

No Faster-Than-Light Observers

Mike Stannett

February 6, 2026

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 (infix <⊕> 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 (infix <⊖> 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** $\langle \perp \rangle$ 150) **where**
orthogonal $u\ v = (u\ dot\ v = 0)$

lemma *lemVecZeroMinus*:

shows $0 \ominus u = \sim u$
<proof>

lemma *lemVecSelfMinus*:

shows $u \ominus u = 0$
<proof>

lemma *lemVecPlusCommute*:

shows $u \oplus v = v \oplus u$
<proof>

lemma *lemVecPlusAssoc*:

shows $u \oplus (v \oplus w) = (u \oplus v) \oplus w$
<proof>

lemma *lemVecPlusMinus*:

shows $u \oplus (\sim v) = u \ominus v$
<proof>

lemma *lemDotCommute*:

shows $(u\ dot\ v) = (v\ dot\ u)$
<proof>

lemma *lemMDotCommute*:

shows $(u\ mdot\ v) = (v\ mdot\ u)$
<proof>

lemma *lemScaleScale*:

shows $a**(b**u) = (a*b)**u$
<proof>

lemma *lemScale1*:
 shows $1 ** u = u$
 ⟨*proof*⟩

lemma *lemScale0*:
 shows $0 ** u = 0$
 ⟨*proof*⟩

lemma *lemScaleNeg*:
 shows $(-k)**u = \sim (k**u)$
 ⟨*proof*⟩

lemma *lemScaleOrigin*:
 shows $k**0 = 0$
 ⟨*proof*⟩

lemma *lemScaleOverAdd*:
 shows $k**(u \oplus v) = k**u \oplus k**v$
 ⟨*proof*⟩

lemma *lemAddOverScale*:
 shows $a**u \oplus b**u = (a+b)**u$
 ⟨*proof*⟩

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

lemma *lemOrthoSym*:
 assumes $u \perp v$

shows $v \perp u$
<proof>

end

class *Points* = *Quantities* + *Vectors*
begin

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

fun *vectorJoining* :: 'a *Point* \Rightarrow 'a *Point* \Rightarrow 'a *Vector* (*<from - to ->*) **where**
vectorJoining *p q*
= (\mid *tdir* = *tval* *q* - *tval* *p*, *xdir* = *xval* *q* - *xval* *p*,
ydir = *yval* *q* - *yval* *p*, *zdir* = *zval* *q* - *zval* *p* \mid)

fun *moveBy* :: 'a *Point* \Rightarrow 'a *Vector* \Rightarrow 'a *Point* (**infixl** *<~>* 100) **where**
moveBy *p u*
= (\mid *tval* = *tval* *p* + *tval* *u*, *xval* = *xval* *p* + *xval* *u*,
yval = *yval* *p* + *yval* *u*, *zval* = *zval* *p* + *zval* *u* \mid)

fun *positionVector* :: 'a *Point* \Rightarrow 'a *Vector* **where**
positionVector *p* = (\mid *tval* = *tval* *p*, *xdir* = *xval* *p*, *ydir* = *yval* *p*, *zdir* = *zval* *p* \mid)

fun *before* :: 'a *Point* \Rightarrow 'a *Point* \Rightarrow *bool* (**infixr** *<~>* 100) **where**
before *p q* = (*tval* *p* < *tval* *q*)

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

fun *sametime* :: 'a *Point* \Rightarrow 'a *Point* \Rightarrow *bool* (**infixr** *<~>* 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>

lemma *lemMoveByMove*:

shows $p \rightsquigarrow u \rightsquigarrow v = p \rightsquigarrow (u \oplus v)$
<proof>

lemma *lemScaleLinear*:

shows $p \rightsquigarrow a**u \rightsquigarrow b**v = p \rightsquigarrow (a**u \oplus b**v)$
<proof>

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*⟩

lemma *lemParallelSym*:
 assumes $lineA \parallel lineB$
 shows $lineB \parallel lineA$
 ⟨*proof*⟩

lemma *lemParallelTrans*:
 assumes $lineA \parallel lineB$
 and $lineB \parallel lineC$
 and $direction\ lineB \neq vecZero$
 shows $lineA \parallel lineC$
 ⟨*proof*⟩

lemma (in $-$) *lemLineIdentity*:
 assumes $lineA = (\mid basepoint = basepoint\ lineB, direction = direction\ lineB)$
 shows $lineA = lineB$
 ⟨*proof*⟩

lemma *lemDirectionJoining*:
 shows $vectorJoining\ p\ (p \rightsquigarrow v) = v$
 ⟨*proof*⟩

lemma *lemDirectionFromTo*:
 shows $direction\ (line\ joining\ p\ to\ (p \rightsquigarrow dir)) = dir$
 ⟨*proof*⟩

lemma *lemLineEndpoint*:
 shows $q = p \rightsquigarrow (from\ p\ to\ q)$
 ⟨*proof*⟩

lemma *lemNullLine*:
 assumes $direction\ lineA = vecZero$
 and $inLine\ x\ lineA$
 shows $x = basepoint\ lineA$
 ⟨*proof*⟩

lemma *lemLineContainsBasepoint*:
 shows $inLine\ p\ (line\ joining\ p\ to\ q)$
 ⟨*proof*⟩

lemma *lemLineContainsEndpoint*:
 shows $inLine\ q\ (line\ joining\ p\ to\ q)$

<proof>

lemma *lemDirectionReverse:*

shows *from q to p = vecNegate (from p to q)*

<proof>

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>

lemma *lemDirectionCollinear:*

shows *collinear u v (v \rightsquigarrow d) \longleftrightarrow ($\exists \beta.$ (*from u to v = (- β)**d*)*

<proof>

lemma *lemParallelNotMeet:*

assumes *lineA || 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>

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>

lemma *lemSpace2Sym:*

shows *space2 x y = space2 y x*

<proof>

lemma *lemTime2Sym:*

shows *time2 x y = time2 y x*

<proof>

end

```

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

  fun coplanar :: 'a Point  $\Rightarrow$  'a Point  $\Rightarrow$  'a Point  $\Rightarrow$  'a Point  $\Rightarrow$  bool where
    coplanar e x y z
      = ( $\exists \alpha \beta \gamma. (\alpha + \beta + \gamma = 1) \wedge$ 
        positionVector e
          = ( $\alpha ** (\text{positionVector } x) \oplus \beta ** (\text{positionVector } y) \oplus \gamma ** (\text{positionVector } z)$ 
            )))

  fun inPlane :: 'a Point  $\Rightarrow$  'a Plane  $\Rightarrow$  bool where
    inPlane e pl = coplanar e (pbasepoint pl) (pbasepoint pl  $\rightsquigarrow$  direction1 pl)
                  (pbasepoint pl  $\rightsquigarrow$  direction2 pl)

  fun samePlane :: 'a Plane  $\Rightarrow$  'a Plane  $\Rightarrow$  bool where
    samePlane pl pl' = (inPlane (pbasepoint pl) pl'  $\wedge$ 
                       inPlane (pbasepoint pl  $\rightsquigarrow$  direction1 pl) pl'  $\wedge$ 
                       inPlane (pbasepoint pl  $\rightsquigarrow$  direction2 pl) pl')

lemma lemPlaneContainsBasePoint:
  shows inPlane (pbasepoint pl) pl
  <proof>

end

class Cones = Quantities + Lines + Planes +
fixes

  tangentPlane :: 'a Point  $\Rightarrow$  'a Cone  $\Rightarrow$  'a Plane
assumes

  AxTangentBase: pbasepoint (tangentPlane e cone) = e
and

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

  AxConeTangent: (onCone e cone)  $\longrightarrow$ 
    ((inPlane pt (tangentPlane e cone)  $\wedge$  onCone pt cone)
      $\longleftrightarrow$  collinear (vertex cone) e pt)
and

```

AxParallelCones: (*onCone e econ* \wedge *e* \neq *vertex econ* \wedge *onCone f fc* \wedge *f* \neq *vertex fc*
 \wedge *inPlane f (tangentPlane e econ)*
 \longrightarrow (*samePlane (tangentPlane e econ) (tangentPlane f fc)*
 \wedge (*lineJoining (vertex econ) e* \parallel *lineJoining (vertex fc)*
f))
and

AxParallelConesE: *outsideCone f cone*
 \longrightarrow ($\exists e.$ (*onCone e cone* \wedge *e* \neq *vertex cone* \wedge *inPlane f (tangentPlane e cone)*))
and

AxSlopedLineInVerticalPlane: [*onAxisT e*; *onAxisT f*; *e* \neq *f*; \neg (*onAxisT g*)]
 \implies ($\forall s.$ ($\exists p.$ (*collinear e g p* \wedge (*space2 p f* = (*s*s*)**time2 p f*))))

begin

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

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

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

fun *mkCone* :: '*a Point* \Rightarrow '*a* \Rightarrow '*a Cone* **where**
mkCone v s = (*vertex* = *v*, *slope* = *s*)

lemma *lemVertexOnCone*:
shows *onCone (vertex cone) cone*
<proof>

lemma *lemOutsideNotOnCone*:
assumes *outsideCone f cone*
shows \neg (*onCone f cone*)
<proof>

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⟩

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⟩
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* \Rightarrow 'a *Point* \Rightarrow 'a *Cone* **where**
lightcone *m* *v* = *mkCone* *v* (*c m*)

lemma *lemCProps*:
assumes *IOb m*
and *v = c m*
shows $(v > 0) \wedge (\forall x y. (\exists p. (Ph\ p \wedge W\ m\ p\ x \wedge W\ m\ p\ y)))$
 $\longleftrightarrow (space2\ x\ y = (c\ m * c\ m) * time2\ x\ y))$
<proof>

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>

lemma *lemCPos*:
assumes *IOb m*
shows *c m > 0*
<proof>

lemma *lemCPhoton*:
assumes *IOb m*
shows $\forall x y. (\exists p. (Ph\ p \wedge W\ m\ p\ x \wedge W\ m\ p\ y)) \longleftrightarrow (space2\ x\ y = (c\ m * c\ m) * (time2\ x\ y))$
<proof>

end

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

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

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

$$))$$

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 $z - e = g - e + (z - g)$
 $\langle proof \rangle$

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 $space2\ e\ f \leq (c\ m * c\ m) * time2\ e\ f$
 $\langle proof \rangle$

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.