No Faster-Than-Light Observers

Mike Stannett

October 11, 2017

Abstract
We provide a formal proof within First Order Relativity Theory that no observer can travel faster than the speed of light. Originally reported by Stannett and Németi [1].

Contents

theory SpaceTime
imports Main
begin

record 'a Vector =
tdir :: 'a
xdir :: 'a
ydir :: 'a
zdir :: 'a

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

record 'a Line =
basepoint :: 'a Point
direction :: 'a Vector
record 'a Plane =
  phbasepoint :: 'a Point
  direction1 :: 'a Vector
  direction2 :: 'a Vector

record 'a Cone =
  vertex :: 'a Point
  slope :: 'a

class Quantities = linordered-field

class Vectors = Quantities
begin

abbreviation vecZero :: 'a Vector (0) where
  vecZero ≡ ( | tdir = (0::'a), xdir = 0, ydir = 0, zdir = 0 |

fun vecPlus :: 'a Vector ⇒ 'a Vector ⇒ 'a Vector (infixr ⊕ 100) where
  vecPlus u v = ( | tdir = tdir u + tdir v, xdir = xdir u + xdir v,
                      ydir = ydir u + ydir v, zdir = zdir u + zdir v |

fun vecMinus :: 'a Vector ⇒ 'a Vector ⇒ 'a Vector (infixr ⊖ 100) where
  vecMinus u v = ( | tdir = tdir u - tdir v, xdir = xdir u - xdir v,
                      ydir = ydir u - ydir v, zdir = zdir u - zdir v |

fun vecNegate :: 'a Vector ⇒ 'a Vector (¬) where
  vecNegate u = ( | tdir = uminus (tdir u), xdir = uminus (xdir u),
                         ydir = uminus (ydir u), zdir = uminus (zdir u) |

fun innerProd :: 'a Vector ⇒ 'a Vector ⇒ 'a (infix dot 50) where
  innerProd u v = (tdir u * tdir v) + (xdir u * xdir v) +
                   (ydir u * ydir v) + (zdir u * zdir v)

fun sqrlen :: 'a Vector ⇒ 'a where sqrlen u = (u dot u)

fun minkowskiProd :: 'a Vector ⇒ 'a Vector ⇒ 'a (infix mdot 50) where
  minkowskiProd u v = (tdir u * tdir v) − ((xdir u * xdir v) + (ydir u * ydir v) + (zdir u * zdir v))

fun mSqrLen :: 'a Vector ⇒ 'a where mSqrLen u = (u mdot u)

fun vecScale :: 'a ⇒ 'a Vector ⇒ 'a Vector (infix ** 200) where

vecScale k u = \{
  tdir = k * tdir u, xdir = k * xdir u, ydir = k * ydir u, zdir = k * zdir u
}\)

fun orthogonal :: 'a Vector ⇒ 'a Vector ⇒ bool (infix ⊥ 150) where
orthogonal u v = (u dot v = 0)

lemma lemVecZeroMinus:
  shows 0 ⊕ u = ~ u
  by simp

lemma lemVecSelfMinus:
  shows u ⊕ u = 0
  by simp

lemma lemVecPlusCommute:
  shows u ⊕ v = v ⊕ u
  by (simp add: add.commute)

lemma lemVecPlusAssoc:
  shows u ⊕ (v ⊕ w) = (u ⊕ v) ⊕ w
  by (simp add: add.assoc)

lemma lemVecPlusMinus:
  shows u ⊕ (~ v) = u ⊖ v
  by (simp add: local.add-uminus-conv-diff)

lemma lemDotCommute:
  shows (u dot v) = (v dot u)
  by (simp add: mult.commute)

lemma lemMDotCommute:
  shows (u mdot v) = (v mdot u)
  by (simp add: mult.commute)

lemma lemScaleScale:
  shows a**(b**u) = (a*b)**u
  by (simp add: mult.assoc)
lemma lemScale1:
  shows $1 \ast\ast u = u$
  by simp

lemma lemScale0:
  shows $0 \ast\ast u = 0$
  by simp

lemma lemScaleNeg:
  shows $(-k)\ast\ast u = \sim (k\ast\ast u)$
  by simp

lemma lemScaleOrigin:
  shows $k\ast\ast 0 = 0$
  by auto

lemma lemScaleOverAdd:
  shows $k\ast\ast (u \oplus v) = k\ast\ast u \oplus k\ast\ast v$
  by (simp add: semiring-normalization-rules(34))

lemma lemAddOverScale:
  shows $a\ast\ast u \oplus b\ast\ast u = (a+b)\ast\ast u$
  by (simp add: semiring-normalization-rules(1))

lemma lemScaleInverse:
  assumes $k \neq (0::'a)$
  and $v = k\ast\ast u$
  shows $u = (inverse k)\ast\ast v$

proof
  have $(inverse k)\ast\ast v = (inverse k \ast k)\ast\ast u$
    by (simp add: lemScaleScale assms(2) mult_assoc)
  thus ?thesis by (metis (lifting) field-inverse assms(1) lemScale1)
qed
lemma lemOrthoSym:
  assumes $a \perp v$
  shows $v \perp u$
  by (metis assms(1) lemDotCommute orthogonal.lem_simps)

end

class Points = Quantities + Vectors
begin

abbreviation origin :: 'a Point where
  origin ≡ ($\{ tval = 0, xval = 0, yval = 0, zval = 0 \}$)

fun vectorJoining :: 'a Point ⇒ 'a Point ⇒ 'a Vector (infix - to -) where
  vectorJoining p q
  = ($\{ tdir = tval q - tval p, xdir = xval q - xval p,
  ydir = yval q - yval p, zdir = zval q - zval p \}$)

fun moveBy :: 'a Point ⇒ 'a Vector ⇒ 'a Point (infixl ⇝ 100) where
  moveBy p u
  = ($\{ tval = tval p + tdir u, xval = xval p + xdir u,
  yval = yval p + ydir u, zval = zval p + zdir u \}$)

fun positionVector :: 'a Point ⇒ 'a Vector where
  positionVector p = ($\{ tdir = tval p, xdir = xval p, ydir = yval p, zdir = zval p \}$)

fun before :: 'a Point ⇒ 'a Point ⇒ bool (infix ≲ 100) where
  before p q = (tval p < tval q)

fun after :: 'a Point ⇒ 'a Point ⇒ bool (infix ≳ 100) where
  after p q = (tval p > tval q)

fun sametime :: 'a Point ⇒ 'a Point ⇒ bool (infixr ≈ 100) where
  sametime p q = (tval p = tval q)

lemma lemFromToTo:
  shows (from p to q) ⊕ (from q to r) = (from p to r)
  proof
    have shared: $\forall$ valp valq valr. ($\forall$ q) - valp + ($\forall$ r - valq) = valr - valp
    by (metis add-uminus-conv-diff add-diff-cancel
    semiring-normalization-rules(24) semiring-normalization-rules(25))
    thus $\{ \text{thesis} \}$ by auto
  qed
lemma lemMoveByMove:
  shows $p \rightsquigarrow u \rightsquigarrow v = p \rightsquigarrow (u \oplus v)$
  by (simp add: add.assoc)

lemma lemScaleLinear:
  shows $p \rightsquigarrow a**u \rightsquigarrow b**v = p \rightsquigarrow (a**u \oplus b**v)$
  by (simp add: add.assoc)
end

class Lines = Quantities + Vectors + Points
begin

fun onAxisT :: 'a Point ⇒ bool where
  onAxisT u = ((xval u = 0) ∧ (yval u = 0) ∧ (zval u = 0))

fun space2 :: ('a Point) ⇒ ('a Point) ⇒ 'a where
  space2 u v = (xval u - xval v)*(xval u - xval v)
  + (yval u - yval v)*(yval u - yval v)
  + (zval u - zval v)*(zval u - zval v)

fun time2 :: ('a Point) ⇒ ('a Point) ⇒ 'a where
  time2 u v = (tval u - tval v)*(tval u - tval v)

fun speed :: ('a Point) ⇒ ('a Point) ⇒ 'a where
  speed u v = (space2 u v / time2 u v)

fun mkLine :: 'a Point => 'a Vector ⇒ 'a Line where
  mkLine b d = (| basepoint = b, direction = d |)

fun lineJoining :: 'a Point ⇒ 'a Point ⇒ 'a Line (line joining - to -) where
  lineJoining p q = (| basepoint = p, direction = from p to q |)

fun parallel :: 'a Line ⇒ 'a Line ⇒ bool (∥) where
  parallel lineA lineB = ((direction lineA = vecZero) ∨ (direction lineB = vecZero)
  ∨ (∃k.(k ≠ (0::'a) ∧ direction lineB = k**direction lineA)))

fun collinear :: 'a Point ⇒ 'a Point ⇒ 'a Point ⇒ bool where
  collinear p q r = (∃α β. ( (α + β = 1) ∧ positionVector p = α**(positionVector q) ⊕ β**(positionVector r) ))

fun inLine :: 'a Point ⇒ 'a Line ⇒ bool where
  inLine p l = collinear p (basepoint l) (basepoint l ⇒ direction l)
fun meets :: 'a Line ⇒ 'a Line ⇒ bool where
meets line1 line2 = (∃p.(inLine p line1 ∧ inLine p line2))

lemma lemParallelReflexive:
  shows lineA ∥ lineA
proof –
  def dir ≡ direction lineA
  have (1 ≠ 0) ∧ (dir = 1 ** dir) by simp
  thus ?thesis by (metis dir-def parallel.simps)
qed

lemma lemParallelSym:
  assumes lineA ∥ lineB
  shows lineB ∥ lineA
proof –
  have case1: direction lineA = vecZero → ?thesis by auto
  have case2: direction lineB = vecZero → ?thesis by auto
  { assume case3: direction lineA ≠ vecZero ∧ direction lineB ≠ vecZero
    have exists-kab: ∃kab.(kab ≠ (0::'a) ∧ direction lineB = kab ** direction lineA)
      by (metis parallel.simps assms(1) case3)
    def kab ≡ SOME kab.(kab ≠ (0::'a) ∧ direction lineB = kab ** direction lineA)
    have kab-nonzero: kab ≠ 0 by (metis inverse-zero-imp-zero kab-props kab-def)
    have direction lineA = kab ** direction lineB by (metis kab-def lemScaleInverse kab-props)
    hence ?thesis by (metis kab-nonzero parallel.simps)
  }
  from this have (direction lineA ≠ vecZero ∧ direction lineB ≠ vecZero) → ?thesis by blast
  thus ?thesis by (metis case1 case2)
qed

lemma lemParallelTrans:
  assumes lineA ∥ lineB
  and lineB ∥ lineC
  and direction lineB ≠ vecZero
  shows lineA ∥ lineC
proof –
have case1: direction lineA = vecZero → thesis by auto
have case2: direction lineC = vecZero → thesis by auto

{ assume case3: direction lineA ≠ vecZero ∧ direction lineC ≠ vecZero
  have exists-kab: ∃ kab.(kab ≠ (0::'a) ∧ direction lineB = kab++direction lineA)
    by (metis parallel.simps assms(1) case3 assms(3))
  then obtain kab where kab-props: kab ≠ 0 ∧ direction lineB = kab++direction lineA by auto
  have exists-kbc: ∃ kbc.(kbc ≠ (0::'a) ∧ direction lineC = kbc++direction lineB)
    by (metis parallel.simps assms(2) case3 assms(3))
  then obtain kbc where kbc-props: kbc ≠ 0 ∧ direction lineC = kbc++direction lineB by auto
  def kac ≡ kbc * kab
  have kac-nonzero: kac ≠ 0 by (metis kab-props kac-def kbc-props no-zero-divisors)
  have direction lineC = kac++direction lineA
    by (metis kab-props kbc-props kac-def lemScaleScale)
  hence thesis by (metis kac-nonzero parallel.simps)
} from this have (direction lineA ≠ vecZero ∧ direction lineC ≠ vecZero) → thesis by blast

thus thesis by (metis case1 case2)
qed

lemma lemLineIdentity:
  assumes lineA = ( basepoint = basepoint lineB, direction = direction lineB )
  shows lineA = lineB
proof –
have basepoint lineA = basepoint lineB ∧ direction lineA = direction lineB
  by (simp add: assms(1))
thus thesis by simp
qed

lemma lemDirectionJoining:
  shows vectorJoining p (p ⇝ v) = v
proof –
  have ∀ a b.(a + b − a = b)
    by (metis add-uminus-cone-diffiff-add-cancel semiring-normalization-rules(24))
  thus thesis by auto
qed

8
lemma lemDirectionFromTo:
  shows direction (line joining p to (p \rightarrow dir)) = dir
proof -
  have direction (line joining p to (p \rightarrow dir)) = from p to (p \rightarrow dir) by simp
  thus ?thesis by (metis lemDirectionJoining)
qed

lemma lemLineEndpoint:
  shows q = p \rightarrow (from p to q)
proof -
  have \forall a b. (b = a + (b - a))
    by (metis diff-add-cancel semiring-normalization-rules(24))
  thus ?thesis by auto
qed

lemma lemNullLine:
  assumes direction lineA = vecZero
  and inLine x lineA
  shows x = basepoint lineA
proof -
  def bp \equiv basepoint lineA
  have collinear x (basepoint lineA) (basepoint lineA \rightarrow direction lineA)
    by (simp add: assms)
  hence collinear x bp (bp \rightarrow vecZero) by (metis bp-def assms)
  hence collinear x bp bp by simp
  hence \exists a b. (a + b = 1) \land
    (positionVector x = a**(positionVector bp) \oplus b**(positionVector bp))
    by (metis collinear.simps)
  hence positionVector x = positionVector bp by (metis lemScale1 lemAddOverScale)
  thus ?thesis by (simp add: bp-def)
qed

lemma lemLineContainsBasepoint:
  shows inLine p (line joining p to q)
proof -
  def linePQ \equiv line joining p to q
  have bp: basepoint linePQ = p by (simp add: linePQ-def)
  have dir: direction linePQ = from p to q by (simp add: linePQ-def)
  have endq: basepoint linePQ \rightarrow direction linePQ = q by (metis bp dir lemLineEndpoint)
  have (1 + 0 = 1) \land (positionVector p = 1**(positionVector p) \oplus 0**(positionVector q))
by auto 

hence collinear p p q by (metis collinear.simps)

hence collinear p (basepoint linePQ) (basepoint linePQ⇾direction linePQ) by (metis bp endq)

thus ?thesis by (simp add: linePQ-def)

qed


lemma lemLineContainsEndpoint:
shows inLine q (line joining p to q)
proof -
  def linePQ ≡ line joining p to q
  have bp: basepoint linePQ = p by (simp add: linePQ-def)
  have dir: direction linePQ = from p to q by (simp add: linePQ-def)
  have endq: basepoint linePQ⇾direction linePQ = q by (metis bp dir lemLineContainsEndpoint)

  have \((0 + 1 = 1) \land (positionVector q = 0** (positionVector p) \oplus 1** (positionVector q))\)
    by auto

  hence collinear q p q by (metis collinear.simps)
  hence collinear q (basepoint linePQ) (basepoint linePQ⇾direction linePQ)
    by (metis bp endq)

  thus ?thesis by (simp add: linePQ-def)

qed


lemma lemDirectionReverse:
shows from q to p = vecNegate (from p to q)
by simp


lemma lemParallelJoin:
assumes line joining p to q \parallel line joining q to r
shows line joining p to q \parallel line joining p to r
proof -
  def linePQ ≡ line joining p to q
  def lineQR ≡ line joining q to r
  def linePR ≡ line joining p to r

  have case1: (direction linePQ = vecZero) \rightarrow ?thesis by (simp add: linePQ-def)
  have case2: (direction linePR = vecZero) \rightarrow ?thesis by (simp add: linePR-def)

  { assume case3: direction linePQ \neq vecZero \land direction linePR \neq vecZero
    { assume case3a: direction lineQR = vecZero
      have inLine r lineQR by (metis lemLineContainsEndpoint lineQR-def)
hence $r =$ basepoint lineQR by (metis lemNullLine case3a)

hence $r = q$ by (simp add: lineQR-def)

hence linePQ = linePR by (simp add: linePQ-def linePR-def)

hence ?thesis by (metis lemParallelReflexive linePQ-def linePR-def)

} from this have rtp3a: direction lineQR = vecZero --- ?thesis by blast

{ assume case3b: direction lineQR ≠ vecZero

  def dirPQ ≡ from p to q

  have dir-pq: direction linePQ = dirPQ by (simp add: linePQ-def dirPQ-def)

  def dirQR ≡ from q to r

  have dir-qr: direction lineQR = dirQR by (simp add: lineQR-def dirQR-def)

  have exists-k: $\exists k. (k ≠ 0 \land direction lineQR = k\cdot direction linePQ)$

  by (metis linePQ-def lineQR-def assms (1) parallel.simps case3b case3)

  then obtain k where k-props: $k ≠ 0 \land direction lineQR = k\cdot direction linePQ$ by (metis dir-pq dir-qr)

  def scalar ≡ $1 + k$

  have $q = p \rightsquigarrow dirPQ \land r = q \rightsquigarrow dirQR$ by (metis lemLineEndpoint dirPQ-def dirQR-def)

  hence $r = p \rightsquigarrow dirPQ \rightsquigarrow (k\cdot dirPQ)$ by (metis k-props)

  hence scalarPR: $r = p \rightsquigarrow scalar\cdot dirPQ$

  by (metis lemScaleLinear lemScale1 lemAddOverScale scalar-def)

  { assume scalar0: scalar = 0

    have $r = p$ by (simp add: lemScale0 scalarPR scalar0)

    hence direction linePR = vecZero by (simp add: linePR-def)

    hence False by (metis case3)

  }

  from this have scalar-nonzero: scalar ≠ 0 by blast

  have linePR = line joining p to $p \rightsquigarrow scalar\cdot dirPQ$

  by (simp add: linePR-def scalarPR)

  hence direction linePR = scalar\cdot dirPQ by (metis lemDirectionFromTo)

  hence scalar-props: scalar ≠ 0 \land direction linePR = scalar\cdot direction linePQ

  by (metis scalar-nonzero dir-pq)

  hence ?thesis by (metis parallel.simps linePR-def linePQ-def)

  } from this have direction lineQR ≠ vecZero --- ?thesis by blast

  hence ?thesis by (metis rtp3a)
from this have \((\text{direction } \text{line}PQ \neq \vec{0} \land \text{direction } \text{line}PR \neq \vec{0})\)

\[\Rightarrow \text{thesis by blast}\]

thus \text{thesis by (metis case1 case2)}

\text{qed}

\text{lemma lemDirectionCollinear:}

\text{shows collinear } u \ v \ (v \rightsquigarrow d) \overset{\Leftrightarrow}{\leftrightarrow} (\exists \beta.(\text{from } u \ \text{to } v = (-\beta)**d))

\text{proof –}

\text{have basic1: } \forall \ u \ v.(\text{positionVector } (u \rightsquigarrow v)) = (\text{positionVector } u) \oplus v \text{ by simp}

\text{have basic2: } \forall \ u \ v \ w.(u = v \oplus w \Rightarrow v \ominus u = \vec{\text{Negate }} w )

\text{apply auto}

\text{by (metis add-uminus-conv-diff diff-add-cancel minus-add semiring-normalization-rules(24)) +}

\text{have basic3: } \forall \ u \ v w.(\text{from } u \ \text{to } v = \text{positionVector } v \ominus \text{positionVector } u) \text{ by simp}

\text{have basic4: } \forall \ u \ v w.(v \ominus u = \vec{\text{Negate }} w \Rightarrow u = v \oplus w)

\text{apply auto}

\text{by (metis add-uminus-conv-diff diff-add-cancel lemScale1 mult.left-neutral semiring-normalization-rules(24) vecScale.simps)}

\{ assume \text{assm: collinear } u \ v \ (v \rightsquigarrow d)

\text{have } \exists \alpha \beta. ( (\alpha + \beta = 1) \land \text{positionVector } u = \alpha**(\text{positionVector } v) \oplus \beta**(\text{positionVector } (v \rightsquigarrow d)) ) \text{ by (metis assm collinear.simps)}

\text{then obtain } \alpha \beta \text{ where props: } (\alpha + \beta = 1) \land \text{positionVector } u = \alpha**(\text{positionVector } v) \oplus \beta**(\text{positionVector } (v \rightsquigarrow d)) \text{ by auto}

\text{hence positionVector } u = 1**(\text{positionVector } v) \oplus \beta**d \text{ by (metis basic1 lemScaleOverAdd lemVecPlusAssoc lemAddOverScale props)}

\text{hence positionVector } u = \text{positionVector } v \oplus \beta**d \text{ by (metis lemScale1)}

\text{hence positionVector } v \ominus \text{positionVector } u = (-\beta)**d \text{ by (metis basic2 lemScaleNeg)}

\text{hence } \exists \beta.(\text{from } u \ \text{to } v = (-\beta)**d) \text{ by (metis basic3)}

\}

from this have \text{fwd: collinear } u \ v \ (v \rightsquigarrow d) \overset{\Rightarrow}{\rightarrow} (\exists \beta.(\text{from } u \ \text{to } v = (-\beta)**d))

\text{by blast}

\{ assume \exists \beta.(\text{from } u \ \text{to } v = (-\beta)**d)

\text{then obtain } \beta \text{ where \text{assm: from } u \ \text{to } v = (-\beta)**d \text{ by auto}

\text{def } \alpha \equiv 1 - \beta

\text{have } \alpha \beta-\text{sum: } \alpha + \beta = 1 \text{ by (simp add: } \alpha\text{-def)}

\text{have from } u \ \text{to } v = \vec{\text{Negate }} (\beta**d) \text{ by (metis assm lemScaleNeg)}

\text{hence positionVector } v \ominus \text{positionVector } u = \vec{\text{Negate }} (\beta**d) \text{ by auto}

\text{hence positionVector } u = \text{positionVector } v \oplus \beta**d \text{ by (metis basic4)}

\text{hence positionVector } u = 1**(\text{positionVector } v) \oplus \beta**d

\}
by (metis lemScale1)
hence \((\alpha + \beta = 1) \land \)
\text{positionVector } u = \alpha**(\text{positionVector } v) + \beta**(\text{positionVector } (v \rightsquigarrow d))

\text{by (metis } \alpha\beta\text{-sam basic1 lemScaleOverAdd lemVecPlusAssoc lemAddOverScale})

\text{hence collinear } u \text{ } v \text{ (v } \rightsquigarrow \text{d) by auto}

\}\text{from this have } (\exists \beta. \text{ (from } u \text{ to } v = (-\beta)**d)) \longrightarrow \text{collinear } u \text{ } v \text{ (v } \rightsquigarrow \text{d) by blast}

\text{thus } ?\text{thesis by (metis fwd)}

\text{qed}

\text{lemma lemParallelNotMeet:}
\text{assumes lineA } \parallel \text{ lineB and direction lineA } \neq \text{ vecZero and direction lineB } \neq \text{ vecZero and inLine x lineA and } \neg\text{(inLine x lineB) shows } \neg\text{(meets lineA lineB)}
\text{proof –}

\text{have basic: } \forall \text{ p q v a. (from } p \text{ to } q = a**v \longrightarrow \text{from } q \text{ to } p = (-a)**v)
\text{apply (simp add: lemScaleNeg) by (metis minus-diff-eq)}

\text{def bpA } \equiv \text{ basepoint lineA}
\text{def dirA } \equiv \text{ direction lineA}
\text{def bpB } \equiv \text{ basepoint lineB}
\text{def dirB } \equiv \text{ direction lineB}

\text{have lineB } \parallel \text{ lineA by (metis lemParallelSym assms(1))}
\text{hence exists-kab: } \exists \text{ kab. (kab } \neq \text{ (0::a) } \land \text{ direction lineA = kab**direction lineB)}

\text{by (metis parallel.simps assms(2) assms(3))}
\text{then obtain kab where kab-props: kab } \neq \text{ 0 } \land \text{ dirA = kab**dirB by (metis dirA-def dirB-def)}

\text{have collinear x bpA (bpA } \rightsquigarrow \text{ dirA) by (metis assms(4) inLine.simps bpA-def dirA-def)}
\text{then obtain } \beta \text{ where from x to bpA = (-}\beta\text{)**dirA by (metis lemDirectionCollinear)}

\text{hence x-to-bpA: from x to bpA = ((-}\beta)\text{)*kab)**dirB by (metis lemScaleScale kab-props)}

{
**assume converse:** meets lineA lineB

**have** \( \exists p. (\text{inLine } p \text{ lineA } \land \text{inLine } p \text{ lineB}) \) by (metis converse meets.simps)

**then obtain** \( p \) **where** p-in-AB: \( \text{inLine } p \text{ lineA } \land \text{inLine } p \text{ lineB} \) by auto

**have** collinear p bpA (bpA \( \rightsquigarrow \) dirA) by (metis p-in-AB inLine.simps bpA-def dirA-def)

**then obtain** \( \beta_A \) **where** from p to bpA = \((-\beta_A)\)**dirA by (metis lemDirectionCollinear)

**hence** from bpA to p = \((\beta_A)\)**dirA by (metis basic minus-minus)

**hence** bpA-to-p: from p to bpA = \((\beta_A)\)\*kab**dirB by (metis lemScaleScale kab-props)

**have** collinear p bpB (bpB \( \rightsquigarrow \) dirB) by (metis p-in-AB inLine.simps bpB-def dirB-def)

**then obtain** \( \beta_B \) **where** p-to-bpB: from p to bpB = \((-\beta_B)\)**dirB by (metis lemDirectionCollinear)

**def** \( \gamma \equiv -((-\beta)\*kab + (\beta_A)\*kab) + (-\beta_B)) \)

**have** x-to-bpB: (from x to bpA) \( \oplus \) (from bpA to p) \( \oplus \) (from p to bpB) = (from x to bpB)

**by** (metis lemFromToTo)

**hence** from x to bpB = \((-\gamma)\)**dirB by (metis x-to-bpA bpA-to-p p-to-bpB)

**hence** from x to bpB = \((-\gamma)\)**dirB by (metis lemAddOverScale add.assoc \( \gamma \)-def minus-minus)

**hence** collinear x bpB (bpB \( \rightsquigarrow \) dirB) by (metis lemDirectionCollinear)

**hence** inLine x lineB by (metis inLine.simps bpB-def dirB-def)

**from** this have meets lineA lineB \( \longrightarrow \) inLine x lineB by blast

**thus** \( \forall \text{thesis by (metis assms(5))} \)

**qed**

**lemma lemAxisIsLine:**

**assumes** onAxisT x

and onAxisT y

and onAxisT z

and \( x \not= y \)

and \( y \not= z \)

and \( z \not= x \)

**shows** collinear x y z

**proof**

**def** ratio \( \equiv -(\text{tval } y - \text{tval } x) \div (\text{tval } z - \text{tval } y) \)

**have** x-onAxis: \( \text{xval } x = 0 \land \text{yval } x = 0 \land \text{zval } x = 0 \) by (metis assms(1) onAxisT.simps)

**have** y-onAxis: \( \text{xval } y = 0 \land \text{yval } y = 0 \land \text{zval } y = 0 \) by (metis assms(2)
have \( z\text{-onAxis} \): \( xval \, z = 0 \land yval \, z = 0 \land zval \, z = 0 \) by (metis assms(3) onAxisT.simps)

hence \( \text{tval} \, z \neq \text{tval} \, y \) by (metis assms(5) eq-iff-diff-eq-0)

hence \( \text{tval}_{z\text{-nonzero}}: \text{tval} \, z - \text{tval} \, y \neq 0 \) by (metis eq-iff-diff-eq-0)

have \( x\text{-to-y}: \text{from} \, x \, \text{to} \, y = (| \, tdir = \text{tval} \, y - \text{tval} \, x, xdir = 0, ydir = 0, zdir = 0 \) |)

by (simp add: x-onAxis y-onAxis)

have \( y\text{-to-z}: \text{from} \, y \, \text{to} \, z = (| \, tdir = \text{tval} \, z - \text{tval} \, y, xdir = 0, ydir = 0, zdir = 0 \) |)

by (simp add: y-onAxis z-onAxis)

have \( \text{from} \, x \, \text{to} \, y = (-\text{ratio})^*(\text{from} \, y \, \text{to} \, z) \)

apply (simp add: x-to-y y-to-z ratio-def)

by (metis diff-self eq-divide-imp minus-diff-eq mult-eq-0-iff tvalyz-nonzero x-onAxis y-onAxis z-onAxis)

hence \( \text{collinear} \, x \, y \; (y \rightsquigarrow (\text{from} \, y \, \text{to} \, z)) \) by (metis lemDirectionCollinear)

thus \( ?\text{thesis} \) by (metis lemLineEndpoint)

qed

lemma lemSpace2Sym:

shows \( \text{space2} \, x \, y = \text{space2} \, y \, x \)

proof -

def \( xsep \equiv xval \, x - xval \, y \)

def \( ysep \equiv yval \, x - yval \, y \)

def \( zsep \equiv zval \, x - zval \, y \)

have \( \text{spacexy} : \text{space2} \, x \, y = (xsep*xsep) + (ysep*ysep) + (zsep*zsep) \)

by (simp add: xsep-def ysep-def zsep-def)

have \( \text{spaceyx} : \text{space2} \, y \, x = (-xsep)*(-xsep) + (-ysep)*(-ysep) + (-zsep)*(-zsep) \)

by (simp add: xsep-def ysep-def zsep-def)

thus \( ?\text{thesis} \) by (metis spacexy diff-0-right minus-diff-eq minus-mult-left minus-mult-right)

qed

lemma lemTime2Sym:

shows \( \text{time2} \, x \, y = \text{time2} \, y \, x \)

proof -

def \( tsep \equiv \text{tval} \, x - \text{tval} \, y \)

have \( \text{timexy} : \text{time2} \, x \, y = tsep*tsep \)

by (simp add: tsep-def)

have \( \text{timeyx} : \text{time2} \, y \, x = (-tsep)*(-tsep) \)

by (simp add: tsep-def)

thus \( ?\text{thesis} \) by (metis timexy diff-0-right minus-diff-eq minus-mult-left minus-mult-right)

qed
class Planes = Quantities + Lines
begin
  fun mkPlane :: 'a Point ⇒ 'a Vector ⇒ 'a Vector ⇒ 'a Plane
    where
    mkPlane b d1 d2 = (| pbasepoint = b, direction1 = d1, direction2 = d2 |)

  fun coplanar :: 'a Point ⇒ 'a Point ⇒ 'a Point ⇒ bool
    where
    coplanar e x y z = (∃ α β γ. (α + β + γ = 1) ∧
      positionVector e = (α**(positionVector x) ⊕ β**(positionVector y) ⊕ γ**(positionVector z))
    )

  fun inPlane :: 'a Point ⇒ 'a Plane ⇒ bool
    where
    inPlane e pl = coplanar e (pbasepoint pl) (pbasepoint pl ⇝ direction1 pl) (pbasepoint pl ⇝ direction2 pl)

  fun samePlane :: 'a Plane ⇒ 'a Plane ⇒ bool
    where
    samePlane pl pl' = (inPlane (pbasepoint pl) pl') ∧
      inPlane (pbasepoint pl ⇒ direction1 pl) pl' ∧
      inPlane (pbasepoint pl ⇒ direction2 pl) pl'

lemma lemPlaneContainsBasePoint:
  shows inPlane (pbasepoint pl) pl
proof –
  def α ≡ 1::'a
  def β ≡ 0::'a
  def γ ≡ 0::'a
  have rtp1: α + β + γ = 1 by (simp add: α-def β-def γ-def)

  def e ≡ pbasepoint pl
  def x ≡ pbasepoint pl
  def y ≡ pbasepoint pl ⇒ direction1 pl
  def z ≡ pbasepoint pl ⇒ direction2 pl
  have rtp2: positionVector e = α**(positionVector x)
    ⊕ β**(positionVector y) ⊕ γ**(positionVector z)
    by (simp add: e-def x-def y-def z-def α-def β-def γ-def)

  have sameplane: coplanar e x y z by (metis coplanar simps rtp1 rtp2)
  hence coplanar e (pbasepoint pl) (pbasepoint pl ⇒ direction1 pl) (pbasepoint pl ⇒ direction2 pl)
    by (simp add: x-def y-def z-def)
  hence inPlane e pl by simp
  thus ?thesis by (simp add: e-def)
qed
end
class Cones = Quantities + Lines + Planes +
fixes

  tangentPlane :: 'a Point ⇒ 'a Cone ⇒ 'a Plane
assumes

  AxTangentBase: pbasepoint (tangentPlane e cone) = e
and

  AxTangentVertex: inPlane (vertex cone) (tangentPlane e cone)
and

  AxConeTangent: (onCone e cone) →
    ((inPlane pt (tangentPlane e cone) ∧ onCone pt cone)
     ←→ collinear (vertex cone) e pt)
and

  AxParallelCones: (onCone e econe ∧ e ≠ vertex econe ∧ onCone f econe ∧ f ≠ vertex fcone
    ∧ inPlane f (tangentPlane e econe))
    → (samePlane (tangentPlane e econe) (tangentPlane f econe)
    ∧ ((lineJoining (vertex econe) e) || (lineJoining (vertex fcone) f)))
and

  AxParallelConesE: outsideCone f cone
    → (∃ e. (onCone e econe ∧ e ≠ vertex econe ∧ inPlane f (tangentPlane e econe)))
and

  AxSlopedLineInVerticalPlane: [onAxisT e; onAxisT f; e ≠ f; ~(onAxisT g)]
    → (∀ s. (collinear e g p ∧ (space2 p f = (s*s)*time2 p f)))

begin

  fun onCone :: 'a Point ⇒ 'a Cone ⇒ bool where
    onCone p cone
    = (space2 (vertex cone) p = (slope cone * slope cone) * time2 (vertex cone) p )

  fun insideCone :: 'a Point ⇒ 'a Cone ⇒ bool where
    insideCone p cone
    = (space2 (vertex cone) p < (slope cone * slope cone) * time2 (vertex cone) p)

  fun outsideCone :: 'a Point ⇒ 'a Cone ⇒ bool where
outsideCone \( p \) cone
= (space2 (vertex cone) \( p \) > (slope cone * slope cone) * time2 (vertex cone) \( p \))

fun \( mkCone :: \{\text{Point} \Rightarrow \{\text{Cone} \} \Rightarrow \{\text{Point} \} \Rightarrow \{\text{Cone} \} \} \) where
\( mkCone v s = (\mid \text{vertex} = v, \text{slope} = s \mid) \)

lemma lemVertexOnCone:
  shows onCone (vertex cone) cone
by simp

lemma lemOutsideNotOnCone:
  assumes outsideCone \( f \) cone
  shows \( \neg \) (onCone \( f \) cone)
by (metis assms less-irrefl onCone.simps outsideCone.simps)
end

class SpaceTime = Quantities + Vectors + Points + Lines + Planes + Cones
end

theory SomeFunc
  imports Main
begin

fun someFunc :: ('a => 'b => bool) => 'a => 'b where
someFunc \( P \) \( x \) = (SOME \( y \). \( P \) \( x \) \( y \))

lemma lemSomeFunc:
  assumes \( \exists \ y . \ P \ x \ y \)
  and \( f = \text{someFunc} \ P \)
  shows \( \ P \ x (f \ x) \)
proof –
  have \( f \ x = (\text{SOME} \ y . (P \ x \ y)) \)
  using assms(2) by simp
  thus \( ?\text{thesis using} \ \text{assms(1)} \)
  by (simp add: someI-ex)
qed

end

theory Axioms
  imports SpaceTime SomeFunc
begin
record Body =
  Ph :: bool
  IOb :: bool

class WorldView = SpaceTime +
fixeds
  W :: Body ⇒ Body ⇒ 'a Point ⇒ bool (- sees - at -)
and
  wvt :: Body ⇒ Body ⇒ 'a Point ⇒ 'a Point
assumes
  AxWVT: [ IOb m; IOb k ] ⇒ (W k b x ↔ W m b (wvt m k x))
and
  AxWVTSym: [ IOb m; IOb k ] ⇒ (y = wvt k m x ↔ x = wvt m k y)
begin
end

class AxiomPreds = WorldView
begin
  fun sqrtTest :: 'a ⇒ 'a ⇒ bool where
    sqrtTest x r = ((r ≥ 0) ∧ (r*r = x))
  fun cTest :: Body ⇒ 'a ⇒ bool where
    cTest m v = ( (v > 0) ∧ ( ∀ x y . ( (∃ p. (Ph p ∧ W m p x ∧ W m p y)) ↔ (space2 x y = (v * v)*time2 x y)) ) )
end

class AxEuclidean = AxiomPreds + Quantities +
assumes
  AxEuclidean: (x ≥ Groups.zero-class.zero) ⇒ (∃ r. sqrtTest x r)
begin
  abbreviation sqrt :: 'a ⇒ 'a where
    sqrt ≡ someFunc sqrtTest
  lemma lemSqrt:
    assumes x ≥ 0
    and r = sqrt x
shows \( r \geq 0 \land r \cdot r = x \)

proof -

have rootExists: \((\exists r. \sqrt{\text{Test}} \ x \ r)\) by (metis AxEuclidean assms(1))

hence sqrtTest \( x \) \((\sqrt{\text{Test}} \ x)\) by (metis lemSomeFunc)

thus ?thesis using assms(2) by simp

qed

class AxLight = WorldView +

assumes

AxLight: \( \exists m \ v. (\text{IOb} \ m \land (v > (0::'a))) \land (\forall x \ y. (\exists p. (\text{Ph} \ p \land W \ m \ p \ x \land W \ m \ p \ y)) \longleftrightarrow (\text{space2} \ x \ y = (v \cdot v) \cdot \text{time2} \ x \ y)) \)

begin

abbreviation \( c :: \text{Body} \Rightarrow 'a \) where

\( c \equiv \text{someFunc} \ c\text{Test} \)

fun lightcone :: \( \text{Body} \Rightarrow 'a \ \text{Point} \Rightarrow 'a \ \text{Cone} \) where

\( \text{lightcone} \ m \ v = \text{mkCone} \ v \ (c \ m) \)

lemma lemCProps:

assumes \( \text{IOb} \ m \)

and \( v = c \ m \)

shows \( (v > 0) \land (\forall x \ y. ((\exists p. (\text{Ph} \ p \land W \ m \ p \ x \land W \ m \ p \ y)) \longleftrightarrow (\text{space2} \ x \ y = (c \ m \cdot c \ m) \cdot \text{time2} \ x \ y))) \)

proof -

have vExists: \((\exists v. \ c\text{Test} \ m \ v)\) by (metis AxPh assms(1))

hence cTest \( m \) \((c \ m)\) by (metis lemSomeFunc)

thus ?thesis using assms(2) by simp

qed

lemma lemCCone:

assumes \( \text{IOb} \ m \)
and \( \text{onCone } y \) (lightcone \( m \times x \))

shows \( \exists p. (\text{Ph } p \land W \ m \ p \ x \land W \ m \ p \ y) \)

proof -

have \( (\exists p. (\text{Ph } p \land W \ m \ p \ x \land W \ m \ p \ y)) \)

by \((\text{smt assms(1) lemCProps})\)

hence \( \text{ph-exists}: (\text{space}2 \ x \ y = (c \ m \ast c \ m) \ast \text{time}2 \ x \ y) \leftrightarrow (\exists p. (\text{Ph } p \land W \ m \ p \ x \land W \ m \ p \ y)) \)

by \(\text{metis}\)

def \( \text{lcm} x \equiv \text{lightcone } m \times x \)

have \( \text{lcm} \times \text{-vertex}: \text{vertex } \text{lcm} x = x \)

by \((\text{simp add: lcmx-def})\)

have \( \text{lcm} \times \text{-slope}: \text{slope } \text{lcm} x = c \ m \)

by \((\text{simp add: lcmx-def})\)

have \( \text{onCone } y \text{ } \text{lcm} x \rightarrow (\text{space}2 \ y \ x = (c \ m \ast c \ m) \ast \text{time}2 \ x \ y) \)

by \((\text{metis lcmx-vertex lcmx-slope onCone.simps})\)

hence \( \text{space}2 \ x \ y = (c \ m \ast c \ m) \ast \text{time}2 \ x \ y \)

by \((\text{metis lcmx-def assms(2)})\)

thus \( ?\text{thesis} \)

by \(\text{metis } \text{ph-exists}\)

qed

lemma \( \text{lemCPos}: \)

assumes \( \text{IOb } m \)

shows \( c \ m > 0 \)

by \((\text{metis assms(1) lemCProps})\)

lemma \( \text{lemCPhoton}: \)

assumes \( \text{IOb } m \)

shows \( \forall x \ y. (\exists p. (\text{Ph } p \land W \ m \ p \ x \land W \ m \ p \ y)) \leftrightarrow (\text{space}2 \ x \ y = (c \ m \ast c \ m) \ast (\text{time}2 \ x \ y)) \)

by \((\text{metis assms(1) lemCProps})\)

end

class \( \text{AxEv} = \text{WorldView} + \text{AxEv} \)

assumes \( \text{AxEv}: [\text{IOb } m; \text{IOb } k] \implies (\exists y. (\forall b. (W \ m \ b \ x \leftrightarrow W \ k \ b \ y))) \)

begin

end

class \( \text{AxThExp} = \text{WorldView} + \text{AxPh} + \text{AxThExp} \)

assumes \( \text{AxThExp}: \text{IOb } m \implies (\forall x \ y. (\exists k. (\text{IOb } k \land W \ m \ k \ x \land W \ m \ k \ y)) \leftrightarrow (\text{space}2 \ x \ y < (c \ m \ast c \ m) \ast \text{time}2 \ x \ y)) \)

21
class \( AxSelf = WorldView + \)
assumes
\( AxSelf: IOb m \implies (W m m x) \implies (onAxisT x) \)
begin
end

class \( AxC = WorldView + AxPh + \)
assumes
\( AxC: IOb m \implies c m = 1 \)
begin
end

class \( AxSym = WorldView + \)
assumes
\( AxSym: [ IOb m; IOb k ] \implies \\
(W m e x \land W m f y \land W k e x' \land W k f y' \land \\
tval x = tval y \land tval x' = tval y') \implies (space2 x y = space2 x' y') \)
begin
end

class \( AxLines = WorldView + \)
assumes
\( AxLines: [ IOb m; IOb k; collinear x p q ] \implies \\
collinear (wvt k m x) (wvt k m p) (wvt k m q) \)
begin
end

class \( AxPlanes = WorldView + \)
assumes
\( \text{AxPlanes: } [ \text{IOb } m; \text{IOb } k ] \mapsto (\text{coplanar } e \ x \ y \ z \mapsto \text{coplanar } (\text{wvt } k \ m \ e) (\text{wvt } k \ m \ x) (\text{wvt } k \ m \ y) (\text{wvt } k \ m \ z)) \)
begin
end

class \text{AxCones} = \text{WorldView} + \text{AxPh} + 
assumes
\( \text{AxCones: } [ \text{IOb } m; \text{IOb } k ] \mapsto (\text{onCone } x (\text{lightCone } m \ v) \mapsto \text{onCone } (\text{wvt } k \ m \ x) (\text{lightcone } k (\text{wvt } k \ m \ v))) \)
begin
end

class \text{AxTime} = \text{WorldView} +
assumes
\( \text{AxTime: } [ \text{IOb } m; \text{IOb } k ] \mapsto (x \lesssim y \mapsto \text{wvt } k \ m \ x \lesssim \text{wvt } k \ m \ y) \)
begin
end

theory \text{SpecRel}
imports \text{Axioms}
begin

class \text{SpecRel} = \text{WorldView} + \text{AxPh} + \text{AxEv} + \text{AxSelf} + \text{AxSym}

+ \text{AxEuclidean}

+ \text{AxLines} + \text{AxPlanes} + \text{AxCones}
begin

23
lemma lemZEG:
  shows \( z - e = g - e + (z - g) \)
proof -
  have \( g - e + (z - g) = (g - e + z) - g \) by (rule add-diff-eq)
  also have \( (g - e + z) - g = (-e + z) \)
  by (metis local.diff-add-cancel
        local.ring-normalization-cancel
        local.semiring-normalization-rules(2)
        local.semiring-normalization-rules(24)
        local.semiring-normalization-rules(25))
  thus \( \neg \text{thesis} \)
  by (simp add: calculation)
qed

lemma noFTLObserver:
  assumes iobm: \( IO\text{b} \, m \)
  and iobk: \( IO\text{b} \, k \)
  and mke: \( m \text{ sees } k \text{ at } e \)
  and mkf: \( m \text{ sees } k \text{ at } f \)
  and enotf: \( e \neq f \)
  shows \( \text{space2} \, e \, f \leq (c\, m \ast c\, m) \ast \text{time2} \, e \, f \)
proof -

\{ 
  assume converse: \( \text{space2} \, e \, f > (c\, m \ast c\, m) \ast \text{time2} \, e \, f \)

  def eCone \equiv mkCone \, e \, (c\, m) 
  have e-on-econe: \( \text{onCone} \, e \, e\text{Cone} \) by (simp add: eCone-def)

  have e-is-vertex: \( e = \text{vertex} \, e\text{Cone} \) by (simp add: eCone-def)
  have cm-is-slope: \( c\, m = \text{slope} \, e\text{Cone} \) by (simp add: eCone-def)
  hence outside: \( \text{outsideCone} \, f \, e\text{Cone} \)
  by (metis (lifting) e-is-vertex cm-is-slope converse outsideCone.simps)

  have outsideCone f eCone
    \( \longrightarrow (\exists x. (\text{onCone} \, x \, e\text{Cone} \land x \neq \text{vertex} \, e\text{Cone} \land \text{inPlane} \, f \, (\text{tangentPlane} \, x \, e\text{Cone}))) \)
  by (rule AxParallelConesE)

  hence tplane-exists: \( \exists x. (\text{onCone} \, x \, e\text{Cone} \land x \neq \text{vertex} \, e\text{Cone} \land \text{inPlane} \, f \, (\text{tangentPlane} \, x \, e\text{Cone})) \)
  by (metis outside)
then obtain $g$ where $\text{g-props}: (\text{onCone } g \text{ eCone} \land g \neq \text{vertex eCone} \land \text{inPlane } f (\text{tangentPlane } g \text{ eCone}))$

by auto
have $\text{g-on-eCone}: \text{onCone } g \text{ eCone}$ by (metis g-props)
have $\text{g-not-vertex}: g \neq \text{vertex eCone}$ by (metis g-props)

def $\text{tplane} \equiv \text{tangentPlane } g \text{ eCone}$
have $\text{e-in-tplane}: \text{inPlane } e \text{ tplane}$ by (metis AxTangentVertex e-is-vertex tplane-def)
have $\text{f-in-tplane}: \text{inPlane } f \text{ tplane}$ by (metis g-props tplane-def)
have $\text{g-in-tplane}: \text{inPlane } g \text{ tplane}$ by (metis lemPlaneContainsBasePoint tplane-def)

have $(\text{onCone } g \text{ eCone}) \rightarrow
t((\text{inPlane } f (\text{tangentPlane } g \text{ eCone}) \land \text{onCone } f \text{ eCone})
\leftrightarrow \text{collinear } (\text{vertex eCone}) g f)$

by (metis AxConeTangent)

have $(\text{onCone } f \text{ eCone}) \rightarrow
t("\text{collinear } e g f \rightarrow \text{onCone } f \text{ eCone})$

by (metis AxConeTangent)

hence $\text{g-not-collinear}: \neg (\text{collinear } e g f)$

by (metis axconetangent)


def $\text{wvte} \equiv \text{wvt } k \text{ m } e$
def $\text{wvtf} \equiv \text{wvt } k \text{ m } f$
def $\text{wvtg} \equiv \text{wvt } k \text{ m } g$

have $W k k \text{ wte}$ by (metis wvte-def AxWVT mke iobm iobk)
hence $\text{wvte-onAxis}: \text{onAxisT } \text{ wte}$ by (metis AxSelf iobk)

have $W k k \text{ wtf}$ by (metis wvtf-def AxWVT mkf iobm iobk)
hence $\text{wvtf-onAxis}: \text{onAxisT } \text{ wtf}$ by (metis AxSelf iobk)

have $\text{wte-inv}: e = \text{wvt } m k \text{ wte}$ by (metis AxWVTSym iobk iobm wte-def)
have $\text{wtf-inv}: f = \text{wvt } m k \text{ wtf}$ by (metis AxWVTSym iobk iobm wtf-def)
have $\text{wtg-inv}: g = \text{wvt } m k \text{ wtg}$ by (metis AxWVTSym iobk iobm wtg-def)

have $\text{e-not-g}: e \neq g$ by (metis e-is-vertex g-not-vertex)
have $\text{f-not-g}: f \neq g$ by (metis outside lemOutsideNotOnCone g-on-eCone)

have $\text{wte-e-not-f}: \text{wte} \neq \text{wtf}$ by (metis wte-inv wtf-inv enotf)
have $\text{wtf-f-not-g}: \text{wtf} \neq \text{wtg}$ by (metis wtf-inv wtg-inv f-not-g)
have $\text{wtg-g-not-e}: \text{wtg} \neq \text{wte}$ by (metis wtg-inv wte-inv e-not-g)

have $\text{if-g-onAxis}: \text{onAxisT } \text{ wtg} \rightarrow \text{collinear } \text{ wte } \text{ wtg } \text{ wtf}$
by \(\text{metis lemAxisIsLine wvte-onAxis wvt-e-not-f wvt-f-not-g wvt-g-not-e}\)

have \(\text{collinear wvte wvtf} \quad \longrightarrow \quad \text{collinear e g f}\)
  by \(\text{metis AzLines iobm iobk wvte-inv wvtf-inv wvtg-inv}\)

hence \(\text{onAxisT wvtf} \quad \longrightarrow \quad \text{collinear e g f}\)
  by \(\text{metis if-g-onAxis}\)

hence \(\text{wvtg-offAxis}: \neg (\text{onAxisT wvtg})\)
  by \(\text{metis g-not-collinear}\)

have \(\forall s. (\exists p. (\text{collinear wvte wvtg p} \land \text{space2 p wvtf} = (s \ast s) \ast \text{time2 p wvtf}))\)
  by \(\text{metis AxSlopedLineInVerticalPlane wvte-onAxis wvt-onAxis wvtg-offAxis wvt-e-not-f}\)

hence exists-wvtz: \(\exists p. (\text{collinear wvte wvtg p} \land \text{space2 p wvtf} = (c k \ast c k) \ast \text{time2 p wvtf})\)
  by \(\text{metis}\)

then obtain \(wvtz\) where
  \(wvtz\)-props: \(\text{collinear wvte wvtf wvtz} \land \text{space2 wvtz wvtf} = (c k \ast c k) \ast \text{time2 wvtz wvtf}\)
  by \(\text{auto}\)

def \(z \equiv \text{wvt m k wvtz}\)
def \(\text{wvtzCone} \equiv \text{lightcone k wvtz}\)

have \(\text{wvtz-is-vertex}: \text{wvtz} = \text{vertex wvtzCone}\)
  by \(\text{simp add: wvtzCone-def}\)

have \(\text{ck-is-slope}: c k = \text{slope wvtzCone}\)
  by \(\text{simp add: wvtzCone-def}\)

hence \(\text{space2 (vertex wvtzCone) wvtf} = ((\text{slope wvtzCone}) \ast \text{time2 wvtz wvtzCone})\)
  by \(\text{metis wvtf-speed wvtz-is-vertex ck-is-slope}\)

hence \(\text{onCone wvtf wvtzCone}\)
  by \(\text{metis onCone.simps}\)

hence \(\text{wvtf-on-wvtzCone}: \text{onCone (wvt m k wvtf)} (\text{lightcone m z})\)
  by \(\text{metis iobm iobk AxCones wvtzCone-def z-def}\)

def \(z\text{Cone} \equiv \text{lightcone m z}\)

have \(\text{z-is-vertex}: z = \text{vertex z\text{Cone}}\)
  by \(\text{simp add: z\text{Cone-def}}\)

have \(\text{cm-is-zSlope}: c m = \text{slope z\text{Cone}}\)
  by \(\text{simp add: z\text{Cone-def}}\)

have \(f\text{-on-zCone}: \text{onCone f z\text{Cone}}\)
  by \(\text{metis wvtf-inv wvtf-on-wvtzCone z\text{Cone-def}}\)
\[
\text{hence } \text{space2 (vertex } z\text{Cone) } f = (\text{slope } z\text{Cone} \ast \text{slope } z\text{Cone})\ast \text{time2 (vertex } z\text{Cone) } f
\]

by \(\text{simp add: } z\text{Cone-def}\)

\[
\text{hence } \text{space2 } f \ z = (c \ m \ast c \ m)\ast \text{time2 } f \ z \text{ by (metis } z\text{-is-vertex cm-is-zSlope)}
\]

\[
\text{hence } f\text{-speed: } \text{space2 } f \ z = (c \ m \ast c \ m)\ast \text{time2 } f \ z \text{ by (metis lemSpace2Sym lemTime2Sym)}
\]

\[
\text{def } f\text{Cone } \equiv \text{ lightcone } m \ f
\]

\[
\text{have } f\text{-is-fVertex: } f = \text{ vertex } f\text{Cone } \text{ by (simp add: } f\text{Cone-def)}
\]

\[
\text{have } cm\text{-is-fSlope: } c \ m = \text{ slope } f\text{Cone } \text{ by (simp add: } f\text{Cone-def)}
\]

\[
\text{hence } \text{space2 (vertex } f\text{Cone) } z = ((\text{slope } f\text{Cone}) \ast (\text{slope } f\text{Cone}))\ast \text{time2 (vertex } f\text{Cone) } z
\]

by \(\text{metis } f\text{-z-speed } f\text{-is-fVertex cm-is-fSlope)}

\[
\text{hence } z\text{-on-fCone: } \text{onCone } z \ f\text{Cone } \text{ by (metis onCone simp z-on-fCone)}
\]

\[
\text{have } \text{collinear wvtg wvtg wvtz } \text{ by (metis wvtz-props)}
\]

\[
\text{hence } \text{egz-collinear: } \text{collinear } e \ g \ z \text{ by (metis wvtg-inv wvtg-inv z-def AxLines iobm iobk)}
\]

\[
\text{hence } z\text{-geometry: } (\text{inPlane } z \ (\text{tangentPlane } g \ e\text{Cone}) \land \text{onCone } z \ e\text{Cone})
\]

by \(\text{metis } \text{AxConeTangent e-is-vertex g-on-eCone)}

\[
\text{have } z\text{-on-eCone: } \text{onCone } z \ e\text{Cone } \text{ by (metis } z\text{-geometry)}
\]

\[
\text{have } z\text{-in-tplane: } \text{inPlane } z \ t\text{plane } \text{ by (metis } z\text{-geometry tplane-def)}
\]

\[
\text{hence } z\text{-not-f: } z \neq f \text{ by (metis } z\text{-on-eCone outside lemOutsideNotOnCone)}
\]

\[
\text{hence } z\text{-not-fVertex: } z \neq \text{ vertex } f\text{Cone } \text{ by (simp add: } f\text{Cone-def } z\text{-not-f)}
\]

\[
\text{def } \text{lineA } \equiv \text{ lineJoining } e \ z
\]

\[
\text{def } \text{lineB } \equiv \text{ lineJoining } f \ z
\]

\[
\text{have } \text{collinear wvtg wvtg wvtz } \text{ by (metis wvtz-props)}
\]

\[
\text{hence } \text{egz-collinear: } \text{collinear } e \ g \ z \text{ by (metis wvtg-inv wvtg-inv z-def AxLines iobm iobk)}
\]

\[
\text{hence } z\text{-geometry: } (\text{inPlane } z \ (\text{tangentPlane } g \ e\text{Cone}) \land \text{onCone } z \ e\text{Cone})
\]

by \(\text{metis } \text{AxConeTangent e-is-vertex g-on-eCone)}

\[
\text{have } z\text{-on-eCone: } \text{onCone } z \ e\text{Cone } \text{ by (metis } z\text{-geometry)}
\]

\[
\text{have } z\text{-in-tplane: } \text{inPlane } z \ t\text{plane } \text{ by (metis } z\text{-geometry tplane-def)}
\]

\[
\text{hence } z\text{-not-f: } z \neq f \text{ by (metis } z\text{-on-eCone outside lemOutsideNotOnCone)}
\]

\[
\text{hence } z\text{-not-fVertex: } z \neq \text{ vertex } f\text{Cone } \text{ by (simp add: } f\text{Cone-def } z\text{-not-f)}
\]

\[
\text{def } \text{lineA } \equiv \text{ lineJoining } e \ z
\]

\[
\text{def } \text{lineB } \equiv \text{ lineJoining } f \ z
\]

\[
\text{assume } \text{assm: } z = e
\]

\[
\text{have } \text{space2 } f \ e = (c \ m \ast c \ m)\ast \text{time2 } f \ e \land \text{space2 } f \ e = \text{space2 } e \ f \land \text{time2 } f \ e = \text{time2 } e \ f
\]

by \(\text{metis } \text{lemSpace2Sym lemTime2Sym fz-speed } \text{assm)}

\[
\text{hence } \text{space2 } e \ f = (c \ m \ast c \ m)\ast \text{time2 } e \ f \text{ by metis}
\]

\[
\text{hence } \text{False } \text{ by (metis } \text{less-irrefl converse)}
\]

\}

\text{from this have } z\text{-not-e: } z \neq e \text{ by blast}

\[
\text{def } \text{lineA } \equiv \text{ lineJoining } e \ z
\]

\[
\text{def } \text{lineB } \equiv \text{ lineJoining } f \ z
\]

\[
\text{assume } \text{assm: } \text{direction } \text{lineA } = \text{ vecZero}
\]

27
have lemnullline: (direction lineA = vecZero ∧ inLine z lineA) → z = basepoint lineA
  by (metis lemNullLine)
have inLine z lineA by (metis lineA-def lemLineContainsEndpoint)
  hence z-is-bp: z = basepoint lineA by (metis lemnullline assum)
have basepoint lineA = e by (simp add: lineA-def)
  hence False by (metis z-is-bp z-not-e)
}
from this have ez-not-null: direction lineA ≠ vecZero by blast

{
  assume assum: direction lineB = vecZero
  have lemnullline: (direction lineB = vecZero ∧ inLine z lineB) → z = basepoint lineB
    by (metis lemNullLine)
  have inLine z lineB by (metis lineB-def lemLineContainsEndpoint)
    hence z-is-bp: z = basepoint lineB by (metis lemnullline assum)
  have basepoint lineB = f by (simp add: lineB-def)
    hence False by (metis z-is-bp z-not-f)
}
from this have fz-not-null: direction lineB ≠ vecZero by blast

{
  have samePlane tplane (tangentPlane z fCone)
    ∧ ((lineJoining e g) || (lineJoining f z))
    by (metis AxParallelCones tplane-def
      g-on-eCone g-not-vertex z-on-fCone z-not-fVertex z-in-tplane
      e-is-vertex f-is-fVertex)
  hence eg-par-fz: (lineJoining e g) || (lineJoining f z) by metis
  {
    assume case1: direction (lineJoining e g) = vecZero
    have direction (lineJoining e g) = from e to g by simp
      hence from e to g = vecZero by (metis case1)
    hence e = g by (simp)
      hence False by (metis e-not-g)
  }
from this have eg-not-null: ¬(direction (lineJoining e g) = vecZero) by blast
then obtain a where a-props: a ≠ 0 ∧ direction (lineJoining f z) = a**direction (lineJoining e g)
  by (metis fz-not-null eg-not-null eg-par-fz parallel.simps lineB-def)
  hence f-to-z: from f to z = a**(from e to g) by simp
  have a-nonzero: a ≠ 0 by (metis a-props)
  have eg-dir: from e to g = direction (lineJoining e g) by simp
  have gz-dir: from g to z = direction (lineJoining g z) by simp
  have egz: z = g ↔ (from g to z) by (metis lemLineEndpoint)
    hence collinear e g (g ↔ (from g to z)) by (metis egz-collinear)
then obtain b where e-to-g: from e to g = (−b)**(from g to z)
by (metis lemDirectionColinear)

{  
assume assm: \(-b = 0\)  
have from e to g = \((-b)***\)\(\text{from g to z}\) by (metis e-to-g)  
hence from e to g = vecZero by (simp add: assm)  
hence direction (line.Joining e g) = vecZero by (simp)  
hence False by (metis eg-not-null lineA-def)  
}  
from this have b-nonzero: \(-b \neq 0\) by blast

def binv \equiv inverse \((-b)\)  
def factor \equiv 1 + binv  
have binv-nonzero: binv \neq 0 by (metis b-nonzero add.comm-neutral binv-def nonzero-imp-inverse-nonzero right-minus)

have from e to g = \((-b)***\)\(\text{from g to z}\) by (metis e-to-g)  
hence g-to-z: \(\text{from g to z}\) = binv**(from e to g)  
  by (metis b-nonzero lemScaleInverse binv-def)

have from e to z = from e to g \oplus from g to z  
  by (simp add: lemZEG)

  hence from e to z = (from e to g) \oplus binv**(from e to g) by (metis g-to-z)
  hence e-to-z: \(\text{from e to z}\) = factor**(from e to g) by (metis lemAddOverScale lemScale1 factor-def)
  have ez-dir: direction (line.Joining e z) = from e to z by simp
  have eg-dir: direction (line.Joining e g) = from e to g by simp

{  
assume assm: factor = 0  
have from e to z = factor**(from e to g) by (metis e-to-z)
  hence from e to z = vecZero by (simp add: assm)
  hence direction (line.Joining e z) = vecZero by (simp)
  hence False by (metis ez-not-null lineA-def)  
}  
from this have factor-nonzero: factor \neq 0 by blast

have direction (line.Joining e z) = factor**(direction (line.Joining e g))  
  by (metis e-to-z ez-dir eg-dir)
  hence (line.Joining e g) || (line.Joining e z) by (metis parallel.simps factor-nonzero)
  hence (line.Joining e z) || (line.Joining e g) by (metis lemParallelSym)

  hence (line.Joining e z) || (line.Joining f z) by (metis lemParallelTrans eg-par-fz eg-not-null)  
}
from this have A-par-B: lineA || lineB by (metis lineA-def lineB-def)

have e-in-lineA: inLine e lineA by (metis lineA-def lemLineContainsBasepoint)

{ have basic: ∀ a b.((−a)*b)*((−a)*b) = (a*a)*(b*b)) 
  by (metis equation-minus-iff minus-mult-commute minus-mult-right
      semiring-normalization-rules(17) semiring-normalization-rules(19))

assume assm: inLine e lineB
hence coll: collinear e f (f ~ direction lineB) by (simp add: lineB-def)
then obtain β where props: from e to f = (−β)**(direction lineB)
  by (metis lemDirectionCollinear)

hence tval f − tval e = (−β)*(tval z − tval f) ∧ xval f − xval e = (−β)*(xval z − xval f)
  ∧ yval f − yval e = (−β)*(yval z − yval f) ∧ zval f − zval e = (−β)*(zval z − zval f)
  by (simp add: basic)

hence speeds: time2 f e = (β*β)*time2 z f ∧ space2 f e = (β*β)*space2 z f
  apply (simp add: basic) apply auto
  apply (metis semiring-normalization-rules(18) semiring-normalization-rules(19))
  by (metis semiring-normalization-rules(18) semiring-normalization-rules(19))

  semiring-normalization-rules(34))

have space2 f z = (c m * c m)*time2 f z by (metis fz-speed)
  hence space2 z f = (c m * c m)*time2 z f 
  by (metis lemSpace2Sym lem-Time2Sym)

have space2 f e = ((β*β)*c m * c m)*time2 z f by (metis speeds mult.assoc)
  hence space2 f e = (c m * c m)*β*time2 z f 
  by (metis mult.assoc mult.commute)

have space2 f e = (c m * c m)*time2 f e by (metis mult.assoc speeds)
  hence space2 e f = (c m * c m)*time2 e f 
  by (metis lemSpace2Sym lem-Time2Sym)

hence False by (metis less-irrefl converse)
}
from this have e-not-in-lineB: ~(inLine e lineB) by blast

have inLine z lineA ∧ inLine z lineB by (metis lemLineContainsEndpoint lineA-def lineB-def)
  hence A-meets-B: meets lineA lineB by auto

}
\textbf{from this have} \( \neg (\text{space2 } e f > (c \ m * c \ m) * \text{time2 } e f) \) \textbf{by blast}

\textbf{thus} \( ?\text{thesis by simp} \)
\texttt{qed}

\texttt{end}

\texttt{end}

\textbf{References}