No Faster-Than-Light Observers

Mike Stannett

December 14, 2021

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 = (| *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 $1 ** u = u$
by *simp*

lemma *lemScale0*:
shows $0 ** u = 0$
by *simp*

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

lemma *lemScaleOrigin*:
shows $k**0 = 0$
by *auto*

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

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

lemma *lemScaleInverse*:
assumes $k \neq (0::'a)$
and $v = k**u$
shows $u = (inverse\ k)**v$
proof −
have $(inverse\ k)**v = (inverse\ k * k)**u$
by (*simp add*: *lemScaleScale assms*(*2*) *mult.assoc*)
thus *?thesis* **by** (*metis* (*lifting*) *field-inverse assms*(*1*) *lemScale1*)
qed

lemma *lemOrthoSym*:
assumes $u \perp v$
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*: ∀ *valp 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 $p \rightsquigarrow u \rightsquigarrow v = p \rightsquigarrow (u \oplus v)$
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 −
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 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 (**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 *collinear x* (*basepoint lineA*) (*basepoint lineA* ⇝ *direction lineA*)
by (*metis inLine.simps assms*(*2*))
hence *collinear x bp* (*bp* ⇝ *vecZero*) **by** (*metis bp-def assms*(*1*))
hence *collinear x bp bp* **by** *simp*
hence ∃ *a b*.((*a* + *b* = *1*) ∧
(*positionVector x* = *a*∗∗(*positionVector bp*) ⊕ *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 −
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 p*) ⊕ *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 −
  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 lemLi-
neEndpoint)

  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 case2: (direction linePR = vecZero) ⟶ ?thesis by (simp add: linePR-def)

  {
   assume case3: direction linePQ ≠ vecZero ∧ direction linePR ≠ 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

    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

  hence ?thesis by (metis rtp3a)
```

}
from *this* **have** (*direction linePQ* $\neq$ *vecZero* $\wedge$ *direction linePR* $\neq$ *vecZero*) $\longrightarrow$ *?thesis* **by** *blast*

thus *?thesis* **by** (*metis case1 case2*)
qed

lemma *lemDirectionCollinear*:
shows *collinear u v* (*v* $\rightsquigarrow$ *d*) $\longleftrightarrow$ ($\exists\,\beta$.(*from u to v* = ($-\beta$)$**$*d*))
proof −
have *basic1*: $\forall\, u\ v$.(*positionVector* (*u* $\rightsquigarrow$ *v*)) = (*positionVector u*) $\oplus$ *v* **by** *simp*
have *basic2*: $\forall\, u\ v\ w$.(*u* = *v* $\oplus$ *w* $\longrightarrow$ *v* $\ominus$ *u* = *vecNegate w*)
apply *auto*
by (*metis add-uminus-conv-diff diff-add-cancel minus-add semiring-normalization-rules*(*24*)) +
have *basic3*: $\forall\, u\ v$.(*from u to v* = *positionVector v* $\ominus$ *positionVector u*) **by** *simp*
have *basic4*: $\forall\, u\ v\ w$.(*v* $\ominus$ *u* = *vecNegate w* $\longrightarrow$ *u* = *v* $\oplus$ *w*)
apply *auto*
by (*metis add-uminus-conv-diff diff-add-cancel lemScale1 mult.left-neutral semiring-normalization-rules*(*24*) *vecScale.simps*)

{
assume *assm*: *collinear u v* (*v* $\rightsquigarrow$ *d*)
have $\exists\,\alpha\ \beta$. ((α + β = *1*) $\wedge$
positionVector u = $\alpha$$**$(*positionVector v*) $\oplus$ $\beta$$**$(*positionVector* (*v* $\rightsquigarrow$ *d*)))
by (*metis assm collinear.simps*)
then obtain α β **where** *props*: (α + β = *1*) $\wedge$
positionVector u = $\alpha$$**$(*positionVector v*) $\oplus$ $\beta$$**$(*positionVector* (*v* $\rightsquigarrow$ *d*)) **by** *auto*
hence *positionVector u* = *1*$**$(*positionVector v*) $\oplus$ $\beta$$**$*d*
by (*metis basic1 lemScaleOverAdd lemVecPlusAssoc lemAddOverScale props*)
hence *positionVector u* = *positionVector v* $\oplus$ $\beta$$**$*d* **by** (*metis lemScale1*)
hence *positionVector v* $\ominus$ *positionVector u* = ($-\beta$)$**$*d* **by** (*metis basic2 lemScaleNeg*)
hence $\exists\,\beta$.(*from u to v* = ($-\beta$)$**$*d*) **by** (*metis basic3*)
}
from *this* **have** *fwd*: *collinear u v* (*v* $\rightsquigarrow$ *d*) $\longrightarrow$ ($\exists\,\beta$.(*from u to v* = ($-\beta$)$**$*d*)) **by** *blast*

{
assume $\exists\,\beta$.(*from u to v* = ($-\beta$)$**$*d*)
then obtain β **where** *assm*: *from u to v* = ($-\beta$)$**$*d* **by** *auto*
define α **where** α = *1* − β
have $\alpha\beta$*-sum*: α + β = *1* **by** (*simp add*: α*-def*)
have *from u to v* = *vecNegate* ($\beta$$**$*d*) **by** (*metis assm lemScaleNeg*)
hence *positionVector v* $\ominus$ *positionVector u* = *vecNegate* ($\beta$$**$*d*) **by** *auto*
hence *positionVector u* = *positionVector v* $\oplus$ $\beta$$**$*d* **by** (*metis basic4*)
hence *positionVector u* = *1*$**$(*positionVector v*) $\oplus$ $\beta$$**$*d*

by (*metis lemScale1*)
hence $(\alpha + \beta = 1) \ \wedge$
$positionVector\ u = \alpha{**}(positionVector\ v) \oplus \beta{**}(positionVector\ (v \rightsquigarrow d))$
by (*metis αβ-sum basic1 lemScaleOverAdd lemVecPlusAssoc lemAddOver-Scale*)
hence *collinear u v* $(v \rightsquigarrow d)$ **by** *auto*
}
from *this* **have** $(\exists \beta.(from\ u\ to\ v = (-\beta){**}d)) \longrightarrow collinear\ u\ v\ (v \rightsquigarrow d)$ **by** *blast*

thus *?thesis* **by** (*metis fwd*)
qed

lemma *lemParallelNotMeet*:
assumes *lineA* $\parallel$ *lineB*
and *direction lineA* $\neq$ *vecZero*
and *direction lineB* $\neq$ *vecZero*
and *inLine x lineA*
and $\neg$(*inLine x lineB*)
shows $\neg$(*meets lineA lineB*)
proof −

have *basic*: $\forall\, p\ q\ v\ a.(from\ p\ to\ q = a{**}v \longrightarrow from\ q\ to\ p = (-a){**}v)$
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*: $\exists\, kab.(kab \neq (0::'a) \wedge direction\ lineA = kab{**}direction\ lineB)$

by (*metis parallel.simps assms*(*2*) *assms*(*3*))
then obtain *kab* **where** *kab-props*: $kab \neq 0 \wedge dirA = kab{**}dirB$ **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 β **where** $from\ x\ to\ bpA = (-\beta){**}dirA$ **by** (*metis lemDirection-Collinear*)
hence *x-to-bpA*: $from\ x\ to\ bpA = ((-\beta){*}kab){**}dirB$ **by** (*metis lemScaleScale kab-props*)

{

assume *converse*: *meets lineA lineB*
have $\exists\, p.(inLine\ p\ lineA \wedge inLine\ p\ lineB)$ **by** (*metis converse meets.simps*)
then obtain *p* **where** *p-in-AB*: *inLine p lineA* $\wedge$ *inLine p lineB* **by** *auto*

have *collinear p bpA* (*bpA* $\rightsquigarrow$ *dirA*) **by** (*metis p-in-AB inLine.simps bpA-def dirA-def*)
then obtain βA **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 bpA to p* $= (\beta A * kab) ** dirB$ **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 βB **where** *p-to-bpB*: *from p to bpB* $= (-\beta B) ** dirB$ **by** (*metis lemDirectionCollinear*)

define γ **where** $\gamma = -((-\beta) * kab + (\beta A * kab) + (-\beta B))$
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* $= ((-\beta) * kab) ** dirB \oplus (\beta A * kab) ** dirB \oplus (-\beta B) ** dirB$
by (*metis x-to-bpA bpA-to-p p-to-bpB*)
hence *from x to bpB* $= (-\gamma) ** dirB$
by (*metis lemAddOverScale add.assoc* γ*-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 *?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* $= -(tval\ y - tval\ x)\ /\ (tval\ z - tval\ y)$

have *x-onAxis*: *xval x* $= 0 \wedge$ *yval x* $= 0 \wedge$ *zval x* $= 0$ **by** (*metis assms*(*1*) *onAxisT.simps*)
have *y-onAxis*: *xval y* $= 0 \wedge$ *yval y* $= 0 \wedge$ *zval y* $= 0$ **by** (*metis assms*(*2*)

onAxisT.simps)
have *z-onAxis*: *xval z = 0 ∧ yval z = 0 ∧ zval z = 0* **by** (*metis assms*(*3*) *onAxisT.simps*)

have *tval z − tval y = 0 ⟶ z = y* **by** (*simp add*: *z-onAxis y-onAxis*)
hence *tval z ≠ tval y* **by** (*metis assms*(*5*) *eq-iff-diff-eq-0*)
hence *tvalyz-nonzero*: *tval z − tval y ≠ 0* **by** (*metis eq-iff-diff-eq-0*)

have *x-to-y*: *from x to y* = ⦇ *tdir* = *tval y − tval x*, *xdir* = *0*, *ydir* = *0*, *zdir* = *0* ⦈
by (*simp add*: *x-onAxis y-onAxis*)
have *y-to-z*: *from y to z* = ⦇ *tdir* = *tval z − tval y*, *xdir* = *0*, *ydir* = *0*, *zdir* = *0* ⦈
by (*simp add*:*y-onAxis z-onAxis*)

have *from x to y* = (−*ratio*)∗∗(*from y 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 *collinear x y* (*y ⇝* (*from y to z*)) **by** (*metis lemDirectionCollinear*)
thus *?thesis* **by** (*metis lemLineEndpoint*)
qed

lemma *lemSpace2Sym*:
shows *space2 x y = 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 *spacexy*: *space2 x y* = (*xsep*∗*xsep*) + (*ysep*∗*ysep*) + (*zsep*∗*zsep*)
by (*simp add*: *xsep-def ysep-def zsep-def*)
have *spaceyx*: *space2 y x* = (−*xsep*)∗(−*xsep*) + (−*ysep*)∗(−*ysep*) + (−*zsep*)∗(−*zsep*)
by (*simp add*: *xsep-def ysep-def zsep-def*)
thus *?thesis* **by** (*metis spacexy diff-0-right minus-diff-eq minus-mult-left minus-mult-right*)
qed

lemma *lemTime2Sym*:
shows *time2 x y = time2 y x*
proof −
define *tsep* **where** *tsep = tval x − tval y*

have *timexy*: *time2 x y = tsep*∗*tsep*
by (*simp add*: *tsep-def*)
have *timeyx*: *time2 y x* = (−*tsep*)∗(−*tsep*)
by (*simp add*: *tsep-def*)
thus *?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, direction2 = 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)  ∧
           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 −
    define α where α = (1::'a)
    define β where β = (0::'a)
    define γ where γ = (0::'a)
    have rtp1: α + β + γ = 1 by (simp add: α-def β-def γ-def)

    define e where e = pbasepoint pl
    define x where x = pbasepoint pl
    define y where y = pbasepoint pl ⇝ direction1 pl
    define z where 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 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 fcone* ∧ *f* ≠ *vertex fcone*
∧ *inPlane f* (*tangentPlane e econe*))
⟶ (*samePlane* (*tangentPlane e econe*) (*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*)

```
  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 = (| vertex = v, slope = 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*
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 ≥ 0  ∧  r*r = x
  proof −
    have rootExists: (∃ r. sqrtTest x r) by (metis AxEuclidean assms(1))
    hence sqrtTest x (sqrt x) by (metis lemSomeFunc)
    thus ?thesis using assms(2) by simp
  qed

end



class AxLight = WorldView +
assumes
  AxLight: ∃ m v.( IOb m ∧ (v > (0::'a)) ∧ ( ∀ x y.(
            (∃ p.(Ph p ∧ W m p x ∧ W m p y)) ⟷ (space2 x y = (v * v)*time2 x
y)
          )))
begin
end



class AxPh = WorldView + AxiomPreds +
assumes
  AxPh: IOb m ⟹ (∃ v. cTest m v)
begin

  abbreviation c :: Body ⇒ 'a where
    c ≡ someFunc cTest

  fun lightcone :: Body ⇒ 'a Point ⇒ 'a Cone where
    lightcone m v = mkCone v (c m)


lemma lemCProps:
  assumes IOb m
     and  v = c m
  shows (v > 0) ∧ (∀ x y.((∃ p. (Ph p ∧ W m p x ∧ W m p y))
                    ⟷ ( space2 x y = (c m * c m)*time2 x y )))
proof −
  have vExists: (∃ 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*:
assumes *IOb m*
and *onCone y* (*lightcone m x*)
shows $\exists p.\ (Ph\ p \wedge W\ m\ p\ x \wedge W\ m\ p\ y)$
proof −
have $(\exists p.(Ph\ p \wedge W\ m\ p\ x \wedge W\ m\ p\ y)) \longleftrightarrow (\ space2\ x\ y = (c\ m * c\ m) * time2\ x\ y\ )$
by (*smt assms*(*1*) *lemCProps*)
hence *ph-exists*: $(space2\ x\ y = (c\ m * c\ m) * time2\ x\ y) \longrightarrow (\exists p.(Ph\ p \wedge W\ m\ p\ x \wedge W\ m\ p\ y))$
by *metis*
define *lcmx* **where** *lcmx* = *lightcone m x*
have *lcmx-vertex*: *vertex lcmx* = *x* **by** (*simp add*: *lcmx-def*)
have *lcmx-slope*: *slope lcmx* = *c m* **by** (*simp add*: *lcmx-def*)
have *onCone y lcmx* $\longrightarrow$ (*space2 x y* = (*c m* ∗ *c m*)∗*time2 x y*)
by (*metis lcmx-vertex lcmx-slope onCone.simps*)
hence *space2 x y* = (*c m* ∗ *c m*)∗*time2 x y* **by** (*metis lcmx-def assms*(*2*))
thus *?thesis* **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 $\forall x\ y.\ (\exists p.\ (Ph\ p \wedge W\ m\ p\ x \wedge W\ m\ p\ y)) \longleftrightarrow (space2\ x\ y = (c\ m * c\ m) * (time2\ x\ y))$
by (*metis assms*(*1*) *lemCProps*)

end

class *AxEv* = *WorldView* +
assumes
AxEv: $[\![\ IOb\ m;\ IOb\ k\ ]\!] \Longrightarrow\ (\exists y.\ (\forall b.\ (W\ m\ b\ x\ \longleftrightarrow\ W\ k\ b\ y)))$
begin
end

class *AxThExp* = *WorldView* + *AxPh* +
assumes

$AxThExp$: $IOb\ m \Longrightarrow (\forall\ x\ y\ .($
$(\exists\ k.(IOb\ k \wedge\ W\ m\ k\ x \wedge\ W\ m\ k\ y)) \longleftrightarrow (space2\ x\ y < (c\ m * c\ m) * time2\ x\ y)$
$))$

begin
end

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

class $AxC = WorldView + AxPh +$
assumes
AxC: $IOb\ m \Longrightarrow c\ m = 1$
begin
end

class $AxSym = WorldView +$
assumes
$AxSym$: $[\![\ IOb\ m;\ IOb\ k\ ]\!] \Longrightarrow$
$(W\ m\ e\ x \wedge\ W\ m\ f\ y \wedge\ W\ k\ e\ x' \wedge\ W\ k\ f\ y' \wedge$
$tval\ x = tval\ y \wedge tval\ x' = tval\ y'\ )$
$\longrightarrow (space2\ x\ y = space2\ x'\ y')$
begin
end

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

class *AxPlanes* = *WorldView* +
assumes
AxPlanes: ⟦ *IOb m*; *IOb k* ⟧ ⟹
(*coplanar e x y z* ⟶ *coplanar* (*wvt k m e*) (*wvt k m x*) (*wvt k m y*) (*wvt k m z*))
begin
end

class *AxCones* = *WorldView* + *AxPh* +
assumes
AxCones: ⟦ *IOb m*; *IOb k* ⟧ ⟹
(*onCone x* (*lightCone m v*) ⟶ *onCone* (*wvt k m x*) (*lightcone k* (*wvt k m v*)))
begin
end

class *AxTime* = *WorldView* +
assumes
AxTime: ⟦ *IOb m*; *IOb k* ⟧
⟹(*x* ≲ *y* ⟶ *wvt k m x* ≲ *wvt k m y*)
begin
end

end

theory *SpecRel*
imports *Axioms*
begin

class *SpecRel* = *WorldView* + *AxPh* + *AxEv* + *AxSelf* + *AxSym*

+ *AxEuclidean*

+ *AxLines* + *AxPlanes* + *AxCones*

begin

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-rules(*2*)
local.semiring-normalization-rules(*24*)
local.semiring-normalization-rules(*25*))
thus *?thesis*
by (*simp add*: *calculation*)
qed

lemma *noFTLObserver*:
assumes *iobm*: *IOb m*
and *iobk*: *IOb k*
and *mke*: *m sees k at e*
and *mkf*: *m sees k at f*
and *enotf*: $e \neq f$
shows *space2 e f* $\leq$ (*c m* $*$ *c m*) $*$ *time2 e f*
proof −

{
assume *converse*: *space2 e f* $>$ (*c m* $*$ *c m*) $*$ *time2 e f*

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

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

have *outsideCone f eCone*
$\longrightarrow$ ($\exists\, x.$(*onCone x eCone* $\wedge$ *x* $\neq$ *vertex eCone* $\wedge$ *inPlane f* (*tangentPlane x eCone*)))
by (*rule AxParallelConesE*)

hence *tplane-exists*: $\exists\, x.$(*onCone x eCone* $\wedge$ *x* $\neq$ *vertex eCone* $\wedge$ *inPlane f* (*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 *if-g-onAxis*: *onAxisT wvtg* $\longrightarrow$ *collinear wvte wvtg wvtf*
by (*metis lemAxisIsLine wvte-onAxis wvtf-onAxis wvt-e-not-f wvt-f-not-g wvt-g-not-e*)

have *collinear wvte wvtg wvtf* $\longrightarrow$ *collinear e g f*
by (*metis AxLines iobm iobk wvte-inv wvtf-inv wvtg-inv*)
hence *onAxisT wvtg* $\longrightarrow$ *collinear e g f* **by** (*metis if-g-onAxis*)

hence *wvtg-offAxis*: $\neg$ (*onAxisT wvtg*) **by** (*metis g-not-collinear*)

have $\forall\, s.(\exists\, p.(\ collinear\ wvte\ wvtg\ p \wedge (space2\ p\ wvtf = (s{*}s){*}time2\ p\ wvtf)))$
by (*metis AxSlopedLineInVerticalPlane wvte-onAxis wvtf-onAxis wvtg-offAxis wvt-e-not-f*)
hence *exists-wvtz*: $\exists\, p.(\ collinear\ wvte\ wvtg\ p \wedge (space2\ p\ wvtf = (c\ k * c\ k){*}time2\ p\ wvtf))$
by *metis*
then obtain *wvtz* **where**
wvtz-props: $collinear\ wvte\ wvtg\ wvtz \wedge (space2\ wvtz\ wvtf = (c\ k * c\ k){*}time2\ wvtz\ wvtf)$ **by** *auto*
hence *wvtf-speed*: $space2\ wvtz\ wvtf = (c\ k * c\ k){*}time2\ wvtz\ wvtf$ **by** *metis*

define *z* **where** *z* = *wvt m k wvtz*
define *wvtzCone* **where** *wvtzCone* = *lightcone k wvtz*

have *wvtz-is-vertex*: *wvtz* = *vertex wvtzCone* **by** (*simp add*: *wvtzCone-def*)
have *ck-is-slope*: *c k* = *slope wvtzCone* **by** (*simp add*: *wvtzCone-def*)
hence $space2\ (vertex\ wvtzCone)\ wvtf = ((slope\ wvtzCone) * (slope\ wvtzCone)){*}time2\ (vertex\ wvtzCone)\ wvtf$
by (*metis wvtf-speed wvtz-is-vertex ck-is-slope*)
hence *onCone wvtf wvtzCone* **by** (*metis onCone.simps*)

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

define *zCone* **where** *zCone* = *lightcone m z*
have *z-is-vertex*: *z* = *vertex zCone* **by** (*simp add*: *zCone-def*)
have *cm-is-zSlope*: *c m* = *slope zCone* **by** (*simp add*: *zCone-def*)

have *f-on-zCone*: *onCone f zCone* **by** (*metis wvtf-inv wvtf-on-wvtzCone zCone-def*)

hence *space2* (*vertex zCone*) *f* = (*slope zCone* ∗ *slope zCone*)∗*time2* (*vertex zCone*) *f*
by (*simp add*: *zCone-def*)
hence *space2 z f* = (*c m* ∗ *c m*)∗*time2 z f* **by** (*metis z-is-vertex cm-is-zSlope*)
hence *fz-speed*: *space2 f z* = (*c m* ∗ *c m*)∗*time2 f z* **by** (*metis lemSpace2Sym lemTime2Sym*)

define *fCone* **where** *fCone* = *lightcone m f*

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

have *collinear wvte wvtg wvtz* **by** (*metis wvtz-props*)
hence *egz-collinear*: *collinear e g z* **by** (*metis wvte-inv wvtg-inv z-def AxLines iobm iobk*)
hence *z-geometry*: (*inPlane z* (*tangentPlane g eCone*) ∧ *onCone z eCone*)
by (*metis AxConeTangent e-is-vertex g-on-eCone*)

have *z-on-eCone*: *onCone z eCone* **by** (*metis z-geometry*)
have *z-in-tplane*: *inPlane z tplane* **by** (*metis z-geometry tplane-def*)

hence *z-not-f*: *z* ≠ *f* **by** (*metis z-on-eCone outside lemOutsideNotOnCone*)
hence *z-not-fVertex*: *z* ≠ *vertex fCone* **by** (*simp add*: *fCone-def z-not-f*)

{
assume *assm*: *z* = *e*
have *space2 f e* = (*c m* ∗ *c m*)∗*time2 f e* ∧ *space2 f e* = *space2 e f* ∧ *time2 f e* = *time2 e f*
by (*metis lemSpace2Sym lemTime2Sym fz-speed assm*)
hence *space2 e f* = (*c m* ∗ *c m*)∗*time2 e f* **by** *metis*
hence *False* **by** (*metis less-irrefl converse*)
}
from *this* **have** *z-not-e*: *z* ≠ *e* **by** *blast*

define *lineA* **where** *lineA* = *lineJoining e z*
define *lineB* **where** *lineB* = *lineJoining f z*

{

assume *assm*: *direction lineA* = *vecZero*
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 assm*)
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 *assm*: *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 assm*)
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 lemDirectionCollinear*)

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

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

have $from\ e\ to\ g = (-b)**(from\ g\ to\ z)$ **by** (*metis e-to-g*)
hence *g-to-z*: $(from\ g\ to\ z) = binv**(from\ e\ to\ g)$
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*: $from\ e\ to\ z = factor**(from\ e\ to\ g)$ **by** (*metis lemAddOverScale lemScale1 factor-def*)
have *ez-dir*: $direction\ (lineJoining\ e\ z) = from\ e\ to\ z$ **by** *simp*
have *eg-dir*: $direction\ (lineJoining\ e\ g) = from\ e\ to\ g$ **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 $direction\ (lineJoining\ e\ z) = vecZero$ **by** (*simp*)
hence *False* **by** (*metis ez-not-null lineA-def*)
}
from *this* **have** *factor-nonzero*: $factor \neq 0$ **by** *blast*

have $direction\ (lineJoining\ e\ z) = factor**(direction\ (lineJoining\ e\ g))$
by (*metis e-to-z ez-dir eg-dir*)
hence $(lineJoining\ e\ g) \parallel (lineJoining\ e\ z)$ **by** (*metis parallel.simps factor-nonzero*)
hence $(lineJoining\ e\ z) \parallel (lineJoining\ e\ g)$ **by** (*metis lemParallelSym*)

hence $(lineJoining\ e\ z) \parallel (lineJoining\ f\ z)$ **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*: *lineB-def*)
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 lemTime2Sym*)
hence *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*∗*β*)∗*time2 z f* **by** (*metis mult.assoc mult.commute*)
hence *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 lemTime2Sym*)
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*

hence *False* **by** (*metis A-par-B ez-not-null fz-not-null e-in-lineA e-not-in-lineB*

lemParallelNotMeet)
}
from *this* **have** ¬ (*space2 e f* > (*c m* ∗ *c m*) ∗ *time2 e f*) **by** *blast*

thus *?thesis* **by** *simp*
qed

end

end

References

[1] M. Stannett and I. Németi. Using Isabelle/HOL to verify first-order relativity theory. *Journal of Automated Reasoning*, 52(4):361–378, 2014.