

# No Faster-Than-Light Observers

Mike Stannett

May 26, 2024

## 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 Némethi [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  $\Rightarrow$  'a Vector  $\Rightarrow$  bool (**infix**  $\perp$  150) **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** *lemVecPlusCommute*:

**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** *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**  $u = (\text{inverse } k)**v$   
**proof** –  
**have**  $(\text{inverse } k)**v = (\text{inverse } k * k)**u$   
**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  $\equiv$  ( $\lfloor$   $tval = 0, xval = 0, yval = 0, zval = 0$   $\rfloor$ )

```

```

  fun vectorJoining :: 'a Point  $\Rightarrow$  'a Point  $\Rightarrow$  'a Vector (from - to -) where
    vectorJoining  $p\ q$ 
      = ( $\lfloor$   $tdir = tval\ q - tval\ p, xdir = xval\ q - xval\ p,$ 
         $ydir = yval\ q - yval\ p, zdir = zval\ q - zval\ p$   $\rfloor$ )

```

```

  fun moveBy :: 'a Point  $\Rightarrow$  'a Vector  $\Rightarrow$  'a Point (infixl  $\rightsquigarrow$  100) where
    moveBy  $p\ u$ 
      = ( $\lfloor$   $tval = tval\ p + tdir\ u, xval = xval\ p + xdir\ u,$ 
         $yval = yval\ p + ydir\ u, zval = zval\ p + zdir\ u$   $\rfloor$ )

```

```

  fun positionVector :: 'a Point  $\Rightarrow$  'a Vector where
    positionVector  $p =$  ( $\lfloor$   $tdir = tval\ p, xdir = xval\ p, ydir = yval\ p, zdir = zval\ p$   $\rfloor$ )

```

```

  fun before :: 'a Point  $\Rightarrow$  'a Point  $\Rightarrow$  bool (infixr  $\lesssim$  100) where
    before  $p\ q = (tval\ p < tval\ q)$ 

```

```

  fun after :: 'a Point  $\Rightarrow$  'a Point  $\Rightarrow$  bool (infixr  $\gtrsim$  100) where
    after  $p\ q = (tval\ p > tval\ q)$ 

```

```

  fun sametime :: 'a Point  $\Rightarrow$  'a Point  $\Rightarrow$  bool (infixr  $\approx$  100) where
    sametime  $p\ q = (tval\ p = tval\ q)$ 

```

```

lemma lemFromToTo:

```

```

  shows (from  $p$  to  $q$ )  $\oplus$  (from  $q$  to  $r$ ) = (from  $p$  to  $r$ )

```

```

proof -

```

```

  have shared:  $\forall\ valp\ valq\ valr. (valq - valp + (valr - valq) = valr - valp)$ 

```

```

  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*  $\Rightarrow$  *bool* **where**  
*onAxisT* *u* = ((*xval* *u* = 0)  $\wedge$  (*yval* *u* = 0)  $\wedge$  (*zval* *u* = 0))

**fun** *space2* :: ('*a* *Point*)  $\Rightarrow$  ('*a* *Point*)  $\Rightarrow$  '*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*)  $\Rightarrow$  ('*a* *Point*)  $\Rightarrow$  '*a* **where**  
*time2* *u* *v* = (*tval* *u* - *tval* *v*)\*(*tval* *u* - *tval* *v*)

**fun** *speed* :: ('*a* *Point*)  $\Rightarrow$  ('*a* *Point*)  $\Rightarrow$  '*a* **where**  
*speed* *u* *v* = (*space2* *u* *v* / *time2* *u* *v*)

**fun** *mkLine* :: '*a* *Point*  $\Rightarrow$  '*a* *Vector*  $\Rightarrow$  '*a* *Line* **where**  
*mkLine* *b* *d* = ( $\lfloor$  *basepoint* = *b*, *direction* = *d*  $\rfloor$ )

**fun** *lineJoining* :: '*a* *Point*  $\Rightarrow$  '*a* *Point*  $\Rightarrow$  '*a* *Line* (*line joining - to -*) **where**  
*lineJoining* *p* *q* = ( $\lfloor$  *basepoint* = *p*, *direction* = *from p to q*  $\rfloor$ )

**fun** *parallel* :: '*a* *Line*  $\Rightarrow$  '*a* *Line*  $\Rightarrow$  *bool* (*- ||*) **where**  
*parallel* *lineA* *lineB* = ((*direction* *lineA* = *vecZero*)  $\vee$  (*direction* *lineB* = *vecZero*)  
 $\vee$  ( $\exists$  *k*.(*k*  $\neq$  (0::*a*)  $\wedge$  *direction* *lineB* = *k\*\*direction*  
*lineA*)))

**fun** *collinear* :: '*a* *Point*  $\Rightarrow$  '*a* *Point*  $\Rightarrow$  '*a* *Point*  $\Rightarrow$  *bool* **where**  
*collinear* *p* *q* *r* = ( $\exists$   $\alpha$   $\beta$ . ( $\alpha + \beta = 1$ )  $\wedge$   
*positionVector* *p* =  $\alpha**$ (*positionVector* *q*)  $\oplus$   $\beta**$ (*positionVector* *r*))

**fun** *inLine* :: '*a* *Point*  $\Rightarrow$  '*a* *Line*  $\Rightarrow$  *bool* **where**  
*inLine* *p* *l* = *collinear* *p* (*basepoint* *l*) (*basepoint* *l*  $\rightsquigarrow$  *direction* *l*)

```

fun meets :: 'a Line  $\Rightarrow$  'a Line  $\Rightarrow$  bool where
  meets line1 line2 = ( $\exists p.(inLine\ p\ line1 \wedge inLine\ p\ line2)$ )

```

```

lemma lemParallelReflexive:
  shows lineA  $\parallel$  lineA
proof -
  define dir where dir = direction lineA
  have (1  $\neq$  0)  $\wedge$  (dir = 1**dir) by simp
  thus ?thesis by (metis dir-def parallel.simps)
qed

```

```

lemma lemParallelSym:
  assumes lineA  $\parallel$  lineB
  shows lineB  $\parallel$  lineA
proof -
  have case1: direction lineA = vecZero  $\longrightarrow$  ?thesis by auto
  have case2: direction lineB = vecZero  $\longrightarrow$  ?thesis by auto
  {
    assume case3: direction lineA  $\neq$  vecZero  $\wedge$  direction lineB  $\neq$  vecZero
    have exists-kab:  $\exists kab.(kab \neq (0::'a) \wedge direction\ lineB = kab**direction\ lineA)$ 

      by (metis parallel.simps assms(1) case3)
      define kab where kab  $\equiv$  (SOME kab.(kab  $\neq$  (0::'a)  $\wedge$  direction lineB =
kab**direction lineA))
      have kab-props: kab  $\neq$  0  $\wedge$  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  $\neq$  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  $\neq$  vecZero  $\wedge$  direction lineB  $\neq$  vecZero)  $\longrightarrow$ 
?thesis by blast

  thus ?thesis by (metis case1 case2)
qed

```

```

lemma lemParallelTrans:
  assumes lineA  $\parallel$  lineB
  and lineB  $\parallel$  lineC
  and direction lineB  $\neq$  vecZero
  shows lineA  $\parallel$  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**  $\forall a b. (b = a + (b - a))$   
**by** (*metis diff-add-cancel semiring-normalization-rules(24)*)  
**thus** *?thesis* **by** *auto*  
**qed**

**lemma** *lemNullLine*:  
**assumes** *direction lineA = vecZero*  
**and** *inLine x lineA*  
**shows** *x = basepoint lineA*  
**proof** –  
**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**  $\exists a b. (a + b = 1) \wedge$   
 $(\text{positionVector } x = a**(\text{positionVector } bp) \oplus b**(\text{positionVector } bp))$   
**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**  $(1 + 0 = 1) \wedge (\text{positionVector } p = 1**(\text{positionVector } p) \oplus 0**(\text{positionVector } q))$

**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 lemLineContainsEndpoint*)  
  
**have** ( $0 + 1 = 1$ )  $\wedge$  (*positionVector q = 0\*\* (positionVector p)  $\oplus$  1\*\* (positionVector q)*)  
**by** *auto*  
**hence** *collinear q p q* **by** (*metis collinear.simps*)  
**hence** *collinear q (basepoint linePQ) (basepoint linePQ ~ direction linePQ)*  
**by** (*metis bp endq*)  
**thus** *?thesis* **by** (*simp add: linePQ-def*)  
**qed**

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

**lemma** *lemParallelJoin*:  
**assumes** *line joining p to q || 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)  $\longrightarrow$  ?thesis* **by** (*simp add: linePQ-def*)  
**have** *case2: (direction linePR = vecZero)  $\longrightarrow$  ?thesis* **by** (*simp add: linePR-def*)  
  
**{**  
**assume** *case3: direction linePQ  $\neq$  vecZero  $\wedge$  direction linePR  $\neq$  vecZero*  
**{**  
**assume** *case3a: direction lineQR = vecZero*  
**have** *inLine r lineQR* **by** (*metis lemLineContainsEndpoint lineQR-def*)  
**}**  
**}**

```

    hence  $r = \text{basepoint } \text{lineQR}$  by (metis lemNullLine case3a)
    hence  $r = q$  by (simp add: lineQR-def)
    hence  $\text{linePQ} = \text{linePR}$  by (simp add: linePQ-def linePR-def)
    hence ?thesis by (metis lemParallelReflexive linePQ-def linePR-def)
  }
from this have rtp3a: direction lineQR = vecZero  $\longrightarrow$  ?thesis by blast

{
  assume case3b: direction lineQR  $\neq$  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:  $\exists k. (k \neq 0 \wedge \text{direction } \text{lineQR} = k**\text{direction } \text{linePQ})$ 
    by (metis linePQ-def lineQR-def assms(1) parallel.simps case3b case3)
  then obtain k where k-props:  $k \neq 0 \wedge \text{dirQR} = k**\text{dirPQ}$  by (metis dir-pq
dir-qr)

  define scalar where scalar = 1+k

  have  $q = p \rightsquigarrow \text{dirPQ} \wedge r = q \rightsquigarrow \text{dirQR}$  by (metis lemLineEndpoint
dirPQ-def dirQR-def)
  hence  $r = p \rightsquigarrow \text{dirPQ} \rightsquigarrow (k**\text{dirPQ})$  by (metis k-props)
  hence scalarPR:  $r = p \rightsquigarrow \text{scalar**dirPQ}$ 
    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  $\neq$  0 by blast

  have linePR = line joining p to (p  $\rightsquigarrow$  scalar**dirPQ)
    by (simp add: linePR-def scalarPR)
  hence direction linePR = scalar**dirPQ by (metis lemDirectionFromTo)

  hence scalar-props: scalar  $\neq$  0  $\wedge$  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  $\neq$  vecZero  $\longrightarrow$  ?thesis by blast

hence ?thesis by (metis rtp3a)

```

```

}
from this have (direction linePQ ≠ vecZero ∧ direction linePR ≠ vecZero) →
?thesis by blast

thus ?thesis by (metis case1 case2)
qed

```

**lemma** *lemDirectionCollinear*:

**shows** *collinear u v (v ~ d) ↔ (∃ β. (from u to v = (-β)\*\*d))*

**proof** -

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

**have** *basic2*:  $\forall u v w. (u = v \oplus w \longrightarrow v \ominus u = \text{vecNegate } w)$

**apply** *auto*

**by** (*metis add-uminus-conv-diff diff-add-cancel minus-add*  
*semiring-normalization-rules(24)*) +

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

**have** *basic4*:  $\forall u v w. (v \ominus u = \text{vecNegate } w \longrightarrow u = v \oplus w)$

**apply** *auto*

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

{

**assume** *asm*: *collinear u v (v ~ d)*

**have**  $\exists \alpha \beta. (\alpha + \beta = 1) \wedge$

*positionVector u = α\*\*(positionVector v) ⊕ β\*\*(positionVector (v ~ d))* )

**by** (*metis asm collinear.simps*)

**then obtain**  $\alpha \beta$  **where** *props*:  $(\alpha + \beta = 1) \wedge$

*positionVector u = α\*\*(positionVector v) ⊕ β\*\*(positionVector (v ~*

*d))* **by** *auto*

**hence** *positionVector u = 1\*\*(positionVector v) ⊕ β\*\*d*

**by** (*metis basic1 lemScaleOverAdd lemVecPlusAssoc lemAddOverScale props*)

**hence** *positionVector u = positionVector v ⊕ β\*\*d* **by** (*metis lemScale1*)

**hence** *positionVector v ⊖ positionVector u = (-β)\*\*d* **by** (*metis basic2*

*lemScaleNeg*)

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

}

**from this have** *fwd*: *collinear u v (v ~ d) → (∃ β. (from u to v = (-β)\*\*d))*

**by** *blast*

{

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

**then obtain**  $\beta$  **where** *asm*: *from u to v = (-β)\*\*d* **by** *auto*

**define**  $\alpha$  **where**  $\alpha = 1 - \beta$

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

**have** *from u to v = vecNegate (β\*\*d)* **by** (*metis asm lemScaleNeg*)

**hence** *positionVector v ⊖ positionVector u = vecNegate (β\*\*d)* **by** *auto*

**hence** *positionVector u = positionVector v ⊕ β\*\*d* **by** (*metis basic4*)

**hence** *positionVector u = 1\*\*(positionVector v) ⊕ β\*\*d*

**by** (*metis lemScale1*)  
**hence**  $(\alpha + \beta = 1) \wedge$   
 $\text{positionVector } u = \alpha ** (\text{positionVector } v) \oplus \beta ** (\text{positionVector } (v \rightsquigarrow d))$   
**by** (*metis  $\alpha\beta$ -sum basic1 lemScaleOverAdd lemVecPlusAssoc lemAddOverScale*)  
**hence** *collinear*  $u\ v\ (v \rightsquigarrow d)$  **by** *auto*  
**}**  
**from this have**  $(\exists \beta. (\text{from } u \text{ to } v = (-\beta) ** d)) \longrightarrow \text{collinear } u\ v\ (v \rightsquigarrow d)$  **by**  
*blast*  
  
**thus** *?thesis* **by** (*metis fwd*)  
**qed**

**lemma** *lemParallelNotMeet*:

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

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

**define** *bpA* **where**  $\text{bpA} = \text{basepoint } \text{lineA}$   
**define** *dirA* **where**  $\text{dirA} = \text{direction } \text{lineA}$   
**define** *bpB* **where**  $\text{bpB} = \text{basepoint } \text{lineB}$   
**define** *dirB* **where**  $\text{dirB} = \text{direction } \text{lineB}$

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

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

**have** *collinear*  $x\ \text{bpA}\ (\text{bpA} \rightsquigarrow \text{dirA})$  **by** (*metis assms(4) inLine.simps bpA-def dirA-def*)

**then obtain**  $\beta$  **where**  $\text{from } x \text{ to } \text{bpA} = (-\beta) ** \text{dirA}$  **by** (*metis lemDirectionCollinear*)

**hence** *x-to-bpA*:  $\text{from } x \text{ to } \text{bpA} = ((-\beta) * \text{kab}) ** \text{dirB}$  **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**  $p\text{-in-}AB: inLine\ p\ lineA \wedge inLine\ p\ lineB$  **by** *auto*

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

**define**  $\gamma$  **where**  $\gamma = -((-\beta)*kab + (\beta A*kab) + (-\beta B))$   
**have**  $x\text{-to-}bpB: (from\ x\ to\ bpA) \oplus (from\ bpA\ to\ p) \oplus (from\ p\ to\ bpB) = (from\ x\ to\ bpB)$   
**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  $\gamma$ -def minus-minus*)  
**hence** *collinear x bpB (bpB  $\rightsquigarrow$  dirB)* **by** (*metis lemDirectionCollinear*)  
**hence** *inLine x lineB* **by** (*metis inLine.simps bpB-def dirB-def*)  
**}**  
**from this have** *meets lineA lineB  $\longrightarrow$  inLine x lineB* **by** *blast*  
**thus** *?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**  $x\text{-onAxis}: xval\ x = 0 \wedge yval\ x = 0 \wedge zval\ x = 0$  **by** (*metis assms(1) onAxisT.simps*)

**have**  $y\text{-onAxis}: xval\ y = 0 \wedge yval\ y = 0 \wedge zval\ y = 0$  **by** (*metis assms(2)*)

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

**have**  $tval\ z - tval\ y = 0 \longrightarrow z = y$  **by** (*simp* *add*: *z-onAxis* *y-onAxis*)  
**hence**  $tval\ z \neq tval\ y$  **by** (*metis* *assms*(5) *eq-iff-diff-eq-0*)  
**hence** *tvalyz-nonzero*:  $tval\ z - tval\ y \neq 0$  **by** (*metis* *eq-iff-diff-eq-0*)

**have** *x-to-y*: *from* *x* *to* *y* = ( $\lfloor\ tdir = tval\ y - tval\ x, xdir = 0, ydir = 0, zdir = 0\ \rfloor$ )  
**by** (*simp* *add*: *x-onAxis* *y-onAxis*)  
**have** *y-to-z*: *from* *y* *to* *z* = ( $\lfloor\ tdir = tval\ z - tval\ y, xdir = 0, ydir = 0, zdir = 0\ \rfloor$ )  
**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* ( $y \rightsquigarrow$  (*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 fccone* ∧ *f* ≠  
*vertex fccone*

∧ *inPlane* *f* (*tangentPlane* *e econe*))  
→ (*samePlane* (*tangentPlane* *e econe*) (*tangentPlane* *f fccone*)  
∧ ((*lineJoining* (*vertex econe*) *e*) ∥ (*lineJoining* (*vertex fccone*)

*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  $\Rightarrow$  Body  $\Rightarrow$  'a Point  $\Rightarrow$  bool (- sees - at -)
and

  wvt :: Body  $\Rightarrow$  Body  $\Rightarrow$  'a Point  $\Rightarrow$  'a Point
assumes
  AxWVT:  $\llbracket IOb\ m; IOb\ k \rrbracket \Longrightarrow (W\ k\ b\ x \longleftrightarrow W\ m\ b\ (wvt\ m\ k\ x))$ 
and
  AxWVTSym:  $\llbracket IOb\ m; IOb\ k \rrbracket \Longrightarrow (y = wvt\ k\ m\ x \longleftrightarrow x = wvt\ m\ k\ y)$ 
begin
end

class AxiomPreds = WorldView
begin
  fun sqrtTest :: 'a  $\Rightarrow$  'a  $\Rightarrow$  bool where
    sqrtTest x r =  $((r \geq 0) \wedge (r*r = x))$ 

  fun cTest :: Body  $\Rightarrow$  'a  $\Rightarrow$  bool where
    cTest m v =  $( (v > 0) \wedge ( \forall x\ y . ($ 
       $(\exists p. (Ph\ p \wedge W\ m\ p\ x \wedge W\ m\ p\ y)) \longleftrightarrow (space2\ x\ y = (v * v)*(time2$ 
x y))
      )))
end

class AxEuclidean = AxiomPreds + Quantities +
assumes
  AxEuclidean:  $(x \geq Groups.zero-class.zero) \Longrightarrow (\exists r. sqrtTest\ x\ r)$ 
begin

  abbreviation sqrt :: 'a  $\Rightarrow$  'a where
    sqrt  $\equiv$  someFunc sqrtTest

  lemma lemSqrt:

```



```

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 (simp add: 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:  $\llbracket IOb\ m; IOb\ k \rrbracket \implies (\exists y. (\forall b. (W\ m\ b\ x \longleftrightarrow W\ k\ b\ y)))$ 
begin
end

```

```

class AxThExp = WorldView + AxPh +
assumes

```

```

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

```

```

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  $\implies$   $c\ m = 1$ 
begin
end

```

```

class AxSym = WorldView +
assumes
  AxSym:  $\llbracket IOb\ m; IOb\ k \rrbracket \implies$ 
    ( $W m e x \wedge W m f y \wedge W k e x' \wedge W k f y' \wedge$ 
      $tval\ x = tval\ y \wedge tval\ x' = tval\ y'$ )
     $\longrightarrow$  ( $space2\ x\ y = space2\ x'\ y'$ )
begin
end

```

```

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

```

```

class AxPlanes = WorldView +
assumes
  AxPlanes:  $\llbracket \text{IOb } m; \text{IOb } k \rrbracket \implies$ 
    (coplanar e x y z  $\longrightarrow$  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:  $\llbracket \text{IOb } m; \text{IOb } k \rrbracket \implies$ 
    (onCone x (lightCone m v)  $\longrightarrow$  onCone (wvt k m x) (lightcone k (wvt k m v)))
begin
end

```

```

class AxTime = WorldView +
assumes
  AxTime:  $\llbracket \text{IOb } m; \text{IOb } k \rrbracket$ 
     $\implies (x \lesssim y \longrightarrow \text{wvt } k \ m \ x \lesssim \text{wvt } k \ m \ y)$ 
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**  $\text{space2 } e f \leq (c m * c m) * \text{time2 } e f$   
**proof** –

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

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

**have** *e-is-vertex*:  $e = \text{vertex } eCone$  **by** (*simp add: eCone-def*)  
**have** *cm-is-slope*:  $c m = \text{slope } eCone$  **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*  
 $\longrightarrow (\exists x. (\text{onCone } x eCone \wedge x \neq \text{vertex } eCone \wedge \text{inPlane } f (\text{tangentPlane } x eCone)))$   
**by** (*rule AxParallelConesE*)

**hence** *tplane-exists*:  $\exists x. (\text{onCone } x eCone \wedge x \neq \text{vertex } eCone \wedge \text{inPlane } f (\text{tangentPlane } x eCone))$



**by** (*metis outside*)  
**then obtain**  $g$  **where**  $g$ -props: ( $onCone\ g\ eCone \wedge g \neq vertex\ eCone \wedge inPlane\ f\ (tangentPlane\ g\ eCone)$ )  
**by** *auto*  
**have**  $g$ -on-eCone:  $onCone\ g\ eCone$  **by** (*metis g-props*)  
**have**  $g$ -not-vertex:  $g \neq vertex\ eCone$  **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** ( $onCone\ g\ eCone$ )  $\longrightarrow$   
 $((inPlane\ f\ (tangentPlane\ g\ eCone) \wedge onCone\ f\ eCone)$   
 $\longleftrightarrow collinear\ (vertex\ eCone)\ g\ f)$

**by** (*metis AxConeTangent*)  
**hence**  $axconetangent$ :  $collinear\ e\ g\ f \longrightarrow onCone\ f\ eCone$   
**by** (*metis g-on-eCone e-is-vertex*)  
**have**  $\neg(onCone\ f\ eCone)$  **by** (*metis outside lemOutsideNotOnCone*)  
**hence**  $g$ -not-collinear:  $\neg(collinear\ e\ g\ f)$   
**by** (*metis axconetangent*)

**define**  $wte$  **where**  $wte = wvt\ k\ m\ e$   
**define**  $wtf$  **where**  $wtf = wvt\ k\ m\ f$   
**define**  $wtg$  **where**  $wtg = wvt\ k\ m\ g$

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

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

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

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

**have**  $wt$ - $e$ -not- $f$ :  $wte \neq wtf$  **by** (*metis wte-inv wtf-inv enotf*)  
**have**  $wt$ - $f$ -not- $g$ :  $wtf \neq wtg$  **by** (*metis wtf-inv wtg-inv f-not-g*)  
**have**  $wt$ - $g$ -not- $e$ :  $wtg \neq wte$  **by** (*metis wtg-inv wte-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** *wvtz-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 wvtz-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**  $collinear\ wute\ wvtg\ wtz$  **by** (*metis wvtz-props*)  
**hence**  $egz-collinear: collinear\ e\ g\ z$  **by** (*metis wute-inv wvtg-inv z-def AxLines iobm iobk*)  
**hence**  $z-geometry: (inPlane\ z\ (tangentPlane\ g\ eCone) \wedge onCone\ z\ eCone)$   
**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**  $z-not-f: z \neq f$  **by** (*metis z-on-eCone outside lemOutsideNotOnCone*)  
**hence**  $z-not-fVertex: z \neq vertex\ fCone$  **by** (*simp add: fCone-def z-not-f*)

{  
**assume**  $assm: z = e$   
**have**  $space2\ f\ e = (c\ m * c\ m)*time2\ f\ e \wedge space2\ f\ e = space2\ e\ f \wedge time2\ f\ e = time2\ e\ f$   
**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**  $z-not-e: z \neq e$  **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  $\wedge$  inLine z lineA)  $\longrightarrow$  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  $\neq$  vecZero by blast

{
  assume assm: direction lineB = vecZero
  have lemnullline: (direction lineB = vecZero  $\wedge$  inLine z lineB)  $\longrightarrow$  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  $\neq$  vecZero by blast

{
  have samePlane tplane (tangentPlane z fCone
     $\wedge$  (lineJoining e g  $\parallel$  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  $\parallel$  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:  $\neg$ (direction (lineJoining e g) = vecZero) by blast
  then obtain a where a-props: a  $\neq$  0  $\wedge$  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  $\neq$  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  $\rightsquigarrow$  (from g to z) by (metis lemLineEndpoint)
  hence collinear e g (g  $\rightsquigarrow$  (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  $asm: -b = 0$**   
**have  $from\ e\ to\ g = (-b)**(from\ g\ to\ z)$  by (metis e-to-g)**  
**hence  $from\ e\ to\ g = vecZero$  by (simp add: asm)**  
**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  $asm: factor = 0$**   
**have  $from\ e\ to\ z = factor**(from\ e\ to\ g)$  by (metis e-to-z)**  
**hence  $from\ e\ to\ z = vecZero$  by (simp add: asm)**  
**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 fac-**  
**tor-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)**

$eg\text{-not-null}$   
 $\}$   
**from this have**  $A\text{-par-}B: lineA \parallel lineB$  **by** (*metis lineA-def lineB-def*)

**have**  $e\text{-in-}lineA: inLine\ e\ lineA$  **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*:  $collinear\ e\ f$  ( $f \rightsquigarrow direction\ lineB$ ) **by** (*simp add: lineB-def*)  
**then obtain**  $\beta$  **where** *props*:  $from\ e\ to\ f = (-\beta)**(direction\ lineB)$   
**by** (*metis lemDirectionCollinear*)

**hence**  $tval\ f - tval\ e = (-\beta)*(tval\ z - tval\ f) \wedge xval\ f - xval\ e = (-\beta)*(xval\ z - xval\ f)$   
 $\wedge yval\ f - yval\ e = (-\beta)*(yval\ z - yval\ f) \wedge zval\ f - zval\ e = (-\beta)*(zval\ z - zval\ f)$   
**by** (*simp add: lineB-def*)  
**hence** *speeds*:  $time2\ f\ e = (\beta*\beta)*time2\ z\ f \wedge space2\ f\ e = (\beta*\beta)*space2\ z\ f$   
**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)*)

*semiring-normalization-rules(34)*)

**have**  $space2\ f\ z = (c\ m * c\ m)*time2\ f\ z$  **by** (*metis fz-speed*)  
**hence**  $space2\ z\ f = (c\ m * c\ m)*time2\ z\ f$  **by** (*metis lemSpace2Sym lem-Time2Sym*)  
**hence**  $space2\ f\ e = ((\beta*\beta)*(c\ m * c\ m))*time2\ z\ f$  **by** (*metis speeds mult.assoc*)  
**hence**  $space2\ f\ e = (c\ m * c\ m)*(\beta*\beta)*time2\ z\ f$  **by** (*metis mult.assoc mult commute*)  
**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 lem-Time2Sym*)  
**hence** *False* **by** (*metis less-irrefl converse*)

$\}$   
**from this have**  $e\text{-not-in-}lineB: \neg(inLine\ e\ lineB)$  **by** *blast*

**have**  $inLine\ z\ lineA \wedge inLine\ z\ lineB$  **by** (*metis lemLineContainsEndpoint lineA-def lineB-def*)  
**hence**  $A\text{-meets-}B: meets\ lineA\ lineB$  **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  $\neg (\text{space2 } e f > (c\ m * c\ m) * \text{time2 } e f)$  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.