDistrib*: $((a + b) \otimes p) = ((a \otimes p) \oplus (b \otimes p))$
  **using** *distrib-right* **by** *auto*

**lemma** *lemScaleLeftDiffDistrib*: $((a - b) \otimes p) = ((a \otimes p) \ominus (b \otimes p))$
  **using** *left-diff-distrib* **by** *auto*

**lemma** *lemScaleAssoc*: $(\alpha \otimes (\beta \otimes p)) = ((\alpha * \beta) \otimes p)$
  **using** *semiring-normalization-rules*(*18*) **by** *auto*

**lemma** *lemScaleCommute*: $(\alpha \otimes (\beta \otimes p)) = (\beta \otimes (\alpha \otimes p))$
  **using** *mult.left-commute* **by** *auto*

**lemma** *lemScaleDistribSum*: $(\alpha \otimes (p \oplus q)) = ((\alpha \otimes p) \oplus (\alpha \otimes q))$
  **using** *distrib-left* **by** *auto*

**lemma** *lemScaleDistribDiff*: $(\alpha \otimes (p \ominus q)) = ((\alpha \otimes p) \ominus (\alpha \otimes q))$
  **using** *right-diff-distrib* **by** *auto*

**lemma** *lemScaleOrigin*: $(\alpha \otimes origin) = origin$
  **by** *auto*

**lemma** *lemMNorm2OfScaled*: *mNorm2* (*scaleBy* $\alpha$ *p*) = (*sqr* $\alpha$) $*$ *mNorm2 p*
  **using** *lemSqrMult distrib-left right-diff-distrib′* **by** *simp*

**lemma** *lemSNorm2OfScaled*: *sNorm2* (*sScaleBy* $\alpha$ *p*) = (*sqr* $\alpha$) $*$ *sNorm2 p*
  **using** *lemSqrMult distrib-left* **by** *auto*

**lemma** *lemNorm2OfScaled*: *norm2* ($\alpha \otimes p$) = (*sqr* $\alpha$) $*$ *norm2 p*
  **using** *lemSqrMult distrib-left* **by** *auto*

**lemma** *lemScaleSep2*: $(sqr\ a) * (sep2\ p\ q) = sep2\ (a \otimes p)\ (a \otimes q)$
  **using** *lemNorm2OfScaled*[*of a p*$\ominus$*q*] *lemScaleDistribDiff* **by** *auto*

**lemma** *lemSScaleAssoc*: $(\alpha \otimes s\ (\beta \otimes s\ p)) = ((\alpha * \beta) \otimes s\ p)$
  **using** *semiring-normalization-rules*(*18*) **by** *auto*

**lemma** *lemScaleBall*:
  **assumes** *x within e of y*
**and** $a \neq 0$
**shows** $(a \otimes x)$ *within* $(a * e)$ *of* $(a \otimes y)$
**proof** −
  **have** *a2pos*: $sqr\ a > 0$ **using** *assms*(*2*) *lemSquaresPositive* **by** *auto*
  **have** $sep2\ (a \otimes x)\ (a \otimes y) = (sqr\ a) * (sep2\ x\ y)$ **using** *lemScaleSep2*
**by** *auto*
  **hence** $sep2\ (a \otimes x)\ (a \otimes y) < (sqr\ a) * (sqr\ e)$
    **using** *assms mult-strict-left-mono a2pos* **by** *auto*
  **thus** *?thesis* **using** *mult-commute mult-assoc* **by** *auto*
**qed**

**lemma** *lemScaleBallAndBoundary*:
  **assumes** $sep2\ x\ y \leq sqr\ e$
**and** $a \neq 0$
**shows** $sep2\ (a \otimes x)\ (a \otimes y) \leq sqr\ (a * e)$
**proof** −
  **have** *a2pos*: $sqr\ a > 0$ **using** *assms*(*2*) *lemSquaresPositive* **by** *auto*
  **have** $sep2\ (a \otimes x)\ (a \otimes y) = (sqr\ a) * (sep2\ x\ y)$ **using** *lemScaleSep2*
**by** *auto*
  **hence** $sep2\ (a \otimes x)\ (a \otimes y) \leq (sqr\ a) * (sqr\ e)$
    **using** *assms mult-left-mono a2pos* **by** *auto*
  **thus** *?thesis* **using** *mult-commute mult-assoc* **by** *auto*
**qed**

**lemma** *lemTimeAxisIsLine*: *isLine timeAxis*
**proof** −
  **{ fix** *p*
    **{ assume** *p*: $p \in timeAxis$
      **hence** $p = (origin \oplus ((tval\ p) \otimes tUnit))$ **by** *auto*
    **}**
    **hence** *l2r*: $onTimeAxis\ p \longrightarrow (\exists\ v\ .\ (p = (origin \oplus (v \otimes tUnit))))$
**by** *blast*

    **{ assume** *v*: $\exists\ v\ .\ p = (origin \oplus (v \otimes tUnit))$

```
    hence onTimeAxis p by auto
  }
  hence (∃ v . (p = (origin ⊕ (v ⊗ tUnit)))) ⟷ onTimeAxis p
    using l2r by blast
 }
 hence timeAxis = line origin tUnit by blast
 thus ?thesis by blast
qed



lemma lemSameLine:
 assumes p ∈ line b d
shows    sameLine (line b d) (line p d)
proof –
 define l1 where l1: l1 = line b d
 define l2 where l2: l2 = line p d
 have lines: isLine l1 ∧ isLine l2 using l1 l2 by blast

 obtain A where p: p = (b ⊕ (A ⊗ d)) using assms by auto
 hence b: b = (p ⊖ (A ⊗ d)) by auto

 { fix x
  { assume x: x ∈ l1
    then obtain a where a: x = (b ⊕ (a⊗d)) using l1 by auto
    hence x = ((p ⊖ (A ⊗ d)) ⊕ (a⊗d)) using b by simp
    also have ... = (p ⊕ ((a⊗d)⊖(A⊗d)))
      using add-diff-eq diff-add-eq add-commute add-assoc by simp
    finally have x = (p ⊕ ((a−A)⊗d))
      using lemScaleLeftDiffDistrib by presburger
    hence x ∈ l2 using l2 by auto
  }
  hence l2r: (x ∈ l1) ⟶ (x ∈ l2) using l2 by simp

  { assume x: x ∈ l2
    then obtain a where a: x = (p ⊕ (a ⊗ d)) using l2 by auto
    hence x = (b ⊕ ((A+ a)⊗d))
      using p add-assoc lemScaleAssoc distrib by auto
    hence x ∈ l1 using l1 by auto
  }
  hence (x ∈ l1) ⟷ (x ∈ l2) using l2r by auto
 }
 thus ?thesis using lines l1 l2 by auto
qed
```

```
lemma lemSSep2Symmetry: sSep2 p q = sSep2 q p
  using lemSqrDiffSymmetrical by simp

lemma lemSep2Symmetry: sep2 p q = sep2 q p
  using lemSqrDiffSymmetrical by simp


lemma lemSpatialNullImpliesSpatialOrigin:
assumes sNorm2 s = 0
shows s = sOrigin
  using assms local.add-nonneg-eq-0-iff by auto


lemma lemNorm2NonNeg: norm2 p ≥ 0
  by simp


lemma lemNullImpliesOrigin:
assumes norm2 p = 0
shows p = origin
proof −
   have norm2 p = sqr (tval p) + sNorm2 (sComponent p) using
add-assoc by simp
  hence a: sqr (tval p) + sNorm2 (sComponent p) = 0 using assms
by auto
  { assume b: sNorm2 (sComponent p) > 0
    have sqr (tval p) + sNorm2 (sComponent p) > 0
      using b lemSumOfNonNegAndPos by auto
    hence False using a by auto
  }
  hence c: ¬(sNorm2 (sComponent p) > 0) by auto
  have d: sNorm2 (sComponent p) ≥ 0 by auto

  have ∀ x . (¬(x > 0)) ∧ (x ≥ 0) ⟶ x = 0 by auto
  hence e: sNorm2 (sComponent p) = 0 using c d by force
  hence f: sComponent p = sOrigin
    using lemSpatialNullImpliesSpatialOrigin by blast

  have     norm2 p = sqr (tval p) using e add-assoc by auto
  hence    sqr (tval p) = 0 using assms by simp
  hence    (tval p) = 0 using lemZeroRoot by simp

  thus ?thesis using f by auto
qed


lemma lemNotOriginImpliesPosNorm2:
assumes p ≠ origin
```

**shows** *norm2 p* $>$ *0*
**proof** $-$
**have** *1*: *norm2 p* $\geq$ *0* **by** *simp*
**have** *2*: *norm2 p* $\neq$ *0* **using** *assms*(*1*) *lemNullImpliesOrigin* **by** *force*
**thus** *?thesis* **using** *1 2 dual-order.not-eq-order-implies-strict* **by** *fast*
**qed**

**lemma** *lemNotEqualImpliesSep2Pos*:
  **assumes** $y \neq x$
  **shows** *sep2 y x* $>$ *0*
**proof** $-$
  **have** $(y \ominus x) \neq$ *origin* **using** *assms* **by** *auto*
  **hence** *1*: *norm2* $(y \ominus x) >$ *0* **using** *lemNotOriginImpliesPosNorm2* **by** *fast*
  **have** *sep2 y x* $=$ *norm2* $(y \ominus x)$ **by** *auto*
  **thus** *?thesis* **using** *1* **by** *auto*
**qed**

**lemma** *lemBallContainsCentre*:
  **assumes** $\varepsilon >$ *0*
  **shows** *x within* $\varepsilon$ *of x*
**proof** $-$
  **have** *sep2 x x* $=$ *0* **by** *auto*
  **thus** *?thesis* **using** *assms* **by** *auto*
**qed**

**lemma** *lemPointLimit*:
  **assumes** $\forall\ \varepsilon > 0$ . (*v within* $\varepsilon$ *of u*)
  **shows** $v = u$
**proof** $-$
  **define** *d* **where** *d*: *d* $=$ *sep2 v u*
  **{** **assume** $v \neq u$
    **hence** $d > 0$ **using** *lemNotEqualImpliesSep2Pos d* **by** *auto*
    **then obtain** *s* **where** *s*: $(0 < s) \wedge ($*sqr s* $< d)$ **using** *lemSmallSquares* **by** *auto*
    **hence** *v within s of u* **using** *d assms*(*1*) **by** *auto*
    **hence** *sep2 v u* $<$ *sep2 v u* **using** *s d* **by** *auto*
    **hence** *False* **by** *auto*
  **}**
  **thus** *?thesis* **by** *auto*
**qed**

**lemma** *lemBallPopulated*:
  **assumes** $e > 0$
  **shows** $\exists\ y$ . (*y within e of x*) $\wedge$ $(y \neq x)$

**proof** −
  **obtain** *e1* **where** *e1*: $(0 < e1) \wedge (e1 < e) \wedge (sqr\ e1 < e1)$
    **using** *assms lemReducedBound* **by** *auto*
  **hence** *e2*: $sqr\ e1 < sqr\ e$ **using** *lemSqrMonoStrict*[*of e1 e*] **by** *auto*

  **define** *y* **where** *y*: $y = (x \oplus (\!|\ tval = e1,\ xval{=}0,\ yval{=}0,\ zval{=}0$
$|\!))$
  **hence** $(y \ominus x) = (\!|\ tval = e1,\ xval{=}0,\ yval{=}0,\ zval{=}0\ |\!)$ **by** *auto*
  **hence** *sep2 y x* = *sqr e1* **by** *auto*
  **hence** *1*: *y within e of x* **using** *e2* **by** *auto*

  **have** *tval y* = *tval x* + *e1* **using** *y* **by** *simp*
  **hence** $y \neq x$ **using** *e1* **by** *auto*
  **thus** *?thesis* **using** *1* **by** *auto*
**qed**

**lemma** *lemBallInBall*:
  **assumes** *p within x of q*
**and** $0 < x \leq y$
**shows** *p within y of q*
**proof** −
  **have** $sqr\ x \leq sqr\ y$ **using** *assms*(*2*) *lemSqrMono* **by** *auto*
  **thus** *?thesis* **using** *le-less-trans* **using** *assms*(*1*) **by** *auto*
**qed**

**lemma** *lemSmallPoints*:
  **assumes** $e > 0$
  **shows** $\exists\ a > 0\ .\ norm2\ (a \otimes p) < sqr\ e$
**proof** −

  **{ assume** *po*: *p* = *origin*
    **define** *a* **where** *a*: $a = 1$
    **hence** *apos*: $a > 0$ **by** *auto*
    **have** $norm2\ (a \otimes p) < sqr\ e$ **using** *a po assms* **by** *auto*
    **hence** *?thesis* **using** *apos* **by** *auto*
  **}**
  **hence** *case1*: *p* = *origin* ⟶ *?thesis* **by** *auto*

  **{ assume** *pnoto*: $p \neq origin$

    **obtain** *e1* **where** *e1*: $(e1 > 0) \wedge (e1 < e) \wedge (sqr\ e1 < e1)$
      **using** *lemReducedBound assms* **by** *auto*
    **hence** *e1sqr*: $0 < (sqr\ e1) < (sqr\ e)$ **using** *lemSqrMonoStrict* **by**
*auto*

    **define** *n2* **where** *n2*: *n2* = *norm2 p*
    **hence** *n2pos*: $n2 > 0$ **using** *pnoto lemNotOriginImpliesPosNorm2*

**by** *auto*
  **then obtain** *s* **where** *s*: $(s > 0) \wedge (sqr\ s > n2)$
    **using** *lemSquareExistsAbove* **by** *auto*
  **hence** $0 < (n2/(sqr\ s)) < 1$ **using** *n2pos* **by** *auto*
  **hence** $(sqr\ e1)*(n2/(sqr\ s)) < sqr\ e1$
    **using** *lemMultPosLT1*[*of sqr e1* $(n2/(sqr\ s))$] *e1sqr* **by** *auto*
  **hence** *ineq*: $(sqr\ e1)*(n2/(sqr\ s)) < sqr\ e$ **using** *e1sqr* **by** *auto*

  **define** *a* **where** *a*: $a = e1/s$
  **have** $e1 > 0 \wedge s > 0$ **using** *e1 s* **by** *auto*
  **hence** *apos*: $a > 0$ **using** *a* **by** *auto*
  **have** $norm2\ (a \otimes p) = (sqr\ e1)*(n2/(sqr\ s))$
    **using** *lemNorm2OfScaled*[*of a*] *a n2* **by** *auto*
  **hence** $norm2\ (a \otimes p) < sqr\ e$ **using** *ineq* **by** *auto*
  **hence** *?thesis* **using** *apos* **by** *auto*
 **}**
 **hence** $p \neq origin \longrightarrow$ *?thesis* **by** *auto*

 **thus** *?thesis* **using** *case1* **by** *auto*
**qed**

**lemma** *lemLineJoiningContainsEndPoints*:
 **assumes** $l = lineJoining\ x\ p$
 **shows** $onLine\ x\ l \wedge onLine\ p\ l$
**proof** −
 **have** *line*: *isLine l* **using** *assms*(*1*) **by** *blast*
 **have** *p*: $x = (x \oplus (0 \otimes (p \ominus x)))$ **by** *simp*
 **have** *x*: $p = (x \oplus (1 \otimes (p \ominus x)))$ **using** *add-commute diff-add-cancel*
**by** *fastforce*
 **thus** *?thesis* **using** *p line assms*(*1*) **by** *blast*
**qed**

**lemma** *lemLineAndPoints*:
 **assumes** $p \neq q$
 **shows** $(onLine\ p\ l \wedge onLine\ q\ l) \longleftrightarrow (l = lineJoining\ p\ q)$
**proof** −

 **define** *lj* **where** *lj* : $lj = lineJoining\ p\ q$
 **define** *lhs* **where** *lhs*: $lhs = (onLine\ p\ l \wedge onLine\ q\ l)$
 **define** *rhs* **where** *rhs*: $rhs = (l = lj)$

 **{ assume** *hyp*: *lhs*
  **then obtain** *b d* **where** *bd*: $l = \{\ x.\ \exists\ a.\ x = (b \oplus (a \otimes d))\ \}$
**using** *lhs* **by** *auto*

  **obtain** *ap* **where** *ap*: $p = (b \oplus (ap \otimes d))$ **using** *hyp lhs bd* **by**
*auto*

**obtain** *aq* **where** *aq*: $q = (b \oplus (aq \otimes d))$ **using** *hyp lhs bd* **by** *auto*
**hence** $(q \ominus p) = ((b \oplus (aq \otimes d)) \ominus (b \oplus (ap \otimes d)))$ **using** *ap* **by** *fast*
**also have** ... $= ((aq \otimes d) \ominus (ap \otimes d))$ **using** *add-diff-cancel* **by** *auto*
**finally have** *qdiffp*: $(q \ominus p) = ((aq - ap) \otimes d)$
  **using** *lemScaleLeftDiffDistrib*[*of aq ap d*] **by** *auto*

**define** *R* **where** *R*: $R = aq - ap$
**hence** *Rnz*: $R \neq 0$ **using** *assms(1) qdiffp* **by** *auto*
**define** *r* **where** *r*: $r = 1/R$
**hence** $(r \otimes (R \otimes d)) = (r \otimes (q \ominus p))$ **using** *R qdiffp* **by** *auto*
**hence** *d*: $d = (r \otimes (q \ominus p))$ **using** *lemScaleAssoc*[*of r R d*] *r Rnz* **by** *force*

**have** $b = (p \ominus (ap \otimes d))$ **using** *ap* **by** *auto*
**also have** ... $= (p \ominus (ap \otimes (r \otimes (q \ominus p))))$ **using** *d* **by** *auto*
**finally have** *b*: $b = (p \ominus ( (ap*r) \otimes (q \ominus p)))$
  **using** *lemScaleAssoc*[*of ap r q⊖p*] **by** *auto*

**{ fix** *x*
  **assume** $x \in l$
  **then obtain** *a* **where** $x = (b \oplus (a \otimes d))$ **using** *bd* **by** *auto*
  **hence** $x = ((p \ominus ((ap*r) \otimes (q \ominus p))) \oplus ((a*r) \otimes (q \ominus p)))$
    **using** *b d lemScaleAssoc*[*of a r q⊖p*] **by** *fastforce*
  **also have** ... $= (p \oplus ( ((a*r) \otimes (q \ominus p)) \ominus ((ap*r) \otimes (q \ominus p)) ))$
    **using** *add-diff-eq diff-add-eq* **by** *force*
  **also have** ... $= (p \oplus (((a*r)-(ap*r)) \otimes (q \ominus p)))$
    **using** *left-diff-distrib* **by** *force*
  **finally have** $x \in lj$ **using** *lj* **by** *auto*
**}**
**hence** *l2r*: $l \subseteq lj$ **by** *auto*

**{ fix** *x*
  **assume** $x \in lj$
  **then obtain** *a* **where** *a*: $x = (p \oplus (a \otimes (q \ominus p)))$ **using** *lj* **by** *auto*
  **hence** $x = ((b \oplus (ap \otimes d)) \oplus (a \otimes (R \otimes d)))$ **using** *ap qdiffp R* **by** *auto*
  **also have** ... $= (b \oplus ((ap + a*R) \otimes d))$
    **using** *add-assoc distrib-right lemScaleAssoc*
    **by** *auto*
  **finally have** *onLine x l* **using** *bd* **by** *auto*
**}**
**hence** $lj \subseteq l$ **by** *auto*
**hence** $l = lj$ **using** *l2r* **by** *auto*
**}**
**hence** *L2R*: $lhs \longrightarrow rhs$ **using** *rhs* **by** *auto*

{ **assume** *l*: *rhs*
**hence** *line*: *isLine l* **using** *rhs lj* **by** *blast*
**have** *p*: $p = (p \oplus (0 \otimes (q \ominus p)))$ **by** *simp*
**have** *q*: $q = (p \oplus (1 \otimes (q \ominus p)))$ **using** *add-commute diff-add-cancel* **by** *fastforce*
**hence** *lhs* **using** *p line l lhs rhs lj* **by** *blast*
}
**hence** *rhs* $\longrightarrow$ *lhs* **by** *auto*

**hence** *lhs* $\longleftrightarrow$ *rhs* **using** *L2R* **by** *auto*

**thus** *?thesis* **using** *lhs rhs lj* **by** *auto*
**qed**

**lemma** *lemLineDefinedByPair*:
**assumes** $x \neq p$
**and** (*onLine p l1*) $\wedge$ (*onLine x l1*)
**and** (*onLine p l2*) $\wedge$ (*onLine x l2*)
**shows** *l1* = *l2*
**proof** −
**have** *l1* = *lineJoining x p*
**using** *lemLineAndPoints*[*of x p l1*] *assms*(*1*) *assms*(*2*) **by** *auto*
**also have** ... = *l2*
**using** *lemLineAndPoints*[*of x p l2*] *assms*(*1*) *assms*(*3*) **by** *auto*
**finally show** *l1* = *l2* **by** *auto*
**qed**

**lemma** *lemDrtn*:
**assumes** { *d1*, *d2* } $\subseteq$ *drtn l*
**shows** $\exists$ $\alpha \neq 0$ . *d2* = ($\alpha \otimes$ *d1*)
**proof** −
**have** *d1d2*: {*d1*,*d2*} $\subseteq$ { *d* . $\exists$ *p q* . ($p \neq q$) $\wedge$ *onLine p l* $\wedge$ *onLine q l* $\wedge$ ($d = (q \ominus p)$) }
**using** *assms*(*1*) **by** *auto*
**have** *d1*: $\exists$ *p1 q1* . (*p1* $\neq$ *q1*) $\wedge$ (*onLine p1 l*) $\wedge$ (*onLine q1 l*) $\wedge$ (*d1* = (*q1* $\ominus$ *p1*))
**using** *d1d2* **by** *auto*
**then obtain** *p1 q1*
**where** *pq1*: (*p1* $\neq$ *q1*) $\wedge$ (*onLine p1 l*) $\wedge$ (*onLine q1 l*) $\wedge$ (*d1* = (*q1* $\ominus$ *p1*))
**by** *blast*
**hence** *l1*: *l* = *lineJoining p1 q1* **using** *lemLineAndPoints*[*of p1 q1 l*] **by** *auto*

**have** *d2*: $\exists$ *p2 q2* . (*p2* $\neq$ *q2*) $\wedge$ (*onLine p2 l*) $\wedge$ (*onLine q2 l*) $\wedge$ (*d2* = (*q2* $\ominus$ *p2*))

**using** *d1d2* **by** *auto*
**then obtain** *p2 q2*
**where** *pq2*: $(p2 \neq q2) \wedge (onLine\ p2\ l) \wedge (onLine\ q2\ l) \wedge (d2 = (q2 \ominus p2))$
**by** *blast*

**hence** $(p2 \in lineJoining\ p1\ q1) \wedge (q2 \in lineJoining\ p1\ q1)$ **using** *l1* **by** *blast*
**then obtain** *ap aq*
**where** *apaq*: $(p2 = (p1 \oplus (ap \otimes (q1 \ominus p1)))) \wedge ((q2 = (p1 \oplus (aq \otimes (q1 \ominus p1)))))$
**by** *blast*

**define** *diff* **where** *diff*: $diff = aq - ap$
**hence** *diffnz*: $diff \neq 0$ **using** *apaq pq2* **by** *auto*

**have** $d2 = (q2 \ominus p2)$ **using** *pq2* **by** *simp*
**also have** ... $= ((p1 \oplus (aq \otimes (q1 \ominus p1))) \ominus (p1 \oplus (ap \otimes (q1 \ominus p1))))$ **using** *apaq* **by** *force*
**also have** ... $= ((aq \otimes (q1 \ominus p1)) \ominus (ap \otimes (q1 \ominus p1)))$ **by** *auto*
**also have** ... $= ((aq - ap) \otimes d1)$
**using** *pq1 lemScaleLeftDiffDistrib*[ *of aq ap d1*] **by** *auto*
**finally have** $(d2 = (diff \otimes d1)) \wedge (diff \neq 0)$ **using** *diff diffnz* **by** *auto*

**thus** *?thesis* **by** *auto*
**qed**

**lemma** *lemLineDeterminedByPointAndDrtn*:
**assumes** $(x \neq p) \wedge (p \in l1) \wedge (onLine\ x\ l1) \wedge (onLine\ x\ l2)$
**and** $drtn\ l1 = drtn\ l2$
**shows** $l1 = l2$
**proof** −
**define** *d1* **where** *d1*: $d1 = drtn\ l1$
**define** *d2* **where** *d2*: $d2 = drtn\ l2$
**hence** *dd*: $d1 = d2$ **using** *assms*(*2*) *d1* **by** *auto*

**define** *px* **where** *px*: $px = (p \ominus x)$

**have** *l1*: $(x \neq p) \wedge (onLine\ p\ l1) \wedge (onLine\ x\ l1)$ **using** *assms*(*1*) **by** *auto*
**hence** $\exists\ p\ q\ .\ (p \neq q) \wedge onLine\ p\ l1 \wedge onLine\ q\ l1 \wedge (px = (q \ominus p))$ **using** *px* **by** *blast*
**hence** $px \in \{\ d\ .\ \exists\ p\ q\ .\ (p \neq q) \wedge onLine\ p\ l1 \wedge onLine\ q\ l1 \wedge (d = (q \ominus p))\ \}$
**by** *blast*
**hence** $px \in d1$ **using** *d1 subst*[*of d1 drtn l1* $\lambda s.\ px \in s$] **by** *auto*
**hence** $px \in d2$ **using** *dd* **by** *simp*

**hence** *pxonl2*: $px \in drtn\ l2$ **using** *d2* **by** *simp*
**hence** $\exists\ u\ v\ .\ (u \neq v) \wedge onLine\ u\ l2 \wedge onLine\ v\ l2 \wedge (px = (v \ominus u))$ **by** *auto*
**then obtain** *u v* **where** *uv*: $(u \neq v) \wedge onLine\ u\ l2 \wedge onLine\ v\ l2 \wedge (px = (v \ominus u))$ **by** *blast*

**hence** $(x \neq u) \vee (x \neq v)$ **by** *blast*
**then obtain** *w* **where** *w*: $((w = u) \vee (w = v)) \wedge (x \neq w)$ **by** *blast*
**hence** *xw*: $(x \neq w) \wedge (onLine\ x\ l2) \wedge (onLine\ w\ l2)$ **using** *uv assms*(*1*) **by** *blast*
**hence** *l2*: $l2 = lineJoining\ x\ w$ **using** *lemLineAndPoints*[*of x w l2*] **by** *auto*
**hence** $(w \ominus x) \in drtn\ l2 \wedge px \in drtn\ l2$ **using** *xw pxonl2* **by** *auto*
**then obtain** *a* **where** *a*: $(a \neq 0) \wedge (p \ominus x) = (a \otimes (w \ominus x))$
**using** *lemDrtn*[*of w⊖x p⊖x l2*] *px xw pxonl2* **by** *blast*
**hence** $p = (x \oplus (a \otimes (w \ominus x)))$ **by** (*auto simp add*: *field-simps*)
**hence** *onLine p* (*lineJoining x w*) **by** *blast*

**hence** *l2lj*: $l2 = lineJoining\ x\ p$
**using** *lemLineAndPoints*[*of x p l2*] *assms*(*1*) *l2 xw*
**by** *auto*

**have** *l1lj*: $l1 = lineJoining\ x\ p$
**using** *lemLineAndPoints*[*of x p l1*] *assms*(*1*)
**by** *auto*

**thus** *?thesis* **using** *l2lj* **by** *blast*
**qed**

**end**

**end**

# 3 WorldView

This theory defines worldview transformations. These form the ultimate foundation for all of GenRel's axioms.

**theory** *WorldView*
**imports** *Points*
**begin**

**class** *WorldView* = *Points* +
**fixes**

$W :: Body \Rightarrow Body \Rightarrow {}'a\ Point \Rightarrow bool$ (- *sees* - *at* -)

**begin**

**abbreviation** *ev* :: *Body* ⇒ *'a Point* ⇒ *Body set*
  **where** *ev h x* ≡ { *b* . *h sees b at x* }

**fun** *wvt* :: *Body* ⇒ *Body* ⇒ *'a Point* ⇒ *'a Point set*
  **where** *wvt m k p* = { *q*. (∃ *b* . (*m sees b at p*)) ∧ (*ev m p* = *ev k q*)
}

**abbreviation** *wvtFunc* :: *Body* ⇒ *Body* ⇒ (*'a Point* ⇒ *'a Point* ⇒ *bool*)
  **where** *wvtFunc m k* ≡ (λ *p q* . *q* ∈ *wvt m k p*)

**abbreviation** *wvtLine* :: *Body* ⇒ *Body* ⇒ *'a Point set* ⇒ *'a Point set* ⇒ *bool*
  **where** *wvtLine m k l l'* ≡ ∃ *p q p' q'* . (
(*wvtFunc m k p p'*) ∧ (*wvtFunc m k q q'*) ∧
(*l* = *lineJoining p q*) ∧ (*l'* = *lineJoining p' q'*))

**end**

**end**

# 4 Functions

This theory characterises the various types of function (injective, bijective, etc).

**theory** *Functions*
  **imports** *Points*
**begin**

We do not assume a priori that all of the functions we define are well-defined or total. We therefore need to allow for functions which are only partially defined, and also for "functions" which might be multi-valued. For example, we cannot say in advance whether one observer might see another's worldline as a bifurcating structure rather than a basic single-valued trajectory.

To achieve this we'll often think of functions as relations and write "f x y = true" instead of "f x = y". Similarly, a spacetime set S will be sometimes be expressed as its characteristic function.

**class** *Functions* = *Points*
**begin**

**abbreviation** *bounded* :: (*'a Point* ⇒ *'a Point*) ⇒ *bool*

**where** $bounded\ f \equiv \exists\ \ bnd\ >\ 0\ .\ (\forall\ \ p\ .\ (norm2\ \ (f\ p)\ \ \le\ bnd\ *$ $(norm2\ p)))$

**abbreviation** *composeRel* ::
$('a\ Point \Rightarrow\ 'a\ Point \Rightarrow bool)$
$\Rightarrow('a\ Point \Rightarrow\ 'a\ Point \Rightarrow bool)$
$\Rightarrow('a\ Point \Rightarrow\ 'a\ Point \Rightarrow bool)$
**where** $(composeRel\ g\ f)\ \ p\ \ r \equiv (\exists\ \ q\ .\ ((f\ p\ q) \wedge (g\ q\ r)))$

**abbreviation** *injective* :: $('a\ Point \Rightarrow\ 'a\ Point \Rightarrow bool) \Rightarrow bool$
**where** $injective\ f \equiv \forall\ \ x1\ x2\ y1\ y2.$
$(f\ x1\ \ y1\ \wedge f\ x2\ y2) \wedge (x1\ \neq\ x2) \longrightarrow (y1\ \neq\ y2)$

**abbreviation** *definedAt* :: $('a\ Point \Rightarrow\ 'a\ Point \Rightarrow bool) \Rightarrow\ 'a\ Point$ $\Rightarrow bool$
**where** $definedAt\ f\ x \equiv \exists\ \ y\ .\ f\ x\ y$

**abbreviation** *domain* :: $('a\ Point\ => \ 'a\ Point \Rightarrow bool) \Rightarrow\ 'a\ Point$ $set$
**where** $domain\ f\ \equiv \{\ x\ .\ definedAt\ f\ x\ \}$

**abbreviation** *total* :: $('a\ Point \Rightarrow\ 'a\ Point \Rightarrow bool) \Rightarrow bool$
**where** $total\ f \equiv \forall\ \ x\ .\ (definedAt\ f\ x)$

**abbreviation** *surjective* :: $('a\ Point \Rightarrow\ 'a\ Point \Rightarrow bool) \Rightarrow bool$
**where** $surjective\ f \equiv \forall\ \ y\ .\ \exists\ \ x\ .\ f\ x\ y$

**abbreviation** *bijective* :: $('a\ Point \Rightarrow\ 'a\ Point \Rightarrow bool) \Rightarrow bool$
**where** $bijective\ f \equiv (injective\ f) \wedge (surjective\ f)$

**abbreviation** *invertible* :: $('a\ Point \Rightarrow\ 'a\ Point) \Rightarrow bool$
**where** $invertible\ f \equiv \forall\ \ q\ .\ (\exists\ \ p\ .\ (f\ p = q) \wedge (\forall x.\ f\ x = q \longrightarrow x =$ $p))$

**fun** *applyToSet* :: $('a\ Point \Rightarrow\ 'a\ Point \Rightarrow bool) \Rightarrow\ 'a\ Point\ set \Rightarrow\ 'a$ $Point\ set$
**where** $applyToSet\ f\ s = \{\ q\ .\ \exists\ \ p \in s\ .\ f\ p\ q\ \}$

**abbreviation** *singleValued* :: $('a\ Point \Rightarrow\ 'a\ Point \Rightarrow bool) \Rightarrow\ 'a\ Point$ $\Rightarrow bool$
**where** $singleValued\ f\ x\ \equiv\ \ \forall\ \ y\ z\ .\ (((f\ x\ y) \wedge (f\ x\ z)) \longrightarrow (y = z))$

**abbreviation** *isFunction* :: $('a\ Point \Rightarrow 'a\ Point \Rightarrow bool) \Rightarrow bool$
  **where** $isFunction\ f \equiv \forall\ x\ .\ singleValued\ f\ x$

**abbreviation** *isTotalFunction* :: $('a\ Point \Rightarrow 'a\ Point \Rightarrow bool) \Rightarrow bool$
  **where** $isTotalFunction\ f \equiv (total\ f) \wedge (isFunction\ f)$

**fun** *toFunc*:: $('a\ Point \Rightarrow 'a\ Point \Rightarrow bool) \Rightarrow 'a\ Point \Rightarrow 'a\ Point$
  **where** $toFunc\ f\ x = (SOME\ y\ .\ f\ x\ y)$

**fun** *asFunc* :: $('a\ Point \Rightarrow 'a\ Point) \Rightarrow ('a\ Point \Rightarrow 'a\ Point \Rightarrow bool)$
  **where** $(asFunc\ f)\ x\ y = (y = f\ x)$

## 4.1 Differentiable approximation

Here we define differentiable approximation. This will be used later when we define what it means for a worldview transformation to be "approximated" by an affine transformation.

**abbreviation** *diffApprox* :: $('a\ Point \Rightarrow 'a\ Point \Rightarrow bool) \Rightarrow$
$('a\ Point \Rightarrow 'a\ Point \Rightarrow bool) \Rightarrow$
$'a\ Point \Rightarrow bool$
  **where** $diffApprox\ g\ f\ x \equiv (definedAt\ f\ x) \wedge$
  $(\forall\ \varepsilon > 0\ .\ (\exists\ \delta > 0\ .\ (\forall\ y\ .$
  $(\ (y\ within\ \delta\ of\ x)$
  $\longrightarrow$
  $(\ (definedAt\ f\ y) \wedge (\forall\ u\ v\ .\ (f\ y\ u \wedge g\ y\ v) \longrightarrow$
  $(\ sep2\ v\ u\ ) \leq (sqr\ \varepsilon) * sep2\ y\ x\ )))\ \ )$
  ))

**abbreviation** *cts* :: $('a\ Point \Rightarrow 'a\ Point \Rightarrow bool) \Rightarrow 'a\ Point \Rightarrow bool$
  **where** $cts\ f\ x \equiv \forall\ y\ .\ (f\ x\ y) \longrightarrow (\forall \varepsilon{>}0.\ \exists \delta{>}0.$
  $(applyToSet\ f\ (ball\ x\ \delta)) \subseteq ball\ y\ \varepsilon)$

**fun** *invFunc* :: $('a\ Point \Rightarrow 'a\ Point \Rightarrow bool) \Rightarrow ('a\ Point \Rightarrow 'a\ Point$
$\Rightarrow bool)$
  **where** $(invFunc\ f)\ p\ q = f\ q\ p$

**lemma** *lemBijInv*: $bijective\ (asFunc\ f) \longleftrightarrow invertible\ f$
  **by** $(metis\ asFunc.elims(1))$

## 4.2 lemApproxEqualAtBase

The following lemma shows (as one would expect) that when one function differentiably approximates another at a point, they take equal values at that point.

**lemma** *lemApproxEqualAtBase*:
**assumes** *diffApprox g f x*
**shows** $(f\ x\ y \wedge g\ x\ z) \longrightarrow (y = z)$
**proof** −
{ **fix** *y z*
**assume** *hyp*: $f\ x\ y \wedge g\ x\ z$

**have** *lt01*: $0 < 1$ **by** *auto*
**then obtain** *d* **where** *dprops*: $(d > 0) \wedge (\forall\ y\ .$
$(\ (y\ within\ d\ of\ x)$
$\longrightarrow$
$(\ \forall\ u\ v\ .\ (f\ y\ u \wedge g\ y\ v) \longrightarrow$
$(\ sep2\ v\ u\ ) \leq (sqr\ 1) * sep2\ y\ x\ ))\ \ )$
**using** *assms*(*1*) **by** *best*

**hence** *x within d of x* **by** *auto*
**hence** $\forall\ u\ v\ .\ (f\ x\ u \wedge g\ x\ v) \longrightarrow (sep2\ v\ u) \leq (sqr\ 1) * sep2\ x\ x$
**using** *dprops* **by** *blast*
**hence** *sep0*: $(sep2\ z\ y) \leq 0$ **using** *hyp* **by** *auto*
{ **assume** $z \neq y$
**hence** $sep2\ z\ y > 0$ **using** *lemNotEqualImpliesSep2Pos*[*of z y*]
**by** *auto*
**hence** *False* **using** *sep0* **by** *auto*
}
**hence** $z = y$ **by** *auto*
}
**thus** *?thesis* **by** *auto*
**qed**

**lemma** *lemCtsOfCtsIsCts*:
**assumes** *cts f x*
**and** $\forall y\ .\ (f\ x\ y) \longrightarrow (cts\ g\ y)$
**shows** $cts\ (composeRel\ g\ f)\ x$
**proof** −
{ **fix** *z*
**assume** *z*: $(composeRel\ g\ f)\ x\ z$
**then obtain** *y* **where** *y*: $f\ x\ y \wedge g\ y\ z$ **by** *auto*

{ **fix** *e*
**assume** *epos*: $e > 0$

**have** $(\forall \varepsilon{>}0.\ \exists \delta{>}0.(applyToSet\ g\ (ball\ y\ \delta)) \subseteq ball\ z\ \varepsilon)$
**using** *assms*(*2*) *y* **by** *auto*

```
      then obtain dy
        where dy: (dy > 0) ∧ ((applyToSet g (ball y dy)) ⊆ ball z e)
        using epos y by auto

      have (∀ε>0. ∃δ>0.(applyToSet f (ball x δ)) ⊆ ball y ε)
        using y assms(1) by auto
      then obtain d
        where d: (d > 0) ∧ ((applyToSet f (ball x d)) ⊆ ball y dy)
        using dy by auto

      { fix w
        assume w: w ∈ applyToSet (composeRel g f) (ball x d)
        then obtain u v
          where v: (u ∈ ball x d) ∧ (f u v) ∧ (g v w) by auto
        hence v ∈ ball y dy using d by auto
        hence w ∈ ball z e using v dy by auto
      }
      hence applyToSet (composeRel g f) (ball x d) ⊆ ball z e by auto
       hence ∃ d>0. (applyToSet (composeRel g f) (ball x d) ⊆ ball z
e)
        using d by auto
    }
    hence ∀ e>0. ∃ d>0. applyToSet (composeRel g f) (ball x d) ⊆ ball
z e by auto
  }
  thus ?thesis by auto
qed


lemma lemInjOfInjIsInj:
  assumes injective f
and       injective g
shows     injective (composeRel g f)
proof −
  { fix x1 z1 x2 z2
    assume hyp: (composeRel g f) x1 z1 ∧ (composeRel g f) x2 z2 ∧
(x1 ≠ x2)
    then obtain y1 y2
      where ys: (f x1 y1) ∧ (g y1 z1) ∧ (f x2 y2) ∧ (g y2 z2) by auto
    hence y1 ≠ y2 using hyp assms(1) by auto
    hence z1 ≠ z2 using  assms(2) ys by auto
  }
  thus ?thesis by auto
qed


lemma lemInverseComposition:
  assumes h = composeRel g f
  shows   (invFunc h) = composeRel (invFunc f) (invFunc g)
```

**proof** –
{ **fix** *p r*
{ **assume** *hyp*: *h p r*
**then obtain** *q* **where** *f p q* ∧ *g q r* **using** *assms* **by** *auto*
**hence** (*invFunc g*) *r q* ∧ (*invFunc f*) *q p* **by** *force*
**hence** (*composeRel* (*invFunc f*) (*invFunc g*)) *r p* **by** *blast*
}
**hence** *l2r*: (*invFunc h*) *r p* ⟶ (*composeRel* (*invFunc f*) (*invFunc g*)) *r p* **by** *auto*

{ **assume** (*composeRel* (*invFunc f*) (*invFunc g*)) *r p*
**then obtain** *q* **where** (*invFunc g*) *r q* ∧ (*invFunc f*) *q p* **by** *auto*
**hence** (*invFunc h*) *r p* **using** *assms* **by** *auto*
}

**hence** (*composeRel* (*invFunc f*) (*invFunc g*)) *r p* ⟷ (*invFunc h*) *r p*
**using** *l2r* **by** *auto*
}
**thus** *?thesis* **by** *fastforce*
**qed**

**lemma** *lemToFuncAsFunc*:
**assumes** *isFunction f*
**and** *total f*
**shows** *asFunc* (*toFunc f*) = *f*
**proof** –
{ **fix** *p r*
{ **assume** (*asFunc* (*toFunc f*)) *p r*
**hence** *f p r* **using** *someI*[*of f p*] *assms*(*2*) **by** *auto*
}
**hence** *l2r*: (*asFunc* (*toFunc f*)) *p r* ⟶ *f p r* **by** *auto*
{ **assume** *fpr*: *f p r*
**hence** (*asFunc* (*toFunc f*)) *p r* **using** *someI*[*of f p*] *assms*(*1*) **by** *auto*
}

**hence** *f p r* ⟷ (*asFunc* (*toFunc f*)) *p r* **using** *l2r* **by** *auto*
}
**thus** *?thesis* **by** *blast*
**qed**

**lemma** *lemAsFuncToFunc*: *toFunc* (*asFunc f*) = *f*
**by** *fastforce*

**end**

**end**

# 5 WorldLine

This theory defines worldlines.

**theory** *WorldLine*
**imports** *WorldView Functions*
**begin**

**class** *WorldLine* = *WorldView* + *Functions*
**begin**

**abbreviation** *wline* :: *Body* ⇒ *Body* ⇒ *'a Point set*
**where** *wline m k* ≡ { *p* . *m sees k at p* }

**lemma** *lemWorldLineUnderWVT*:
**shows** *applyToSet* (*wvtFunc m k*) (*wline m b*) ⊆ *wline k b*
**by** *auto*

**lemma** *lemFiniteLineVelocityUnique*:
**assumes** (*u* ∈ *lineVelocity l*) ∧ (*v* ∈ *lineVelocity l*)
**and** *lineSlopeFinite l*
**shows** *u* = *v*
**proof** −
**have** ∃ *d1* ∈ *drtn l* . *u* = *velocityJoining origin d1* **using** *assms* **by** *simp*
**then obtain** *d1*
**where** *d1*: *d1* ∈ *drtn l* ∧ *u* = *velocityJoining origin d1* **by** *blast*

**have** ∃ *d2* ∈ *drtn l* . *v* = *velocityJoining origin d2* **using** *assms* **by** *simp*
**then obtain** *d2*
**where** *d2*: *d2* ∈ *drtn l* ∧ *v* = *velocityJoining origin d2* **by** *blast*

**hence** (*d1* ∈ *drtn l*) ∧ (*d2* ∈ *drtn l*) **using** *d1 d2* **by** *auto*

**then obtain** *a* **where** *a*: (*a* ≠ *0*) ∧ (*d2* = (*a* ⊗ *d1*))
**using** *lemDrtn*[*of d1 d2 l*] **by** *blast*

**have** *slopes*: (*tval d1* ≠ *0*) ∧ (*tval d2* ≠ *0*)
∧ (*slopeFinite origin d1*) ∧ (*slopeFinite origin d2*)

**proof** −
**obtain** *x y* **where** *xy*: $(x \neq y) \wedge (onLine\ x\ l) \wedge (onLine\ y\ l) \wedge (slopeFinite\ x\ y)$
**using** *assms(2)* **by** *blast*
**hence** *slopeFinite x y* **by** *blast*
**hence** *tvalnz*: $tval\ y - tval\ x \neq 0$ **by** *simp*

**define** *yx* **where** $yx = (y \ominus x)$
**hence** $(x \neq y) \wedge (onLine\ x\ l) \wedge (onLine\ y\ l) \wedge (yx = (y \ominus x))$ **using** *xy* **by** *simp*
**hence** $\exists\ x\ y\ .\ (x \neq y) \wedge (onLine\ x\ l) \wedge (onLine\ y\ l) \wedge (yx = (y \ominus x))$ **by** *blast*
**hence** $(y \ominus x) \in drtn\ l$ **using** *yx-def* **by** *auto*
**then obtain** *b* **where** *b*: $(b \neq 0) \wedge (d2 = (b \otimes (y \ominus x)))$
**using** *d2 lemDrtn[of y⊖x d2 l]* **by** *blast*

**hence** *tval2*: $tval\ d2 \neq tval\ origin$ **using** *tvalnz b* **by** *simp*
**hence** *tval1*: $tval\ d1 \neq tval\ origin$ **using** *a* **by** *auto*
**hence** *finite*: $(slopeFinite\ origin\ d1) \wedge (slopeFinite\ origin\ d2)$
**using** *tval2* **by** *auto*
**have** $tval\ origin = 0$ **by** *simp*
**thus** *?thesis* **using** *tval1 tval2 finite* **by** *blast*
**qed**

**have** *t1nz*: $tval\ d1 \neq 0$ **using** *slopes* **by** *auto*
**have** *anz*: $a \neq 0$ **using** *a* **by** *blast*
**hence** *equ*: $1/(tval\ d1) = (1/(a*tval\ d1))*a$ **by** *simp*

**hence** $sloper\ origin\ d1 = (((1/(a*tval\ d1))*a) \otimes d1)$ **using** *slopes* **by** *auto*
**also have** ... $= ((1/(tval\ d2)) \otimes d2)$
**using** *lemScaleAssoc[of 1/(a∗tval d1) a d1] a* **by** *auto*
**finally have** *equalslopers*: $sloper\ origin\ d1 = sloper\ origin\ d2$ **using** *slopes* **by** *auto*

**thus** *?thesis* **using** *d1 d2* **by** *auto*
**qed**

**end**

**end**

# 6 Translations

This theory describes translation maps.

**theory** *Translations*
**imports** *Functions*
**begin**

**class** *Translations* = *Functions*
**begin**

**abbreviation** *mkTranslation* :: $'a$ *Point* $\Rightarrow$ ($'a$ *Point* $\Rightarrow$ $'a$ *Point*)
  **where** (*mkTranslation* $t$) $\equiv$ ($\lambda$ $p$ . ($p \oplus t$))

**abbreviation** *translation* :: ($'a$ *Point* $\Rightarrow$ $'a$ *Point*) $\Rightarrow$ *bool*
  **where** *translation* $T \equiv \exists\ q\ .\ \forall\ p\ .\ ((T\ p) = (p \oplus q))$

**lemma** *lemMkTrans*: $\forall\ t$ . *translation* (*mkTranslation* $t$)
  **by** *auto*

**lemma** *lemInverseTranslation*:
  **assumes** ($T$ = *mkTranslation* $t$) $\wedge$ ($T'$ = *mkTranslation* (*origin* $\ominus$
$t$))
  **shows** ($T' \circ T = id$) $\wedge$ ($T \circ T' = id$)
**using** *assms* **by** *auto*

**lemma** *lemTranslationSum*:
  **assumes** *translation* $T$
  **shows** $T\ (u \oplus v) = ((T\ u) \oplus v)$
**proof** −
  **obtain** $t$ **where** $t$: $\forall\ x.\ T\ x = (x \oplus t)$ **using** *assms*(*1*) **by** *auto*
  **thus** *?thesis* **using** *add-commute add-assoc* $t$ **by** *auto*
**qed**

**lemma** *lemIdIsTranslation*: *translation id*
**proof** −
  **have** $\forall\ p$ . (*id* $p$) = ($p \oplus$ *origin*) **by** *simp*
  **thus** *?thesis* **by** *blast*
**qed**

**lemma** *lemTranslationCancel*:
  **assumes** *translation* $T$
  **shows** $((T\ p) \ominus (T\ q)) = (p \ominus q)$

**proof** −
  **obtain** *t* **where** *t*: $\forall\ x.\ T\ x = (x \oplus t)$ **using** *assms*(*1*) **by** *auto*
  **hence** $((p \oplus t) \ominus (q \oplus t)) = (p \ominus q)$ **by** *simp*
  **thus** *?thesis* **using** *t* **by** *auto*
**qed**

**lemma** *lemTranslationSwap*:
  **assumes** *translation T*
  **shows** $(p \oplus (T\ q)) = ((T\ p) \oplus q)$
**proof** −
  **obtain** *t* **where** *t*: $\forall\ x\ .\ T\ x = (x \oplus t)$ **using** *assms*(*1*) **by** *auto*
  **thus** *?thesis* **using** *add-commute add-assoc* **by** *simp*
**qed**

**lemma** *lemTranslationPreservesSep2*:
  **assumes** *translation T*
  **shows** *sep2 p q* = *sep2* (*T p*) (*T q*)
**proof** −
  **obtain** *t* **where** $\forall x.\ T\ x = (x \oplus t)$ **using** *assms*(*1*) **by** *auto*
  **thus** *?thesis* **by** *force*
**qed**

**lemma** *lemTranslationInjective*:
  **assumes** *translation T*
  **shows** *injective* (*asFunc T*)
**proof** −
  **obtain** *t* **where** *t*: $\forall\ x\ .\ T\ x = (x \oplus t)$ **using** *assms*(*1*) **by** *auto*
  **define** *Tinv* **where** *Tinv*: *Tinv* = *mkTranslation* $(origin \ominus t)$
  **{ fix** *x y*
    **assume** $T\ x = T\ y$
    **hence** $(Tinv \circ T)\ x = (Tinv \circ T)\ y$ **by** *auto*
    **hence** $x = y$ **using** *Tinv t* **by** *auto*
  **}**
  **thus** *?thesis* **by** *auto*
**qed**

**lemma** *lemTranslationSurjective*:
  **assumes** *translation T*
  **shows** *surjective* (*asFunc T*)
**proof** −
  **obtain** *t* **where** *t*: $\forall\ x\ .\ T\ x = (x \oplus t)$ **using** *assms*(*1*) **by** *auto*
  **hence** *mkT*: *T* = *mkTranslation t* **by** *auto*

**define** *Tinv* **where** *Tinv*: $Tinv = mkTranslation\ (origin \ominus t)$
**hence** $\forall\, y\ .\ y = T\ (Tinv\ y)$ **using** *mkT lemInverseTranslation* **by** *auto*
**thus** *?thesis* **by** *auto*
**qed**

**lemma** *lemTranslationTotalFunction*:
**assumes** *translation T*
**shows** *isTotalFunction (asFunc T)*
**by** *simp*

**lemma** *lemTranslationOfLine*:
**assumes** *translation T*
**shows** $(applyToSet\ (asFunc\ T)\ (line\ B\ D)) = line\ (T\ B)\ D$
**proof** −
**define** *l* **where** *l*: $l = line\ B\ D$
**{ fix** $q'$
**{ assume** $q' \in (applyToSet\ (asFunc\ T)\ l)$
**then obtain** *q* **where** *q*: $q \in l \wedge (asFunc\ T)\ q\ q'$ **by** *auto*
**then obtain** $\alpha$ **where** $\alpha$: $q = (B \oplus (\alpha \otimes D))$ **using** *l* **by** *auto*
**have** $q' = T\ q$ **using** *q* **by** *auto*
**also have** $\ldots = ((T\ B) \oplus (\alpha \otimes D))$ **using** $\alpha$ *assms lemTranslationSum* **by** *blast*
**finally have** $q' \in line\ (T\ B)\ D$ **by** *auto*
**}**
**hence** *l2r*: $q' \in (applyToSet\ (asFunc\ T)\ l) \longrightarrow q' \in line\ (T\ B)\ D$ **by** *auto*
**{ assume** $q' \in line\ (T\ B)\ D$
**then obtain** $\alpha$ **where** $\alpha$: $q' = ((T\ B) \oplus (\alpha \otimes D))$ **by** *auto*
**hence** $q' = T\ (B \oplus (\alpha \otimes D))$ **using** *assms lemTranslationSum*[*of T B* $(\alpha \otimes D)$] **by** *auto*
**moreover have** $(B \oplus (\alpha \otimes D)) \in l$ **using** *l* **by** *auto*
**ultimately have** $q' \in (applyToSet\ (asFunc\ T)\ l)$ **by** *auto*
**}**
**hence** $q' \in line\ (T\ B)\ D \longleftrightarrow q' \in (applyToSet\ (asFunc\ T)\ l)$ **using** *l2r* **by** *auto*
**}**
**thus** *?thesis* **using** *l* **by** *auto*
**qed**

**lemma** *lemOnLineTranslation*:
**assumes** $(translation\ T) \wedge (onLine\ p\ l)$
**shows** $onLine\ (T\ p)\ (applyToSet\ (asFunc\ T)\ l)$
**proof** −
**obtain** *B D* **where** *BD*: $l = line\ B\ D$ **using** *assms* **by** *auto*
**hence** $(applyToSet\ (asFunc\ T)\ l) = line\ (T\ B)\ D$ **using** *assms*

*lemTranslationOfLine* **by** *auto*
**moreover have** $T\ p \in (applyToSet\ (asFunc\ T)\ l)$ **using** *assms* **by** *auto*
**ultimately show** *?thesis* **by** *blast*
**qed**

**lemma** *lemLineJoiningTranslation*:
**assumes** *translation T*
**shows** $applyToSet\ (asFunc\ T)\ (lineJoining\ p\ q) = lineJoining\ (T\ p)\ (T\ q)$
**proof** −
**define** $D$ **where** $D$: $D = (q \ominus p)$
**hence** $lineJoining\ p\ q = line\ p\ D$ **by** *auto*
**hence** $applyToSet\ (asFunc\ T)\ (lineJoining\ p\ q) = line\ (T\ p)\ D$
**using** *assms lemTranslationOfLine* **by** *auto*
**moreover have** $((T\ q) \ominus (T\ p)) = (q \ominus p)$ **using** *assms lemTranslationCancel* **by** *auto*
**ultimately show** *?thesis* **using** $D$ **by** *auto*
**qed**

**lemma** *lemBallTranslation*:
**assumes** *translation T*
**and** *x within e of y*
**shows** $(T\ x)$ *within e of* $(T\ y)$
**proof** −
**have** $sep2\ (T\ x)\ (T\ y) = sep2\ x\ y$
**using** *assms*(*1*) *lemTranslationPreservesSep2*[*of T*] **by** *auto*
**thus** *?thesis* **using** *assms*(*2*) **by** *auto*
**qed**

**lemma** *lemBallTranslationWithBoundary*:
**assumes** *translation T*
**and** $sep2\ x\ y \leq sqr\ e$
**shows** $sep2\ (T\ x)\ (T\ y) \leq sqr\ e$
**proof** −
**have** $sep2\ (T\ x)\ (T\ y) = sep2\ x\ y$
**using** *assms*(*1*) *lemTranslationPreservesSep2*[*of T x y*] **by** *simp*
**thus** *?thesis* **using** *assms*(*2*) **by** *auto*
**qed**

**lemma** *lemTranslationIsCts*:

```
  assumes translation T
  shows cts (asFunc T) x
proof -
  { fix x′
    assume x′: x′ = T x

    { fix e
      assume epos: e > 0
      { fix p′
        assume p′: p′ ∈ applyToSet (asFunc T) (ball x e)
        then obtain p where p: (p ∈ ball x e) ∧ p′ = T p by auto
        hence sep2 p x < sqr e using lemSep2Symmetry by force
       hence sep2 p′ x′ < sqr e using assms(1) p x′ lemBallTranslation
by auto
      }
      hence applyToSet (asFunc T) (ball x e) ⊆ ball x′ e
        using lemSep2Symmetry by force
      hence ∃ d>0. applyToSet (asFunc T) (ball x d) ⊆ ball x′ e
        using epos lemSep2Symmetry by auto
    }
    hence ∀ e>0. ∃ d>0. applyToSet (asFunc T) (ball x d) ⊆ ball x′ e
      by auto
  }
  thus ?thesis by auto
qed



lemma lemAccPointTranslation:
  assumes translation T
and        accPoint x s
shows      accPoint (T x) (applyToSet (asFunc T) s)
proof -
  { fix e
    assume e > 0
    then obtain q where q: q ∈ s ∧ (x ≠ q) ∧ (inBall q e x)
      using assms(2) by auto

    have acc1: q ∈ s using q by auto
    have acc2: x ≠ q using q by auto
    have acc3: inBall q e x using q by auto

    define q′ where q′: q′ = T q

    have rtp1: q′ ∈ applyToSet (asFunc T) s using q′ acc1 by auto
   have rtp2: T x ≠ q′ using assms(1) acc2 lemTranslationInjective[of
T] q′ by force
    have rtp3: inBall q′ e (T x)
      using assms(1) acc3 q′ lemBallTranslation[of T q x e] by auto
```

**hence** $\exists\ q'\ .\ (q' \in applyToSet\ (asFunc\ T)\ s) \wedge (T\ x \neq q')$
$\wedge\ (inBall\ q'\ e\ (T\ x))$
**using** *rtp1 rtp2* **by** *auto*
**}**
**thus** *?thesis* **by** *auto*
**qed**

**lemma** *lemInverseOfTransIsTrans*:
**assumes** *translation T*
**and** $T' = invFunc\ (asFunc\ T)$
**shows** *translation* $(toFunc\ T')$
**proof** −
**obtain** *t* **where** *t*: $\forall\ p\ .\ T\ p = (p \oplus t)$ **using** *assms*(*1*) **by** *auto*
**hence** *mkT*: $T = mkTranslation\ t$ **by** *auto*
**define** *T1* **where** *T1*: $T1 = mkTranslation\ (origin \ominus t)$
**hence** *transT1*: *translation T1* **using** *lemMkTrans* **by** *blast*

**have** *TT1*: $(T \circ T1 = id) \wedge (T1 \circ T = id)$ **using** *t T1 lemInverseTranslation* **by** *auto*

**{ fix** *p r*
**{ assume** *invFunc* (*asFunc T*) *p r*
**hence** $T\ r = p$ **by** *simp*
**hence** $T1\ p = (T1 \circ T)\ r$ **by** *auto*
**hence** $T1\ p = r$ **using** *TT1* **by** *simp*
**}**
**hence** *l2r*: $invFunc\ (asFunc\ T)\ p\ r \longrightarrow (asFunc\ T1)\ p\ r$ **by** *auto*
**{ assume** (*asFunc T1*) *p r*
**hence** *T′p*: $T1\ p = r$ **by** *simp*
**have** $(T \circ T1)\ p = T\ r$ **using** *T′p* **by** *auto*
**hence** $p = T\ r$ **using** *TT1* **by** *auto*
**}**
**hence** $(asFunc\ T1)\ p\ r \longleftrightarrow invFunc\ (asFunc\ T)\ p\ r$ **using** *l2r* **by** *force*
**}**
**hence** $(asFunc\ T1) = T'$ **using** *assms*(*2*) **by** *fastforce*

**hence** $toFunc\ T' = toFunc\ (asFunc\ T1)$ **using** *assms*(*2*) **by** *fastforce*
**hence** $toFunc\ T' = T1$ **by** *fastforce*
**thus** *?thesis* **using** *transT1* **by** *auto*
**qed**

**lemma** *lemInverseTrans*:
**assumes** *translation T*
**shows** $\exists\, T'\ .\ (translation\ T') \wedge (\forall\ \ p\ q\ .\ T\ p = q \longleftrightarrow T'\ q = p)$

**proof** −
  **obtain** $t$ **where** $t$: $\forall\, p$ . $T\ p = (p \oplus t)$ **using** *assms* **by** *auto*
  **hence** *mkT*: $T = mkTranslation\ t$ **by** *auto*
  **define** $T'$ **where** $T'$: $T' = mkTranslation\ (origin \ominus t)$
  **hence** *trans'*: *translation* $T'$ **using** *lemMkTrans* **by** *blast*

  **have** $TT'$: $(T' \circ T = id) \wedge (T \circ T' = id)$ **using** *mkT* $T'$ *lemInverse-Translation* **by** *auto*

  **{ fix** $p\ q$
    **{ assume** $T\ p = q$
      **hence** $T'\ q = (T' \circ T)\ p$ **by** *auto*
      **hence** $T'\ q = p$ **using** $TT'$ **by** *auto*
    **}**
    **hence** *l2r*: $T\ p = q \longrightarrow T'\ q = p$ **by** *auto*
    **{ assume** $T'\ q = p$
      **hence** $T\ p = (T \circ T')\ q$ **by** *auto*
      **hence** $T\ p = q$ **using** $TT'$ **by** *auto*
    **}**
    **hence** $T'\ q = p \longleftrightarrow T\ p = q$ **using** *l2r* **by** *blast*
  **}**
  **thus** *?thesis* **using** *trans'* **by** *blast*
**qed**

**end**

**end**

# 7 AXIOM: AxSelfMinus

This theory declares the axiom AxSelfMinus.

**theory** *AxSelfMinus*
  **imports** *WorldView*
**begin**

AxSelfMinus: The worldline of an observer is a subset of the time axis in their own worldview.

**class** *axSelfMinus* = *WorldView*
**begin**
  **abbreviation** *axSelfMinus* :: *Body* $\Rightarrow$ *'a Point* $\Rightarrow$ *bool*
    **where** *axSelfMinus m p* $\equiv$ (*m sees m at p*) $\longrightarrow$ *onTimeAxis p*
**end**

**class** *AxSelfMinus* = *axSelfMinus* +
  **assumes** *AxSelfMinus* : $\forall$ *m p* . *axSelfMinus* *m p*
**begin**
**end**

**end**

# 8 TangentLines

This theory defines tangent lines and establishes their key properties.

**theory** *TangentLines*
  **imports** *Translations AxSelfMinus*
**begin**

At each point along the worldline of a body, we can ask what its instantaneous direction of motion is. Unfortunately we do not know a priori that the "worldline" actually has tangents. Dealing with tangent lines is one of the more complicated aspects of the main proof.

**class** *TangentLines* = *Translations* + *AxSelfMinus*
**begin**

**abbreviation** *tangentLine* :: $'a$ *Point set* $\Rightarrow$ $'a$ *Point set* $\Rightarrow$ $'a$ *Point* $\Rightarrow$ *bool*
  **where** *tangentLine l s x* $\equiv$
   $(x \in s) \wedge (\mathit{onLine}\ x\ l) \wedge (\mathit{accPoint}\ x\ s)$
 $\wedge$
   $(\exists\ p\ .\ (\ (\mathit{onLine}\ p\ l) \wedge (p \neq x) \wedge$
     $(\forall\ \varepsilon > 0\ .\ \ \exists\ \delta > 0\ .\ \forall\ y \in s.\ ($
       $(\ (y\ \mathit{within}\ \delta\ \mathit{of}\ x) \wedge (y \neq x)\ )$
       $\longrightarrow$
       $(\ \exists\ r\ .\ ((\mathit{onLine}\ r\ (\mathit{lineJoining}\ x\ y)) \wedge (r\ \mathit{within}\ \varepsilon\ \mathit{of}\ p))))$
     $)$
   $))$

**abbreviation** *tangentLineA* :: $'a$ *Point set* $\Rightarrow$ $'a$ *Point set* $\Rightarrow$ $'a$ *Point* $\Rightarrow$ *bool*
  **where** *tangentLineA l s x* $\equiv$
   $(x \in s) \wedge (\mathit{onLine}\ x\ l) \wedge (\mathit{accPoint}\ x\ s)$
 $\wedge$
   $(\forall\ p\ .\ (\ ((\mathit{onLine}\ p\ l) \wedge (p \neq x)) \longrightarrow$
     $(\forall\ \varepsilon > 0\ .\ \ \exists\ \delta > 0\ .\ \forall\ y \in s.\ ($
       $(\ (y\ \mathit{within}\ \delta\ \mathit{of}\ x) \wedge (y \neq x)\ )$
       $\longrightarrow$
       $(\ \exists\ r\ .\ ((\mathit{onLine}\ r\ (\mathit{lineJoining}\ x\ y)) \wedge (r\ \mathit{within}\ \varepsilon\ \mathit{of}\ p))))$

```
    )
  ))
```

```
abbreviation hasTangent :: 'a Point set ⇒ 'a Point ⇒ bool
  where hasTangent s p ≡ ∃ l . tangentLine l s p
```

The instantaneous velocity of a body is defined to be the velocity of a co-moving body moving along the tangent line (assuming a tangent line exists).

```
fun vel :: 'a Point set ⇒ 'a Point ⇒ 'a Space ⇒ bool
  where vel wl p v = ( ∃ l . ( (tangentLine l wl p) ∧ (v ∈ lineVelocity
l) ))
```

```
lemma lemTangentLineTranslation:
  assumes translation T
and       tangentLine l s x
shows     tangentLine (applyToSet (asFunc T) l)
                  (applyToSet (asFunc T) s)  (T x)
proof −
  define x' where x': x' = T x
  define l' where l': l' = applyToSet (asFunc T) l
  define s' where s': s' = applyToSet (asFunc T) s

  have tgt1: x ∈ s using assms(2) by simp

  have tgt2: onLine x l using assms(2) by simp
  hence linel: isLine l by auto

  have tgt3: accPoint x s using assms(2) by simp
  have tgt4: ∃ p . ( ((onLine p l) ∧ (p ≠ x)) ∧
      (∀ ε > 0 . ∃ δ > 0 . ∀ y ∈ s. (
        ( (y within δ of x) ∧ (y ≠ x) )
        ⟶
        ( ∃ r . ((onLine r (lineJoining x y)) ∧ (r within ε of p))))
      )
   ) using assms(2) by simp

  have rtp1: x' ∈ s' using x' s' tgt1 by auto
  have rtp2: onLine x' l'
    using lemOnLineTranslation[of T l x] x' l' assms(1) linel tgt2
    by auto
  have rtp3: accPoint x' s'
    using assms(1) tgt3 lemAccPointTranslation x' s'
    by simp
```

**obtain** $p$ **where** $p$: (($onLine\ p\ l$) $\wedge$ ($p \neq x$)) $\wedge$
($\forall\ \varepsilon > 0$ . $\exists\ \delta > 0$ . $\forall\ y \in s$. (
( ($y$ *within* $\delta$ *of* $x$) $\wedge$ ($y \neq x$) )
$\longrightarrow$
( $\exists\ r$ . (($onLine\ r$ ($lineJoining\ x\ y$)) $\wedge$ ($r$ *within* $\varepsilon$ *of* $p$))))
) **using** *tgt4* **by** *auto*

**define** $p'$ **where** $p'$: $p' = (T\ p)$
**hence** $p'$-*on*-$l'$: *onLine* $p'$ $l'$ **using** $l'$ *rtp2* $p$ **by** *auto*
**have** $p'$-*not*-$x'$: $p' \neq x'$
**using** $p'$ $p$ *assms*(*1*) $x'$ *lemTranslationInjective*[*of* $T$] **by** *force*

**{ fix** $e$
**assume** *epos*: $e > 0$
**then obtain** $d$ **where** $d$: ($d > 0$) $\wedge$ ($\forall\ y \in s$. (
( ($y$ *within* $d$ *of* $x$) $\wedge$ ($y \neq x$) )
$\longrightarrow$
( $\exists\ r$ . (($onLine\ r$ ($lineJoining\ x\ y$)) $\wedge$ ($r$ *within* $e$ *of* $p$))))
) **using** $p$ **by** *blast*

**{ fix** $y'$
**assume** $y'$: ($y' \in s'$) $\wedge$ ($y'$ *within* $d$ *of* $x'$) $\wedge$ ($y' \neq x'$)
**then obtain** $y$ **where** $y$: $y \in s \wedge y' = T\ y$ **using** $s'$ **by** *force*

**hence** *y1*: $y \in s$ **using** $y$ **by** *auto*
**have** *y2*: $y$ *within* $d$ *of* $x$
**using** *assms*(*1*) $x'$ $y$ $y'$ *lemBallTranslation* **by** *fastforce*
**have** *y3*: $y \neq x$ **using** $y'$ $y$ $x'$ *assms*(*1*) **by** *fastforce*

**then obtain** $r$
**where** $r$: ($onLine\ r$ ($lineJoining\ x\ y$)) $\wedge$ ($r$ *within* $e$ *of* $p$)
**using** *y1* *y2* $d$ **by** *force*

**define** $r'$ **where** $r'$: $r' = T\ r$
**hence** $r' \in$ *applyToSet* (*asFunc* $T$) (*lineJoining* $x$ $y$) **using** $r$ **by** *auto*
**hence** *r1*: *onLine* $r'$ (*lineJoining* $x'$ $y'$)
**using** *assms*(*1*) *lemLineJoiningTranslation*[*of* $T$ $x$ $y$] $x'$ $y$
**by** *blast*
**have** *r2*: $r'$ *within* $e$ *of* $p'$
**using** *assms*(*1*) $r$ $r'$ $p'$ *lemBallTranslation* **by** *auto*

**hence** $\exists\, r'$. (*onLine* $r'$ (*lineJoining* $x'$ $y'$)) $\wedge$ ($r'$ *within* $e$ *of* $p'$)
**using** *r1* **by** *auto*
**hence** ($y'$ *within* $d$ *of* $x'$) $\wedge$ ($y' \neq x'$)
$\longrightarrow$ ($\exists\, r'$. (*onLine* $r'$ (*lineJoining* $x'$ $y'$)) $\wedge$ ($r'$ *within* $e$ *of* $p'$))
**using** $y'$ **by** *blast*
**}**

**hence** $\forall\ y' \in s'.\ (y'\ \mathit{within}\ d\ \mathit{of}\ x') \wedge (y' \neq x')$
$\longrightarrow (\exists\, r'.\ (\mathit{onLine}\ r'\ (\mathit{lineJoining}\ x'\ y')) \wedge (r'\ \mathit{within}\ e\ \mathit{of}$
$p'))$
**by** *auto*

**hence** $\exists\, d{>}0.\ \forall\ y' \in s'.\ (y'\ \mathit{within}\ d\ \mathit{of}\ x') \wedge (y' \neq x')$
$\longrightarrow (\exists\, r'.\ (\mathit{onLine}\ r'\ (\mathit{lineJoining}\ x'\ y')) \wedge (r'\ \mathit{within}\ e\ \mathit{of}$
$p'))$
**using** *d* **by** *auto*
**}**
**hence** $\forall\, e{>}0.\ \exists\, d{>}0.\ \forall\ y' \in s'.\ (y'\ \mathit{within}\ d\ \mathit{of}\ x') \wedge (y' \neq x')$
$\longrightarrow (\exists\, r'.\ (\mathit{onLine}\ r'\ (\mathit{lineJoining}\ x'\ y')) \wedge (r'\ \mathit{within}\ e\ \mathit{of}$
$p'))$
**by** *force*
**hence** $(\mathit{onLine}\ p'\ l') \wedge (p' \neq x')$
$\wedge\ (\forall\, e{>}0.\ \exists\, d{>}0.\ \forall\ y' \in s'.\ (y'\ \mathit{within}\ d\ \mathit{of}\ x') \wedge (y' \neq x')$
$\longrightarrow (\exists\, r'.\ (\mathit{onLine}\ r'\ (\mathit{lineJoining}\ x'\ y')) \wedge (r'\ \mathit{within}\ e\ \mathit{of}$
$p')))$
**using** *p′-not-x′ p′-on-l′* **by** *auto*
**hence** *rtp4*: $\exists\ p'.\ (\ ((\mathit{onLine}\ p'\ l') \wedge (p' \neq x'))$
$\wedge\ (\forall\, e{>}0.\ \exists\, d{>}0.\ \forall\ y' \in s'.\ (y'\ \mathit{within}\ d\ \mathit{of}\ x') \wedge (y' \neq x')$
$\longrightarrow (\exists\, r'.\ (\mathit{onLine}\ r'\ (\mathit{lineJoining}\ x'\ y')) \wedge (r'\ \mathit{within}\ e\ \mathit{of}\ p'))))$
**by** *auto*

**hence** *?thesis* $\longleftrightarrow (x' \in s') \wedge (\mathit{onLine}\ x'\ l') \wedge (\mathit{accPoint}\ x'\ s')$
**using** $x'\ s'\ l'$ **by** *simp*
**thus** *?thesis* **using** *rtp1 rtp2 rtp3* **by** *blast*
**qed**

**lemma** *lemTangentLineA*:
**assumes** *tangentLine l s x*
**shows** *tangentLineA l s x*
**proof** −
**have** *1*: $(x \in s) \wedge (\mathit{onLine}\ x\ l) \wedge (\mathit{accPoint}\ x\ s)$ **using** *assms* **by**
*auto*

**have** $\exists\ P\ .\ (\mathit{onLine}\ P\ l) \wedge (P \neq x) \wedge$
$(\forall\ \varepsilon > 0\ .\ \ \exists\ \delta > 0\ .\ \forall\ y \in s.\ ($
$(\ (y\ \mathit{within}\ \delta\ \mathit{of}\ x) \wedge (y \neq x)\ )$
$\longrightarrow$
$(\ \exists\ r\ .\ ((\mathit{onLine}\ r\ (\mathit{lineJoining}\ x\ y)) \wedge (r\ \mathit{within}\ \varepsilon\ \mathit{of}\ P))))$
$)$
**using** *assms* **by** *simp*
**then obtain** *P* **where** *P*: $(\mathit{onLine}\ P\ l) \wedge (P \neq x) \wedge$
$(\forall\ \varepsilon > 0\ .\ \ \exists\ \delta > 0\ .\ \forall\ y \in s.\ ($
$(\ (y\ \mathit{within}\ \delta\ \mathit{of}\ x) \wedge (y \neq x)\ )$
$\longrightarrow$
$(\ \exists\ r\ .\ ((\mathit{onLine}\ r\ (\mathit{lineJoining}\ x\ y)) \wedge (r\ \mathit{within}\ \varepsilon\ \mathit{of}\ P))))$

```
    )
  by blast


{ fix p
  assume p: onLine p l ∧ p ≠ x

  hence onLine x l ∧ onLine p l ∧ x≠p using 1 by auto
  hence lxp: l = lineJoining x p
    using 1 lemLineAndPoints[of x p l] by auto

 then obtain a where a: P = (x ⊕ (a ⊗ (p⊖x))) using P by auto
  hence anz: a ≠ 0 using P by auto

  { fix e
    assume epos: e > 0
    hence aenz: a ∗ e ≠ 0 using anz by auto
    define e1 where e1: e1 = abs (a∗e)
    hence e1pos: e1 > 0 using aenz by auto

    then obtain d where d: (d > 0) ∧ ( ∀ y ∈ s. (
      ( (y within d of x) ∧ (y ≠ x) )
      ⟶
      ( ∃ r . ((onLine r (lineJoining x y)) ∧ (r within e1 of P))))
    )
      using P by auto

    { fix y
      assume y: (y ∈ s) ∧ (y within d of x) ∧ (y ≠ x)
      then obtain R
        where R: (onLine R (lineJoining x y)) ∧ (R within e1 of P)
        using d by blast

      define r where r: r = (x ⊕((1/a)⊗(R⊖x)))
      hence (r⊖x) = ((x ⊕((1/a)⊗(R⊖x))) ⊖ x) using r by auto
      also have ... = ((1/a)⊗(R⊖x))
        using add-commute add-assoc diff-add-cancel by auto
      finally have nrx: (r⊖x) = ((1/a)⊗(R⊖x)) by metis



      define T where T: T = mkTranslation (origin ⊖ x)
      hence transT: translation T using lemMkTrans by blast
      have R within e1 of P using R by simp
      hence (T R) within e1 of (T P)
        using transT lemBallTranslation[of T R P e1]
        by fastforce
     hence near1: ((1/a)⊗(R⊖x)) within (e1/a) of ((1/a)⊗(P⊖x))
        using lemScaleBall[of R⊖x P⊖x e1 1/a] anz T
        by auto
```

**define** $T'$ **where** $T'$: $T' = mkTranslation\ x$
**hence** $transT'$: $translation\ T'$ **using** *lemMkTrans* **by** *blast*
**hence** $near2$: $(T'\ ((1/a)\otimes(R\ominus x)))$ *within* $(e1/a)$ *of* $(T'\ ((1/a)\otimes(P\ominus x)))$
**using** *near1* $transT'$
*lemBallTranslation*[*of* $T'$ $(1/a)\otimes(R\ominus x)$ $(1/a)\otimes(P\ominus x)$ $e1/a$]
**by** *blast*

**have** $term1$: $(T'\ ((1/a)\otimes(R\ominus x))) = r$ **using** $T'$ *add-commute* $r$ **by** *auto*

**have** $(P \ominus x) = (a \otimes (p\ominus x))$ **using** $a$ **by** *auto*
**hence** $(T'\ ((1/a)\otimes(P\ominus x))) = (x \oplus ((1/a)\otimes(a \otimes (p\ominus x))))$
**using** $T'$ *add-commute* **by** *auto*
**hence** $(T'\ ((1/a)\otimes(P\ominus x))) = (x \oplus (p\ominus x))$
**using** *lemScaleAssoc*[*of* $1/a$ $a$ $P\ominus x$] *anz* **by** *auto*
**hence** $term2$: $(T'\ ((1/a)\otimes(P\ominus x))) = p$
**using** *diff-add-cancel add-commute* **by** *auto*

**have** $e1/a = abs\ (a*e)/a$ **using** $e1$ **by** *auto*
**hence** $sqr\ (e1/a) = (sqr\ (abs\ (a*e)))/\ (sqr\ a)$ **by** *auto*
**hence** $sqr\ (e1/a) = (sqr\ (a*e))/\ (sqr\ a)$ **by** *auto*
**hence** $sqr\ (e1/a) = (sqr\ a)*(sqr\ e)/(sqr\ a)$ **using** *lemSqrMult* **by** *auto*
**hence** $term3$: $sqr\ (e1/a) = (sqr\ e)$ **using** *anz* **by** *simp*

**hence** *r-near-p*: $r$ *within* $e$ *of* $p$ **using** *near2 term1 term2 term3* **by** *auto*

**have** *cases*: $(R = x) \vee (R \neq x)$ **by** *auto*
**have** *x-on-xy*: *onLine* $x$ (*lineJoining* $x$ $y$)
**using** $y$ *lemLineAndPoints*[*of* $x$ $y$ *lineJoining* $x$ $y$] **by** *auto*
**{ assume** $R = x$
**hence** $r = x$ **using** *nrx anz* **by** *auto*
**hence** *onLine* $r$ (*lineJoining* $x$ $y$) **using** *x-on-xy* **by** *blast*
**}**
**hence** $case1$: $(R = x) \longrightarrow$ (*onLine* $r$ (*lineJoining* $x$ $y$)) **by** *auto*
**{ assume** $R \neq x$
**hence** *lineJoining* $x$ $R$ = *lineJoining* $x$ $y$
**using** $R$ *x-on-xy lemLineAndPoints*[*of* $x$ $R$ *lineJoining* $x$ $y$]
**by** *auto*
**hence** *onLine* $r$ (*lineJoining* $x$ $y$) **using** $r$ **by** *blast*
**}**
**hence** $(R \neq x) \longrightarrow$ (*onLine* $r$ (*lineJoining* $x$ $y$)) **by** *auto*
**hence** *onLine* $r$ (*lineJoining* $x$ $y$) **using** *cases case1* **by** *auto*

**hence** $\exists\ r.$ (*onLine* $r$ (*lineJoining* $x$ $y$)) $\wedge$ ($r$ *within* $e$ *of* $p$)

**using** *r-near-p* **by** *auto*
**}**
**hence** $\forall\, y \in s$ . $(y$ *within* $d$ *of* $x) \wedge (y \neq x)$
$\longrightarrow (\exists\ r.\ (onLine\ r\ (lineJoining\ x\ y)) \wedge (r\ within\ e\ of\ p))$
**by** *auto*
**hence** $\exists\, d{>}0.\ \forall\, y \in s$ . $(y$ *within* $d$ *of* $x) \wedge (y \neq x)$
$\longrightarrow (\exists\ r.\ (onLine\ r\ (lineJoining\ x\ y)) \wedge (r\ within\ e\ of\ p))$
**using** *d* **by** *auto*
**}**
**hence** $\forall\, e{>}0.\ \exists\, d{>}0.\ \forall\, y \in s$ . $(y$ *within* $d$ *of* $x) \wedge (y \neq x)$
$\longrightarrow (\exists\ r.\ (onLine\ r\ (lineJoining\ x\ y)) \wedge (r\ within\ e\ of\ p))$
**by** *blast*
**}**
**hence** *2*: $\forall\, p$ . $(onLine\ p\ l \wedge p \neq x) \longrightarrow$
$(\forall\, e{>}0.\ \exists\, d{>}0.\ \forall\, y \in s$ . $(y$ *within* $d$ *of* $x) \wedge (y \neq x)$
$\longrightarrow (\exists\ r.\ (onLine\ r\ (lineJoining\ x\ y)) \wedge (r\ within\ e\ of$
$p)))$
**by** *blast*
**thus** *?thesis* **using** *1* **by** *auto*
**qed**

**lemma** *lemTangentLineE*:
**assumes** *tangentLineA l s x*
**and** $\exists\, p \neq x$ . *onLine p l*
**shows** *tangentLine l s x*
**proof** −
**have** *1*: $(x \in s) \wedge (onLine\ x\ l) \wedge (accPoint\ x\ s)$ **using** *assms*(*1*) **by** *auto*

**obtain** *p* **where** *p*: $(p \neq x) \wedge (onLine\ p\ l)$ **using** *assms*(*2*) **by** *auto*
**hence** $\forall\ \varepsilon > 0\ .\ \exists\ \delta > 0\ .\ \forall\ y \in s.\ ($
$(\ (y\ within\ \delta\ of\ x) \wedge (y \neq x)\ )$
$\longrightarrow$
$(\ \exists\ r\ .\ ((onLine\ r\ (lineJoining\ x\ y)) \wedge (r\ within\ \varepsilon\ of\ p))))$
**using** *assms*(*1*) **by** *blast*
**thus** *?thesis* **using** *1 p* **by** *auto*
**qed**

**end**

**end**

# 9 Cones

This theory defines (light)cones, regular cones, and their properties.

**theory** *Cones*
  **imports** *WorldLine TangentLines*
**begin**

**class** *Cones* = *WorldLine* + *TangentLines*
**begin**

**abbreviation** *tl* :: $'a$ *Point set* ⇒ *Body* ⇒ *Body* ⇒ $'a$ *Point* ⇒ *bool*
  **where** *tl l m b x* ≡ *tangentLine l* (*wline m b*) *x*

The cone of a body at a point comprises the set of points that lie on tangent lines of photons emitted by the body at that point.

**abbreviation** *cone* :: *Body* ⇒ $'a$ *Point* ⇒ $'a$ *Point* ⇒ *bool*
  **where** *cone m x p*
    ≡ ∃ *l* . (*onLine p l*) ∧ (*onLine x l*) ∧ (∃ *ph* . *Ph ph* ∧ *tl l m ph x*)

**abbreviation** *regularCone* :: $'a$ *Point* ⇒ $'a$ *Point* ⇒ *bool*
  **where** *regularCone x p* ≡ ∃ *l* . (*onLine p l*) ∧ (*onLine x l*)
    ∧ (∃ *v* ∈ *lineVelocity l* . *sNorm2 v* = *1*)

**abbreviation** *coneSet* :: *Body* ⇒ $'a$ *Point* ⇒ $'a$ *Point set*
  **where** *coneSet m x* ≡ { *p* . *cone m x p* }

**abbreviation** *regularConeSet* :: $'a$ *Point* ⇒ $'a$ *Point set*
  **where** *regularConeSet x* ≡ { *p* . *regularCone x p* }

**end**

**end**

# 10 AXIOM: AxLightMinus

This theory declares the axiom AxLightMinus.

**theory** *AxLightMinus*
  **imports** *WorldLine TangentLines*

**begin**

AxLightMinus: If an observer sends out a light signal, then the speed of the light signal is 1 according to the observer. Moreover it is possible to send out a light signal in any direction.

**class** *axLightMinus* = *WorldLine* + *TangentLines*
**begin**

The definition of AxLightMinus used in this Isabelle proof is slightly different to the one used in the paper-based proof on which it is based. We have established elsewhere, however, that each entails the other in all relevant contexts.

**abbreviation** $axLightMinusOLD :: Body \Rightarrow 'a\ Point \Rightarrow 'a\ Space \Rightarrow bool$
**where** $axLightMinusOLD\ m\ p\ v \equiv (m\ sees\ m\ at\ p) \longrightarrow ($
$(\exists\ ph\ .\ (Ph\ ph \wedge (vel\ (wline\ m\ ph)\ p\ v))) \longleftrightarrow (sNorm2\ v = 1)$
$)$

**abbreviation** $axLightMinus :: Body \Rightarrow 'a\ Point \Rightarrow 'a\ Space \Rightarrow bool$
**where** $axLightMinus\ m\ p\ v \equiv (m\ sees\ m\ at\ p)$
$\longrightarrow (\ \forall\ l\ .\ \forall\ v \in lineVelocity\ l\ .$
$(\exists\ ph\ .\ (Ph\ ph \wedge (tangentLine\ l\ (wline\ m\ ph)\ p))) \longleftrightarrow$
$(sNorm2\ v = 1))$

**end**

**class** *AxLightMinus* = *axLightMinus* +
**assumes** *AxLightMinus*: $\forall\ m\ p\ v\ .\ axLightMinus\ m\ p\ v$
**begin**
**end**

**end**

# 11 Proposition1

This theory shows that observers consider their own lightcones to be upright.

**theory** *Proposition1*
**imports** *Cones AxLightMinus*
**begin**

**class** *Proposition1* = *Cones* + *AxLightMinus*
**begin**

**lemma** *lemProposition1*:

**assumes** $x \in$ *wline m m*
**shows** *cone m x p* = *regularCone x p*
**proof** –
**have** *mmx*: *m sees m at x* **using** *assms* **by** *simp*

**have** *axlight*: $\forall$ *l* . $\forall$ *v* $\in$ *lineVelocity l* .
($\exists$ *ph* . (*Ph ph* $\wedge$ (*tangentLine l* (*wline m ph*) *x*))) $\longleftrightarrow$ (*sNorm2 v* = *1*)
**using** *AxLightMinus mmx* **by** *auto*

**define** *axph* **where** *axph*: *axph* = ($\lambda$ *l* . $\lambda$ *ph* . (*Ph ph* $\wedge$ (*tangentLine l* (*wline m ph*) *x*)))

**define** *lhs* **where** *lhs*: *lhs* = *cone m x p*
**define** *rhs* **where** *rhs*: *rhs* = *regularCone x p*

{ **assume** *lhs*
**hence** $\exists$ *l* . *onLine p l* $\wedge$ *onLine x l* $\wedge$ ($\exists$ *ph* . *axph l ph*)
**using** *lhs axph* **by** *auto*
**then obtain** *l*
**where** *l*: *onLine p l* $\wedge$ *onLine x l* $\wedge$ ($\exists$ *ph* . *axph l ph*) **by** *auto*

**have** *xonl*: *onLine x l* **using** *l* **by** *auto*
**have** *ponl*: *onLine p l* **using** *l* **by** *auto*

**have** *exph*: $\exists$ *ph* . *axph l ph* **using** *l* **by** *auto*
**then obtain** *ph* **where** *ph*: *axph l ph* **by** *auto*

**have** *axlight'*: $\forall$ *v* $\in$ *lineVelocity l* . ($\exists$ *ph* . *axph l ph*) $\longleftrightarrow$ (*sNorm2 v* = *1*)
**using** *axph axlight* **by** *force*

**hence** *lv1*: $\forall$ *v* $\in$ *lineVelocity l* . (*sNorm2 v* = *1*) **using** *exph* **by** *blast*

**have** *tterm1*: *tl l m ph x* **using** *ph axph* **by** *force*

**hence** $\exists$ *p* . ( (*onLine p l*) $\wedge$ ($p \neq x$) $\wedge$ ($\forall$ $\varepsilon > 0$ . $\exists$ $\delta > 0$ . $\forall$ $y \in$ (*wline m ph*). (
( (*y within $\delta$ of x*) $\wedge$ ($y \neq x$) ) $\longrightarrow$
( $\exists$ *r* . ((*onLine r* (*lineJoining x y*)) $\wedge$ (*r within $\varepsilon$ of p*))))))
**by** *auto*
**then obtain** *q* **where** *q*: *onLine q l* $\wedge$ $q \neq x$ **by** *auto*
**define** *qx* **where** *qx*: *qx* = ($q \ominus x$)

**hence** $(x \neq q) \wedge$ *onLine x l* $\wedge$ *onLine q l* $\wedge$ $(qx = (q \ominus x))$ **using** *q xonl* **by** *auto*
**hence** $\exists$ *p q* . $(p \neq q) \wedge$ *onLine p l* $\wedge$ *onLine q l* $\wedge$ $(qx = (q \ominus p))$ **by** *blast*
**hence** *qxl*: $qx \in$ *drtn l* **by** *auto*

**define** *v* **where** *v*: $v =$ *velocityJoining origin qx*
**hence** $\exists$ $d \in$ *drtn l* . $v =$ *velocityJoining origin d* **using** *qxl* **by** *blast*
**hence** *existsv*: $v \in$ *lineVelocity l* **by** *auto*
**hence** *norm2v*: *sNorm2 v* $= 1$ **using** *lv1* **by** *auto*
**hence** $\exists$ $v \in$ *lineVelocity l* . *sNorm2 v* $= 1$ **using** *existsv* **by** *force*

**hence** *onLine p l* $\wedge$ *onLine x l* $\wedge$ ($\exists$ $v \in$ *lineVelocity l* . *sNorm2 v* $= 1$)
**using** *ponl xonl* **by** *auto*
**hence** $\exists$ *l* . *onLine p l* $\wedge$ *onLine x l* $\wedge$ ($\exists$ $v \in$ *lineVelocity l* . *sNorm2 v* $= 1$)
**by** *blast*
**hence** *regularCone x p* **by** *auto*
**}**
**hence** *l2r*: *lhs* $\longrightarrow$ *rhs* **using** *rhs* **by** *blast*

**{ assume** *rhs*
**hence** $\exists$ *l* . *onLine p l* $\wedge$ *onLine x l* $\wedge$ ($\exists$ $v \in$ *lineVelocity l* . *sNorm2 v* $= 1$)
**using** *rhs* **by** *auto*
**then obtain** *l*
**where** *l*: (*onLine p l*) $\wedge$ (*onLine x l*) $\wedge$ ($\exists$ $v \in$ *lineVelocity l* . *sNorm2 v* $= 1$)
**by** *blast*

**have** *xonl*: *onLine x l* **using** *l* **by** *auto*
**have** *ponl*: *onLine p l* **using** *l* **by** *auto*

**have** $\exists$ $v \in$ *lineVelocity l* . *sNorm2 v* $= 1$ **using** *l* **by** *blast*
**then obtain** *v* **where** *v*: ($v \in$ *lineVelocity l*) $\wedge$ (*sNorm2 v* $= 1$) **by** *blast*

**define** *final*
**where** *final*: *final* $=$ ($\lambda$ *l* . *onLine p l* $\wedge$ *onLine x l* $\wedge$ ($\exists$ *ph* . *axph l ph*))

**have** $\exists$ *ph* . *axph l ph* **using** *v axlight axph* **by** *blast*
**hence** *final l* **using** *ponl xonl final* **by** *auto*
**hence** $\exists$ *l* . *final l* **by** *auto*
**hence** *cone m x p* **using** *final axph* **by** *auto*
**hence** *lhs* **using** *lhs* **by** *auto*

}
**hence** *r2l*: *rhs* $\longrightarrow$ *lhs* **using** *lhs* **by** *blast*

**hence** *lhs* $\longleftrightarrow$ *rhs* **using** *l2r* **by** *auto*
**thus** *?thesis* **using** *lhs rhs* **by** *auto*
**qed**

**end**

**end**

# 12 AXIOM: AxEField

This theory defines the axiom AxEField, which states that the linearly ordered field of quantities is Euclidean, i.e. that all non-negative values have square roots in the field.

**theory** *AxEField*
**imports** *Sorts*
**begin**

**class** *axEField* = *Quantities*
**begin**
**abbreviation** *axEField* :: $'a \Rightarrow$ *bool*
**where** *axEField x* $\equiv (x \geq 0) \longrightarrow$ *hasRoot x*
**end**

**class** *AxEField* = *axEField* +
**assumes** *AxEField*: $\forall$ *x* . *axEField x*
**begin**
**end**

**end**

# 13 Norms

This theory defines norms, assuming that roots exist.

**theory** *Norms*
**imports** *Points AxEField*
**begin**

**class** *Norms* = *Points* + *AxEField*

**begin**

**abbreviation** *norm* :: *'a Point* ⇒ *'a* (‖ - ‖)
  **where** *norm p* ≡ *sqrt* (*norm2 p*)

**abbreviation** *sNorm* :: *'a Space* ⇒ *'a*
  **where** *sNorm p* ≡ *sqrt* (*sNorm2 p*)

## 13.1 axTriangleInequality

Given that norms exist, we can define the triangle inequality for specific cases. This will be asserted more generally as an axiom later.

**abbreviation** *axTriangleInequality* :: *'a Point* ⇒ *'a Point* ⇒ *bool*
  **where** *axTriangleInequality p q* ≡ (*norm* (*p*⊕*q*) ≤ *norm p* + *norm q*)

**lemma** *lemNormSqrIsNorm2*: *norm2 p* = *sqr* (*norm p*)
**proof** −
  **have** *norm2 p* ≥ *0* **by** *simp*
  **moreover have** *axEField* (*norm2 p*) **using** *AxEField* **by** *simp*
  **ultimately show** *?thesis* **using** *lemSquareOfSqrt*[*of norm2 p norm p*] **by** *force*
**qed**

**lemma** *lemZeroNorm*:
  **shows** (*p* = *origin*) ⟷ (*norm p* = *0*)
**proof** −
  { **assume** *p* = *origin*
    **hence** *norm2 p* = *0* **by** *auto*
    **hence** *norm p* = *0* **using** *lemSquareOfSqrt lemZeroRoot AxEField* **by** *force*
  }
  **hence** *l2r*: (*p* = *origin*) ⟶ (*norm p* = *0*) **by** *auto*

  { **assume** *norm p* = *0*
    **hence** *norm2 p* = *0* **using** *lemNormSqrIsNorm2*[*of p*] **by** *auto*
    **hence** *p* = *origin* **using** *lemNullImpliesOrigin* **by** *auto*
  }
  **hence** (*norm p* = *0*) ⟶ (*p* = *origin*) **by** *auto*

**thus** *?thesis* **using** *l2r* **by** *blast*
**qed**

**lemma** *lemNormNonNegative*: *norm p* $\geq$ *0*
**proof** $-$
**have** *norm2 p* $\geq$ *0* **by** *auto*
**hence** *unique*: $\exists!r.$ *0* $\leq$ *r* $\wedge$ *norm2 p* = *sqr r* **using** *AxEField lemSqrt*[*of norm2 p*] **by** *auto*
**then obtain** *r* **where** *r*: *0* $\leq$ *r* $\wedge$ *norm2 p* = *sqr r* $\wedge$ ($\forall$ *x* . *isNonNegRoot* (*norm2 p*) *x* $\longrightarrow$ *x*= *r*)
**by** *auto*
**hence** *r* = *norm p* **using** *the-equality*[*of isNonNegRoot* (*norm2 p*) *r*] **by** *blast*
**moreover have** *r* $\geq$ *0* **using** *r* **by** *blast*
**ultimately show** *?thesis* **by** *auto*
**qed**

**lemma** *lemNotOriginImpliesPositiveNorm*:
**assumes** *p* $\neq$ *origin*
**shows** (*norm p* > *0*)
**proof** $-$
**have** *1*: *norm p* $\neq$ *0* **using** *lemZeroNorm assms*(*1*)**by** *auto*
**have** *norm p* $\geq$ *0* **using** *lemNormNonNegative assms*(*1*) **by** *auto*
**hence** *2*: *norm p* > *0* **using** *1* **by** *auto*
**thus** *?thesis* **by** *auto*
**qed**

**lemma** *lemNormSymmetry*: *norm* (*p*$\ominus$*q*) = *norm* (*q*$\ominus$*p*)
**proof** $-$
**have** *norm2* (*p* $\ominus$ *q*) = *norm2* (*q* $\ominus$ *p*) **using** *lemSep2Symmetry* **by** *simp*
**thus** *?thesis* **by** *presburger*
**qed**

**lemma** *lemNormOfScaled*: *norm* ($\alpha$$\otimes$*p*) = (*abs* $\alpha$) $*$ (*norm p*)
**proof** $-$
**have** *sqr* (*norm* ($\alpha$$\otimes$*p*)) = *norm2* ($\alpha$$\otimes$*p*) **using** *lemNormSqrIsNorm2* **by** *presburger*
**also have** ... = (*sqr* $\alpha$)$*$(*norm2 p*) **using** *lemNorm2OfScaled* **by** *auto*
**also have** ... = (*sqr* $\alpha$)$*$(*sqr* (*norm p*)) **using** *lemNormSqrIsNorm2* **by** *force*

**also have** ... = *sqr* ( $\alpha *$(*norm p*) ) **using** *lemSqrMult* **by** *auto*
**finally have** *abs* (*norm* ($\alpha\otimes p$)) = *abs* ( $\alpha$ *(*norm p*) ) **using** *lemEqualSquares* **by** *blast*
**moreover have** *abs* (*norm* ($\alpha\otimes p$)) = *norm* ($\alpha\otimes p$)
**using** *lemNormNonNegative*[*of* ($\alpha\otimes p$)] *abs-of-nonneg* **by** *auto*
**moreover have** *abs* ( $\alpha$ *(*norm p*) ) = (*abs* $\alpha$)*(*abs* (*norm p*)) **using** *abs-mult* **by** *auto*
**ultimately show** *?thesis* **using** *lemNormNonNegative*[*of p*] *abs-of-nonneg* **by** *auto*
**qed**

**lemma** *lemDistancesAdd*:
**assumes** *triangle*: *axTriangleInequality* ($q\ominus p$) ($r\ominus q$)
**and** *distances*: ($x > 0$) $\wedge$ ($y > 0$) $\wedge$ (*sep2 p q* < *sqr x*) $\wedge$ (*sep2 r q* < *sqr y*)
**shows** *r within* ($x+y$) *of p*
**proof** −
**define** *npq* **where** *npq*: *npq* = *norm* ($q\ominus p$)
**hence** *sqr npq* < *sqr x*
**using** *lemNormSqrIsNorm2 distances lemSep2Symmetry* **by** *presburger*
**hence** *npqx*: *npq* < *x* **using** *lemSqrOrderedStrict distances* **by** *blast*

**define** *nqr* **where** *nqr*: *nqr* = *norm* ($r\ominus q$)
**hence** *sqr nqr* < *sqr y* **using** *lemNormSqrIsNorm2 distances* **by** *presburger*
**hence** *nqry*: *nqr* < *y* **using** *lemSqrOrderedStrict distances* **by** *blast*

**have** *rminusp*: ($r\ominus p$) = (($q\ominus p$)$\oplus$($r\ominus q$)) **using** *lemDiffDiffAdd* **by** *fastforce*
**define** *npr* **where** *npr*: *npr* = *norm* ($r\ominus p$)

**have** *nx*: *norm* ($q\ominus p$) = *npq* **using** *npq lemSqrt* **by** *fast*
**have** *ny*: *norm* ($r\ominus q$) = *nqr* **using** *nqr lemSqrt* **by** *fast*
**have** *nz*: *norm* ($r\ominus p$) = *npr* **using** *npr lemSqrt* **by** *fast*

**have** *norm* ($r\ominus p$) $\leq$ (*norm* ($q\ominus p$) + *norm* ($r\ominus q$)) **using** *triangle rminusp* **by** *fastforce*
**hence** *npr* $\leq$ (*npq* + *nqr*) **using** *nx ny nz lemSqrt npq nqr npr* **by** *simp*
**hence** *npr* < *x* + *y* **using** *npqx nqry add-strict-mono*[*of npq x nqr y*]
**by** *simp*

**hence** *sqr npr* < *sqr* ($x+y$) **using** *npr lemNormNonNegative*[*of* ($r\ominus p$)] *lemSqrMonoStrict* **by** *auto*
**hence** *sep*: *sep2 r p* < *sqr* ($x+y$) **using** *npr lemSquareOfSqrt AxEField* **by** *auto*

**thus** *?thesis* **using** *npr lemSep2Symmetry* **by** *auto*
**qed**

**lemma** *lemDistancesAddStrictR*:
**assumes** *triangle*: *axTriangleInequality* $(q \ominus p)$ $(r \ominus q)$
**and** *distances*: $(x > 0) \wedge (y > 0) \wedge (sep2\ p\ q \leq sqr\ x) \wedge (sep2\ r\ q < sqr\ y)$
**shows** *r within* $(x+y)$ *of p*
**proof** −
**define** *npq* **where** *npq*: $npq = norm\ (q \ominus p)$
**hence** $sqr\ npq \leq sqr\ x$ **using** *lemNormSqrIsNorm2 distances lemSep2Symmetry* **by** *presburger*
**hence** *npqx*: $npq \leq x$ **using** *lemSqrOrdered*[*of x npq*] *distances npq* **by** *auto*

**define** *nqr* **where** *nqr*: $nqr = norm\ (r \ominus q)$
**hence** $sqr\ nqr < sqr\ y$ **using** *lemNormSqrIsNorm2 distances* **by** *presburger*
**hence** *nqry*: $nqr < y$ **using** *lemSqrOrderedStrict distances* **by** *blast*

**define** *npr* **where** *npr*: $npr = norm\ (r \ominus p)$

**have** *nx*: $norm\ (q \ominus p) = npq$ **using** *npq lemSqrt* **by** *blast*
**have** *ny*: $norm\ (r \ominus q) = nqr$ **using** *nqr lemSqrt* **by** *blast*
**have** *nz*: $norm\ (r \ominus p) = npr$ **using** *npr lemSqrt* **by** *blast*

**have** $norm\ (r \ominus p) \leq (norm\ (q \ominus p) + norm\ (r \ominus q))$ **using** *triangle lemDiffDiffAdd* **by** *fastforce*
**hence** $npr \leq (npq + nqr)$ **using** *nx ny nz* **by** *simp*
**hence** $npr < x + y$ **using** *npqx nqry add-le-less-mono*[*of npq x nqr y*]
**by** *auto*

**hence** $sqr\ npr < sqr\ (x+y)$ **using** *npr lemNormNonNegative*[*of* $(r \ominus p)$] *lemSqrMonoStrict* **by** *auto*
**hence** *sep*: $sep2\ r\ p < sqr\ (x+y)$ **using** *npr lemSquareOfSqrt AxEField* **by** *auto*

**thus** *?thesis* **using** *npr lemSep2Symmetry*[*of r p*] **by** *auto*
**qed**

**end**

**end**

# 14 AxTriangleInequality

This theory declares the Triangle Inequality as an axiom.

**theory** *AxTriangleInequality*
  **imports** *Norms*
**begin**

Although AxTriangleInequality can be proven rather than asserted we have left it as an axiom to illustrate the flexibility of using Isabelle for mathematical physics: well-known mathematical results can be asserted, leaving the researcher free to concentrate on the physics. We can return later to prove the mathematical results when time permits.

**class** *AxTriangleInequality* = *Norms* +
  **assumes** *AxTriangleInequality*: $\forall$ *p q* . *axTriangleInequality p q*
**begin**
**end**

**end**

# 15 Sublemma3

This theory establishes how closely tangent lines approximate world lines.

**theory** *Sublemma3*
  **imports** *WorldLine AxTriangleInequality TangentLines*
**begin**

**class** *Sublemma3* = *WorldLine* + *AxTriangleInequality* + *TangentLines*
**begin**

**lemma** *sublemma3*:
**assumes** *onLine p l*
**and** *norm2 p* = *1*
**and** *tangentLine l wl origin*
**shows**
$\forall$ $\varepsilon$ > *0* . $\exists$ $\delta$ > *0* . $\forall$ *y ny* . (
  ((*y within* $\delta$ *of origin*) $\wedge$ (*y* $\neq$ *origin*) $\wedge$ (*y* $\in$ *wl*) $\wedge$ (*norm y* = *ny*))
   $\longrightarrow$
  ( (((*1*/*ny*)$\otimes$*y*) *within* $\varepsilon$ *of p*) $\vee$ (((−*1*/*ny*)$\otimes$*y*) *within* $\varepsilon$ *of p*))
)

```
proof –
  { fix e :: 'a
    { assume epos: e > 0

      hence e2pos: e/2 > 0 by simp

      have prop1: origin ∈ wl  using assms(3) by auto
      have prop2: onLine origin l  using assms(3) by auto
       hence prop3: ∀ ε > 0. ∃ q ∈ wl. (origin ≠ q) ∧ (inBall q ε
origin)
        using assms(3) by auto


      have prop4: ∀ p .( ((onLine p l) ∧ (p ≠ origin)) ⟶
          (∀ ε > 0 . ∃ δ > 0 . ∀ y ∈ wl . (
            ( (y within δ of origin) ∧ (y ≠ origin) )
            ⟶
             ( ∃ r . ((onLine r (lineJoining origin y)) ∧ (r within ε of
p))))
          )
       ) using assms(3) lemTangentLineA[of origin]
        by auto



      have p ≠ origin using assms(2) lemNullImpliesOrigin by auto

      hence ballprops: ∀ ε > 0 . ∃ δ > 0 . ∀ y ∈ wl . (
            ( (y within δ of origin) ∧ (y ≠ origin) )
            ⟶
             ( ∃ r . ((onLine r (lineJoining origin y)) ∧ (r within ε of
p)))
        )
        using assms(1) prop4 by auto


     define eps where eps = (if (e/2 < 1/2) then (e/2) else (1/2))

      hence eps-le-e2: eps ≤ e/2 by auto

      have epspos: eps > 0 using e2pos eps-def by simp

      { assume ass1: e/2 < 1/2
        hence eps = e/2 using eps-def by auto
        hence eps < 1/2 using ass1 by simp
        hence eps ≤ 1/2 by simp
      }
      hence case1: (e/2 < 1/2) ⟶ eps ≤ 1/2 by auto
      have ¬(e/2 < 1/2) ⟶ eps = 1/2 using eps-def by simp
```

**hence** *case2*: $\neg(e/2 < 1/2) \longrightarrow eps \leq 1/2$ **by** *auto*
**hence** $(eps \leq (1/2))$ **using** *case1 case2* **by** *auto*
**hence** *eps-lt-1*: $eps < 1$ **using** *le-less-trans* **by** *auto*
**hence** *sqr eps* $<$ *eps* **using** *epspos lemMultPosLT1* **by** *auto*
**hence** *epssqu*: *sqr eps* $< 1$ **using** *eps-lt-1 le-less-trans* **by** *auto*

**then obtain** $d$ **where** *dprops*: $(d > 0) \wedge (\forall\ y \in wl.$ (
( (*y within d of origin*) $\wedge$ ($y \neq$ *origin*) )
$\longrightarrow$
( $\exists\ r$ . ((*onLine r* (*lineJoining origin y*)) $\wedge$ (*r within eps of p*))))
) **using** *epspos ballprops* **by** *auto*

**{ fix** *y ny* **assume** *ny*: *ny* = *norm y*
**{ assume** *y*: (*y within d of origin*) $\wedge$ ($y \neq$ *origin*) $\wedge$ ($y \in wl$)

**hence** $\exists\ r$ . ((*onLine r* (*lineJoining origin y*)) $\wedge$ (*r within eps of p*))
**using** *dprops* **by** *blast*
**then obtain** *r*
**where** *r*: (*onLine r* (*lineJoining origin y*)) $\wedge$ (*r within eps of p*)
**by** *auto*

**hence** $\exists\ \alpha$ . $r = (\alpha \otimes y)$ **by** *simp*
**then obtain** $\alpha$ **where** *alpha*: $r = (\alpha \otimes y)$ **by** *auto*

**{ assume** $\alpha = 0$
**hence** *rnull*: $r$ = *origin* **using** *alpha* **by** *simp*
**hence** *one*: *sep2 r p* $= 1$ **using** *assms*(2) **by** *auto*
**have** *sep2 r p* $<$ *sqr eps* **using** *r* **by** *auto*
**hence** *not-one*: *sep2 r p* $< 1$ **using** *epssqu* **by** *auto*
**hence** *False* **using** *one not-one* **by** *auto*
**}**
**hence** *anz*: $\alpha \neq 0$ **by** *auto*

**define** *np* **where** *np* = *norm p*
**hence** *np*: *np* $= 1$ **using** *assms*(2) *lemSqrt1* **by** *auto*

**define** *npr* **where** *npr* = *norm* $(p \ominus r)$
**hence** *sqr npr* = *sep2 p r* **using** *local.lemNormSqrIsNorm2* **by** *presburger*
**hence** *sqr npr* $<$ *sqr eps* **using** *r lemSep2Symmetry* **by** *auto*
**hence** *sqr npr* $<$ *sqr eps* $\wedge$ *eps* $> 0$ **using** *epspos* **by** *auto*
**hence** *npr*: *npr* $<$ *eps*
**using** *lemSqrOrderedStrict*[*of eps npr*] **by** *auto*
**hence** *npr1*: $1 -$ *npr* $> 1 -$ *eps*

**using** *diff-strict-left-mono* **by** *simp*

**have** *npr-lt-e2*: $npr < e/2$ **using** *npr eps-le-e2 le-less-trans* **by** *auto*

**define** *nr* **where** $nr = norm\ r$
**hence** $sqr\ nr = norm2\ (\alpha \otimes y)$ **using** *alpha lemNormSqrIsNorm2* **by** *presburger*
**hence** *nr*: $sqr\ nr = (sqr\ \alpha) * norm2\ y$ **using** *lemNorm2OfScaled* **by** *auto*

**have** *axTriangleInequality* $(p\ominus r)\ r$ **using** *AxTriangleInequality* **by** *blast*
**hence** $(np \leq npr + nr)$ **using** *np-def npr-def nr-def* **by** *simp*
**hence** $nr \geq 1 - npr$ **using** *np lemLEPlus* **by** *auto*
**hence** *triangle1*: $nr > 1 - eps$ **using** *npr1 le-less-trans* **by** *simp*

**define** *nrp* **where** $nrp = norm\ (r\ominus p)$
**hence** *nrppr*: $nrp = npr$ **using** *npr-def nrp-def lemSep2Symmetry*[*of p r*] **by** *auto*

**have** *axTriangleInequality* $(r\ominus p)\ p$ **using** *AxTriangleInequality* **by** *blast*
**hence** $(nr \leq npr + 1)$
**using** *np-def npr-def nr-def np nrp-def nrppr* **by** *auto*
**hence** *triangle2*: $nr < 1 + eps$
**using** *npr add-strict-right-mono le-less-trans add-commute* **by** *simp*

**have** *range*: $(1 - eps) < nr < (1 + eps)$
**using** *triangle1 triangle2* **by** *simp*

**have** $(ny = 0) \longrightarrow (y = origin)$
**using** *ny lemNormSqrIsNorm2*[*of y*] *lemNullImpliesOrigin*
**by** *auto*
**hence** *nynz*: $ny \neq 0$ **using** *y* **by** *auto*

**have** $norm\ ((1/ny)\otimes y) = ((abs\ (1/ny)) * ny)$ **using** *ny lemNormOfScaled*[*of 1/ny y*] **by** *auto*
**hence** *nyunit*: $norm\ ((1/ny)\otimes y) = 1$ **using** *y nynz ny lemNormNonNegative* **by** *auto*

**have** $norm\ r = ((abs\ \alpha) * ny)$ **using** *ny alpha lemNormOfScaled*[*of* $\alpha$ *y*] **by** *auto*
**hence** *nr-is-any*: $nr = ((abs\ \alpha) * ny)$ **using** *nr-def lemSqrt* **by** *auto*

**hence** $(1 - eps) < ((abs\ \alpha) * ny) < (1 + eps)$ **using** *range* **by** *auto*
**hence** *star*: $abs\ (((abs\ \alpha) * ny) - 1) < eps$
**using** *epspos lemAbsRange*[*of eps 1* $((abs\ \alpha) * ny)$] **by** *auto*

**have** *cases*: $(\alpha > 0) \vee (\alpha < 0)$ **using** *anz* **by** *auto*

**{ assume** *apos*: $\alpha > 0$
**hence** $abs\ \alpha = \alpha$ **by** *auto*
**hence** *case1range*: $abs\ ((\alpha * ny) - 1) < eps$ **using** *star* **by** *auto*

**define** *w1* **where** $w1 = ((\alpha \otimes y) \ominus ((1/ny)\otimes y))$
**define** *nw1* **where** $nw1 = norm\ w1$

**have** $(\alpha \otimes y) = ((1/ny) \otimes ((\alpha * ny) \otimes y))$
**using** *nynz lemScaleAssoc* **by** *auto*
**hence** $w1 = (((1/ny) \otimes ((\alpha * ny) \otimes y)) \ominus ((1/ny)\otimes y))$
**using** *w1-def* **by** *simp*
**hence** $w1 = ((1/ny) \otimes\ (((\alpha * ny) \otimes y) \ominus y\ ))$
**using** *lemScaleDistribDiff* [*of* $1/ny$ $(\alpha * ny) \otimes y\ y$] **by** *force*
**hence** $w1 = (((\alpha * ny) - 1) \otimes ((1/ny) \otimes y))$
**using** *lemScaleLeftDiffDistrib lemScaleCommute* **by** *auto*
**hence** *2*: $norm\ w1 = (abs\ ((\alpha * ny) - 1))$
**using** *lemNormOfScaled*[*of* $((\alpha * ny) - 1)$ $(1/ny) \otimes y$] *nyunit* **by** *auto*

**{**
**define** *pp* **where** *pp*: $pp = (p\ominus(\alpha\otimes y))$
**define** *qq* **where** *qq*: $qq = ((\alpha\otimes y) \ominus ((1/ny)\otimes y))$
**have** *axTriangleInequality pp qq* **using** *AxTriangleInequality* **by** *simp*
**hence** $norm\ (pp \oplus qq) \leq norm\ pp + norm\ qq$ **by** *auto*
**hence** $norm\ ((p \ominus ((1/ny)\otimes y))) \leq norm\ pp + norm\ qq$
**using** *lemSumDiffCancelMiddle pp qq* **by** *simp*
**hence** $norm\ ((p \ominus ((1/ny)\otimes y))) \leq norm\ (p\ominus r) + norm\ w1$

**using** *alpha w1-def pp qq* **by** *auto*
**}**
**hence** *3*: $norm\ ((p \ominus ((1/ny)\otimes y))) \leq npr + nw1$
**using** *nw1-def npr-def* **by** *force*

**define** *nminus* **where** $nminus = norm\ ((p \ominus ((1/ny)\otimes y)))$

**hence** *almost1*: $nminus \leq npr + nw1$ **using** *3 nminus-def* **by** *auto*

**have** *abs* $((ny * \alpha) - 1) \geq 0$ **by** *auto*
**hence** $nw1 = abs\ ((\alpha * ny) - 1)$ **using** *nw1-def 2 lemSqrt* **by** *blast*
**hence** $nw1 < eps$ **using** *case1range le-less-trans* **by** *auto*
**hence** $nw1 < e/2$ **using** *eps-le-e2 le-less-trans* **by** *auto*

**hence** $nminus < (e/2 + e/2)$
**using** *almost1 npr-lt-e2 add-strict-mono le-less-trans* **by** *simp*

**hence** $nminus < e$ **using** *lemSumOfTwoHalves* **by** *simp*

**hence** $sqr\ nminus < sqr\ e$
**using** *lemSqrMonoStrict*[*of nminus e*] *nminus-def*
*lemNormNonNegative*[*of* $((p \ominus ((1/ny) \otimes y)))$]
**by** *auto*

**hence** $norm2\ ((p \ominus ((1/ny) \otimes y))) < sqr\ e$
**using** *lemNormSqrIsNorm2*[*of* $((p \ominus ((1/ny) \otimes y)))$] *nminus-def* **by** *auto*
**hence** *p within e of* $((1/ny) \otimes y)$ **by** *auto*
**hence** $((1/ny) \otimes y)$ *within e of p*
**using** *lemSep2Symmetry*[*of* $((1/ny) \otimes y)$] **by** *auto*
**}**
**hence** *case1*: $(\alpha > 0) \longrightarrow (((1/ny) \otimes y)$ *within e of p*) **by** *blast*

**{ assume** *aneg*: $\alpha < 0$
**hence** $abs\ \alpha = -\alpha$ **by** *auto*
**hence** $abs\ (-(\alpha * ny) - 1) < eps$ **using** *star* **by** *auto*
**hence** *case2range*: $abs\ (\alpha * ny + 1) < eps$
**using** *lemAbsNegNeg*[*of* $\alpha * ny$ *1*] **by** *auto*

**define** *w2* **where** $w2 = ((\alpha \otimes y) \oplus ((1/ny) \otimes y))$
**define** *nw2* **where** $nw2 = norm\ w2$

**have** $(\alpha \otimes y) = ((1/ny) \otimes ((\alpha * ny) \otimes y))$
**using** *nynz lemScaleAssoc* **by** *auto*
**hence** $w2 = (((1/ny) \otimes ((\alpha * ny) \otimes y)) \oplus ((1/ny) \otimes y))$
**using** *w2-def* **by** *simp*
**also have** ... $= ((1/ny) \otimes (((\alpha * ny) \otimes y) \oplus y))$
**using** *lemScaleDistribSum*[*of* $1/ny\ (\alpha * ny) \otimes y\ y$] **by** *simp*
**also have** ... $= (((\alpha * ny) + 1) \otimes ((1/ny) \otimes y))$
**using** *lemScaleLeftDiffDistrib*[**where** $b=-1$] *lemScaleCommute* **by** *auto*
**finally have** *4*: $norm\ w2 = (abs\ ((\alpha * ny) + 1))$
**using** *lemNormOfScaled*[*of* $((\alpha * ny) + 1)\ (1/ny) \otimes y$] *nyunit* **by** *auto*

{
define *pp* where *pp*: $pp = (p \ominus (\alpha \otimes y))$
define *qq* where *qq*: $qq = ((\alpha \otimes y) \oplus ((1/ny) \otimes y))$
have *axTriangleInequality pp qq* using *AxTriangleInequality* by *simp*
hence $norm\ (pp \oplus qq) \leq norm\ pp + norm\ qq$ by *auto*
hence $norm\ ((p \oplus ((1/ny) \otimes y))) \leq norm\ pp + norm\ qq$
using *lemDiffSumCancelMiddle pp qq* by *force*
hence $norm\ ((p \oplus ((1/ny) \otimes y))) \leq norm\ (p \ominus r) + norm\ w2$
using *alpha w2-def pp qq* by *auto*
}
hence *5*: $norm\ ((p \oplus ((1/ny) \otimes y))) \leq npr + nw2$ using *nw2-def npr-def* by *auto*

define *nplus* where $nplus = norm\ ((p \oplus ((1/ny) \otimes y)))$

hence *almost2*: $nplus \leq npr + nw2$ using *5 nplus-def* by *auto*

have $abs\ ((ny * \alpha) - 1) \geq 0$ by *auto*
hence $nw2 = abs\ ((\alpha * ny) + 1)$ using *nw2-def 4 lemSqrt*[*of norm2 w2*] by *auto*
hence $nw2 < eps$ using *case2range le-less-trans* by *auto*
hence $nw2 < e/2$ using *eps-le-e2 le-less-trans* by *auto*

hence $nplus < (e/2 + e/2)$
using *almost2 npr-lt-e2 add-strict-mono le-less-trans* by *simp*

hence $nplus < e$ using *lemSumOfTwoHalves* by *simp*
hence $sqr\ nplus < sqr\ e$ using
*lemSqrMonoStrict*[*of nplus e*] *nplus-def*
*lemNormNonNegative*[*of* $((p \oplus ((1/ny) \otimes y)))$]
by *auto*

hence $norm2\ ((p \oplus ((1/ny) \otimes y))) < sqr\ e$
using *lemNormSqrIsNorm2*[*of* $((p \oplus ((1/ny) \otimes y)))$] *nplus-def* by *auto*

hence $sep2\ p\ ((-1/ny) \otimes y) < sqr\ e$ by *simp*
hence $(((-1/ny) \otimes y)\ within\ e\ of\ p)$
using *lemSep2Symmetry*[*of* $((-1/ny) \otimes y)$] by *auto*
}
hence *case2*: $(\alpha < 0) \longrightarrow (((-1/ny) \otimes y)\ within\ e\ of\ p)$ by *blast*

**hence** $(((1/ny)\otimes y)$ *within e of p*$) \vee (((-1/ny)\otimes y)$ *within e of p*$)$
**using** *cases case1* **by** *auto*
**}**
**hence** $((y$ *within d of origin*$) \wedge (y \neq$ *origin*$) \wedge (y \in wl) \wedge$ $($*norm* $y = ny))$
$\longrightarrow ((((1/ny)\otimes y)$ *within e of p*$) \vee (((-1/ny)\otimes y)$ *within e of p*$))$
**by** *blast*
**}**
**hence** $\exists\ \delta > 0\ .\forall\ y\ ny\ .((y$ *within* $\delta$ *of origin*$)$
$\wedge\ (y \neq$ *origin*$) \wedge (y \in wl) \wedge ($*norm* $y = ny))$
$\longrightarrow ((((1/ny)\otimes y)$ *within e of p*$) \vee (((-1/ny)\otimes y)$ *within e of p*$))$
**using** *dprops* **by** *blast*
**}**
**hence** $e > 0 \longrightarrow$
$(\exists\ \delta > 0\ .\forall\ y\ ny\ .((y$ *within* $\delta$ *of origin*$) \wedge (y \neq$ *origin*$) \wedge (y \in wl) \wedge ($*norm* $y = ny))$
$\longrightarrow ((((1/ny)\otimes y)$ *within e of p*$) \vee (((-1/ny)\otimes y)$ *within e of p*$)))$
**by** *blast*
**}**
**thus** *?thesis* **by** *blast*
**qed**

**lemma** *sublemma3Translation*:
**assumes** *onLine p l*
**and** *norm2* $(p\ominus x) = 1$
**and** *tangentLine l wl x*
**shows** $\forall\ \varepsilon > 0\ .\ \exists\ \delta > 0\ .\ \forall\ y\ nyx\ .$
$((y$ *within* $\delta$ *of* $x) \wedge (y \neq x) \wedge (y \in wl) \wedge ($*norm* $(y\ominus x) = nyx))$
$\longrightarrow$
$(((1/nyx)\otimes(y\ominus x))$ *within* $\varepsilon$ *of* $(p\ominus x))$
$\vee\ (((-1/nyx)\otimes(y\ominus x))$ *within* $\varepsilon$ *of* $(p\ominus x))$
**proof** $-$
**define** *pre*
**where** *pre*: *pre* $= (\lambda\ d\ y\ nyx\ .\ (y$ *within d of* $x) \wedge (y \neq x) \wedge (y \in wl) \wedge ($*norm* $(y\ominus x) = nyx))$
**define** *post*
**where** *post*: *post* $= (\lambda\ e\ y\ nyx\ .\ (((1/nyx)\otimes(y\ominus x))$ *within e of* $(p\ominus x))$
$\vee\ (((-1/nyx)\otimes(y\ominus x))$ *within e of* $(p\ominus x)))$

**define** $T$ **where** $T\ =$ *mkTranslation* $($*origin* $\ominus\ x)$

**hence** *transT*: *translation T* **using** *lemMkTrans* **by** *blast*
**have** *T*: $\forall p.\ T\ p = (p \oplus (origin \ominus x))$ **using** *T-def* **by** *simp*

**define** $p'$ **where** $p'$: $p' = T\ p$
**define** $l'$ **where** $l'$: $l' = (applyToSet\ (asFunc\ T)\ l)$
**define** $x'$ **where** $x'$: $x' = T\ x$
**define** $wl'$ **where** $wl'$: $wl' = (applyToSet\ (asFunc\ T)\ wl)$

**have** *1*: *onLine* $p'\ l'$
  **using** *assms(1)* $T\ p'\ l'$ *lemOnLineTranslation[of T l p]*
  **by** *blast*

**have** $x'0$: $x' = origin$ **using** $T\ x'$ *add-diff-eq* **by** *auto*
**hence** *sep2* $p'$ *origin* = *1*
  **using** *T assms(2)* $p'$ *lemTranslationPreservesSep2* **by** *simp*
**hence** *2*: *norm2* $p'$ = *1* **by** *auto*

**have** *tangentLine (applyToSet (asFunc T) l)*
    *(applyToSet (asFunc T) wl) (T x)*
  **using** *transT assms(3) lemTangentLineTranslation[of T x wl l]*
  **by** *auto*
**hence** *3*: *tangentLine* $l'\ wl'$ *origin* **using** $l'\ wl'\ x'\ x'0$ **by** *auto*

**hence** *conc*: $\forall\ \varepsilon > 0\ .\ \exists\ \delta > 0\ .\ \forall\ y'\ ny'\ .\ ($
  $((y'\ within\ \delta\ of\ origin) \wedge (y' \neq origin) \wedge (y' \in wl') \wedge (norm\ y'$
$= ny'))$
    $\longrightarrow$
    $(\ (((1/ny') \otimes y')\ within\ \varepsilon\ of\ p') \vee (((-1/ny') \otimes y')\ within\ \varepsilon\ of$
$p')))$
  **using** *1 2 3 sublemma3[of* $l'\ p'$*]*
  **by** *auto*

**{ fix** *e*
  **assume** *epos*: $e > 0$
  **then obtain** *d* **where** *d*: $(d > 0) \wedge (\forall\ y'\ ny'\ .\ ($
    $((y'\ within\ d\ of\ origin) \wedge (y' \neq origin) \wedge (y' \in wl') \wedge (norm\ y'$
$= ny'))$
    $\longrightarrow$
    $(\ (((1/ny') \otimes y')\ within\ e\ of\ p') \vee (((-1/ny') \otimes y')\ within\ e\ of$
$p'))))$
    **using** *conc* **by** *blast*

  **{ fix** *y nyx*
    **assume** *hyp*: *pre d y nyx*

    **define** $y'$ **where** $y'$: $y' = T\ y$
    **hence** *rtp1*: $y'$ *within d of origin*
      **using** *transT hyp* $x'\ x'0$ *lemBallTranslation pre* **by** *auto*

**have** *p′px*: $p' = (p \ominus x)$ **using** *p′ T* **by** *simp*
**have** *y′yx*: $y' = (y \ominus x)$ **using** *y′ T* **by** *simp*
**hence** *nyx*: *norm* $y' = nyx$ **using** *hyp pre* **by** *force*

**{ have** $(T\ x = x') \ \wedge\ (T\ y = y') \ \wedge\ (injective\ (asFunc\ T))$
**using** *x′ y′ lemTranslationInjective*[*of T*] *transT* **by** *blast*
**moreover have** $x \neq y$ **using** *hyp pre* **by** *auto*
**ultimately have** $y' \neq x'$**by** *auto*
**}**
**hence** *rtp2*: $y' \neq origin$ **using** *x′0* **by** *simp*

**have** *rtp3*: $y' \in wl'$ **using** *hyp pre y′ wl′* **by** *force*

**hence** $(y'\ within\ d\ of\ origin) \wedge (y' \neq origin) \wedge (y' \in wl') \wedge (norm\ y' = nyx)$
**using** *rtp1 rtp2 rtp3 nyx* **by** *blast*

**hence** $(((1/nyx) \otimes y')\ within\ e\ of\ p') \vee (((-1/nyx) \otimes y')\ within\ e\ of\ p')$
**using** *d* **by** *auto*
**hence** *post e y nyx* **using** *post y′yx p′px* **by** *auto*
**}**
**hence** $\forall\ y\ nyx\ .\ pre\ d\ y\ nyx \longrightarrow post\ e\ y\ nyx$ **by** *auto*
**hence** $\exists\delta{>}0.\ \forall\ y\ nyx\ .\ pre\ \delta\ y\ nyx \longrightarrow post\ e\ y\ nyx$ **using** *d* **by** *auto*
**}**
**hence** $\forall\varepsilon{>}0\ .\ \exists\delta{>}0.\ \forall\ y\ nyx\ .\ pre\ \delta\ y\ nyx \longrightarrow post\ \varepsilon\ y\ nyx$ **by** *auto*
**thus** *?thesis* **using** *post pre* **by** *blast*
**qed**

**end**

**end**

# 16 Vectors

In this theory we define dot-products, and explain what we mean by timelike, lightlike (null), causal and spacelike vectors.

**theory** *Vectors*
**imports** *Norms*
**begin**

**class** *Vectors = Norms*
**begin**

**fun** *dot* :: *'a Point* ⇒ *'a Point* ⇒ *'a* (- ⊙ -)
  **where** *dot u v* = (*tval u*)∗(*tval v*) + (*xval u*)∗(*xval v*) +
                    (*yval u*)∗(*yval v*) + (*zval u*)∗(*zval v*)

**fun** *sdot* :: *'a Space* ⇒ *'a Space* ⇒ *'a* (- ⊙*s* -)
  **where** *sdot u v* = (*svalx u*)∗(*svalx v*) + (*svaly u*)∗(*svaly v*) + (*svalz u*)∗(*svalz v*)

**fun** *mdot* :: *'a Point* ⇒ *'a Point* ⇒ *'a* (- ⊙*m* - )
 **where** *mdot u v* = (*tval u*)∗(*tval v*) − ((*sComponent u*) ⊙*s* (*sComponent v*))

**abbreviation** *timelike* :: *'a Point* ⇒ *bool*
  **where** *timelike p* ≡ *mNorm2 p* > *0*

**abbreviation** *lightlike* :: *'a Point* ⇒ *bool*
  **where** *lightlike p* ≡ (*p* ≠ *origin* ∧ *mNorm2 p* = *0*)

**abbreviation** *spacelike* :: *'a Point* ⇒ *bool*
  **where** *spacelike p* ≡ *mNorm2 p* < *0*

**abbreviation** *causal* :: *'a Point* ⇒ *bool*
  **where** *causal p* ≡ *timelike p* ∨ *lightlike p*

**abbreviation** *orthog* :: *'a Point* ⇒ *'a Point* ⇒ *bool*
  **where** *orthog p q* ≡ (*p* ⊙ *q*) = *0*

**abbreviation** *orthogs* :: *'a Space* ⇒ *'a Space* ⇒ *bool*
  **where** *orthogs p q* ≡ (*p* ⊙*s q*) = *0*

**abbreviation** *orthogm* :: *'a Point* ⇒ *'a Point* ⇒ *bool*
  **where** *orthogm p q* ≡ (*p* ⊙*m q*) = *0*

**lemma** *lemDotDecomposition*:
 **shows** (*u* ⊙ *v*) = (*tval u* ∗ *tval v*) + ((*sComponent u*) ⊙*s* (*sComponent v*))
  **by** (*simp add*: *add-commute local.add.left-commute*)

**lemma** *lemDotCommute*: *dot u v* = *dot v u*
  **by** (*simp add*: *mult-commute*)

**lemma** *lemDotScaleLeft*: *dot* (*a*⊗*u*) *v* = *a* ∗ (*dot u v*)

**using** *mult-assoc distrib-left* **by** *force*

**lemma** *lemDotScaleRight*: *dot u* $(a \otimes v) = a * (dot\ u\ v)$
**using** *mult-assoc mult-commute distrib-left* **by** *auto*

**lemma** *lemDotSumLeft*: *dot* $(u \oplus v)\ w = (dot\ u\ w) + (dot\ v\ w)$
**using** *distrib-right add-assoc add-commute* **by** *force*

**lemma** *lemDotSumRight*: *dot u* $(v \oplus w) = (dot\ u\ v) + (dot\ u\ w)$
**using** *distrib-left add-assoc add-commute* **by** *auto*

**lemma** *lemDotDiffLeft*: *dot* $(u \ominus v)\ w = (dot\ u\ w) - (dot\ v\ w)$
**by** (*simp add*: *field-simps*)

**lemma** *lemDotDiffRight*: *dot u* $(v \ominus w) = (dot\ u\ v) - (dot\ u\ w)$
**by** (*simp add*: *field-simps*)

**lemma** *lemNorm2OfSum*: *norm2* $(u \oplus v) =$ *norm2* $u + 2*(u \odot v)$ $+$ *norm2* $v$
**proof** −
**have** *norm2* $(u \oplus v) = ((u \oplus v) \odot (u \oplus v))$ **by** *auto*
**also have** $\ldots = (u \odot (u \oplus v)) + (v \odot (u \oplus v))$
**using** *lemDotSumLeft*[*of u v* $(u \oplus v)$] **by** *auto*
**also have** $\ldots = (u \odot u) + ((u \odot v) + (v \odot u)) + (v \odot v)$
**using** *lemDotSumRight*[*of u u v*] *lemDotSumRight*[*of v u v*]
*add-assoc* **by** *auto*
**finally show** *?thesis* **using** *mult-2 lemDotCommute*[*of u v*]
**by** *auto*
**qed**

**lemma** *lemSDotCommute*: *sdot u v = sdot v u*
**by** (*simp add*: *mult-commute*)

**lemma** *lemSDotScaleLeft*: *sdot* $(a \otimes s\ u)\ v = a * (sdot\ u\ v)$
**using** *mult-assoc distrib-left* **by** *force*

**lemma** *lemSDotScaleRight*: *sdot u* $(a \otimes s\ v) = a * (sdot\ u\ v)$
**using** *mult-assoc mult-commute distrib-left* **by** *auto*

**lemma** *lemSDotSumLeft*: *sdot* $(u \oplus s\ v)\ w = (sdot\ u\ w) + (sdot\ v\ w)$
**using** *distrib-right add-assoc add-commute* **by** *force*

**lemma** *lemSDotSumRight*: *sdot u* $(v \oplus s\ w) = (sdot\ u\ v) + (sdot\ u\ w)$
**using** *distrib-left add-assoc add-commute* **by** *auto*

**lemma** *lemSDotDiffLeft*: *sdot* $(u \ominus s\ v)\ w = (sdot\ u\ w) - (sdot\ v\ w)$
**by** (*simp add*: *field-simps*)

**lemma** *lemSDotDiffRight*: *sdot u* ( *v*⊖*s w*) = (*sdot u v*) − (*sdot u w*)
**by** (*simp add*: *field-simps*)

**lemma** *lemMDotDiffLeft*: *mdot* (*u*⊖*v*) *w* = (*mdot u w*) − (*mdot v w*)
**by** (*simp add*: *field-simps*)

**lemma** *lemMDotSumLeft*: *mdot* (*u* ⊕ *v*) *w* = (*mdot u w*) + (*mdot v w*)
**proof** −
**have** *mdot* (*u*⊕*v*) *w* = (*tval* (*u*⊕*v*))∗(*tval w*) − ((*sComponent* (*u*⊕*v*))⊙*s*(*sComponent w*))
**by** *auto*
**also have** . . . = (*tval u*∗*tval w*) + (*tval v*∗*tval w*)
− ( ((*sComponent u*)⊙*s*(*sComponent w*)) + ((*sComponent v*)⊙*s*(*sComponent w*)))
**using** *distrib lemSDotSumLeft*[*of* (*sComponent u*) (*sComponent v*) (*sComponent w*)]
**by** *auto*
**also have** . . . = ((*tval u*∗*tval w*) − ((*sComponent u*)⊙*s*(*sComponent w*)))
+ ((*tval v*∗*tval w*) − ((*sComponent v*)⊙*s*(*sComponent w*)))
**using** *add-diff-eq add-commute diff-diff-add* **by** *auto*
**finally show** *?thesis* **by** *simp*
**qed**

**lemma** *lemMDotScaleLeft*: *mdot* (*a* ⊗ *u*) *v* = *a* ∗ (*mdot u v*)
**proof** −
**have** *mdot* (*a* ⊗ *u*) *v* = *a*∗(*tval u*∗*tval v*) − *a*∗((*sComponent u*)⊙*s*(*sComponent v*))
**using** *lemSDotScaleLeft*[*of a sComponent u sComponent v*]
**by** (*simp add*: *mult-assoc*)
**thus** *?thesis* **by** (*simp add*: *local.right-diff-distrib′*)
**qed**

**lemma** *lemMDotScaleRight*: *mdot u* (*a* ⊗ *v*) = *a* ∗ (*mdot u v*)
**proof** −
**have** *mdot u* (*a* ⊗ *v*) = *a*∗(*tval u*∗*tval v*) − *a*∗((*sComponent u*)⊙*s*(*sComponent v*))
**using** *lemSDotScaleRight*[*of sComponent u a sComponent v*]
**by** (*simp add*: *local.mult.left-commute*)
**thus** *?thesis* **by** (*simp add*: *local.right-diff-distrib′*)
**qed**

**lemma** *lemSNorm2OfSum*: *sNorm2* $(u \oplus s\ v) = sNorm2\ u + 2*(u \odot s\ v) + sNorm2\ v$
**proof** −
  **have** *sNorm2* $(u \oplus s\ v) = ((u \oplus s\ v) \odot s\ (u \oplus s\ v))$ **by** *auto*
  **also have** $\ldots = (u \odot s\ (u \oplus s\ v)) + (v \odot s\ (u \oplus s\ v))$
    **using** *lemSDotSumLeft*[*of u v* $(u \oplus s\ v)$] **by** *auto*
  **also have** $\ldots = (u \odot s\ u) + ((u \odot s\ v) + (v \odot s\ u)) + (v \odot s\ v)$
    **using** *lemSDotSumRight*[*of u u v*] *lemSDotSumRight*[*of v u v*]
        *add-assoc* **by** *auto*
  **finally show** *?thesis* **using** *mult-2 lemSDotCommute*[*of u v*]
    **by** *auto*
**qed**

**lemma** *lemSNormNonNeg*:
  **shows** *sNorm v* $\geq 0$
**proof** −
  **have** *hasUniqueRoot* (*sNorm2 v*) **using** *AxEField lemSqrt* **by** *auto*
  **thus** *?thesis* **using** *the1-equality*[*of isNonNegRoot* (*sNorm2 v*)] **by** *blast*
**qed**

**lemma** *lemMNorm2OfSum*: *mNorm2* $(u \oplus v) = mNorm2\ u + 2*(u \odot m\ v) + mNorm2\ v$
**proof** −
  **define** *su* **where** *su*: *su* = *sComponent u*
  **define** *sv* **where** *sv*: *sv* = *sComponent v*

  **have** *mNorm2* $(u \oplus v) = ((u \oplus v) \odot m\ (u \oplus v))$ **by** *auto*
  **also have** $\ldots = (sqr\ (tval\ u) + 2*(tval\ u)*(tval\ v) + sqr\ (tval\ v)) - sNorm2\ (su \oplus s\ sv)$
    **using** *lemSqrSum su sv* **by** *auto*
  **also have** $\ldots = (sqr\ (tval\ u) + 2*(tval\ u)*(tval\ v) + sqr\ (tval\ v))$
        $- (\ sNorm2\ su + 2*(su \odot s\ sv) + sNorm2\ sv)$
    **using** *lemSNorm2OfSum* **by** *auto*
  **also have** $\ldots = (\ sqr\ (tval\ u) - sNorm2\ su\ )$
        $+ (\ 2*(tval\ u)*(tval\ v) - 2*(su \odot s\ sv)\ )$
        $+ (\ sqr\ (tval\ v) - sNorm2\ sv\ )$
    **using** *add-commute add-assoc add-diff-eq diff-add-eq diff-diff-add* **by** *simp*
  **finally show** *?thesis* **using** *su sv right-diff-distrib′ mult-assoc* **by** *auto*
**qed**

**lemma** *lemMNorm2OfDiff*: *mNorm2* $(u \ominus v)$ = *mNorm2* *u* − *2*∗(*u* ⊙*m* *v*) + *mNorm2* *v*
**proof** −
  **define** *vm* **where** *vm*: *vm* = ((−*1*)⊗*v*)
  **hence** *mNorm2* $(u \ominus v)$ = *mNorm2* $(u \oplus vm)$ **by** *auto*
  **hence** *mNorm2* $(u \ominus v)$ = *mNorm2* *u* + *2*∗(*u* ⊙*m* *vm*) + *mNorm2* *vm*
    **using** *lemMNorm2OfSum* **by** *auto*
  **moreover have** (*u* ⊙*m* *vm*) = −(*u* ⊙*m* *v*)
    **using** *lemMDotScaleRight*[*of* *u* (−*1*) *v*] *vm* **by** *auto*
  **moreover have** *mNorm2* *vm* = *mNorm2* *v* **using** *vm* *lemMNorm2OfScaled* **by** *auto*
  **ultimately show** *?thesis*
    **by** (*metis* *local.diff-conv-add-uminus* *local.mult-minus-right*)
**qed**

**lemma** *lemMNorm2Decomposition*: *mNorm2* *p* = (*p* ⊙*m* *p*)
  **by** *auto*

**lemma** *lemMDecomposition*:
  **assumes** (*u* ⊙*m* *v*) ≠ *0*
**and** *mNorm2* *v* ≠ *0*
**and** *a* = (*u* ⊙*m* *v*)/(*mNorm2* *v*)
**and** *up* = (*a* ⊗ *v*)
**and** *uo* = $(u \ominus up)$
  **shows** *u* = $(up \oplus uo)$ ∧ *parallel* *up* *v* ∧ *orthogm* *uo* *v* ∧ (*up* ⊙*m* *v*) = (*u* ⊙*m* *v*)
**proof** −
  **have** *anz*: *a* ≠ *0* **using** *assms* **by** *auto*

  **have** *psum*: *u* = $(up \oplus uo)$ **using** *assms* *add-diff-eq* **by** *auto*
  **moreover have** *parallel* *up* *v* **using** *assms*(*4*) *anz* **by** *auto*
  **moreover have** *ppdot*: (*up* ⊙*m* *v*) = (*u* ⊙*m* *v*)
  **proof** −
    **have** (*up* ⊙*m* *v*) = *a*∗(*v* ⊙*m* *v*) **using** *assms* *lemMDotScaleLeft*[*of* *a* *v* *v*] **by** *auto*
    **thus** *?thesis* **using** *assms* **by** *auto*
  **qed**
  **moreover have** *orthogm* *uo* *v*
  **proof** −
    **have** (*uo* ⊙*m* *v*) = (*u* ⊙*m* *v*) − (*up* ⊙*m* *v*) **using** *lemMDotSumLeft* *psum* **by** *force*
    **thus** *?thesis* **using** *ppdot* **by** *auto*
  **qed**
  **ultimately show** *?thesis* **by** *blast*
**qed**

**end**

**end**

# 17 CauchySchwarz

This theory defines and proves the Cauchy-Schwarz inequality for both spatial and spacetime vectors.

```
theory CauchySchwarz
  imports Vectors
begin
```

We essentially prove the same result twice, once for 3-dimensional spatial points, and once for 4-dimensional spacetime points. While this is clearly inefficient, it keeps things straightforward for non-Isabelle experts.

```
class CauchySchwarz = Vectors
begin


lemma lemCauchySchwarz4:
  shows abs (dot u v) ≤ (norm u)∗(norm v)
proof −
  have vorigin: v = origin ⟶ abs (dot u v) ≤ (norm u)∗(norm v)
  proof −
    { assume v = origin
      hence abs (dot u v) = 0 by simp
     also have . . . ≤ (norm u)∗(norm v) using lemNormNonNegative
by simp
      finally have abs (dot u v) ≤ (norm u)∗(norm v) by auto
    }
    thus ?thesis by blast
  qed

  define a where a = dot v v
  define b where b = 2 ∗ dot u v
  define c where c = dot u u

  { fix x :: 'a
    define w where w = (u ⊕ (x ⊗ v))
    have ww: (dot w w) ≥ 0 by simp

    define xv where xv: xv = (x ⊗ v)
    define middle2 where middle2 = dot u xv + dot xv u
```

**have** *dot xv u* = *dot u xv* **using** *lemDotCommute* **by** *blast*
**hence** *middle2* = *dot u xv* + *dot u xv* **using** *middle2-def* **by** *simp*
**also have** ... = *2* ∗ *dot u xv* **using** *mult-2* **by** *simp*
**finally have** *bterm*: *middle2* = *b* ∗ *x*
**using** *lemDotScaleRight mult-assoc mult-commute b-def xv* **by** *auto*

**have** *vxv*: (*dot v xv*) = (*x* ∗ *dot v v*)**using** *xv lemDotScaleRight* **by** *blast*
**have** *dot xv xv* = *x* ∗ (*dot v xv*) **using** *lemDotScaleLeft xv* **by** *blast*
**also have** ... = (*sqr x*)∗(*dot v v*) **using** *vxv mult-assoc* **by** *simp*
**finally have** *aterm*: *dot xv xv* = *a*∗(*sqr x*) **using** *mult-commute a-def* **by** *simp*

**have** *uw*: *dot u w* = *dot u u* + *dot u xv* **using** *lemDotSumRight w-def xv* **by** *blast*
**have** *vw*: *dot xv w* = *dot xv u* + *dot xv xv* **using** *lemDotSumRight w-def xv* **by** *blast*
**have** *dot w w* = *dot u w* + *dot xv w* **using** *lemDotSumLeft w-def xv* **by** *blast*
**also have** ... = (*dot u u* + *dot u xv*) + (*dot xv u* + *dot xv xv*)
**using** *uw vw* **by** *simp*
**also have** ... = (*dot u u*) + (*dot u xv* + *dot xv u*) + *dot xv xv*
**using** *add-assoc* **by** *force*
**also have** ... = (*dot u u*) + *middle2* + *dot xv xv*
**using** *middle2-def* **by** *simp*
**also have** ... = *c* + *b*∗*x* + *a*∗(*sqr x*) **using** *c-def bterm aterm* **by** *force*
**finally have** *dot w w* = *a*∗(*sqr x*) + *b*∗*x* + *c* **using** *add-commute add-assoc* **by** *auto*

**hence** *a*∗*sqr*(*x*) + *b*∗*x* + *c* ≥ *0* **using** *ww* **by** *simp*
**}**
**hence** *quadratic*: ∀ *x*. *a*∗*sqr*(*x*) + *b*∗*x* + *c* ≥ *0* **by** *auto*

**{ assume** *vnot0*: *v* ≠ *origin*
**hence** *a* > *0* **using** *a-def lemNullImpliesOrigin*[*of v*]
**by** (*metis local.AxEField local.not-less local.not-less-iff-gr-or-eq local.not-sum-squares-lt-zero dot.simps*)
**hence** (*sqr b*) ≤ *4*∗*a*∗*c* **using** *lemQuadraticGEZero quadratic* **by** *auto*
**hence** (*sqr b*) ≤ *4*∗(*dot v v*)∗(*dot u u*) **using** *a-def c-def* **by** *auto*
**hence** *sqrle*: (*sqr* (*abs b*)) ≤ *4*∗(*dot v v*)∗(*dot u u*) **by** *auto*

**define** *nv* **where** *nv*: *nv* = *norm v*
**define** *nu* **where** *nu*: *nu* = *norm u*

**have** *nvpos*: $nv \geq 0$ **using** *nv lemNormNonNegative* **by** *auto*
**have** *nupos*: $nu \geq 0$ **using** *nu lemNormNonNegative* **by** *auto*
**hence** *nvnu*: $2*nv*nu \geq 0$ **using** *nvpos* **by** *auto*

**have** *n2v*: *norm2 v* = *sqr nv* **using** *AxEField nv nvpos lemNorm-SqrIsNorm2* **by** *presburger*
**have** *n2u*: *norm2 u* = *sqr nu* **using** *AxEField nu nupos lemNorm-SqrIsNorm2* **by** *presburger*

**have** $4*(dot\ v\ v)*(dot\ u\ u) = 4*(norm2\ v)*(norm2\ u)$ **by** *auto*
**also have** ... = $(sqr\ 2)*(sqr\ nv)*(sqr\ nu)$ **using** *n2u n2v* **by** *auto*
**also have** ... = $(sqr\ (2*\ nv))*(sqr\ nu)$ **using** *lemSqrMult*[*of 2 nv*] **by** *auto*
**also have** ... = $sqr\ (2*nv*nu)$ **using** *lemSqrMult*[*of 2*nv nu*] **by** *auto*

**finally have** $(sqr\ (abs\ b)) \leq sqr\ (2*nv*nu)$ **using** *sqrle* **by** *auto*
**hence** *bnvnu*: $abs\ b \leq 2*nv*nu$
**using** *nu nv nvnu lemSqrOrdered*[*of 2*nv*nu*]
**by** *auto*

**have** *pos2*: $0 < 2$ **by** *simp*
**have** $b = 2*dot\ u\ v$ **using** *b-def* **by** *auto*
**hence** $abs\ b = 2*abs(dot\ u\ v)$ **using** *abs-mult* **by** *auto*
**hence** $2*abs(dot\ u\ v) \leq 2*(nv*nu)$ **using** *bnvnu mult-assoc* **by** *auto*
**hence** $2*abs(dot\ u\ v) \leq 2*(nu*nv)$ **using** *mult-commute* **by** *simp*
**hence** $abs(dot\ u\ v) \leq (nu*nv)$ **using** *mult-le-cancel-left*[*of 2*] *pos2* **by** *blast*
**hence** *?thesis* **using** *nu nv* **by** *auto*
**}**
**hence** $(v \neq origin) \longrightarrow$ *?thesis* **by** *auto*

**thus** *?thesis* **using** *vorigin* **by** *auto*
**qed**

**lemma** *lemCauchySchwarzSqr4*:
**shows** $sqr(dot\ u\ v) \leq (norm2\ u)*(norm2\ v)$
**proof** −
**have** *1*: $abs(dot\ u\ v) \geq 0$ **by** *simp*
**have** $sqr(dot\ u\ v) = sqr(abs(dot\ u\ v))$ **by** *simp*
**also have** $\ldots \leq sqr((norm\ u)*(norm\ v))$ **using** *1 lemCauchySchwarz4 lemSqrMono* **by** *blast*
**also have** $\ldots = sqr(norm\ u) * sqr(norm\ v)$ **using** *lemSqrMult* **by** *auto*
**also have** $\ldots = norm2\ u * norm2\ v$
**using** *lemSquareOfSqrt lemSqrt AxEField lemNormSqrIsNorm2*
**by** *force*

**finally show** *?thesis* **by** *simp*
**qed**

**lemma** *lemCauchySchwarz*:
**shows** *abs* (*sdot u v*) $\leq$ (*sNorm u*)$*$(*sNorm v*)
**proof** $-$
**have** *vorigin*: $v$ = *sOrigin* $\longrightarrow$ *abs* (*sdot u v*) $\leq$ (*sNorm u*)$*$(*sNorm v*)
**proof** $-$
**{ assume** $v$ = *sOrigin*
**hence** *abs* (*sdot u v*) = *0* **by** *simp*
**also have** ... $\leq$ (*sNorm u*)$*$(*sNorm v*) **using** *lemSNormNonNeg* **by** *simp*
**finally have** *abs* (*sdot u v*) $\leq$ (*sNorm u*)$*$(*sNorm v*) **by** *auto*
**}**
**thus** *?thesis* **by** *blast*
**qed**

**define** *a* **where** *a* = *sdot v v*
**define** *b* **where** *b* = *2* $*$ *sdot u v*
**define** *c* **where** *c* = *sdot u u*

**{ fix** *x* :: *'a*
**define** *w* **where** *w* = (*u* $\oplus$*s* (*x* $\otimes$*s v*))
**have** *ww*: (*sdot w w*) $\geq$ *0* **by** *simp*

**define** *xv* **where** *xv*: *xv* = (*x* $\otimes$*s v*)
**define** *middle2* **where** *middle2* = *sdot u xv* + *sdot xv u*

**have** *sdot xv u* = *sdot u xv* **using** *lemSDotCommute* **by** *blast*
**hence** *middle2* = *sdot u xv* + *sdot u xv* **using** *middle2-def* **by** *simp*
**also have** ... = *2* $*$ *sdot u xv* **using** *mult-2* **by** *simp*
**finally have** *bterm*: *middle2* = *b* $*$ *x*
**using** *lemSDotScaleRight mult-assoc mult-commute b-def xv* **by** *auto*

**have** *vxv*: (*sdot v xv*) = (*x* $*$ *sdot v v*)**using** *xv lemSDotScaleRight* **by** *blast*
**have** *sdot xv xv* = *x* $*$ (*sdot v xv*) **using** *lemSDotScaleLeft xv* **by** *blast*
**also have** ... = (*sqr x*)$*$(*sdot v v*) **using** *vxv mult-assoc* **by** *simp*
**finally have** *aterm*: *sdot xv xv* = *a*$*$(*sqr x*) **using** *mult-commute a-def* **by** *simp*

**have** *uw*: *sdot u w* = *sdot u u* + *sdot u xv* **using** *lemSDotSumRight w-def xv* **by** *blast*

**have** *vw*: *sdot xv w* = *sdot xv u* + *sdot xv xv* **using** *lemSDotSumRight w-def xv* **by** *blast*

**have** *sdot w w* = *sdot u w* + *sdot xv w* **using** *lemSDotSumLeft w-def xv* **by** *blast*

**also have** ... = (*sdot u u* + *sdot u xv*) + (*sdot xv u* + *sdot xv xv*)
**using** *uw vw* **by** *simp*

**also have** ... = (*sdot u u*) + (*sdot u xv* + *sdot xv u*) + *sdot xv xv*
**using** *add-assoc* **by** *force*

**also have** ... = (*sdot u u*) + *middle2* + *sdot xv xv*
**using** *middle2-def* **by** *simp*

**also have** ... = *c* + *b***x* + *a**(*sqr x*) **using** *c-def bterm aterm* **by** *force*

**finally have** *sdot w w* = *a**(*sqr x*) + *b***x* + *c* **using** *add-commute add-assoc* **by** *auto*

**hence** *a***sqr*(*x*) + *b***x* + *c* ≥ *0* **using** *ww* **by** *simp*

**}**

**hence** *quadratic*: ∀ *x*. *a***sqr*(*x*) + *b***x* + *c* ≥ *0* **by** *auto*

**{ assume** *vnot0*: *v* ≠ *sOrigin*

**hence** *a* > *0* **using** *a-def lemSpatialNullImpliesSpatialOrigin*[*of v*]
**by** (*metis local.AxEField local.not-less local.not-less-iff-gr-or-eq local.not-sum-squares-lt-zero sdot.simps*)

**hence** (*sqr b*) ≤ *4***a***c* **using** *lemQuadraticGEZero quadratic* **by** *auto*

**hence** (*sqr b*) ≤ *4**(*sdot v v*)*(*sdot u u*) **using** *a-def c-def* **by** *auto*

**hence** *sqrle*: (*sqr* (*abs b*)) ≤ *4**(*sdot v v*)*(*sdot u u*) **by** *auto*

**define** *nv* **where** *nv*: *nv* = *sNorm v*

**define** *nu* **where** *nu*: *nu* = *sNorm u*

**have** *nvpos*: *nv* ≥ *0* **using** *nv lemSNormNonNeg* **by** *auto*

**have** *nupos*: *nu* ≥ *0* **using** *nu lemSNormNonNeg* **by** *auto*

**hence** *nvnu*: *2***nv***nu* ≥ *0* **using** *nvpos* **by** *auto*

**have** *n2v*: *sNorm2 v* = *sqr nv* **using** *AxEField lemSquareOfSqrt nv nvpos* **by** *auto*

**have** *n2u*: *sNorm2 u* = *sqr nu* **using** *AxEField lemSquareOfSqrt nu nvpos* **by** *auto*

**have** *4**(*sdot v v*)*(*sdot u u*) = *4**(*sNorm2 v*)*(*sNorm2 u*) **by** *auto*

**also have** ... = (*sqr 2*)*(*sqr nv*)*(*sqr nu*) **using** *n2u n2v* **by** *auto*

**also have** ... = (*sqr* (*2** *nv*))*(*sqr nu*) **using** *lemSqrMult*[*of 2 nv*] **by** *auto*

**also have** ... = *sqr* (*2***nv***nu*) **using** *lemSqrMult*[*of 2***nv nu*] **by** *auto*

**finally have** $(sqr\ (abs\ b)) \leq sqr\ (2{*}nv{*}nu)$ **using** *sqrle* **by** *auto*
**hence** *bnvnu*: $abs\ b \leq 2{*}nv{*}nu$
**using** *nu nv nvnu lemSqrOrdered*[*of* $2{*}nv{*}nu$]
**by** *auto*

**have** *pos2*: $0 < 2$ **by** *simp*
**have** $b = 2{*}sdot\ u\ v$ **using** *b-def* **by** *auto*
**hence** $abs\ b = 2{*}abs(sdot\ u\ v)$ **using** *abs-mult* **by** *auto*
**hence** $2{*}abs(sdot\ u\ v) \leq 2{*}(nv{*}nu)$ **using** *bnvnu mult-assoc* **by** *auto*
**hence** $2{*}abs(sdot\ u\ v) \leq 2{*}(nu{*}nv)$ **using** *mult-commute* **by** *simp*
**hence** $abs(sdot\ u\ v) \leq (nu{*}nv)$ **using** *mult-le-cancel-left*[*of* 2] *pos2* **by** *blast*
**hence** *?thesis* **using** *nu nv* **by** *auto*
**}**
**hence** $(v \neq sOrigin) \longrightarrow$ *?thesis* **by** *auto*

**thus** *?thesis* **using** *vorigin* **by** *auto*
**qed**

**lemma** *lemCauchySchwarzSqr*:
**shows** $sqr(sdot\ u\ v) \leq (sNorm2\ u){*}(sNorm2\ v)$
**proof** −
**have** *1*: $abs(sdot\ u\ v) \geq 0$ **by** *simp*
**have** $sqr(sdot\ u\ v) = sqr(abs(sdot\ u\ v))$ **by** *simp*
**also have** $\ldots \leq sqr((sNorm\ u){*}(sNorm\ v))$ **using** *1 lemCauchySchwarz lemSqrMono* **by** *blast*
**also have** $\ldots = sqr(sNorm\ u) * sqr(sNorm\ v)$ **using** *lemSqrMult* **by** *auto*
**also have** $\ldots = sNorm2\ u * sNorm2\ v$ **using** *lemSquareOfSqrt lemSqrt AxEField* **by** *auto*
**finally show** *?thesis* **by** *simp*
**qed**

**lemma** *lemCauchySchwarzEquality*:
**assumes** $sqr\ (sdot\ u\ v) = (sNorm2\ u){*}(sNorm2\ v)$
**and** $u \neq sOrigin \land v \neq sOrigin$
**shows** $\exists\ a \neq 0\ .\ u = (a \otimes s\ v)$
**proof** −
**define** *a* **where** *a*: $a = (sdot\ u\ v)/(sNorm2\ v)$
**have** *uvnz*: $sNorm2\ u \neq 0 \land sNorm2\ v \neq 0$ **using** *assms lemSpatialNullImpliesSpatialOrigin* **by** *blast*
**hence** $sqr\ (sdot\ u\ v) \neq 0$ **using** *assms* **by** *auto*

**hence** *anz*: $a \neq 0$ **using** *assms uvnz a* **by** *auto*

**define** *upv* **where** *upv*: $upv = (a \otimes s\ v)$
**hence** *sdotupv*: $sdot\ upv\ v = sdot\ u\ v$
**proof** −
**have** $sdot\ upv\ v = a * sNorm2\ v$ **using** *upv lemSDotScaleLeft* **by** *auto*
**thus** *?thesis* **using** *a uvnz* **by** *auto*
**qed**
**have** *sn2upv*: $sNorm2\ upv = (sqr\ a)*sNorm2\ v$ **using** *upv lemSNorm2OfScaled* **by** *auto*

**define** *uov* **where** *uov*: $uov = (u \ominus s\ upv)$
**have** *usum*: $u = (upv \oplus s\ uov)$ **using** *uov add-diff-eq* **by** *auto*
**hence** *sdotuov*: $sdot\ uov\ v = 0$ **using** *lemSDotSumLeft sdotupv* **by** *force*
**hence** *pdoto*: $sdot\ uov\ upv = 0$ **using** *upv lemSDotScaleRight local.mult-not-zero* **by** *metis*

**have** $sqr\ (sdot\ u\ v) = sqr\ (sdot\ (a \otimes s\ v)\ v)$ **using** *sdotupv upv* **by** *auto*
**also have** $\ldots = (sqr\ a) * sqr\ (sNorm2\ v)$
**using** *lemSDotScaleLeft*[*of a v v*] *lemSqrMult*[*of a*] **by** *auto*
**finally have** *lhs*: $sqr\ (sdot\ u\ v) = (sqr\ a) * sqr\ (sNorm2\ v)$ **by** *auto*

**have** $sNorm2\ u = sNorm2\ upv + 2*(upv \odot s\ uov) + sNorm2\ uov$ **using** *lemSNorm2OfSum usum* **by** *auto*
**also have** $\ldots = (sqr\ a)*sNorm2\ v + sNorm2\ uov$ **using** *sn2upv pdoto lemSDotCommute* **by** *auto*
**finally have** *rhs*: $(sNorm2\ u)*(sNorm2\ v) = (sqr\ a)*sqr(sNorm2\ v) + (sNorm2\ uov)*(sNorm2\ v)$
**using** *distrib-right*[*of* $(sqr\ a)*sNorm2\ v\ sNorm2\ uov\ sNorm2\ v$]
*mult-assoc* **by** *auto*

**hence** $(sqr\ a) * sqr\ (sNorm2\ v) = (sqr\ a)*sqr(sNorm2\ v) + (sNorm2\ uov)*(sNorm2\ v)$
**using** *lhs assms*(*1*) **by** *auto*
**hence** $(sNorm2\ uov)*(sNorm2\ v) = 0$ **using** *add-diff-eq* **by** *auto*
**hence** $uov = sOrigin$ **using** *uvnz lemSpatialNullImpliesSpatialOrigin* **by** *auto*

**hence** $a \neq 0 \wedge u = (a \otimes s\ v)$ **using** *anz usum upv* **by** *auto*
**thus** *?thesis* **by** *auto*
**qed**

**lemma** *lemCauchySchwarzEqualityInUnitSphere*:
**assumes** $(sNorm2\ u \leq 1) \wedge (sNorm2\ v \leq 1)$
**and** $sdot\ u\ v = 1$
**shows** $u = v$
**proof** −
**have** *uvnz*: $u \neq sOrigin \wedge v \neq sOrigin$ **using** *assms*(*2*) **by** *auto*
**{ assume** *ass*: $(sNorm2\ u < 1) \vee (sNorm2\ v < 1)$
**have** $(sNorm2\ u > 0) \wedge (sNorm2\ v > 0)$
**using** *uvnz lemSpatialNullImpliesSpatialOrigin add-less-zeroD less-linear not-square-less-zero*
**by** *blast*
**hence** $(sNorm2\ u)*(sNorm2\ v) < 1$
**by** (*metis ass assms*(*1*) *local.dual-order.not-eq-order-implies-strict local.leD*
*local.less-imp-le local.mult-le-one local.mult-less-cancel-left1*
*local.mult-less-cancel-right1*)
**hence** *False* **using** *lemCauchySchwarzSqr assms*(*2*)
**by** (*metis local.dual-order.strict-iff-not local.mult-cancel-right1*)
**}**
**hence** *norms1*: $sNorm2\ u = 1 \wedge sNorm2\ v = 1$ **using** *assms*(*1*) **by** *force*
**hence** $sqr\ (sdot\ u\ v) = (sNorm2\ u)*(sNorm2\ v)$ **using** *assms*(*2*) **by** *auto*
**hence** $\exists\ a \neq 0\ .\ u = (a \otimes s\ v)$ **using** *lemCauchySchwarzEquality uvnz* **by** *blast*
**then obtain** *a* **where** *a*: $a \neq 0 \wedge u = (a \otimes s\ v)$ **by** *auto*
**hence** $sdot\ u\ v = a * sNorm2\ v$ **using** *lemSDotScaleLeft* **by** *auto*
**hence** $a = 1$ **using** *assms*(*2*) *norms1* **by** *auto*
**thus** *?thesis* **using** *a* **by** *auto*
**qed**

**lemma** *lemCausalOrthogmToLightlikeImpliesParallel*:
**assumes** *causal p*
**and** *lightlike q*
**and** *orthogm p q*
**shows** *parallel p q*
**proof** −
**have** *tpnz*: $tval\ p \neq 0$
**proof** −
**have** $p \neq origin$ **using** *assms*(*1*) **by** *auto*
**have** *case1*: $lightlike\ p \longrightarrow ?thesis$
**by** (*metis local.diff-add-cancel local.lemNorm2Decomposition*
*local.lemNullImpliesOrigin local.lemZeroRoot*)
**have** *case2*: $timelike\ p \longrightarrow ?thesis$
**by** (*metis local.add-less-zeroD local.diff-gt-0-iff-gt*
*local.lemZeroRoot local.not-square-less-zero*)
**thus** *?thesis* **using** *assms*(*1*) *case1* **by** *blast*
**qed**

**have** *tqnz*: *tval q* $\neq$ *0* **using** *assms*(*2*)
**by** (*metis local.diff-add-cancel local.lemNorm2Decomposition local.lemNullImpliesOrigin local.lemZeroRoot*)

**define** *phat* **where** *phat*: *phat* = ((*1*/*tval p*)$\otimes$*p*)
**define** *qhat* **where** *qhat*: *qhat* = ((*1*/*tval q*)$\otimes$*q*)

**have** *phatcausal*: *causal phat*
**proof** −
**have** *n2*: *mNorm2 phat* = (*sqr* (*1*/*tval p*))$*$*mNorm2 p* **using** *phat lemMNorm2OfScaled* **by** *blast*
**have** *lightlike p* $\longrightarrow$ *lightlike phat* **using** *phat n2 tpnz* **by** *auto*
**moreover have** *timelike p* $\longrightarrow$ *timelike phat* **using** *phat n2 tpnz*
**by** (*simp add*: *local.lemSquaresPositive*)
**ultimately show** *?thesis* **using** *assms*(*1*) **by** *blast*
**qed**

**have** *qhatlightlike*: *lightlike qhat*
**proof** −
**have** *mNorm2 qhat* = (*sqr* (*1*/*tval q*))$*$*mNorm2 q* **using** *qhat lemMNorm2OfScaled* **by** *blast*
**thus** *?thesis* **using** *assms*(*2*) *tqnz qhat local.divide-eq-0-iff* **by** *force*

**qed**

**have** *hatsorthog*: *orthogm phat qhat*
**proof** −
**have** (*phat* $\odot$*m qhat*) = (*1*/*tval p*)$*$(*p* $\odot$*m qhat*)
**using** *phat lemMDotScaleLeft*[*of 1*/*tval p p qhat*] **by** *auto*
**thus** *?thesis*
**using** *qhat lemMDotScaleRight*[*of p 1*/*tval q q*] *tpnz tqnz assms*(*3*) **by** *auto*
**qed**

**define** *ps* **where** *ps*: *ps* = *sComponent phat*
**define** *qs* **where** *qs*: *qs* = *sComponent qhat*

**have** *p*: *phat* = *stPoint 1 ps* **using** *phat ps tpnz* **by** *auto*
**have** *q*: *qhat* = *stPoint 1 qs* **using** *qhat qs tqnz* **by** *auto*

**have** *sNorm2 ps* $\leq$ *1* **using** *p phatcausal* **by** *auto*
**moreover have** *sNorm2 qs* = *1* **using** *q qhatlightlike* **by** *auto*
**moreover have** *sdot ps qs* = *1* **using** *hatsorthog p q* **by** *auto*
**ultimately have** *ps* = *qs*
**using** *lemCauchySchwarzEqualityInUnitSphere* **by** *auto*

**hence** *phat* = *qhat* **using** *p q* **by** *auto*
**hence** ((*1*/*tval p*)$\otimes$*p*) = ((*1*/*tval q*)$\otimes$*q*) **using** *phat qhat* **by** *auto*

**hence** $p = (((tval\ p)/(tval\ q)) \otimes q)$
  **using** *tpnz tqnz*
      *lemScaleAssoc*[*of tval p 1/tval p p*]
      *lemScaleAssoc*[*of tval p 1/tval q q*]
  **by** *auto*
**thus** *?thesis* **using** *tpnz tqnz* **using** *local.divide-eq-0-iff*
  **by** *blast*
**qed**

**end**

**end**

# 18 Matrices

This theory defines $4 \times 4$ matrices.

**theory** *Matrices*
 **imports** *Vectors*
**begin**

**record** *'a Matrix* =
 *trow* :: *'a Point*
 *xrow* :: *'a Point*
 *yrow* :: *'a Point*
 *zrow* :: *'a Point*

**class** *Matrices* = *Vectors*
**begin**

**fun** *applyMatrix* :: *'a Matrix* ⇒ *'a Point* ⇒ *'a Point*
 **where** *applyMatrix m p* = (| *tval* = *dot* (*trow m*) *p*, *xval* = *dot* (*xrow m*) *p*,
      *yval* = *dot* (*yrow m*) *p*, *zval* = *dot* (*zrow m*) *p* |)

**fun** *tcol* :: *'a Matrix* ⇒ *'a Point*
 **where** *tcol m* = (| *tval* = *tval* (*trow m*), *xval* = *tval* (*xrow m*),
      *yval* = *tval* (*yrow m*), *zval* = *tval* (*zrow m*) |)

**fun** *xcol* :: *'a Matrix* ⇒ *'a Point*
 **where** *xcol m* = (| *tval* = *xval* (*trow m*), *xval* = *xval* (*xrow m*),
      *yval* = *xval* (*yrow m*), *zval* = *xval* (*zrow m*) |)

**fun** *ycol* :: *'a Matrix* ⇒ *'a Point*
 **where** *ycol m* = (| *tval* = *yval* (*trow m*), *xval* = *yval* (*xrow m*),
      *yval* = *yval* (*yrow m*), *zval* = *yval* (*zrow m*) |)

**fun** *zcol* :: *'a Matrix* ⇒ *'a Point*
**where** *zcol m* = ⦇ *tval* = *zval* (*trow m*), *xval* = *zval* (*xrow m*),
*yval* = *zval* (*yrow m*), *zval* = *zval* (*zrow m*) ⦈

**fun** *transpose* :: *'a Matrix* ⇒ *'a Matrix*
**where** *transpose m* = ⦇ *trow* = (*tcol m*), *xrow* = (*xcol m*),
*yrow* = (*ycol m*), *zrow* = (*zcol m*) ⦈

**fun** *mprod* :: *'a Matrix* ⇒ *'a Matrix* ⇒ *'a Matrix*
**where** *mprod m1 m2* =
*transpose* ⦇ *trow* = *applyMatrix m1* (*tcol m2*), *xrow* = *applyMatrix m1* (*xcol m2*),
*yrow* = *applyMatrix m1* (*ycol m2*), *zrow* = *applyMatrix m1* (*zcol m2*) ⦈

**end**

**end**

# 19 LinearMaps

This theory defines linear maps and establishes their main properties.

**theory** *LinearMaps*
**imports** *Functions CauchySchwarz Matrices*
**begin**

**class** *LinearMaps* = *Functions* + *CauchySchwarz* + *Matrices*
**begin**

**abbreviation** *linear* :: (*'a Point* ⇒ *'a Point*) ⇒ *bool* **where**
*linear L* ≡ (*L origin* = *origin*)
∧ (∀ *a p* . *L* (*a* ⊗ *p*) = (*a* ⊗ (*L p*)))
∧ (∀ *p q* . *L* (*p* ⊕ *q*) = ((*L p*) ⊕ (*L q*)))
∧ (∀ *p q* . *L* (*p* ⊖ *q*) = ((*L p*) ⊖ (*L q*)))

**lemma** *lemLinearProps*:
  **assumes** *linear L*
  **shows** $(L\ origin = origin) \wedge (L\ (a \otimes p) = (a \otimes (L\ p)))$
    $\wedge\ (L\ (p \oplus q) = ((L\ p) \oplus (L\ q)))$
    $\wedge\ (L\ (p \ominus q) = ((L\ p) \ominus (L\ q)))$
**using** *assms* **by** *simp*

**lemma** *lemMatrixApplicationIsLinear*: *linear* (*applyMatrix m*)
  **using** *lemDotScaleRight lemDotSumRight lemDotDiffRight*
  **by** *fastforce*

**lemma** *lemLinearIsMatrixApplication*:
  **assumes** *linear L*
  **shows** $\exists\ m\ .\ L = (applyMatrix\ m)$
**proof** −
  **define** *Lt* **where** $Lt = L\ tUnit$
  **define** *Lx* **where** $Lx = L\ xUnit$
  **define** *Ly* **where** $Ly = L\ yUnit$
  **define** *Lz* **where** $Lz = L\ zUnit$
  **define** *M* **where** $M = transpose\ (\!|\ trow = Lt,\ xrow = Lx,\ yrow = Ly,\ zrow = Lz\ |\!)$

  **have** *trowM*: $trow\ M = (\!|\ tval = (tval\ Lt),\ xval = (tval\ Lx),$
    $yval = (tval\ Ly),\ zval = (tval\ Lz)\ |\!)$
    **using** *M-def* **by** *auto*
  **have** *xrowM*: $xrow\ M = (\!|\ tval = (xval\ Lt),\ xval = (xval\ Lx),$
    $yval = (xval\ Ly),\ zval = (xval\ Lz)\ |\!)$
    **using** *M-def* **by** *auto*
  **have** *yrowM*: $yrow\ M = (\!|\ tval = (yval\ Lt),\ xval = (yval\ Lx),$
    $yval = (yval\ Ly),\ zval = (yval\ Lz)\ |\!)$
    **using** *M-def* **by** *auto*
  **have** *zrowM*: $zrow\ M = (\!|\ tval = (zval\ Lt),\ xval = (zval\ Lx),$
    $yval = (zval\ Ly),\ zval = (zval\ Lz)\ |\!)$
    **using** *M-def* **by** *auto*

  **{ fix** *u* :: *'a Point*
    **define** *tvu* **where** *tvu*: $tvu = ((tval\ u) \otimes tUnit)$
    **define** *xvu* **where** *xvu*: $xvu = ((xval\ u) \otimes xUnit)$
    **define** *yvu* **where** *yvu*: $yvu = ((yval\ u) \otimes yUnit)$
    **define** *zvu* **where** *zvu*: $zvu = ((zval\ u) \otimes zUnit)$

    **have** *u*: $u = (tvu \oplus (xvu \oplus (yvu \oplus zvu)))$
      **using** *tvu xvu yvu zvu lemPointDecomposition*[*of u*] **by** *simp*

**have** *Mu*: *applyMatrix M u* = (| *tval* = *dot* (*trow M*) *u*,
*xval* = *dot* (*xrow M*) *u*,
*yval* = *dot* (*yrow M*) *u*,
*zval* = *dot* (*zrow M*) *u* |) **by** *simp*

**have** *tvalMu*: *tval* (*applyMatrix M u*) =
(*tval Lt*)∗(*tval u*) + (*tval Lx*)∗(*xval u*) + (*tval Ly*)∗(*yval u*) + (*tval Lz*)∗(*zval u*)
**using** *Mu trowM* **by** *force*
**have** *xvalMu*: *xval* (*applyMatrix M u*) =
(*xval Lt*)∗(*tval u*) + (*xval Lx*)∗(*xval u*) + (*xval Ly*)∗(*yval u*) + (*xval Lz*)∗(*zval u*)
**using** *Mu xrowM* **by** *force*
**have** *yvalMu*: *yval* (*applyMatrix M u*) =
(*yval Lt*)∗(*tval u*) + (*yval Lx*)∗(*xval u*) + (*yval Ly*)∗(*yval u*) + (*yval Lz*)∗(*zval u*)
**using** *Mu yrowM* **by** *force*
**have** *zvalMu*: *zval* (*applyMatrix M u*) =
(*zval Lt*)∗(*tval u*) + (*zval Lx*)∗(*xval u*) + (*zval Ly*)∗(*yval u*) + (*zval Lz*)∗(*zval u*)
**using** *Mu zrowM* **by** *force*

**hence** *Lu*: *L u* = ((*L tvu*) ⊕ ((*L xvu*) ⊕ ((*L yvu*) ⊕ (*L zvu*))))
**using** *assms u*
*lemLinearProps*[*of L 0 tvu xvu* ⊕ (*yvu* ⊕ *zvu*)]
*lemLinearProps*[*of L 0 xvu yvu* ⊕ *zvu*]
**by** *auto*
**have** *Ltvu*: *L tvu* = ((*tval u*)⊗*Lt*)
**using** *tvu Lt-def assms lemLinearProps*[*of L tval u tUnit*] **by** *auto*
**have** *Lxvu*: *L xvu* = ((*xval u*)⊗*Lx*)
**using** *xvu Lx-def assms lemLinearProps*[*of L xval u xUnit*] **by** *auto*
**have** *Lyvu*: *L yvu* = ((*yval u*)⊗*Ly*)
**using** *yvu Ly-def assms lemLinearProps*[*of L yval u yUnit*] **by** *auto*
**have** *Lzvu*: *L zvu* = ((*zval u*)⊗*Lz*)
**using** *zvu Lz-def assms lemLinearProps*[*of L zval u zUnit*] **by** *auto*

**hence** *Lu'*: *L u* = (((*tval u*)⊗*Lt*) ⊕ (((*xval u*)⊗*Lx*)
⊕ (((*yval u*)⊗*Ly*) ⊕ ((*zval u*)⊗*Lz*))))
**using** *Lu Ltvu Lxvu Lyvu Lzvu* **by** *force*

**hence** *L u* = *applyMatrix M u*
**using** *Lu' add-assoc tvalMu xvalMu yvalMu zvalMu mult-commute* **by** *simp*
**}**

**hence** ∀ *u*. *L u* = *applyMatrix M u* **by** *auto*
**thus** *?thesis* **by** *force*
**qed**

**lemma** *lemLinearIffMatrix*: *linear L* ⟷ (∃ *M*. *L* = *applyMatrix M*)
**using** *lemMatrixApplicationIsLinear lemLinearIsMatrixApplication*
**by** *auto*

**lemma** *lemIdIsLinear*: *linear id*
**by** *simp*

**lemma** *lemLinearIsBounded*:
**assumes** *linear L*
**shows** *bounded L*
**proof** −
**obtain** *M* **where** *M*: *L* = *applyMatrix M* **using** *assms lemLinearIffMatrix* **by** *auto*
**define** *tr* **where** *tr* = *trow M*
**define** *xr* **where** *xr* = *xrow M*
**define** *yr* **where** *yr* = *yrow M*
**define** *zr* **where** *zr* = *zrow M*
**define** *bnd* **where** *bnd* = (*sqr*(*norm tr*)+*sqr*(*norm xr*)+*sqr*(*norm yr*)+*sqr*(*norm zr*))

**define** *n*
**where** *n*: *n* = (| *tval*=*norm tr*, *xval*=*norm xr*, *yval*=*norm yr*, *zval*=*norm zr* |)
**hence** *bnd* = *dot n n* **using** *bnd-def* **by** *auto*
**hence** *norm2n*: *bnd* = *norm2 n* **by** *simp*
**hence** *bndnonneg*: *bnd* ≥ *0* **by** *simp*

{ **assume** *bndpos*: *bnd* > *0*
{ **fix** *p* :: *'a Point*

**define** *q* **where** *q* = *applyMatrix M p*
**hence** *q* = (| *tval*=*dot tr p*, *xval*=*dot xr p*,*yval*=*dot yr p*, *zval*=*dot zr p* |)
**using** *tr-def xr-def yr-def zr-def* **by** *auto*
**hence** *1*: *dot q q* = *sqr* (*dot tr p*) + *sqr* (*dot xr p*)
+ *sqr* (*dot yr p*) + *sqr*(*dot zr p*)
**by** *auto*
**also have** ... ≤ *sqr* (*dot tr p*) + *sqr* (*dot xr p*) + *sqr* (*dot yr p*)
+ (*sqr*(*norm zr*)∗*sqr*(*norm p*))

**using** *lemCauchySchwarzSqr4*[*of zr p*] *lemNormSqrIsNorm2*
**by** *auto*
**also have** ... $\leq$ *sqr* (*dot tr p*) + *sqr* (*dot xr p*) + (*sqr*(*norm yr*)$*$*sqr*(*norm p*))
+ (*sqr*(*norm zr*)$*$*sqr*(*norm p*))
**using** *lemCauchySchwarzSqr4*[*of yr p*] *lemNormSqrIsNorm2*
**by** *auto*
**also have** ... $\leq$ *sqr* (*dot tr p*) + (*sqr*(*norm xr*)$*$*sqr*(*norm p*)) + (*sqr*(*norm yr*)$*$*sqr*(*norm p*))
+ (*sqr*(*norm zr*)$*$*sqr*(*norm p*))
**using** *lemCauchySchwarzSqr4*[*of xr p*] *lemNormSqrIsNorm2*
**by** *auto*
**also have** ... $\leq$ (*sqr*(*norm tr*)$*$*sqr*(*norm p*)) + (*sqr*(*norm xr*)$*$*sqr*(*norm p*)) + (*sqr*(*norm yr*)$*$*sqr*(*norm p*))
+ (*sqr*(*norm zr*)$*$*sqr*(*norm p*))
**using** *lemCauchySchwarzSqr4*[*of tr p*] *lemNormSqrIsNorm2*
**by** *auto*
**finally have** *dot q q* $\leq$ (*sqr*(*norm tr*)$*$*sqr*(*norm p*)) + (*sqr*(*norm xr*)$*$*sqr*(*norm p*)) + (*sqr*(*norm yr*)$*$*sqr*(*norm p*))
+ (*sqr*(*norm zr*)$*$*sqr*(*norm p*)) **by** *auto*
**hence** *dot q q* $\leq$ (*sqr*(*norm tr*)+*sqr*(*norm xr*)+*sqr*(*norm yr*)+*sqr*(*norm zr*))$*$*sqr*(*norm p*)
**using** *distrib-right* **by** *auto*
**hence** *norm2 q* $\leq$ *bnd* $*$ *sqr*(*norm p*) **using** *bnd-def* **by** *simp*
**hence** *norm2* (*applyMatrix M p*) $\leq$ *bnd* $*$ *norm2 p*
**using** *q-def lemNormSqrIsNorm2* **by** *simp*
**}**
**hence** $\forall$ *p*. *norm2* (*applyMatrix M p*) $\leq$ *bnd* $*$ *norm2 p* **by** *auto*
**hence** $\exists$ *bnd* $>$ *0* . $\forall$ *p*. *norm2* (*applyMatrix M p*) $\leq$ *bnd* $*$ *norm2 p*
**using** *bndpos* **by** *auto*
**}**
**hence** *case1*: (*bnd* $>$ *0*) $\longrightarrow$ (*bounded* (*applyMatrix M*)) **by** *simp*

**{ assume** *bnd0*: *bnd* = *0*
**hence** *n* = *origin* **using** *lemNullImpliesOrigin norm2n* **by** *auto*
**hence** (*norm tr* = *0*) $\wedge$ (*norm xr* = *0*) $\wedge$ (*norm yr* = *0*) $\wedge$ (*norm zr* = *0*)
**using** *n* **by** *simp*
**hence** *allzero*: (*tr* = *origin*)$\wedge$(*xr*=*origin*)$\wedge$(*yr*=*origin*)$\wedge$(*zr*=*origin*)
**using** *lemZeroNorm* **by** *auto*

**define** *one* **where** *one* = (*1*::*'a*)
**hence** *onepos*: *one* $>$ *0* **by** *simp*
**{ fix** *p* :: *'a Point*
**have** *applyMatrix M p* = *origin*
**using** *allzero tr-def xr-def yr-def zr-def* **by** *auto*
**hence** *norm2*(*applyMatrix M p*) = *0* **by** *auto*
**hence** *norm2*(*applyMatrix M p*) $\leq$ *one* $*$ (*norm2 p*) **using** *onepos*

```
by auto
    }
    hence ∀ p . norm2(applyMatrix M p) ≤ one ∗ (norm2 p) by auto
   hence ∃ one > 0 . ∀ p . norm2(applyMatrix M p) ≤ one ∗ (norm2
p)
      using onepos by auto
    hence bounded (applyMatrix M) by simp
  }
  hence case2: (bnd = 0) ⟶ (bounded (applyMatrix M)) by simp

  thus ?thesis using case1 case2 bndnonneg M by auto
qed



lemma lemLinearIsCts:
  assumes linear L
  shows   cts (asFunc L) x
proof −
  { fix x′
    assume x′: x′ = L x

   have bounded L using assms(1) lemLinearIsBounded[of L] by auto
    then obtain bnd where bnd: (bnd > 0) ∧ (∀ p. norm2(L p) ≤
bnd∗(norm2 p))
      by auto
    then obtain bb where bb: (bb > 0) ∧ (sqr bb) > bnd
      using bnd lemSquareExistsAbove[of bnd] by auto

    { fix p
      have p1: norm2 (L p) ≤ bnd∗(norm2 p) using bnd by simp
      have bnd∗(norm2 p) ≤ (sqr bb)∗(norm2 p) using bb mult-mono
by auto
      hence norm2 (L p) ≤ (sqr bb)∗(norm2 p) using p1 by simp
    }
    hence bbbnd: ∀ p . norm2 (L p) ≤ (sqr bb)∗(norm2 p) by auto

    { fix e
      assume epos: e > 0
      define d where d: d = e/bb
      hence dpos: d > 0 using epos bb by simp
      have (d = e/bb) ∧ (bb ≠ 0) using d bb by auto
      hence esqr: (sqr d)∗(sqr bb) = sqr e by simp


      { fix p′
        assume p′: p′ ∈ applyToSet (asFunc L) (ball x d)
        then obtain p where p: (p ∈ ball x d) ∧ (p′ = L p) by auto
         hence p-near-x: p within d of x using lemSep2Symmetry[of p
```

$x$] **by** *force*

**have** *norm2* ($L$ ($p \ominus x$)) $\leq$ (*sqr bb*) $*$ *norm2* ($p \ominus x$) **using** *bbbnd* **by** *blast*

**hence** *1*: *norm2* ($L$ ($p \ominus x$)) $\leq$ (*sqr bb*) $*$ (*sep2 p x*) **by** *auto*

**have** (*sqr bb*)$*$(*sep2 p x*) $<$ (*sqr bb*)$*$(*sqr d*)

**using** *lemMultPosLT bb p-near-x* **by** *auto*

**hence** *2*: *norm2* ($L$ ($p \ominus x$)) $<$ (*sqr bb*) $*$ (*sqr d*) **using** *1* **by** *simp*

**have** ($L$ ($p \ominus x$)) = (($L$ $p$) $\ominus$ ($L$ $x$)) **using** *assms*(*1*) **by** *auto*

**hence** *norm2* ($L$ ($p \ominus x$)) = *sep2* $p'$ $x'$ **using** $p$ $x'$ **by** *force*

**hence** *sep2* $p'$ $x'$ $<$ (*sqr bb*) $*$ (*sqr d*) **using** *2* **by** *simp*

**hence** *sep2* $p'$ $x'$ $<$ *sqr e* **using** $d$ *bb* **by** *auto*

**hence** $p' \in$ *ball* $x'$ $e$ **using** *lemSep2Symmetry* **by** *auto*

**}**

**hence** *applyToSet* (*asFunc L*) (*ball x d*) $\subseteq$ *ball* $x'$ $e$ **by** *auto*

**hence** $\exists$ *d*>*0*. *applyToSet* (*asFunc L*) (*ball x d*) $\subseteq$ *ball* $x'$ $e$

**using** *dpos* **by** *auto*

**}**

**hence** $\forall$ *e*>*0*. $\exists$ *d*>*0*. *applyToSet* (*asFunc L*) (*ball x d*) $\subseteq$ *ball* $x'$ $e$

**by** *auto*

**}**

**thus** *?thesis* **by** *auto*

**qed**

**lemma** *lemLinOfLinIsLin*:

**assumes** (*linear A*) $\wedge$ (*linear B*)

**shows** *linear* ($B \circ A$)

**proof** $-$

**have** *1*: ($B \circ A$) *origin* = *origin* **using** *assms* **by** *auto*

**have** *2*: $\forall$ $a$ $p$ . ($B \circ A$)($a \otimes p$) = ($a \otimes$ (($B \circ A$) $p$)) **using** *assms* **by** *auto*

**have** *3*: $\forall$ $p$ $q$ . ($B \circ A$) ($p \oplus q$) = ((($B \circ A$) $p$) $\oplus$ (($B \circ A$) $q$)) **using** *assms* **by** *auto*

**have** *4*: $\forall$ $p$ $q$ . ($B \circ A$) ($p \ominus q$) = ((($B \circ A$) $p$) $\ominus$ (($B \circ A$) $q$)) **using** *assms* **by** *auto*

**thus** *?thesis* **using** *1 2 3* **by** *force*

**qed**

**lemma** *lemInverseLinear*:

**assumes** *linear A*

**and** *invertible A*

**shows** $\exists$ $A'$ . (*linear* $A'$) $\wedge$ ($\forall$ $p$ $q$. $A$ $p$ = $q$ $\longleftrightarrow$ $A'$ $q$ = $p$)

**proof** $-$

**obtain** $L$ **where** $L$: ($\forall$ $p$ $q$. $A$ $p$ = $q$ $\longleftrightarrow$ $L$ $q$ = $p$)

using *assms*(*2*) **by** *metis*

**have** *1*: $L$ *origin* = *origin* **using** *assms* $L$ **by** *auto*

**{ fix** $p'$ $q'$ $a$
**obtain** $p$ **where** $p$: $(A\ p = p') \wedge (\forall z.\ A\ z = p' \longrightarrow z = p)$ **using** *assms*(*2*) **by** *blast*
**obtain** $q$ **where** $q$: $(A\ q = q') \wedge (\forall z.\ A\ z = q' \longrightarrow z = q)$ **using** *assms*(*2*) **by** *blast*

**have** $L\ (a \otimes p') = L\ (\ a \otimes (A\ p)\ )$ **using** $p$ **by** *auto*
**also have** ... $= L\ (\ A\ (\ a \otimes p\ )\ )$ **using** *assms*(*1*) **by** *auto*
**also have** ... $= (a \otimes p)$ **using** $L$ **by** *blast*
**finally have** *2*: $L\ (a \otimes p') = (a \otimes (L\ p'))$ **using** $p$ $L$ **by** *auto*

**have** $L\ (p' \oplus q') = L\ ((A\ p) \oplus (A\ q))$ **using** $p$ $q$ **by** *auto*
**also have** ... $= L(\ A\ (p \oplus q)\ )$ **using** *assms*(*1*) **by** *auto*
**also have** ... $= (p \oplus q)$ **using** $p$ $q$ $L$ **by** *auto*
**finally have** *3*: $L\ (p' \oplus q') = (\ (L\ p') \oplus (L\ q')\ )$ **using** $p$ $q$ $L$ **by** *auto*

**have** $L\ (p' \ominus q') = L\ ((A\ p) \ominus (A\ q))$ **using** $p$ $q$ **by** *auto*
**also have** ... $= L(\ A\ (p \ominus q)\ )$ **using** *assms*(*1*) **by** *auto*
**also have** ... $= (p \ominus q)$ **using** $p$ $q$ $L$ **by** *auto*
**finally have** *4*: $L\ (p' \ominus q') = (\ (L\ p') \ominus (L\ q')\ )$ **using** $p$ $q$ $L$ **by** *auto*

**hence** $(L$ *origin* = *origin*$) \wedge$
$(L\ (a \otimes p') = (a \otimes (L\ p'))) \wedge$
$(L\ (p' \oplus q') = (\ (L\ p') \oplus (L\ q')\ )) \wedge$
$(L\ (p' \ominus q') = (\ (L\ p') \ominus (L\ q')\ ))$
**using** *1 2 3* **by** *auto*
**}**
**hence** *linear* $L$ **by** *auto*

**thus** *?thesis* **using** $L$ **by** *auto*
**qed**

**end**

**end**

# 20 Affine

This theory defines affine transformations and established their key properties.

**theory** *Affine*
  **imports** *Translations LinearMaps*
**begin**

**class** *Affine* = *Translations* + *LinearMaps*
**begin**

**abbreviation** *affine* :: $('a\ Point \Rightarrow 'a\ Point) \Rightarrow bool$
  **where** $affine\ A \equiv \exists\ L\ T\ .\ (linear\ L) \wedge (translation\ T) \wedge (A = T \circ L)$

**abbreviation** *affInvertible* :: $('a\ Point \Rightarrow 'a\ Point) \Rightarrow bool$
  **where** $affInvertible\ A \equiv affine\ A \wedge invertible\ A$

**abbreviation** *isLinearPart* :: $('a\ Point \Rightarrow 'a\ Point) \Rightarrow ('a\ Point \Rightarrow 'a\ Point) \Rightarrow bool$
  **where** $isLinearPart\ A\ L \equiv (affine\ A) \wedge (linear\ L) \wedge$
    $(\exists\ T.\ (translation\ T \wedge A = T \circ L))$

**abbreviation** *isTranslationPart* :: $('a\ Point \Rightarrow 'a\ Point) \Rightarrow ('a\ Point \Rightarrow 'a\ Point) \Rightarrow bool$
  **where** $isTranslationPart\ A\ T \equiv (affine\ A) \wedge (translation\ T) \wedge$
    $(\exists\ L.\ (linear\ L \wedge A = T \circ L))$

## 20.1 Affine approximation

A key concept in the proof is affine approximation. We will eventually assert that worldview transformation can be approximated by invertible affine transformations.

**abbreviation** *affineApprox* :: $('a\ Point \Rightarrow 'a\ Point) \Rightarrow$
    $('a\ Point \Rightarrow 'a\ Point => bool) \Rightarrow$
    $'a\ Point \Rightarrow bool$
  **where** $affineApprox\ A\ f\ x \equiv (isFunction\ f) \wedge$
    $(affInvertible\ A) \wedge (diffApprox\ (asFunc\ A)\ f\ x)$

**fun** *applyAffineToLine* :: $('a\ Point \Rightarrow 'a\ Point)$
    $\Rightarrow 'a\ Point\ set \Rightarrow 'a\ Point\ set \Rightarrow bool$
  **where** $applyAffineToLine\ A\ l\ l' \longleftrightarrow (affine\ A) \wedge$

$(\exists\ T\ L\ b\ d\ .\ ((linear\ L) \wedge (translation\ T) \wedge (A = T \circ L) \wedge$
$(l = line\ b\ d) \wedge (l' = (line\ (A\ b)\ (L\ d)))))$

**abbreviation** *affConstantOn* :: $('a\ Point \Rightarrow 'Point) \Rightarrow 'a\ Point \Rightarrow$ $'a\ Point\ set \Rightarrow bool$
**where** *affConstantOn A x s* $\equiv (\exists \varepsilon{>}0.\ \forall y{\in}s.\ (y\ within\ \varepsilon\ of\ x) \longrightarrow ((A\ y) = (A\ x)))$

**lemma** *lemTranslationPartIsUnique*:
**assumes** *isTranslationPart A T1*
**and** *isTranslationPart A T2*
**shows** $T1 = T2$
**proof** −
**obtain** *L1* **where** *T1*: $linear\ L1 \wedge A = T1 \circ L1$ **using** *assms*(1) **by** *auto*
**obtain** *L2* **where** *T2*: $linear\ L2 \wedge A = T2 \circ L2$ **using** *assms*(2) **by** *auto*
**obtain** *t1* **where** *t1*: $\forall\ x.\ T1\ x = (x \oplus t1)$ **using** *assms*(1) **by** *auto*
**obtain** *t2* **where** *t2*: $\forall\ x.\ T2\ x = (x \oplus t2)$ **using** *assms*(2) **by** *auto*

**have** $T1\ origin = A\ origin$ **using** *T1 assms*(1) **by** *auto*
**also have** $... = T2\ origin$ **using** *T2 assms*(2) **by** *auto*
**finally have** $T1\ origin = T2\ origin$ **by** *auto*
**hence** $t1 = t2$ **using** *t1 t2* **by** *auto*
**hence** $\forall\ x.\ (T1\ x = T2\ x)$ **using** *t1 t2* **by** *auto*
**thus** *?thesis* **by** *auto*
**qed**

**lemma** *lemLinearPartIsUnique*:
**assumes** *isLinearPart A L1*
**and** *isLinearPart A L2*
**shows** $L1 = L2$
**proof** −
**obtain** *T1* **where** *T1*: $translation\ T1 \wedge A = T1 \circ L1$ **using** *assms*(1) **by** *auto*
**obtain** *T2* **where** *T2*: $translation\ T2 \wedge A = T2 \circ L2$ **using** *assms*(2) **by** *auto*

**have** *1*: *isTranslationPart A T1* **using** *assms*(1) *T1* **by** *auto*
**have** *2*: *isTranslationPart A T2* **using** *assms*(2) *T2* **by** *auto*

**hence** *T1T2*: *T1* = *T2* **using** *1 2 lemTranslationPartIsUnique*[*of A T1 T2*] **by** *auto*

**obtain** *t* **where** *t*: ∀ *x*. *T1 x* = (*x* ⊕ *t*) **using** *T1* **by** *auto*
**define** *T* **where** *T* = *mkTranslation* (*origin* ⊖ *t*)
**hence** *3*: *T* ∘ *A* = *L1* **using** *T1 t lemInverseTranslation* **by** *auto*
**have** *T* ∘ *A* = *L2* **using** *T-def T2 t T1T2 lemInverseTranslation* **by** *auto*

**thus** *?thesis* **using** *3* **by** *auto*
**qed**

**lemma** *lemLinearImpliesAffine*:
**assumes** *linear L*
**shows** *affine L*
**proof** −
**have** *1*: *L* = *id* ∘ *L* **by** *fastforce*
**thus** *?thesis* **using** *assms lemIdIsTranslation* **by** *blast*
**qed**

**lemma** *lemTranslationImpliesAffine*:
**assumes** *translation T*
**shows** *affine T*
**proof** −
**have** *T* = *T* ∘ *id* **by** *force*
**thus** *?thesis* **using** *assms lemIdIsLinear* **by** *blast*
**qed**

**lemma** *lemAffineDiff*:
**assumes** *linear L*
**and** ∃ *T* . ((*translation T*) ∧ (*A* = *T* ∘ *L*))
**shows** ((*A p*) ⊖ (*A q*)) = *L* (*p* ⊖ *q*)
**proof** −
**obtain** *T* **where** *T*: (*translation T*) ∧ (*A* = *T* ∘ *L*) **using** *assms*(*2*) **by** *auto*
**thus** *?thesis* **using** *assms*(*1*) **by** *auto*
**qed**

**lemma** *lemAffineImpliesTotalFunction*:
**assumes** *affine A*
**shows** *isTotalFunction* (*asFunc A*)
**by** *simp*

**lemma** *lemAffineEqualAtBase*:

**assumes** *affineApprox A f x*
**shows** $\forall y.\ (f\ x\ y) \longleftrightarrow (y = A\ x)$
**proof** –
**have** *diff*: *diffApprox* (*asFunc A*) *f x* **using** *assms*(*1*) **by** *simp*
**{ fix** *y*
**assume** *y*: *f x y*
**hence** $f\ x\ y \wedge (asFunc\ A)\ x\ (A\ x)$ **by** *auto*
**hence** $A\ x = y$ **using** *diff lemApproxEqualAtBase*[*of f x asFunc A*
*y*]
**by** *auto*
**}**
**hence** *l2r*: $\forall y\ .\ f\ x\ y \longrightarrow y = A\ x$ **by** *auto*

**{ obtain** *y* **where** *y*: *f x y* **using** *diff* **by** *auto*
**hence** $y = A\ x$ **using** *l2r* **by** *auto*
**hence** $f\ x\ (A\ x)$ **using** *y* **by** *auto*
**}**
**thus** *?thesis* **using** *l2r* **by** *blast*
**qed**

**lemma** *lemAffineOfPointOnLine*:
**assumes** $(linear\ L) \wedge (translation\ T) \wedge (A = T \circ L)$
**and** $x = (b \oplus (a \otimes d))$
**shows** $A\ x = ((A\ b) \oplus (a \otimes (L\ d)))$
**proof** –
**have** $(L\ x = ((L\ b) \oplus (L\ (a \otimes d)))) \wedge (L\ (a \otimes d) = (a \otimes (L\ d)))$
**using** *assms* **by** *blast*
**hence** $A\ x = T\ ((L\ b) \oplus (a \otimes (L\ d)))$ **using** *assms*(*1*) **by** *auto*
**also have** ... $= ((T\ (L\ b)) \oplus (a \otimes (L\ d)))$
**using** *assms*(*1*) *lemTranslationSum*[*of T L b a* $\otimes$ (*L d*)] **by** *auto*
**finally show** *?thesis* **using** *assms*(*1*) **by** *auto*
**qed**

**lemma** *lemAffineOfLineIsLine*:
**assumes** *isLine l*
**shows** $(applyAffineToLine\ A\ l\ l') \longleftrightarrow (affine\ A \wedge l' = applyToSet$
$(asFunc\ A)\ l)$
**proof** –
**{ assume** *lhs*: $applyAffineToLine\ A\ l\ l'$
**hence** *affA*: *affine A* **by** *fastforce*
**have** $\exists\ T\ L\ b\ d\ .\ (linear\ L) \wedge (translation\ T) \wedge (A = T \circ L) \wedge$
$(l = line\ b\ d) \wedge (l' = (line\ (A\ b)\ (L\ d)))$ **using** *lhs* **by** *auto*
**then obtain** *T L b d* **where** *TL*: $(linear\ L) \wedge (translation\ T) \wedge$
$(A = T \circ L) \wedge$
$(l = line\ b\ d) \wedge (l' = (line\ (A\ b)\ (L\ d)))$

**using** *lhs* **by** *blast*
**{ fix** $p'$
**{ assume** $p' \in l'$
**then obtain** $a$ **where** $a$: $p' = ( (A\ b) \oplus (a \otimes (L\ d)) )$ **using** *TL* **by** *auto*
**define** $p$ **where** $p$: $p = (b \oplus (a \otimes d))$
**hence** $p' \in$ *applyToSet* (*asFunc* $A$) $l$ **using** $a$ *TL lemAffineOfPointOnLine* **by** *auto*
**}**
**hence** $(p' \in l') \longrightarrow (p' \in$ *applyToSet* (*asFunc* $A$) $l)$ **by** *auto*
**}**
**hence** *l2r*: $l' \subseteq$ (*applyToSet* (*asFunc* $A$) $l$) **by** *auto*

**{ fix** $p'$
**{ assume** $p' \in$ *applyToSet* (*asFunc* $A$) $l$
**then obtain** $p$ **where** $p$: $p \in l \wedge p' = A\ p$ **by** *auto*
**then obtain** $a$ **where** $a$: $p = (b \oplus (a \otimes d))$ **using** *TL* **by** *auto*
**hence** $A\ p = ((A\ b) \oplus (a \otimes (L\ d)))$ **using** *TL lemAffineOfPointOnLine* **by** *auto*
**hence** $p' \in l'$ **using** *TL* $p$ **by** *auto*
**}**
**hence** $(p' \in$ *applyToSet* (*asFunc* $A$) $l) \longrightarrow (p' \in l')$ **using** *l2r* **by** *auto*
**}**
**hence** (*applyToSet* (*asFunc* $A$) $l$) $\subseteq l'$ **by** *auto*
**hence** *affine* $A \wedge l' =$ *applyToSet* (*asFunc* $A$) $l$ **using** *affA l2r* **by** *auto*
**}**
**hence** *rtp1*: (*applyAffineToLine* $A\ l\ l'$) $\longrightarrow$ (*affine* $A \wedge l' =$ *applyToSet* (*asFunc* $A$) $l$)
**by** *blast*

**{ assume** *rhs*: (*affine* $A$) $\wedge$ ($l' =$ *applyToSet* (*asFunc* $A$) $l$)

**obtain** $b\ d$ **where** *bd*: $l =$ *line* $b\ d$ **using** *assms*(*1*) **by** *auto*
**obtain** $T\ L$ **where** *TL*: (*linear* $L$) $\wedge$ (*translation* $T$) $\wedge$ $(A = T \circ L)$
**using** *rhs* **by** *auto*

**{ fix** $p'$
**assume** $p' \in l'$
**then obtain** $p$ **where** $p$: $(p \in l) \wedge (A\ p = p')$ **using** *rhs* **by** *auto*
**then obtain** $a$ **where** $a$: $p = (b \oplus (a \otimes d))$ **using** *bd* **by** *auto*
**hence** $A\ p = ((A\ b) \oplus (a \otimes (L\ d)))$
**using** *TL lemAffineOfPointOnLine* **by** *auto*
**hence** $p' \in$ *line* $(A\ b)\ (L\ d)$ **using** $p$ **by** *auto*
**}**
**hence** *l2r*: $l' \subseteq$ *line* $(A\ b)\ (L\ d)$ **by** *force*

{ **fix** $p'$
**assume** $p' \in$ *line* $(A\ b)$ $(L\ d)$
**then obtain** $a$ **where** $a$: $p' = (\ (A\ b) \oplus (a \otimes (L\ d))\ )$ **using** *TL* **by** *auto*
**define** $p$ **where** $p$: $p = (b \oplus (a {\otimes} d))$
**hence** $A\ p = ((A\ b) \oplus (a \otimes (L\ d)))$
**using** *TL lemAffineOfPointOnLine* **by** *auto*
**hence** $A\ p = p'$ **using** $a$ **by** *simp*
**hence** $p' \in$ *applyToSet* (*asFunc* $A$) $l$ **using** $p$ *bd* **by** *auto*
}
**hence** *line* $(A\ b)$ $(L\ d) = l'$ **using** *rhs l2r* **by** *blast*

**hence** *applyAffineToLine* $A\ l\ l'$ **using** *TL bd* **by** *auto*
}
**hence** (*affine* $A$) $\wedge$ ($l' =$ *applyToSet* (*asFunc* $A$) $l$)
$\longrightarrow$ (*applyAffineToLine* $A\ l\ l'$)
**by** *blast*

**thus** *?thesis* **using** *rtp1* **by** *blast*
**qed**

**lemma** *lemOnLineUnderAffine*:
**assumes** (*affine* $A$) $\wedge$ (*onLine* $p\ l$)
**shows** *onLine* $(A\ p)$ (*applyToSet* (*asFunc* $A$) $l$)
**proof** $-$

**define** $l'$ **where** $l'$: $l' =$ *applyToSet* (*asFunc* $A$) $l$
**have** *lineL*: *isLine* $l$ **using** *assms* **by** *auto*

**hence** *Tll'*: *applyAffineToLine* $A\ l\ l'$
**using** *lemAffineOfLineIsLine*[*of* $l\ A\ l'$] *assms* $l'$
**by** *blast*
**hence** $\exists\ T'\ L\ b\ d$ . (*linear* $L$) $\wedge$ (*translation* $T'$) $\wedge$ ($A = T' \circ L$) $\wedge$
($l =$ *line* $b\ d$) $\wedge$ ($l' =$ (*line* $(A\ b)$ $(L\ d)$)) **by** *force*
**then obtain** $T'\ L\ b\ d$
**where** *TLbd*: (*linear* $L$) $\wedge$ (*translation* $T'$) $\wedge$ ($A = T' \circ L$) $\wedge$
($l =$ *line* $b\ d$) $\wedge$ ($l' =$ (*line* $(A\ b)$ $(L\ d)$)) **by** *blast*
**then obtain** $a$ **where** $a$: $p = (b \oplus (a {\otimes} d))$ **using** *assms* **by** *auto*

**hence** $A\ p = ((A\ b) \oplus (a \otimes (L\ d)))$ **using** *lemAffineOfPointOnLine* *TLbd* **by** *auto*
**thus** *?thesis* **using** $l'$ *TLbd* **by** *blast*
**qed**

**lemma** *lemLineJoiningUnderAffine*:

**assumes** *affine A*
**shows** *applyToSet* (*asFunc A*) (*lineJoining p q*) = *lineJoining* (*A p*) (*A q*)
**proof** –
**obtain** *T L* **where** *TL*: *translation T* $\wedge$ *linear L* $\wedge$ *A* = *T*$\circ$*L* **using** *assms*(*1*) **by** *auto*
**hence** ((*A q*) $\ominus$ (*A p*)) = *L* (*q*$\ominus$*p*) **by** *auto*

{ **fix** *a*
**have** (*a*$\otimes$((*A q*) $\ominus$ (*A p*))) = *L* (*a* $\otimes$ (*q*$\ominus$*p*))
**using** *TL lemLinearProps*[*of L a q*$\ominus$*p*] **by** *force*
}
**hence** *as*: $\forall$ *a*. (*a*$\otimes$((*A q*) $\ominus$ (*A p*))) = *L* (*a* $\otimes$ (*q*$\ominus$*p*)) **by** *auto*

{ **fix** *x′*
**assume** *x′* $\in$ *applyToSet* (*asFunc A*) (*lineJoining p q*)
**then obtain** *x* **where** *x*: *x* $\in$ (*lineJoining p q*) $\wedge$ *x′* = *A x* **by** *force*
**then obtain** *a* **where** *a*: *x* = (*p* $\oplus$ (*a*$\otimes$(*q*$\ominus$*p*))) **by** *force*

**have** *expandL*: *L* (*p* $\oplus$ (*a*$\otimes$(*q*$\ominus$*p*))) = ((*L p*) $\oplus$ (*L* (*a*$\otimes$(*q*$\ominus$*p*))))
**using** *TL lemLinearProps*[*of L 0 p* (*a*$\otimes$(*q*$\ominus$*p*))]
**by** *fast*

**have** *x′* = *A* (*p* $\oplus$ (*a*$\otimes$(*q*$\ominus$*p*))) **using** *x a* **by** *fast*
**also have** ... = (*T* (*L* (*p* $\oplus$ (*a*$\otimes$(*q*$\ominus$*p*))))) **using** *TL* **by** *force*
**also have** ... = *T* ((*L p*) $\oplus$ (*L* (*a*$\otimes$(*q*$\ominus$*p*)))) **using** *expandL* **by** *force*
**finally have** *x′* = ((*T* (*L p*)) $\oplus$ (*L* (*a*$\otimes$(*q*$\ominus$*p*))))
**using** *TL lemTranslationSum*[*of T L p L* (*a*$\otimes$(*q*$\ominus$*p*))]
**by** *auto*
**hence** *x′* $\in$ *lineJoining* (*A p*) (*A q*) **using** *TL as* **by** *auto*
}
**hence** *l2r*: *applyToSet* (*asFunc A*) (*lineJoining p q*) $\subseteq$ *lineJoining* (*A p*) (*A q*)
**by** *force*

{ **fix** *x′*
**assume** *x′* $\in$ *lineJoining* (*A p*) (*A q*)
**hence** $\exists$ *a* . *x′* = ((*T* (*L p*)) $\oplus$ (*a*$\otimes$((*A q*)$\ominus$(*A p*))))
**using** *TL* **by** *auto*
**then obtain** *a* **where** *a*: *x′* = ((*T* (*L p*)) $\oplus$ (*a*$\otimes$((*A q*)$\ominus$(*A p*))))
**using** *TL* **by** *fast*
**hence** *x′* = ((*T* (*L p*)) $\oplus$ (*L* (*a*$\otimes$(*q*$\ominus$*p*)))) **using** *as* **by** *force*
**also have** ... = *T* ( (*L p*) $\oplus$ (*L* (*a*$\otimes$(*q*$\ominus$*p*))))
**using** *TL lemTranslationSum*[*of T L p L* (*a*$\otimes$(*q*$\ominus$*p*))] **by** *simp*
**also have** ... = *T* ( *L* (*p* $\oplus$ (*a*$\otimes$(*q*$\ominus$*p*))) )
**using** *TL lemLinearProps*[*of L 0 p a*$\otimes$(*q*$\ominus$*p*)] **by** *auto*
**finally have** *x′* = *A* (*p* $\oplus$ (*a*$\otimes$(*q*$\ominus$*p*))) **using** *TL* **by** *auto*

hence $x' \in$ *applyToSet* (*asFunc A*) (*lineJoining p q*) **by** *auto*
}
**thus** *?thesis* **using** *l2r* **by** *auto*
**qed**

**lemma** *lemAffineIsCts*:
**assumes** *affine A*
**shows** *cts* (*asFunc A*) *x*
**proof** −

**have** $\exists$ *T L* . (*translation T*)$\wedge$(*linear L*)$\wedge$(*A* = *T* $\circ$ *L*) **using** *assms* **by** *auto*
**then obtain** *T L* **where** *TL*: (*translation T*)$\wedge$(*linear L*)$\wedge$(*A* = *T* $\circ$ *L*) **by** *auto*

**define** *f* **where** *f*: *f* = *asFunc L*
**define** *g* **where** *g*: *g* = *asFunc T*
**have** *1*: *cts f x* **using** *f TL lemLinearIsCts*[*of L x*] **by** *auto*
**have** *2*: $\forall$ *y*. (*f x y*) $\longrightarrow$ (*cts g y*)
**using** *f g TL lemTranslationIsCts*[*of T x*] **by** *auto*
**have** *cts* (*composeRel g f*) *x* **using** *1 2 lemCtsOfCtsIsCts*[*of f x g*] **by** *simp*
**thus** *?thesis* **using** *f g TL* **by** *auto*
**qed**

**lemma** *lemAffineContinuity*:
**assumes** *affine A*
**shows** $\forall$ *x*. $\forall\varepsilon$>*0*. $\exists\delta$>*0* . $\forall$ *p*. (*p within* $\delta$ *of x*) $\longrightarrow$ ((*A p*) *within* $\varepsilon$ *of* (*A x*))
**proof** −
{ **fix** *x*
{ **fix** *e*
**assume** *epos*: *e* > *0*

**have** (*asFunc A*) *x* (*A x*) $\wedge$ (*cts* (*asFunc A*) *x*)
**using** *assms lemAffineIsCts*[*of A x*] **by** *auto*
**hence** *u*: ($\forall\varepsilon$>*0*. $\exists\delta$>*0*. (*applyToSet* (*asFunc A*) (*ball x* $\delta$)) $\subseteq$ *ball* (*A x*) $\varepsilon$)
**by** *force*
**then obtain** *d* **where** *d*: (*d* > *0*) $\wedge$
(*applyToSet* (*asFunc A*) (*ball x d*)) $\subseteq$ *ball* (*A x*) *e*
**using** *epos* **by** *force*

{ **fix** *p*

**assume** *p within d of x*
**hence** (*A p*) *within e of* (*A x*) **using** *d lemSep2Symmetry* **by** *force*
**}**
**hence** $\exists$ *d>0* . $\forall$ *p*. (*p within d of x*) $\longrightarrow$ ((*A p*) *within e of* (*A x*))
**using** *d* **by** *auto*
**}**
**hence** $\forall$ *e>0*. $\exists$ *d>0* . $\forall$ *p*. (*p within d of x*) $\longrightarrow$ ((*A p*) *within e of* (*A x*))
**by** *auto*
**}**
**thus** *?thesis* **by** *auto*
**qed**

**lemma** *lemAffOfAffIsAff*:
**assumes** (*affine A*) $\wedge$ (*affine B*)
**shows** *affine* (*B* $\circ$ *A*)
**proof** –
**obtain** *TA LA TB LB* **where** *props*:
*translation TA* $\wedge$ *linear LA* $\wedge$ *translation TB* $\wedge$ *linear LB* $\wedge$
*A* = *TA* $\circ$ *LA* $\wedge$ *B* = *TB* $\circ$ *LB* **using** *assms* **by** *blast*
**then obtain** *ta tb* **where** *ts*: ($\forall$ *p*. *TA p* = (*p* $\oplus$ *ta*)) $\wedge$ ($\forall$ *p*. *TB p* = (*p* $\oplus$ *tb*)) **by** *auto*

**{ fix** *p*
**have** (*B* $\circ$ *A*) *p* = ((*LB* ( (*LA p*) $\oplus$ *ta* )) $\oplus$ *tb*) **using** *props ts* **by** *force*
**also have** ... = (((*LB* (*LA p*)) $\oplus$ (*LB ta*)) $\oplus$ *tb*) **using** *props* **by** *force*
**also have** ... = (((*LB*$\circ$*LA*) *p*) $\oplus$ ((*LB ta*)$\oplus$*tb*)) **using** *add-assoc* **by** *force*
**finally have** (*B* $\circ$ *A*) *p* = ((*mkTranslation* ((*LB ta*)$\oplus$*tb*)) $\circ$ (*LB*$\circ$*LA*)) *p* **by** *force*
**}**
**hence** *BA*: (*B* $\circ$ *A*) = (*mkTranslation* ((*LB ta*)$\oplus$*tb*)) $\circ$ (*LB*$\circ$*LA*) **by** *auto*

**define** *T* **where** *T*: *T* = *mkTranslation* ((*LB ta*)$\oplus$*tb*)
**hence** *trans*: *translation T* **using** *lemMkTrans* **by** *blast*
**define** *L* **where** *L*: *L* = (*LB*$\circ$*LA*)
**hence** *lin*: *linear L* **using** *lemLinOfLinIsLin*[*of LA LB*] *props* **by** *auto*

**hence** (*translation T*) $\wedge$ (*linear L*) $\wedge$ ((*B*$\circ$*A*) = (*T*$\circ$*L*)) **using** *T L trans lin BA* **by** *auto*
**thus** *?thesis* **by** *auto*
**qed**

**lemma** *lemInverseAffine*:
**assumes** *affInvertible* $A$
**shows** $\exists A'$ . (*affine* $A'$) $\wedge$ ($\forall$ $p$ $q$ . $A$ $p = q \longleftrightarrow A'$ $q = p$)
**proof** –
**obtain** $A'$ **where** $A'$: ($\forall$ $p$ $q$. $A$ $p = q \longleftrightarrow A'$ $q = p$)
**using** *assms* **by** *metis*

**obtain** $T$ $L$ **where** $TL$: *translation* $T$ $\wedge$ *linear* $L$ $\wedge$ ($A = T \circ L$) **using** *assms*(*1*) **by** *auto*

**obtain** $T'$ **where** $T'$: (*translation* $T'$) $\wedge$ ($\forall$ $p$ $q$ . $T$ $p = q \longleftrightarrow T'$ $q = p$)
**using** $TL$ *lemInverseTrans*[*of* $T$] **by** *auto*

**{ fix** $p$
**{ fix** $q$
**assume** $Ap$: $A$ $p = q$
**hence** $T$ ($L$ $p$) $= q$ **using** $TL$ **by** *auto*
**hence** $L$ $p = T'$ $q$ **using** $T'$ **by** *auto*
**hence** $L$ $p = (T' \circ A)$ $p$ **using** $Ap$ **by** *auto*
**}**
**}**
**hence** $L$: $L = (T' \circ A)$ **by** *auto*

**{ fix** $q$
**obtain** $r$ **where** $r$: ($T'$ $r = q$) **using** $T'$ **by** *auto*
**then obtain** $p$ **where** $p$: ($A$ $p = r$) $\wedge$ ($\forall x$. $A$ $x = r \longrightarrow x = p$) **using** $A'$ **by** *auto*
**hence** *1*: $L$ $p = q$ **using** $L$ $r$ **by** *auto*
**{ fix** $x$
**assume** $L$ $x = q$
**hence** $T'$ ($A$ $x$) $= q$ **using** $L$ **by** *auto*
**hence** $A$ $x = r$ **using** $r$ $T'$ *lemTranslationInjective*[*of* $T'$] **by** *force*
**hence** $x = p$ **using** $p$ $A'$ **by** *blast*
**} hence** $\exists p$ . ($L$ $p = q$) $\wedge$ ($\forall x$. $L$ $x = q \longrightarrow x = p$) **using** *1* **by** *auto*
**}**
**hence** *invL*: *invertible* $L$ **by** *blast*

**then obtain** $L'$ **where** $L'$: (*linear* $L'$) $\wedge$ ($\forall$ $p$ $q$ . $L$ $p = q \longleftrightarrow L'$ $q = p$)
**using** $TL$ *lemInverseLinear*[*of* $L$] **by** *blast*

{ **fix** $p\ q$
  **have** $A'\ q = p \ \longleftrightarrow\ T\ (L\ p) = q$ **using** $A'\ TL$ **by** *auto*
  **also have** ... $\longleftrightarrow\ T'\ q = L\ p$ **using** $T'$ **by** *auto*
  **also have** ... $\longleftrightarrow\ L\ p = T'\ q$ **by** *auto*
  **also have** ... $\longleftrightarrow\ L'\ (T'\ q) = p$ **using** $L'$ **by** *auto*
  **finally have** $A'\ q = p \ \longleftrightarrow\ (L' \circ T')\ q = p$ **by** *auto*
}
**hence** $A' = L' \circ T'$ **by** *auto*
**hence** *affine* $A'$ **using** *lemAffOfAffIsAff*[*of* $T'\ L'$]
*lemTranslationImpliesAffine*[*of* $T'$] $T'$
*lemLinearImpliesAffine*[*of* $L'$] $L'$
  **by** *auto*

**thus** *?thesis* **using** $A'$ **by** *auto*
**qed**

**lemma** *lemAffineApproxDomainTranslation*:
**assumes** *translation* $T$
**and** *affineApprox* $A\ f\ x$
**and** $\forall\ p\ q\ .\ T\ p = q \ \longleftrightarrow\ T'\ q = p$
**shows** *affineApprox* $(A \circ T)$ (*composeRel* $f$ (*asFunc* $T$)) $(T'\ x)$
**proof** −

**define** $A0$ **where** $A0$: $A0 = A \circ T$
**define** $g$ **where** $g$: $g =$ *composeRel* $f$ (*asFunc* $T$)

**have** $ToT'$: $\forall\ p\ .\ T\ (T'\ p) = p$ **using** *assms*(*3*) **by** *force*
**have** $T'oT$: $\forall\ p\ .\ T'\ (T\ p) = p$ **using** *assms*(*3*) **by** *force*
**obtain** $t$ **where** $t$: $\forall\ p\ .\ T\ p = (p \oplus t)$ **using** *assms*(*1*) **by** *force*
**hence** *mkT*: $T =$ *mkTranslation* $t$ **by** *force*

{ **fix** $p\ q$
  **have** $T'\ p = q \ \longleftrightarrow\ T\ q = p$ **using** *assms*(*3*) **by** *auto*
  **also have** ... $\longleftrightarrow (q \oplus t) = p$ **using** $t$ **by** *auto*
  **also have** ... $\longleftrightarrow q = (p \oplus (origin \ominus t))$ **by** *force*
  **finally have** $T'\ p = q \ \longleftrightarrow\ q = (p \oplus (origin \ominus t))$ **by** *force*
  **hence** $T'\ p = q \ \longleftrightarrow\ q =$ *mkTranslation* $(origin \ominus t)\ p$ **by** *force*
}
**hence** $mkT'$: $T' =$ *mkTranslation* $(origin \ominus t)$ **by** *force*
**hence** $transT'$: *translation* $T'$ **using** *lemMkTrans* **by** *blast*

**have** *funcF*: *isFunction* $f$ **using** *assms*(*2*) **by** *auto*
**hence** *rtp3a*: *isFunction* $g$ **using** $g$ **by** *auto*

**have** *affA*: *affine* $A$ **using** *assms*(*2*) **by** *auto*

**hence** *rtp3b*: *affine A0*
**using** *lemAffOfAffIsAff*[*of T A*] *lemTranslationImpliesAffine*[*of T*]
*A0 affA assms*(*1*)
**by** *blast*

{ **fix** $q$
**obtain** $p$ **where** $p$: $(A\ p = q) \land (\forall x.\ A\ x = q \longrightarrow x = p)$ **using** *assms*(*2*) **by** *blast*
**define** *p0* **where** *p0*: $p0 = T'\ p$
**hence** *Tp0*: $T\ p0 = p$ **using** *assms*(*3*) **by** *blast*
**hence** *1*: $A0\ p0 = q$ **using** *A0 p* **by** *auto*
{ **fix** $x$
**assume** $A0\ x = q$
**hence** $T\ x = p$ **using** *A0 p* **by** *fastforce*
**hence** $x = p0$ **using** *Tp0 assms*(*1*) *lemTranslationInjective*[*of T*] **by** *force*
}
**hence** $\forall x.\ A0\ x = q \longrightarrow x = p0$ **by** *auto*

**hence** $\exists p0.\ (A0\ p0 = q) \land (\forall x.\ A0\ x = q \longrightarrow x = p0)$ **using** *1* **by** *auto*
}
**hence** *rtp3c*: *invertible A0* **by** *auto*

**have** *diffApprox* (*asFunc A*) *f x* **using** *assms*(*2*) **by** *auto*
**hence** *dAx*: (*definedAt f x*) $\land$
$(\forall\ \varepsilon > 0\ .\ (\exists\ \delta > 0\ .\ (\forall\ y\ .$
( (*y within* $\delta$ *of x*)
$\longrightarrow$
( (*definedAt f y*) $\land$ $(\forall\ u\ v\ .$ (*f y u* $\land$ (*asFunc A*) *y v*) $\longrightarrow$
( *sep2 v u* ) $\leq$ (*sqr* $\varepsilon$) $*$ *sep2 y x* ))) )
)) **by** *blast*
**hence** (*definedAt f x*) $\land$ $(x = T\ (T'\ x))$ **using** *assms*(*1*) $ToT'$ **by** *auto*
**hence** *rtp3d1*: (*definedAt g* $(T'\ x)$) **using** *g* **by** *auto*

{ **fix** $e$
**assume** *epos*: $e > 0$
**then obtain** $d$ **where** $d$: $(d > 0) \land (\forall\ y\ .$
( (*y within d of x*)
$\longrightarrow$
( (*definedAt f y*) $\land$ $(\forall\ u\ v\ .$ (*f y u* $\land$ (*asFunc A*) *y v*) $\longrightarrow$
( *sep2 v u* ) $\leq$ (*sqr e*) $*$ *sep2 y x* ))) )
**using** *dAx* **by** *force*
{ **fix** $y$
**assume** *y within d of* $(T'\ x)$
**hence** $(T\ y)$ *within d of* $(T\ (T'\ x))$ **using** *assms*(*1*) *lemBall-*

*Translation* **by** *auto*
**hence** $(T\ y)$ *within* $d$ *of* $x$ **using** $ToT'$ **by** *auto*
**hence** $(definedAt\ f\ (T\ y)) \wedge (\forall\ u\ v\ .\ (f\ (T\ y)\ u \wedge (asFunc\ A)\ (T\ y)\ v) \longrightarrow$
$(\ sep2\ v\ u\ ) \leq (sqr\ e) * sep2\ (T\ y)\ x\ )$
**using** $d$ **by** *blast*
**hence** $(definedAt\ g\ y) \wedge (\forall\ u\ v\ .\ (g\ y\ u \wedge (asFunc\ A0)\ y\ v) \longrightarrow$
$(\ sep2\ v\ u\ ) \leq (sqr\ e) * sep2\ (T\ y)\ x\ )$ **using** $g$ $A0$ **by** *auto*
**hence** $(definedAt\ g\ y) \wedge (\forall\ u\ v\ .\ (g\ y\ u \wedge (asFunc\ A0)\ y\ v) \longrightarrow$
$(\ sep2\ v\ u\ ) \leq (sqr\ e) * sep2\ y\ (T'\ x)\ )$
**using** $transT'$ *lemTranslationPreservesSep2*[*of* $T'$ $T$ $y$ $x$] $T'oT$
**by** *auto*
**}**
**hence** $\exists\, d > 0.\ \forall\, y\ .\ (y$ *within* $d$ *of* $(T'\ x)) \longrightarrow$
$(definedAt\ g\ y) \wedge (\forall\ u\ v\ .\ (g\ y\ u \wedge (asFunc\ A0)\ y\ v) \longrightarrow$
$(\ sep2\ v\ u\ ) \leq (sqr\ e) * sep2\ y\ (T'\ x)\ )$
**using** $d$ **by** *fast*
**}**
**hence** *rtp3d2*: $\forall\, e{>}0.\ \exists\, d > 0.\ \forall\, y\ .\ (y$ *within* $d$ *of* $(T'\ x)) \longrightarrow$
$(definedAt\ g\ y) \wedge (\forall\ u\ v\ .\ (g\ y\ u \wedge (asFunc\ A0)\ y\ v) \longrightarrow$
$(\ sep2\ v\ u\ ) \leq (sqr\ e) * sep2\ y\ (T'\ x)\ )$
**by** *auto*
**hence** *rtp3d*: *diffApprox* $(asFunc\ A0)\ g\ (T'\ x)$ **using** *rtp3d1* **by** *fast*

**have** *rtp3*: *affineApprox* $A0\ g\ (T'\ x)$ **using** *rtp3a rtp3b rtp3c rtp3d* **by** *blast*

**thus** *?thesis* **using** $A0$ $g$ **by** *fast*
**qed**

**lemma** *lemAffineApproxRangeTranslation*:
**assumes** *translation* $T$
**and** *affineApprox* $A\ f\ x$
**shows** *affineApprox* $(T \circ A)\ (composeRel\ (asFunc\ T)\ f)\ x$
**proof** –

**define** $A0$ **where** $A0$: $A0 = T \circ A$
**define** $g$ **where** $g$: $g = composeRel\ (asFunc\ T)\ f$

**obtain** $T'$ **where** $T'$: $(translation\ T') \wedge (\forall\ p\ q\ .\ T\ p = q \longleftrightarrow T'\ q = p)$
**using** *assms*(1) *lemInverseTrans*[*of* $T$] **by** *auto*

**have** $ToT'$: $\forall\ p\ .\ T\ (T'\ p) = p$ **using** $T'$ **by** *force*
**have** $T'oT$: $\forall\ p\ .\ T'\ (T\ p) = p$ **using** $T'$ **by** *force*
**obtain** $t$ **where** $t$: $\forall\ p\ .\ T\ p = (p \oplus t)$ **using** *assms*(1) **by** *auto*

**hence** *mkT*: $T$ = *mkTranslation* $t$ **by** *auto*

**{ fix** $p$ $q$
**have** $T'$ $p = q \longleftrightarrow T\ q = p$ **using** $T'$ **by** *auto*
**also have** ... $\longleftrightarrow (q \oplus t) = p$ **using** $t$ **by** *auto*
**also have** ... $\longleftrightarrow q = (p \oplus (origin \ominus t))$ **by** *force*
**finally have** $T'\ p = q \longleftrightarrow q = (p \oplus (origin \ominus t))$ **by** *force*
**hence** $T'\ p = q \longleftrightarrow q$ = *mkTranslation* $(origin \ominus t)\ p$ **by** *force*
**}**
**hence** *mkT'*: $T'$ = *mkTranslation* $(origin \ominus t)$ **by** *auto*
**hence** *transT'*: *translation* $T'$ **using** *lemMkTrans* **by** *blast*

**have** *funcF*: *isFunction f* **using** *assms(2)* **by** *auto*
**hence** *rtp3a*: *isFunction g* **using** *g* **by** *auto*

**have** *affA*: *affine A* **using** *assms(2)* **by** *auto*
**hence** *rtp3b*: *affine A0*
**using** *lemAffOfAffIsAff[of A T] lemTranslationImpliesAffine[of T]*
*A0 affA assms(1)*
**by** *blast*

**{ fix** $q$
**obtain** $p$ **where** $p$: $(A\ p = T'\ q) \wedge (\forall x.\ A\ x = T'\ q \longrightarrow x = p)$
**using** *assms(2)* **by** *blast*
**hence** $T'\ q = A\ p$ **by** *auto*
**hence** $T\ (A\ p) = q$ **using** $T'$ *ToT'* **by** *auto*
**hence** *1*: $A0\ p = q$ **using** *A0* **by** *auto*

**{ fix** $x$
**assume** $A0\ x = q$
**hence** $T\ (A\ x) = q$ **using** *A0* **by** *auto*
**hence** $T'\ (T\ (A\ x)) = T'\ q$ **by** *auto*
**hence** $A\ x = T'\ q$ **using** *T'oT* **by** *auto*
**hence** $x = p$ **using** $p$ **by** *auto*
**}**
**hence** $\forall x.\ A0\ x = q \longrightarrow x = p$ **by** *auto*

**hence** $\exists p0.\ (A0\ p0 = q) \wedge (\forall x.\ A0\ x = q \longrightarrow x = p0)$ **using** *1*
**by** *auto*
**}**
**hence** *rtp3c*: *invertible A0* **by** *auto*

**have** *diffApprox (asFunc A) f x* **using** *assms(2)* **by** *auto*
**hence** *dAx*: *(definedAt f x)* $\wedge$
$(\forall\ \varepsilon > 0\ .\ (\exists\ \delta > 0\ .\ (\forall\ y\ .$

( (*y within* $\delta$ *of x*)
$\longrightarrow$
( (*definedAt f y*) $\wedge$ ($\forall$ *u v* . (*f y u* $\wedge$ (*asFunc A*) *y v*) $\longrightarrow$
( *sep2 v u* ) $\leq$ (*sqr* $\varepsilon$) $*$ *sep2 y x* ))) )
)) **by** *blast*
**hence** *rtp3d1*: *definedAt g x* **using** *g* **by** *auto*

**{ fix** *e*
**assume** *epos*: *e* $>$ *0*
**then obtain** *d* **where** *d*: (*d* $>$ *0*) $\wedge$ ($\forall$ *y* .
( (*y within d of x*)
$\longrightarrow$
( (*definedAt f y*) $\wedge$ ($\forall$ *u v* . (*f y u* $\wedge$ (*asFunc A*) *y v*) $\longrightarrow$
( *sep2 v u* ) $\leq$ (*sqr e*) $*$ *sep2 y x* ))) )
**using** *dAx* **by** *auto*
**{ fix** *y*
**assume** *y within d of x*
**hence** (*definedAt f y*) $\wedge$ ($\forall$ *u v* . (*f y u* $\wedge$ (*asFunc A*) *y v*) $\longrightarrow$
( *sep2 v u* ) $\leq$ (*sqr e*) $*$ *sep2 y x* ) **using** *d* **by** *force*
**hence** (*definedAt g y*) $\wedge$ ($\forall$ *u v* . (*f y u* $\wedge$ (*asFunc A*) *y v*) $\longrightarrow$
( *sep2 v u* ) $\leq$ (*sqr e*) $*$ *sep2 y x* ) **using** *g* **by** *force*
**hence** (*definedAt g y*) $\wedge$ ($\forall$ *u v* . (*g y u* $\wedge$ (*asFunc A0*) *y v*) $\longrightarrow$
( *sep2 v u* ) $\leq$ (*sqr e*) $*$ *sep2 y x* )
**using** *g A0 assms*(*1*) *lemBallTranslation* **by** *force*
**}**
**hence** $\exists$ *d*$>$*0*. $\forall$ *y* . (*y within d of x*) $\longrightarrow$
(*definedAt g y*) $\wedge$ ($\forall$ *u v* . (*g y u* $\wedge$ (*asFunc A0*) *y v*) $\longrightarrow$
( *sep2 v u* ) $\leq$ (*sqr e*) $*$ *sep2 y x* )
**using** *d* **by** *force*
**}**
**hence** *rtp3d2*: $\forall$ *e*$>$*0*. $\exists$ *d* $>$ *0*. $\forall$ *y* . (*y within d of x*) $\longrightarrow$
(*definedAt g y*) $\wedge$ ($\forall$ *u v* . (*g y u* $\wedge$ (*asFunc A0*) *y v*) $\longrightarrow$
( *sep2 v u* ) $\leq$ (*sqr e*) $*$ *sep2 y x* )
**by** *auto*
**hence** *rtp3d*: *diffApprox* (*asFunc A0*) *g x* **using** *rtp3d1* **by** *auto*

**hence** *rtp3*: *affineApprox A0 g x* **using** *rtp3a rtp3b rtp3c rtp3d* **by** *auto*

**thus** *?thesis* **using** *g A0 mkT* **by** *best*

**qed**

**lemma** *lemAffineIdentity*:
**assumes** *affine A*
**and** *e* $>$ *0*

**and** $\forall\ y\ .\ (y\ \textit{within}\ e\ \textit{of}\ x) \longrightarrow (A\ y = y)$
**shows** $A = \textit{id}$
**proof** −

**obtain** $T\ L$ **where** *TL*: $\textit{translation}\ T \wedge \textit{linear}\ L \wedge A = T \circ L$ **using** *assms*(*1*) **by** *auto*

**have** $x\ \textit{within}\ e\ \textit{of}\ x$ **using** *assms*(*2*) **by** *auto*
**hence** *xfixed*: $A\ x = x$ **using** *assms*(*3*) **by** *auto*

**{ fix** $p$
**define** $d$ **where** $d$: $d = (p \ominus x)$
**then obtain** $a$ **where** $a$: $(a > \mathit{0}) \wedge (\textit{norm2}\ (a \otimes d) < \textit{sqr}\ e)$
**using** *assms*(*2*) *lemSmallPoints*[*of e d*] **by** *auto*

**define** $p'$ **where** $p'$: $p' = ((a \otimes d) \oplus x)$
**hence** $p'$*fixed*: $A\ p' = p'$ **using** *a assms*(*3*) *lemSep2Symmetry* **by** *auto*

**have** $p'x$: $(p' \ominus x) = (a \otimes (p \ominus x))$ **using** $p'$ $d$ **by** *auto*
**hence** $((\mathit{1}/a) \otimes (p' \ominus x)) = (p \ominus x)$ **using** *a lemScaleAssoc*[*of* $\mathit{1}/a$ *a* $p \ominus x$] **by** *auto*
**hence** $p$: $p = (((\mathit{1}/a) \otimes (p' \ominus x)) \oplus x)$ **by** *auto*

**hence** $L\ p = L\ (((\mathit{1}/a) \otimes (p' \ominus x)) \oplus x)$ **by** *auto*
**also have** ... $= ((L\ ((\mathit{1}/a) \otimes (p' \ominus x))) \oplus (L\ x))$ **using** *TL* **by** *blast*
**also have** ... $= ((L\ x) \oplus (L\ ((\mathit{1}/a) \otimes (p' \ominus x))))$ **using** *add-commute* **by** *simp*
**finally have** $A\ p = ((A\ x) \oplus (L\ ((\mathit{1}/a) \otimes (p' \ominus x))))$
**using** *TL lemTranslationSum* **by** *auto*

**hence** *1*: $A\ p = (x \oplus (L\ ((\mathit{1}/a) \otimes (p' \ominus x))))$ **using** *xfixed* **by** *auto*

**have** $(L\ ((\mathit{1}/a) \otimes (p' \ominus x))) = ((\mathit{1}/a) \otimes (L\ (p' \ominus x)))$ **using** *TL* **by** *blast*
**also have** ... $= ((\mathit{1}/a) \otimes (\ (L\ p') \ominus (L\ x)\ ))$ **using** *TL* **by** *auto*
**also have** ... $= ((\mathit{1}/a) \otimes (\ (A\ p') \ominus (A\ x)\ ))$ **using** *TL* **by** *auto*
**also have** ... $= ((\mathit{1}/a) \otimes (p' \ominus x))$ **using** $p'$*fixed xfixed* **by** *auto*
**finally have** $(L\ ((\mathit{1}/a) \otimes (p' \ominus x))) = (p \ominus x)$ **using** $p$ **by** *auto*

**hence** $A\ p = (x \oplus (p \ominus x))$ **using** *1* **by** *auto*
**hence** $A\ p = p$ **using** *add-diff-eq* **by** *auto*
**}**
**thus** *?thesis* **by** *auto*
**qed**

**end**

**end**

# 21 Sublemma4

This theory shows that functions with affine approximations are continuous where approximated.

**theory** *Sublemma4*
  **imports** *Affine AxTriangleInequality*
**begin**

Our naming of lemmas, propositions, etc., is sometimes counterintuitive. This is because the proof follows a hand-written proof, and we need to maintain the link between the paper-based and Isabelle versions. We will specifically be discussing how we translated from one to the other in a forthcoming paper (under construction). In fact, sublemmas 1 and 2 were eventually found to be unnecessary during construction of the Isabelle proof, and so do not appear in this documentation.

**class** *Sublemma4* = *Affine* + *AxTriangleInequality*
**begin**

**lemma** *sublemma4*:
  **assumes** *affineApprox A f x*
  **shows** $(\exists\, \delta{>}0.\ \forall\, p.\ (p\ \mathit{within}\ \delta\ \mathit{of}\ x) \longrightarrow (\mathit{definedAt}\ f\ p)) \land (\mathit{cts}\ f\ x)$
**proof** −

  **have** *diff*: $(\mathit{definedAt}\ f\ x) \land$
    $(\forall\ \varepsilon > 0\ .\ (\exists\ \delta > 0\ .\ (\forall\ y\ .$
      $(\ (y\ \mathit{within}\ \delta\ \mathit{of}\ x)$
        $\longrightarrow$
        $(\mathit{definedAt}\ f\ y) \land (\ \forall\ u\ v\ .\ (f\ y\ u \land (\mathit{asFunc}\ A)\ y\ v) \longrightarrow$
         $(\ \mathit{sep2}\ v\ u\ ) \le (\mathit{sqr}\ \varepsilon) * \mathit{sep2}\ y\ x\ ))\ \ )$
  $))$ **using** *assms* **by** *simp*

  **have** $0 < 1$ **by** *simp*
  **then obtain** *d* **where** *d*: $(d > 0) \land (\forall\ y\ .$
      $(\ (y\ \mathit{within}\ d\ \mathit{of}\ x)$
        $\longrightarrow$
        $((\mathit{definedAt}\ f\ y) \land (\forall\ u\ v\ .\ (f\ y\ u \land (\mathit{asFunc}\ A)\ y\ v) \longrightarrow$
         $(\ \mathit{sep2}\ v\ u\ \le (\mathit{sqr}\ 1) * \mathit{sep2}\ y\ x\ )))))$ **using** *diff* **by** *blast*
  **hence** $\forall\ p\ .\ (p\ \mathit{within}\ d\ \mathit{of}\ x) \longrightarrow (\mathit{definedAt}\ f\ p)$ **by** *blast*
  **hence** *rtp1*: $\exists\ \delta > 0\ .\ \forall\ p\ .\ (p\ \mathit{within}\ \delta\ \mathit{of}\ x) \longrightarrow (\mathit{definedAt}\ f\ p)$
    **using** *d* **by** *auto*

```
have funcF: isFunction f using assms by simp
have affA: affine A using assms by simp
have funcA: isFunction (asFunc A) using assms by simp

{ fix x′
  assume x′: f x x′
  hence ax: x′ = A x
    using assms lemAffineEqualAtBase[of f A x] by blast

  { fix e
    assume epos: e > 0
    hence e2pos: e/2 > 0 by simp


    obtain d1 where d1: (d1 > 0) ∧ (∀ y .
       ((y within d1 of x) ⟶ ((A y) within (e/2) of (A x))))
      using e2pos affA lemAffineContinuity by blast


    obtain d2′ where d2′: (d2′ > 0) ∧ (∀ y .
    ( (y within d2′ of x)  ⟶ ((definedAt f y) ∧
      ( ∀ fy Ay . (f y fy ∧ (asFunc A) y Ay) ⟶
       ( sep2 Ay fy ) ≤ (sqr (e/2)) ∗ sep2 y x ))))
      using e2pos assms by auto
    then obtain d2
     where d2: (d2 > 0) ∧ (d2 < d2′) ∧ (sqr d2 < d2) ∧ (d2 < 1)
      using lemReducedBound[of d2′] by auto

    define d where d: d = min d1 d2
    have dd1: d ≤ d1 using d by auto
    have dd2: d ≤ d2 using d by auto
    have dpos: d > 0 using d1 d2 d by auto



    { fix y′
      assume y′: y′ ∈ applyToSet f (ball x d)
      then obtain y where y: (y ∈ ball x d) ∧ (f y y′) by auto
       hence y-near-x: y within d of x using lemSep2Symmetry by
auto

       have y within d1 of x using lemBallInBall y-near-x dpos dd1
by auto
      hence dist1: (A y) within (e/2) of (A x) using d1 by auto


        have yd2′x: y within d2′ of x using lemBallInBall y-near-x
dpos d2 dd2 by auto
      hence ∀ fy Ay . (f y fy ∧ (asFunc A) y Ay) ⟶
```

( *sep2 Ay fy* $\leq$ (*sqr* (*e*/*2*)) $*$ *sep2 y x* )
**using** *d2'* **by** *auto*
**hence** *conc2*: *sep2* (*A y*) *y'* $\leq$ (*sqr* (*e*/*2*)) $*$ *sep2 y x* **using** *y* **by** *auto*

**have** *y within d2 of x* **using** *lemBallInBall y-near-x dpos d2 dd2* **by** *auto*
**hence** *yx1*: *y within 1 of x* **using** *lemBallInBall d2* **by** *auto*

**have** *sqr* (*e*/*2*) $>$ *0* **using** *e2pos lemSqrMonoStrict*[*of 0 e*/*2*] **by** *auto*
**hence** (*sqr* (*e*/*2*)) $*$ *sep2 y x* $<$ *sqr* (*e*/*2*)
**using** *mult-strict-left-mono*[*of sep2 y x 1 sqr* (*e*/*2*)]
*lemNorm2NonNeg*[*of y*$\ominus$*x*] *yx1*
**by** *auto*
**hence** *dist2*: *sep2* (*A y*) *y'* $<$ *sqr* (*e*/*2*) **using** *conc2* **by** *auto*

**define** *p* **where** *p*: *p* = (*A x*)
**define** *q* **where** *q*: *q* = (*A y*)
**define** *r* **where** *r*: *r* = *y'*

**have** *tri*: *axTriangleInequality* (*q*$\ominus$*p*) (*r*$\ominus$*q*)
**using** *AxTriangleInequality* **by** *blast*
**have** *Dist1*: *p within* (*e*/*2*) *of q*
**using** *dist1 p q lemSep2Symmetry* **by** *auto*
**have** *Dist2*: *r within* (*e*/*2*) *of q*
**using** *dist2 q r lemSep2Symmetry* **by** *auto*

**have** *r within* ((*e*/*2*)+(*e*/*2*)) *of p*
**using** *e2pos Dist1 Dist2 tri*
*lemDistancesAdd*[*of q p r e*/*2 e*/*2*]
**by** *blast*
**hence** *r within e of p* **using** *lemSumOfTwoHalves* **by** *auto*
**hence** *y'* $\in$ *ball x' e* **using** *p r ax lemSep2Symmetry* **by** *auto*
**}**
**hence** $\exists$ *d*$>$*0*. *applyToSet f* (*ball x d*) $\subseteq$ (*ball x' e*) **using** *dpos* **by** *auto*
**}**
**hence** ($\forall$ *e*$>$*0*. $\exists$ *d*$>$*0*. *applyToSet f* (*ball x d*) $\subseteq$ (*ball x' e*))
**by** *auto*
**}**
**hence** $\forall$ *x'*. (*f x x'*) $\longrightarrow$ ($\forall$ *e*$>$*0*. $\exists$ *d*$>$*0*. *applyToSet f* (*ball x d*) $\subseteq$ (*ball x' e*))
**by** *auto*
**hence** *rtp2*: *cts f x* **by** *simp*

**thus** *?thesis* **using** *rtp1* **by** *auto*
**qed**

**end**

**end**

# 22 MainLemma

This theory establishes conditions under which a function maps tangent lines to tangent lines.

**theory** *MainLemma*
  **imports** *Sublemma3 Sublemma4*
**begin**

**class** *MainLemma* = *Sublemma3* + *Sublemma4*
**begin**

**lemma** *lemMainLemmaBasic*:
  **assumes** *tgt*: *tangentLine l wl origin*
**and** *injf*: *injective f*
**and** *affapp*: *affineApprox A f origin*
**and** *f00*: *f origin origin*
**and** *ctsf'0*: *cts* (*invFunc f*) *origin*
**and** *affline*: *applyAffineToLine A l l'*
**shows** *tangentLine l'* (*applyToSet f wl*) *origin*
**proof** −

  **define** *goal1* **where**
    *goal1*: *goal1* ≡ *origin* ∈ (*applyToSet f wl*)
  **define** *goal2* **where**
    *goal2*: *goal2* ≡ *onLine origin l'*
  **define** *goal3* **where**
    *goal3*: *goal3* ≡ *accPoint origin* (*applyToSet f wl*)
  **define** *subgoal4a* **where**
    *subgoal4a*: *subgoal4a* ≡ (λ *p'* . *onLine p' l'*)
  **define** *subgoal4b* **where**
    *subgoal4b*: *subgoal4b* ≡ (λ *p'* . *p'* ≠ *origin*)
  **define** *subgoal4c1* **where**
    *subgoal4c1*: *subgoal4c1* ≡ (λ *p' d e* .
     (∀ *y'* ∈ (*applyToSet f wl*) . (*y' within d of origin*) ∧ (*y'* ≠ *origin*)

    ⟶ (∃ *r*. (*onLine r* (*lineJoining origin y'*)) ∧ (*r within e of p'*))))
  **define** *subgoal4c* **where**
    *subgoal4c*: *subgoal4c* ≡ (λ *p'* .∀ *e*>*0*. ∃ *d*>*0* . *subgoal4c1 p' d e*)
  **define** *goal4* **where**

*goal4*: *goal4* $\equiv$ ($\exists$ *p′*. (*subgoal4a p′*) $\wedge$ (*subgoal4b p′*) $\wedge$ (*subgoal4c p′*))

**have** *GOAL*: *goal1* $\wedge$ *goal2* $\wedge$ *goal3* $\wedge$ *goal4*
$\longrightarrow$ *tangentLine l′* (*applyToSet f wl*) *origin*
**using** *goal1 goal2 goal3 goal4 subgoal4a subgoal4b subgoal4c1 subgoal4c*
**by** *force*

**have** *affA*: *affine A* **using** *affapp* **by** *auto*
**then obtain** *T L* **where** *TL*: *translation T* $\wedge$ *linear L* $\wedge$ *A*=*T*$\circ$*L* **by** *auto*
**then obtain** *t* **where** *t*: $\forall$ *u*. *T u* = (*u* $\oplus$ *t*) **by** *auto*
**define** *Tinv* **where** *Tinv*: *Tinv* = *mkTranslation* (*origin* $\ominus$ *t*)
**hence** *transTinv*: *translation Tinv* **using** *lemMkTrans* **by** *blast*

**have** *linel*: *isLine l* **using** *tgt* **by** *auto*

**hence** *linel′*: *isLine l′*
**using** *affA affline lemAffineOfLineIsLine*
**by** *auto*

**have** *funcF*: *isFunction f* **using** *affapp* **by** *auto*

**have** *A00*: *A origin* = *origin*
**using** *lemAffineEqualAtBase*[*of f A origin*] *affapp f00*
**by** *auto*

**have** *A origin* = ((*L origin*) $\oplus$ *t*) **using** *TL t* **by** *auto*
**also have** ... = (*origin* $\oplus$ *t*) **using** *TL* **by** *auto*
**finally have** *origin* = *t* **using** *A00* **by** *auto*
**hence** $\forall$ *p*. *T p* = *p* **using** *t* **by** *auto*
**hence** *T* = *id* **by** *auto*
**hence** *A* = *L* **using** *TL* **by** *auto*

**hence** *linA*: *linear A* **using** *TL* **by** *auto*

**have** ((*invFunc f*) *origin origin*)
$\wedge$ ($\forall$ *x* . ((*invFunc f*) *origin x*) $\longrightarrow$ ($\forall$ $\varepsilon$>*0*. $\exists$ $\delta$>*0*.
(*applyToSet* (*invFunc f*) (*ball origin* $\delta$)) $\subseteq$ *ball x* $\varepsilon$))
**using** *f00 ctsf′0* **by** *auto*

**hence** *ctsfinv*: $(\forall\varepsilon>0.\ \exists\delta>0.$
$(applyToSet\ (invFunc\ f)\ (ball\ origin\ \delta)) \subseteq ball\ origin\ \varepsilon)$
**by** *blast*

**have** *ctsA*: $\forall\ x.\ \forall\varepsilon>0.\ \exists\delta>0\ .\ \forall p.$
$(p\ within\ \delta\ of\ x) \longrightarrow ((A\ p)\ within\ \varepsilon\ of\ (A\ x))$
**using** *affA lemAffineContinuity* **by** *auto*

**have** *tgt1*: $origin \in wl$ **using** *tgt* **by** *auto*
**have** *tgt2*: *onLine origin l* **using** *tgt* **by** *auto*
**have** *tgt3*: $\forall\ \varepsilon > 0.\ \exists\ q \in wl.\ (origin \neq q) \wedge (inBall\ q\ \varepsilon\ origin)$
**using** *tgt* **by** *auto*

**have** *sub4*: $(\exists\delta>0.\ \forall p.\ (p\ within\ \delta\ of\ origin)$
$\longrightarrow (definedAt\ f\ p)) \wedge (cts\ f\ origin)$
**using** *affapp sublemma4*[*of f A origin*] **by** *auto*

**hence** *ctsfx*: $(\forall\varepsilon>0.\ \exists\delta>0.\ (applyToSet\ f\ (ball\ origin\ \delta)) \subseteq ball$
$origin\ \varepsilon)$
**using** *f00* **by** *auto*

**obtain** *ddef* **where** *ddef*: $(ddef > 0) \wedge$
$(\forall p.\ (p\ within\ ddef\ of\ origin) \longrightarrow (definedAt\ f$
$p))$
**using** *sub4* **by** *auto*

**have** *rtp1*: *goal1* **using** *tgt1 f00 goal1* **by** *auto*

**have** *l′-from-l*: $l' = applyToSet\ (asFunc\ A)\ l$
**using** *tgt affline lemAffineOfLineIsLine* **by** *auto*
**have** $(asFunc\ A)\ origin\ origin$ **using** *linA* **by** *auto*
**hence** *rtp2*: *goal2* **using** *l′-from-l tgt2 affline goal2* **by** *auto*

**{ fix** *e*
**assume** *epos*: $e > 0$
**then obtain** $dd'$
**where** $dd'$: $(dd' > 0) \wedge ((applyToSet\ f\ (ball\ origin\ dd')) \subseteq ball$

*origin e*)
**using** *ctsfx* **by** *auto*

**define** *dd* **where** *dd*: $dd = min\ dd'\ ddef$
**hence** *ddpos*: $dd > 0$ **using** *dd' ddef* **by** *simp*
**then obtain** *q* **where** *q*: $(q \in wl) \wedge (origin \neq q) \wedge (q\ within\ dd$ *of origin*)
**using** *tgt3* **by** *auto*

**have** $dd \leq ddef$ **using** *dd* **by** *auto*
**hence** *q within ddef of origin*
**using** *ddpos q lemBallInBall*[*of q origin dd ddef*] **by** *auto*
**then obtain** $q'$ **where** $q'$: $(f\ q\ q')$ **using** *ddef* **by** *auto*

**hence** *fact3a*: $q' \in (applyToSet\ f)\ wl$ **using** *q* **by** *auto*

**have** $q \neq origin$ **using** *q* **by** *auto*
**hence** *fact3b*: $q' \neq origin$ **using** *injf* $q'$ *f00* **by** *auto*

**have** $dd \leq dd'$ **using** *dd* **by** *auto*
**hence** $q \in ball\ origin\ dd'$
**using** *q lemBallInBall*[*of q origin dd dd'*] *ddpos* **by** *auto*
**hence** $q' \in ball\ origin\ e$ **using** *dd'* $q'$ **by** *auto*
**hence** *fact3c*: $q'$ *within e of origin* **using** *lemSep2Symmetry* **by** *auto*
**hence** $\exists\ y' \in ((applyToSet\ f)\ wl)\ .\ (origin \neq y') \wedge (y'\ within\ e\ of$ *origin*)
**using** *fact3a fact3b* $q'$ **by** *auto*
**}**
**hence** *rtp3*: *goal3* **using** *goal3* **by** *auto*

**obtain** *P* **where** *P*: $(onLine\ P\ l) \wedge (P \neq origin) \wedge$
$(\forall\ \varepsilon > 0\ .\ \exists\ \delta > 0\ .\ \forall\ y \in wl.\ ($
$(\ (y\ within\ \delta\ of\ origin) \wedge (y \neq origin)\ )$
$\longrightarrow$
$(\ \exists\ r\ .\ ((onLine\ r\ (lineJoining\ origin\ y)) \wedge (r\ within\ \varepsilon\ of\ P)))))$
**using** *tgt* **by** *auto*

**define** *nP* **where** *nP*: $nP = norm\ P$
**have** $P \neq origin$ **using** *P* **by** *auto*
**hence** *nPpos*: $nP > 0$ **using** *P nP lemNotOriginImpliesPositiveNorm*[*of P*]
**by** *auto*
**define** *a* **where** *a*: $a = 1/nP$
**hence** *apos*: $a > 0$ **using** *nPpos* **by** *auto*

**define** $p$ **where** $p$: $p = (a \otimes P)$
**{ assume** $p = origin$
**hence** $(a \otimes P) = origin$ **using** $p$ **by** *auto*
**hence** $(nP \otimes (a \otimes P)) = (nP \otimes origin)$ **by** *simp*
**hence** $P = origin$ **using** *a apos lemScaleAssoc* **by** *auto*
**}**
**hence** *p-not-0*: $p \neq origin$ **using** $P$ **by** *auto*

**define** $p'$ **where** $p'$: $p' = A\ p$
**obtain** $A'$ **where** $A'$: $(affine\ A') \land ((affine\ A') \land (\forall\ p\ q\ .\ A\ p = q \longleftrightarrow A'\ q = p))$
**using** *affapp lemInverseAffine*[*of A*] **by** *auto*
**hence** $A'\ origin = origin \land A'\ p' = p$ **using** *A00* $p'$ **by** *blast*
**hence** *p′-not-0*: $p' \neq origin$ **using** *p-not-0* **by** *auto*

**have** $(onLine\ origin\ l) \land (onLine\ P\ l) \land (origin \neq P)$ **using** *P tgt2* **by** *auto*
**hence** *l-is-0P*: $l = lineJoining\ origin\ P$
**using** *lemLineAndPoints*[*of origin P l*] **by** *auto*

**have** $p = (origin \oplus (a \otimes (P \ominus origin)))$ **using** $p$ **by** *auto*
**hence** *onLine p* (*lineJoining origin P*) **by** *blast*

**hence** *p-on-l*: *onLine p l* **using** *l-is-0P* **by** *auto*
**moreover have** $l' = applyToSet\ (asFunc\ A)\ l \land isLine\ l'$
**using** *lemAffineOfLineIsLine* [*of l A l′*]
*affline*
**by** *auto*
**ultimately have** *p′-on-l′*: *onLine p′ l′* **using** *p-on-l p′* **by** *auto*

**have** $p = (a \otimes P)$ **using** $p$ **by** *auto*
**hence** $norm2\ p = (sqr\ a) * (norm2\ P)$
**using** *lemNorm2OfScaled*[*of a P*] **by** *auto*
**hence** $norm2\ p = (sqr\ a) * (sqr\ nP)$
**using** *nP lemNormSqrIsNorm2*[*of P*] **by** *auto*
**hence** *np1*: $norm2\ p = 1$ **using** *a nPpos apos mult-assoc mult-commute* **by** *auto*

**have** $(onLine\ p\ l) \land (norm2\ p = 1) \land (tangentLine\ l\ wl\ origin)$
**using** *p-on-l np1 tgt* **by** *auto*
**hence** *sub3*: $\forall\ \varepsilon > 0\ .\ \exists\ \delta > 0\ .\ \forall\ y\ ny\ .\ ($
$((y\ within\ \delta\ of\ origin) \land (y \neq origin) \land (y \in wl) \land (norm\ y = ny))$
$\longrightarrow$
$(\ (((1/ny) \otimes y)\ within\ \varepsilon\ of\ p) \lor (((-1/ny) \otimes y)\ within\ \varepsilon\ of\ p)))$

**using** *sublemma3*[*of l p wl*]
**by** *auto*

{ **fix** *e*
**assume** *epos*: $e > 0$
**define** *e1* **where** *e1*: $e1 = nP * e$
**hence** *e1pos*: $e1 > 0$ **using** *nPpos epos* **by** *auto*
**define** *e2* **where** *e2*: $e2 = e/2$
**hence** *e2pos*: $e2 > 0$ **using** *epos* **by** *auto*

**obtain** *dctsA0* **where** ($dctsA0 > 0$) ∧ (∀ *q*.
(*q within dctsA0 of origin*) ⟶ ((*A q*) *within e2 of* (*A origin*)))
**using** *ctsA e2pos A00* **by** *blast*
**hence** *dctsA0*: ($dctsA0 > 0$) ∧ (∀ *q*.
(*q within dctsA0 of origin*) ⟶ ((*A q*) *within e2 of origin*))
**using** *A00* **by** *auto*

**obtain** *dctsAp* **where** *dctsAp*: ($dctsAp > 0$) ∧ (∀ *q*.
(*q within dctsAp of p*) ⟶ ((*A q*) *within e2 of* (*A p*)))
**using** *ctsA e2pos* **by** *blast*

**obtain** *dsub* **where** *dsub*: ($dsub > 0$) ∧ (∀ *y ny* .
((*y within dsub of origin*) ∧ (*y* ≠ *origin*) ∧ (*y* ∈ *wl*) ∧ (*norm y*
= *ny*))
⟶
(((*1*/*ny*)⊗*y*) *within* (*dctsAp*) *of p*)
∨ (((−*1*/*ny*)⊗*y*) *within* (*dctsAp*) *of p*))
**using** *apos dctsAp sub3* **by** *blast*

**obtain** *daff* **where** *daff*: ($daff > 0$) ∧ (∀ *y* .
( (*y within daff of origin*)
⟶
( (*definedAt f y*) ∧ (∀ *fy Ay* . (*f y fy* ∧ (*asFunc A*) *y Ay*) ⟶
( *sep2 Ay fy* ) ≤ (*sqr e2*) ∗ *sep2 y origin* ))) )
**using** *e2pos affapp* **by** *auto*

**define** *dmin* **where** *dmin*: *dmin* = *min dsub daff*
**hence** *dminsub*: *dmin* ≤ *dsub* **by** *auto*
**have** *dminaff*: *dmin* ≤ *daff* **using** *dmin* **by** *auto*
**have** *dminpos*: $dmin > 0$ **using** *dmin dsub daff* **by** *auto*

**obtain** *dfinv*
**where** *dfinv*: ($dfinv > 0$)
∧ ((*applyToSet* (*invFunc f*) (*ball origin dfinv*))

$\subseteq$ *ball origin dmin*)
**using** *ctsfinv dminpos* **by** *auto*

**{ fix** $y'$
**assume** $y'$: ($y' \in$ (*applyToSet f wl*)) $\wedge$ ($y' \neq$ *origin*)
**then obtain** *y* **where** *y*: (*f y* $y'$) $\wedge$ ($y \in$ *wl*) **by** *auto*

**have** *y-not-0*: $y \neq$ *origin* **using** *y* $y'$ *f00 funcF* **by** *auto*
**obtain** *ny* **where** *ny*: *norm y* = *ny* **by** *auto*

**hence** *nypos*: *ny* >*0*
**using** *y-not-0 lemNotOriginImpliesPositiveNorm*[*of y*] *ny* **by** *auto*

**define** *p1* **where** *p1*: *p1* = ((*1*/*ny*)$\otimes y'$)
**define** *q1* **where** *q1*: *q1* = (*A* ((*1*/*ny*)$\otimes$*y*))
**define** *p2* **where** *p2*: *p2* = (($-$*1*/*ny*)$\otimes y'$)
**define** *q2* **where** *q2*: *q2* = (*A* (($-$*1*/*ny*)$\otimes$*y*))
**define** *r* **where** *r*: *r* = (*A p*)

**assume** $y'2$: ($y'$ *within dfinv of origin*)
**hence** $y' \in$ *ball origin dfinv* **using** *lemSep2Symmetry* **by** *auto*
**hence** $y \in$ *applyToSet* (*invFunc f*) (*ball origin dfinv*) **using** *y* **by** *auto*
**hence** *ydmin*: $y \in$ *ball origin dmin* **using** *dfinv* **by** *auto*

**have** *dmin* $\leq$ *dsub* **using** *dmin* **by** *auto*
**hence** *ydsub*: *y within dsub of origin*
**using** *lemBallInBall*[*of y origin dmin dsub*] *dminpos ydmin*
**by** *auto*

**hence** (*y within dsub of origin*) $\wedge$ ($y \neq$ *origin*)
$\wedge$ ($y \in$ *wl*) $\wedge$ (*norm y* = *ny*)
**using** *ydsub y-not-0 y ny* **by** *force*
**hence** *cases*: (((*1*/*ny*)$\otimes$*y*) *within dctsAp of p*)
$\vee$ ((($-$*1*/*ny*)$\otimes$*y*) *within dctsAp of p*)
**using** *dsub* **by** *blast*

**hence** *casesA*: (*q1 within e2 of r*) $\vee$ (*q2 within e2 of r*)
**using** *dctsAp q1 q2 r* **by** *auto*

**have** *dmin* $\leq$ *daff* **using** *dmin* **by** *auto*
**hence** *y within daff of origin*
**using** *lemBallInBall*[*of y origin dmin daff*] *dminpos ydmin*
**by** *auto*

**hence** ($definedAt\ f\ y$) $\wedge$ ($\forall$ $fy$ $Ay$ . ($f\ y\ fy$ $\wedge$ ($asFunc\ A$) $y$ $Ay$) $\longrightarrow$
( $sep2\ Ay\ fy$ ) $\leq$ ($sqr\ e2$) $*$ $sep2\ y\ origin$)
**using** *daff* **by** *auto*
**hence** $sep2$ ($A$ $y$) $y' \leq$ ($sqr\ ny$) $*$ ($sqr\ e2$)
**using** *y ny lemNormSqrIsNorm2 mult-commute* **by** *auto*
**hence** $sep2$ ($A$ $y$) $y' \leq sqr$ ($ny{*}e2$)
**using** *lemSqrMult*[*of ny e2*] **by** *auto*
**hence** $sep2$ (($1/ny$)$\otimes$($A$ $y$)) (($1/ny$)$\otimes y'$) $\leq$ $sqr\ e2$
**using** *nypos*
*lemScaleBallAndBoundary*[*of A y* $y'$ *ny*$*$*e2 1*/*ny*]
**by** *auto*
**hence** *part1*: $sep2$ ($A$ (($1/ny$)$\otimes y$)) (($1/ny$)$\otimes y'$) $\leq$ $sqr\ e2$
**using** *linA lemLinearProps*[*of A 1*/*ny y*] **by** *auto*

**{**
**assume** *case1*: *q1 within e2 of r*

**have** *pq*: $sep2\ p1\ q1 \leq sqr\ e2$
**using** *part1 lemSep2Symmetry*[*of p1 q1*] *p1 q1* **by** *auto*
**hence** *rq*: $sep2\ r\ q1 < sqr\ e2$ **using** *case1 lemSep2Symmetry r q1* **by** *auto*

**{ define** *pp* **where** *pp*: $pp = (q1 \ominus p1)$
**define** *qq* **where** *qq*: $qq = (r \ominus q1)$
**have** *tri1*: *axTriangleInequality pp qq* **using** *AxTriangleInequality* **by** *simp*
**hence** *r within* ($e2$ + $e2$) *of p1*
**using** *pp qq pq rq e2pos lemDistancesAddStrictR*[*of q1 p1 r*]
**by** *blast*
**}**
**hence** *done1*: *p1 within e of r* **using** *lemSep2Symmetry lemSumOfTwoHalves e2* **by** *auto*

**have** $p1$ = ($origin \oplus$ (($1/ny$)$\otimes$($y' \ominus origin$))) **using** *p1* **by** *auto*
**hence** *onLine p1* (*lineJoining origin* $y'$) **by** *fastforce*
**hence** *onLine p1* (*lineJoining origin* $y'$) $\wedge$ (*p1 within e of* $p'$)
**using** $p'$ *r done1* **by** *blast*
**}**
**hence** *case1*: (*q1 within e2 of r*)
$\longrightarrow$ (*onLine p1* (*lineJoining origin* $y'$) $\wedge$ (*p1 within e of* $p'$))
**by** *blast*

**{**
**assume** *case2*: *q2 within e2 of r*

**have** $p2$ = ((($-1$)$*$($1/ny$))$\otimes y'$) **using** *p2* **by** *auto*

**hence** $p2'$: $p2 = ((-1)\otimes p1)$ **using** *lemScaleAssoc*[*of* $-1$ $1/ny$ $y'$] *p1* **by** *auto*
**have** $q2 = (A\ (((-1)*(1/ny))\otimes y))$ **using** *q2* **by** *auto*
**hence** $q2q1$: $q2 = ((-1)\otimes q1)$
**using** *linA lemLinearProps*[*of A* $-1$ $((1/ny)\otimes y)$] *q1*
**by** *auto*
**hence** *sep2 p2 q2* = *sep2 p1 q1* **using** *lemScaleSep2*[*of* $-1$] $p2'$ **by** *auto*
**hence** *pq*: *sep2 p2 q2* $\leq$ *sqr e2*
**using** *part1 lemSep2Symmetry*[*of p1 q1*] *p1 q1* **by** *auto*

**hence** *rq*: *sep2 r q2* $<$ *sqr e2* **using** *case2 lemSep2Symmetry r q2* **by** *auto*

{ **define** *pp* **where** *pp*: $pp = (q2\ominus p2)$
**define** *qq* **where** *qq*: $qq = (r\ominus q2)$
**have** *tri2*: *axTriangleInequality pp qq* **using** *AxTriangleInequality* **by** *simp*
**hence** *r within* $(e2 + e2)$ *of p2*
**using** *pp qq pq rq e2pos lemDistancesAddStrictR*[*of q2 p2 r*]
**by** *blast*
}
**hence** *p2 within e of r* **using** *lemSep2Symmetry lemSumOfTwoHalves e2* **by** *auto*
**hence** *done2*: *p2 within e of* $p'$ **using** *r* $p'$ **by** *simp*

**have** $p2 = (origin \oplus ((-1/ny)\otimes(y'\ominus origin)))$ **using** *p2* **by** *auto*
**hence** *onLine p2* (*lineJoining origin* $y'$) **by** *blast*
**hence** *onLine p2* (*lineJoining origin* $y'$) $\wedge$ (*p2 within e of* $p'$)
**using** $p'$ *done2* **by** *blast*
}
**hence** *case2*: (*q2 within e2 of r*)
$\longrightarrow$ (*onLine p2* (*lineJoining origin* $y'$) $\wedge$ (*p2 within e of* $p'$))
**by** *blast*

**hence** $\exists\, r$. (*onLine r* (*lineJoining origin* $y'$)) $\wedge$ (*r within e of* $p'$)
**using** *casesA case1 case2* **by** *blast*

**hence** ( ($y' \in$ *applyToSet f wl*) $\wedge$ ($y'$ *within dfinv of origin*) $\wedge$ ($y' \neq$ *origin*) )
$\longrightarrow$ ($\exists\, r$. (*onLine r* (*lineJoining origin* $y'$)) $\wedge$ (*r within e of* $p'$))
**using** *dfinv* **by** *blast*

}
**hence** *subgoal4c1* $p'$ *dfinv e* **using** *dfinv subgoal4c1* **by** *blast*
**hence** $\exists\, d>0$ . *subgoal4c1* $p'$ *d e* **using** *dfinv* **by** *auto*
}
**hence** $\forall\, e>0$ . $\exists\, d>0$ . *subgoal4c1* $p'$ *d e* **by** *auto*

```
  hence subgoal4c p′ using subgoal4c subgoal4c1 by force

  hence (subgoal4a p′) ∧ (subgoal4b p′) ∧ (subgoal4c p′)
    using p′-not-0 p′-on-l′ subgoal4a subgoal4b by auto

 hence rtp4: goal4 using goal4 subgoal4a subgoal4b subgoal4c by blast


  thus ?thesis using rtp1 rtp2 rtp3 GOAL by fastforce
qed




lemma lemMainLemmaOrigin:
  assumes tgtx:     tangentLine l wl x
and       injf:     injective f
and       affappx:  affineApprox A f x
and       fx0:      f x origin
and       ctsf′0:   cts (invFunc f) origin
and       affline:  applyAffineToLine A l l′
shows      tangentLine l′ (applyToSet f wl) origin
proof −

  define T where T: T = mkTranslation x
  hence transT: translation T using lemMkTrans by blast
  define T′ where T′: T′ = mkTranslation (origin ⊖ x)
  hence transT′: translation T′ using lemMkTrans by blast

  have TT′: ∀ p q . T p = q ⟷ T′ q = p using T T′ by auto


  define g where g: g = composeRel f (asFunc T)

  define l0 where l0: l0 = applyToSet (asFunc T′) l
  define wl0 where wl0: wl0 = applyToSet (asFunc T′) wl
  define A0 where A0: A0 = A ∘ T


  have T′ x = origin using T′ by auto
  hence rtp1: tangentLine l0 wl0 origin
    using l0 wl0 transT′ tgtx lemTangentLineTranslation[of T′ x wl l]
    by auto


  have rtp2: injective g
     using transT lemTranslationInjective[of T] lemInjOfInjIsInj[of
asFunc T f]
```

*injf g*
**by** *blast*

**have** $T'$ *x* = *origin* **using** $T'$ **by** *auto*
**hence** *rtp3*: *affineApprox A0 g origin*
**using** *transT TT'*
*lemAffineApproxDomainTranslation*[*of T f A x T'*]
*affappx g A0*
**by** *auto*

**have** (*T origin* = *x*) ∧ (*f x origin*) **using** *T fx0* **by** *auto*
**hence** ∃ *x* . ((*asFunc T*) *origin x*) ∧ (*f x origin*) **by** *auto*
**hence** *rtp4*: *g origin origin* **using** *g T fx0* **by** *auto*

**define** *h* **where** *h*: *h* = (*invFunc* (*asFunc T*))
**hence** *invcomp*: *composeRel h* (*invFunc f*) = *invFunc g*
**using** *lemInverseComposition*[*of g asFunc T f*] *g* **by** *auto*

**{ fix** *p r*
**have** *inv1*: *invFunc* (*asFunc T*) *p r* ⟷ ($T'$∘*T*) *r* = $T'$ *p*
**using** *transT' lemTranslationInjective* **by** *auto*
**hence***invFunc* (*asFunc T*) *p r* ⟷ *r* = $T'$ *p*
**using** *T T' lemInverseTranslation*[*of T x T'*] **by** *auto*
**}**
**hence** *hT*: *h* = *asFunc* $T'$ **using** *h* **by** *force*
**hence** ∀ *y*. *cts h y*
**using** *transT' lemTranslationImpliesAffine*[*of T'*]
*lemAffineIsCts*[*of T'*]
**by** *blast*
**hence** *ctsh*: ∀ *y*. (*invFunc f*) *origin y* ⟶ *cts h y* **by** *auto*

**define** $g'$ **where** $g'$: $g'$ = *composeRel h* (*invFunc f*)
**hence** *invg*: $g'$ = *invFunc g* **using** *hT invcomp* **by** *simp*
**have** *cts* $g'$ *origin*
**using** *ctsf'0 ctsh lemCtsOfCtsIsCts*[*of invFunc f origin h*] $g'$
**by** *auto*
**hence** *rtp5*: *cts* (*invFunc g*) *origin* **using** *invg* **by** *auto*

**have** *affA*: *affine A* **using** *assms*(*3*) **by** *auto*
**hence** *rtp3b*: *affine A0*
**using** *lemAffOfAffIsAff*[*of T A*] *lemTranslationImpliesAffine*[*of T*]
*A0 affA transT*

**by** *auto*
**define** *l0′* **where** *l0′*: $l0' = applyToSet\ (asFunc\ A0)\ l0$
**hence** *rtp6*: *applyAffineToLine A0 l0 l0′*
**using** *rtp1 rtp3b lemAffineOfLineIsLine*[*of l0 A0 l0′*]
**by** *auto*

**have** (*tangentLine l0 wl0 origin*) $\longrightarrow$ (
(*injective g*) $\longrightarrow$
(*affineApprox A0 g origin*) $\longrightarrow$
(*g origin origin*) $\longrightarrow$
((*cts* (*invFunc g*) *origin*) $\longrightarrow$
((*applyAffineToLine A0 l0 l0′*) $\longrightarrow$
(*tangentLine l0′* (*applyToSet g wl0*) *origin*))))
**using** *lemMainLemmaBasic*[*of wl0 l0 g A0 l0′*]
**by** *blast*

**hence** *basic*: (*tangentLine l0′* (*applyToSet g wl0*) *origin*)
**using** *rtp1 rtp2 rtp3 rtp4 rtp5 rtp6* **by** *meson*

**obtain** $A'$ **where** $A'$: $\forall\ p\ q\ .\ A\ p = q \longleftrightarrow A'\ q = p$
**using** *affappx* **by** *metis*

**have** $ToT'$: $T \circ T' = id$ **using** $TT'$ **by** *auto*
**have** $A0 \circ T' = (A \circ T) \circ T'$ **using** *A0* **by** *auto*
**also have** ... $= A \circ (T \circ T')$ **by** *auto*
**finally have** $A0T'$: $A0 \circ T' = A$ **using** $ToT'$ **by** *auto*

**have** $l0' = applyToSet\ (asFunc\ (A0 \circ T'))\ l$ **using** *l0 l0′* **by** *auto*
**hence** $l0' = applyToSet\ (asFunc\ A)\ l$ **using** $A0T'$ **by** *auto*
**hence** *l0′l*: $l0' = l'$ **using** *tgtx affline lemAffineOfLineIsLine*[*of l A l′*] **by** *auto*

**have** $applyToSet\ g\ wl0 = applyToSet\ (composeRel\ f\ (asFunc\ (T \circ T')))\ wl$ **using** *wl0 g* **by** *auto*
**also have** ... $= applyToSet\ (composeRel\ f\ (asFunc\ id))\ wl$ **using** $ToT'$ **by** *auto*
**also have** ... $= applyToSet\ f\ wl$ **by** *auto*
**finally have** $applyToSet\ g\ wl0 = applyToSet\ f\ wl$ **by** *auto*

**hence** *tangentLine l′* (*applyToSet f wl*) *origin* **using** *basic l0′l* **by** *auto*
**thus** *?thesis* **by** *auto*

**qed**

**lemma** *lemMainLemma*:
**assumes** *tgtx*: *tangentLine l wl x*
**and** *injf*: *injective f*
**and** *affappx*: *affineApprox A f x*
**and** *fxy*: *f x y*
**and** *ctsf'y*: *cts (invFunc f) y*
**and** *affline*: *applyAffineToLine A l l'*
**shows** *tangentLine l' (applyToSet f wl) y*
**proof** −
**define** *Ty* **where** *Ty*: *Ty = mkTranslation y*
**hence** *transTy*: *translation Ty* **using** *lemMkTrans* **by** *auto*
**define** *Ty'* **where** *Ty'*: *Ty' = mkTranslation (origin ⊖ y)*
**hence** *transTy'*: *translation Ty'* **using** *lemMkTrans* **by** *blast*
**define** *g* **where** *g*: *g = composeRel (asFunc Ty') f*
**define** *Ay* **where** *Ay*: *Ay = Ty' ∘ A*

**define** *ly'* **where** *ly'*: *ly' = applyToSet (asFunc Ty') l'*

**have** *lineL*: *isLine l* **using** *tgtx* **by** *auto*
**have** *affA*: *affine A* **using** *affappx* **by** *auto*

**have** *TT'*: *∀ p q . Ty p = q ⟷ Ty' q = p* **using** *Ty Ty'* **by** *auto*

**have** *rtp1*: *tangentLine l wl x* **by** (*rule tgtx*)

**have** *rtp2*: *injective g*
**using** *transTy' lemTranslationInjective[of Ty'] lemInjOfInjIsInj[of f asFunc Ty']*
*injf g*
**by** *blast*

**have** (*translation Ty'*) ⟶ (*affineApprox A f x*)
⟶ (*affineApprox (Ty' ∘ A) (composeRel (asFunc Ty') f) x*)
**using** *lemAffineApproxRangeTranslation[of Ty' f A x]*
**by** *blast*
**hence** *rtp3*: *affineApprox Ay g x* **using** *Ay g transTy' affappx* **by** *meson*

**have** *rtp4*: *g x origin* **using** *g Ty' fxy* **by** *auto*

**define** $f'$ **where** $f'$: $f' = invFunc\ f$
**define** $h$ **where** $h$: $h = (invFunc\ (asFunc\ Ty'))$
**define** $g'$ **where** $g'$: $g' = invFunc\ g$

**hence** *invcomp*: $g' = composeRel\ f'\ h$
  **using** *lemInverseComposition* $g\ f'\ h$ **by** *auto*

**{ fix** $p\ r$
  **have** *inv1*: $invFunc\ (asFunc\ Ty')\ p\ r \longleftrightarrow (Ty \circ Ty')\ r = Ty\ p$
    **using** *transTy lemTranslationInjective* **by** *auto*
  **hence** $invFunc\ (asFunc\ Ty')\ p\ r\ \longleftrightarrow r = Ty\ p$ **using** $Ty\ Ty'$ **by** *auto*
**}**
**hence** *hT*: $h = asFunc\ Ty$ **using** $h$ **by** *force*

**hence** *ctsh0*: *cts h origin*
  **using** *transTy lemTranslationImpliesAffine*[*of Ty*]
    *lemAffineIsCts*[*of Ty*]
  **by** *blast*

**{ fix** $p$
  **assume** *h origin p*
  **hence** (*asFunc Ty*) *origin p* **using** *hT* **by** *auto*
  **hence** $p = y$ **using** *Ty* **by** *auto*
  **hence** *cts* (*invFunc f*) *p* **using** *ctsf'y* **by** *auto*
**}**
**hence** *ctsf*: $\forall p.\ h\ origin\ p \longrightarrow cts\ f'\ p$ **using** $f'$ **by** *auto*

**have** *cts* $g'$ *origin*
  **using** *invcomp ctsh0 ctsf lemCtsOfCtsIsCts*[*of h origin f'*]
  **by** *blast*

**hence** *rtp5*: *cts* (*invFunc g*) *origin* **using** $g'$ **by** *auto*

**have** *affAy*: *affine Ay*
  **using** *affA lemTranslationImpliesAffine*[*of Ty'*] *transTy'*
    *lemAffOfAffIsAff* [*of A Ty'*] *Ay*
  **by** *auto*
**have** $l' = applyToSet\ (asFunc\ A)\ l$
  **using** *affline lineL affA lemAffineOfLineIsLine*[*of l A l'*] **by** *auto*
**hence** $ly' = applyToSet\ (asFunc\ Ay)\ l$ **using** $ly'$ *Ay* **by** *auto*
**hence** *rtp6*: *applyAffineToLine Ay l ly'*
  **using** *lineL affAy lemAffineOfLineIsLine*[*of l Ay ly'*]
  **by** *auto*

```
have (tangentLine l wl x) ⟶
      (injective g) ⟶
      (affineApprox Ay g x) ⟶
      (g x origin) ⟶
      (cts (invFunc g) origin) ⟶
      (applyAffineToLine Ay l ly') ⟶
      (tangentLine ly' (applyToSet g wl) origin)
  using lemMainLemmaOrigin[of x wl l g Ay ly']
  by fastforce

hence tgt': tangentLine ly' (applyToSet g wl) origin
  using rtp1 rtp2 rtp3 rtp4 rtp5 rtp6 by meson

define wl' where wl': wl' = (applyToSet g wl)
define Term1 where Term1: Term1 = applyToSet (asFunc Ty) ly'
define Term2 where Term2: Term2 = applyToSet (asFunc Ty) wl'
define Term3 where Term3: Term3 = Ty origin

have tangentLine ly' wl' origin using tgt' wl' by auto

hence goal: tangentLine (applyToSet (asFunc Ty) ly')
                  (applyToSet (asFunc Ty) wl')
                  (Ty origin)
  using transTy lemTangentLineTranslation[of Ty origin wl' ly']
  by fastforce
hence goal: tangentLine Term1 Term2 Term3
  using Term1 Term2 Term3 by auto

have ToT': Ty ∘ Ty' = id using TT' by auto
have Term1 = applyToSet (asFunc Ty) (applyToSet (asFunc Ty')
l')
  using ly' Term1 by auto
also have ... = applyToSet (asFunc (Ty ∘ Ty')) l' by auto
also have ... = applyToSet (asFunc id) l' using ToT' by auto
finally have term1: Term1 = l' by auto

have composeRel (asFunc Ty) g = composeRel (asFunc Ty) (composeRel
(asFunc Ty') f)
  using g by auto
also have ... = composeRel (asFunc (Ty∘Ty')) f by auto
also have ... = composeRel (asFunc id) f using ToT' by auto
finally have Tog: composeRel (asFunc Ty) g = f by auto

have Term2 = applyToSet (asFunc Ty) (applyToSet g wl)
  using wl' Term2 by auto
also have ... = applyToSet (composeRel (asFunc Ty) g) wl by auto
finally have term2: Term2 = applyToSet f wl using Tog by auto

have term3: Term3 = y using Ty Term3 by auto
```

**thus** *?thesis* **using** *goal term1 term2 term3*
  **by** *fastforce*

**qed**

**end**

**end**

# 23 AXIOM: AxDiff

This theory declares the axiom AxDiff.

**theory** *AxDiff*
  **imports** *Affine WorldView*
**begin**

AxDiff: Worldview transformations are differentiable wherever they are defined - they can be approximated locally by affine transformations.

**class** *axDiff* = *Affine* + *WorldView*
**begin**
  **abbreviation** *axDiff* :: *Body* ⇒ *Body* ⇒ *'a Point* ⇒ *bool*
    **where** *axDiff m k p* ≡ (*definedAt* (*wvtFunc m k*) *p*)
      ⟶ (∃ *A* . (*affineApprox A* (*wvtFunc m k*) *p* ))
**end**

**class** *AxDiff* = *axDiff* +
  **assumes** *AxDiff*: ∀ *m k p* . *axDiff m k p*
**begin**
**end**

**end**

# 24 TangentLineLemma

This theory shows that affine approximations map tangent lines to tangent lines.

**theory** *TangentLineLemma*
  **imports** *MainLemma AxDiff Cones*
**begin**

**class** *TangentLineLemma* = *MainLemma* + *AxDiff* + *Cones*

**begin**

**lemma** *lemWVTImpliesFunction*: *isFunction* (*wvtFunc k h*)
**proof** −
  { **fix** *x p q*
    **assume** *hyp*: *wvtFunc k h x p* ∧ *wvtFunc k h x q*

    **have** *axDiff k h x* **using** *AxDiff* **by** *blast*
    **hence** *axdiff*: (∃ *r* . *wvtFunc k h x r*)
                    ⟶ (∃ *A* . (*affineApprox A* (*wvtFunc k h*) *x* ))
      **by** *auto*

    **then obtain** *A* **where** *A*: *affineApprox A* (*wvtFunc k h*) *x* **using** *hyp* **by** *auto*
    **hence** ∀ *z*. (*wvtFunc k h x z*) ⟷ (*z* = *A x*)
      **using** *lemAffineEqualAtBase*[*of wvtFunc k h A x*]
      **by** *auto*
    **hence** *p* = *A x* ∧ *q* = *A x* **using** *hyp* **by** *blast*
    **moreover have** *affine A* **using** *A* **by** *auto*
    **ultimately have** *p* = *q* **by** *auto*
  }
  **thus** *?thesis* **by** *force*
**qed**

**lemma** *lemWVTCts*:
  **assumes** *definedAt* (*wvtFunc h k*) *p*
  **shows** *cts* (*wvtFunc h k*) *p*
**proof** −
  **have** *axDiff h k p* **using** *AxDiff* **by** *blast*
  **hence** *axdiff*: (∃ *r* . *wvtFunc h k p r*) ⟶ (∃ *A* . (*affineApprox A* (*wvtFunc h k*) *p* ))
    **by** *auto*
  **then obtain** *A* **where** *A*: *affineApprox A* (*wvtFunc h k*) *p* **using** *assms* **by** *auto*
  **thus** *?thesis* **using** *sublemma4*[*of wvtFunc h k A p*] **by** *auto*
**qed**

**lemma** *lemWVTInverse*: *invFunc* (*wvtFunc k h*) = *wvtFunc h k*
  **by** *force*

**lemma** *lemWVTInverseCts*:
  **assumes** *wvtFunc k h p q*
  **shows** *cts* (*wvtFunc h k*) *q*
**proof** −

**define** *whk* **where** *whk*: $whk = wvtFunc\ h\ k$
**have** *definedAt whk q* $\longrightarrow$ *cts whk q*
**using** *lemWVTCts*[*of h k q*] *whk* **by** *fast*
**moreover have** *definedAt whk q* **using** *whk assms* **by** *auto*
**ultimately have** *cts whk q* **by** *auto*
**thus** *?thesis* **using** *whk* **by** *auto*
**qed**

**lemma** *lemWVTInjective*: *injective* (*wvtFunc k h*)
**proof** –
**define** *wkh* **where** *wkh*: $wkh = wvtFunc\ k\ h$
**define** *inv* **where** *inv*: $inv = invFunc\ wkh$
**define** *inv2* **where** *inv2*: $inv2 = invFunc\ inv$
**define** *whk* **where** *whk*: $whk = wvtFunc\ h\ k$

**have** *1*: $inv = whk$ **using** *inv whk wkh* **by** *force*
**have** *2*: $inv2 = wkh$ **using** *inv2 inv wkh* **by** *force*

**have***isFunction whk* **using** *lemWVTImpliesFunction whk* **by** *auto*
**hence** *isFunction inv* **using** *1* **by** *auto*
**hence** *injective inv2* **using** *inv2* **by** *auto*
**hence** *injective wkh* **using** *2* **by** *auto*
**thus** *?thesis* **using** *wkh* **by** *auto*
**qed**

**lemma** *lemPresentation*:
**assumes** $x \in wline\ m\ b$
**and** *tangentLine l* (*wline m b*) *x*
**and** *affineApprox A* (*wvtFunc m k*) *x*
**and** *wvtFunc m k x y*
**and** *applyAffineToLine A l l'*
**shows** *tangentLine l'* (*wline k b*) *y*
**proof** –

**have** *main*: (*tangentLine l* (*wline m b*) *x*) $\longrightarrow$
(*injective* (*wvtFunc m k*)) $\longrightarrow$
(*affineApprox A* (*wvtFunc m k*) *x*) $\longrightarrow$
((*wvtFunc m k*) *x y*) $\longrightarrow$
(*cts* (*invFunc* (*wvtFunc m k*)) *y*) $\longrightarrow$
(*applyAffineToLine A l l'*) $\longrightarrow$
(*tangentLine l'* (*applyToSet* (*wvtFunc m k*) (*wline m b*)) *y*)
**using** *lemMainLemma*[*of x wline m b l wvtFunc m k A y l'*]
**by** *blast*

**have** *1*: (*tangentLine l* (*wline m b*) *x*) **using** *assms*(*2*) **by** *auto*
**have** *2*: *injective* (*wvtFunc m k*) **using** *lemWVTInjective* **by** *auto*

**have** *3*: *affineApprox A* (*wvtFunc m k*) *x* **using** *assms*(*3*) **by** *auto*
**have** *4*: (*wvtFunc m k*) *x y* **using** *assms*(*4*) **by** *auto*

**have** *invFunc* (*wvtFunc m k*) = *wvtFunc k m* **using** *lemWVTInverse* **by** *auto*
**moreover have** *cts* (*wvtFunc k m*) *y*
**using** *assms*(*4*) *lemWVTInverseCts*[*of y m k x*] **by** *auto*
**ultimately have** *5*: *cts* (*invFunc* (*wvtFunc m k*)) *y* **by** *force*

**have** *6*: *applyAffineToLine A l l′* **using** *assms*(*5*) **by** *auto*

**hence** *tgt*: *tangentLine l′* (*applyToSet* (*wvtFunc m k*) (*wline m b*)) *y*
**using** *main 1 2 3 4 5* **by** *meson*

**have** *axdiff*: *axDiff k m y* **using** *AxDiff* **by** *blast*
**hence** ($\exists$ *r* . *wvtFunc k m y r*)
$\longrightarrow$ ($\exists$ *A′* . (*affineApprox A′* (*wvtFunc k m*) *y* )) **by** *blast*
**then obtain** *A′* **where** *A′*: *affineApprox A′* (*wvtFunc k m*) *y* **using** *assms*(*4*) **by** *auto*
**hence** ($\exists\delta$>*0*. $\forall$ *p*. (*p within* $\delta$ *of y*) $\longrightarrow$ (*definedAt* (*wvtFunc k m*) *p*))
**using** *sublemma4*[*of wvtFunc k m A′ y*] **by** *auto*
**then obtain** *d* **where** *d*: (*d* > *0*) $\wedge$ ($\forall$ *p*.
(*p within d of y*) $\longrightarrow$ (*definedAt* (*wvtFunc k m*) *p*))
**by** *auto*
**hence** *dpos*: *d* > *0* **by** *auto*

**define** *Ball* **where** *Ball*: *Ball* = *ball y d*

**have** *l2r*: (*applyToSet* (*wvtFunc m k*) (*wline m b*)) $\cap$ *Ball* $\subseteq$ (*wline k b*) $\cap$ *Ball*
**using** *Ball* **by** *auto*

**{ fix** *q*
**assume** *q*: *q* $\in$ (*wline k b*) $\cap$ *Ball*
**hence** *q within d of y* **using** *Ball lemSep2Symmetry* **by** *auto*
**hence** *definedAt* (*wvtFunc k m*) *q* **using** *d* **by** *auto*
**hence** *qset*: *q* $\in$ *applyToSet* (*wvtFunc m k*) (*wvt k m q*) **by** *auto*

**have** *wvt k m q* $\subseteq$ *applyToSet* (*wvtFunc k m*) (*wline k b*) **using** *q* **by** *auto*
**hence** *wvt k m q* $\subseteq$ *wline m b* **by** *auto*
**hence** *applyToSet* (*wvtFunc m k*) (*wvt k m q*)

$\subseteq$ *applyToSet* (*wvtFunc m k*) (*wline m b*) **by** *auto*
**hence** $q \in$ *applyToSet* (*wvtFunc m k*) (*wline m b*) **using** *qset* **by** *auto*
**hence** $q \in$ (*applyToSet* (*wvtFunc m k*) (*wline m b*)) $\cap$ *Ball* **using** *qset q* **by** *auto*
**}**
**hence** *r2l*: (*wline k b*) $\cap$ *Ball* $\subseteq$ (*applyToSet* (*wvtFunc m k*) (*wline m b*)) $\cap$ *Ball*
**by** *auto*

**define** *lBall* **where** *lBall*: *lBall* = (*applyToSet* (*wvtFunc m k*) (*wline m b*)) $\cap$ *Ball*
**define** *rBall* **where** *rBall*: *rBall* = (*wline k b*) $\cap$ *Ball*

**hence** *equ*: *lBall* = *rBall* **using** *l2r r2l lBall rBall* **by** *auto*

**have** *yinBall*: $y \in$ *Ball* **using** *Ball d* **by** *auto*

**have** *tgt1*: $y \in$ (*applyToSet* (*wvtFunc m k*) (*wline m b*)) **using** *tgt* **by** *auto*
**hence** $y \in$ *lBall* **using** *yinBall lBall* **by** *auto*
**hence** *rtp1*: $y \in$ *wline k b* **using** *equ rBall* **by** *auto*

**have** *rtp2*: *onLine y l'* **using** *tgt* **by** *auto*

**have** *tgt3*: *accPoint y* (*applyToSet* (*wvtFunc m k*) (*wline m b*)) **using** *tgt* **by** *auto*
**hence** *tgt3'*: $\forall\ \varepsilon > 0.\ \exists\ q \in$ (*applyToSet* (*wvtFunc m k*) (*wline m b*)) . $(y \neq q) \wedge$ (*inBall q* $\varepsilon$ *y*)
**by** *auto*
**{ fix** *e*
**assume** *epos*: $e > 0$
**define** *d1* **where** *d'*: *d1* = *min d e*
**have** *dd*: $d1 \leq d$ **using** *d'* **by** *auto*
**have** *de*: $d1 \leq e$ **using** *d'* **by** *auto*
**have** *d'pos*: $d1 > 0$ **using** *dpos epos d'* **by** *auto*

**then obtain** *q*
**where** *q*: $q \in$ (*applyToSet* (*wvtFunc m k*) (*wline m b*)) $\wedge$ $(y \neq q)$ $\wedge$ (*inBall q d1 y*)
**using** *tgt3'* **by** *blast*
**hence** $q \in$ (*applyToSet* (*wvtFunc m k*) (*wline m b*)) $\wedge$ (*inBall q d y*) $\wedge$ $(y \neq q)$
**using** *lemBallInBall*[*of q y d1 d*] *d'pos dd* **by** *auto*
**hence** $q \in$ *lBall* $\wedge$ $(y \neq q)$ $\wedge$ (*inBall q d1 y*)

**using** *q Ball lemSep2Symmetry lBall* **by** *auto*
**hence** $q \in rBall \wedge (y \neq q) \wedge (inBall\ q\ e\ y)$
**using** *lemBallInBall[of q y d1 e] d'pos de equ* **by** *auto*

**hence** $\exists\ q \in rBall\ .\ (y \neq q) \wedge (inBall\ q\ e\ y)$ **by** *auto*
**}**
**hence** *rtp3*: $\forall\ e > 0.\ \exists\ q \in wline\ k\ b\ .\ (y \neq q) \wedge (inBall\ q\ e\ y)$
**using** *rBall* **by** *auto*

**have** *tgt4*: $(\exists\ p\ .\ (\ (onLine\ p\ l') \wedge (p \neq y) \wedge$
$(\forall\ \varepsilon > 0\ .\ \ \exists\ \delta > 0\ .\ \forall\ y' \in (applyToSet\ (wvtFunc\ m\ k)\ (wline$
$m\ b)).\ ($
$(\ (y'\ within\ \delta\ of\ y) \wedge (y' \neq y)\ )$
$\longrightarrow$
$(\ \exists\ r\ .\ ((onLine\ r\ (lineJoining\ y\ y')) \wedge (r\ within\ \varepsilon\ of\ p))))$
$)$
$))$ **using** *tgt* **by** *auto*
**then obtain** *p* **where** *p*: $(\ (onLine\ p\ l') \wedge (p \neq y) \wedge$
$(\forall\ \varepsilon > 0\ .\ \ \exists\ \delta > 0\ .\ \forall\ y' \in (applyToSet\ (wvtFunc\ m\ k)\ (wline$
$m\ b)).\ ($
$(\ (y'\ within\ \delta\ of\ y) \wedge (y' \neq y)\ )$
$\longrightarrow$
$(\ \exists\ r\ .\ ((onLine\ r\ (lineJoining\ y\ y')) \wedge (r\ within\ \varepsilon\ of\ p))))$
$))$ **by** *auto*
**have** *p1*: $onLine\ p\ l'$ **using** *p* **by** *auto*
**have** *p2*: $p \neq y$ **using** *p* **by** *auto*
**{ fix** *e*
**assume** *epos*: $e > 0$
**then obtain** *d2* **where** *d2*: $(d2 > 0) \wedge$
$(\forall\ y' \in (applyToSet\ (wvtFunc\ m\ k)\ (wline\ m\ b)).\ ($
$(\ (y'\ within\ d2\ of\ y) \wedge (y' \neq y)\ )$
$\longrightarrow$
$(\ \exists\ r\ .\ ((onLine\ r\ (lineJoining\ y\ y')) \wedge (r\ within\ e\ of\ p))))$
$)$ **using** *p* **by** *auto*
**hence** *d2pos*: $d2 > 0$ **by** *auto*

**define** *dm* **where** *dm*: $dm = min\ d2\ d$
**have** *dmd2*: $dm \leq d2$ **using** *dm* **by** *auto*
**have** *dmd*: $dm \leq d$ **using** *dm* **by** *auto*
**have** *dmpos*: $dm > 0$ **using** *dpos d2pos dm* **by** *auto*

**{ fix** $y'$
**assume** $y'$: $(y' \in wline\ k\ b) \wedge (y'\ within\ dm\ of\ y) \wedge (y' \neq y)$
**hence** *ydm*: $y'\ within\ dm\ of\ y$ **by** *auto*
**hence** $y'\ within\ d\ of\ y$ **using** *dmpos dmd lemBallInBall[of y' y*
*dm d]* **by** *auto*
**hence** $y' \in Ball$ **using** *Ball lemSep2Symmetry* **by** *auto*
**hence** $y' \in rBall$ **using** *y' rBall* **by** *auto*

**hence** *yL*: $y' \in$ *lBall* **using** *equ* **by** *auto*

**have** $y'$ *within d2 of y*
**using** *ydm dmpos dmd2 lemBallInBall*[*of* $y'$ *y dm d2*] **by** *auto*

**hence** $y' \in$ (*applyToSet* (*wvtFunc m k*) (*wline m b*)) $\wedge$ ($y'$ *within d2 of y*) $\wedge$ ($y' \neq y$)
**using** $y'$ *yL lBall* **by** *auto*
**hence** $\exists$ *r* . ((*onLine r* (*lineJoining y* $y'$)) $\wedge$ (*r within e of p*))
**using** *d2* **by** *auto*
**}**
**hence** $\exists$ *dm* $>$ *0*. $\forall$ $y' \in$ (*wline k b*) .
($y'$ *within dm of y*) $\wedge$ ($y' \neq y$)
$\longrightarrow$ ($\exists$ *r* . ((*onLine r* (*lineJoining y* $y'$)) $\wedge$ (*r within e of p*)))
**using** *dmpos* **by** *blast*
**}**
**hence** $\forall$ *e* $>$ *0* . $\exists$ *dm* $>$ *0*. $\forall$ $y' \in$ (*wline k b*) .
($y'$ *within dm of y*) $\wedge$ ($y' \neq y$)
$\longrightarrow$ ($\exists$ *r* . ((*onLine r* (*lineJoining y* $y'$)) $\wedge$ (*r within e of p*)))
**by** *auto*
**hence** *rtp4*: $\exists$ *p* . ( (*onLine p* $l'$) $\wedge$ ($p \neq y$) $\wedge$ ($\forall$ *e* $>$ *0* . $\exists$ *dm* $>$ *0*. $\forall$ $y' \in$ (*wline k b*) .
($y'$ *within dm of y*) $\wedge$ ($y' \neq y$)
$\longrightarrow$ ($\exists$ *r* . ((*onLine r* (*lineJoining y* $y'$)) $\wedge$ (*r within e of p*)))))
**using** *p1 p2* **by** *auto*

**hence** *tangentLine* $l'$ (*wline k b*) *y* **using** *rtp1 rtp2 rtp3 rtp4* **by** *blast*
**thus** *?thesis* **by** *auto*
**qed**

**lemma** *lemTangentLines*:
**assumes** *affineApprox A* (*wvtFunc m k*) *x*
**and** *tl l m b x*
**and** *applyAffineToLine A l* $l'$
**and** *wvtFunc m k x y*
**shows** *tl* $l'$ *k b y*
**proof** −
**have** *pres*: $x \in$ *wline m b*
$\longrightarrow$ *tangentLine l* (*wline m b*) *x*
$\longrightarrow$ *affineApprox A* (*wvtFunc m k*) *x*
$\longrightarrow$ *wvtFunc m k x y*
$\longrightarrow$ *applyAffineToLine A l* $l'$
$\longrightarrow$ *tangentLine* $l'$ (*wline k b*) *y*

**using** *lemPresentation*[*of* $x$ $m$ $b$ $l$ $k$ $A$ $y$ $l'$]
**by** *blast*

**have** *1*: $x \in$ *wline* $m$ $b$ **using** *assms*(*2*) **by** *auto*
**have** *2*: *tangentLine* $l$ (*wline* $m$ $b$) $x$ **using** *assms*(*2*) **by** *auto*
**have** *3*: *affineApprox* $A$ (*wvtFunc* $m$ $k$) $x$ **using** *assms*(*1*) **by** *simp*
**have** *4*: *wvtFunc* $m$ $k$ $x$ $y$ **using** *assms*(*4*) **by** *simp*
**have** *5*: *applyAffineToLine* $A$ $l$ $l'$ **using** *assms*(*3*) **by** *simp*

**have** *tangentLine* $l'$ (*wline* $k$ $b$) $y$ **using** *pres 1 2 3 4 5* **by** *meson*
**thus** *?thesis* **by** *auto*
**qed**

**lemma** *lemSelfTangentIsTimeAxis*:
**assumes** *tangentLine* $l$ (*wline* $k$ $k$) $x$
**shows** $l$ = *timeAxis*
**proof** −
**define** $s$ **where** $s$: $s$ = *wline* $k$ $k$
**hence** $s \subseteq$ *timeAxis* **using** $s$ *AxSelfMinus* **by** *blast*
**hence** *xOnAxis*: *onTimeAxis* $x$ **using** *assms*(*1*) $s$ **by** *auto*

**have** $x$: ($x \in s$) $\wedge$ (*onLine* $x$ $l$) $\wedge$ (*accPoint* $x$ $s$)
$\wedge$ ($\exists$ $p$ . ( (*onLine* $p$ $l$) $\wedge$ ($p \neq x$) $\wedge$
($\forall$ $\varepsilon > 0$ . $\exists$ $\delta > 0$ . $\forall$ $z \in s$. (
( ($z$ *within* $\delta$ *of* $x$) $\wedge$ ($z \neq x$) )
$\longrightarrow$
( $\exists$ $r$ . ((*onLine* $r$ (*lineJoining* $x$ $z$)) $\wedge$ ($r$ *within* $\varepsilon$ *of* $p$))))
))) **using** $s$ *assms*(*1*) **by** *auto*
**then obtain** $p$ **where**
$p$: ( (*onLine* $p$ $l$) $\wedge$ ($p \neq x$) $\wedge$
($\forall$ $\varepsilon > 0$ . $\exists$ $\delta > 0$ . $\forall$ $z \in s$. (
( ($z$ *within* $\delta$ *of* $x$) $\wedge$ ($z \neq x$) )
$\longrightarrow$
( $\exists$ $r$ . ((*onLine* $r$ (*lineJoining* $x$ $z$)) $\wedge$ ($r$ *within* $\varepsilon$ *of* $p$))))
)) **by** *auto*

**have** *accxs*: *accPoint* $x$ $s$ **using** $x$ **by** *auto*

**define** *p0* **where**
*p0*: *p0* = (| *tval* = *tval* $p$, *xval* = *0*, *yval* = *0*, *zval* = *0* |)
**hence** *p0OnAxis*: *onTimeAxis p0* **by** *auto*

**define** *dp* **where** *dp*: *dp* = *sep2* $p$ *p0*

**have** *pp0*: $dp = sqr\ (tval\ p0\ -\ tval\ p) + sqr\ (xval\ p0\ -\ xval\ p) +$
$sqr\ (yval\ p0\ -\ yval\ p) + sqr\ (zval\ p0\ -\ zval\ p)$
**using** *dp p0* **by** *simp*
**moreover have** $\ldots = sqr\ (xval\ p) + sqr\ (yval\ p) + sqr\ (zval\ p)$
**using** *p0* **by** *auto*
**ultimately have** *dpval*: $dp = sqr\ (xval\ p) + sqr\ (yval\ p) + sqr\ (zval\ p)$
**using** *dp* **by** *simp*

**define** *e* **where** *e*: $e = (min\ dp\ 1)\ /\ 2$
**define** *e2* **where** *e2*: $e2 = sqr\ e$
**have** *e2ledp*: $e2 \leq dp$
**proof** $-$
**have** *msmall*: $0 \leq (min\ dp\ 1) \leq 1$ **using** *lemNorm2NonNeg dp* **by** *auto*
**hence** *esmall*: $0 \leq e < 1$ **using** *e leI* **by** *force*
**hence** *e2lte*: $e2 \leq e$ **using** *e2 mult-left-le* **by** *force*

**have** *mrange*: $0 \leq (min\ dp\ 1) \leq dp$ **using** *lemNorm2NonNeg dp* **by** *auto*
**hence** $e \leq dp/2$ **using** *e divide-right-mono zero-le-numeral* **by** *blast*
**hence** $e \leq dp$ **using** *msmall e add-increasing2 divide-nonneg-nonneg*

*le-cases lemSumOfTwoHalves min-def zero-le-numeral*
**by** *metis*

**thus** *?thesis* **using** *e2lte* **by** *auto*
**qed**

**have** *offaxis*: $\forall\ z\ .\ (dp > 0) \land onTimeAxis\ z \longrightarrow \neg\ (z\ within\ e\ of\ p)$
**proof** $-$
**{ fix** *z*
**{ assume** *z*: $(dp > 0) \land onTimeAxis\ z$

**have** $sep2\ z\ p = sqr\ (tval\ z\ -\ tval\ p)$
$+\ sqr\ (xval\ z\ -\ xval\ p)$
$+\ sqr\ (yval\ z\ -\ yval\ p)$
$+\ sqr\ (zval\ z\ -\ zval\ p)$ **using** *p0* **by** *simp*
**moreover have** $\ldots = sep2\ z\ p0$
$+\ sqr\ (xval\ p) + sqr\ (yval\ p) + sqr\ (zval\ p)$
**using** *p0 z* **by** *auto*
**moreover have** $\ldots = sep2\ z\ p0 + dp$
**using** *dpval add-assoc*
**by** *presburger*
**moreover have** $\ldots \geq dp$ **using** *lemNorm2NonNeg* **by** *simp*
**ultimately have** $sep2\ z\ p \geq e2$

**using** *e2ledp dual-order.trans* **by** *presburger*
**}**
**hence** $(0 < dp) \wedge onTimeAxis\ z \longrightarrow \neg\ (z\ within\ e\ of\ p)$
**using** *e2* **by** *auto*
**}**
**thus** *?thesis* **by** *auto*
**qed**

**{ assume** *dpnz*: $dp > 0$

**hence** *enz*: $e > 0$ **using** *e* **by** *auto*
**then obtain** *d* **where** *d*: $(d > 0) \wedge (\forall\ z \in s.\ ($
$(\ (z\ within\ d\ of\ x) \wedge (z \neq x)\ )$
$\longrightarrow$
$(\ \exists\ r\ .\ ((onLine\ r\ (lineJoining\ x\ z)) \wedge (r\ within\ e\ of\ p)))))$
**using** *p* **by** *blast*

**obtain** *q* **where** *q*: $(q \in s) \wedge (x \neq q) \wedge (inBall\ q\ d\ x)$
**using** *accxs dpnz enz d* **by** *blast*

**hence** *qOnAxis*: $q \in timeAxis$ **using** *s AxSelfMinus* **by** *blast*

**have** *qprops*: $(q\ within\ d\ of\ x) \wedge (q \neq x)$ **using** *q* **by** *auto*
**then obtain** *r* **where** *r*: $(onLine\ r\ (lineJoining\ x\ q)) \wedge (r\ within\ e\ of\ p)$
**using** *d q* **by** *blast*

**have** $x \neq q$ **using** *q* **by** *auto*
**moreover have** *onLine x timeAxis* **using** *xOnAxis lemTimeAxisIsLine* **by** *auto*
**moreover have** *onLine q timeAxis* **using** *qOnAxis lemTimeAxisIsLine* **by** *auto*
**ultimately have** $timeAxis = lineJoining\ x\ q$
**using** *lemLineAndPoints*[*of x q timeAxis*]
**by** *auto*
**hence** *rOnAxis*: (*onTimeAxis r*) **using** *r* **by** *auto*

**hence** $\neg\ (r\ within\ e\ of\ p)$ **using** *offaxis dpnz* **by** *blast*

**hence** *False* **using** *r* **by** *blast*
**}**
**hence** $\neg\ (dp > 0) \wedge (dp \geq 0)$ **using** *lemNorm2NonNeg dp* **by** *auto*
**hence** $dp = 0$ **by** *auto*
**hence** $p = p0$ **using** *dp lemNullImpliesOrigin*[*of* $p \ominus p0$] **by** *auto*

**hence** *onLine p timeAxis* **using** *p0OnAxis lemTimeAxisIsLine* **by** *auto*

```
  moreover have onLine x timeAxis using xOnAxis lemTimeAxisIs-
Line by auto
  moreover have pnotx: p ≠ x using p by blast
  ultimately have xp: timeAxis = lineJoining x p
    using lemLineAndPoints[of x p timeAxis]
    by auto

  have onLine p l using p by auto
  moreover have onLine x l using x by auto
  ultimately have l = lineJoining x p
    using lemLineAndPoints[of x p l]  pnotx
    by auto

  hence timeAxis = l using xp by auto
  thus ?thesis using s by blast
qed



lemma lemTangentLineUnique:
  assumes tl l1 m k x
and        tl l2 m k x
and        affineApprox A (wvtFunc m k) x
and        wvtFunc m k x y
and        x ∈ wline m k
shows      l1 = l2
proof −
  define L1 where L1: L1 = applyToSet (asFunc A) l1
  define L2 where L2: L2 = applyToSet (asFunc A) l2

  define p1  where p1:  p1  = (x ∈ wline m k)
  define p2a where p2a: p2a = tangentLine l1 (wline m k) x
  define p2b where p2b: p2b = tangentLine l2 (wline m k) x
  define p3  where p3:  p3  = affineApprox A (wvtFunc m k) x
  define p4  where p4:  p4  = wvtFunc m k x y
  define p5a where p5a: p5a = applyAffineToLine A l1 L1
  define p5b where p5b: p5b = applyAffineToLine A l2 L2

  define tgt1 where tgt1: tgt1 = tangentLine L1 (wline k k) y
  define tgt2 where tgt2: tgt2 = tangentLine L2 (wline k k) y

  have pre1:  p1  using p1  assms(5) by auto
  have pre2a: p2a using p2a assms(1) by auto
  have pre2b: p2b using p2b assms(2) by auto
  have pre3:  p3  using p3  assms(4) using assms(3) by auto
  have pre4:  p4  using p4  assms(4) by auto

  have isLine l1 using assms(1) by auto
  hence pre5a: p5a using p5a L1 assms(3) lemAffineOfLineIsLine[of
```

*l1 A L1*] **by** *auto*

**have** *isLine l2* **using** *assms*(*2*) **by** *auto*
**hence** *pre5b*: *p5b* **using** *p5b L2 assms*(*3*) *lemAffineOfLineIsLine*[*of l2 A L2*] **by** *auto*

**have** $[\![$ *p1*; *p2a*; *p3*; *p4*; *p5a* $]\!] \Longrightarrow$ *tgt1*
**using** *p1 p2a p3 p4 p5a tgt1 lemPresentation*[*of x m k l1 k A y L1*]
**by** *fast*
**hence** *tgt1* **using** *pre1 pre2a pre3 pre4 pre5a* **by** *auto*
**hence** *L1axis*: *L1* = *timeAxis* **using** *tgt1 lemSelfTangentIsTimeAxis* **by** *auto*

**have** $[\![$ *p1*; *p2b*; *p3*; *p4*; *p5b* $]\!] \Longrightarrow$ *tgt2*
**using** *p1 p2b p3 p4 p5b tgt2 lemPresentation*[*of x m k l2 k A y L2*]
**by** *fast*
**hence** *tgt2* **using** *pre1 pre2b pre3 pre4 pre5b* **by** *auto*
**hence** *L2* = *timeAxis* **using** *tgt2 lemSelfTangentIsTimeAxis* **by** *auto*

**hence** *L1L2*: *L1* = *L2* **using** *L1axis* **by** *auto*

**obtain** $A'$ **where** $A'$: (*affine* $A'$) $\wedge$ ($\forall$ *p q* . *A p* = *q* $\longleftrightarrow$ $A'$ *q* = *p*)
**using** *assms*(*3*) *lemInverseAffine*[*of A*] **by** *auto*
**{ fix** *p*
**define** *q* **where** *q*: *q* = *A p*
**hence** $A'q$: $A'$ *q* = *p* **using** $A'$ **by** *auto*

**{ assume** *p* $\in$ *l1*
**hence** *q* $\in$ *L2* **using** *q L1 L1L2* **by** *auto*
**then obtain** *p2* **where** *p2*: *q* = *A p2* $\wedge$ *p2* $\in$ *l2* **using** *L2* **by** *auto*
**hence** $A'$ *q* = *p2* **using** $A'$ **by** *auto*
**hence** *p* = *p2* **using** $A'q$ **by** *auto*
**hence** *p* $\in$ *l2* **using** *p2* **by** *auto*
**}**
**hence** *l2r*: *p* $\in$ *l1* $\longrightarrow$ *p* $\in$ *l2* **by** *blast*

**{ assume** *p* $\in$ *l2*
**hence** *q* $\in$ *L1* **using** *q L2 L1L2* **by** *auto*
**then obtain** *p1* **where** *p1*: *q* = *A p1* $\wedge$ *p1* $\in$ *l1* **using** *L1* **by** *auto*
**hence** $A'$ *q* = *p1* **using** $A'$ **by** *auto*
**hence** *p* = *p1* **using** $A'q$ **by** *auto*
**hence** *p* $\in$ *l1* **using** *p1* **by** *auto*
**}**
**hence** *p* $\in$ *l2* $\longrightarrow$ *p* $\in$ *l1* **by** *blast*

```
    hence p ∈ l1 ⟷ p ∈ l2 using l2r by auto
  }

  thus ?thesis by blast
qed



end

end
```

# 25 Proposition2

This theory shows that affine approximations map surfaces of cones to (subsets of) surfaces of cones.

```
theory Proposition2
  imports TangentLineLemma
begin


class Proposition2 = TangentLineLemma
begin

lemma lemProposition2:
  assumes affineApprox A (wvtFunc m k) x
  shows  applyToSet (asFunc A) (coneSet m x) ⊆ coneSet k (A x)
proof -

  define y where y: y = A x
  define lhs where lhs: lhs = applyToSet (asFunc A) (coneSet m x)
  define rhs where rhs: rhs = coneSet k y

  have mkxy: wvtFunc m k x y
    using assms lemAffineEqualAtBase[of wvtFunc m k A x] y
    by auto
  have affA: affine A using assms by auto


  { fix q
    { assume q: q ∈ lhs
      hence ∃ p . (p ∈ coneSet m x) ∧ (asFunc A) p q using lhs by
auto
      then obtain p where
        p: (p ∈ coneSet m x) ∧ (asFunc A) p q
        by presburger
```

```
      hence qAp: q = A p using affA by auto

      have  cone m x p using p by auto
      then obtain l where
        l: (onLine p l) ∧  (onLine x l) ∧ (∃ ph . Ph ph ∧ tl l m ph x)
        by auto
      then obtain ph where ph: Ph ph ∧ tl l m ph x by auto
      have lineL: isLine l using l by auto
      have tll: tl l m ph x using ph by auto

      define l' where l': l' = applyToSet (asFunc A) l
      hence aatl: applyAffineToLine A l l'
        using lineL affA lemAffineOfLineIsLine[of l A l']
        by simp

      hence tll': tl l' k ph y
        using assms(1) tll mkxy
              lemTangentLines[of m k A x ph l l' y]
        by simp

      hence (Ph ph ∧ tl l' k ph y)
        using ph by auto
      hence exPh: ∃ ph . (Ph ph ∧ tl l' k ph y)
        using exI[of λ b. (Ph b ∧ tl l' k b y) ph]
        by auto

      have p ∈ l using l by auto
      hence q ∈ l' using qAp q l' by auto
      moreover have lineL': isLine l' using tll' by auto
      ultimately have qonl': onLine q l' by auto

      hence (onLine q l') ∧  (onLine y l') ∧ (∃ ph . Ph ph ∧ tl l' k ph
y)
        using exPh tll' by blast

      hence q ∈ rhs using y tll' rhs by auto
    }
    hence q ∈ lhs ⟶ q ∈ rhs by auto
  }
  hence l2r: lhs ⊆ rhs by auto
  thus ?thesis using lhs rhs y by auto
qed




end
```

**end**

# 26 AXIOM: AxEventMinus

This theory declares the axiom AxEventMinus

**theory** *AxEventMinus*
  **imports** *WorldView*
**begin**

AxEventMinus: An observer encounters the events in which they are observed.

**class** *axEventMinus* = *WorldView*
**begin**

  **abbreviation** $axEventMinus :: Body \Rightarrow Body \Rightarrow {'a}\ Point \Rightarrow bool$
    **where** $axEventMinus\ m\ k\ p \equiv (m\ sees\ k\ at\ p)$
$\longrightarrow (\exists\ q\ .\ \forall\ b\ .\ (\ (m\ sees\ b\ at\ p) \longleftrightarrow (k\ sees\ b\ at\ q)))$

**end**

**class** *AxEventMinus* = *axEventMinus* +
  **assumes** *AxEventMinus*: $\forall\ m\ k\ p\ .\ axEventMinus\ m\ k\ p$
**begin**
**end**

**end**

# 27 Proposition3

This theory collects together earlier results to show that worldview transformations can be approximated by affine transformations that have various useful properties.

**theory** *Proposition3*
  **imports** *Proposition1 Proposition2 AxEventMinus*
**begin**

**class** *Proposition3* = *Proposition1* + *Proposition2* + *AxEventMinus*
**begin**

**lemma** *lemProposition3*:
**assumes** *m sees k at x*
**shows** $\exists$ *A y* . (*wvtFunc m k x y*)
$\wedge$ (*affineApprox A* (*wvtFunc m k*) *x*)
$\wedge$ (*applyToSet* (*asFunc A*) (*coneSet m x*) $\subseteq$ *coneSet k y*)
$\wedge$ (*coneSet k y* = *regularConeSet y*)
**proof** −
**define** *g1* **where** *g1*: *g1* = ($\lambda$ *y* . *wvtFunc m k x y*)
**define** *g2* **where** *g2*: *g2* = ($\lambda$ *A* . *affineApprox A* (*wvtFunc m k*) *x*)
**define** *g3* **where** *g3*: *g3* = ($\lambda$ *A y* . *applyToSet* (*asFunc A*) (*coneSet m x*) $\subseteq$ *coneSet k y*)
**define** *g4* **where** *g4*: *g4* = ($\lambda$ *y* . *coneSet k y* = *regularConeSet y*)

**have** *axEventMinus m k x* **using** *AxEventMinus* **by** *simp*
**hence** ($\exists$ *q* . $\forall$ *b* . ( (*m sees b at x*) $\longleftrightarrow$ (*k sees b at q*)))
**using** *assms* **by** *simp*

**then obtain** *y* **where** *y*: $\forall$ *b* . ( (*m sees b at x*) $\longleftrightarrow$ (*k sees b at y*)) **by** *auto*
**hence** *ev m x* = *ev k y* **by** *blast*
**hence** *goal1*: *g1 y* **using** *assms g1* **by** *auto*

**have** *axDiff m k x* **using** *AxDiff* **by** *simp*
**hence** $\exists$ *A* . *affineApprox A* (*wvtFunc m k*) *x* **using** *g1 goal1* **by** *blast*
**then obtain** *A* **where** *goal2*: *g2 A* **using** *g2* **by** *auto*

**have** *applyToSet* (*asFunc A*) (*coneSet m x*) $\subseteq$ *coneSet k* (*A x*)
**using** *g2 goal2 lemProposition2*[*of m k A x*]
**by** *auto*
**moreover have** *A x* = *y*
**using** *goal1 goal2 g1 g2 lemAffineEqualAtBase*[*of wvtFunc m k A x*]
**by** *blast*
**ultimately have** *goal3*: *g3 A y* **using** *g3* **by** *auto*

**have** *k sees k at y* **using** *assms*(*1*) *g1 goal1* **by** *fastforce*
**hence** $\forall$ *p* . *cone k y p* = *regularCone y p*
**using** *lemProposition1* [*of y k*] **by** *auto*
**hence** *goal4*: *g4 y* **using** *g4* **by** *force*

**hence** $\exists$ *A y* . (*g1 y*) $\wedge$ (*g2 A*) $\wedge$ (*g3 A y*) $\wedge$ (*g4 y*)
**using** *goal1 goal2 goal3 goal4* **by** *blast*

**thus** *?thesis* **using** *g1 g2 g3 g4* **by** *fastforce*
**qed**

**end**

**end**

# 28 ObserverConeLemma

This theory gives sufficient conditions for an observed observer's cone to appear upright to that observer.

**theory** *ObserverConeLemma*
**imports** *Proposition3*
**begin**

**class** *ObserverConeLemma* = *Proposition3*
**begin**

**lemma** *lemConeOfObserved*:
**assumes** *affineApprox A* (*wvtFunc m k*) *x*
**and** *m sees k at x*
**shows** *coneSet k* (*A x*) = *regularConeSet* (*A x*)
**proof** −
**have** *Ax*: $\forall$ *y*. (*wvtFunc m k x y*) $\longleftrightarrow$ (*y* = *A x*)
**using** *assms*(*1*) *lemAffineEqualAtBase*[*of* (*wvtFunc m k*) *A x*]
**by** *auto*

**define** *set1* **where** *set1*: *set1* = *coneSet k* (*A x*)
**define** *set2* **where** *set2*: *set2* = *regularConeSet* (*A x*)
**define** *P* **where** *P*: *P* = ($\lambda$ *A′ y* . (*wvtFunc m k x y*)
$\wedge$ (*affineApprox A′* (*wvtFunc m k*) *x*)
$\wedge$ (*applyToSet* (*asFunc A′*) (*coneSet m x*) $\subseteq$ *coneSet k y*)
$\wedge$ (*coneSet k y* = *regularConeSet y*))
**have** *m sees k at x* **using** *assms*(*2*) **by** *auto*
**hence** $\exists$ *A′ y* . *P A′ y* **using** *P lemProposition3*[*of m k x*] **by** *fast*
**then obtain** *A′ y* **where** *A′y*: *P A′ y* **by** *auto*

**have** *wvtFunc m k x y* **using** *P A′y* **by** *auto*
**hence** *y* = *A x* **using** *Ax* **by** *auto*
**moreover have** *coneSet k y* = *regularConeSet y* **using** *A′y P* **by** *auto*
**ultimately show** *?thesis* **using** *set1 set2* **by** *auto*
**qed**

**end**

**end**

# 29 Quadratics

This theory shows how to find the roots of a quadratic, assuming that roots exist (AxEField).

**theory** *Quadratics*
  **imports** *Functions AxEField*
**begin**

**class** *Quadratics* = *Functions* + *AxEField*
**begin**

**abbreviation** *quadratic* :: $'a \Rightarrow 'a \Rightarrow 'a \Rightarrow ('a \Rightarrow 'a)$
  **where** $quadratic\ a\ b\ c \equiv \lambda\ x\ .\ a*(sqr\ x) + b*x + c$

**abbreviation** *qroot* :: $'a \Rightarrow 'a \Rightarrow 'a \Rightarrow 'a \Rightarrow bool$
  **where** $qroot\ a\ b\ c\ r \equiv (quadratic\ a\ b\ c)\ r = 0$

**abbreviation** *qroots* :: $'a \Rightarrow 'a \Rightarrow 'a \Rightarrow 'a\ set$
  **where** $qroots\ a\ b\ c \equiv \{\ r\ .\ qroot\ a\ b\ c\ r\ \}$

**abbreviation** *discriminant* :: $'a \Rightarrow 'a \Rightarrow 'a \Rightarrow 'a$
  **where** $discriminant\ a\ b\ c \equiv (sqr\ b) - 4*a*c$

**abbreviation** *qcase1* :: $'a \Rightarrow 'a \Rightarrow 'a \Rightarrow bool$
  **where** $qcase1\ a\ b\ c \equiv (a = 0 \wedge b = 0 \wedge c = 0)$
**abbreviation** *qcase2* :: $'a \Rightarrow 'a \Rightarrow 'a \Rightarrow bool$
  **where** $qcase2\ a\ b\ c \equiv (a = 0 \wedge b = 0 \wedge c \neq 0)$
**abbreviation** *qcase3* :: $'a \Rightarrow 'a \Rightarrow 'a \Rightarrow bool$
  **where** $qcase3\ a\ b\ c \equiv (a = 0 \wedge b \neq 0 \wedge (c = 0 \vee c \neq 0))$
**abbreviation** *qcase4* :: $'a \Rightarrow 'a \Rightarrow 'a \Rightarrow bool$
  **where** $qcase4\ a\ b\ c \equiv (a \neq 0 \wedge discriminant\ a\ b\ c < 0)$
**abbreviation** *qcase5* :: $'a \Rightarrow 'a \Rightarrow 'a \Rightarrow bool$
  **where** $qcase5\ a\ b\ c \equiv (a \neq 0 \wedge discriminant\ a\ b\ c = 0)$
**abbreviation** *qcase6* :: $'a \Rightarrow 'a \Rightarrow 'a \Rightarrow bool$
  **where** $qcase6\ a\ b\ c \equiv (a \neq 0 \wedge discriminant\ a\ b\ c > 0)$

**lemma** *lemQuadRootCondition*:
  **assumes** $a \neq 0$
  **shows** $(sqr\ (2*a*r + b) = discriminant\ a\ b\ c) \longleftrightarrow qroot\ a\ b\ c\ r$

**proof** −
**have** *sqr* (*2*∗*a*∗*r*) = (*4*∗*a*) ∗ (*a* ∗ *sqr r*)
**using** *lemSqrMult local.numeral-sqr mult-assoc sqr.simps*(*1*) *sqr.simps*(*2*)
**by** *metis*
**moreover have** *2*∗(*2*∗*a*∗*r*)∗*b* = (*4*∗*a*) ∗ (*b*∗*r*)
**by** (*metis dbl-def dbl-simps*(*5*) *mult.left-commute mult-2 mult-2-right mult-assoc*)
**ultimately have** *s*: *sqr* (*2*∗*a*∗*r*) + *2*∗(*2*∗*a*∗*r*)∗*b* = (*4*∗*a*) ∗ ((*a* ∗ *sqr r*) + *b* ∗*r*)
**by** (*simp add*: *local.distrib-left*)

**have** *sqr*(*2*∗*a*∗*r* + *b*) = *sqr* (*2*∗*a*∗*r*) + *2*∗(*2*∗*a*∗*r*)∗*b* + *sqr b*
**using** *lemSqrSum* **by** *auto*
**moreover have** . . . = (*4*∗*a*) ∗ ((*a* ∗ *sqr r*) + *b* ∗*r*) + *sqr b* **using** *s* **by** *auto*
**moreover have** . . . = (*4*∗*a*) ∗ ((*a* ∗ *sqr r*) + *b* ∗*r* + *c*) − (*4*∗*a*)∗*c* + *sqr b*
**by** (*simp add*: *distrib-left*)
**ultimately have** *eqn1*: *sqr*(*2*∗*a*∗*r* + *b*) = (*4*∗*a*)∗(*quadratic a b c r*) + (*discriminant a b c*)
**by** (*simp add*: *add-diff-eq diff-add-eq*)

{ **assume** *qroot a b c r*
**hence** *sqr* (*2*∗*a*∗*r* + *b*) = *discriminant a b c* **using** *eqn1* **by** *simp*
}
**hence** *l2r*: *qroot a b c r* ⟶ *sqr* (*2*∗*a*∗*r* + *b*) = *discriminant a b c* **by** *auto*

{ **assume** *sqr* (*2*∗*a*∗*r* + *b*) = *discriminant a b c*
**hence** *0* = (*4*∗*a*)∗(*quadratic a b c r*) **using** *eqn1* **by** *auto*
**hence** *qroot a b c r* **by** (*metis assms divisors-zero zero-neq-numeral*)
}
**hence** (*sqr* (*2*∗*a*∗*r* + *b*) = *discriminant a b c*) ⟶ *qroot a b c r* **by** *blast*

**thus** *?thesis* **using** *l2r* **by** *blast*
**qed**

**lemma** *lemQuadraticCasesComplete*:
**shows** *qcase1 a b c* ∨ *qcase2 a b c* ∨ *qcase3 a b c* ∨ *qcase4 a b c* ∨
*qcase5 a b c* ∨ *qcase6 a b c*
**using** *not-less-iff-gr-or-eq* **by** *blast*

**lemma** *lemQCase1*:
**assumes** *qcase1 a b c*
**shows** ∀ *r* . *qroot a b c r*
**using** *assms* **by** *simp*

**lemma** *lemQCase2*:
  **assumes** *qcase2 a b c*
  **shows** $\neg$ ($\exists$ *r* . *qroot a b c r*)
  **by** (*simp add*: *assms*)

**lemma** *lemQCase3*:
  **assumes** *qcase3 a b c*
  **shows** *qroot a b c r* $\longleftrightarrow$ $r = -c/b$
**proof** –
  **have** *qroot a b c r* $\longrightarrow$ $r = -c/b$
  **proof** –
    **{ assume** *hyp*: *qroot a b c r*
      **hence** $b{*}r + c = 0$ **using** *assms* **by** *auto*
      **hence** $b{*}r = -c$ **by** (*simp add*: *local.eq-neg-iff-add-eq-0*)
    **hence** $r = -c/b$ **by** (*metis assms local.nonzero-mult-div-cancel-left*)
    **}**
    **thus** *?thesis* **by** *auto*
  **qed**
  **moreover have** $r = -c/b$ $\longrightarrow$ *qroot a b c r* **by** (*simp add*: *assms*)
  **ultimately show** *?thesis* **by** *blast*
**qed**

**lemma** *lemQCase4*:
  **assumes** *qcase4 a b c*
  **shows** $\neg$ ($\exists$ *r* . *qroot a b c r*)
**proof** –
  **have** *props*: ($a \neq 0 \wedge$ *discriminant a b c* $< 0$) **using** *assms* **by** *auto*
  **{ assume** *hyp*: $\exists$ *r* . *qroot a b c r*
    **then obtain** *r* **where** *r*: *qroot a b c r* **by** *auto*
    **hence** *sqr* $(2 * a * r + b) =$ *discriminant a b c*
      **using** *props lemQuadRootCondition*[*of a r b c*] **by** *auto*
    **hence** *sqr* $(2{*}a{*}r + b) < 0$ **using** *props* **by** *auto*
    **hence** *False* **using** *lemSquaresPositive* **by** *auto*
  **}**
  **thus** *?thesis* **by** *auto*
**qed**

**lemma** *lemQCase5*:
  **assumes** *qcase5 a b c*
  **shows** *qroot a b c r* $\longleftrightarrow$ $r = -b/(2{*}a)$
**proof** –

**have** *qroot a b c r* $\longrightarrow$ $r = -b/(2*a)$
**proof** –
{ **assume** *hyp*: *qroot a b c r*
**hence** $sqr\ (2 * a * r + b) = 0$
**using** *assms lemQuadRootCondition*[*of a r b c*] **by** *auto*
**hence** $2*a*r + b = 0$ **by** *simp*
**hence** $2*a*r = -b$ **using** *local.eq-neg-iff-add-eq-0* **by** *auto*
**moreover have** $2*a \neq 0$ **using** *assms* **by** *auto*
**ultimately have** $r = ((-b)/(2*a))$ **by** (*metis local.nonzero-mult-div-cancel-left*)
}
**thus** *?thesis* **by** *auto*
**qed**
**moreover have** $r = -b/(2*a)$ $\longrightarrow$ *qroot a b c r*
**proof** –
{ **assume** *hyp*: $r = -b/(2*a)$
**hence** $(2*a)*r + b =$ *discriminant a b c* **by** (*simp add*: *assms*)
**hence** *qroot a b c r* **using** *lemQuadRootCondition*[*of a r b c*]
*assms* **by** *auto*
}
**thus** *?thesis* **by** *auto*
**qed**
**ultimately show** *?thesis* **by** *blast*
**qed**

**lemma** *lemQCase6*:
**assumes** *qcase6 a b c*
**and** *rd* = *sqrt* (*discriminant a b c*)
**and** $rp = ((-b) + rd)\ /\ (2*a)$
**and** $rm = ((-b) - rd)\ /\ (2*a)$
**shows** $(rp \neq rm) \wedge$ *qroots a b c* = { *rp*, *rm* }
**proof** –
**define** *d* **where** *d*: *d* = *discriminant a b c*

**have** *dpos*: $d > 0$ **using** *assms d* **by** *auto*
**hence** *rootd*: *hasUniqueRoot d* **using** *AxEField lemSqrt*[*of d*] **by**
*auto*
**hence** *rdprops*: $0 \leq rd \wedge d = sqr\ rd$
**using** *assms*(*2*) *d theI'*[*of isNonNegRoot d*] **by** *auto*
**hence** *rdnot0*: $rd \neq 0$ **using** *assms dpos mult-nonneg-nonpos* **by**
*auto*
**hence** *rdpos*: $rd > 0$ **using** *rdprops* **by** *auto*

**define** *pp* **where** *pp*: $pp = (-b) + rd$
**define** *mm* **where** *mm*: $mm = (-b) - rd$
**have** $rd \neq -rd$ **using** *rdnot0* **by** *simp*
**hence** $pp \neq mm$ **using** *pp mm add-left-imp-eq*[*of* $-b$ *rd* $-rd$] **by**

```
auto
  moreover have aa: 2∗a ≠ 0 using assms by auto
  ultimately have pp/(2∗a) ≠ mm/(2∗a) by auto
  hence conj1: rp ≠ rm using assms pp mm by simp

  have conj2: qroots a b c = {rp, rm}
  proof −
    { fix r assume r ∈ qroots a b c
      hence sqr (2∗a∗r + b) = d
        using assms lemQuadRootCondition d by auto
     hence sqrt d = abs (2∗a∗r + b) using lemSqrtOfSquare by blast
      moreover have sqrt d = rd using d assms by auto
      ultimately have rdprops: rd = abs (2∗a∗r + b) by auto

      define v :: 'a where v: v = 2∗a∗r + b
      hence vnot0: v ≠ 0 using rdprops rdnot0 by simp
      hence cases: (v < 0) ∨ (v > 0) by auto

      { assume v < 0
        hence 2∗a∗r + b = −rd using v rdprops
          by (metis local.abs-if local.minus-minus)
        hence 2∗a∗r = (−b) − rd
         by (metis local.add-diff-cancel-right' local.minus-diff-commute)
        hence r = rm using aa assms(4)
          by (metis local.nonzero-mult-div-cancel-left)
      }
      hence case1: v < 0 ⟶ r = rm by auto

      { assume v > 0
        hence 2∗a∗r + b = rd using v rdprops by simp
        hence 2∗a∗r = (−b) + rd by auto
        hence r = rp using aa assms(3)
          by (metis local.nonzero-mult-div-cancel-left)
      }
      hence v > 0 ⟶ r = rp by auto

      hence r = rm ∨ r = rp using case1 cases by blast
      hence r ∈ { rm, rp } by blast
    }
    hence ∀ r . r ∈ qroots a b c ⟶ r ∈ { rm, rp } by blast
    hence l2r: qroots a b c ⊆ {rm, rp} by auto

    have rootm: qroot a b c rm
    proof −
      have rm = ((−b) − rd) / (2∗a) using assms by auto
      hence (2∗a)∗rm = (−b) − rd using aa by simp
      hence (2∗a)∗rm + b = − rd by (simp add: local.diff-add-eq)
      hence sqr( (2∗a)∗rm + b ) = sqr rd by simp
      moreover have ... = discriminant a b c
```

using *assms*(*2*) *rootd d lemSquareOfSqrt*[*of discriminant a b c rd*] **by** *auto*
**ultimately show** *?thesis*
**using** *assms lemQuadRootCondition*[*of a rm b c*] **by** *auto*
**qed**

**have** *rootp*: *qroot a b c rp*
**proof** −
**have** *rp* = ((−*b*) + *rd*) / (*2*∗*a*) **using** *assms* **by** *auto*
**hence** (*2*∗*a*)∗*rp* = (−*b*) + *rd* **using** *aa* **by** *simp*
**hence** (*2*∗*a*)∗*rp* + *b* = *rd* **by** (*simp add*: *local.diff-add-eq*)
**hence** *sqr*( (*2*∗*a*)∗*rp* + *b* ) = *sqr rd* **by** *simp*
**moreover have** ... = *discriminant a b c*
**using** *assms*(*2*) *rootd d lemSquareOfSqrt*[*of discriminant a b c rd*] **by** *auto*
**ultimately show** *?thesis*
**using** *assms lemQuadRootCondition*[*of a rp b c*] **by** *auto*
**qed**

**hence** {*rm*, *rp*} ⊆ *qroots a b c* **using** *rootm rootp* **by** *auto*
**thus** *?thesis* **using** *l2r* **by** *blast*
**qed**

**thus** *?thesis* **using** *conj1* **by** *blast*
**qed**

**lemma** *lemQuadraticRootCount*:
**assumes** ¬(*qcase1 a b c*)
**shows** *finite* (*qroots a b c*) ∧ *card* (*qroots a b c*) ≤ *2*
**proof** −
**define** *d* **where** *d*: *d* = *discriminant a b c*

**have** *case1*: *qcase1 a b c* ⟶ *?thesis* **using** *assms* **by** *auto*
**moreover have** *case2*: *qcase2 a b c* ⟶ *?thesis* **using** *lemQCase2* **by** *auto*
**moreover have** *case3*: *qcase3 a b c* ⟶ *?thesis* **using** *lemQCase3* **by** *auto*
**moreover have** *case4*: *qcase4 a b c* ⟶ *?thesis* **using** *lemQCase4* **by** *auto*
**moreover have** *case5*: *qcase5 a b c* ⟶ *?thesis* **using** *lemQCase5* **by** *auto*
**moreover have** *case6*: *qcase6 a b c* ⟶ *?thesis* **using** *lemQCase6 card-2-iff* **by** *auto*
**ultimately show** *?thesis* **using** *lemQuadraticCasesComplete* **by** *blast*
**qed**

**end**

**end**

# 30 Classification

This theory explains how to establish whether a point lies inside, on or outside a cone.

**theory** *Classification*
**imports** *Cones Quadratics CauchySchwarz*
**begin**

We want to establish where a point lies in relation to a cone, and will later show that this relationship is preserved under relevant affine transformations. We therefore need a classification scheme that relies on purely affine concepts. To do this we consider lines that can be drawn through the point, and ask how many points lie in the intersection of such a line and the cone.

**class** *Classification* = *Cones* + *Quadratics* + *CauchySchwarz*
**begin**

**abbreviation** *vertex* :: $'a$ *Point* ⇒ $'a$ *Point* ⇒ *bool*
**where** *vertex x p* ≡ $(x = p)$

**abbreviation** *insideRegularCone* :: $'a$ *Point* ⇒ $'a$ *Point* ⇒ *bool*
**where** *insideRegularCone x p* ≡
(*slopeFinite x p*) ∧ (∃ *v* ∈ *lineVelocity* (*lineJoining x p*) . *sNorm2 v* < *1*)

**abbreviation** *outsideRegularCone* :: $'a$ *Point* ⇒ $'a$ *Point* ⇒ *bool*
**where** *outsideRegularCone x p* ≡
$(x \neq p)$ ∧
((*slopeInfinite x p*) ∨ (∃ *v* ∈ *lineVelocity* (*lineJoining x p*) . *sNorm2 v* > *1*))

**abbreviation** *onRegularCone* :: $'a$ *Point* ⇒ $'a$ *Point* ⇒ *bool*
**where** *onRegularCone x p* ≡ $(x = p)$ ∨ (∃ *v* ∈ *lineVelocity* (*lineJoining x p*) . *sNorm2 v* = *1*)

**lemma** *lemDrtnLineJoining*:
**assumes** $l = lineJoining\ x\ p$
**and** $x \neq p$
**shows** $(p \ominus x) \in drtn\ l$
**proof** −
**define** $d$ **where** $d$: $d = (p \ominus x)$
**have** *lprops*: $onLine\ x\ l \wedge onLine\ p\ l$
**using** *assms(1) lemLineJoiningContainsEndPoints* **by** *blast*
**hence** $\exists\ x\ p\ .\ (x \neq p) \wedge (onLine\ x\ l) \wedge (onLine\ p\ l) \wedge (d = (p \ominus x))$
**using** *assms(2) d* **by** *blast*
**thus** *?thesis* **using** *d* **by** *auto*
**qed**

**lemma** *lemVelocityLineJoining*:
**assumes** $l = lineJoining\ x\ p$
**and** $v = velocityJoining\ origin\ (p \ominus x)$
**and** $x \neq p$
**shows** $v \in lineVelocity\ l$
**proof** −
**define** $d$ **where** $d$: $d = (p \ominus x)$
**hence** $d \in drtn\ l$ **using** *assms lemDrtnLineJoining* **by** *auto*
**hence** $\exists\ d \in drtn\ l\ .\ v = velocityJoining\ origin\ d$ **using** *assms d* **by** *blast*
**thus** *?thesis* **by** *auto*
**qed**

**lemma** *lemSlopeLineJoining*:
**assumes** $l = lineJoining\ p\ q$
**and** $p \neq q$
**shows** $lineSlopeFinite\ l \longleftrightarrow slopeFinite\ p\ q$
**proof** −
**have** *pql*: $onLine\ p\ l \wedge onLine\ q\ l$
**using** *assms(1) lemLineJoiningContainsEndPoints* **by** *auto*

**have** *l2r*: $lineSlopeFinite\ l \longrightarrow slopeFinite\ p\ q$
**proof** −
{ **assume** *lineSlopeFinite l*
**then obtain** $x\ y$
**where** *xy*: $(onLine\ x\ l) \wedge (onLine\ y\ l) \wedge (x \neq y) \wedge (slopeFinite\ x\ y)$ **by** *blast*
**hence** *lxy*: $l = lineJoining\ x\ y$ **using** *lemLineAndPoints[of x y l]*

**by** *auto*

**define** *tdiff* **where** *tdiff*: $tdiff = tval\ y - tval\ x$
**hence** *tdnot0*: $tdiff \neq 0$ **using** *xy* **by** *auto*

**obtain** *a* **where** *a*: $p = (x \oplus (a \otimes (y \ominus x)))$ **using** *pql lxy* **by** *auto*
**hence** *tvalp*: $tval\ p = tval\ x + a*(tval\ y - tval\ x)$ **by** *simp*

**obtain** *b* **where** *b*: $q = (x \oplus (b \otimes (y \ominus x)))$ **using** *pql lxy* **by** *auto*
**hence** *tvalq*: $tval\ q = tval\ x + b*(tval\ y - tval\ x)$ **by** *simp*

**have** *anotb*: $b - a \neq 0$ **using** *a b assms*(*2*) **by** *auto*

**have** $tval\ q - tval\ p = (b - a)*tdiff$
**using** *tdiff tvalp tvalq*
**by** (*simp add*: *local.left-diff-distrib′*)
**hence** *slopeFinite p q* **using** *anotb tdnot0*
**by** (*metis local.diff-self local.divisors-zero*)
**}**
**thus** *?thesis* **by** *auto*
**qed**

**have** *r2l*: *slopeFinite p q* $\longrightarrow$ *lineSlopeFinite l* **using** *pql assms*(*2*)
**by** *blast*
**thus** *?thesis* **using** *l2r* **by** *blast*
**qed**

**lemma** *lemVelocityJoiningUsingPoints*:
**assumes** $p \neq q$
**shows** *velocityJoining p q* = *velocityJoining origin* $(q \ominus p)$
**proof** −
**define** *t1* **where** *t1*: $t1 = tval\ p - tval\ q$
**define** *t2* **where** *t2*: $t2 = tval\ origin - tval\ (q \ominus p)$
**define** *v1* **where** *v1*: $v1 = (p \ominus q)$
**define** *v2* **where** *v2*: $v2 = (origin \ominus (q \ominus p))$

**have** *ts*: $t1 = t2$ **using** *t1 t2* **by** *simp*

**{ assume** *slopeFinite p q*
**hence** $(tval\ origin) - (tval\ (q \ominus p)) \neq 0$ **by** *simp*
**hence** *sf2*: *slopeFinite origin* $(q \ominus p)$ **using** *diff-self* **by** *metis*
**hence** *sloper p q* = *sloper origin* $(q \ominus p)$ **using** *t2 v2 sloper.simps*
**by** *auto*
**hence** *?thesis* **by** *auto*
**}**
**hence** *sf*: *slopeFinite p q* $\longrightarrow$ *?thesis* **by** *auto*
**{ assume** *hyp*: $\neg$ (*slopeFinite p q*)

**hence** ¬ (*slopeFinite origin* ($q \ominus p$)) **using** *t1 t2 ts* **by** *simp*
**hence** *sloper p q* = *sloper origin* ($q \ominus p$) **using** *hyp* **by** *simp*
**hence** *?thesis* **by** *auto*
**}**
**thus** *?thesis* **using** *sf* **by** *blast*
**qed**

**lemma** *lemLineVelocityNonZeroImpliesFinite*:
**assumes** $u \in$ *lineVelocity l*
**and** *sNorm2 u* $\neq$ *0*
**shows** *lineSlopeFinite l*
**proof** −
**have** $u \in \{ u . \exists d \in$ *drtn l* . $u =$ *velocityJoining origin d* $\}$ **using** *assms*(*1*) **by** *auto*
**then obtain** *d* **where** *d*: $d \in$ *drtn l* $\wedge$ $u =$ *velocityJoining origin d* **by** *blast*
**hence** $d \in \{ d . \exists p\ q . (p \neq q) \wedge$ (*onLine p l*) $\wedge$ (*onLine q l*) $\wedge (d = (q \ominus p)) \}$
**by** *auto*
**then obtain** *p q* **where** *pq*: $(p \neq q) \wedge$ (*onLine p l*) $\wedge$ (*onLine q l*) $\wedge (d = (q \ominus p))$
**by** *blast*

**hence** *upq*: $u =$ *velocityJoining p q* **using** *lemVelocityJoiningUsingPoints d* **by** *auto*

**{ assume** *slopeInfinite p q*
**hence** *sloper p q* = *origin* **by** *simp*
**hence** $u =$ *sOrigin* **using** *upq* **by** *simp*
**hence** *False* **using** *assms*(*2*) **by** *auto*
**}**
**hence** *slopeFinite p q* **by** *auto*
**thus** *?thesis* **using** *pq* **by** *blast*
**qed**

**lemma** *lemLineVelocityUsingPoints*:
**assumes** *slopeFinite p q*
**and** *onLine p l* $\wedge$ *onLine q l*
**shows** *lineVelocity l* = { *velocityJoining p q* }
**proof** −
**define** *v* **where** *v*: $v =$ *velocityJoining p q*
**hence** $v'$: $v =$ *velocityJoining origin* ($q \ominus p$)
**using** *lemVelocityJoiningUsingPoints*[*of p q*] *assms*(*1*) **by** *blast*

**have** *pnotq*: $p \neq q$ **using** *assms*(*1*) **by** *auto*
**hence** *l*: $l =$ *lineJoining p q* **using** *lemLineAndPoints*[*of p q l*] *assms*
**by** *auto*

**hence** *vinlv*: $v \in$ *lineVelocity l*
**using** *lemVelocityLineJoining*[*of l p q v*] $v'$ *assms* **by** *blast*
**hence** *r2l*: $\{v\} \subseteq$ *lineVelocity l* **by** *blast*

**{ fix** *u* **assume** *u*: $u \in$ *lineVelocity l*
**hence** $u = v$
**using** *vinlv pnotq assms lemFiniteLineVelocityUnique*[*of u l v*] **by** *blast*
**}**
**hence** *lineVelocity l* $\subseteq \{v\}$ **by** *blast*

**thus** *?thesis* **using** *r2l v* **by** *blast*
**qed**

**lemma** *lemSNorm2VelocityJoining*:
**assumes** *slopeFinite x p*
**and** $v$ = *velocityJoining x p*
**shows** *sqr* (*tval p* $-$ *tval x*) $*$ *sNorm2 v* = *sNorm2* (*sComponent* ($p \ominus x$))
**proof** $-$
**have** *sloper x p* = ((*1* / (*tval x* $-$ *tval p*)) $\otimes$ ($x \ominus p$)) **using** *assms*(*1*) **by** *auto*
**hence** $v$ = ((*1*/(*tval x* $-$ *tval p*))$\otimes$*s* (*sComponent*($x \ominus p$))) **using** *assms*(*2*) **by** *simp*
**hence** *sNorm2 v* = *sqr* (*1*/ (*tval x* $-$ *tval p*)) $*$ *sNorm2* (*sComponent* ($x \ominus p$))
**using** *lemSNorm2OfScaled assms*(*1*) **by** *blast*
**also have** ... = *sqr* (*1*/ (*tval p* $-$ *tval x*)) $*$ *sNorm2* (*sComponent* ($p \ominus x$))
**using** *lemSSep2Symmetry assms*(*1*) *lemSqrDiffSymmetrical* **by** *simp*
**finally show** *?thesis* **using** *assms*(*1*) **by** *simp*
**qed**

**lemma** *lemOrthogalSpaceVectorExists*:
**shows** $\exists$ *w* . ($w \neq$ *sOrigin*) $\wedge$ ($w \odot s\ v$) = *0*
**proof** $-$
**obtain** *x y z* **where** *xyz*: $v$ = *mkSpace x y z* **using** *Space.cases* **by** *blast*
**define** *w* **where** *w*: $w$ = (*if x = 0 then* (*mkSpace 1 0 0*)
*else* (*mkSpace* (*y*/*x*) ($-$*1*) *0*))

**have** *wnot0*: ($w \neq$ *sOrigin*) **using** *w* **by** *simp*

**moreover have** *orth*: ($w \odot s\ v$) = *0*
**proof** $-$

{ **assume** $x0$: $x = 0$
**hence** $w = mkSpace\ 1\ 0\ 0$ **using** $w$ **by** *simp*
**hence** $(w \odot s\ v) = 0$ **using** $x0\ xyz$ **by** *simp*
}
**hence** $case0$: $x = 0 \longrightarrow ?thesis$ **by** *blast*
{ **assume** $xnot0$: $x \neq 0$
**hence** $w = mkSpace\ (y/x)\ (-1)\ 0$ **using** $w$ **by** *simp*
**hence** $(w \odot s\ v) = 0$ **using** $xnot0\ xyz$ **by** *simp*
}
**hence** $x \neq 0 \longrightarrow ?thesis$ **by** *blast*
**thus** $?thesis$ **using** $case0$ **by** *blast*
**qed**
**ultimately show** $?thesis$ **by** *force*
**qed**

**lemma** *lemNonParallelVectorsExist*:
**shows** $\exists\ w\ .\ ((w \neq origin) \wedge (tval\ v = tval\ w)) \wedge (\neg\ (\exists\ \alpha\ .\ (\alpha \neq 0) \wedge v = (\alpha \otimes w)))$
**proof** −
**have** $cases$: $xval\ v = 0 \vee xval\ v \neq 0$ **by** *auto*
{ **assume** $case1$: $xval\ v = 0$

**define** $diff$ **where** $diff$: $diff = (if\ ((v \oplus xUnit) = origin)\ then\ (2{\otimes}xUnit)\ else\ xUnit)$
**define** $w$ **where** $w$: $w = (v \oplus diff)$
**hence** $w1$: $(xval\ w) = 1$ **using** $case1\ diff$ **by** *auto*

{ **assume** $\exists\ \alpha\ .\ (\alpha \neq 0) \wedge v = (\alpha \otimes w)$
**then obtain** $a$ **where** $a$: $(a \neq 0) \wedge v = (a \otimes w)$ **by** *auto*
**hence** $xval\ v = a * xval\ w$ **by** *simp*
**hence** $0 = a * 1$ **using** $case1\ w1$ **by** *auto*
**hence** $a = 0$ **by** *auto*
**hence** $False$ **using** $a$ **by** *blast*
}
**hence** $(\neg\ (\exists\ \alpha\ .\ (\alpha \neq 0) \wedge v = (\alpha \otimes w)))$ **by** *auto*
**moreover have** $tval\ v = tval\ w$ **using** $w\ diff$ **by** *auto*
**ultimately have** $(w \neq origin) \wedge (tval\ v = tval\ w) \wedge (\neg\ (\exists\ \alpha\ .\ (\alpha \neq 0) \wedge v = (\alpha \otimes w)))$
**using** $w1$ **by** *auto*
}
**hence** $lhs$: $xval\ v = 0 \longrightarrow ?thesis$ **by** *blast*

{ **assume** $case2$: $xval\ v \neq 0$
**define** $w$ **where** $w$: $w = (v \oplus yUnit)$
**hence** $wx$: $xval\ w = xval\ v$ **using** $case2$ **by** *auto*
**have** $wy$: $yval\ w = yval\ v + 1$ **using** $w$ **by** *auto*

{ **assume** $\exists\ \alpha\ .\ (\alpha \neq 0) \wedge v = (\alpha \otimes w)$

```
      then obtain a where a: (a ≠ 0) ∧ v = (a ⊗ w) by auto
      hence xv: xval v = a ∗ xval w by simp
      hence a1: xval v = a ∗ xval v using wx by simp
      hence a = 1 using case2 by simp
      hence yval v = yval w using a by auto
      hence False using wy by auto
    }
    hence (¬ (∃ α . (α ≠ 0) ∧ v = (α ⊗ w))) by auto
    moreover have tval v = tval w using w by auto
    moreover have xval w ≠ 0 using w case2 by auto
    ultimately have (w ≠ origin) ∧(tval v = tval w) ∧ (¬ (∃ α . (α
≠ 0) ∧ v = (α ⊗ w)))
      by auto
  }
  hence rhs: xval v ≠ 0 ⟶ ?thesis by blast

  thus ?thesis using cases lhs by auto
qed


lemma lemConeContainsVertex:
  shows regularCone x x
proof −
  define d where d: d = (tUnit ⊕ xUnit)
  define p where p: p = (d ⊕ x)
  define l where l: l = lineJoining x p
  define v where v: v = velocityJoining origin d

  have xnotp: x ≠ p
  proof −
    { assume x = p
      hence (d ⊕ x) = x using p by auto
      hence d = origin using add-cancel-left-left
        by (metis dot.simps lemDotSumRight lemNullImpliesOrigin)
      hence False using d by auto
    }
    thus ?thesis by auto
  qed
  moreover have d = (p ⊖ x) using p by auto
  ultimately have vel: v ∈ lineVelocity l
    using l v d lemVelocityLineJoining[of l x p v] by blast

  have lprops: onLine x l ∧ onLine p l
    using xnotp l lemLineAndPoints[of x p l] by auto

  have slope: sNorm2 v = 1
  proof −
    define sx where sx: sx = (| svalx = 1, svaly = 0, svalz = 0 |)
    have slopeFinite origin d using d by auto
```

**hence** *sloper origin d* = ((*1* / ((*tval origin*) − (*tval d*))) ⊗ (*origin* ⊖ *d*)) **by** *simp*
**moreover have** . . . = ((−*1*) ⊗ (*origin* ⊖ *d*)) **using** *d* **by** *auto*
**moreover have** . . . = *d* **by** *auto*
**ultimately have** *sloper origin d* = *d* **by** *simp*
**hence** *velocityJoining origin d* = *sComponent d* **by** *simp*
**hence** *v* = *sx* **using** *v d sx* **by** *auto*
**thus** *?thesis* **using** *sx* **by** *auto*
**qed**

**hence** *v* ∈ *lineVelocity l* ∧ *sNorm2 v* = *1* **using** *vel* **by** *auto*
**hence** ∃ *l* . (*onLine x l*) ∧(∃ *v* ∈ *lineVelocity l* . *sNorm2 v* = *1*)
**using** *lprops* **by** *blast*
**thus** *?thesis* **by** *blast*
**qed**

**lemma** *lemConesExist*:
**shows** *regularConeSet x* ≠ {}
**proof** −
**have** *x* ∈ *regularConeSet x* **using** *lemConeContainsVertex* **by** *auto*
**thus** *?thesis* **by** *blast*
**qed**

**lemma** *lemRegularCone*:
**shows** ((*x* = *p*) ∨ *onRegularCone x p*) ⟷ *regularCone x p*
**proof** −
**define** *l* **where** *l*: *l* = *lineJoining x p*
**hence** *lprops*: *onLine p l* ∧ *onLine x l*
**using** *lemLineJoiningContainsEndPoints* **by** *auto*

**define** *LHS* **where** *LHS*: *LHS* = ((*x* = *p*) ∨ (*onRegularCone x p*))
**define** *RHS* **where** *RHS*: *RHS* = (*regularCone x p*)

**have** *LHS* ⟶ *RHS*
**proof** −
**{ assume** *x* = *p*
**hence** *?thesis* **using** *RHS lemConeContainsVertex* **by** *auto*
**}**
**hence** *case1*: *x* = *p* ⟶ *regularCone x p* **using** *LHS RHS* **by** *auto*
**{ assume** *x*≠*p* ∧ *onRegularCone x p*
**then obtain** *v* **where** *v*: *v* ∈ *lineVelocity l* ∧ *sNorm2 v* = *1* **using** *l* **by** *blast*
**hence** ∃ *l* . (*onLine p l*) ∧ (*onLine x l*) ∧ (∃ *v* ∈ *lineVelocity l* . *sNorm2 v* = *1*)
**using** *lprops* **by** *blast*
**}**
**thus** *?thesis* **using** *case1 LHS RHS* **by** *blast*

**qed**

**moreover have** *RHS* ⟶ *LHS*
**proof** −
{ **assume** *rhs*: *RHS*
**have** *cases*: $x = p \lor x \neq p$ **by** *auto*
**have** *case1*: $x = p \longrightarrow (x = p \lor onRegularCone\ x\ p)$ **by** *auto*

{ **assume** *xnotp*: $x \neq p$
**then obtain** *l1* **where**
*l1*: $(onLine\ x\ l1) \land (onLine\ p\ l1)$
$\land (\exists\ v \in lineVelocity\ l1\ .\ sNorm2\ v = 1)$
**using** *rhs RHS* **by** *blast*
**hence** $l1 = l$ **using** *xnotp l l1 lemLineAndPoints*[*of x p l1*] **by** *auto*
**hence** $\exists\ v \in lineVelocity\ l\ .\ sNorm2\ v = 1$ **using** *l1* **by** *blast*
**hence** *onRegularCone x p* **using** *l* **by** *blast*
**hence** $(x = p \lor onRegularCone\ x\ p)$ **by** *blast*
}
**hence** *case2*: $x \neq p \longrightarrow LHS$
**using** *l lprops LHS* **by** *blast*

**hence** $(x = p \lor onRegularCone\ x\ p)$ **using** *cases case1 LHS* **by** *blast*
}
**thus** *?thesis* **using** *LHS RHS* **by** *auto*
**qed**

**ultimately have** *LHS* ⟷ *RHS* **by** *blast*
**thus** *?thesis* **using** *LHS RHS* **by** *fastforce*
**qed**

**lemma** *lemSlopeInfiniteImpliesOutside*:
**assumes** $x \neq p$
**and** *slopeInfinite x p*
**shows** $\exists\ l\ p'\ .\ (p' \neq p) \land onLine\ p'\ l \land onLine\ p\ l$
$\land (l \cap regularConeSet\ x = \{\})$
**proof** −
**define** *dxp* **where** *dxp*: $dxp = (x \ominus p)$
**hence** $x = (dxp \oplus p)$ **by** *simp*
**hence** *xdxp*: $x = (p \oplus dxp)$ **using** *add-commute* **by** *blast*

**have** *xp*: $tval\ x = tval\ p$ **using** *assms*(*2*) **by** *blast*
**hence** *tvaldxp*: $tval\ dxp = 0$ **using** *dxp* **by** *simp*

**obtain** *dnew* **where**
*dnew*: $(dnew \neq origin) \land (tval\ dnew = tval\ dxp) \land \neg(\exists\ \alpha.\ \alpha \neq 0 \land dxp = (\alpha \otimes dnew))$

using *lemNonParallelVectorsExist*[*of dxp*]
by *auto*
**hence** *tvaldnew*: *tval dnew* = *0* **using** *tvaldxp* **by** *simp*

**define** *w* **where** *w*: *w* = (*p* ⊕ *dnew*)
**hence** *wmp*: (*w* ⊖ *p*) = *dnew* **by** *simp*

**have** *wx*: *tval w* = *tval x*
**proof** −
**have** *tval dnew* = *tval x* − *tval p* **using** *dnew dxp* **by** *auto*
**hence** *tval w* = *tval p* + (*tval x* − *tval p*) **using** *w* **by** *auto*
**thus** *?thesis* **using** *add-commute diff-add-cancel* **by** *auto*
**qed**

**define** *lw* **where** *lw*: *lw* = *lineJoining w p*

**have** *xNotOnLw*: ¬ (*x* ∈ *lw*)
**proof** −
**{ assume** *x* ∈ *lw*
**then obtain** *a* **where** *a*: *x* = (*w* ⊕ (*a* ⊗ (*p*⊖*w*))) **using** *lw* **by** *auto*
**hence** (*p* ⊕ *dxp*) = ((*p* ⊕ *dnew*) ⊕ (*a* ⊗ (*p*⊖*w*))) **using** *xdxp w* **by** *auto*
**hence** *dxp* = (*dnew* ⊕ (*a* ⊗ (*p*⊖*w*))) **using** *add-assoc* **by** *auto*
**moreover have** (*p*⊖*w*) = ((−*1*) ⊗ (*w*⊖*p*)) **by** *simp*
**hence** (*a* ⊗ (*p*⊖*w*)) = ((−*a*) ⊗ (*w*⊖*p*)) **using** *lemScaleAssoc*[*of a* −*1 w*⊖*p*] **by** *simp*
**ultimately have** *dxp* = (*dnew* ⊕ ((−*a*) ⊗ (*w*⊖*p*))) **by** *auto*
**hence** *dxp* = ((*1* ⊗ *dnew*) ⊕ ((−*a*) ⊗ *dnew*)) **using** *wmp* **by** *auto*
**hence** *dxp* = ((*1*−*a*) ⊗ *dnew*) **using** *left-diff-distrib′* **by** *fastforce*
**hence** (*1*−*a*) = *0* **using** *dnew* **by** *blast*
**hence** *a* = *1* **by** *simp*
**hence** *x* = (*w* ⊕ (*p* ⊖ *w*)) **using** *a* **by** *auto*
**hence** *x* = *p* **by** (*simp add*: *local.add-diff-eq*)
**}**
**thus** *?thesis* **using** *assms*(*1*) **by** *auto*
**qed**

**have** *dnew* ≠ *origin* **using** *dnew* **by** *auto*
**hence** *wNotp*: *w* ≠ *p* **using** *w diff-self wmp* **by** *blast*
**hence** *pwOnLw*: *onLine p lw* ∧ *onLine w lw*
**using** *lw lemLineAndPoints*[*of w p lw*] **by** *auto*

**hence** *target1*: *w* ≠ *p* ∧ *onLine w lw* ∧ *onLine p lw* **using** *wNotp* **by** *auto*

**define** *MeetW* **where** *MeetW*: *MeetW* = *lw* ∩ *regularConeSet x*
**{ assume** *nonempty*: ¬ (*MeetW* = {})

**then obtain** *z* **where** *z*: $z \in MeetW$ **by** *blast*

**have** *zx*: *tval z = tval x*
**proof** −
**have** $z \in$ *lineJoining w p* **using** *z MeetW lw* **by** *auto*
**then obtain** *a* **where** *a*: $z = (w \oplus (a \otimes (p \ominus w)))$ **by** *blast*

**have** *tval* $(p \ominus w) = 0$ **using** *w tvaldnew* **by** *auto*
**hence** *tval z = tval w* **using** *a* **by** *auto*
**thus** *?thesis* **using** *wx* **by** *auto*
**qed**

**have** *regularCone x z* **using** *z MeetW* **by** *auto*
**then obtain** *l1* **where** *l1*: (*onLine z l1*) ∧ (*onLine x l1*)
∧ (∃ *v* ∈ *lineVelocity l1* . *sNorm2 v* =
*1*) **by** *blast*
**then obtain** *v* **where** *v*: *v* ∈ *lineVelocity l1* ∧ *sNorm2 v* = *1* **by**
*blast*
**hence** ∃ *d* ∈ *drtn l1* . *v* = *velocityJoining origin d* ∧ *sNorm2 v* =
*1* **by** *auto*
**then obtain** *d1* **where** *d1*: *d1* ∈ *drtn l1* ∧ *v* = *velocityJoining*
*origin d1* ∧ *sNorm2 v* = *1*
**by** *blast*
**hence** *v* ≠ *sOrigin* **by** *fastforce*
**hence** *velocityJoining origin d1* ≠ *sOrigin* **using** *d1* **by** *auto*
**hence** *drtnNotZero*: *tval d1* ≠ *0* **by** *auto*

**define** *d2* **where** *d2*: *d2* = (*z* ⊖ *x*)
**hence** *tvald2*: *tval d2* = *0* **using** *zx* **by** *simp*
**have** *zNotz*: *x* ≠ *z* **using** *xNotOnLw z MeetW* **by** *blast*
**hence** (*x* ≠ *z*) ∧ (*onLine z l1*) ∧ (*onLine x l1*) ∧ (*d2* = (*z* ⊖ *x*))
**using** *l1 d2* **by** *auto*
**hence** ∃ *x z* . (*x* ≠ *z*) ∧ (*onLine x l1*) ∧ (*onLine z l1*) ∧ (*d2* = (*z*
⊖ *x*)) **by** *blast*
**hence** *d2* ∈ *drtn l1* **by** *auto*

**then obtain** *b* **where** *b*: *b* ≠ *0* ∧ *d1* = (*b* ⊗ *d2*)
**using** *lemDrtn*[*of d2 d1 l1*] *d1* **by** *blast*
**hence** *tval d1* = *b* ∗ *tval d2* **by** *simp*
**hence** *tval d1* = *0* **using** *tvald2* **by** *simp*

**hence** *False* **using** *drtnNotZero* **by** *auto*
**}**
**hence** *MeetW* = {} **by** *auto*

**hence** (*w* ≠ *p*) ∧ *onLine w lw* ∧ *onLine p lw* ∧ (*lw* ∩ *regularConeSet*
*x* = {})
**using** *target1 MeetW* **by** *auto*

**thus** *?thesis* **by** *blast*
**qed**

**lemma** *lemClassification*:
**shows** (*insideRegularCone x p*) $\vee$ (*vertex x p* $\vee$ *outsideRegularCone x p* $\vee$ *onRegularCone x p*)
**proof** −
**define** *l* **where** *l*: *l* = *lineJoining x p*
**define** *v* **where** *v*: *v* = *velocityJoining origin* (*p*$\ominus$*x*)
**{ assume** *xnotp*: *x* $\neq$ *p*
**hence** *vel*: *v* $\in$ *lineVelocity l*
**using** *l v lemVelocityLineJoining*[*of l x p v*] **by** *auto*
**have** (*sNorm2 v* < *1*) $\vee$ (*sNorm2 v* > *1*) $\vee$ (*sNorm2 v* = *1*) **by** *auto*
**hence** *?thesis* **using** *xnotp l v vel* **by** *blast*
**}**
**hence** *x* $\neq$ *p* $\longrightarrow$ *?thesis* **by** *auto*
**moreover have** *x* = *p* $\longrightarrow$ *?thesis* **by** *auto*
**ultimately show** *?thesis* **by** *blast*
**qed**

**lemma** *lemQuadCoordinates*:
**assumes** *p* = (*B* $\oplus$ ($\alpha$ $\otimes$ *D*))
**and** *a* = *mNorm2 D*
**and** *b* = *2*∗(*tval* (*B*$\ominus$*x*))∗(*tval D*) − *2*∗((*sComponent D*) $\odot$*s* (*sComponent* (*B*$\ominus$*x*)))
**and** *c* = *mNorm2* (*B*$\ominus$*x*)
**shows** *sqr* (*tval* (*p*$\ominus$*x*)) − *sNorm2* (*sComponent* (*p*$\ominus$*x*)) = *a*∗(*sqr* $\alpha$) + *b*∗$\alpha$ + *c*
**proof** −
**define** *X* **where** *X*: *X* = (*B*$\ominus$*x*)
**have** *pmx*: (*p* $\ominus$ *x*) = (*X* $\oplus$ ($\alpha$ $\otimes$ *D*)) **using** *diff-add-eq assms X* **by** *simp*

**have** *pmxt*: *tval p* − *tval x* = *tval X* + $\alpha$∗*tval D* **using** *pmx* **by** *simp*
**have** *pmxs*: *sComponent* (*p*$\ominus$*x*) = ((*sComponent X*) $\oplus$*s* ($\alpha$ $\otimes$*s* (*sComponent D*)))
**using** *pmx* **by** *simp*

**have** *tsqr*: *sqr* (*tval* (*p*$\ominus$*x*))
= *sqr* (*tval X*) + $\alpha$∗(*2*∗(*tval X*)∗(*tval D*)) + (*sqr* $\alpha$)∗(*sqr* (*tval D*))
**using** *pmxt lemSqrSum*[*of tval X* $\alpha$∗(*tval D*)] *mult-assoc mult-commute* **by** *auto*

**have** *ssqr*: *sNorm2* (*sComponent* ($p \ominus x$))
= (*sNorm2* (*sComponent* *X*))
+ $\alpha *$(*2*$*$((*sComponent* *X*) $\odot s$ (*sComponent* *D*)))
+ (*sqr* $\alpha$)$*$(*sNorm2* (*sComponent* *D*))
**using** *lemSDotScaleRight lemSNorm2OfScaled lemSNorm2OfSum mult.left-commute pmxs*
**by** *presburger*

**hence** *sqr* (*tval* ($p \ominus x$)) $-$ *sNorm2* (*sComponent* ($p \ominus x$))
= ( *sqr* (*tval* *X*) + $\alpha *$(*2*$*$(*tval* *X*)$*$(*tval* *D*)) + (*sqr* $\alpha$)$*$(*sqr* (*tval* *D*)) )
$-$ ((*sNorm2* (*sComponent* *X*))
+ $\alpha *$(*2*$*$((*sComponent* *X*) $\odot s$ (*sComponent* *D*)))
+ (*sqr* $\alpha$)$*$(*sNorm2* (*sComponent* *D*)) )
**using** *tsqr* **by** *auto*
**also have** ...
= ( *sqr* (*tval* *X*) + $\alpha *$(*2*$*$(*tval* *X*)$*$(*tval* *D*)) )
+ ( (*sqr* $\alpha$)$*$(*sqr* (*tval* *D*)) $-$ (*sqr* $\alpha$)$*$(*sNorm2* (*sComponent* *D*)) )
$-$ ((*sNorm2* (*sComponent* *X*))
+ $\alpha *$(*2*$*$((*sComponent* *X*) $\odot s$ (*sComponent* *D*))))
**using** *diff-add-eq add-diff-eq diff-add-eq-diff-diff-swap* **by** *fastforce*
**also have** ...
= *sqr* (*tval* *X*) +
( $\alpha *$(*2*$*$(*tval* *X*)$*$(*tval* *D*)) $-$ $\alpha *$(*2*$*$((*sComponent* *X*) $\odot s$ (*sComponent* *D*))) )
+ ( (*sqr* $\alpha$)$*$(*sqr* (*tval* *D*)) $-$ (*sqr* $\alpha$)$*$(*sNorm2* (*sComponent* *D*)) )
$-$ (*sNorm2* (*sComponent* *X*))
**using** *diff-add-eq add-diff-eq diff-add-eq-diff-diff-swap add-commute* **by** *simp*
**also have** ...
= *sqr* (*tval* *X*) + $\alpha * b$ + (*sqr* $\alpha$)$*$ *a* $-$ (*sNorm2* (*sComponent* *X*))
**using** *right-diff-distrib′ assms*(*2*) *assms*(*3*) *X lemSDotCommute* **by** *presburger*
**also have** ... = *c* + $\alpha * b$ + (*sqr* $\alpha$)$*a$
**using** *right-diff-distrib′ assms*(*4*) *X add-commute add-diff-eq* **by** *simp*

**finally show** *?thesis* **using** *add-commute mult-commute add-assoc* **by** *auto*
**qed**

**lemma** *lemConeCoordinates*:
**shows** (*onRegularCone* *x* *p* $\longleftrightarrow$ *sqr* (*tval* *p* $-$ *tval* *x*) = *sNorm2* (*sComponent* ($p \ominus x$)))

∧ ( *insideRegularCone x p* ⟷ *sqr* (*tval p* − *tval x*) > *sNorm2* (*sComponent* (*p*⊖*x*)))
∧ ( *outsideRegularCone x p* ⟷ *sqr* (*tval p* − *tval x*) < *sNorm2* (*sComponent* (*p*⊖*x*)))
**proof** −
**define** *tdiff* **where** *tdiff*: *tdiff* = *tval p* − *tval x*
**define** *sdiff* **where** *sdiff*: *sdiff* = *sComponent* (*p*⊖*x*)

**have** *cases*: *x* = *p* ∨ *x* ≠ *p* **by** *simp*
**have** *case1*: *x* = *p* ⟶ *?thesis*
**proof** −
**{ assume** *xisp*: *x* = *p*
**hence** *on*: *onRegularCone x p* **by** *auto*
**moreover have** *both0*: *sqr tdiff* = *0* ∧ *sNorm2 sdiff* = *0*
**using** *xisp tdiff sdiff* **by** *simp*
**ultimately have** *onRegularCone x p* ⟷ *sqr tdiff* = *sNorm2 sdiff* **by** *simp*

**moreover have** *outsideRegularCone x p* ⟷ *sqr tdiff* > *sNorm2 sdiff*
**proof** −
**have** ¬*outsideRegularCone x p* **using** *xisp* **by** *simp*
**moreover have** ¬ (*sqr tdiff* > *sNorm2 sdiff*) **using** *both0* **by** *simp*
**ultimately show** *?thesis* **by** *blast*
**qed**

**moreover have** *insideRegularCone x p* ⟷ *sqr tdiff* < *sNorm2 sdiff*
**proof** −
**have** ¬*insideRegularCone x p* **using** *xisp* **by** *simp*
**moreover have** ¬ (*sqr tdiff* < *sNorm2 sdiff*) **using** *both0* **by** *simp*
**ultimately show** *?thesis* **by** *blast*
**qed**

**ultimately have** *?thesis* **using** *tdiff sdiff* **by** *blast*
**}**
**thus** *?thesis* **by** *blast*
**qed**

**have** *case2*: *x* ≠ *p* ⟶ *?thesis*
**proof** −
**define** *l* **where** *l*: *l* = *lineJoining x p*
**hence** *onl*: *onLine x l* ∧ *onLine p l* **using** *lemLineJoiningContainsEndPoints* **by** *blast*
**define** *v* **where** *v*: *v* = *velocityJoining x p*

{ **assume** *xnotp*: $x \neq p$

{ **assume** *sinf*: *slopeInfinite x p*

**hence** *t0*: *sqr tdiff* = *0* **using** *tdiff* **by** *simp*
**hence** *sdiff* ≠ *sOrigin* **using** *xnotp sdiff tdiff* **by** *auto*
**hence** *sNorm2 sdiff* ≠ *0* **using** *lemSpatialNullImpliesSpatialOrigin* **by** *blast*
**moreover have** *sNorm2 sdiff* ≥ *0* **by** *simp*
**ultimately have** *sNorm2 sdiff* > *0* **using** *lemGENZGT* **by** *auto*

**hence** *eqn*: *sqr tdiff* < *sNorm2 sdiff* **using** *t0* **by** *auto*
**have** *out*: *outsideRegularCone x p* **using** *sinf xnotp* **by** *blast*

**have** *notin*: ¬ *insideRegularCone x p* **using** *sinf* **by** *blast*
**have** *notgt*: ¬ (*sqr tdiff* > *sNorm2 sdiff*) **using** *eqn* **by** *auto*

**have** *noton*: ¬ *onRegularCone x p*
**proof** −
{ **assume** *onRegularCone x p*
**then obtain** *u* **where** *u*: *u* ∈ *lineVelocity l* ∧ *sNorm2 u* = *1*
**using** *l xnotp* **by** *blast*
**hence** *slopeFinite x p*
**using** *xnotp lemLineVelocityNonZeroImpliesFinite*[*of u l*]
*zero-neq-one l*
**by** *fastforce*
**hence** *False* **using** *sinf* **by** *auto*
}
**thus** *?thesis* **by** *blast*
**qed**
**have** *noteq*: ¬ (*sqr tdiff* = *sNorm2 sdiff*) **using** *eqn* **by** *auto*

**have** *outs*: (*outsideRegularCone x p*) ⟷ (*sqr tdiff* < *sNorm2 sdiff*)
**using** *out eqn* **by** *blast*
**have** *ins*: (*insideRegularCone x p*) ⟷ (*sqr tdiff* > *sNorm2 sdiff*)
**using** *notin notgt* **by** *blast*
**have** *ons*: (*onRegularCone x p*) ⟷ (*sqr tdiff* = *sNorm2 sdiff*)

**using** *noton noteq* **by** *blast*

**hence** *?thesis* **using** *ins outs ons tdiff sdiff* **by** *blast*
}
**hence** *ifsinf*: *slopeInfinite x p* ⟶ *?thesis* **by** *blast*

**{ assume** *sf*: *slopeFinite x p*
**hence** *lv*: *lineVelocity l* = {*v*}
**using** *lemLineVelocityUsingPoints*[*of x p l*] *v onl xnotp* **by** *auto*
**have** *formula*: *sqr tdiff* ∗(*sNorm2 v*) = *sNorm2 sdiff*
**using** *lemSNorm2VelocityJoining*[*of x p v*] *sf v tdiff sdiff* **by** *auto*

**{ assume** *onRegularCone x p*
**hence** (∃ *v* ∈ *lineVelocity l* . *sNorm2 v* = *1*) **using** *xnotp l* **by** *auto*
**then obtain** *u* **where** *u*: *u* ∈ *lineVelocity l* ∧ *sNorm2 u* = *1* **by** *blast*
**hence** *u* = *v* **using** *lv* **by** *blast*
**hence** *sNorm2 v* = *1* **using** *u* **by** *auto*
**hence** *sqr tdiff* = *sNorm2 sdiff* **using** *formula* **by** *auto*
**}**
**hence** *on1*: (*onRegularCone x p*) ⟶ (*sqr tdiff* = *sNorm2 sdiff*) **by** *auto*

**{ assume** *insideRegularCone x p*
**hence** (∃ *v* ∈ *lineVelocity l* . *sNorm2 v* < *1*) **using** *xnotp l* **by** *auto*
**then obtain** *u* **where** *u*: *u* ∈ *lineVelocity l* ∧ *sNorm2 u* < *1* **by** *blast*
**hence** *u* = *v* **using** *lv* **by** *blast*
**hence** *vlt1*: *sNorm2 v* < *1* **using** *u* **by** *auto*

**{ assume** *sNorm2 v* = *0*
**hence** *v0*: *v* = *sOrigin* **using** *lemSpatialNullImpliesSpatialOrigin* **by** *auto*
**have** *sloper x p* = ((*1*/(*tval x* − *tval p*))⊗(*x*⊖*p*)) **using** *sf* **by** *auto*
**hence** *v* = ((*1*/(*tval x* − *tval p*))⊗*s* (*sComponent* (*x*⊖*p*))) **using** *v* **by** *simp*
**hence** *sOrigin* = ((*1*/(*tval x* − *tval p*))⊗*s* (*sComponent* (*x*⊖*p*)))
**using** *v0* **by** *force*
**hence** ((*tval x* − *tval p*) ⊗*s sOrigin*) = *sComponent* (*x*⊖*p*)
**using** *lemSScaleAssoc*[*of* (*tval x* − *tval p*) *1*/(*tval x* − *tval p*)
(*sComponent* (*x*⊖*p*))] *sf*
*mult-eq-0-iff right-minus-eq* **by** *auto*
**hence** *s0*: *sComponent* (*x*⊖*p*) = *sOrigin* **by** *auto*
**hence** *pmxs*: *sNorm2 sdiff* = *0* **using** *sdiff lemSSep2Symmetry* **by** *auto*

**have** *tdiff* ≠ *0* **using** *tdiff xnotp s0* **by** *auto*

**hence** *sqr tdiff* > *sNorm2 sdiff* **using** *pmxs lemSquaresPositive* **by** *auto*
**}**
**hence** *ifv0*: *sNorm2 v* = *0* ⟶ *sqr tdiff* > *sNorm2 sdiff* **by** *blast*

**{ assume** *vne0*: *sNorm2 v* ≠ *0*
**hence** *sNorm2 v* > *0* **using** *lemGENZGT* **by** *auto*
**moreover have** *tpos*: *sqr tdiff* > *0*
**using** *sf lemSquaresPositive tdiff* **by** *auto*
**ultimately have** *lpos*: (*sqr tdiff*)∗(*sNorm2 v*) > *0* **by** *auto*
**hence** *rpos*: *sNorm2 sdiff* > *0* **using** *formula* **by** *auto*

**hence** (*sqr tdiff*)∗(*sNorm2 v*) < (*sqr tdiff*) **using** *tpos lpos vlt1*
**using** *lemMultPosLT1*[*of sqr tdiff sNorm2 v*] *tpos* **by** *auto*
**hence** *sqr tdiff* > *sNorm2 sdiff* **using** *formula* **by** *auto*
**}**
**hence** *sNorm2 v* ≠ *0* ⟶ *sqr tdiff* > *sNorm2 sdiff* **by** *auto*

**hence** *sqr tdiff* > *sNorm2 sdiff* **using** *ifv0* **by** *blast*
**}**
**hence** *in1*: *insideRegularCone x p* ⟶ *sqr tdiff* > *sNorm2 sdiff* **by** *auto*

**{ assume** *out*: *outsideRegularCone x p*
**have** *xnotp*: (*x* ≠ *p*) **using** *out* **by** *simp*
**have** (∃ *v* ∈ *lineVelocity* (*lineJoining x p*) . *sNorm2 v* > *1*)
**using** *sf out* **by** *blast*
**then obtain** *u* **where** *u*: *u* ∈ *lineVelocity* (*lineJoining x p*) ∧ (*sNorm2 u* > *1*)
**by** *blast*
**hence** *u* = *v* **using** *lv l* **by** *blast*
**hence** *sNorm2 v* > *1* **using** *u* **by** *auto*
**moreover have** *sqr tdiff* > *0* **using** *sf tdiff lemSquaresPositive* **by** *auto*
**ultimately have** (*sqr tdiff*)∗(*sNorm2 v*) > (*sqr tdiff*)
**using** *local.mult-strict-left-mono* **by** *fastforce*
**hence** *sqr tdiff* < *sNorm2 sdiff* **using** *formula* **by** *auto*
**}**
**hence** *out1*: (*outsideRegularCone x p*) ⟶ (*sqr tdiff* < *sNorm2 sdiff*) **by** *auto*

**have** *in2*: (*sqr tdiff* > *sNorm2 sdiff*) ⟶ (*insideRegularCone x p*)
**proof** −
**{ assume** *lhs*: *sqr tdiff* > *sNorm2 sdiff*
**{ assume** ¬ *insideRegularCone x p*

```
          hence options: onRegularCone x p ∨ outsideRegularCone
x p
            using lemClassification xnotp by blast

          { assume onRegularCone x p
          hence sqr tdiff = sNorm2 sdiff using xnotp on1 by blast
            hence False using lhs by auto
          }
          hence notOn: ¬onRegularCone x p by blast

          { assume outsideRegularCone x p
              hence sqr tdiff < sNorm2 sdiff using xnotp out1 by
blast
            hence False using lhs by auto
          }
          hence notIn: ¬outsideRegularCone x p by blast

          hence False using notOn options by blast
        }
        hence insideRegularCone x p by blast
      }
      thus ?thesis by blast
    qed

    have out2: (sqr tdiff < sNorm2 sdiff) ⟶ (outsideRegularCone
x p)
    proof −
      { assume lhs: sqr tdiff < sNorm2 sdiff
        { assume ¬ outsideRegularCone x p
          hence options: onRegularCone x p ∨ insideRegularCone x
p
            using lemClassification xnotp by blast

          { assume onRegularCone x p
          hence sqr tdiff = sNorm2 sdiff using xnotp on1 by blast
            hence False using lhs by auto
          }
          hence notOn: ¬onRegularCone x p by blast

          { assume insideRegularCone x p
          hence sqr tdiff > sNorm2 sdiff using xnotp in1 by blast
            hence False using lhs by auto
          }
          hence notIn: ¬insideRegularCone x p by blast

          hence False using notOn options by blast
        }
        hence outsideRegularCone x p by blast
      }
```

**thus** *?thesis* **by** *blast*
**qed**

**have** *on2*: (*sqr tdiff* = *sNorm2 sdiff*) ⟶ (*onRegularCone x p*)
**proof** −
{ **assume** *lhs*: *sqr tdiff* = *sNorm2 sdiff*
{ **assume** ¬ *onRegularCone x p*
**hence** *options*: *outsideRegularCone x p* ∨ *insideRegularCone x p*
**using** *lemClassification xnotp* **by** *blast*

{ **assume** *outsideRegularCone x p*
**hence** *sqr tdiff* < *sNorm2 sdiff* **using** *xnotp out1* **by** *blast*
**hence** *False* **using** *lhs* **by** *auto*
}
**hence** *notOut*: ¬*outsideRegularCone x p* **by** *blast*

{ **assume** *insideRegularCone x p*
**hence** *sqr tdiff* > *sNorm2 sdiff* **using** *xnotp in1* **by** *blast*
**hence** *False* **using** *lhs* **by** *auto*
}
**hence** *notIn*: ¬*insideRegularCone x p* **by** *blast*

**hence** *False* **using** *notOut options* **by** *blast*
}
**hence** *onRegularCone x p* **by** *blast*
}
**thus** *?thesis* **by** *blast*
**qed**

**hence** *?thesis* **using** *in1 in2 out1 out2 on1 on2 tdiff sdiff* **by** *blast*
}
**hence** *slopeFinite x p* ⟶ *?thesis* **by** *blast*

**hence** *?thesis* **using** *ifsinf* **by** *blast*
}
**thus** *?thesis* **by** *blast*
**qed**

**thus** *?thesis* **using** *cases case1* **by** *blast*
**qed**

**lemma** *lemConeCoordinates1*:
**shows** *p* ∈ *regularConeSet x* ⟷ *norm2* (*p*⊖*x*) = *2*∗*sqr* (*tval p* − *tval x*)

**proof** −
  **define** *tdiff* **where** *tdiff*: *tdiff* = *tval p* − *tval x*
  **hence** *tdiff'*: *tdiff* = *tval* ($p \ominus x$) **by** *simp*
  **define** *sdiff* **where** *sdiff*: *sdiff* = (*sComponent* ($p \ominus x$))

  **have** *n*: *norm2* ($p \ominus x$) = *sqr tdiff* + *sNorm2 sdiff*
    **using** *lemNorm2Decomposition sdiff tdiff'* **by** *blast*

  **have** *reg*: *onRegularCone x p* ⟷ *sqr tdiff* = *sNorm2 sdiff*
    **using** *lemConeCoordinates tdiff sdiff* **by** *blast*

  **{ assume** *p* ∈ *regularConeSet x*
    **hence** *onRegularCone x p* **using** *lemRegularCone*[*of x p*] **by** *auto*
    **hence** *sqr tdiff* = *sNorm2 sdiff* **using** *reg* **by** *blast*
    **hence** *norm2* ($p \ominus x$) = *2∗sqr tdiff* **using** *n mult-2* **by** *force*
  **}**
  **hence** *l2r*: *p* ∈ *regularConeSet x* ⟶ *norm2* ($p \ominus x$) = *2∗sqr tdiff*
**by** *auto*

  **{ assume** *norm2* ($p \ominus x$) = *2∗sqr tdiff*
    **hence** *sqr tdiff* + *sNorm2 sdiff* = *2∗sqr tdiff* **using** *n* **by** *auto*
    **hence** *sNorm2 sdiff* = *sqr tdiff* **using** *mult-2 add-diff-eq* **by** *auto*
    **hence** *onRegularCone x p* **using** *reg* **by** *auto*
    **hence** *p* ∈ *regularConeSet x*
      **using** *lemConeContainsVertex lemRegularCone*[*of x p*] **by** *blast*
  **}**
  **hence** *norm2* ($p \ominus x$) = *2∗sqr tdiff* ⟶ *p* ∈ *regularConeSet x* **by**
*blast*
  **thus** *?thesis* **using** *l2r tdiff* **by** *blast*
**qed**

**lemma** *lemWhereLineMeetsCone*:
  **assumes** *a* = *mNorm2 D*
**and** *b* = *2∗*(*tval* ($B \ominus x$))*∗*(*tval D*) − *2∗*((*sComponent D*) ⊙*s*
(*sComponent* ($B \ominus x$)))
**and** *c* = *mNorm2* ($B \ominus x$)
**shows** *qroot a b c α* ⟷ *regularCone x* ($B \oplus (\alpha \otimes D)$)
**proof** −
  **{ fix** *α* **assume** *α*: *qroot a b c α*
    **define** *p* **where** *p*: *p* = ($B \oplus (\alpha \otimes D)$)
    **hence** *mNorm2* ($p \ominus x$) = *a∗*(*sqr α*) + *b∗α* + *c*
      **using** *lemQuadCoordinates*[*of p B α D a b x c*] *assms* **by** *auto*
    **hence** *sqr* (*tval* ($p \ominus x$)) − *sNorm2* (*sComponent* ($p \ominus x$)) = *0* **using**
*α* **by** *auto*
    **hence** *onRegularCone x p* **using** *lemConeCoordinates*[*of x p*] **by**
*auto*
    **hence** *regularCone x* ($B \oplus (\alpha \otimes D)$) **using** *lemRegularCone p* **by**

```
blast
  }
  hence l2r: qroot a b c α ⟶ regularCone x (B ⊕ (α⊗D)) by blast

  { assume reg: regularCone x (B ⊕ (α⊗D))
    define p where p: p = (B ⊕ (α⊗D))
    hence onRegularCone x p using lemRegularCone reg by blast
    hence sqr (tval (p⊖x)) − sNorm2 (sComponent (p⊖x)) = 0
      using lemConeCoordinates[of x p] by auto
    hence a∗(sqr α) + b∗α + c = 0
      using lemQuadCoordinates[of p B α D a b x c] p assms
      by auto
    hence qroot a b c α by auto
  }
  hence regularCone x (B ⊕ (α⊗D)) ⟶ qroot a b c α by auto

  thus ?thesis using l2r by blast
qed


lemma lemLineMeetsCone1:
  assumes ¬ (x ∈ l)
and       isLine l
and       S = l ∩ regularConeSet x
and  l: l = line B D
and  X: X = (B ⊖ x)
and  a: a = mNorm2 D
and  b: b = 2∗(tval X)∗(tval D) − 2∗((sComponent D) ⊙s (sComponent
X))
and  c: c = mNorm2 X
shows (qcase1 a b c ⟶ S = {B})
proof −
  { assume hyp1: qcase1 a b c

    have impa: norm2 D = 2∗sqr (tval D)
    proof −
      have a = 0 using hyp1 by simp
     hence sqr (tval D) = sNorm2 (sComponent D) using a by auto
      hence onRegularCone origin D
        using lemConeCoordinates[of origin D] by auto
      hence regularCone origin D using lemRegularCone by blast
      thus ?thesis using lemConeCoordinates1 by auto
    qed

    have impb: (D⊙X) = 2 ∗ tval X ∗ tval D
    proof −
     have 2∗(tval X)∗(tval D) = 2∗((sComponent D) ⊙s (sComponent
X))
        using hyp1 b by auto
```

**hence** (*tval X*)∗(*tval D*) = ((*sComponent D*) ⊙*s* (*sComponent X*))
**by** (*simp add*: *mult-assoc*)
**thus** *?thesis* **using** *mult-2 lemDotDecomposition*[*of X D*]
*lemSDotCommute mult-assoc lemDotCommute* **by** *metis*
**qed**

**have** *impc*: *norm2 X* = *2*∗*sqr* (*tval X*)
**proof** −
**have** *sqr* (*tval X*) = *sNorm2* (*sComponent X*) **using** *hyp1 c* **by** *auto*
**hence** *onRegularCone x B* **using** *X lemConeCoordinates* **by** *auto*
**hence** *regularCone x B* **using** *lemRegularCone* **by** *blast*
**thus** *?thesis* **using** *X lemConeCoordinates1* **by** *auto*
**qed**

**have** *allOnCone*: ∀ α . *regularCone x* (*B* ⊕ (α ⊗ *D*))
**proof** −
**{ fix** α
**define** *y* **where** *y*: *y* = (*B* ⊕ (α ⊗ *D*))
**have** *qroot a b c* α **using** *hyp1* **by** *simp*
**hence** *regularCone x y*
**using** *lemWhereLineMeetsCone*[*of a D b B x c* α] **using** *y assms* **by** *auto*
**}**
**thus** *?thesis* **by** *auto*
**qed**

**have** *tval D* = *0*
**proof** −
**{ assume** *Dnot0*: *tval D* ≠ *0*
**define** α **where** α: α = (*tval x* − *tval B*)/(*tval D*)
**define** *y* **where** *y*: *y* = (*B* ⊕ (α⊗*D*))
**hence** *yOnl*: *y* ∈ *l* **using** *l* **by** *blast*

**hence** *ty0*: *tval y* = *tval x*
**proof** −
**have** *tval y* = *tval* ((*B* ⊕ (α⊗*D*))) **using** *y* **by** *auto*
**also have** . . . = *tval B* + α∗(*tval D*) **by** *simp*
**also have** . . . = *tval B* + (*tval x* − *tval B*)/(*tval D*)∗(*tval D*) **using** α **by** *simp*
**also have** . . . = *tval B* + (*tval x* − *tval B*) **using** *Dnot0* **by** *simp*
**finally show** *?thesis* **using** *add-commute local.diff-add-cancel* **by** *auto*
**qed**

**have** *regularCone x y* **using** *y allOnCone* **by** *blast*
**hence** *norm2* (*y*⊖*x*) = *2*∗*sqr* (*tval y* − *tval x*)

```
      using lemConeCoordinates1 by auto
    hence norm2 (y⊖x) = 0 using ty0 by auto
    hence (y⊖x) = origin using lemNullImpliesOrigin by blast
    hence y = x by simp

    hence False using yOnl assms by blast
   }
   thus ?thesis by blast
  qed

  hence norm2 D = 0 using impa by auto
  hence D0: D = origin using lemNullImpliesOrigin by auto

  have B0: B = (B ⊕ (0⊗D)) by simp

  have regularCone x (B ⊕ (0⊗D)) using allOnCone by blast
  hence BonCone: regularCone x B
   using B0 by (metis (mono-tags, lifting))
  hence BinS: B ∈ S using assms BonCone B0 l by blast

  hence SisB: S = {B}
  proof −
   { fix y assume y: y ∈ S
     then obtain α where y = (B ⊕ (α⊗D)) using assms l by
blast
    hence y = B using D0 by simp
    hence y ∈ {B} by blast
   }
   hence S ⊆ {B} by blast
   thus ?thesis using BinS by blast
  qed

 }
 thus ?thesis by auto
qed



lemma lemLineMeetsCone2:
 assumes ¬ (x ∈ l)
and      isLine l
and      S = l ∩ regularConeSet x
and  l: l = line B D
and  X: X = (B ⊖ x)
and a = mNorm2 D
and b = 2∗(tval (B⊖x))∗(tval D) − 2∗((sComponent D) ⊙s (sComponent
(B⊖x)))
and c = mNorm2 (B⊖x)
shows qcase2 a b c ⟶ S = {}
```

**proof** −
  **{ assume** *hyp2*: *qcase2 a b c*
    **{ assume** $S \neq \{\}$
      **then obtain** $y$ **where** $y$: $y \in S$ **by** *auto*
       **then obtain** $\alpha$ **where** $\alpha$: $y = (B \oplus (\alpha \otimes D))$ **using** *assms* **by**
*blast*
      **hence** *regularCone* $x$ $(B \oplus (\alpha \otimes D))$ **using** *y assms* **by** *blast*
      **hence** *qroot* $a\ b\ c\ \alpha$
        **using** *lemWhereLineMeetsCone*[*of* $a\ D\ b\ B\ x\ c\ \alpha$] *assms*
        **by** *auto*
      **hence** *False* **using** *lemQCase2*[*of* $a\ b\ c$] *hyp2* **by** *auto*
    **}**
    **hence** $S = \{\}$ **by** *auto*
  **}**
  **thus** *?thesis* **by** *auto*
**qed**

**lemma** *lemLineMeetsCone3*:
  **assumes** $\neg (x \in l)$
**and** *isLine l*
**and** $S = l \cap$ *regularConeSet x*
**and** $l$: $l =$ *line B D*
**and** $X$: $X = (B \ominus x)$
**and** $a$: $a =$ *mNorm2 D*
**and** $b$: $b = 2*(tval\ X)*(tval\ D) - 2*((sComponent\ D) \odot s\ (sComponent$
$X))$
**and** $c$: $c = sqr\ (tval\ X) - sNorm2\ (sComponent\ X)$
**and** *y3*: $y3 = (B \oplus ((-c/b) \otimes D))$
**shows** *qcase3 a b c* $\longrightarrow S = \{y3\}$
**proof** −
  **{ assume** *hyp3*: *qcase3 a b c*
    **define** $T$ **where** $T$: $T = \{y3\}$

    **have** $S \subseteq T$
    **proof** −
      **{ fix** $y$ **assume** $y$: $y \in S$
        **then obtain** $r$ **where** $r$: $y = (B \oplus (r \otimes D))$ **using** *l assms* **by**
*blast*
        **hence** *regularCone x y* **using** *y assms* **by** *auto*
        **hence** *abcr*: *qroot a b c r*
          **using** *a b c r X*
                *lemWhereLineMeetsCone*[*of* $a\ D\ b\ B\ x\ c\ r$]
          **by** *auto*
         **hence** $r = -c/b$ **using** *lemQCase3*[*of* $a\ b\ c\ r$] *abcr hyp3* **by**
*blast*
        **hence** $y = y3$ **using** *y3 r* **by** *auto*

**hence** $y \in T$ **using** $T$ **by** *blast*
**}**
**thus** *?thesis* **by** *auto*
**qed**

**moreover have** $T \subseteq S$
**proof** $-$
**{ fix** $y$ **assume** $y \in T$
**hence** $y$: $y = (B \oplus ((-c/b) \otimes D))$ **using** $T$ *assms* **by** *blast*
**have** *qroot a b c* $(-c/b)$ **using** *lemQCase3 hyp3* **by** *auto*

**hence** *rc*: *regularCone x y*
**using** *hyp3 assms y lemWhereLineMeetsCone*[*of a D b B x c*
$(-c/b)$]
**by** *auto*
**have** $y \in l$ **using** *assms y* **by** *blast*
**hence** $y \in S$ **using** *rc assms* **by** *auto*
**}**
**thus** *?thesis* **by** *blast*
**qed**

**ultimately have** $S = \{y3\}$ **using** $T$ **by** *auto*
**}**
**thus** *?thesis* **by** *blast*
**qed**

**lemma** *lemLineMeetsCone4*:
**assumes** $\neg (x \in l)$
**and** *isLine l*
**and** $S = l \cap$ *regularConeSet x*
**and** $l$: $l =$ *line B D*
**and** $X$: $X = (B \ominus x)$
**and** $a$: $a =$ *mNorm2 D*
**and** $b$: $b = 2*(tval\ X)*(tval\ D) - 2*((sComponent\ D) \odot s\ (sComponent$
$X))$
**and** $c$: $c = sqr\ (tval\ X) - sNorm2\ (sComponent\ X)$
**shows** $(qcase4\ a\ b\ c \longrightarrow S = \{\})$
**proof** $-$
**{ assume** *hyp4*: *qcase4 a b c*
**{ assume** $S \neq \{\}$
**then obtain** $y$ **where** $y$: $y \in S$ **by** *blast*
**then obtain** $r$ **where** $r$: $y = (B \oplus (r \otimes D))$ **using** $l$ *assms* **by**
*blast*
**hence** *regularCone x y* **using** *y assms* **by** *auto*
**hence** *abcr*: *qroot a b c r*
**using** *a b c r X*
*lemWhereLineMeetsCone*[*of a D b B x c r*]

```
      by auto
    hence False using lemQCase4 hyp4 by auto
  }
  hence S = {} by auto
}
thus ?thesis by blast
qed
```

```
lemma lemLineMeetsCone5:
  assumes ¬ (x ∈ l)
and       isLine l
and       S = l ∩ regularConeSet x
and   l: l = line B D
and   X: X = (B ⊖ x)
and   a: a = mNorm2 D
and   b: b = 2∗(tval X)∗(tval D) − 2∗((sComponent D) ⊙s (sComponent
X))
and   c: c = sqr (tval X) − sNorm2 (sComponent X)
and y5: y5 = (B ⊕ ((−b/(2∗a))⊗D))
shows (qcase5 a b c ⟶ S = {y5})
proof −
  { assume hyp5: qcase5 a b c
    define T where T: T = {y5}

    have S ⊆ T
    proof −
      { fix y assume y: y ∈ S
        then obtain r where r: y = (B ⊕ (r⊗D)) using l assms by
blast
        hence regularCone x y using y assms by auto
        hence abcr: qroot a b c r
          using a b c r X
                lemWhereLineMeetsCone[of a D b B x c r]
          by auto
        hence r = −b/(2∗a) using lemQCase5 abcr hyp5 by blast
        hence y = y5 using r y5 by auto
        hence y ∈ T using T by blast
      }
      thus ?thesis by blast
    qed

    moreover have T ⊆ S
    proof −
      { fix y assume y ∈ T
       hence y: y = (B ⊕ ( (−b/(2∗a))⊗D)) using T assms by blast
        have qroot a b c (−b/(2∗a)) using lemQCase5 hyp5 by blast
        hence rc: regularCone x y
```

**using** *hyp5 assms y lemWhereLineMeetsCone*[*of a D b B x c* (−*b*/(*2*∗*a*))]
**by** *auto*
**have** $y \in l$ **using** *assms y* **by** *blast*
**hence** $y \in S$ **using** *rc assms* **by** *auto*
**}**
**thus** *?thesis* **by** *blast*
**qed**

**ultimately have** $S = \{y5\}$ **using** *T* **by** *auto*
**}**
**thus** *?thesis* **by** *blast*
**qed**

**lemma** *lemLineMeetsCone6*:
**assumes** $\neg$ ($x \in l$)
**and** *isLine l*
**and** $S = l \cap$ *regularConeSet x*
**and** *l*: *l* = *line B D*
**and** *X*: $X = (B \ominus x)$
**and** *a*: *a* = *mNorm2 D*
**and** *b*: *b* = *2*∗(*tval X*)∗(*tval D*) − *2*∗((*sComponent D*) ⊙*s* (*sComponent X*))
**and** *c*: *c* = *sqr* (*tval X*) − *sNorm2* (*sComponent X*)
**and** *ym*: *ym* = (*B* ⊕ (((−*b* − (*sqrt* (*discriminant a b c*))) / (*2*∗*a*)) ⊗ *D*))
**and** *yp*: *yp* = (*B* ⊕ (((−*b* + (*sqrt* (*discriminant a b c*))) / (*2*∗*a*)) ⊗ *D*))
**shows** (*qcase6 a b c* ⟶ (*ym* ≠ *yp*) ∧ *S* = {*ym*, *yp*})
**proof** −
**{ assume** *hyp6*: *qcase6 a b c*

**define** *T* **where** *T*: *T* = {*ym*, *yp*}
**define** *rm* **where** *rm*: *rm* = (−*b* − (*sqrt* (*discriminant a b c*))) / (*2*∗*a*)
**define** *rp* **where** *rp*: *rp* = (−*b* + (*sqrt* (*discriminant a b c*))) / (*2*∗*a*)

**have** *ymnotyp*: *ym* ≠ *yp*
**proof** −
**define** *d* **where** *d*: *d* = *discriminant a b c*
**define** *sd* **where** *sd*: *sd* = *sqrt d*

**have** *sdnot0*: *sqrt d* ≠ *0*
**proof** −
**have** *dpos*: *d* > *0* **using** *d hyp6* **by** *simp*

```
        hence hasRoot d using AxEField by auto
        thus ?thesis using lemSquareOfSqrt[of d] dpos by auto
      qed

      have Dnot0: D ≠ origin
      proof –
        { assume D = origin
          hence a = 0 using a by simp
          hence False using hyp6 by simp
        }
        thus ?thesis by auto
      qed

      have rmnotrp: rm ≠ rp
      proof –
        { assume rm = rp
          hence (−b − sd) / (2∗a) = (−b + sd)/(2∗a) using sd d rm
rp by simp
          hence −b−sd = −b+sd using hyp6 by simp
          hence −sd = sd using add-left-imp-eq diff-conv-add-uminus
by metis
          hence False using sdnot0 sd by simp
        }
        thus ?thesis by auto
      qed

      { assume ym = yp
        hence (B ⊕ (rm ⊗ D)) = (B ⊕ (rp ⊗ D)) using ym yp rm rp
by auto
        hence (rm ⊗ D) = (rp ⊗ D) by simp
        hence ((rm − rp)⊗D) = origin by auto
        hence rm − rp = 0 using Dnot0 by auto
        hence False using rmnotrp by auto
      }
      thus ?thesis by auto
    qed


    have S ⊆ T
    proof –
      { fix y assume y: y ∈ S
        then obtain r where r: y = (B ⊕ (r⊗D)) using l assms by
blast
        hence regularCone x y using y assms by auto
        hence abcr: qroot a b c r
          using a b c r X
               lemWhereLineMeetsCone[of a D b B x c r]
          by auto
        hence qroots a b c = {rp, rm}
```

**using** *lemQCase6*[*of a b c sqrt* (*discriminant a b c*) *rp rm*]
*rp rm hyp6* **by** *auto*
**hence** *rchoice*: $(r = rm \vee r = rp)$ **using** *abcr* **by** *blast*
**hence** *ychoice*: $(y = ym \vee y = yp)$ **using** *r ym yp rm rp* **by** *blast*
**hence** *yinT*: $y \in T$ **using** *T* **by** *blast*
**}**
**thus** *?thesis* **by** *auto*
**qed**

**moreover have** $T \subseteq S$
**proof** −
**{ fix** *y* **assume** $y \in T$
**hence** *y*: $y = ym \vee y = yp$ **using** *T assms* **by** *blast*

**have** *qroot a b c rm* **using** *rm lemQCase6 hyp6* **by** *blast*
**hence** *rcm*: *regularCone x ym*
**using** *hyp6 assms ym rm lemWhereLineMeetsCone*[*of a D b B x c rm*]
**by** *auto*
**have** *qroot a b c rp* **using** *rp lemQCase6 hyp6* **by** *blast*
**hence** *rcp*: *regularCone x yp*
**using** *hyp6 assms yp rp lemWhereLineMeetsCone*[*of a D b B x c rp*]
**by** *auto*
**hence** *regularCone x y* **using** *rcm y* **by** *blast*
**moreover have** $y \in l$ **using** *assms y* **by** *blast*
**ultimately have** $y \in S$ **using** *assms* **by** *blast*
**}**
**thus** *?thesis* **by** *blast*
**qed**

**ultimately have** $(ym \neq yp) \wedge S = \{ym, yp\}$ **using** *T ymnotyp* **by** *auto*
**}**
**thus** *?thesis* **by** *blast*
**qed**

**lemma** *lemConeLemma1*:
**assumes** $\neg (x \in l)$
**and** *isLine l*
**and** $S = l \cap$ *regularConeSet x*
**shows** *finite S* $\wedge$ *card S* $\leq 2$
**proof** −
**obtain** *B D* **where** *BD*: $l =$ *line B D* **using** *assms*(*2*) **by** *auto*

**define** *X* **where** *X*: $X = (B \ominus x)$
**define** *a* **where** *a*: $a = mNorm2\ D$

**define** *b* **where** *b*: $b = 2*(tval\ X)*(tval\ D) - 2*((sComponent\ D)$ $\odot s\ (sComponent\ X))$
**define** *c* **where** *c*: $c = sqr\ (tval\ X) - sNorm2\ (sComponent\ X)$

**have** *qcase1 a b c* ⟶ *?thesis*
**using** *assms X a b c lemLineMeetsCone1[of x l S B D X a b c] BD*
**by** *auto*
**moreover have** *qcase2 a b c* ⟶ *?thesis*
**using** *assms X a b c lemLineMeetsCone2[of x l S B D X a b c] BD*
**by** *auto*
**moreover have** *qcase3 a b c* ⟶ *?thesis*
**using** *assms X a b c lemLineMeetsCone3[of x l S B D X a b c] BD*
**by** *auto*
**moreover have** *qcase4 a b c* ⟶ *?thesis*
**using** *assms X a b c lemLineMeetsCone4[of x l S B D X a b c] BD*
**by** *auto*
**moreover have** *qcase5 a b c* ⟶ *?thesis*
**using** *assms X a b c lemLineMeetsCone5[of x l S B D X a b c] BD*
**by** *auto*
**moreover have** *qcase6 a b c* ⟶ *?thesis*
**proof** −
**{ assume** *hyp6*: *qcase6 a b c*
**define** *ym* **where** *ym*: $ym = (B \oplus (((-b - (sqrt\ (discriminant$ $a\ b\ c))) / (2*a)) \otimes D))$
**define** *yp* **where** *yp*: $yp = (B \oplus (((-b + (sqrt\ (discriminant\ a$ $b\ c))) / (2*a)) \otimes D))$

**have** $(ym \neq yp) \wedge S = \{\ ym,\ yp\ \}$
**using** *assms X a b c ym yp hyp6*
*lemLineMeetsCone6[of x l S B D X a b c ym yp] BD*
**by** *auto*
**hence** *card S = 2* **using** *card-2-iff* **by** *blast*
**hence** *finite S* ∧ *card S* ≤ *2* **using** *card.infinite* **by** *fastforce*
**}**
**thus** *?thesis* **by** *auto*
**qed**

**ultimately show** *?thesis* **using** *lemQuadraticCasesComplete* **by** *blast*
**qed**

**lemma** *lemConeLemma2*:
**assumes** ¬ (*regularCone x w*)
**shows** $\exists\ l\ .\ (onLine\ w\ l) \wedge (\neg\ (x \in l)) \wedge (card\ (l \cap (regularConeSet$ $x)) = 2)$
**proof** −
**have** *xnotw*: $x \neq w$ **using** *assms lemConeContainsVertex* **by** *blast*

```
have iftvalsequal: tval x = tval w ⟶ ?thesis
proof −
  { assume ts: tval x = tval w
    define l where l: l = line w tUnit

    hence wonl: onLine w l
    proof −
      have w = (w ⊕ (0⊗tUnit)) by simp
      thus ?thesis using l by blast
    qed

    have xnotinl: ¬(x ∈ l)
    proof −
      { assume x ∈ l
         then obtain a where a: x = (w ⊕ (a⊗tUnit)) using l by
blast
        hence tval x = tval w + a by simp
        hence a = 0 using ts by simp
        hence x = w using a by simp
        hence False using xnotw by simp
      }
      thus ?thesis by blast
    qed

    have card (l ∩ (regularConeSet x)) = 2
    proof −
      define S where S: S = l ∩ regularConeSet x
      hence cardS: finite S ∧ card S ≤ 2
        using xnotinl l lemConeLemma1[of x l S] by blast

      have (sNorm2 (sComponent (w⊖x))) ≥ 0 by simp
      hence sExists: hasRoot (sNorm2 (sComponent (w⊖x))) using
AxEField by auto

      define s  where  s: s = sqrt (sNorm2 (sComponent (w⊖x)))
      define yp where yp: yp = (w ⊕ (s⊗tUnit))
      define ym where ym: ym = (w ⊖ (s⊗tUnit))

      have ypnotym: yp ≠ ym
      proof −
        { assume yp = ym
           hence (w ⊕ (s⊗tUnit)) = (w ⊖ (s⊗tUnit)) using yp ym
by auto
          hence tval w + s = tval w − s by simp
          hence s = 0
            by (metis local.add-cancel-right-right
                  local.double-zero-sym local.lemDiffSumCancelMiddle)
          hence sNorm2 (sComponent (w⊖x)) = sqr 0
           using s lemSquareOfSqrt[of sNorm2 (sComponent (w⊖x))
```

$s$] *sExists*
**by** *auto*
**hence** *norm2* $(w \ominus x) = 0$ **using** *lemNorm2Decomposition ts* **by** *auto*
**hence** $(w \ominus x) =$ *origin* **using** *lemNullImpliesOrigin* **by** *blast*
**hence** $w = x$ **by** *simp*
**hence** *False* **using** *xnotw* **by** *simp*
**}**
**thus** *?thesis* **by** *auto*
**qed**

**have** *ypinl*: $yp \in l$ **using** *yp l* **by** *blast*
**have** *yminl*: $ym \in l$
**proof** $-$
**have** $ym = (w \oplus ((-s) \otimes tUnit))$ **using** *ym* **by** *simp*
**thus** *?thesis* **using** *l* **by** *blast*
**qed**

**have** *ypcone*: $yp \in$ *regularConeSet x*
**proof** $-$
**have** $(yp \ominus x) = ((w \oplus (s \otimes tUnit)) \ominus x)$ **using** *yp* **by** *auto*
**hence** *tval* $(yp \ominus x) = s$ **using** *ts* **by** *simp*
**hence** *tsqr*: *sqr* (*tval* $(yp \ominus x)$) = (*sNorm2* (*sComponent* $(w \ominus x)$))
**using** *s sExists lemSquareOfSqrt AxEField* **by** *blast*
**hence** *sComponent* $(yp \ominus x)$ = *sComponent* $((w \oplus (s \otimes tUnit)) \ominus x)$ **using** *yp* **by** *auto*
**also have** ... = ((*sComponent* $(w \oplus (s \otimes tUnit))$) $\ominus s$ (*sComponent x*)) **by** *simp*
**also have** ... = (((*sComponent w*) $\oplus s$ (*sComponent* $(s \otimes tUnit)$)) $\ominus s$ (*sComponent x*)) **by** *simp*
**also have** ... = ((*sComponent w*) $\ominus s$ (*sComponent x*)) **by** *simp*
**finally have** *sComponent* $(yp \ominus x)$ = *sComponent* $(w \ominus x)$ **by** *simp*
**hence** *ssqr*: *sNorm2* (*sComponent* $(yp \ominus x)$) = (*sNorm2* (*sComponent* $(w \ominus x)$))
**by** *auto*
**hence** *sqr* (*tval* $(yp \ominus x)$) = (*sNorm2* (*sComponent* $(yp \ominus x)$)) **using** *tsqr* **by** *auto*
**hence** *onRegularCone x yp* **using** *lemConeCoordinates*[*of x yp*] **by** *auto*
**thus** *?thesis* **using** *lemRegularCone* **by** *blast*
**qed**

**have** *ymcone*: $ym \in$ *regularConeSet x*
**proof** $-$
**have** $(ym \ominus x) = ((w \ominus (s \otimes tUnit)) \ominus x)$ **using** *ym* **by** *auto*

**hence** *tval* $(ym \ominus x) = tval\ (w \ominus (s \otimes tUnit)) - tval\ x$ **by** *simp*
**also have** $\ldots = (tval\ w - tval(s \otimes tUnit)) - tval\ x$ **by** *simp*
**also have** $\ldots = (tval\ w - s) - tval\ w$ **using** *ts* **by** *simp*
**finally have** *tval* $(ym \ominus x) = -s$ **using** *diff-right-commute*
**by** (*metis local.add-implies-diff local.uminus-add-conv-diff*)
**hence** *sqr* $(tval\ (ym \ominus x)) = sqr\ s$ **by** *simp*
**hence** *tsqr*: *sqr* $(tval\ (ym \ominus x)) = (sNorm2\ (sComponent\ (w \ominus x)))$
**using** *s sExists lemSquareOfSqrt AxEField* **by** *force*

**hence** *sComponent* $(ym \ominus x) = sComponent\ ((w \ominus (s \otimes tUnit)) \ominus x)$ **using** *ym* **by** *auto*
**also have** $\ldots = ((sComponent\ (w \ominus (s \otimes tUnit))) \ominus s\ (sComponent\ x))$ **by** *simp*
**also have** $\ldots = (((sComponent\ w) \ominus s\ (sComponent\ (s \otimes tUnit))) \ominus s\ (sComponent\ x))$ **by** *simp*
**also have** $\ldots = ((sComponent\ w) \ominus s\ (sComponent\ x))$ **by** *simp*
**finally have** *sComponent* $(ym \ominus x) = sComponent\ (w \ominus x)$ **by** *simp*
**hence** *ssqr*: *sNorm2* $(sComponent\ (ym \ominus x)) = (sNorm2\ (sComponent\ (w \ominus x)))$
**by** *auto*
**hence** *sqr* $(tval\ (ym \ominus x)) = (sNorm2\ (sComponent\ (ym \ominus x)))$ **using** *tsqr* **by** *auto*
**hence** *onRegularCone x ym* **using** *lemConeCoordinates*[*of x ym*] **by** *auto*
**thus** *?thesis* **using** *lemRegularCone* **by** *blast*
**qed**

**define** *T* **where** *T*: $T = \{yp, ym\}$

**hence** $T \subseteq S$ **using** *ypinl ypcone yminl ymcone S* **by** *auto*
**hence** *TleS*: *card* $T \leq card\ S$ **using** *cardS card-mono* **by** *blast*
**have** *cardT*: *card* $T = 2$ **using** *T ypnotym card-2-iff* **by** *blast*

**hence** $(2 \leq card\ S) \wedge finite\ S \wedge card\ S \leq 2$ **using** *TleS cardS* **by** *auto*
**thus** *?thesis* **using** *S* **by** *simp*
**qed**

**hence** *?thesis* **using** *xnotinl wonl* **by** *blast*
**}**
**thus** *?thesis* **by** *auto*

**qed**

**have** *iftvalsne*: *tval* $x \neq tval\ w \longrightarrow ?thesis$

**proof** –
{ **assume** *ts*: *tval x* ≠ *tval w*

**define** *x0* **where** *x0*: *x0* = *mkPoint* (*tval w*) (*xval x*) (*yval x*) (*zval x*)
**have** *xnotx0*: *x* ≠ *x0* **using** *x0 ts* **by** (*metis Point.select-convs*(*1*))
**have** *tdiff0*: *tval w* = *tval x0* **using** *x0* **by** *simp*

**define** *dir* **where** *dir*: *dir* = (*if* (*w*≠*x0*) *then* (*w*⊖*x0*) *else xUnit*)

**hence** *tdir0*: *tval dir* = *0*
**proof** –
{ **assume** *w*≠*x0*
**hence** *dir* = (*w*⊖*x0*) **using** *dir* **by** *simp*
}
**hence** *wnotx0*: (*w*≠*x0*) ⟶ *?thesis* **using** *tdiff0* **by** *auto*
{ **assume** *w* = *x0*
**hence** *dir* = *xUnit* **using** *dir* **by** *simp*
}
**hence** (*w*=*x0*) ⟶ *?thesis* **by** *simp*
**thus** *?thesis* **using** *wnotx0* **by** *auto*
**qed**

**define** *l* **where** *l*: *l* = *lineJoining x0* (*dir*⊕*x0*)
**hence** *lprops*: *l* = *line x0 dir* **using** *dir* **by** *auto*

**hence** *wonl*: *onLine w l*
**proof** –
{ **assume** *wnotx0*: *w* ≠ *x0*
**hence** *dir* = (*w*⊖*x0*) **using** *dir* **by** *simp*
**hence** (*dir*⊕*x0*) = ((*w*⊖*x0*)⊕*x0*) **by** *simp*
**hence** *w* = (*dir* ⊕ *x0*) **using** *diff-add-eq* **by** *auto*
**hence** *?thesis* **using** *dir lemLineJoiningContainsEndPoints l* **by** *blast*
}
**moreover have** (*w*=*x0*) ⟶ *?thesis* **using** *lemLineJoiningContainsEndPoints l* **by** *blast*
**ultimately show** *?thesis* **by** *auto*
**qed**

**then obtain** *A* **where** *A*: *w* = (*x0* ⊕ (*A* ⊗ *dir* )) **using** *l* **by** *auto*

**have** *xnotinl*: ¬(*x* ∈ *l*)
**proof** –
{ **assume** *x* ∈ *l*
**then obtain** *a* **where** *a*: *x* = (*x0* ⊕ (*a*⊗*dir*)) **using** *l* **by** *auto*
**hence** *tval x* = *tval x0* **using** *tdir0* **by** *simp*
**hence** *False* **using** *ts tdiff0* **by** *auto*

}
thus *?thesis* **by** *blast*
**qed**

**have** *card* $(l \cap (regularConeSet\ x)) = 2$
**proof** −
**define** *S* **where** *S*: $S = l \cap regularConeSet\ x$
**hence** *cardS*: *finite S* $\wedge$ *card S* $\leq 2$
**using** *xnotinl l lemConeLemma1*[*of x l S*] **by** *blast*

**have** (*sNorm2* (*sComponent* ($w \ominus x0$))) $\geq 0$ **by** *simp*
**hence** *sExists*: *hasRoot* (*sNorm2* (*sComponent* ($w \ominus x0$))) **using** *AxEField* **by** *auto*
**define** *s* **where** *s*: *s* = *sqrt* (*sNorm2* (*sComponent* ($w \ominus x0$)))

**define** *unit* **where** *unit*: *unit* = (*if* (*w* = *x0*) *then xUnit else* ((*1*/*s*)$\otimes$($w \ominus x0$)))

**have** *tunit0*: *tval unit* = *0*
**proof** −
{ **assume** *w* = *x0*
**hence** *unit* = *xUnit* **using** *unit* **by** *simp*
}
**hence** *w*=*x0* $\longrightarrow$ *?thesis* **by** *auto*
**moreover have** *w*$\neq$*x0* $\longrightarrow$ *?thesis*
**proof** −
{ **assume** *wnotx0*: *w* $\neq$ *x0*
**hence** *unit* = ((*1*/*s*)$\otimes$*dir*) **using** *unit dir* **by** *simp*
}
**thus** *?thesis* **using** *tdir0* **by** *auto*
**qed**
**ultimately show** *?thesis* **by** *auto*
**qed**

**have** *snot0*: *w* $\neq$ *x0* $\longrightarrow$ *s* $\neq$ *0*
**proof** −
{ **assume** *wnotx0*: *w* $\neq$ *x0*
**hence** *norm2* ($w \ominus x0$) > *0*
**using** *local.lemNotEqualImpliesSep2Pos* **by** *presburger*
**also have** *norm2* ($w \ominus x0$) = *sNorm2* (*sComponent* ($w \ominus x0$))
**using** *tdiff0 lemNorm2Decomposition*[*of* $w \ominus x0$] **by** *auto*
**finally have** *s2pos*: *sNorm2* (*sComponent* ($w \ominus x0$)) > *0* **by** *auto*
{ **assume** *s* = *0*
**hence** *False* **using** *lemSquareOfSqrt*[*of sNorm2* (*sComponent* ($w \ominus x0$)) *s*]
*s2pos s sExists* **by** *auto*
}
**hence** *s* $\neq$ *0* **by** *auto*

}
thus *?thesis* **by** *auto*
**qed**

**hence** *unit1*: *sNorm2* (*sComponent unit*) = *1*
**proof** −
**have** *case0*: $w{=}x0 \longrightarrow$ *?thesis* **using** *unit* **by** *auto*
**have** *case1*: $w{\neq}x0 \longrightarrow$ *?thesis*
**proof** −
{ **assume** *case1*: $w \neq x0$
**have** *unit* = ((*1*/*s*)⊗(*w*⊖*x0*)) **using** *unit case1* **by** *simp*
**hence** *sComponent unit* = ((*1*/*s*) ⊗*s* (*sComponent* (*w*⊖*x0*))) **by** *simp*
**hence** *sNorm2* (*sComponent unit*) = *sqr* (*1*/*s*) ∗ *sNorm2* (*sComponent* (*w*⊖*x0*))
**using** *lemSNorm2OfScaled*[*of* (*1*/*s*) *sComponent* (*w*⊖*x0*)]
**by** *auto*
**also have** . . . = *sqr* (*1*/*s*) ∗ *sqr s*
**using** *lemSquareOfSqrt*[*of sNorm2* (*sComponent* (*w*⊖*x0*)) *s*] *sExists s*
**by** *auto*
**finally have** *sNorm2* (*sComponent unit*) = *1* **using** *snot0 case1* **by** *simp*
}
**thus** *?thesis* **by** *auto*
**qed**
**thus** *?thesis* **using** *case0* **by** *blast*
**qed**

**define** *dt* **where** *dt*: *dt* = *tval w* − *tval x*
**define** *mdt* **where** *mdt*: *mdt* = −*dt*
**define** *yp* **where** *yp*: *yp* = (*x0* ⊕ (*dt* ⊗ *unit*))
**define** *ym* **where** *ym*: *ym* = (*x0* ⊖ (*dt* ⊗ *unit*))
**hence** *ymalt*: *ym* = (*x0* ⊕ (*mdt* ⊗ *unit*)) **using** *mdt* **by** *simp*

**have** *ypnotym*: *yp* ≠ *ym*
**proof** −
{ **assume** *yp* = *ym*
**hence** (*x0* ⊕ (*dt*⊗*unit*)) = (*x0* ⊖ (*dt*⊗*unit*)) **using** *yp ym* **by** *auto*
**hence** ((*x0* ⊕ (*dt*⊗*unit*)) ⊕ (*dt*⊗*unit*)) = *x0* **by** *auto*
**hence** (*x0* ⊕ (*2*⊗(*dt*⊗*unit*))) = *x0* **using** *add-assoc mult-2* **by** *auto*
**hence** ((*x0* ⊕ (*2*⊗(*dt*⊗*unit*))) ⊖ *x0*) = *origin* **by** *simp*
**hence** (*2*⊗(*dt*⊗*unit*)) = *origin* **using** *add-diff-eq* **by** *auto*
**hence** *False* **using** *unit1 ts dt* **by** *simp*
}
**thus** *?thesis* **by** *auto*
**qed**

**have** *ypinl*: $yp \in l$
**proof** −
{ **assume** $w = x0$
**hence** $yp = (w \oplus (dt \otimes dir))$ **using** *dir unit yp* **by** *simp*
**hence** $\exists\ a\ .\ yp = (w \oplus (a \otimes dir))$ **using** *yp* **by** *auto*
}
**hence** *wx0*: $w=x0 \longrightarrow ?thesis$ **using** *l* **by** *auto*

{ **assume** *wnotx0*: $w \neq x0$
**hence** *u*: $unit = ((1/s) \otimes dir)$ **using** *unit dir* **by** *auto*
**hence** $yp = (x0 \oplus ((dt/s) \otimes dir))$ **using** *lemScaleAssoc yp* **by** *auto*
**hence** $\exists\ a\ .\ yp = (x0 \oplus (a \otimes dir))$ **using** *snot0* **by** *blast*
}
**hence** $w \neq x0 \longrightarrow ?thesis$ **using** *l* **by** *auto*
**thus** *?thesis* **using** *wx0* **by** *blast*
**qed**

**have** *yminl*: $ym \in l$
**proof** −
{ **assume** $w = x0$
**hence** $ym = (x0 \oplus (mdt \otimes dir))$ **using** *dir unit ymalt* **by** *simp*
**hence** $\exists\ a\ .\ ym = (x0 \oplus (a \otimes dir))$ **using** *ym* **by** *auto*
}
**hence** *wx0*: $w=x0 \longrightarrow ?thesis$ **using** *l* **by** *auto*

{ **assume** *wnotx0*: $w \neq x0$
**hence** *u*: $unit = ((1/s) \otimes dir)$ **using** *unit dir* **by** *auto*
**hence** $ym = (x0 \oplus ((mdt/s) \otimes dir))$ **using** *lemScaleAssoc ymalt* **by** *auto*
**hence** $\exists\ a\ .\ ym = (x0 \oplus (a \otimes dir))$ **using** *snot0* **by** *blast*
}
**hence** $w \neq x0 \longrightarrow ?thesis$ **using** *l* **by** *auto*
**thus** *?thesis* **using** *wx0* **by** *blast*
**qed**

**have** *ypcone*: $yp \in regularConeSet\ x$
**proof** −
**have** $sNorm2\ (sComponent\ (yp \ominus x0)) = sqr\ dt$
**proof** −
**have** $yp = (x0 \oplus (dt \otimes unit))$ **using** *yp* **by** *simp*
**hence** $(yp \ominus x0) = (dt \otimes unit)$ **using** *add-diff-eq diff-add-eq* **by** *auto*
**hence** $sComponent\ (yp \ominus x0) = (dt \otimes s\ (sComponent\ unit))$ **by** *auto*
**thus** *?thesis*
**using** *lemSNorm2OfScaled*[*of dt sComponent unit*] *unit1* **by** *auto*

**qed**
**hence** *sNorm2* (*sComponent* (*yp*⊖*x*)) = *sqr dt* **using** *x0* **by** *simp*
**also have** ... = *sqr* (*tval* (*yp*⊖*x*)) **using** *dt tunit0 yp tdiff0* **by** *simp*
**finally have** *sNorm2* (*sComponent* (*yp*⊖*x*)) = *sqr* (*tval* (*yp*⊖*x*)) **by** *blast*
**hence** *onRegularCone x yp* **using** *lemConeCoordinates*[*of x yp*] **by** *auto*
**thus** *?thesis* **using** *lemRegularCone* **by** *blast*
**qed**

**have** *ymcone*: *ym* ∈ *regularConeSet x*
**proof** −
**have** *sNorm2* (*sComponent* (*ym*⊖*x0*)) = *sqr dt*
**proof** −
**have** *ym* = (*x0* ⊕ (*mdt* ⊗ *unit*)) **using** *ymalt* **by** *simp*
**hence** (*ym* ⊖ *x0*) = (*mdt* ⊗ *unit*) **using** *add-diff-eq diff-add-eq* **by** *auto*
**hence** *sComponent* (*ym* ⊖ *x0*) = (*mdt* ⊗*s* (*sComponent unit*)) **by** *auto*
**thus** *?thesis*
**using** *lemSNorm2OfScaled*[*of mdt sComponent unit*] *unit1 mdt* **by** *auto*
**qed**
**hence** *sNorm2* (*sComponent* (*ym*⊖*x*)) = *sqr dt* **using** *x0* **by** *simp*
**also have** ... = *sqr* (*tval* (*ym*⊖*x*)) **using** *ym mdt dt tunit0 tdiff0* **by** *auto*
**finally have** *sNorm2* (*sComponent* (*ym*⊖*x*)) = *sqr* (*tval* (*ym*⊖*x*)) **by** *blast*
**hence** *onRegularCone x ym* **using** *lemConeCoordinates*[*of x ym*] **by** *auto*
**thus** *?thesis* **using** *lemRegularCone* **by** *blast*
**qed**

**define** *T* **where** *T*: *T* = {*yp*, *ym*}

**hence** *T* ⊆ *S* **using** *ypinl ypcone yminl ymcone S* **by** *auto*
**hence** *TleS*: *card T* ≤ *card S* **using** *cardS card-mono* **by** *blast*
**have** *cardT*: *card T* = *2* **using** *T ypnotym card-2-iff* **by** *blast*

**hence** (*2* ≤ *card S*) ∧ *finite S* ∧ *card S* ≤ *2* **using** *TleS cardS* **by** *auto*
**thus** *?thesis* **using** *S* **by** *simp*
**qed**

**hence** *?thesis* **using** *xnotinl wonl* **by** *blast*
**}**

**thus** *?thesis* **by** *auto*

**qed**
**thus** *?thesis* **using** *iftvalsequal* **by** *blast*
**qed**

**lemma** *lemLineInsideRegularConeHasFiniteSlope*:
**assumes** *insideRegularCone x p*
**and** $l$ = *lineJoining x p*
**shows** *lineSlopeFinite l*
**proof** –
{ **assume** *converse*: ¬ (*lineSlopeFinite l*)
**hence** *slope*: *slopeInfinite x p*
**using** *assms lemSlopeLineJoining*[*of l*] **by** *blast*
**hence** *False* **using** *assms*(*1*) *assms*(*2*) *slope* **by** *simp*
}
**thus** *?thesis* **by** *auto*
**qed**

**lemma** *lemInvertibleOnMeet*:
**assumes** *invertible f*
**and** $S = A \cap B$
**shows** *applyToSet* (*asFunc f*) $S$ = (*applyToSet* (*asFunc f*) $A$) $\cap$ (*applyToSet* (*asFunc f*) $B$)
**proof** –
**define** $S'$ **where** $S'$: $S'$ = *applyToSet* (*asFunc f*) $S$
**define** $A'$ **where** $A'$: $A'$ = *applyToSet* (*asFunc f*) $A$
**define** $B'$ **where** $B'$: $B'$ = *applyToSet* (*asFunc f*) $B$

**have** $S' \subseteq A' \cap B'$
**proof** –
{ **fix** $s'$ **assume** $s' \in S'$
**then obtain** $s$ **where** $s$: $s \in S \wedge f\ s = s'$ **using** $S'$ **by** *auto*
**have** *inA*: $s' \in A'$
**proof** –
**have** $s \in A$ **using** *assms s* **by** *auto*
**thus** *?thesis* **using** $s$ $A'$ **by** *auto*
**qed**
**have** *inB*: $s' \in B'$
**proof** –
**have** $s \in B$ **using** *assms s* **by** *auto*
**thus** *?thesis* **using** $s$ $B'$ **by** *auto*
**qed**
**hence** $s' \in A' \cap B'$ **using** *inA* **by** *auto*
}

**thus** *?thesis* **by** *auto*
**qed**

**moreover have** $A' \cap B' \subseteq S'$
**proof** −
{ **fix** $s'$ **assume** $s'$: $s' \in A' \cap B'$
**then obtain** $a$ **where** $a$: $a \in A \wedge f\ a = s'$ **using** $A'$ **by** *auto*
**obtain** $b$ **where** $b$: $b \in B \wedge f\ b = s'$ **using** $s'$ $B'$ **by** *auto*

**have** $(\exists\ p\ .\ (f\,p = s') \wedge (\forall\,x.\ f\,x = s' \longrightarrow x = p))$ **using** *assms*(*1*) **by** *auto*
**then obtain** $p$ **where** $p$: $(f\ p = s') \wedge (\forall\,x.\ f\ x = s' \longrightarrow x = p)$ **by** *auto*
**hence** $a = b$ **using** $a$ $b$ **by** *blast*
**hence** $a \in S \wedge f\ a = s'$ **using** $a$ $b$ *assms*(*2*) **by** *auto*
**hence** $s' \in S'$ **using** $S'$ **by** *auto*
}
**thus** *?thesis* **by** *auto*
**qed**

**ultimately show** *?thesis* **using** $S'$ $A'$ $B'$ **by** *auto*
**qed**

**lemma** *lemInsideCone*:
**shows** *insideRegularCone x p* $\longleftrightarrow$
$\neg$(*vertex x p* $\vee$ *outsideRegularCone x p* $\vee$ *onRegularCone x p*)
**proof** −
{ **assume** *lhs*: *insideRegularCone x p*
**hence** (*slopeFinite x p*) $\wedge$ ($\exists$ $v$ $\in$ *lineVelocity* (*lineJoining x p*) . *sNorm2 v* $<$ *1*)
**by** *auto*
**hence** *rtp1*: $\neg$(*vertex x p*) **by** *blast*

**define** $l$ **where** $l$: $l$ = *lineJoining x p*

**obtain** *vin* **where** *vin*: *vin* $\in$ *lineVelocity l* $\wedge$ *sNorm2 vin* $<$ *1* **using** *l lhs* **by** *blast*
**hence** *vs*: $\forall$ $v$ . $v \in$ *lineVelocity l* $\longrightarrow$ *sNorm2 v* $<$ *1*
**proof** −
{ **fix** $v$ **assume** $v$: $v \in$ *lineVelocity l*
**have** *slopeFinite x p* **using** *lhs* **by** *blast*
**moreover have** *onLine x l* $\wedge$ *onLine p l* **using** *l lemLineJoiningContainsEndPoints*
**by** *auto*
**ultimately have** $v$ = *vin*
**using** *rtp1 v vin lemFiniteLineVelocityUnique*[*of v l vin*] **by** *blast*

}
thus *?thesis* **using** *vin* **by** *blast*
**qed**

{ **assume** *outsideRegularCone x p*
**then obtain** *v* **where** *v*: $v \in$ *lineVelocity l* $\wedge$ *sNorm2 v* $> 1$ **using** *l lhs* **by** *blast*
**hence** *sNorm2 v* $< 1$ **using** *vs* **by** *blast*
**hence** *False* **using** *v* **by** *force*
}
**hence** *rtp2*: $\neg$ *outsideRegularCone x p* **by** *blast*
{ **assume** *onRegularCone x p*
**then obtain** *v* **where** *v*: $v \in$ *lineVelocity l* $\wedge$ *sNorm2 v* $= 1$ **using** *l lhs* **by** *blast*
**hence** *sNorm2 v* $< 1$ **using** *vs* **by** *blast*
**hence** *False* **using** *v* **by** *force*
}
**hence** *rtp3*: $\neg$ *onRegularCone x p* **by** *blast*
**hence** $\neg$(*vertex x p* $\vee$ *outsideRegularCone x p* $\vee$ *onRegularCone x p*)
**using** *rtp1 rtp2* **by** *blast*
}
**hence** *l2r*: *insideRegularCone x p* $\longrightarrow$
$\neg$(*vertex x p* $\vee$ *outsideRegularCone x p* $\vee$ *onRegularCone x p*)
**by** *blast*

{ **assume** *rhs*: $\neg$(*vertex x p* $\vee$ *outsideRegularCone x p* $\vee$ *onRegularCone x p*)
**define** *v* **where** *v*: $v =$ (*insideRegularCone x p*)
**define** *z* **where** *z*: $z =$ (*vertex x p* $\vee$ *outsideRegularCone x p* $\vee$ *onRegularCone x p*)
**hence** $v \vee z$ **using** *v z lemClassification*[*of x p*] **by** *auto*
**hence** *insideRegularCone x p* **using** *rhs v z* **by** *blast*
}
**thus** *?thesis* **using** *l2r* **by** *blast*
**qed**

**lemma** *lemOnRegularConeIff*:
**assumes** $l =$ *lineJoining x p*
**shows** *onRegularCone x p* $\longleftrightarrow$ ($l \cap$ *regularConeSet x* $= l$)
**proof** –
{ **assume** *rc*: *onRegularCone x p*
**hence** *reg*: *regularCone x p* **using** *lemRegularCone* **by** *blast*
**define** *S* **where** *S*: $S = l \cap$ *regularConeSet x*

**have** *SinL*: $S \subseteq l$ **using** *S* **by** *blast*

```
    have l ⊆ S
    proof –
      { fix q assume q: q ∈ l
        then obtain a where a: q = (x ⊕ (a ⊗ (p⊖x))) using assms
by blast
        hence qmx: (q⊖x) = (a ⊗ (p⊖x)) by simp

        hence sqr (tval (q⊖x)) = sqr (tval (a ⊗ (p⊖x))) by auto
      also have ... = (sqr a)∗(sqr (tval p − tval x)) using lemSqrMult
by auto
        also have ... = (sqr a)∗(sNorm2 (sComponent (p⊖x)))
          using rc lemConeCoordinates[of x p] by auto
        also have ... = sNorm2 ( a ⊗s (sComponent (p⊖x)) )
         using lemSNorm2OfScaled[of a (sComponent (p⊖x))] by auto
       also have ... = sNorm2 (sComponent ( a ⊗ (p⊖x) )) by simp
        finally have  sqr (tval (q⊖x)) = sNorm2 (sComponent (q⊖x)
) using qmx by simp
         hence onRegularCone x q using lemConeCoordinates[of x q]
by auto
        hence regularCone x q using lemRegularCone by blast
        hence q ∈ S using S q by auto
      }
      hence ∀ q . q ∈ l ⟶ q ∈ S by blast
      thus ?thesis by blast
    qed

    hence (l ∩ regularConeSet x = l) using S SinL by blast
  }
  hence l2r: onRegularCone x p ⟶ (l ∩ regularConeSet x = l) by
blast

  { assume rhs: (l ∩ regularConeSet x = l)
    have p ∈ l
      using lemLineJoiningContainsEndPoints[of l x p] assms(1) by
auto
    hence regularCone x p using rhs by blast
    hence onRegularCone x p using lemRegularCone by blast
  }
  thus ?thesis using l2r by blast
qed



lemma lemOutsideRegularConeImplies:
  shows     outsideRegularCone x p
              ⟶ (∃ l p′ . (p′ ≠ p) ∧ onLine p′ l ∧ onLine p l
                                  ∧ (l ∩ regularConeSet x = {}))
proof –
  { assume lhs: outsideRegularCone x p
```

**hence** *xnotp*: $(x \neq p)$ **by** *auto*
**hence** *formula*: *sqr* (*tval p* − *tval x*) < *sNorm2* (*sComponent* ($p \ominus x$))
**using** *lemConeCoordinates*[*of x p*] **using** *lhs* **by** *auto*

**have** *cases*: (*slopeInfinite x p*) ∨
((*slopeFinite x p*) ∧
(∃ *v* ∈ *lineVelocity* (*lineJoining x p*) . *sNorm2 v* > *1*))
**using** *lhs* **by** *blast*

**have** *case1*: *slopeInfinite x p* ⟶
(∃ *l p′* . ($p' \neq p$) ∧ *onLine p′ l* ∧ *onLine p l*
∧ (*l* ∩ *regularConeSet x* = {}))
**using** *xnotp lemSlopeInfiniteImpliesOutside*
**by** *blast*

**have** *case2*:
((*slopeFinite x p*) ∧ (∃ *v* ∈ *lineVelocity* (*lineJoining x p*) . *sNorm2 v* > *1*))
⟶ (∃ *l p′* . ($p' \neq p$) ∧ *onLine p′ l* ∧ *onLine p l*
∧ (*l* ∩ *regularConeSet x* = {}))
**proof** −
**define** *l* **where** *l*: *l* = *lineJoining x p*
**hence** *onl*: *onLine x l* ∧ *onLine p l* **using** *lemLineJoiningContainsEndPoints* **by** *blast*

{ **assume** *hyp*: (*slopeFinite x p*) ∧
(∃ *v* ∈ *lineVelocity* (*lineJoining x p*) . *sNorm2 v* > *1*)
**then obtain** *v* **where** *v*: *v* ∈ *lineVelocity l* ∧ *sNorm2 v* > *1*
**using** *l* **by** *blast*

**define** *x0* **where** *x0*: *x0* = *mkPoint* (*tval p*) (*xval x*) (*yval x*) (*zval x*)
**define** *dsqr* **where** *dsqr*: *dsqr* = *norm2* ($p \ominus x0$)
**define** *d* **where** *d*: *d* = *sqrt dsqr*

**have** *dExists*: *hasRoot dsqr* **using** *dsqr lemNorm2NonNegAxEField* **by** *auto*

**have** *xnotp*: $x \neq p$ **using** *hyp* **by** *auto*

**have** *dnot0*: $d \neq 0$
**proof** −
{ **assume** *d0*: *d* = *0*

**hence** *dsqr* = *0* **using** *lemSquareOfSqrt*[*of dsqr d*] *dExists d* **by** *auto*
**hence** (*p*⊖*x0*) = *origin* **using** *dsqr lemNullImpliesOrigin*[*of* (*p*⊖*x0*)] **by** *auto*
**hence** *p* = *x0* **by** *simp*
**hence** *sloper x p* = ((*1*/(*tval x* − *tval p*))⊗(*x*⊖*x0*)) **using** *x0* **by** *auto*
**moreover have** *sComponent* (*x*⊖*x0*) = *sOrigin* **using** *x0* **by** *simp*
**ultimately have** *velocityJoining x p* = *sOrigin* **using** *hyp* **by** *auto*
**hence** *sOrigin* ∈ *lineVelocity l*
**using** *lemLineVelocityUsingPoints*[*of x p l*] *l hyp xnotp onl* **by** *auto*
**hence** *sOrigin* = *v*
**using** *lemFiniteLineVelocityUnique*[*of sOrigin l v*]
*hyp v onl xnotp* **by** *blast*
**hence** *sNorm2 v* = *0* **by** *auto*
**hence** *False* **using** *v* **by** *auto*
**}**
**thus** *?thesis* **by** *auto*
**qed**

**hence** *dsqrnot0*: *dsqr* ≠ *0*
**using** *d dExists lemSquareOfSqrt*[*of dsqr d*] *lemZeroRoot* **by** *blast*

**have** *dpos*: *d* > *0*
**using** *d theI'*[*of isNonNegRoot dsqr*] *lemSqrt dExists dnot0* **by** *auto*

**define** *T* **where** *T*: *T* = *tval p*
**define** *radius* **where** *radius*: *radius* = *tval p* − *tval x*
**define** *R0* **where** *R0*: *R0* = *sComponent* (*p*⊖*x*)

**have** *R0gtRadius*: *sqr radius* < *sNorm2 R0* **using** *formula radius R0* **by** *auto*

**have** *dsqr'*: *dsqr* = *sNorm2 R0*
**proof** −
**have** *sComponent x* = *sComponent x0* **using** *x0* **by** *simp*
**hence** *R0* = *sComponent* (*p* ⊖ *x0*) **using** *R0* **by** *auto*
**moreover have** *tval* (*p*⊖*x0*) = *0* **using** *x0* **by** *simp*
**ultimately show** *?thesis* **using** *lemNorm2Decomposition dsqr* **by** *auto*
**qed**

**hence** *radialnot0*: *R0* ≠ *sOrigin* **using** *dsqrnot0* **by** *auto*

**obtain** *D0* **where** *D0*: $D0 \neq sOrigin \ \wedge ((D0 \odot s\ R0) = 0)$
**using** *lemOrthogalSpaceVectorExists*[*of R0*] **by** *auto*

**define** *D* **where** *D*: $D = stPoint\ 0\ D0$
**define** *L* **where** *L*: $L = line\ p\ D$

**hence** *pOnLine*: *onLine p L*
**using** *lemLineJoiningContainsEndPoints*[*of L p* $(p \oplus D)$] **by** *auto*

**have** *meetEmpty*: $L \cap regularConeSet\ x = \{\}$
**proof** –
{ **assume** $L \cap regularConeSet\ x \neq \{\}$
**then obtain** *Q* **where** *Q*: $Q \in L \cap regularConeSet\ x$ **by** *blast*
**then obtain** $\alpha$ **where** $\alpha$: $Q = (p \oplus (\alpha \otimes D))$ **using** *L* **by** *blast*

**have** $((p \oplus (\alpha \otimes D)) \ominus x) = ((p \ominus x) \oplus (\alpha \otimes D))$
**using** *add-diff-eq diff-add-eq* **by** *auto*
**hence** *Qmx*: $(Q \ominus x) = ((p \ominus x) \oplus (\alpha \otimes D))$ **using** $\alpha$ **by** *simp*

**hence** *Qmxt*: $tval\ Q - tval\ x = tval\ (p \ominus x)$ **using** *D* **by** *simp*

**have** $sComponent\ (Q \ominus x) = sComponent\ ((p \ominus x) \oplus (\alpha \otimes D))$ **using** *Qmx* **by** *simp*
**also have** $\ldots = ((sComponent\ (p \ominus x)) \oplus s\ (sComponent\ (\alpha \otimes D)))$ **by** *simp*
**finally have** $sNorm2\ (sComponent\ (Q \ominus x))$
$= sNorm2\ ((sComponent\ (p \ominus x)) \oplus s\ (sComponent\ (\alpha \otimes D)))$ **by** *simp*
**also have** $\ldots = sNorm2\ (R0 \oplus s\ (\alpha \otimes s\ D0))$ **using** *R0 D* **by** *auto*
**also have** $\ldots = sNorm2\ R0 + 2*(R0 \odot s\ (\alpha \otimes s\ D0)) + sNorm2\ (\alpha \otimes s\ D0)$
**using** *lemSNorm2OfSum*[*of R0* $(\alpha \otimes s\ D0)$] **by** *auto*
**finally have**
$sNorm2\ (sComponent\ (Q \ominus x)) = sNorm2\ R0 + 2*(R0 \odot s\ (\alpha \otimes s\ D0)) + sNorm2\ (\alpha \otimes s\ D0)$
**by** *auto*

**moreover have** $(R0 \odot s\ (\alpha \otimes s\ D0)) = 0$
**using** *D0 lemSDotCommute lemSDotScaleRight* **by** *simp*

**moreover have** *sNorm2* ($\alpha \otimes s$ *D0*) $\geq$ *0* **by** *simp*
**ultimately have** *sNorm2* (*sComponent* ($Q \ominus x$)) $\geq$ *sNorm2* *R0* **by** *simp*

**hence** *Qmxs*: *sNorm2* (*sComponent* ($Q \ominus x$)) $>$ *sqr radius*
**using** *R0gtRadius* **by** *simp*

**hence** *sqr* (*tval Q* $-$ *tval x*) $<$ *sNorm2* (*sComponent* ($Q \ominus x$))
**using** *radius Qmxt* **by** *simp*
**hence** $\neg$ (*onRegularCone x Q*)
**using** *lemConeCoordinates*[*of x Q*] **by** *force*
**hence** $\neg$ (*regularCone x Q*) **using** *lemRegularCone* **by** *blast*
**hence** *False* **using** *Q* **by** *blast*
**}**
**thus** *?thesis* **by** *blast*
**qed**

**define** $p'$ **where** $p'$: $p' = (p \oplus D)$
**have** *Dnot0*: $D \neq$ *origin* **using** *D D0* **by** *auto*

**hence** $p' \neq p$
**proof** $-$
**{ assume** $p' = p$
**hence** $(p \oplus D) = p$ **using** $p'$ **by** *auto*
**hence** $((p \oplus D) \ominus p) =$ *origin* **by** *simp*
**hence** $D =$ *origin* **using** *add-diff-cancel* **by** *auto*
**hence** *False* **using** *Dnot0* **by** *auto*
**}**
**thus** *?thesis* **by** *blast*
**qed**
**moreover have** *onLine* $p'$ *L* **using** *L* $p'$ **by** *auto*
**ultimately have** *target1*: $p' \neq p \wedge$ *onLine* $p'$ *L* **by** *blast*

**hence** ($\exists$ $l\ p'$ . $(p' \neq p) \wedge$ *onLine* $p'\ l \wedge$ *onLine* $p\ l$
$\wedge$ ($l \cap$ *regularConeSet* $x = \{\}$)) **using** *meetEmpty pOnLine* **by** *blast*
**}**
**thus** *?thesis* **by** *blast*
**qed**

**hence** ($\exists$ $l\ p'$ . $(p' \neq p) \wedge$ *onLine* $p'\ l \wedge$ *onLine* $p\ l$
$\wedge$ ($l \cap$ *regularConeSet* $x = \{\}$))
**using** *cases case1* **by** *blast*
**}**
**hence** *l2r*: *outsideRegularCone* $x\ p \longrightarrow$
($\exists$ $l\ p'$ . $(p' \neq p) \wedge$ *onLine* $p'\ l \wedge$ *onLine* $p\ l$
$\wedge$ ($l \cap$ *regularConeSet* $x = \{\}$))

```
    by blast

  thus ?thesis by blast
qed


lemma lemTimelikeInsideCone:
  assumes insideRegularCone x p
shows     timelike (p ⊖ x)
proof -
  have tval p - tval x ≠ 0 using assms by auto
  hence tdiffpos: sqr (tval p - tval x) > 0 using lemSquaresPositive
by auto

  define l where l: l = lineJoining x p
  hence slopeFinite x p ∧ (∃ v . v ∈ lineVelocity l ∧ sNorm2 v < 1)
    using assms by auto
  then obtain v where v: v ∈ lineVelocity l ∧ sNorm2 v < 1
    using assms by blast
  have lineVelocity l = { velocityJoining x p }
    using lemLineVelocityUsingPoints[of x p l] assms
          lemLineJoiningContainsEndPoints l
    by blast
  hence vv: v = velocityJoining x p ∧ sNorm2 v < 1 using v by auto
  hence formula: sqr (tval p - tval x)*(sNorm2 v) = sNorm2 (sComponent
(p⊖x))
    using lemSNorm2VelocityJoining[of x p v] assms by blast

  have cases: sNorm2 v = 0 ∨ sNorm2 v > 0
    using local.add-less-zeroD local.not-less-iff-gr-or-eq
          local.not-square-less-zero
    by blast

  have case1: sNorm2 v > 0 ⟶ timelike (p ⊖ x)
  proof -
    define snv where snv: snv = sNorm2 v
    { assume sNorm2 v > 0
      hence 0 < snv < 1 using vv snv by auto
     moreover have sqr (tval p - tval x)*snv = sNorm2 (sComponent
(p⊖x))
        using formula snv by simp
      ultimately have sqr (tval p - tval x) > sNorm2 (sComponent
(p⊖x))
        using lemMultPosLT[of sqr (tval p - tval x) snv]
              tdiffpos by force
      hence timelike (p⊖x) by auto
    }
    thus ?thesis using snv by auto
  qed
```

{ **assume** *sNorm2 v* = *0*
  **hence** *sNorm2* (*sComponent* ($p \ominus x$)) = *0* **using** *formula* **by** *auto*
  **hence** *timelike* ($p \ominus x$) **using** *tdiffpos* **by** *auto*
}
**hence** *case2*: *sNorm2 v* = *0* $\longrightarrow$ *timelike* ($p \ominus x$) **by** *auto*

**thus** *?thesis* **using** *case1 cases* **by** *auto*
**qed**

**end**
**end**

# 31 ReverseCauchySchwarz

This theory defines and proves the "reverse" Cauchy-Schwarz inequality for timelike vectors in the Minkowski metric.

**theory** *ReverseCauchySchwarz*
  **imports** *CauchySchwarz*
**begin**

Rather than construct the proof, one could simply have asserted the claim as an axiom. We did this during development of the main proof, and then returned to this section later. In practice the axiom we chose to assert contained far more information than required, because we eventually found a proof that only required consideration of timelike vectors (our axiom considered lightlike vectors as well).

**class** *ReverseCauchySchwarz* = *CauchySchwarz*

**begin**

**lemma** *lemTimelikeNotZeroTime*:
  **assumes** *timelike v*
  **shows** *tval v* $\neq$ *0*
**proof** –
  { **assume** *converse*: *tval v* = *0*
    **have** *sNorm2* (*sComponent v*) < *sqr* (*tval v*) **using** *assms* **by** *auto*
    **hence** *sNorm2* (*sComponent v*) < *0* **using** *converse* **by** *auto*
    **hence** *False* **using** *local.add-less-zeroD local.not-square-less-zero*
**by** *blast*
  }
  **thus** *?thesis* **by** *auto*
**qed**

**lemma** *lemOrthogmToTimelike*:
**assumes** *timelike u*
**and** *orthogm u v*
**and** $v \neq$ *origin*
**shows** *spacelike v*
**proof** −
**have** *tvalu*: *tval u* $\neq 0$ **using** *assms*(*1*) *lemTimelikeNotZeroTime* **by** *auto*

**define** *us* **where** *us*: *us* = *sComponent u*
**define** *vs* **where** *vs*: *vs* = *sComponent v*
**have** *sqr* ((*tval u*) ∗ (*tval v*)) = *sqr* (*us* ⊙*s vs*) **using** *assms*(*2*) *us vs* **by** *auto*
**also have** . . . ≤ *sNorm2 us* ∗ *sNorm2 vs* **using** *lemCauchySchwarzSqr* **by** *auto*
**finally have** *inequ*: *sqr* (*tval u*) ∗ *sqr* (*tval v*) ≤ *sNorm2 us* ∗ *sNorm2 vs*
**using** *mult-assoc mult-commute* **by** *auto*

**have** *ifvsnz*: *vs* ≠ *sOrigin* ⟶ *sNorm2 vs* > *0*
**by** (*meson local.add-less-zeroD local.antisym-conv3 local.lemSpatialNullImpliesSpatialOrigin local.not-square-less-zero*)

**have** *iftv0*: *tval v* = *0* ⟶ *spacelike v*
**proof** −
{ **assume** *v0*: *tval v* = *0*
**hence** *vs* ≠ *sOrigin* **using** *assms vs* **by** *auto*
**hence** *sNorm2 vs* > *0* **using** *ifvsnz* **by** *auto*
**hence** *spacelike v* **using** *v0 vs*
**by** (*metis local.less-iff-diff-less-0 local.mult-not-zero*)
}
**thus** *?thesis* **by** *auto*
**qed**

**moreover have** (*tval v* ≠ *0* ∧ *vs* ≠ *sOrigin*) ⟶ *spacelike v*
**proof** −
{ **assume** *vnz*: (*tval v* ≠ *0* ∧ *vs* ≠ *sOrigin*)

**have** *utpos*: *sqr* (*tval u*) > *0* **using** *tvalu lemSquaresPositive* **by** *simp*
**have** *vspos*: *sNorm2 vs* > *0* **using** *vnz ifvsnz* **by** *auto*

**have** *sqr* (*tval u*) ∗ *sqr* (*tval v*) ≤ *sNorm2 us* ∗ *sNorm2 vs* **using** *inequ* **by** *simp*
**hence** *sqr* (*tval v*) ≤ *sNorm2 us* ∗ *sNorm2 vs* / *sqr* (*tval u*)
**using** *utpos*
**by** (*metis local.divide-right-mono local.divisors-zero local.dual-order.strict-implies-order*

*local.nonzero-mult-div-cancel-left tvalu*)
**hence** *sqr* (*tval v*) / *sNorm2 vs* ≤ *sNorm2 us* / *sqr* (*tval u*)
**using** *vspos mult-commute* **by** (*simp add*: *local.mult-imp-div-pos-le*)

**moreover have** *sNorm2 us* / *sqr* (*tval u*) < *1* **using** *assms*(*1*) *us utpos* **by** *auto*
**ultimately have** *sqr* (*tval v*) / *sNorm2 vs* < *1* **by** *simp*
**hence** *spacelike v* **using** *vs vspos* **by** *auto*
**}**
**thus** *?thesis* **by** *auto*
**qed**

**moreover have** ¬ (*tval v* ≠ *0* ∧ *vs* = *sOrigin*)
**proof** −
**{ assume** (*tval v* ≠ *0* ∧ *vs* = *sOrigin*)
**hence** (*u* ⊙*m v*) ≠ *0* **using** *tvalu vs* **by** *auto*
**hence** *False* **using** *assms* **by** *auto*
**}**
**thus** *?thesis* **by** *auto*
**qed**

**ultimately show** *?thesis* **by** *blast*
**qed**

**lemma** *lemNormaliseTimelike*:
**assumes** *timelike v*
**and** *s* = *sComponent* ((*1*/*tval v*)⊗*v*)
**shows** (*0* ≤ *sNorm2 s* < *1*) ∧ (*tval* ((*1*/*tval v*)⊗*v*) = *1*)
**proof** −
**have** *sqr* (*tval v*) > *sNorm2* (*sComponent v*) **using** *assms* **by** *auto*
**hence** *1* > *sqr* (*1*/*tval v*) ∗ *sNorm2* (*sComponent v*)
**using** *local.divide-less-eq* **by** *force*
**hence** *sNorm2 s* < *1* **using** *lemSNorm2OfScaled*[*of 1*/*tval v sComponent v*] *assms*
**by** *auto*
**hence** (*0* ≤ *sNorm2 s* < *1*) **by** *simp*
**moreover have** (*tval* ((*1*/*tval v*)⊗*v*) = *1*)
**proof** −
**have** *sqr* (*tval v*) > *sNorm2* (*sComponent v*) **using** *assms* **by** *auto*
**hence** *sqr* (*tval v*) ≠ *0*
**by** (*metis local.add-less-zeroD local.not-square-less-zero*)
**hence** *tval v* ≠ *0* **by** *auto*
**thus** *?thesis* **by** *auto*
**qed**
**ultimately show** *?thesis* **by** *blast*
**qed**

**lemma** *lemReverseCauchySchwarz*:
**assumes** *timelike X* $\wedge$ *timelike D*
**shows** *sqr* ($X \odot m$ $D$) $\geq$ (*mNorm2 X*)$*$(*mNorm2 D*)
**proof** −
**have** *case1*: *parallel X D* $\longrightarrow$ *?thesis*
**proof** −
**{ assume** *parallel X D*
**then obtain** *a* **where** *a*: $X = (a \otimes D)$ **by** *auto*
**hence** ($X \odot m$ $D$) = $a *$ *mNorm2 D* **using** *lemMDotScaleLeft* **by** *auto*
**moreover have** *mNorm2 X* = (*sqr a*) $*$ *mNorm2 D* **using** *lemMNorm2OfScaled a* **by** *auto*
**ultimately have** *sqr* ($X \odot m$ $D$) = (*mNorm2 X*)$*$(*mNorm2 D*)
**using** *local.lemSqrMult mult-assoc* **by** *auto*
**}**
**thus** *?thesis* **by** *simp*
**qed**

**have** ($\neg$ *parallel X D*) $\longrightarrow$ *?thesis*
**proof** −
**{ assume** *case2*: $\neg$ (*parallel X D*)
**define** *u* **where** *u*: $u = ((1/tval\ X) \otimes X)$
**define** *v* **where** *v*: $v = ((1/tval\ D) \otimes D)$
**define** *su* **where** *su*: *su* = (*sComponent u*)
**define** *sv* **where** *sv*: *sv* = (*sComponent v*)

**have** *sphere*: ($0 \leq$ *sNorm2 su* $< 1$) $\wedge$ ($0 \leq$ *sNorm2 sv* $< 1$)
**using** *lemNormaliseTimelike u su v sv assms* **by** *blast*
**have** *tvals1*: *tval u* = *1* $\wedge$ *tval v* = *1*
**using** *lemNormaliseTimelike u su v sv assms* **by** *blast*

**have** *worksuv*: *sqr* ($u \odot m$ $v$) $>$ (*mNorm2 u*)$*$(*mNorm2 v*)
**proof** −

**have** *uupos*: *mNorm2 u* $> 0$ **using** *assms u lemNormaliseTimelike* **by** *auto*
**have** *vvpos*: *mNorm2 v*$> 0$ **using** *assms v lemNormaliseTimelike* **by** *auto*
**have** *uvpos*: ($u \odot m$ $v$) $> 0$
**proof** −
**have** *sqr* (*sdot su sv*) $\leq$ (*sNorm2 su*) $*$ (*sNorm2 sv*)
**using** *lemCauchySchwarzSqr* **by** *auto*
**also have** $\ldots < 1$
**using** *mult-le-one sphere local.mult-strict-mono* **by** *fastforce*
**finally have** *sqr* (*sdot su sv*) $< 1$ **by** *auto*
**hence** (*sdot su sv*) $< 1$
**using** *local.less-1-mult local.not-less-iff-gr-or-eq* **by** *fastforce*
**thus** *?thesis* **using** *u v su sv tvals1* **by** *auto*

**qed**

**define** *a* **where** *a*: $a = (u \odot m\ v)/(mNorm2\ v)$
**define** *up* **where** *up*: $up = (a \otimes v)$
**define** *uo* **where** *uo*: $uo = (u \ominus up)$

**have** *apos*: $a > 0$ **using** *a uvpos vvpos* **by** *auto*

**have** *updotup*: $mNorm2\ up > 0$
**proof** −
**have** $mNorm2\ up = (sqr\ a) * mNorm2\ v$ **using** *up lemM-Norm2OfScaled* **by** *auto*
**thus** *?thesis* **using** *apos lemSquaresPositive vvpos* **by** *auto*
**qed**

**have** *uparts*: $u = (up \oplus uo) \wedge parallel\ up\ v \wedge orthogm\ uo\ v \wedge (up \odot m\ v) = (u \odot m\ v)$
**using** *lemMDecomposition a up uo vvpos uvpos* **by** *auto*

**have** *updotuo*: $(up \odot m\ uo) = 0$
**proof** −
**have** $(up \odot m\ uo) = a*(v \odot m\ uo)$ **using** *up lemMDotScaleLeft* **by** *auto*
**moreover have** $(v \odot m\ uo) = (uo \odot m\ v)$ **using** *mult-commute* **by** *auto*
**ultimately have** $(up \odot m\ uo) = 0$ **using** *uparts* **by** *force*
**thus** *?thesis* **by** *auto*
**qed**

**have** *udotu*: $mNorm2\ u = mNorm2\ up + mNorm2\ uo$
**proof** −
**have** $mNorm2\ u = mNorm2\ (up \oplus uo)$ **using** *uparts* **by** *auto*
**also have** $\ldots = mNorm2\ up + 2*(up \odot m\ uo) + mNorm2\ uo$ **using** *lemMNorm2OfSum* **by** *auto*
**finally show** *?thesis* **using** *updotuo* **by** *auto*
**qed**

**moreover have** *uodotuo*: $mNorm2\ uo < 0$
**proof** −
**have** *timelike up* **using** *updotup* **by** *auto*
**moreover have** *orthogm up uo* **using** *updotuo* **by** *auto*
**moreover have** $uo \neq origin$
**proof** −
**define** $\alpha$ **where** $\alpha$: $\alpha = (tval\ X)*a*(1/tval\ D)$
**have** $\alpha pos$: $\alpha \neq 0$ **using** *apos lemTimelikeNotZeroTime assms* $\alpha$ **by** *simp*

**{ assume** $uo = origin$
**hence** $u = (a \otimes v)$ **using** *uo up* **by** *auto*

**moreover have** $X = ((tval\ X) \otimes u)$
**using** *u lemScaleAssoc assms lemTimelikeNotZeroTime*
**by** *auto*
**ultimately have** $X = ((tval\ X) \otimes (a \otimes v))$ **by** *auto*
**hence** $X = ((tval\ X) \otimes (a \otimes ((1/tval\ D) \otimes D)))$ **using** *v* **by**
*auto*
**hence** $X = (\alpha \otimes D)$ **using** $\alpha$ *lemScaleAssoc mult-assoc*
**by** (*metis Point.select-convs*(*3*−*4*))
**hence** *False* **using** *case2* $\alpha$*pos* **by** *blast*
**}**
**thus** *?thesis* **by** *auto*
**qed**
**ultimately show** *?thesis* **using** *lemOrthogmToTimelike* **by**
*auto*
**qed**

**ultimately have** *upgeu*: *mNorm2 up* > *mNorm2 u* **by** *auto*

**have** $(u \odot m\ v) = (up \odot m\ v)$ **using** *uparts* **by** *auto*
**also have** $\ldots = a * mNorm2\ v$ **using** *up lemMDotScaleLeft* **by**
*auto*
**finally have** *final*: $sqr\ (u \odot m\ v) = ((sqr\ a) * mNorm2\ v) *$
$(mNorm2\ v)$
**using** *lemSqrMult*[*of a mNorm2 v*] *mult-assoc* **by** *auto*

**hence** $sqr\ (u \odot m\ v) = (mNorm2\ up) * (mNorm2\ v)$ **using**
*lemMNorm2OfScaled up* **by** *auto*
**thus** *?thesis*
**using** *upgeu vvpos local.mult-strict-right-mono* **by** *simp*
**qed**

**have** $(u \odot m\ v) = (((1/tval\ X) \otimes X) \odot m\ ((1/tval\ D) \otimes D))$ **using**
*u v* **by** *auto*
**hence** *udotv*: $(u \odot m\ v) = (1/tval\ X) * (1/tval\ D) * (X \odot m\ D)$
**using** *lemMDotScaleRight lemMDotScaleLeft mult-assoc mult-commute*
**by** *metis*

**have** *udotu*: $mNorm2\ u = sqr\ (1/tval\ X) * mNorm2\ X$ **using** *u*
*lemMNorm2OfScaled* **by** *blast*
**moreover have** *vdotv*: $mNorm2\ v = sqr\ (1/tval\ D) * mNorm2$
$D$ **using** *v lemMNorm2OfScaled* **by** *blast*
**ultimately have** $(mNorm2\ u) * (mNorm2\ v) = sqr\ ((1/tval$
$X) * (1/tval\ D)) * mNorm2\ X * mNorm2\ D$
**using** *mult-commute mult-assoc* **by** *auto*

**hence**
$sqr\ ((1/tval\ X) * (1/tval\ D) * (X \odot m\ D)) >$
$sqr\ ((1/tval\ X) * (1/tval\ D)) * mNorm2\ X *$

*mNorm2 D*
    **using** *worksuv udotv* **by** *auto*
   **moreover have** *sqr* $((1/tval\ X)*(1/tval\ D)) > 0$
    **using** *lemTimelikeNotZeroTime*
   **by** (*metis calculation local.lemSquaresPositive local.mult-cancel-left1*)
   **ultimately have** *?thesis*
    **using** *mult-less-cancel-left-pos*[*of sqr* $((1/tval\ X)*(1/tval\ D))$]
    **by** *auto*
  **}**

  **thus** *?thesis* **by** *auto*
 **qed**

 **thus** *?thesis* **using** *case1* **by** *auto*
**qed**

**end**

**end**

# 32 KeyLemma

This theory establishes a "key lemma": if you draw a line through a point inside a cone, that line will intersect the cone in no fewer than 1 and no more than 2 points.

**theory** *KeyLemma*
 **imports** *Classification ReverseCauchySchwarz*
**begin**

**class** *KeyLemma* = *Classification* + *ReverseCauchySchwarz*
**begin**

**lemma** *lemInsideRegularConeImplies*:
 **assumes** *insideRegularCone x p*
**and** $D \neq origin$
**and** $l = line\ p\ D$
**shows** $0 < card\ (l \cap regularConeSet\ x) \leq 2$
**proof** −
 **define** *S* **where** *S*: $S = (l \cap regularConeSet\ x)$
 **define** *X* **where** *X*: $X = (p \ominus x)$
 **define** *a* **where** *a*: $a = mNorm2\ D$
 **define** *b* **where** *b*: $b = 2*(tval\ X)*(tval\ D) - 2*((sComponent\ D)$
$\odot s\ (sComponent\ X))$

**define** *c* **where** *c*: $c = mNorm2\ X$
**define** *d* **where** *d*: $d = (sqr\ b) - (4*a*c)$

**have** *tlX*: *timelike X* **using** *lemTimelikeInsideCone assms(1) X* **by** *auto*
**hence** *cpos*: $c > 0$ **using** *c* **by** *auto*

**have** *xnotp*: $x \neq p$ **using** *assms(1)* **by** *auto*

**have** *aval*: $a = mNorm2\ D$ **using** *a* **by** *auto*
**have** *bval*: $b = 2*(X \odot m\ D)$
  **using** *b local.lemSDotCommute local.right-diff-distrib′ mult-assoc*
  **using** *local.mdot.simps* **by** *presburger*
**have** *cval*: $c = mNorm2\ X$ **using** *c* **by** *auto*
**have** *dval*: $d = 4 * ( (sqr\ (X \odot m\ D)) - (mNorm2\ X)*(mNorm2\ D) )$
**proof** −
  **have** $d = (sqr\ b) - (4*a*c)$ **using** *d* **by** *simp*
  **moreover have** $(sqr\ b) = 4*(sqr\ (X \odot m\ D))$
    **using** *lemSqrMult[of 2 (X ⊙m D)] bval* **by** *auto*
  **moreover have** $4*a*c = 4*(mNorm2\ X)*(mNorm2\ D)$
    **using** *aval cval mult-commute mult-assoc* **by** *auto*
  **ultimately show** *?thesis* **using** *right-diff-distrib′ mult-assoc* **by** *metis*
**qed**

**define** *r2p* **where** *r2p*: $r2p = (\lambda\ r\ .\ (p \oplus (r \otimes D)))$
**define** *p2r* **where** *p2r*: $p2r = (\lambda\ q\ .\ (THE\ a\ .\ q = (p \oplus (a \otimes D))))$

**have** *bij*: $\forall\ r\ q\ .\ r2p\ r = q \longleftrightarrow (\exists\ w\ .\ (r2p\ w = q)) \wedge (p2r\ q = r)$
**proof** −
  **have** *uniqueroots*: $\forall\ a\ r\ .\ r2p\ a = r2p\ r \longrightarrow a = r$
  **proof** −
    **{ fix** *a r* **assume** $r2p\ a = r2p\ r$
      **hence** $(a \otimes D) = (r \otimes D)$ **using** *r2p add-diff-eq* **by** *auto*
      **hence** $((a-r) \otimes D) = origin$ **using** *lemScaleDistribDiff* **by** *auto*
      **hence** $(a-r) = 0$ **using** *assms(2)* **by** *auto*
      **hence** $a = r$ **by** *simp*
    **}**
    **thus** *?thesis* **by** *blast*
  **qed**
  **{ fix** *q r* **assume** *lhs*: $r2p\ r = q$
    **have** $(THE\ a\ .\ q = r2p\ a) = r$
    **proof** −
      **{ fix** *a* **assume** $q = r2p\ a$
        **hence** $a = r$ **using** *uniqueroots lhs r2p* **by** *blast*
      **}**
      **hence** $\forall\ a\ .\ q = r2p\ a \longrightarrow a = r$ **by** *auto*

**thus** *?thesis* **using** *lhs the-equality*[*of* $\lambda a$ . $q = r2p\ a\ r$]
**by** *force*
**qed**
**}**
**hence** *l2r*: $\forall\ q\ r\ .\ r2p\ r = q \longrightarrow (\exists\ w\ .\ (r2p\ w = q)) \wedge (p2r\ q = r)$
**using** *p2r r2p* **by** *blast*

**{ fix** *r q* **assume** *ass*: $(\exists\ w\ .\ (r2p\ w = q)) \wedge (p2r\ q = r)$
**then obtain** *w* **where** *w*: $r2p\ w = q$ **by** *blast*
**hence** *unique*: $\forall\ a\ .\ q = r2p\ a \longrightarrow a = w$ **using** *uniqueroots* **by** *auto*
**have** *rdef*: $r = (THE\ a\ .\ q = r2p\ a)$ **using** *ass r2p p2r* **by** *simp*

**have** $q = r2p\ w$ **using** *w* **by** *simp*
**hence** $q = r2p\ r$ **using** *theI*[*of* $\lambda$ *a*. $q = r2p\ a\ w$] *rdef unique*
**by** *blast*
**}**
**hence** $\forall\ q\ r\ .\ (\exists\ w\ .\ (r2p\ w = q)) \wedge (p2r\ q = r) \longrightarrow q = r2p\ r$
**by** *blast*

**thus** *?thesis* **using** *l2r* **by** *blast*
**qed**

**have** *equalr2p*: $\forall\ x\ y\ .\ r2p\ x = r2p\ y \longrightarrow x = y$ **using** *bij* **by** *metis*

**have** *SbijRoots*: $S = \{\ y\ .\ \exists\ x \in qroots\ a\ b\ c\ .\ y = r2p\ x\ \}$
**proof** −
**{ fix** *y* **assume** *y*: $y \in S$
**then obtain** *r* **where** *r*: $y = r2p\ r$ **using** *r2p S assms* **by** *blast*
**hence** *regularCone x* $(p \oplus (r \otimes D))$ **using** *r2p y S* **by** *auto*
**hence** $r \in qroots\ a\ b\ c$
**using** *lemWhereLineMeetsCone*[*of a D b p x c r*]
*a b c X* **by** *auto*
**hence** $\exists\ r \in qroots\ a\ b\ c\ .\ y = r2p\ r$ **using** *r* **by** *blast*
**}**
**hence** *l2r*: $S \subseteq \{\ y\ .\ \exists\ x \in qroots\ a\ b\ c\ .\ y = r2p\ x\ \}$ **by** *blast*
**{ fix** *y* **assume** *y*: $y \in \{\ y\ .\ \exists\ x \in qroots\ a\ b\ c\ .\ y = r2p\ x\ \}$
**then obtain** *r* **where** *r*: $r \in qroots\ a\ b\ c \wedge y = r2p\ r$ **by** *blast*
**hence** *regularCone x* $(r2p\ r)$
**using** *lemWhereLineMeetsCone*[*of a D b p x c r*]
*a b c X r2p* **by** *auto*
**moreover have** $r2p\ r \in l$ **using** *assms(3) r2p* **by** *auto*
**ultimately have** $y \in S$ **using** *S r* **by** *auto*
**}**
**thus** *?thesis* **using** *l2r* **by** *blast*
**qed**

```
 have equalcard: ((card (qroots a b c) = 1) ∨ (card (qroots a b c) =
2))
                              ⟶ (card S = card (qroots a b c))
 proof −
  { assume cases: card (qroots a b c) = 1 ∨ card (qroots a b c) = 2

    have case1: card (qroots a b c) = 1 ⟶ (card S = card (qroots
a b c))
    proof −
     { assume card1: card (qroots a b c) = 1
     hence ∃ r . (qroots a b c) = {r} by (meson card-1-singletonE)
      then obtain r where r: (qroots a b c) = {r} by blast
      hence l2r: { r2p r } ⊆ S using SbijRoots by auto
      { fix y assume y: y ∈ S
        then obtain x where x: x ∈ qroots a b c ∧ y = r2p x
          using SbijRoots by blast
        hence r2p r = y using bij using r by auto
      }
      hence ∀ y . y ∈ S  ⟶ y ∈ { r2p r } by auto
      hence S = { r2p r } using l2r by blast
      hence ∃ r . S = {r} by blast
      hence card S = 1
        using card-1-singleton-iff[of S] by auto
     }
     thus ?thesis by auto
    qed

    have case2: card (qroots a b c) = 2 ⟶ (card S = card (qroots
a b c))
    proof −
     { assume card2: card (qroots a b c) = 2
       hence ∃ r1 r2 . (qroots a b c) = {r1, r2} ∧ r1 ≠ r2
         using card-2-iff by blast
         then obtain r1 r2 where rs: (qroots a b c) = {r1,r2} ∧
r1≠r2 by blast
       hence l2r: { r2p r1, r2p r2} ⊆ S using SbijRoots by auto
       { fix y assume y: y ∈ S
         then obtain x where x: x ∈ qroots a b c ∧ y = r2p x
           using SbijRoots by blast
         hence x = r1 ∨ x = r2 using rs by auto
         hence r2p r1 = y ∨ r2p r2 = y using x by blast
       }
       hence ∀ y . y ∈ S  ⟶ y ∈ { r2p r1, r2p r2 } by auto
       hence S2: S = { r2p r1, r2p r2 } using l2r by blast
       moreover have r2p r1 ≠ r2p r2 using rs bij by metis
       ultimately have ∃ y1 y2 . S = {y1, y2} ∧ y1≠y2 by blast
       hence card S = 2 using card-2-iff by blast
     }
     thus ?thesis by auto
```

**qed**

**hence** (*card S* = *card* (*qroots a b c*)) **using** *case1 cases* **by** *auto*
**}**
**thus** *?thesis* **by** *auto*
**qed**

**have** *qc1*: ¬ *qcase1 a b c* **using** *cpos* **by** *auto*

**have** *qc2*: ¬ *qcase2 a b c*
**proof** −
**{ assume** *qcase2 a b c*
**hence** *qc2*: $a = 0 \wedge b = 0 \wedge c > 0$ **using** *d cpos* **by** *auto*

**have** *llD*: *lightlike D* **using** *qc2 aval assms*(*2*) **by** *simp*

**have** *sqr* ($X \odot m$ *D*) = (*mNorm2 X*)∗(*mNorm2 D*)
**using** *qc2 bval aval* **by** *simp*
**hence** *orthogm X D* **using** *llD lemSqrt0* **by** *auto*
**hence** *parXD*: *parallel X D*
**using** *lemCausalOrthogmToLightlikeImpliesParallel tlX llD* **by**
*auto*

**then obtain** $\alpha$ **where** $\alpha$: $\alpha \neq 0 \wedge X = (\alpha \otimes D)$ **by** *blast*

**have** *Dnot0*: *origin* ≠ *D* **using** *assms*(*2*) **by** *simp*
**hence** *lightlike X*
**proof** −
**have** *tsqr*: *sqr* (*tval X*) = (*sqr* $\alpha$)∗ *sqr* (*tval D*)
**using** *lemSqrMult* $\alpha$ **by** *simp*
**have** *sComponent X* = ($\alpha$ ⊗*s* (*sComponent D*)) **using** $\alpha$ **by**
*simp*
**hence** *sNorm2* (*sComponent X*) = (*sqr* $\alpha$) ∗ *sNorm2* (*sComponent*
*D*)
**using** *lemSNorm2OfScaled*[*of* $\alpha$ *sComponent D*] **by** *auto*
**hence** *mNorm2 X* = (*sqr* $\alpha$) ∗ *mNorm2 D*
**using** *lemMNorm2Decomposition*[*of X*] *tsqr*
**by** (*simp add*: *local.right-diff-distrib′*)
**thus** *?thesis* **using** *llD qc2 xnotp X* **by** *auto*
**qed**

**hence** *False* **using** *tlX* **by** *auto*
**}**
**thus** *?thesis* **by** *auto*
**qed**

**have** *qc3*: *qcase3 a b c* $\longrightarrow$ *card S* = *1*
**proof** −
  **{ assume** *qcase3 a b c*
    **hence** *qc3*: *qroots a b c* = {(−*c*/*b*)} **using** *lemQCase3* **by** *auto*
    **hence** ∃ *val* . (*qroots a b c* = {*val*}) **by** *simp*
    **hence** *card* (*qroots a b c*) = *1* **using** *card-1-singleton-iff* **by** *auto*
    **hence** *card S* = *1* **using** *equalcard* **by** *auto*
  **}**
  **thus** *?thesis* **by** *auto*
**qed**

**have** *qc4*: ¬ *qcase4 a b c*
**proof** −
  **{ assume** *qcase4 a b c*
    **hence** *qc4*: *a* ≠ *0* ∧ *d* < *0* **using** *d* **by** *auto*
    **{ assume** *a* > *0*
      **hence** *tlD*: *timelike D* **using** *aval* **by** *auto*

      **hence** *sqr* (*X* ⊙*m D*) ≥ (*mNorm2 X*)∗(*mNorm2 D*)
        **using** *lemReverseCauchySchwarz*[*of X D*] *tlX*
        **using** *local.dual-order.order-iff-strict* **by** *blast*
      **hence** *EQN*: *4*∗*sqr* (*X* ⊙*m D*) ≥ *4*∗(*mNorm2 X*)∗(*mNorm2*
*D*)
        **using** *qc4 d dval local.leD* **by** *fastforce*

      **have** (*sqr b*) < *4*∗*a*∗*c* **using** *d qc4* **by** *simp*
      **hence** *4*∗*sqr* (*X* ⊙*m D*) < *4*∗(*mNorm2 X*)∗(*mNorm2 D*)
        **using** *aval bval cval mult-assoc mult-commute*
            *lemSqrMult*[*of 2* (*X* ⊙*m D*)] **by** *auto*

      **hence** *False* **using** *EQN* **by** *force*
    **}**
    **hence** *aneg*: *a* < *0* **using** *qc4* **by** *force*
    **hence** *4*∗*a*∗*c* < *0* **using** *cpos*
      **by** (*simp add*: *local.mult-pos-neg local.mult-pos-neg2*)
    **hence** *d* > *sqr b* **using** *d*
    **by** (*metis add-commute local.add-less-same-cancel2 local.diff-add-cancel*)
    **hence** *d* > *0*
      **using** *local.less-trans local.not-square-less-zero qc4* **by** *blast*
    **hence** *False* **using** *qc4* **by** *auto*
  **}**
  **thus** *?thesis* **by** *auto*
**qed**

**have** *qc5*: *qcase5 a b c* $\longrightarrow$ *card S* = *1*
**proof** −
  **{**

```
    assume qc5: qcase5 a b c
    hence qroots a b c = {(−b/(2∗a))} using lemQCase5 by auto
    hence ∃ val . qroots a b c = {val} by simp
    hence card (qroots a b c) = 1 using card-1-singleton-iff by auto
    hence card S = 1 using equalcard by simp
  }
  thus ?thesis by simp
qed

have qc6: qcase6 a b c ⟶ card S = 2
proof −
  { define rd where rd: rd = sqrt (discriminant a b c)
    define rp where rp: rp = (−b + rd) / (2 ∗ a)
    define rm where rm: rm = (−b − rd) / (2 ∗ a)

    assume qc6: qcase6 a b c
    hence rp ≠ rm ∧ qroots a b c = {rp, rm}
      using lemQCase6[of a b c rd rp rm] a b c rd rm rp
      by auto

    hence ∃ v1 v2 . qroots a b c = { v1, v2 } ∧ (v1 ≠ v2) by blast
    hence card (qroots a b c) = 2 using card-2-iff[of qroots a b c]
by blast
    hence card S = 2 using equalcard by simp
  }
  thus ?thesis by simp
qed

define n where n: n = card S
hence (n = 1 ∨ n = 2)
  using qc1 qc2 qc3 qc4 qc5 qc6 lemQuadraticCasesComplete
  by blast
hence 0 < n ≤ 2 by auto
thus ?thesis using n S by auto
qed

end
end
```

# 33 Cardinalities

For our purposes the only relevant cardinalities are 0, 1, 2 and more-than-2 (a proxy for "infinite"). We will use these cardinalities when looking at how lines intersect cones, using the size of the intersection set to characterise whether points are inside, on or outside of lightcones.

```
theory Cardinalities
  imports Functions
```

**begin**

**class** *Cardinalities* = *Functions*
**begin**

**lemma** *lemInjectiveValueUnique*:
**assumes** *injective f*
**and** *isFunction f*
**and** *f x y*
**shows** { *q*. *f x q* } = { *y* }
**using** *assms*(*2*) *assms*(*3*) **by** *force*

**lemma** *lemBijectionOnTwo*:
**assumes** *bijective f*
**and** *isFunction f*
**and** $s \subseteq$ *domain f*
**and** *card s* = *2*
**shows** *card* (*applyToSet f s*) = *2*
**proof** −
**obtain** *x y* **where** *xy*: $s = \{x,y\} \wedge x \neq y$ **using** *assms*(*4*)
**by** (*meson card-2-iff*)
**obtain** *fx* **where** *fx*: *f x fx* **using** *xy assms*(*1*) *assms*(*3*) **by** *blast*
**obtain** *fy* **where** *fy*: *f y fy* **using** *xy assms*(*1*) *assms*(*3*) **by** *blast*
**have** *applyToSet f s* = { *q* . $\exists$ *p* $\in$ *s* . *f p q* } **by** *simp*
**moreover have** ... = { *q*. *f x q* $\vee$ *f y q* } **using** *xy* **by** *auto*
**moreover have** ... = { *q*. *f x q* } $\cup$ { *q*. *f y q* } **by** *auto*
**ultimately have** *applyToSet f s* = { *fx* } $\cup$ { *fy* }
**using** *fx fy assms*(*1*) *assms*(*2*) *lemInjectiveValueUnique* **by** *force*
**moreover have** *fx* $\neq$ *fy* **using** *fx fy assms*(*1*) *xy* **by** *blast*
**thus** *?thesis* **using** *calculation* **by** *force*
**qed**

**lemma** *lemElementsOfSet2*:
**assumes** *card S* = *2*
**shows** $\exists$ *p q* . (*p* $\neq$ *q*) $\wedge$ *p* $\in$ *S* $\wedge$ *q* $\in$ *S*
**by** (*meson assms card-2-iff'*)

**lemma** *lemThirdElementOfSet2*:
**assumes** (*p* $\neq$ *q*) $\wedge$ *p* $\in$ *S* $\wedge$ *q* $\in$ *S* $\wedge$ (*card S* = *2*)
**and** *r* $\in$ *S*
**shows** *p* = *r* $\vee$ *q* = *r*
**proof** −
**have** *card S* = *2* **using** *assms*(*1*) **by** *auto*

```
  then obtain x y where xy: (x ∈ S) ∧ (y ∈ S) ∧ (x ≠ y) ∧ (∀ z∈S.
z = x ∨ z = y)
    using card-2-iff'[of S] by auto
  have p: p = x ∨ p = y using xy assms(1) by auto
  have q: q = x ∨ q = y using xy assms(1) by auto
  hence pq: (p = x ∧ q = y) ∨ (p = y ∧ q = x) using assms(1) p
by blast
  moreover have r = x ∨ r = y using xy assms(2) by auto
  ultimately show ?thesis by auto
qed


lemma lemSmallCardUnderInvertible:
  assumes invertible f
and      0 < card S ≤ 2
shows card S = card (applyToSet (asFunc f) S)
proof −
  have cases: card S = 1 ∨ card S = 2 using assms(2) by auto

  have case1: card S = 1 ⟶ ?thesis
  proof −
    { assume card1: card S = 1
      hence ∃ a . S = {a} by (meson card-1-singletonE)
      then obtain a where a: S = {a} by blast
      define b where b: b = f a
      hence applyToSet (asFunc f) S = { b }
      proof −
        have {b} ⊆ applyToSet (asFunc f) S using a b by auto
        moreover have applyToSet (asFunc f) S ⊆ {b}
        proof −
          { fix c assume c: c ∈ applyToSet (asFunc f) S
            hence c ∈ { c . ∃ a' ∈ S . (asFunc f) a' c } by auto
           then obtain a' where a': a' ∈ S ∧ (asFunc f) a' c by blast
            hence a' = a ∧ f a = c using a by auto
            hence c ∈ {b} using b by auto
          }
          thus ?thesis by blast
        qed
        ultimately show ?thesis by blast
      qed
      hence ∃ b . applyToSet (asFunc f) S = { b } by blast
      hence card (applyToSet (asFunc f) S) = 1 by auto
    }
    thus ?thesis by auto
  qed

  have case2: card S = 2 ⟶ ?thesis
  proof −
```

{ **assume** *card2*: *card S* = *2*
  **hence** ∃ *a u* . *a*≠*u* ∧ *S* = {*a*, *u*} **by** (*meson card-2-iff*)
  **then obtain** *a u* **where** *au*: *a*≠*u* ∧ *S* = {*a*, *u*} **by** *blast*
  **define** *b* **where** *b*: *b* = *f a*
  **define** *v* **where** *v*: *v* = *f u*
  **hence** *applyToSet* (*asFunc f*) *S* = { *b*, *v* }
  **proof** −
    **have** {*b*, *v*} ⊆ *applyToSet* (*asFunc f*) *S* **using** *au b v* **by** *auto*
    **moreover have** *applyToSet* (*asFunc f*) *S* ⊆ {*b*, *v*}
    **proof** −
      { **fix** *c* **assume** *c*: *c* ∈ *applyToSet* (*asFunc f*) *S*
        **hence** *c* ∈ { *c* . ∃ *a′* ∈ *S* . (*asFunc f*) *a′ c* } **by** *auto*
        **then obtain** *a′* **where** *a′*: *a′* ∈ *S* ∧ (*asFunc f*) *a′ c* **by** *blast*
        **hence** (*a′* = *a* ∧ *f a* = *c*) ∨ (*a′* = *u* ∧ *f u* = *c*) **using** *au*
**by** *auto*
        **hence** *c* ∈ {*b*, *v*} **using** *b v* **by** *auto*
      }
      **thus** *?thesis* **by** *blast*
    **qed**
    **ultimately show** *?thesis* **by** *blast*
  **qed**
  **moreover have** *b* ≠ *v*
  **proof** −
    { **assume** *b* = *v*
      **hence** *f a* = *f u* **using** *b v* **by** *simp*
      **hence** *a* = *u* **using** *assms*(*1*) **by** *blast*
      **hence** *False* **using** *au* **by** *auto*
    }
    **thus** *?thesis* **by** *auto*
  **qed**
  **ultimately have** ∃ *b v* . *b*≠*v* ∧ *applyToSet* (*asFunc f*) *S* = { *b*,
*v* } **by** *blast*

  **hence** *card* (*applyToSet* (*asFunc f*) *S*) = *2* **using** *card-2-iff* **by**
*auto*
}
**thus** *?thesis* **by** *auto*
**qed**

**thus** *?thesis* **using** *cases case1* **by** *blast*
**qed**

**lemma** *lemCardOfLineIsBig*:
**assumes** *x* ≠ *p*
**and** *onLine x l* ∧ *onLine p l*
**shows** ∃ *p1 p2 p3* . (*onLine p1 l* ∧ *onLine p2 l* ∧ *onLine p3 l*)
∧ (*p1*≠*p2* ∧ *p2*≠*p3* ∧ *p3*≠*p1*)
**proof** −

**obtain** *b d* **where** *bd*: $l = line\ b\ d$ **using** *assms*(*2*) **by** *blast*
**hence** *dnot0*: $d \neq origin$ **using** *assms* **by** *auto*
**have** *lpd*: $l = line\ p\ d$ **using** *lemSameLine*[*of p b d*] *bd assms*(*2*) **by** *auto*

**define** *p1* **where** *p1*: $p1 = (p \oplus (1 \otimes d))$
**define** *p2* **where** *p2*: $p2 = (p \oplus (2 \otimes d))$
**define** *p3* **where** *p3*: $p3 = (p \oplus (3 \otimes d))$

**have** *onl*: $onLine\ p1\ l \wedge onLine\ p2\ l \wedge onLine\ p3\ l$ **using** *lpd p1 p2 p3* **by** *auto*

**have** *psdiff*: $p1 \neq p2 \wedge p2 \neq p3 \wedge p3 \neq p1$
**proof** −
  **have** $p1 \neq p2$ **using** *p1 p2 dnot0* **by** *auto*
  **moreover have** $p2 \neq p3$ **using** *p2 p3 dnot0* **by** *auto*
  **moreover have** $p3 \neq p1$ **using** *p3 p1 dnot0* **by** *auto*
  **ultimately show** *?thesis* **by** *blast*
**qed**

**hence** $(onLine\ p1\ l \wedge onLine\ p2\ l \wedge onLine\ p3\ l) \wedge (p1{\neq}p2 \wedge p2{\neq}p3 \wedge p3{\neq}p1)$
  **using** *onl* **by** *blast*
**thus** *?thesis* **using** *p1 p2 p3* **by** *meson*
**qed**

**end**
**end**

# 34 AffineConeLemma

This theory shows that affine approximations preserve "insideness" of points relative to cones.

**theory** *AffineConeLemma*
  **imports** *KeyLemma TangentLineLemma Cardinalities*
**begin**

**class** *AffineConeLemma* = *KeyLemma* + *TangentLineLemma* + *Cardinalities*
**begin**

**lemma** *lemInverseOfAffInvertibleIsAffInvertible*:
  **assumes** *affInvertible A*
**and** $\forall\ x\ y\ .\ A\ x = y \longleftrightarrow A'\ y = x$
**shows** *affInvertible* $A'$
**proof** −

**have** *invA′*: *invertible A′* **using** *assms*(*2*) **by** *force*
**moreover have** *affine A′*
**proof** −
**obtain** $L$ $T$ **where** *LT*: (*linear* $L$) $\wedge$ (*translation* $T$) $\wedge$ ($A = T \circ L$)
**using** *assms*(*1*) **by** *blast*
**then obtain** $t$ **where** $t$: $\forall\ x\ .\ T\ x = (x \oplus t)$ **using** *LT* **by** *auto*

**have** *invertible L*
**proof** −
**{ fix** $q$
**define** $p$ **where** $p$: $p = A'\ (T\ q)$
**hence** *Lpq*: ($L\ p = q$)
**proof** −
**have** $A\ p = T\ q$ **using** $p$ *assms*(*2*) **by** *simp*
**thus** *?thesis* **using** *LT* **by** *auto*
**qed**
**moreover have** ($\forall x.\ L\ x = q \longrightarrow x = p$)
**proof** −
**{ fix** $x$ **assume** $L\ x = q$
**hence** $L\ x = L\ p$ **using** *Lpq* **by** *simp*
**hence** $A\ x = A\ p$ **using** *LT* **by** *auto*
**hence** $x = p$ **using** *assms*(*2*) **by** *force*
**}**
**thus** *?thesis* **by** *auto*
**qed**
**ultimately have** $\exists\ p\ .\ (L\ p = q) \wedge (\forall x.\ L\ x = q \longrightarrow x = p)$ **by** *blast*
**}**
**thus** *?thesis* **by** *blast*
**qed**
**then obtain** $L'$ **where** $L'$: $\forall\ x\ y\ .\ L\ x = y \longleftrightarrow L'\ y = x$ **by** *metis*

**have** *linL*: *linear L* **using** *LT* **by** *auto*
**have** *linL′*: *linear L′*
**proof** −
**have** *part1*: $L'$ *origin* = *origin* **using** *linL* $L'$ **by** *auto*
**have** *part2*: $\forall\ a\ p\ .\ L'\ (a \otimes p) = (a \otimes (L'\ p))$
**proof** −
**{ fix** $a$ $p$
**have** $L\ (L'\ p) = p$ **using** $L'$ **by** *auto*
**hence** $L\ (a \otimes\ (L'\ p)) = (a \otimes p)$
**using** *linL lemLinearProps*[*of L a* ($L'\ p$)] **by** *auto*
**hence** $(a \otimes\ (L'\ p)) = (L'\ (a \otimes p))$ **using** $L'$ **by** *auto*
**}**
**thus** *?thesis* **by** *auto*
**qed**
**have** $\forall\ p\ q\ .\ (L'\ (p \oplus q) = ((L'\ p) \oplus (L'\ q))) \wedge\ (L'\ (p \ominus q) = ((L'\ p) \ominus (L'\ q)))$

**proof** −
{ **fix** $p$ $q$
**have** $(L\ ((L'\ p) \oplus (L'\ q)) = ((L\ (L'\ p)) \oplus (L\ (L'\ q))))$
$\wedge\ (L\ ((L'\ p) \ominus (L'\ q)) = ((L\ (L'\ p)) \ominus (L\ (L'\ q))))$
**using** *linL lemLinearProps*[*of L 0* $(L'\ p)$ $(L'\ q)$] **by** *auto*
**moreover have** $L\ (L'\ p) = p \wedge L\ (L'\ q) = q$ **using** $L'$ **by** *auto*
**ultimately have** $(L\ ((L'\ p) \oplus (L'\ q)) = (p \oplus q)) \wedge\ (L\ ((L'\ p) \ominus (L'\ q)) = (p \ominus q))$
**using** $L'$ **by** *auto*
**hence** $((L'\ p) \oplus (L'\ q)) = L'\ (p \oplus q) \wedge ((L'\ p) \ominus (L'\ q)) = L'\ (p \ominus q)$
**using** $L'$ **by** *force*
}
**thus** *?thesis* **by** *force*
**qed**

**thus** *?thesis* **using** *part1 part2* **by** *blast*
**qed**

**define** $t'$ **where** $t'$: $t' = (origin \ominus (L'\ t))$
**define** $T'$ **where** $T'$: $T' = mkTranslation\ t'$

**have** *transT'*: *translation* $T'$ **using** $T'$ $t'$ **by** *fastforce*

**have** $A' = T'\ o\ L'$
**proof** −
{ **fix** $q$ **define** $p$ **where** $p$: $p = A'\ q$
**hence** $A\ p = q$ **using** *assms*(*2*) **by** *force*
**hence** $((L\ p) \oplus t) = q$ **using** *LT t* **by** *auto*
**hence** $L\ p = (q \ominus t)$ **using** *add-diff-eq* **by** *auto*
**hence** $p = L'\ (q \ominus t)$ **using** $L'$ **by** *auto*
**hence** $p = ((L'\ q) \ominus (L'\ t))$ **using** *lemLinearProps*[*of* $L'$] *linL'* **by** *auto*
**hence** $p = T'\ (L'\ q)$ **using** $T'$ $t'$ **by** *auto*
**hence** $A'\ q = (T'\ o\ L')\ q$ **using** $p$ **by** *auto*
}
**thus** *?thesis* **by** *blast*
**qed**

**thus** *?thesis* **using** *linL' transT'* **by** *blast*
**qed**

**ultimately show** *?thesis* **by** *blast*
**qed**

**lemma** *lemInsideRegularConeUnderAffInvertible*:
**assumes** *affInvertible A*
**and** *insideRegularCone x p*
**and** *regularConeSet* (*A x*) = *applyToSet* (*asFunc A*) (*regularConeSet x*)
**shows** *insideRegularCone* (*A x*) (*A p*)
**proof** −
**define** *y* **where** *y*: *y* = *A x*
**define** *q* **where** *q*: *q* = *A p*
**define** *cx* **where** *cx*: *cx* = *regularConeSet x*
**define** *cy* **where** *cy*: *cy* = *regularConeSet y*

**obtain** *A′* **where** *A′*: ∀ *x y* . *A x* = *y* ⟷ *A′ y* = *x* **using** *assms*(*1*) **by** *metis*
**hence** *invA′*: *invertible A′* **by** *force*
**have** *affA′*: *affine A′*
**using** *A′ assms*(*1*) *lemInverseOfAffInvertibleIsAffInvertible*
**by** *auto*

**have** *p′*: *A′ q* = *p* **using** *A′ q* **by** *auto*
**have** *x′*: *A′ y* = *x* **using** *A′ y* **by** *auto*

**have** *xnotp*: *x* ≠ *p* **using** *assms*(*2*) **by** *auto*
**have** *ynotq*: *y* ≠ *q* **using** *p′ x′ xnotp* **by** *auto*

**have** *cy′*: *cy* = *applyToSet* (*asFunc A*) *cx* **using** *y cx cy assms*(*3*) **by** *auto*
**have** *cx′*: *cx* = *applyToSet* (*asFunc A′*) *cy*
**proof** −
**{ fix** *z* **assume** *z* ∈ *cx*
**hence** (*A z*) ∈ *cy* **using** *cy′* **by** *auto*
**hence** *A′* (*A z*) ∈ *applyToSet* (*asFunc A′*) *cy* **by** *auto*
**hence** *z* ∈ *applyToSet* (*asFunc A′*) *cy* **using** *A′* **by** *metis*
**}**
**hence** *l2r*: *cx* ⊆ *applyToSet* (*asFunc A′*) *cy* **by** *blast*
**{ fix** *z* **assume** *rhs*: *z* ∈ *applyToSet* (*asFunc A′*) *cy*
**hence** *z* ∈ { *z* . ∃ *z′* . *z′* ∈ *cy* ∧ (*asFunc A′*) *z′ z* } **by** *auto*
**then obtain** *z1* **where** *z1*: *z1* ∈ *cy* ∧ (*asFunc A′*) *z1 z* **by** *blast*
**hence** *z1* ∈ { *z1* . ∃ *z2* . *z2* ∈ *cx* ∧ (*asFunc A*) *z2 z1* } **using** *cy′* **by** *auto*
**then obtain** *z2* **where** *z2*: *z2* ∈ *cx* ∧ (*asFunc A*) *z2 z1* **by** *blast*
**hence** *z* = *z2* **using** *z1 A′* **by** *auto*
**hence** *z* ∈ *cx* **using** *z2* **by** *auto*
**}**
**thus** *?thesis* **using** *l2r* **by** *blast*
**qed**

```
  have noton: ¬ onRegularCone y q
  proof –
    { assume on: onRegularCone y q

      define lx where lx: lx = lineJoining x p
      define ly where ly': ly = applyToSet (asFunc A) lx
      have onlx: onLine x lx ∧ onLine p lx
        using lemLineJoiningContainsEndPoints[of lx x p] lx by auto

      have linelx: isLine lx using lx by blast
      have linely: applyAffineToLine A lx ly
        using lemAffineOfLineIsLine[of lx A ly] assms(1) ly' linelx by
auto

      have ∃ D . lx = line p D
      proof –
        obtain b d where lx = line b d using linelx by blast
       hence lx = line p d using lemSameLine[of p b d] onlx by auto
        thus ?thesis by auto
      qed
      then obtain D where D: lx = line p D by auto

      have Dnot0: D ≠ origin
      proof –
        { assume D = origin
          hence False using D onlx xnotp by auto
        }
        thus ?thesis by auto
      qed

      have ly: ly = lineJoining y q
      proof –
         have applyToSet (asFunc A) {x,p} ⊆ applyToSet (asFunc A)
lx using onlx by auto
        hence {y,q} ⊆ ly using y q ly' by auto
        moreover have isLine ly using linely by auto
        ultimately show ?thesis using lemLineAndPoints[of y q ly]
          by (simp add: ynotq)
      qed

      hence only: { y, q } ⊆ ly
        using lemLineJoiningContainsEndPoints[of ly y q] ly' by auto

      have SxSy: applyToSet (asFunc A) (lx ∩ cx) = (ly ∩ cy)
       using lemInvertibleOnMeet[of A lx ∩ cx lx cx] assms(1) ly' cy'
        by auto


      have cardx: 0 < card (lx ∩ cx) ≤ 2
```

**using** *lemInsideRegularConeImplies*[*of x p D lx*]
*assms*(*2*) *Dnot0 lx D cx*
**by** *fastforce*

**hence** *cardy*: *card* (*ly* $\cap$ *cy*) = *card* (*lx* $\cap$ *cx*)
**using** *lemSmallCardUnderInvertible*[*of A lx* $\cap$ *cx*] *assms*(*1*) *SxSy* **by** *auto*

**hence** *lycy*: *ly* $\cap$ *cy* = *ly*
**using** *lemOnRegularConeIff*[*of ly y q*] *ly ynotq cy on*
**by** *blast*
**hence** $\exists$ *p1 p2 p3* . (*p1* $\in$ *ly* $\wedge$ *p2* $\in$ *ly* $\wedge$ *p3* $\in$ *ly*)
$\wedge$ (*p1*$\neq$*p2* $\wedge$ *p2*$\neq$*p3* $\wedge$ *p3*$\neq$*p1*)
**using** *lemCardOfLineIsBig*[*of y q ly*] *ynotq only linely* **by** *auto*
**then obtain** *p1 p2 p3*
**where** *ps*: (*p1* $\in$ *ly* $\wedge$ *p2* $\in$ *ly* $\wedge$ *p3* $\in$ *ly*) $\wedge$ (*p1*$\neq$*p2* $\wedge$ *p2*$\neq$*p3* $\wedge$ *p3*$\neq$*p1*)
**by** *auto*

**have** *not1*: *card ly* $\neq$ *1* **using** *ps card-1-singleton-iff*[*of ly*] **by** *auto*
**have** *not2*: *card ly* $\neq$ *2* **using** *ps card-2-iff*[*of ly*] **by** *auto*
**hence** $\neg$ (*0* < *card* (*ly* $\cap$ *cy*) $\leq$ *2*) **using** *lycy not1* **by** *auto*
**hence** *False* **using** *cardy cardx* **by** *auto*
**}**
**thus** *?thesis* **by** *blast*
**qed**

**have** *notout*: $\neg$ *outsideRegularCone y q*
**proof** –
**{ assume** *out*: *outsideRegularCone y q*
**hence** ($\exists$ *l q*$'$ . (*q*$'$ $\neq$ *q*) $\wedge$ *onLine q*$'$ *l* $\wedge$ *onLine q l*
$\wedge$ (*l* $\cap$ *cy* = {}))
**using** *lemOutsideRegularConeImplies*[*of y q*] *cy*
**by** *auto*
**then obtain** *l q*$'$
**where** *l*: (*q*$'$ $\neq$ *q*) $\wedge$ *onLine q*$'$ *l* $\wedge$ *onLine q l* $\wedge$ (*l* $\cap$ *cy* = {}) **by** *blast*

**define** *lx* **where** *lx*: *lx* = *applyToSet* (*asFunc A*$'$) *l*
**have** (*lx* $\cap$ *cx*) = *applyToSet* (*asFunc A*$'$) (*l* $\cap$ *cy*)
**using** *lemInvertibleOnMeet*[*of A*$'$ *l* $\cap$ *cy l cy*]
*invA*$'$ *lx cx*$'$ **by** *auto*
**hence** (*lx* $\cap$ *cx*) = *applyToSet* (*asFunc A*$'$){} **using** *l* **by** *auto*
**hence** *int0*: (*lx* $\cap$ *cx*) = {} **by** *simp*

**hence** *card0*: *card* (*lx* $\cap$ *cx*) = *0* **by** *simp*

**have** *linelx*: *isLine lx*
**proof** –
  **have** *isLine l* **using** *l* **by** *blast*
  **thus** *?thesis* **using** *lemAffineOfLineIsLine*[*of l A′ lx*] *lx affA′*
    **by** *auto*
**qed**

**have** *ponlx*: *onLine p lx*
**proof** –
  **have** $q \in l$ **using** *l* **by** *simp*
  **thus** *?thesis* **using** *lx p′ linelx* **by** *auto*
**qed**

**have** $\exists\ D\ .\ lx = line\ p\ D$
**proof** –
  **obtain** *b d* **where** $lx = line\ b\ d$ **using** *linelx* **by** *blast*
  **hence** $lx = line\ p\ d$ **using** *lemSameLine*[*of p b d*] *ponlx* **by** *auto*
  **thus** *?thesis* **by** *auto*
**qed**
**then obtain** *D* **where** *D*: $lx = line\ p\ D$ **by** *auto*

**have** *Dnot0*: $D \neq origin$
**proof** –
  **{ assume** *D0*: $D = origin$
    **have** *allp*: $\forall\ pt.\ onLine\ pt\ lx \longrightarrow pt = p$
    **proof** –
      **{ fix** *pt* **assume** *onLine pt lx*
        **then obtain** *a* **where** $pt = (p \oplus (a \otimes D))$ **using** *D* **by**
*auto*
        **hence** $pt = p$ **using** *D0* **by** *simp*
      **}**
      **thus** *?thesis* **by** *blast*
    **qed**

    **define** *p1* **where** *p1*: $p1 = A'\ q'$
    **have** *AA′*: $\forall\ pt\ .\ A\ (A'\ pt) = pt$ **by** (*simp add*: *A′*)

    **hence** $p1 \neq p$
    **proof** –
      **{ assume** *pp*: $p1 = p$
        **hence** $A\ (A'\ q') = A\ (A'\ q)$ **using** *p′ p1* **by** *auto*
        **hence** $q' = q$ **using** *AA′* **by** *simp*
        **hence** *False* **using** *l* **by** *auto*
      **}**
      **thus** *?thesis* **by** *auto*
    **qed**

```
      moreover have onLine p1 lx
      proof −
        have p1 = A′ q′ using l p1 by blast
        hence p1 ∈ applyToSet (asFunc A′) l using l by auto
        hence p1 ∈ lx by (simp add: lx)
        thus ?thesis using linelx by auto
      qed

      ultimately have False using l allp by blast
    }
    thus ?thesis by auto
  qed

  have 0 < card (lx ∩ cx) ≤ 2
    using lemInsideRegularConeImplies[of x p D lx]
          assms(2) Dnot0 D cx
    by blast

  hence False using card0 by simp
 }
 thus ?thesis by blast
qed

hence ¬ (vertex y q) ∧ ¬(onRegularCone y q) ∧ ¬(outsideRegularCone
y q)
  using ynotq noton notout by blast
 hence insideRegularCone y q using lemInsideCone[of y q]
  by fastforce

 thus ?thesis using y q by blast
qed



end
end
```

# 35 NoFTLGR

This theory completes the proof of NoFTLGR.

```
theory NoFTLGR
  imports ObserverConeLemma AffineConeLemma
begin

class NoFTLGR = ObserverConeLemma + AffineConeLemma
begin
```

The theorem says: if observer m encounters observer k (so that they are both present at the same spacetime point x), then k is moving at sub-light speed relative to m. In other words, no observer ever encounters another observer who appears to be moving at or above lightspeed.

**theorem** *lemNoFTLGR*:
  **assumes** *ass1*: $x \in$ *wline m m* $\cap$ *wline m k*
**and** *ass2*: *tl l m k x*
**and** *ass3*: $v \in$ *lineVelocity l*
**and** *ass4*: $\exists\ p\ .\ (p \neq x) \land (p \in l)$
**shows** (*lineSlopeFinite l*) $\land$ (*sNorm2 v* $<$ *1*)
**proof** –

  **define** *s* **where** *s*: *s* = (*wline k k*)

  **have** *axEventMinus m k x* **using** *AxEventMinus* **by** *force*
  **hence** ($\exists$ *q* . $\forall$ *b* . ( (*m sees b at x*) $\longleftrightarrow$ (*k sees b at q*)))
    **using** *ass1* **by** *blast*
  **then obtain** *y* **where** *y*: $\forall$ *b* . ( (*m sees b at x*) $\longleftrightarrow$ (*k sees b at y*)) **by** *auto*
  **hence** *mkxy*: *wvtFunc m k x y* **using** *ass1* **by** *auto*

  **have** *axDiff m k x* **using** *AxDiff* **by** *simp*
  **hence** $\exists$ *A* . (*affineApprox A* (*wvtFunc m k*) *x* ) **using** *mkxy* **by** *fast*
  **then obtain** *A* **where** *A*: *affineApprox A* (*wvtFunc m k*) *x* **by** *auto*

  **hence** *affA*: *affine A* **by** *auto*
  **have** *lineL*: *isLine l* **using** *ass2* **by** *auto*

  **define** $l'$ **where** $l'$: $l'$ = *applyToSet* (*asFunc A*) *l*

  **hence** *lineL*$'$: *isLine* $l'$
    **using** *lineL affA lemAffineOfLineIsLine*[*of l A* $l'$]
    **by** *auto*

  **have** *tgtl*$'$: *tangentLine* $l'$ *s y*
  **proof** –
    **define** *g1* **where** *g1*: *g1* $\equiv$ $x \in$ *wline m k*
    **define** *g2* **where** *g2*: *g2* $\equiv$ *tangentLine l* (*wline m k*) *x*
    **define** *g3* **where** *g3*: *g3* $\equiv$ *affineApprox A* (*wvtFunc m k*) *x*
    **define** *g4* **where** *g4*: *g4* $\equiv$ *wvtFunc m k x y*
    **define** *g5* **where** *g5*: *g5* $\equiv$ *applyAffineToLine A l* $l'$
    **define** *g6* **where** *g6*: *g6* $\equiv$ *tangentLine* $l'$ (*wline k k*) *y*

    **have** $x \in$ *wline m k*
        $\longrightarrow$ *tangentLine l* (*wline m k*) *x*
        $\longrightarrow$ *affineApprox A* (*wvtFunc m k*) *x*
        $\longrightarrow$ *wvtFunc m k x y*
        $\longrightarrow$ *applyAffineToLine A l* $l'$

```
        ⟶ tangentLine l′ (wline k k) y
    using lemPresentation[of x m k l k A y l′]
    by blast

    hence pres: g1 ⟶ g2 ⟶ g3 ⟶ g4 ⟶ g5 ⟶ g6
      using g1 g2 g3 g4 g5 g6 by fastforce

    have 1: g1 using ass1 g1 by auto
    have 2: g2 using ass2 g2 by fast
    have 3: g3 using A g3 by fast
    have 4: g4 using mkxy g4 by fast
    have 5: g5 using 1 lineL l′ affA lemAffineOfLineIsLine[of l A l′]
g5
      by auto

    hence g6   using 1 2 3 4 5 pres by meson
    thus ?thesis using s g6 by auto
  qed

  have ykk: y ∈ wline k k using ass1 y by auto


  have c2: l′ = timeAxis
  proof −
    have tl l′ k k y using tgtl′ l′ s by auto
    thus ?thesis
      using lemSelfTangentIsTimeAxis[of y k l′] by auto
  qed


  have yOnAxis: onLine y timeAxis
    using lemTimeAxisIsLine ykk AxSelfMinus by blast


  hence yOnl′: onLine y l′ using c2 by auto


  have ∀ p . cone k y p ⟷ regularCone y p
    using ykk lemProposition1[of y k] by auto
  hence ycone: coneSet k y = regularConeSet y by auto


  have xcone: coneSet m x = regularConeSet x
  proof −
    have x ∈ wline m m using ass1 by auto
    hence ∀ p . cone m x p ⟷ regularCone x p
      using lemProposition1[of x m] by auto
    thus ?thesis by auto
  qed
```

**have** *assm1'*: $y \in$ *wline k k* $\cap$ *wline k m*
**using** *ass1 y* **by** *auto*
**have** *axEventMinus k m y* **using** *AxEventMinus* **by** *force*
**hence** $(\exists\ q\ .\ \forall\ b\ .\ (\ (k\ sees\ b\ at\ y) \longleftrightarrow (m\ sees\ b\ at\ q)))$
**using** *assm1'* **by** *blast*
**then obtain** *z* **where** *z*: $\forall\ b\ .\ (\ (k\ sees\ b\ at\ y) \longleftrightarrow (m\ sees\ b\ at\ z))$ **by** *auto*
**hence** *kmyz*: *wvtFunc k m y z* **using** *assm1'* **by** *auto*

**have** *axDiff k m y* **using** *AxDiff* **by** *simp*
**hence** $\exists\ A$ . (*affineApprox A* (*wvtFunc k m*) *y* ) **using** *kmyz* **by** *fast*
**then obtain** *Akmy* **where** *Akmy*: *affineApprox Akmy* (*wvtFunc k m*) *y* **by** *auto*

**hence** *affA'*: *affine Akmy* **by** *auto*
**have** *invA'*: *invertible Akmy* **using** *Akmy* **by** *auto*

**then obtain** *Amkx* **where**
*Amkx*: (*affine Amkx*) $\wedge$ ($\forall\ p\ q$ . *Akmy* $p = q \longleftrightarrow$ *Amkx* $q = p$)
**using** *lemInverseAffine*[*of Akmy*] *affA'* **by** *blast*

**have** *wvtFunc m k x y* **using** *mkxy* **by** *auto*
**hence** *kmyx*: *wvtFunc k m y x* **by** *auto*
**hence** *xisz*: $x = z$ **using** *kmyz lemWVTImpliesFunction* **by** *blast*

**moreover have** $z =$ *Akmy y*
**using** *lemAffineEqualAtBase*[*of wvtFunc k m Akmy y*] *Akmy kmyz*
**by** *blast*
**ultimately have** *xA'y*: $x =$ *Akmy y* **by** *auto*

**hence** *p35a*: *applyToSet* (*asFunc Akmy*) (*coneSet k y*) $\subseteq$ *coneSet m x*
**using** *Akmy lemProposition2*[*of k m Akmy y*]
**by** *simp*

**have** *p35aRegular*: *applyToSet* (*asFunc Akmy*) (*regularConeSet y*) = *regularConeSet x*
**proof** –
**have** *applyToSet* (*asFunc Akmy*) (*regularConeSet y*) $\subseteq$ *coneSet m x*
**using** *ycone p35a* **by** *auto*
**hence** *l2r*: *applyToSet* (*asFunc Akmy*) (*regularConeSet y*) $\subseteq$ *regularConeSet x*

**using** *xcone* **by** *auto*

**have** *r2l*: *regularConeSet x* $\subseteq$ *applyToSet* (*asFunc Akmy*) (*regularConeSet y*)
**proof** −
{ **assume** *converse*: $\neg$(*regularConeSet x* $\subseteq$ *applyToSet* (*asFunc Akmy*) (*regularConeSet y*))
**then obtain** *z* **where**
*z*: *z* $\in$ *regularConeSet x* $\wedge$ $\neg$(*z* $\in$ *applyToSet* (*asFunc Akmy*) (*regularConeSet y*))
**by** *blast*
**define** *z′* **where** *z′*: *z′* = *Amkx z*

**have** *z′NotOnCone*:$\neg$(*z′* $\in$ *regularConeSet y*)
**proof** −
{ **assume** *conv*: *z′* $\in$ *regularConeSet y*
**have** *Akmy z′* = *z* **using** *Amkx z′* **by** *auto*
**hence** (*asFunc Akmy*) *z′ z* **by** *auto*
**hence** $\exists$ *z′* $\in$ *regularConeSet y* . (*asFunc Akmy*) *z′ z* **using** *conv* **by** *blast*
**hence** *z* $\in$ *applyToSet* (*asFunc Akmy*) (*regularConeSet y*) **by** *auto*
**hence** *False* **using** *z* **by** *blast*
}
**thus** *?thesis* **by** *blast*
**qed**

**hence** $\neg$ (*regularCone y z′*) **by** *auto*
**then obtain** *l* **where**
*l*: (*onLine z′ l*) $\wedge$ ($\neg$ (*y* $\in$ *l*)) $\wedge$ (*card* (*l* $\cap$ (*regularConeSet y*)) = *2*)
**using** *lemConeLemma2*[*of z′ y*] **by** *blast*
**then obtain** *p q* **where**
*pq*: (*p* $\neq$ *q*) $\wedge$ *p* $\in$ (*l* $\cap$ (*regularConeSet y*)) $\wedge$ *q* $\in$ (*l* $\cap$ (*regularConeSet y*))
**using** *lemElementsOfSet2*[*of l* $\cap$ (*regularConeSet y*)] **by** *blast*

**have** *lineL*: *isLine l* **using** *l* **by** *auto*

**define** *p′* **where** *p′*: *p′* = *Akmy p*
**define** *q′* **where** *q′*: *q′* = *Akmy q*
**have** *p′inv*: *Amkx p′* = *p* **using** *Amkx p′* **by** *auto*
**have** *q′inv*: *Amkx q′* = *q* **using** *Amkx q′* **by** *auto*

**have** *pOnCone*: *p* $\in$ *regularConeSet y* **using** *pq* **by** *blast*
**moreover have** (*asFunc Akmy*) *p p′* **using** *p′* **by** *auto*
**ultimately have** $\exists$ *p* $\in$ *regularConeSet y* . (*asFunc Akmy*) *p p′* **by** *blast*
**hence** *p′* $\in$ *applyToSet* (*asFunc Akmy*) (*regularConeSet y*) **by**

*auto*
**hence** *Ap*: $p' \in$ *regularConeSet x* **using** *l2r* **by** *blast*

**have** *qOnCone*: $q \in$ *regularConeSet y* **using** *pq* **by** *blast*
**moreover have** (*asFunc Akmy*) *q* $q'$ **using** $q'$ **by** *auto*
**ultimately have** $\exists$ $q \in$ *regularConeSet y* . (*asFunc Akmy*) *q* $q'$ **by** *blast*
**hence** $q' \in$ *applyToSet* (*asFunc Akmy*) (*regularConeSet y*) **by** *auto*
**hence** *Aq*: $q' \in$ *regularConeSet x* **using** *l2r* **by** *blast*

**have** $p'q'$: $p' \neq q'$
**proof** –
{ **assume** $p' = q'$
**hence** *Akmy* $p'$ = *Akmy* $q'$ **by** *auto*
**hence** $p = q$ **by** (*metis* $p'$ $q'$ *Amkx*)
**hence** *False* **using** *pq* **by** *simp*
}
**thus** *?thesis* **by** *auto*
**qed**

**have** $p'z$: $p' \neq z$
**proof** –
{ **assume** $p' = z$
**hence** $p = z'$ **using** $p'$*inv* $z'$ **by** *auto*
**hence** *False* **using** *pOnCone* $z'$*NotOnCone* **by** *auto*
}
**thus** *?thesis* **by** *auto*
**qed**

**have** $q'z$: $q' \neq z$
**proof** –
{ **assume** $q' = z$
**hence** $q = z'$ **using** $q'$*inv* $z'$ **by** *auto*
**hence** *False* **using** *qOnCone* $z'$*NotOnCone* **by** *auto*
}
**thus** *?thesis* **by** *auto*
**qed**

**define** $l'$ **where** $l'$: $l'$ = *applyToSet* (*asFunc Akmy*) *l*
**have** *affine Akmy* **using** *Akmy* **by** *auto*
**hence** *All*$'$: *applyAffineToLine Akmy l* $l'$
**using** $l'$ *lineL lemAffineOfLineIsLine*[*of l Akmy* $l'$]
**by** *blast*

**have** *lineL*$'$: *isLine* $l'$ **using** *All*$'$ **by** *auto*

**define** *S* **where** *S*: $S = l' \cap$ *regularConeSet x*

**have** *xNotInL'*: $\neg$ ($x \in l'$)
**proof** –
 **{ assume** $x \in l'$
  **hence** $\exists$ *y1* $\in$ *l* . (*asFunc Akmy*) *y1 x* **using** *l'* **by** *auto*
  **then obtain** *y1* **where** *y1*: (*y1* $\in$ *l*) $\wedge$ (*Akmy y1* = *x*) **by**
*auto*
  **hence** *Akmy y1* = *Akmy y* **using** *xA'y* **by** *auto*
  **hence** *y1* = *y* **using** *invA'* **by** *auto*
  **hence** $y \in l$ **using** *y1* **by** *auto*
  **hence** *False* **using** *l* **by** *auto*
 **}**
 **thus** *?thesis* **by** *auto*
**qed**

**have** *p'InMeet*: $p' \in S$
**proof** –
 **have** $p \in l \wedge$ (*asFunc Akmy*) *p p'* **using** *p' pq* **by** *auto*
 **hence** $\exists$ $p \in l$ . (*asFunc Akmy*) *p p'* **by** *auto*
 **hence** $p' \in l'$ **using** *l'* **by** *auto*
 **thus** *?thesis* **using** *Ap S* **by** *blast*
**qed**

**have** *q'InMeet*: $q' \in S$
**proof** –
 **have** $q \in l \wedge$ (*asFunc Akmy*) *q q'* **using** *q' pq* **by** *auto*
 **hence** $\exists$ $q \in l$ . (*asFunc Akmy*) *q q'* **by** *auto*
 **hence** $q' \in l'$ **using** *l'* **by** *auto*
 **thus** *?thesis* **using** *Aq S* **by** *blast*
**qed**

**have** *zInMeet*: $z \in S$
**proof** –
 **have** *Akmy z'* = *z* **using** *z' Amkx* **by** *blast*
 **moreover have** $z' \in l$ **using** *l* **by** *auto*
 **ultimately have** $z' \in l \wedge$ (*asFunc Akmy*) *z' z* **by** *auto*
 **hence** $\exists$ $z' \in l$ . (*asFunc Akmy*) *z' z* **by** *auto*
 **hence** $z \in l'$ **using** *l'* **by** *auto*
 **thus** *?thesis* **using** *z S* **by** *blast*
**qed**

**have** *finite S* $\wedge$ *card S* $\leq$ *2*
 **using** *xNotInL' lineL' S lemConeLemma1*[*of x l' S*]
 **by** *auto*

**moreover have** $S \neq \{\}$ **using** *zInMeet* **by** *auto*

**ultimately have** *card S* = *1* $\vee$ *card S* = *2*
 **using** *card-0-eq* **by** *fastforce*

```
      moreover have card S ≠ 2
      proof –
        { assume card S = 2
          hence p′ = z ∨ q′ = z
            using p′q′ p′InMeet q′InMeet zInMeet
                  lemThirdElementOfSet2[of p′ q′ S z]
            by auto
          hence False using p′z q′z by auto
        }
        thus ?thesis by auto
      qed

      moreover have card S ≠ 1
        using p′InMeet q′InMeet p′q′ card-1-singletonE by force


      ultimately have False by blast
    }
    thus ?thesis by blast
  qed
  thus ?thesis using l2r by blast
qed



have lprops: l = applyToSet (asFunc Akmy) timeAxis
proof –
  define t′ where t′: t′ = applyToSet (asFunc Akmy) timeAxis

  define p1 where p1: p1 = (y ∈ wline k k)
  define p2 where p2: p2 = tangentLine timeAxis (wline k k) y
  define p3 where p3: p3 = affineApprox Akmy (wvtFunc k m) y
  define p4 where p4: p4 = wvtFunc k m y x
  define p5 where p5: p5 = applyAffineToLine Akmy timeAxis t′

  define tgt where tgt: tgt = tangentLine t′ (wline m k) x

  have pre1: p1 using p1 ykk by auto
  have pre2: p2
  proof –
    have tangentLine l′ (wline k k) y using tgtl′ s by auto
    hence tangentLine timeAxis (wline k k) y using c2 by meson
    thus ?thesis using p2 by blast
  qed
  have pre3: p3 using p3 Akmy by auto
  have pre4: p4 using p4 kmyx by auto
  have pre5: p5
```

**using** *p5 affA′ lemTimeAxisIsLine t′ Akmy*
*lemAffineOfLineIsLine*[*of timeAxis Akmy t′*]
**by** *blast*

**have** $p1 \longrightarrow p2 \longrightarrow p3 \longrightarrow p4 \longrightarrow p5 \longrightarrow tgt$
**using** *p1 p2 p3 p4 p5 tgt*
*lemPresentation*[*of y k k timeAxis m Akmy x t′*]
**by** *fast*
**hence** *tl t′ m k x* **using** *tgt pre1 pre2 pre3 pre4 pre5* **by** *auto*
**moreover have** *tl l m k x* **using** *ass2* **by** *auto*
**moreover have** *affineApprox A* (*wvtFunc m k*) *x* **using** *A* **by** *auto*
**moreover have** *wvtFunc m k x y* **using** *mkxy* **by** *auto*
**moreover have** $x \in$ *wline m k* **using** *ass1* **by** *auto*
**ultimately have** $t' = l$
**using** *lemTangentLineUnique*[*of x m k t′ l A y*]
**by** *fast*
**thus** *?thesis* **using** *t′* **by** *blast*
**qed**

{ **fix** *py* **assume** *py*: *onTimeAxis py* $\wedge$ $py \neq y$

**have** *pyInsideCone*: *insideRegularCone y py*
**proof** −
**have** *pyOnAxis*: *onLine py timeAxis* **using** *py lemTimeAxisIsLine* **by** *blast*

**hence** *pyprops*: *timeAxis* = *lineJoining y py*
**using** *py yOnAxis lemLineAndPoints*[*of y py timeAxis*] **by** *auto*

**define** *d* **where** *d*: $d = (y \ominus py)$
**hence** $\exists$ *py y* . ($py \neq y$) $\wedge$ (*onLine py timeAxis*)
$\wedge$ (*onLine y timeAxis*) $\wedge$ ($d = (y \ominus py)$)
**using** *py pyOnAxis yOnAxis* **by** *blast*
**hence** *ddrtn*: $d \in$ *drtn timeAxis* **by** *simp*

**have** *scomp0*: *sComponent d* = *sOrigin* **using** *d py yOnAxis* **by** *auto*

**have** *sf*: *slopeFinite py y* **using** *py yOnAxis* **by** *force*
**hence** *sloper py y* = $((-1) \otimes ((1\ /\ (tval\ py\ -\ tval\ y)) \otimes d))$
**using** *d* **by** *auto*
**hence** *velocityJoining py y* = *sOrigin* **using** *scomp0* **by** *simp*
**hence** *velocityJoining origin d* = *sOrigin* **using** *d* **by** *auto*

**hence** ($d \in$ *drtn timeAxis*) $\wedge$ (*sOrigin* = *velocityJoining origin d*)
**using** *ddrtn* **by** *auto*

**hence** ∃ *d* . (*d* ∈ *drtn timeAxis*) ∧ (*sOrigin* = *velocityJoining origin d*)
**by** *blast*
**hence** (*sOrigin* ∈ *lineVelocity timeAxis*) **by** *auto*

**hence** (*sOrigin* ∈ *lineVelocity timeAxis*) ∧ (*sNorm2 sOrigin* < *1*)
**by** *auto*
**hence** ∃ *v* . (*v* ∈ *lineVelocity timeAxis*) ∧ (*sNorm2 v* < *1*)
**by** *blast*
**thus** *?thesis* **using** *pyprops sf* **by** *auto*
**qed**

**define** *px* **where** *px*: *px* = *Akmy py*

**have** *insideRegularCone x px*
**proof** −
**have** *insideRegularCone y py* **using** *pyInsideCone* **by** *blast*
**moreover have** *affInvertible Akmy* **using** *affA′ invA′* **by** *blast*
**moreover have** *x* = *Akmy y* **by** (*simp add*: *xA′y*)
**moreover have** *px* = *Akmy py* **by** (*simp add*: *px*)
**moreover have** *regularConeSet x* = *applyToSet* (*asFunc Akmy*) (*regularConeSet y*)
**using** *p35aRegular* **by** *simp*
**ultimately show** *?thesis*
**using** *lemInsideRegularConeUnderAffInvertible*[*of Akmy y py*]
**by** *auto*
**qed**

**moreover have** *x* ≠ *px*
**proof** −
**{ assume** *xispx*:*x* = *px*
**hence** *False* **using** *xispx invA′ px xA′y py* **by** *auto*
**}**
**thus** *?thesis* **by** *auto*
**qed**

**ultimately have** *insideRegularCone x* (*Akmy py*) ∧ *x* ≠ (*Akmy py*)
**using** *px* **by** *blast*
**}**
**hence** *result*: ∀ *py* . (*onTimeAxis py* ∧ *py* ≠ *y*)
⟶ *insideRegularCone x* (*Akmy py*) ∧ *x* ≠ (*Akmy py*)
**by** *blast*

**{**

**obtain** *p* **where** *p*: $(p \neq x) \wedge (p \in l)$ **using** *assms(4)* **by** *blast*

**have** $l$ = *applyToSet* (*asFunc Akmy*) *timeAxis* **using** *lprops* **by** *simp*
**hence** $p \in \{ p . \exists\ py \in timeAxis . (asFunc\ Akmy)\ py\ p \}$ **using** *p* **by** *auto*
**then obtain** *py* **where** *py*: $py \in timeAxis \wedge (asFunc\ Akmy)\ py\ p$ **by** *blast*

**hence** *onTimeAxis py* **by** *blast*
**moreover have** $py \neq y$
**proof** −
{ **assume** $py = y$
**hence** *False* **using** *py p* **by** (*simp add*: *xA′y*)
}
**thus** *?thesis* **by** *auto*
**qed**
**ultimately have** *onTimeAxis py* $\wedge$ $py \neq y$ **by** *blast*

**hence** *inside*: *insideRegularCone x p* $\wedge$ $x \neq p$ **using** *result py* **by** *auto*
**have** *onl*: *onLine x l* $\wedge$ *onLine p l* **using** *ass2* **using** *p* **by** *blast*
**have** *pnotx*: $p \neq x$ **using** *inside* **by** *auto*
**hence** *xnotp*: $x \neq p$ **by** *simp*

**hence** *lj*: $l$ = *lineJoining x p*
**using** *lemLineAndPoints*[*of x p l*] *xnotp onl* **by** *auto*

**hence** *lineSlopeFinite l* **using** *onl inside* **by** *blast*

**moreover have** (*sNorm2 v* < *1*)
**proof** −
**have** ($\exists$ *v* $\in$ *lineVelocity l* . *sNorm2 v* < *1*) **using** *lj inside* **by** *auto*
**then obtain** *u* **where** *u*: *u* $\in$ *lineVelocity l* $\wedge$ *sNorm2 u* < *1* **by** *blast*
**hence** $u = v$
**using** *lemFiniteLineVelocityUnique*[*of u l v*] *ass3 calculation*
**by** *presburger*
**thus** *?thesis* **using** *u* **by** *auto*
**qed**

**ultimately have** (*lineSlopeFinite l*) $\wedge$ (*sNorm2 v* < *1*) **by** *auto*
**}**
**thus** *?thesis* **by** *auto*
**qed**

**end**

**end**

# References


[1] M. Stannett and I. Németi. Using Isabelle/HOL to verify first-order relativity theory. *Journal of Automated Reasoning*, 52(4):361–378, 2014.

[2] M. Stannett and I. Németi. No faster-than-light observers. *Archive of Formal Proofs*, April 2016. https://isa-afp.org/entries/No_FTL_observers.html, Formal proof development.