No Faster-Than-Light Observers

Mike Stannett

April 20, 2020

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 =
  pbasepoint :: '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 = ( \text{tdir} = k \ast \text{tdir} u, \text{xdir} = k \ast \text{xdir} u, \text{ydir} = k \ast \text{ydir} u, \text{zdir} = k \ast \text{zdir} u )

fun orthogonal :: 'a Vector => 'a Vector => bool (infix \perp) where
  orthogonal u v = (u dot v = 0)

lemma lemVecZeroMinus:
  shows 0 \ominus u = \sim u
  by simp

lemma lemVecSelfMinus:
  shows u \ominus u = 0
  by simp

lemma lemVecPlusCommut:
  shows u \oplus v = v \oplus u
  by (simp add: add.commute)

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

lemma lemVecPlusMinus:
  shows u \oplus (\sim v) = u \ominus v
  by (simp add: local.add-uminus-conv-diff)

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

lemma lemMDotCommut:
  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 $1 \times^* u = u$
  by simp

lemma lemScale0:
  shows $0 \times^* u = 0$
  by simp

lemma lemScaleNeg:
  shows $(-k) \times^* u = \sim (k \times^* u)$
  by simp

lemma lemScaleOrigin:
  shows $k \times^* 0 = 0$
  by auto

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

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

lemma lemScaleInverse:
  assumes $k \neq (0::'a)$
  and $v = k \times^* u$
  shows $u = (\text{inverse } k) \times^* v$

proof
  have $(\text{inverse } k) \times^* v = (\text{inverse } k \times k \times^* u)$
    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.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 (from - 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 (infixr ≲ 100) where
  before p q = (tval p < tval q)

fun after :: 'a Point ⇒ 'a Point ⇒ bool (infixr ≳ 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 valq valr. (valq - valp + (valr - valq) = valr - valp)
    by (metis add-uminus-conv-diff add-diff-cancel
        semiring-normalization-rules(24) semiring-normalization-rules(25))
  thus \{thesis by auto
qed
lemma lemMoveByMove:
  shows \( p \leadsto u \leadsto v = p \leadsto (u \oplus v) \)
  by (simp add: add.assoc)

lemma lemScaleLinear:
  shows \( p \leadsto a \ast^* u \leadsto b \ast^* v = p \leadsto (a \ast^* u \oplus b \ast^* v) \)
  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)\ast(xval u - xval v) + (yval u - yval v)\ast(yval u - yval v) + (zval u - zval v)\ast(zval u - zval v)

fun time2 :: '(a Point) ⇒ (′a Point) ⇒ 'a
where
  time2 u v
  = (tval u - tval v)\ast(tval u - tval v)

fun speed :: '(a Point) ⇒ (′a Point) ⇒ 'a
where
  speed u v
  = (space2 u v \div\, 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\ast direction lineA))

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

fun inLine :: 'a Point ⇒ 'a Line ⇒ bool
where
  inLine p l = collinear p (basepoint l) (basepoint l \→ direction l)

end
fun meets :: 'a Line ⇒ 'a Line ⇒ bool where
meets line1 line2 = (∃p.(inLine p line1 ∧ inLine p line2))

lemma lemParallelReflexive:
  shows lineA || lineA
proof –
  define dir where 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)
   define kab where kab ≡ (SOME kab.(kab ≠ (0::'a) ∧ direction lineB =
     kab**direction lineA))
   have kab-props: kab ≠ 0 ∧ direction lineB = kab**direction lineA
     using exists-kab kab-def
     by (rule Hilbert-Choice.exE-some)
   define kba where kba = inverse kab
   have kba-nonzero: kba ≠ 0 by (metis inverse-zero-imp-zero kab-props kba-def)
   have direction lineA = kba**direction lineB by (metis kba-def lemScaleInverse
     kab-props)
   hence ?thesis by (metis kba-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

define kac where 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 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 (in −) 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-conv-diff diff-add-cancel semiring-normalization-rules(24))
  thus thesis by auto
qed
lemma lemDirectionFromTo:
  shows direction (line joining p to (p ⇝ dir)) = dir 
proof -
  have direction (line joining p to (p ⇝ dir)) = from p to (p ⇝ dir) by simp
  thus ?thesis by (metis lemDirectionJoining)
qed

lemma lemLineEndpoint:
  shows q = p ⇝ (from p to q)
proof -
  have ∀ 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 -
  define bp where bp = basepoint lineA
  have bp: basepoint lineA = 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 lemLineEndpoint)
  have (1 + 0 = 1) ∧ (positionVector p = 1**(positionVector bp) ⊕ 0**(positionVector bp))
    by (metis lemScale1 lemAddOver-Scale)
  thus ?thesis by (simp add: bp-def)
qed

lemma lemLineContainsBasepoint:
  shows inLine p (line joining p to q)
proof -
  define linePQ where 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 lemLineEndpoint)
  have (1 + 0 = 1) ∧ (positionVector p = 1**(positionVector bp) ⊕ 0**(positionVector bp))
    by (metis lemScale1 lemAddOver-Scale)
  thus ?thesis by (simp add: bp-def)
qed
lemma lemLineContainsEndpoint:
  shows inLine q (line joining p to q)
proof –
  define linePQ where 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) ∧ (positionVector q = 0** (positionVector p) ⊕ 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 ∥ line joining q to r
  shows line joining p to q ∥ line joining p to r
proof –
  define linePQ where linePQ = line joining p to q
  define lineQR where lineQR = line joining q to r
  define linePR where linePR = line joining p to r

  have case1: (direction linePQ = vecZero) —> ?thesis by (simp add: linePQ-def)
  have case1: (direction linePR = vecZero) —> ?thesis by (simp add: linePR-def)

  { assume case3: direction linePQ ≠ vecZero ∧ direction linePR ≠ vecZero
    { assume case3a: direction lineQR = vecZero
      
      qed
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

define dirPQ where dirPQ = from p to q
have dir-pq: direction linePQ = dirPQ by (simp add: linePQ-def dirPQ-def)

define dirQR where dirQR = from q to r
have dir-qr: direction lineQR = dirQR by (simp add: lineQR-def dirQR-def)

have exists-k: ∃k. (k ≠ 0 ∧ direction lineQR = k**direction linePQ)
  by (metis linePQ-def lineQR-def assms(1) parallel.simps case3b case3)
then obtain k where k-props: k ≠ 0 ∧ dirQR = k**dirPQ by (metis dir-pq dir-qr)

define scalar where scalar = 1+k

have q = p ↦ dirPQ ∧ r = q ↦ dirQR by (metis lemLineEndpoint
dirPQ-def dirQR-def)
hence r = p ↦ dirPQ ↦ (k**dirPQ) by (metis k-props)
hence scalarPR: r = p ↦ scalar**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 ↦ scalar**dirPQ)
  by (simp add: linePR-def scalarPR)
  hence direction linePR = scalar**dirPQ by (metis lemDirectionFromTo)

  hence scalar.props: scalar ≠ 0 ∧ direction linePR = scalar**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

11
hence \( \text{thesis by (metis rtp3a)} \)

from this have \((\text{direction linePQ} \neq \text{vecZero} \wedge \text{direction linePR} \neq \text{vecZero})\) \(
\longrightarrow \text{thesis by blast} \)

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

qed

lemma \(\text{lemDirectionCollinear} : \)
shows \(\text{collinear } u \; v \; (v \rightarrow d) \iff (\exists \beta. (\text{from } u \; to \; v = (-\beta)**d))\)

proof –

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

have \(\text{basic2} : \forall \; u \; v \; w. (u = v \oplus w \longrightarrow v \ominus u = \text{vecNegate} \; w) \; \text{by apply auto} \)

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

have \(\text{basic4} : \forall \; u \; v \; w. (v \ominus u = \text{vecNegate} \; w \longrightarrow u = v \oplus w) \; \text{by apply auto} \)

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

have \(\exists \alpha \; \beta. \; (\alpha + \beta = 1) \wedge \text{positionVector} \; u = \alpha**(\text{positionVector} \; v) \oplus \beta**(\text{positionVector} \; (v \rightarrow d)) \)

by (metis \text{assm} \; \text{collinear}.simps)

then obtain \(\alpha \; \beta \) where \(\text{props} : (\alpha + \beta = 1) \wedge \text{positionVector} \; u = \alpha**(\text{positionVector} \; v) \oplus \beta**(\text{positionVector} \; (v \rightarrow d)) \)

by auto

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

by (metis \text{basic1} \; \text{lemScaleOverAdd} \; \text{lemVecPlusAssoc} \; \text{lemAddOverScale} \; \text{props})

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

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

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

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

by blast

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

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

define \(\alpha \) where \(\alpha = 1 - \beta \)

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

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

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

hence \(\text{positionVector} \; u = \text{positionVector} \; v \oplus \beta**d \) by (metis \text{basic4})
hence $\text{positionVector } u = 1**(\text{positionVector } v) \oplus \beta**d$
   by (metis lemScale1)

hence $(\alpha + \beta = 1) \land$
   \begin{align*}
   \text{positionVector } u &= \alpha**(\text{positionVector } v) \oplus \beta**(\text{positionVector } (v \rightsquigarrow d)) \\
   \text{by (metis } \alpha\beta\text{-sam basic1 lemScaleOverAdd lemVecPlusAssoc lemAddOverScale)}
   \end{align*}

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

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

thus ?thesis by (metis fwd)

qed

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

proof -

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

define bpA where bpA = basepoint lineA
define dirA where dirA = direction lineA
define bpB where bpB = basepoint lineB
define dirB where dirB = direction lineB

have lineB \parallel lineA by (metis lemParallelSym assms(1))

hence exists-kab: $\exists \text{kab: } (\text{kab} \neq (0::'a) \land \text{direction } lineA = \text{kab**direction } lineB)$
   by (metis parallel.simps assms(2) assms(3))

then obtain kab where kab-props: $\text{kab} \neq 0 \land \text{dirA} = \text{kab**dirB}$ by (metis dirA-def dirB-def)

have collinear x bpA (bpA \rightsquigarrow dirA) by (metis assms(4) inLine.simps bpA-def dirA-def)
   then obtain $\beta$ where from x to bpA = $(-\beta)**dirA$ by (metis lemDirectionCollinear)
    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 \( p\text{-in-AB}: \text{inLine } p \text{ lineA} \land \text{inLine } p \text{ lineB} \) by auto

 have collinear \( p \) bpA (bpA \sim \text{dirA}) by (metis p-in-AB inLine劾 bpA-def dirA-def)
 then obtain \( \beta A \) where from \( p \) to \( bpA = (\beta A)^**\text{dirA} \) by (metis lemDirectionCollinear)
 hence from \( bpA \) to \( p = (\beta A)^**\text{dirA} \) by (metis basic minus-minus)
 hence bpA-to-p: from \( bpA \) to \( p = (\beta A*kab)^**\text{dirB} \) by (metis lemScaleScale kab-props)

 have collinear \( p \) bpB (bpB \sim \text{dirB}) by (metis p-in-AB inLine劾 bpB-def dirB-def)
 then obtain \( \beta B \) where p-to-bpB: from \( p \) to \( bpB = (\beta B)^**\text{dirB} \) by (metis lemDirectionCollinear)

 define \( \gamma \) where \( \gamma = -(\beta A*kab + (\beta A*kab) + (\beta B)) \)
 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 \( bpB = ((\beta A*kab)^**\text{dirB} \oplus (\beta A*kab)^**\text{dirB} \oplus (\beta B)^**\text{dirB} \)
 by (metis x-to-bpA bpA-to-p p-to-bpB)
 hence from \( x \) to \( bpB = (\gamma)^**\text{dirB} \)
 by (metis lemAddOverScale add.assoc \gamma-def minus-minus)
 hence collinear \( x \) bpB (bpB \sim \text{dirB}) by (metis lemDirectionCollinear)
 hence inLine \( x \) lineB by (metis inLine劾 bpB-def dirB-def)
}

 from this have meets lineA lineB \rightarrow inLine \( x \) lineB by blast
 thus \$thesis \ by (metis assms(5))
 qed

 lemma lemAxisIsLine:
 assumes onAxisT \( x \)
 and onAxisT \( y \)
 and onAxisT \( z \)
 and \( x \neq y \)
 and \( y \neq z \)
 and \( z \neq x \)
 shows collinear \( x \) \( y \) \( z \)
 proof =
 define ratio where ratio = \( -(\text{tval } y - \text{tval } x) / (\text{tval } z - \text{tval } y) \)
 have x-onAxis: xval \( x = 0 \) \land yval \( x = 0 \) \land zval \( x = 0 \) by (metis assms(1) onAxisT.simps)
have \( y\text{-onAxis} \): \( xval \ y = 0 \land yval \ y = 0 \land zval \ y = 0 \) by (metis assms(2) onAxisT.simps)

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

have \( tval \ z - tval \ y = 0 \rightarrow z = y \) by (simp add: onAxis y-onAxis)

hence \( tval \ z = tval \ y \rightarrow \) by (metis assms(5) eq_iff_diff_eq_0)
	hence \( tvalyz\text{-nonzero} \): \( tval \ z - tval \ y \neq 0 \) by (metis eq_iff_diff_eq_0)

have \( x\text{-to}-y \): \( \langle tdir = tval \ y - tval \ x, xdir = 0, ydir = 0, zdir = 0 \rangle \)
by (simp add: x-onAxis y-onAxis)

have \( y\text{-to}-z \): \( \langle tdir = tval \ z - tval \ y, xdir = 0, ydir = 0, zdir = 0 \rangle \)
by (simp add: y-onAxis z-onAxis)

have \( \text{from} \ x \ \text{to} \ y = \langle \rangle \ \\
by (simp add: x-onAxis y-onAxis)

have \( \text{from} \ y \ \text{to} \ z = \langle \rangle \ \\
by (simp add: y-onAxis z-onAxis)

have \( \text{collinear} \ x \ y \rightarrow \) by (metis lemDirectionCollinear)

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

qed

lemma lemSpace2Sym:
shows \( \text{space2} \ x \ y = \text{space2} \ y \ x \)

proof -
  define \( xsep \) where \( xsep = xval \ x - xval \ y \)
  define \( ysep \) where \( ysep = yval \ x - yval \ y \)
  define \( zsep \) where \( zsep = zval \ x - zval \ y \)

  have \( \text{spacexy} \): \( \text{space2} \ x \ y = (xsep \times xsep) + (ysep \times ysep) + (zsep \times zsep) \)
  by (simp add: xsep-def ysep-def zsep-def)

  have \( \text{spaceyx} \): \( \text{space2} \ y \ x = (-xsep \times -xsep) + (-ysep \times -ysep) + (-zsep \times -zsep) \)
  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 -
  define \( tsep \) where \( tsep = tval \ x - tval \ y \)

  have \( \text{timexy} \): \( \text{time2} \ x \ y = tsep \times tsep \)
  by (simp add: tsep-def)

  have \( \text{timeyx} \): \( \text{time2} \ y \ x = (-tsep) \times (-tsep) \)
  by (simp add: tsep-def)

  thus \( ?\text{thesis} \) by (metis timexy diff-0_right minus_diff_eq minus_mult_left minus_mult_right)

qed
end

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

  fun coplanar :: 'a Point ⇒ 'a Point ⇒ 'a Point ⇒ 'a Point ⇒ bool  where
    coplanar e x y z = (∃\( α, β, γ. \) (\( α + β + γ = 1 \)) ∧
    \( \text{positionVector } e = (α∗∗\text{positionVector } x) \oplus β∗∗\text{positionVector } y \oplus γ∗∗\text{positionVector } z \))

  fun inPlane :: 'a Point ⇒ 'a Plane ⇒ bool  where
    inPlane e pl = coplanar e (\( \text{pbasepoint } pl \) \( \text{pbasepoint pl} \)) (\( \text{pbasepoint pl} \))
    (\( \text{pbasepoint pl} \))

  fun samePlane :: 'a Plane ⇒ 'a Plane ⇒ bool  where
    \( \text{samePlane } pl pl' = \) inPlane (\( \text{pbasepoint } pl \)) \( pl' \) \( \)∧.

lemma lemPlaneContainsBasePoint:
  shows inPlane (\( \text{pbasepoint } pl \)) pl
  proof −
    define \( α \) where \( α = (1::'a) \)
    define \( β \) where \( β = (0::'a) \)
    define \( γ \) where \( γ = (0::'a) \)
    have \( \text{rtp1: } α + β + γ = 1 \) by (simp add: α-def β-def γ-def)

    define \( e \) where \( e = \text{pbasepoint } pl \)
    define \( x \) where \( x = \text{pbasepoint } pl \)
    define \( y \) where \( y = \text{pbasepoint } pl \) \( \text{pbasepoint pl} \)
    define \( z \) where \( z = \text{pbasepoint } pl \) \( \text{pbasepoint pl} \)
    have \( \text{rtp2: } \text{positionVector } e = α∗∗\text{positionVector } x \oplus β∗∗\text{positionVector } y \oplus γ∗∗\text{positionVector } z \)
      by (simp add: e-def x-def y-def z-def β-def γ-def)

    have \( \text{sameplane: } \)coplanar \( e \) \( x \) \( y \) \( z \) by (metis coplanar.simps rtp1 rtp2)
    hence \( \text{coplanar } e \) \( (\text{pbasepoint } pl) \) \( \text{pbasepoint pl} \)
      \( \text{pbasepoint pl} \) \( \text{pbasepoint pl} \) \( \text{pbasepoint pl} \)
    by (simp add: x-def y-def z-def)
    hence \( \text{inPlane } e \) \( pl \) by simp
    thus \( ?\text{thesis} \) by (simp add: e-def)
  qed
class Cones = Quantities + Lines + Planes +
fixes

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

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

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

AxSlopedLineInVerticalPlane: [onAxisT e; onAxisT f; e ≠ f; ¬(onAxisT g)]
⇒ (∀ s.( ∃ p . (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)

end
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 :: 'a Point ⇒ 'a ⇒ 'a Cone where
mkCone v s = ()

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

lemma lemOutsideNotOnCone:
  assumes outsideCone f cone
  shows ¬ (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 ∃ y . P x y
  and f = someFunc P
  shows P x (f x)
proof –
  have f x = (SOME y. (P x y))
  using assms(2) by simp
  thus ?thesis using assms(1)
  by (simp add: someI-ex)
qed

end

theory Axioms
  imports SpaceTime SomeFunc

18
begin

record Body =
    Ph :: bool
    IOb :: bool

class WorldView = SpaceTime +
fixes
  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 \( r \geq 0 \land r \ast r = x \)

proof –

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

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

thus \(?thesis \) using assms(2) by simp

qed

end

class AxLight = WorldView +

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

begin

abbreviation \( c :: \text{Body} \Rightarrow 'a \) where
\( c \equiv \text{someFunc} \, cTest \)

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 \( (v > 0) \land (\forall x \, y. (\exists p. (\text{Ph} \, p \land W \, m \, p \, x \land W \, m \, p \, y) \leftrightarrow (space2 \, x \, y = (c \, m \ast c \, m) \ast time2 \, x \, y))) \)

proof –

have vExists: \( \exists v. \, cTest \, m \, v \) by (metis AxPh assms(1))

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

thus \(?thesis \) using assms(2) by simp

qed

lemma lemCCone:

20
assumes IOb m
and onCone y (lightcone m x)
sshows \( \exists p. (\text{Ph } p \land W m \ p \ x \land W m \ p \ y) \)

proof

have \((\exists p. (\text{Ph } p \land W m \ p \ x \land W m \ p \ y))\)
  \(\text{by (smt assms(1) lemCProps)}\)
hence \(\text{ph-exists: (space2 } x \ y = (c \ m \ast c \ m) \ast \text{time2 } x \ y \longrightarrow (\exists p. (\text{Ph } p \land W m \ p \ x \land W m \ p \ y))}\)
  \(\text{by metis}\)

define lcmx where \( \text{lcmx} = \text{lightcone } m \ x \)

have \(\text{lcmx-vertex: vertex } \text{lcmx} = x\) by (simp add: lcmx-def)

have \(\text{lcmx-slope: slope } \text{lcmx} = c \ m\) by (simp add: lcmx-def)

have \(\text{onCone } y \ \text{lcmx} \longrightarrow (\text{space2 } x \ y = (c \ m \ast c \ m) \ast \text{time2 } x \ y)\)
  \(\text{by (metis lcmx-vertex lcmx-slope onCone.simps)}\)
hence \(\text{space2 } x \ y = (c \ m \ast c \ m) \ast \text{time2 } x \ y\) by (metis lcmx-def assms(2))
thus \(\exists p. (\text{Ph } p \land W m \ p \ x \land W m \ p \ y)\) by (metis ph-exists)

qed


lemma lemCPos:
assumes IOb m
shows \( c \ m > 0 \)
by (metis assms(1) lemCProps)

lemma lemCPhoton:
assumes IOb m
shows \( \forall \ x \ y. (\exists p. (\text{Ph } p \land W m \ p \ x \land W m \ p \ y)) \longleftrightarrow (\text{space2 } x \ y = (c \ m \ast c \ m) \ast (\text{time2 } x \ y)) \)
by (metis assms(1) lemCProps)

end

class AxEv = WorldView +
assumes
  AxEv: \[ IOb m; IOb k \] \(\Longrightarrow (\exists y. (\forall b. (W m b \ x \longleftrightarrow W k b \ y)))\)
begnin
dend

class AxThExp = WorldView + AxPh +
assumes
  AxThExp: IOb m \(\Longrightarrow (\forall x \ y .\{\}

21
\[ (\exists k. (IOb k \land W m k x \land W m k y)) \leftrightarrow (\text{space2} x y < (c m \ast c m) \ast \text{time2} x y) \]

begin
end

class AxSelf = WorldView +
assumes
AxSelf: IOb m \implies (W m m x) \rightarrow (\text{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
\text{tval} x = \text{tval} y \land \text{tval} x' = \text{tval} y')
\rightarrow (\text{space2} x y = \text{space2} x' y')
begin
end

class AxLines = WorldView +
assumes
AxLines: [ IOb m; IOb k; \text{collinear} x p q ] \implies
\text{collinear} (\text{wvt} k m x) (\text{wvt} k m p) (\text{wvt} k m q)
begin
end
class \texttt{AxPlanes} = \texttt{WorldView} +
\texttt{assumes}
\texttt{AxPlanes: [ IOb m; IOb k ] \rightarrow}
\texttt{(coplanar e x y z \rightarrow coplanar (wvt k m e) (wvt k m x) (wvt k m y) (wvt k m z))}
\texttt{begin}
\texttt{end}

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

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

\texttt{end}

\texttt{theory SpecRel}
\texttt{imports Axioms}
\texttt{begin}
\texttt{class SpecRel = WorldView + AxPh + AxEv + AxSelf + AxSym}
\texttt{+ AxEuclidean}
\texttt{+ AxLines + AxPlanes + AxCones}

\texttt{begin}
lemma lemZEG:
\[ z - e = g - e + (z - g) \]
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-rules(2) local.semiring-normalization-rules(24) local.semiring-normalization-rules(25))
thus \( \text{thesis} \)
by (simp add: calculation)
qed

lemma noFTLObserver:
assumes \( iobm: IOb m \)
and \( iobk: IOb k \)
and \( mke: m \text{ sees } k \text{ at } e \)
and \( mkf: m \text{ sees } k \text{ at } f \)
and \( enotf: e \neq f \)
shows \( \text{space2 } e \ f \leq (c \ m \times c \ m) \times \text{time2 } e \ f \)
proof
{
assume converse: \( \text{space2 } e \ f > (c \ m \times c \ m) \times \text{time2 } e \ f \)

define eCone where \( eCone = \text{mkCone } e \text{ } (c \ m) \)
have e-on-econe: \( \text{onCone } e \ eCone \) by (simp add: eCone-def)

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

have outsideCone f eCone
\[ \rightarrow (\exists x. (\text{onCone } x \ eCone \land x \neq \text{vertex } eCone \land \text{inPlane } f \ (\text{tangentPlane } x \ eCone))) \]
by (rule AxParallelConesE)
hence tplane-exists: \( \exists x. (\text{onCone } x \ eCone \land x \neq \text{vertex } eCone \land \text{inPlane } f \ (\text{tangentPlane } x \ eCone)) \)
by (metis outside)
then obtain g where g-props: (onCone g eCone ∧ g ≠ vertex eCone ∧ inPlane f (tangentPlane g eCone))
  by auto
have g-on-eCone: onCone g eCone by (metis g-props)
have g-not-vertex: g ≠ vertex eCone by (metis g-props)

define tplane where tplane = tangentPlane g eCone
have e-in-tplane: inPlane e tplane by (metis AxTangentVertex e-is-vertex tplane-def)
have f-in-tplane: inPlane f tplane by (metis g-props tplane-def)
have g-in-tplane: inPlane g tplane by (metis lemPlaneContainsBasePoint tplane-def AxTangentBase)

have (onCone g eCone) →
  ((inPlane f (tangentPlane g eCone) ∧ onCone f eCone)
  → collinear (vertex eCone) g f)
  by (metis AxConeTangent)
hence axconetangent: collinear e g f → onCone f eCone
by (metis g-on-eCone e-is-vertex)
have ¬(onCone f eCone) by (metis outside lemOutsideNotOnCone)
hence g-not-collinear: ¬(collinear e g f)
by (metis axconetangent)

define wvte where wvte = wvt k m e
define wvtf where wvtf = wvt k m f
define wvtg where wvtg = wvt k m g

have W k k wvte by (metis wvte-def AxWVT mke iobm iobk)
hence wvte-onAxis: onAxisT wvte by (metis AxSelf iobk)

have W k k wvtf by (metis wvtf-def AxWVT mkf iobm iobk)
hence wvtf-onAxis: onAxisT wvtf by (metis AxSelf iobk)

have wvte-inv: e = wvt m k wvte by (metis AxWVTSym iobk iobm wvte-def)
have wvtf-inv: f = wvt m k wvtf by (metis AxWVTSym iobk iobm wvtf-def)
have wvtg-inv: g = wvt m k wvtg by (metis AxWVTSym iobk iobm wvtg-def)

have e-not-g: e ≠ g by (metis e-is-vertex g-not-vertex)
have f-not-g: f ≠ g by (metis outside lemOutsideNotOnCone g-on-eCone)

have wvt-e-not-f: wvte ≠ wvtf by (metis wvte-inv wvtf-inv enotf)
have wvt-f-not-g: wvtf ≠ wvtg by (metis wvtf-inv wvtg-inv f-not-g)
have wvt-g-not-e: wvtg ≠ wvte by (metis wvtg-inv wvte-inv e-not-g)
have \( \text{if-g-onAxis} : \text{onAxisT wvtg} \rightarrow \text{collinear wvte wvtg wvtf} \)
by (metis lemAxisIsLine wvte-onAxis wvtf-onAxis wvt-e-not-f wvt-f-not-g wvt-g-not-e)

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

hence \( \text{onAxisT wvtg} \rightarrow \text{collinear e g f} \) by (metis if-g-onAxis)

hence \( \text{wvtg-offAxis} : \neg (\text{onAxisT wvtg}) \) by (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 (metis AxSlopedLineInVerticalPlane wvte-onAxis wvtf-onAxis wvtg-offAxis wvt-e-not-f)

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

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

hence \( \text{wvtz-speed}: \text{space2 wvtz wvtf} = (c k \ast c k) \ast \text{time2 wvtz wvtf} \) by metis

define \( \text{z where} \ z = \text{wet m k wvtz} \)
define \( \text{wvtzCone where} \ wvtzCone = \text{lightcone k wvtz} \)

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

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

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

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

define \( \text{zCone where} \ zCone = \text{lightcone m z} \)
define \( \text{z-is-vertex: z = vertex zCone} \) by (simp add: zCone-def)

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

have \( \text{f-on-zCone: onCone f zCone} \) by (metis wvtz-inv wvtf-on-wvtzCone zCone-def)

26
\begin{verbatim}

  hence  \textit{space2 (vertex zCone)} \( f = (\text{slope zCone} * \text{slope zCone}) * \text{time2 (vertex zCone)} \) 
  \hspace{1em} by (simp add: zCone-def)
  hence  \textit{space2 z f = (c m * c m) * time2 z f} \hspace{1em} by (metis z-is-vertex cm-is-zSlope)
  hence  \textit{fz-speed: space2 f z = (c m * c m) * time2 f z} \hspace{1em} by (metis lemSpace2Sym lemTime2Sym)

  define \textit{fCone where fCone = lightcone m f}

  have  \textit{f-is-fVertex: f = vertex fCone} \hspace{1em} by (simp add: fCone-def)
  have  \textit{cm-is-fSlope: c m = slope fCone} \hspace{1em} by (simp add: fCone-def)
  hence  \textit{space2 (vertex fCone) z = ((slope fCone) * (slope fCone)) * time2 (vertex fCone) z} 
  \hspace{1em} by (metis fz-speed f-is-fVertex cm-is-fSlope)
  hence  \textit{z-on-fCone: onCone z fCone} \hspace{1em} by (metis onCone.simps)

  have  \textit{collinear wvte wvtg wvtz} \hspace{1em} by (metis wvtz-props)
  hence  \textit{egz-collinear: collinear e g z} \hspace{1em} by (metis wvte-inv wvtg-inv z-def AxLines iobm iobk)
  hence  \textit{z-geometry: (inPlane z (tangentPlane g eCone) \& onCone z eCone)} 
  \hspace{1em} by (metis AxConeTangent e-is-vertex g-on-eCone)

  have  \textit{z-on-eCone: onCone z eCone} \hspace{1em} by (metis z-geometry)
  have  \textit{z-in-tplane: inPlane z tplane} \hspace{1em} by (metis z-geometry tplane-def)

  hence  \textit{z-not-f: z \neq f} \hspace{1em} by (metis z-on-eCone outside lemOutsideNotOnCone)
  hence  \textit{z-not-fVertex: z \neq vertex fCone} \hspace{1em} by (simp add: fCone-def z-not-f)

  {\quote
    assume  \textit{assm: z = e}
    have  \textit{space2 f e = (c m * c m) * time2 f e \& space2 f e = space2 e f \& time2 f e = time2 e f} 
    \hspace{1em} by (metis lemSpace2Sym lemTime2Sym fz-speed assm)
    hence  \textit{space2 e f = (c m * c m) * time2 e f} \hspace{1em} by metis
    hence  \textit{False} \hspace{1em} by (metis less-irrefl converse)
  }

  from  \textit{this have z-not-e: z \neq e} \hspace{1em} by blast

  define \textit{lineA where lineA = lineJoining e z}
  define \textit{lineB where lineB = lineJoining f z}
\end{verbatim}
assume \textit{assm}: \text{direction line}A = \text{vec}Zero

have \textit{lemnullline}: (\text{direction line}A = \text{vec}Zero \land \text{inLine} z \text{ line}A) \rightarrow z = \text{basepoint line}A
  by (\text{metis lemNullLine})

have \textit{inLine} z \text{ line}A by (\text{metis line}A-def lemLineContainsEndpoint)

hence \textbf{z-is-bp}: z = \text{basepoint line}A by (\text{metis lemnullline assm})

hence \textbf{False} by (\text{metis z-is-bp z-not-e})

from this have \textbf{ez-not-null}: \textit{direction line}A \neq \text{vec}Zero by blast

assume \textit{assm}: \text{direction line}B = \text{vec}Zero

have \textit{lemnullline}: (\text{direction line}B = \text{vec}Zero \land \text{inLine} z \text{ line}B) \rightarrow z = \text{basepoint line}B
  by (\text{metis lemNullLine})

have \textit{inLine} z \text{ line}B by (\text{metis line}B-def lemLineContainsEndpoint)

hence \textbf{z-is-bp}: z = \text{basepoint line}B by (\text{metis lemnullline assm})

hence \textbf{z-is-bp} by (\text{metis line}B-def lemLineContainsEndpoint)

hence \textbf{False} by (\text{metis z-is-bp z-not-f})

from this have \textbf{fz-not-null}: \textit{direction line}B \neq \text{vec}Zero by blast

have \textit{samePlane} tplane (\text{tangentPlane} z fCone)
  \land ((\text{lineJoining} e \ g) \parallel (\text{lineJoining} f \ z))
  by (\text{metis AxParallelCones tplane-def e-is-vertex f-is-fVertex z-not-fVertex z-in-tplane})

hence \textbf{eg-par-fz}: (\text{lineJoining} e \ g) \parallel (\text{lineJoining} f \ z) by metis

assume \textit{case}1: \text{direction} (\text{lineJoining} e \ g) = \text{vec}Zero

have \textit{direction} (\text{lineJoining} e \ g) = \textit{from e to g} by simp

hence \textit{from e to g} = \text{vec}Zero by (\text{metis case}1)

hence \textbf{e = g} by (simp)

hence \textbf{False} by (\text{metis e-not-g})

from this have \textbf{eg-not-null}: \neg (\text{direction} (\text{lineJoining} e \ g) = \text{vec}Zero) by blast

then obtain \textit{a where} \textit{a-props}: a \neq 0 \land \textit{direction} (\text{lineJoining} f \ z) = a**\textit{direction}

have \textit{a-nonzero}: a \neq 0 by (\text{metis a-props})

have \textit{eg-dir}: \textit{from e to g} = \textit{direction} (\text{lineJoining} e \ g) by simp

have \textit{gz-dir}: \textit{from g to z} = \textit{direction} (\text{lineJoining} g \ z) by simp

have \textit{egz}: z = g \rightsquigarrow (\text{from g to z}) by (\text{metis lemLineEndpoint})

hence \textit{collinear e g (g \rightsquigarrow (\text{from g to z}))} by (\text{metis egz-collinear})
then obtain $b$ where $e$-to-$g$: from $e$ to $g = (-b)**(from \ g \ to \ z)$
by (metis lemDirectionCollinear)

\[
\begin{align*}
\{ & \text{assume } assm: -b = 0 \\
& \text{have } from \ e \ to \ g = (-b)**(from \ g \ to \ z) \text{ by (metis e-to-g)} \\
& \text{hence } from \ e \ to \ g = vecZero \text{ by (simp add: assm)} \\
& \text{hence } direction \ (lineJoining \ e \ g) = vecZero \text{ by (simp)} \\
& \text{hence False by (metis eg-not-null lineA-def)} \\
\}
\]

\text{from this have } b\text{-nonzero: } -b \neq 0 \text{ by blast}

\begin{align*}
& \text{define } binv \text{ where } binv = inverse (-b) \\
& \text{define factor where } factor = 1+binv \\
& \text{have } binv\text{-nonzero: } binv \neq 0 \text{ by (metis b-nonzero add.comm-neutral binv-def nonzero-imp-inverse-nonzero right-minus)} \\
\end{align*}

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

\begin{align*}
& \text{have from } e \ to \ z = from \ e \ to \ g \oplus from \ g \ to \ z \\
& \text{by (simp add: lemZEG)} \\
\end{align*}

\begin{align*}
& \text{hence from } e \ to \ z = (from \ e \ to \ g) \oplus binv**(from \ e \ to \ g) \text{ by (metis g-to-z)} \\
& \text{hence } e\text{-to-}z: \ from \ e \ to \ z = factor**(from \ e \ to \ g) \text{ by (metis lemAddOverScale lemScale1 factor-def)} \\
& \text{have ez-dir: direction } (lineJoining \ e \ z) = from \ e \ to \ z \text{ by simp} \\
& \text{have eg-dir: direction } (lineJoining \ e \ g) = from \ e \ to \ g \text{ by simp} \\
\end{align*}

\begin{align*}
\{ & \text{assume assm: factor = 0} \\
& \text{have from } e \ to \ z = factor**(from \ e \ to \ g) \text{ by (metis e-to-z)} \\
& \text{hence from } e \ to \ z = vecZero \text{ by (simp add: assm)} \\
& \text{hence direction } (lineJoining \ e \ z) = vecZero \text{ by (simp)} \\
& \text{hence False by (metis ez-not-null lineA-def)} \\
\}
\]

\text{from this have } factor\text{-nonzero: } factor \neq 0 \text{ by blast}

\begin{align*}
& \text{have direction } (lineJoining \ e \ z) = factor**(direction \ (lineJoining \ e \ g)) \\
& \text{by (metis e-to-z ez-dir eg-dir)} \\
& \text{hence } (lineJoining \ e \ g) \parallel (lineJoining \ e \ z) \text{ by (metis parallel.simps factor-nonzero)} \\
& \text{hence } (lineJoining \ e \ z) \parallel (lineJoining \ e \ g) \text{ by (metis lemParallelSym)} \\
\end{align*}

\begin{align*}
& \text{hence } (lineJoining \ e \ z) \parallel (lineJoining \ f \ z) \text{ by (metis lemParallelTrans eg-par-fz eg-not-null)} \\
\end{align*}
have e-in-lineA: inLine e lineA by (metis lineA-def lemLineContainsBasepoint)

{ have basic: \( \forall a \ b. ((-a)*b)*((-a)*b) = (a*a)*(b*b) \)
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 \((f \sim direction lineB)\) by (simp add: lineB-def)
then obtain \( \beta \) where props: from e to f \( = (-\beta)**(direction lineB) \)
by (metis lemDirectionCollinear)

then obtain \( \beta \) where props: from e to f = \( (-\beta)**(direction lineB) \)
by (simp add: lineB-def)

hence speeds: time2 f e = \( (\beta*\beta)*time2 z f \) \land space2 f e = \( (\beta*\beta)*space2 z f \)
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))

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

have space2 f e = \( (c \ m \ c \ m)*time2 e f \) 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 \land inLine z lineB by (metis lemLineContainsEndpoint lineA-def lineB-def)
hence A-meets-B: meets lineA lineB by auto


}
from this have \( \neg (\text{space} \ e \ f > (c \ m \ast c \ m) \ast \text{time} \ e \ f) \) by blast

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

end

end

References