No Faster-Than-Light Observers (GenRel)

Mike Stannett*, Edward Higgins
University of Sheffield, UK

Hajnal Andréka, Judit Madarász, István Németi, Gergely Székely†
Alfréd Rényi Institute of Mathematics, Budapest, Hungary
(† Secondary affiliation: University of Public Service, Budapest, Hungary)

September 13, 2023

Abstract

We have previously verified, in the first order theory SpecRel of Special Relativity, that inertial observers cannot travel faster than light [1, 2]. We now prove the corresponding result for GenRel, the first-order theory of General Relativity. Specifically, we prove that whenever an observer m encounters another observer k (so that m and k are both present at some spacetime location x), k will necessarily be observed by m to be traveling at less than light speed.

Contents

*Corresponding author: m.stannett@sheffield.ac.uk

1 Sorts

GenRel is a 2-sorted first-order logic. This theory introduces the two sorts and proves a number of basic arithmetical results. The two sorts are Bodies (things that can move) and Quantities (used to specify coordinates, masses, etc).

theory *Sorts*
imports *Main*
begin

1.1 Bodies

There are two types of Body: photons and observers. We do not assume a priori that these sorts are disjoint.

record *Body* =
Ph :: *bool*
Ob :: *bool*

1.2 Quantities

The quantities are assumed to form a linearly ordered field. We may sometimes need to assume that the field is also Euclidean, i.e. that square roots exist, but this is not a general requirement so it will be added later using a separate axiom class, AxEField.

class *Quantities* = *linordered-field*
begin

abbreviation *inRangeOO* :: $'a \Rightarrow 'a \Rightarrow 'a \Rightarrow bool$ (- < - < -)
where $(a < b < c) \equiv (a < b) \wedge (b < c)$

abbreviation *inRangeOC* :: $'a \Rightarrow 'a \Rightarrow 'a \Rightarrow bool$ (- < - ≤ -)
where $(a < b \leq c) \equiv (a < b) \wedge (b \leq c)$

abbreviation *inRangeCO* :: $'a \Rightarrow 'a \Rightarrow 'a \Rightarrow bool$ (- ≤ - < -)
where $(a \leq b < c) \equiv (a \leq b) \wedge (b < c)$

abbreviation *inRangeCC* :: $'a \Rightarrow 'a \Rightarrow 'a \Rightarrow bool$ (- ≤ - ≤ -)

where $(a \leq b \leq c) \equiv (a \leq b) \wedge (b \leq c)$

lemma *lemLEPlus*: $a \leq b + c \longrightarrow c \geq a - b$
⟨*proof*⟩

lemma *lemMultPosLT1*:
assumes $(a > 0) \wedge (b \geq 0) \wedge (b < 1)$
shows $(a * b) < a$
⟨*proof*⟩

lemma *lemAbsRange*: $e > 0 \longrightarrow ((a-e) < b < (a+e)) \longleftrightarrow (abs\ (b-a) < e)$
⟨*proof*⟩

lemma *lemAbsNeg*: $abs\ x = abs\ (-x)$
⟨*proof*⟩

lemma *lemAbsNegNeg*: $abs\ (-a-b) = abs\ (a+b)$
⟨*proof*⟩

lemma *lemGENZGT*: $(x \geq 0) \wedge (x \neq 0) \longrightarrow x > 0$
⟨*proof*⟩

lemma *lemLENZLT*: $(x \leq 0) \wedge (x \neq 0) \longrightarrow x < 0$
⟨*proof*⟩

lemma *lemSumOfNonNegAndPos*: $x \geq 0 \wedge y > 0 \longrightarrow x+y > 0$
⟨*proof*⟩

lemma *lemSumOfTwoHalves*: $x = x/2 + x/2$
⟨*proof*⟩

lemma *lemDiffDiffAdd*: $(b-a)+(c-b) = (c-a)$
⟨*proof*⟩

lemma *lemSumDiffCancelMiddle*: $(a - b) + (b - c) = (a - c)$
⟨*proof*⟩

lemma *lemDiffSumCancelMiddle*: $(a - b) + (b + c) = (a + c)$
⟨*proof*⟩

lemma *lemMultPosLT*: $((0 < a) \wedge (b < c)) \longrightarrow (a*b < a*c)$
⟨*proof*⟩

lemma *lemMultPosLE*: $((0 < a) \wedge (b \leq c)) \longrightarrow (a*b \leq a*c)$
⟨*proof*⟩

lemma *lemNonNegLT*: $((0 \leq a) \wedge (b < c)) \longrightarrow (a*b \leq a*c)$
⟨*proof*⟩

lemma *lemMultNonNegLE*: $((0 \leq a) \wedge (b \leq c)) \longrightarrow (a*b \leq a*c)$
⟨*proof*⟩

abbreviation *sqr* :: $'a \Rightarrow 'a$
where $sqr\ x \equiv\ x*x$

abbreviation *hasRoot* :: $'a \Rightarrow bool$
where $hasRoot\ x \equiv \exists\ r\ .\ x = sqr\ r$

abbreviation *isNonNegRoot* :: $'a \Rightarrow 'a \Rightarrow bool$
where $isNonNegRoot\ x\ r \equiv (r \geq 0) \wedge (x = sqr\ r)$

abbreviation *hasUniqueRoot* :: $'a \Rightarrow bool$
where $hasUniqueRoot\ x \equiv \exists!\ r\ .\ isNonNegRoot\ x\ r$

abbreviation *sqrt* :: $'a \Rightarrow 'a$
where $sqrt\ x \equiv THE\ r\ .\ isNonNegRoot\ x\ r$

lemma *lemAbsIsRootOfSquare*: $isNonNegRoot\ (sqr\ x)\ (abs\ x)$
⟨*proof*⟩

lemma *lemSqrt*:
assumes *hasRoot x*
shows *hasUniqueRoot x*
⟨*proof*⟩

lemma *lemSqrMonoStrict*: **assumes** $(0 \leq u) \wedge (u < v)$
shows $(sqr\ u) < (sqr\ v)$
⟨*proof*⟩

lemma *lemSqrMono*: $(0 \leq u) \wedge (u \leq v) \longrightarrow (sqr\ u) \leq (sqr\ v)$
⟨*proof*⟩

lemma *lemSqrOrderedStrict*: $(v > 0) \wedge (sqr\ u < sqr\ v) \longrightarrow (u < v)$
⟨*proof*⟩

lemma *lemSqrOrdered*: $(v \geq 0) \wedge (sqr\ u \leq sqr\ v) \longrightarrow (u \leq v)$
⟨*proof*⟩

lemma *lemSquaredNegative*: $sqr\ x = sqr\ (-x)$
⟨*proof*⟩

lemma *lemSqrDiffSymmetrical*: $sqr\ (x - y) = sqr\ (y - x)$
⟨*proof*⟩

lemma *lemSquaresPositive*: $x \neq 0 \longrightarrow sqr\ x > 0$
⟨*proof*⟩

lemma *lemZeroRoot*: $(sqr\ x = 0) \longleftrightarrow (x = 0)$
⟨*proof*⟩

lemma *lemSqrMult*: $sqr\ (a * b) = (sqr\ a) * (sqr\ b)$
⟨*proof*⟩

lemma *lemEqualSquares*: $sqr\ u = sqr\ v \longrightarrow abs\ u = abs\ v$
⟨*proof*⟩

lemma *lemSqrtOfSquare*:
assumes $b = sqr\ a$
shows $sqrt\ b = abs\ a$
⟨*proof*⟩

lemma *lemSquareOfSqrt*:
assumes $hasRoot\ b$
and $a = sqrt\ b$
shows $sqr\ a = b$
⟨*proof*⟩

lemma *lemSqrt1*: $sqrt\ 1 = 1$
⟨*proof*⟩

lemma *lemSqrt0*: $sqrt\ 0 = 0$
⟨*proof*⟩

lemma *lemSqrSum*: $sqr\ (x + y) = (x*x) + (2*x*y) + (y*y)$
⟨*proof*⟩

lemma *lemQuadraticGEZero*:
assumes $\forall\ x.\ a*(sqr\ x) + b*x + c \geq 0$
and $a > 0$
shows $(sqr\ b) \leq 4*a*c$

⟨*proof*⟩

lemma *lemSquareExistsAbove*:
shows $\exists\ x > 0\ .\ (sqr\ x) > y$
⟨*proof*⟩

lemma *lemSmallSquares*:
assumes $x > 0$
shows $\exists\ y > 0.\ (sqr\ y < x)$
⟨*proof*⟩

lemma *lemSqrLT1*:
assumes $0 < x < 1$
shows $0 < (sqr\ x) < x$
⟨*proof*⟩

lemma *lemReducedBound*:
assumes $x > 0$
shows $\exists\ y > 0\ .\ (y < x) \wedge (sqr\ y < y) \wedge (y < 1)$
⟨*proof*⟩

end

end

2 Points

This theory defines (1+3)-dimensional spacetime points. The first coordinate is the time coordinate, and the remaining three coordinates give the spatial component.

theory *Points*
imports *Sorts*
begin

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

record *'a Space* =
  svalx :: *'a*
  svaly :: *'a*
  svalz :: *'a*

abbreviation *tComponent* :: *'a Point* ⇒ *'a* **where**
  tComponent p ≡ *tval p*

abbreviation *sComponent* :: *'a Point* ⇒ *'a Space* **where**
  sComponent p ≡ (| *svalx* = *xval p*, *svaly* = *yval p*, *svalz* = *zval p* |)

abbreviation *mkPoint* :: *'a* ⇒ *'a* ⇒ *'a* ⇒ *'a* ⇒ *'a Point* **where**
  mkPoint t x y z ≡ (| *tval* = *t*, *xval* = *x*, *yval* =*y*, *zval* = *z* |)

abbreviation *stPoint* :: *'a* ⇒ *'a Space* ⇒ *'a Point* **where**
  stPoint t s ≡ *mkPoint t* (*svalx s*) (*svaly s*) (*svalz s*)

abbreviation *mkSpace* :: *'a* ⇒ *'a* ⇒ *'a* ⇒ *'a Space* **where**
  mkSpace x y z ≡ (| *svalx* = *x*, *svaly* =*y*, *svalz* = *z* |)

Points have coordinates in the field of quantities, and can be thought of as the end-points of vectors pinned to the origin. We can translate and scale them, define accumulation points, etc.

class *Points* = *Quantities*
begin

abbreviation *moveBy* :: *'a Point* ⇒ *'a Point* ⇒ *'a Point* (- ⊕ -)
where
(*p* ⊕ *q*) ≡ (| *tval* = *tval p* + *tval q*,
  xval = *xval p* + *xval q*,
  yval = *yval p* + *yval q*,
  zval = *zval p* + *zval q* |)

abbreviation *movebackBy* :: *'a Point* ⇒ *'a Point* ⇒ *'a Point* (- ⊖ -)
where
(*p* ⊖ *q*) ≡ (| *tval* = *tval p* − *tval q*,
  xval = *xval p* − *xval q*,
  yval = *yval p* − *yval q*,
  zval = *zval p* − *zval q* |)

abbreviation *sMoveBy* :: *'a Space* ⇒ *'a Space* ⇒ *'a Space* (- ⊕*s* -)
where
(*p* ⊕*s q*) ≡ (| *svalx* = *svalx p* + *svalx q*,
  svaly = *svaly p* + *svaly q*,

$$svalz = svalz\ p + svalz\ q\ \|)$$

abbreviation *sMovebackBy* :: *'a Space* ⇒ *'a Space* ⇒ *'a Space* (- ⊖*s* -) **where**
$(p \ominus s\ q) \equiv (\|\ svalx = svalx\ p - svalx\ q,$
$svaly = svaly\ p - svaly\ q,$
$svalz = svalz\ p - svalz\ q\ \|)$

abbreviation *scaleBy* :: *'a* ⇒ *'a Point* ⇒ *'a Point* (- ⊗ -) **where**
scaleBy a p ≡ (‖ *tval* = *a*∗*tval p*, *xval* = *a*∗*xval p*,
yval = *a*∗*yval p*, *zval* = *a*∗*zval p*‖)

abbreviation *sScaleBy* :: *'a* ⇒ *'a Space* ⇒ *'a Space* (- ⊗*s* -) **where**
sScaleBy a p ≡ (‖ *svalx* = *a*∗*svalx p*,
svaly = *a*∗*svaly p*,
svalz = *a*∗*svalz p*‖)

abbreviation *sOrigin* :: *'a Space* **where**
sOrigin ≡ (‖ *svalx* = *0*, *svaly* = *0*, *svalz* = *0* ‖)

abbreviation *origin* :: *'a Point* **where**
origin ≡ (‖ *tval* = *0*, *xval* = *0*, *yval* = *0*, *zval* = *0* ‖)

abbreviation *tUnit* :: *'a Point* **where**
tUnit ≡ (‖ *tval* = *1*, *xval* = *0*, *yval* = *0*, *zval* = *0* ‖)

abbreviation *xUnit* :: *'a Point* **where**
xUnit ≡ (‖ *tval* = *0*, *xval* = *1*, *yval* = *0*, *zval* = *0* ‖)

abbreviation *yUnit* :: *'a Point* **where**
yUnit ≡ (‖ *tval* = *0*, *xval* = *0*, *yval* = *1*, *zval* = *0* ‖)

abbreviation *zUnit* :: *'a Point* **where**
zUnit ≡ (‖ *tval* = *0*, *xval* = *0*, *yval* = *0*, *zval* = *1* ‖)

abbreviation *timeAxis* :: *'a Point set* **where**
timeAxis ≡ { *p* . *xval p* = *0* ∧ *yval p* = *0* ∧ *zval p* = *0* }

abbreviation *onTimeAxis* :: *'a Point* ⇒ *bool*
where *onTimeAxis p* ≡ (*p* ∈ *timeAxis*)

2.1 Squared norms and separation functions

This theory defines squared norms and separations. We do not yet define unsquared norms because we are not assuming here that quantities necessarily have square roots.

abbreviation *norm2* :: *'a Point* ⇒ *'a* **where**
norm2 p ≡ *sqr* (*tval p*) + *sqr* (*xval p*) + *sqr* (*yval p*) + *sqr* (*zval p*)

abbreviation *sep2* :: *'a Point* ⇒ *'a Point* ⇒ *'a* **where**
sep2 p q ≡ *norm2* (*p* ⊖ *q*)

abbreviation *sNorm2* :: *'a Space* ⇒ *'a* **where**
sNorm2 s ≡ *sqr* (*svalx s*)
+ *sqr* (*svaly s*)
+ *sqr* (*svalz s*)

abbreviation *sSep2* :: *'a Point* ⇒ *'a Point* ⇒ *'a* **where**
sSep2 p q ≡ *sqr* (*xval p* − *xval q*)
+ *sqr* (*yval p* − *yval q*)
+ *sqr* (*zval p* − *zval q*)

abbreviation *mNorm2* :: *'a Point* ⇒ *'a* (‖ - ‖*m*)
where ‖ *p* ‖*m* ≡ *sqr* (*tval p*) − *sNorm2* (*sComponent p*)

2.2 Topological concepts

We will need to define topological concepts like continuity and affine approximation later, so here we define open balls and accumulation points.

abbreviation *inBall* :: *'a Point* ⇒ *'a* ⇒ *'a Point* ⇒ *bool*
(- *within* - *of* -)
where *inBall q ε p* ≡ *sep2 q p* < *sqr ε*

abbreviation *ball* :: *'a Point* ⇒ *'a* ⇒ *'a Point set*
where *ball q ε* ≡ { *p* . *inBall q ε p* }

abbreviation *accPoint* :: *'a Point* ⇒ *'a Point set* ⇒ *bool*
where *accPoint p s* ≡ ∀ *ε* > *0*. ∃ *q* ∈ *s*. (*p* ≠ *q*) ∧ (*inBall q ε p*)

2.3 Lines

A line is specified by giving a point on the line, and a point (thought of as a vector) giving its direction. For these purposes it doesn't matter whether the direction is "positive" or "negative".

abbreviation *line* :: *'a Point* ⇒ *'a Point* ⇒ *'a Point set*
where *line base drtn* ≡ { *p* . ∃ *α* . *p* = (*base* ⊕ (*α*⊗*drtn*)) }

abbreviation *lineJoining* :: *'a Point* ⇒ *'a Point* ⇒ *'a Point set*
  where *lineJoining p q* ≡ *line p* (*q*⊖*p*)

abbreviation *isLine* :: *'a Point set* ⇒ *bool*
  where *isLine l* ≡ ∃ *b d* . (*l* = *line b d*)

abbreviation *sameLine* :: *'a Point set* ⇒ *'a Point set* ⇒ *bool*
  where *sameLine l1 l2* ≡ ((*isLine l1*) ∨ (*isLine l2*)) ∧ (*l1* = *l2*)

abbreviation *onLine* :: *'a Point* ⇒ *'a Point set* ⇒ *bool*
  where *onLine p l* ≡ (*isLine l*) ∧ (*p* ∈ *l*)

2.4 Directions

Given any two distinct points on a line, the vector joining them can be used to specify the line's direction. The direction of a line is therefore a *set* of points/vectors. By lemDrtn these are all parallel

fun *drtn* :: *'a Point set* ⇒ *'a Point set*
  where *drtn l* = { *d* . ∃ *p q* . (*p* ≠ *q*) ∧ (*onLine p l*) ∧ (*onLine q l*) ∧ (*d* = (*q* ⊖ *p*)) }

abbreviation *parallelLines* :: *'a Point set* ⇒ *'a Point set* ⇒ *bool*
  where *parallelLines l1 l2* ≡ (*drtn l1*) ∩ (*drtn l2*) ≠ {}

abbreviation *parallel* :: *'a Point* ⇒ *'a Point* ⇒ *bool* (- ∥ -)
  where *parallel p q* ≡ (∃ *α* ≠ *0* . *p* = (*α* ⊗ *q*))

The "slope" of a line can be either finite or infinite. We will often need to consider these two cases separately.

abbreviation *slopeFinite* :: *'a Point* ⇒ *'a Point* ⇒ *bool*
  where *slopeFinite p q* ≡ (*tval p* ≠ *tval q*)

abbreviation *slopeInfinite* :: *'a Point* ⇒ *'a Point* ⇒ *bool*
  where *slopeInfinite p q* ≡ (*tval p* = *tval q*)

abbreviation *lineSlopeFinite* :: *'a Point set* ⇒ *bool*
  where *lineSlopeFinite l* ≡ (∃ *x y* . (*onLine x l*) ∧ (*onLine y l*)
    ∧ (*x* ≠ *y*) ∧ (*slopeFinite x y*))

2.5 Slopes and slopers

We specify the slope of a line by giving the spatial component ("sloper") of the point on the line at time 1. This is defined if and only if the slope is finite. If the slope is infinite (the

line is "horizontal") we return the spatial origin. This avoids using "option" but means we need to consider carefully whether a sloper with value sOrigin indicates a truly zero slope or an infinite one.

fun *sloper* :: *'a Point* ⇒ *'a Point* ⇒ *'a Point*
where *sloper p q* = (*if* (*slopeFinite p q*) *then* ((*1* / (*tval p* − *tval q*)) ⊗ (*p* ⊖ *q*))
else origin)

fun *velocityJoining* :: *'a Point* ⇒ *'a Point* ⇒ *'a Space*
where *velocityJoining p q* = *sComponent* (*sloper p q*)

fun *lineVelocity* :: *'a Point set* ⇒ *'a Space set*
where *lineVelocity l* = { *v* . ∃ *d* ∈ *drtn l* . *v* = *velocityJoining origin d* }

lemma *lemNorm2Decomposition*:
shows *norm2 u* = *sqr* (*tval u*) + *sNorm2* (*sComponent u*)
⟨*proof*⟩

lemma *lemPointDecomposition*:
shows *p* = (((*tval p*)⊗*tUnit*) ⊕ (((*xval p*)⊗*xUnit*)
⊕ (((*yval p*)⊗*yUnit*) ⊕ ((*zval p*)⊗*zUnit*))))
⟨*proof*⟩

lemma *lemScaleLeftSumDistrib*: ((*a* + *b*) ⊗ *p*) = ((*a*⊗*p*) ⊕ (*b*⊗*p*))
⟨*proof*⟩

lemma *lemScaleLeftDiffDistrib*: ((*a* − *b*) ⊗ *p*) = ((*a*⊗*p*) ⊖ (*b*⊗*p*))
⟨*proof*⟩

lemma *lemScaleAssoc*: (*α* ⊗ (*β* ⊗ *p*)) = ((*α* ∗ *β*) ⊗ *p*)
⟨*proof*⟩

lemma *lemScaleCommute*: $(\alpha \otimes (\beta \otimes p)) = (\beta \otimes (\alpha \otimes p))$
⟨*proof*⟩

lemma *lemScaleDistribSum*: $(\alpha \otimes (p \oplus q)) = ((\alpha \otimes p) \oplus (\alpha \otimes q))$
⟨*proof*⟩

lemma *lemScaleDistribDiff*: $(\alpha \otimes (p \ominus q)) = ((\alpha \otimes p) \ominus (\alpha \otimes q))$
⟨*proof*⟩

lemma *lemScaleOrigin*: $(\alpha \otimes origin) = origin$
⟨*proof*⟩

lemma *lemMNorm2OfScaled*: $mNorm2\ (scaleBy\ \alpha\ p) = (sqr\ \alpha) * mNorm2\ p$
⟨*proof*⟩

lemma *lemSNorm2OfScaled*: $sNorm2\ (sScaleBy\ \alpha\ p) = (sqr\ \alpha) * sNorm2\ p$
⟨*proof*⟩

lemma *lemNorm2OfScaled*: $norm2\ (\alpha \otimes p) = (sqr\ \alpha) * norm2\ p$
⟨*proof*⟩

lemma *lemScaleSep2*: $(sqr\ a) * (sep2\ p\ q) = sep2\ (a \otimes p)\ (a \otimes q)$
⟨*proof*⟩

lemma *lemSScaleAssoc*: $(\alpha \otimes s\ (\beta \otimes s\ p)) = ((\alpha * \beta) \otimes s\ p)$
⟨*proof*⟩

lemma *lemScaleBall*:
assumes $x\ within\ e\ of\ y$
and $a \neq 0$
shows $(a \otimes x)\ within\ (a * e)\ of\ (a \otimes y)$
⟨*proof*⟩

lemma *lemScaleBallAndBoundary*:
assumes $sep2\ x\ y \leq sqr\ e$
and $a \neq 0$
shows $sep2\ (a \otimes x)\ (a \otimes y) \leq sqr\ (a * e)$
⟨*proof*⟩

lemma *lemTimeAxisIsLine*: *isLine timeAxis*
⟨*proof*⟩

lemma *lemSameLine*:
assumes $p \in line\ b\ d$
shows *sameLine* (*line b d*) (*line p d*)
⟨*proof*⟩

lemma *lemSSep2Symmetry*: $sSep2\ p\ q = sSep2\ q\ p$
⟨*proof*⟩

lemma *lemSep2Symmetry*: $sep2\ p\ q = sep2\ q\ p$
⟨*proof*⟩

lemma *lemSpatialNullImpliesSpatialOrigin*:
assumes $sNorm2\ s = 0$
shows $s = sOrigin$
⟨*proof*⟩

lemma *lemNorm2NonNeg*: $norm2\ p \geq 0$
⟨*proof*⟩

lemma *lemNullImpliesOrigin*:
assumes $norm2\ p = 0$
shows $p = origin$
⟨*proof*⟩

lemma *lemNotOriginImpliesPosNorm2*:
assumes $p \neq origin$
shows $norm2\ p > 0$
⟨*proof*⟩

lemma *lemNotEqualImpliesSep2Pos*:
assumes $y \neq x$
shows $sep2\ y\ x > 0$
⟨*proof*⟩

lemma *lemBallContainsCentre*:
assumes $\varepsilon > 0$
shows x *within* ε *of* x
⟨*proof*⟩

lemma *lemPointLimit*:
assumes $\forall\ \varepsilon > 0\ .\ (v$ *within* ε *of* $u)$
shows $v = u$
⟨*proof*⟩

lemma *lemBallPopulated*:
assumes $e > 0$
shows $\exists\ y\ .\ (y$ *within* e *of* $x) \wedge (y \neq x)$
⟨*proof*⟩

lemma *lemBallInBall*:
assumes p *within* x *of* q
and $0 < x \leq y$
shows p *within* y *of* q
⟨*proof*⟩

lemma *lemSmallPoints*:
assumes $e > 0$
shows $\exists\ a > 0\ .\ norm2\ (a \otimes p) < sqr\ e$
⟨*proof*⟩

lemma *lemLineJoiningContainsEndPoints*:
assumes $l = lineJoining\ x\ p$
shows $onLine\ x\ l \wedge onLine\ p\ l$
⟨*proof*⟩

lemma *lemLineAndPoints*:
assumes $p \neq q$
shows $(onLine\ p\ l \wedge onLine\ q\ l) \longleftrightarrow (l = lineJoining\ p\ q)$
⟨*proof*⟩

lemma *lemLineDefinedByPair*:
assumes $x \neq p$
and $(onLine\ p\ l1) \wedge (onLine\ x\ l1)$

and $(onLine\ p\ l2) \wedge (onLine\ x\ l2)$
shows $l1 = l2$
⟨*proof*⟩

lemma *lemDrtn*:
assumes $\{\ d1,\ d2\ \} \subseteq drtn\ l$
shows $\exists\ \alpha \neq 0\ .\ d2 = (\alpha \otimes d1)$
⟨*proof*⟩

lemma *lemLineDeterminedByPointAndDrtn*:
assumes $(x \neq p) \wedge (p \in l1) \wedge (onLine\ x\ l1) \wedge (onLine\ x\ l2)$
and $drtn\ l1 = drtn\ l2$
shows $l1 = l2$
⟨*proof*⟩

end

end

3 WorldView

This theory defines worldview transformations. These form the ultimate foundation for all of GenRel's axioms.

theory *WorldView*
imports *Points*
begin

class *WorldView* = *Points* +
fixes

$W :: Body \Rightarrow Body \Rightarrow 'a\ Point \Rightarrow bool$ (- *sees* - *at* -)
begin

abbreviation $ev :: Body \Rightarrow 'a\ Point \Rightarrow Body\ set$
where $ev\ h\ x \equiv \{\ b\ .\ h\ sees\ b\ at\ x\ \}$

fun $wvt :: Body \Rightarrow Body \Rightarrow 'a\ Point \Rightarrow 'a\ Point\ set$
where $wvt\ m\ k\ p = \{\ q.\ (\exists\ b\ .\ (m\ sees\ b\ at\ p)) \wedge (ev\ m\ p = ev\ k\ q)$
}

abbreviation $wvtFunc :: Body \Rightarrow Body \Rightarrow ('a\ Point \Rightarrow 'a\ Point \Rightarrow bool)$
where $wvtFunc\ m\ k \equiv (\lambda\ p\ q\ .\ q \in wvt\ m\ k\ p)$

abbreviation *wvtLine* :: *Body* ⇒ *Body* ⇒ *'a Point set* ⇒ *'a Point set* ⇒ *bool*
  where *wvtLine m k l l'* ≡ ∃ *p q p' q'* . (*wvtFunc m k p p'*) ∧ (*wvtFunc m k q q'*) ∧ (*l* = *lineJoining p q*) ∧ (*l'* = *lineJoining p' q'*))

end

end

4 Functions

This theory characterises the various types of function (injective, bijective, etc).

theory *Functions*
  imports *Points*
begin

We do not assume a priori that all of the functions we define are well-defined or total. We therefore need to allow for functions which are only partially defined, and also for "functions" which might be multi-valued. For example, we cannot say in advance whether one observer might see another's worldline as a bifurcating structure rather than a basic single-valued trajectory.

To achieve this we'll often think of functions as relations and write "f x y = true" instead of "f x = y". Similarly, a spacetime set S will be sometimes be expressed as its characteristic function.

class *Functions* = *Points*
begin

abbreviation *bounded* :: (*'a Point* ⇒ *'a Point*) ⇒ *bool*
  where *bounded f* ≡ ∃ *bnd* > *0* . (∀ *p* . (*norm2* (*f p*) ≤ *bnd* ∗ (*norm2 p*)))

abbreviation *composeRel* ::
(*'a Point* ⇒ *'a Point* ⇒ *bool*)
⇒(*'a Point* ⇒ *'a Point* ⇒ *bool*)
⇒(*'a Point* ⇒ *'a Point* ⇒ *bool*)
  where (*composeRel g f*) *p r* ≡ (∃ *q* . ((*f p q*) ∧ (*g q r*)))

abbreviation *injective* :: (*'a Point* ⇒ *'a Point* ⇒ *bool*) ⇒ *bool*
  where *injective f* ≡ ∀ *x1 x2 y1 y2*.
        (*f x1 y1* ∧ *f x2 y2*) ∧ (*x1* ≠ *x2*) ⟶ (*y1* ≠ *y2*)

abbreviation *definedAt* :: (*'a Point* ⇒ *'a Point* ⇒ *bool*) ⇒ *'a Point* ⇒ *bool*
  where *definedAt f x* ≡ ∃ *y* . *f x y*

abbreviation *domain* :: (*'a Point* => *'a Point* ⇒ *bool*) ⇒ *'a Point set*
  where *domain f* ≡ { *x* . *definedAt f x* }

abbreviation *total* :: (*'a Point* ⇒ *'a Point* ⇒ *bool*) ⇒ *bool*
  where *total f* ≡ ∀ *x* . (*definedAt f x*)

abbreviation *surjective* :: (*'a Point* ⇒ *'a Point* ⇒ *bool*) ⇒ *bool*
  where *surjective f* ≡ ∀ *y* . ∃ *x* . *f x y*

abbreviation *bijective* :: (*'a Point* ⇒ *'a Point* ⇒ *bool*) ⇒ *bool*
  where *bijective f* ≡ (*injective f*) ∧ (*surjective f*)

abbreviation *invertible* :: (*'a Point* ⇒ *'a Point*) ⇒ *bool*
  where *invertible f* ≡ ∀ *q* . (∃ *p* . (*f p* = *q*) ∧ (∀ *x*. *f x* = *q* ⟶ *x* = *p*))

fun *applyToSet* :: (*'a Point* ⇒ *'a Point* ⇒ *bool*) ⇒ *'a Point set* ⇒ *'a Point set*
  where *applyToSet f s* = { *q* . ∃ *p* ∈ *s* . *f p q* }

abbreviation *singleValued* :: (*'a Point* ⇒ *'a Point* ⇒ *bool*) ⇒ *'a Point* ⇒ *bool*
  where *singleValued f x* ≡ ∀ *y z* . (((*f x y*) ∧ (*f x z*)) ⟶ (*y* = *z*))

abbreviation *isFunction* :: (*'a Point* ⇒ *'a Point* ⇒ *bool*) ⇒ *bool*
  where *isFunction f* ≡ ∀ *x* . *singleValued f x*

abbreviation *isTotalFunction* :: (*'a Point* ⇒ *'a Point* ⇒ *bool*) ⇒ *bool*
  where *isTotalFunction f* ≡ (*total f*) ∧ (*isFunction f*)

fun *toFunc*:: (*'a Point* ⇒ *'a Point* ⇒ *bool*) ⇒ *'a Point* ⇒ *'a Point*
  where *toFunc f x* = (*SOME y* . *f x y*)

fun *asFunc* :: (*'a Point* ⇒ *'a Point*) ⇒ (*'a Point* ⇒ *'a Point* ⇒ *bool*)
  where (*asFunc f*) *x y* = (*y* = *f x*)

4.1 Differentiable approximation

Here we define differentiable approximation. This will be used later when we define what it means for a worldview transformation to be "approximated" by an affine transformation.

abbreviation *diffApprox* :: (*'a Point* ⇒ *'a Point* ⇒ *bool*) ⇒
(*'a Point* ⇒ *'a Point* ⇒ *bool*) ⇒
'a Point ⇒ *bool*
where *diffApprox g f x* ≡ (*definedAt f x*) ∧
(∀ *ε* > *0* . (∃ *δ* > *0* . (∀ *y* .
((*y within δ of x*)
⟶
((*definedAt f y*) ∧ (∀ *u v* . (*f y u* ∧ *g y v*) ⟶
(*sep2 v u*) ≤ (*sqr ε*) ∗ *sep2 y x*))))
))

abbreviation *cts* :: (*'a Point* ⇒ *'a Point* ⇒ *bool*) ⇒ *'a Point* ⇒ *bool*
where *cts f x* ≡ ∀ *y* . (*f x y*) ⟶ (∀*ε*>*0*. ∃*δ*>*0*.
(*applyToSet f* (*ball x δ*)) ⊆ *ball y ε*)

fun *invFunc* :: (*'a Point* ⇒ *'a Point* ⇒ *bool*) ⇒ (*'a Point* ⇒ *'a Point* ⇒ *bool*)
where (*invFunc f*) *p q* = *f q p*

lemma *lemBijInv*: *bijective* (*asFunc f*) ⟷ *invertible f*
⟨*proof*⟩

4.2 lemApproxEqualAtBase

The following lemma shows (as one would expect) that when one function differentiably approximates another at a point, they take equal values at that point.

lemma *lemApproxEqualAtBase*:
assumes *diffApprox g f x*
shows (*f x y* ∧ *g x z*) ⟶ (*y* = *z*)
⟨*proof*⟩

lemma *lemCtsOfCtsIsCts*:
assumes *cts f x*
and ∀ *y* . (*f x y*) ⟶ (*cts g y*)
shows *cts* (*composeRel g f*) *x*

⟨*proof*⟩

lemma *lemInjOfInjIsInj*:
assumes *injective f*
and *injective g*
shows *injective* (*composeRel g f*)
⟨*proof*⟩

lemma *lemInverseComposition*:
assumes $h = composeRel\ g\ f$
shows $(invFunc\ h) = composeRel\ (invFunc\ f)\ (invFunc\ g)$
⟨*proof*⟩

lemma *lemToFuncAsFunc*:
assumes *isFunction f*
and *total f*
shows $asFunc\ (toFunc\ f) = f$
⟨*proof*⟩

lemma *lemAsFuncToFunc*: $toFunc\ (asFunc\ f) = f$
⟨*proof*⟩

end

end

5 WorldLine

This theory defines worldlines.

theory *WorldLine*
imports *WorldView Functions*
begin

class $WorldLine = WorldView + Functions$
begin

abbreviation $wline :: Body \Rightarrow Body \Rightarrow 'a\ Point\ set$
where $wline\ m\ k \equiv \{\ p\ .\ m\ sees\ k\ at\ p\ \}$

lemma *lemWorldLineUnderWVT*:
shows *applyToSet* (*wvtFunc m k*) (*wline m b*) ⊆ *wline k b*
⟨*proof*⟩

lemma *lemFiniteLineVelocityUnique*:
assumes (*u* ∈ *lineVelocity l*) ∧ (*v* ∈ *lineVelocity l*)
and *lineSlopeFinite l*
shows *u* = *v*
⟨*proof*⟩

end

end

6 Translations

This theory describes translation maps.

theory *Translations*
imports *Functions*
begin

class *Translations* = *Functions*
begin

abbreviation *mkTranslation* :: *'a Point* ⇒ (*'a Point* ⇒ *'a Point*)
where (*mkTranslation t*) ≡ (λ *p* . (*p* ⊕ *t*))

abbreviation *translation* :: (*'a Point* ⇒ *'a Point*) ⇒ *bool*
where *translation T* ≡ ∃ *q* . ∀ *p* . ((*T p*) = (*p* ⊕ *q*))

lemma *lemMkTrans*: ∀ *t* . *translation* (*mkTranslation t*)
⟨*proof*⟩

lemma *lemInverseTranslation*:
assumes (*T* = *mkTranslation t*) ∧ (*T'* = *mkTranslation* (*origin* ⊖ *t*))
shows (*T'* ∘ *T* = *id*) ∧ (*T* ∘ *T'* = *id*)
⟨*proof*⟩

lemma *lemTranslationSum*:
assumes *translation T*
shows $T\ (u \oplus v) = ((T\ u) \oplus v)$
⟨*proof*⟩

lemma *lemIdIsTranslation*: *translation id*
⟨*proof*⟩

lemma *lemTranslationCancel*:
assumes *translation T*
shows $((T\ p) \ominus (T\ q)) = (p \ominus q)$
⟨*proof*⟩

lemma *lemTranslationSwap*:
assumes *translation T*
shows $(p \oplus (T\ q)) = ((T\ p) \oplus q)$
⟨*proof*⟩

lemma *lemTranslationPreservesSep2*:
assumes *translation T*
shows *sep2 p q* = *sep2* $(T\ p)\ (T\ q)$
⟨*proof*⟩

lemma *lemTranslationInjective*:
assumes *translation T*
shows *injective* (*asFunc T*)
⟨*proof*⟩

lemma *lemTranslationSurjective*:
assumes *translation T*
shows *surjective* (*asFunc T*)
⟨*proof*⟩

lemma *lemTranslationTotalFunction*:
assumes *translation T*
shows *isTotalFunction* (*asFunc T*)
⟨*proof*⟩

lemma *lemTranslationOfLine*:
assumes *translation T*
shows (*applyToSet* (*asFunc T*) (*line B D*)) = *line* (*T B*) *D*
⟨*proof*⟩

lemma *lemOnLineTranslation*:
assumes (*translation T*) ∧ (*onLine p l*)
shows *onLine* (*T p*) (*applyToSet* (*asFunc T*) *l*)
⟨*proof*⟩

lemma *lemLineJoiningTranslation*:
assumes *translation T*
shows *applyToSet* (*asFunc T*) (*lineJoining p q*) = *lineJoining* (*T p*) (*T q*)
⟨*proof*⟩

lemma *lemBallTranslation*:
assumes *translation T*
and *x within e of y*
shows (*T x*) *within e of* (*T y*)
⟨*proof*⟩

lemma *lemBallTranslationWithBoundary*:
assumes *translation T*
and *sep2 x y* ≤ *sqr e*
shows *sep2* (*T x*) (*T y*) ≤ *sqr e*
⟨*proof*⟩

lemma *lemTranslationIsCts*:
assumes *translation T*
shows *cts* (*asFunc T*) *x*
⟨*proof*⟩

lemma *lemAccPointTranslation*:
assumes *translation T*
and *accPoint x s*

shows *accPoint* (*T x*) (*applyToSet* (*asFunc T*) *s*)
⟨*proof*⟩

lemma *lemInverseOfTransIsTrans*:
assumes *translation T*
and $T' = invFunc\ (asFunc\ T)$
shows *translation* (*toFunc T′*)
⟨*proof*⟩

lemma *lemInverseTrans*:
assumes *translation T*
shows $\exists\ T'\ .\ (translation\ T') \wedge (\forall\ p\ q\ .\ T\ p = q \longleftrightarrow T'\ q = p)$
⟨*proof*⟩

end

end

7 AXIOM: AxSelfMinus

This theory declares the axiom AxSelfMinus.

theory *AxSelfMinus*
imports *WorldView*
begin

AxSelfMinus: The worldline of an observer is a subset of the time axis in their own worldview.

class *axSelfMinus* = *WorldView*
begin
abbreviation *axSelfMinus* :: *Body* ⇒ *'a Point* ⇒ *bool*
where $axSelfMinus\ m\ p \equiv (m\ sees\ m\ at\ p) \longrightarrow onTimeAxis\ p$
end

class *AxSelfMinus* = *axSelfMinus* +
assumes *AxSelfMinus* : $\forall\ m\ p\ .\ axSelfMinus\ m\ p$
begin
end

end

8 TangentLines

This theory defines tangent lines and establishes their key properties.

theory *TangentLines*
  imports *Translations AxSelfMinus*
begin

At each point along the worldline of a body, we can ask what its instantaneous direction of motion is. Unfortunately we do not know a priori that the "worldline" actually has tangents. Dealing with tangent lines is one of the more complicated aspects of the main proof.

class *TangentLines* = *Translations* + *AxSelfMinus*
begin

abbreviation *tangentLine* :: *'a Point set* ⇒ *'a Point set* ⇒ *'a Point* ⇒ *bool*
  where *tangentLine l s x* ≡
   $(x \in s) \wedge (onLine\ x\ l) \wedge (accPoint\ x\ s)$
 ∧
   $(\exists\ p\ .\ (\ (onLine\ p\ l) \wedge (p \neq x) \wedge$
      $(\forall\ \varepsilon > 0\ .\ \exists\ \delta > 0\ .\ \forall\ y \in s.\ ($
        $(\ (y\ within\ \delta\ of\ x) \wedge (y \neq x)\ )$
        $\longrightarrow$
        $(\ \exists\ r\ .\ ((onLine\ r\ (lineJoining\ x\ y)) \wedge (r\ within\ \varepsilon\ of\ p))))$
)
   ))

abbreviation *tangentLineA* :: *'a Point set* ⇒ *'a Point set* ⇒ *'a Point* ⇒ *bool*
  where *tangentLineA l s x* ≡
   $(x \in s) \wedge (onLine\ x\ l) \wedge (accPoint\ x\ s)$
 ∧
   $(\forall\ p\ .\ (\ ((onLine\ p\ l) \wedge (p \neq x)) \longrightarrow$
      $(\forall\ \varepsilon > 0\ .\ \exists\ \delta > 0\ .\ \forall\ y \in s.\ ($
        $(\ (y\ within\ \delta\ of\ x) \wedge (y \neq x)\ )$
        $\longrightarrow$
        $(\ \exists\ r\ .\ ((onLine\ r\ (lineJoining\ x\ y)) \wedge (r\ within\ \varepsilon\ of\ p))))$
)
   ))

abbreviation *hasTangent* :: *'a Point set* ⇒ *'a Point* ⇒ *bool*
  where $hasTangent\ s\ p \equiv \exists\ l\ .\ tangentLine\ l\ s\ p$

The instantaneous velocity of a body is defined to be the velocity

of a co-moving body moving along the tangent line (assuming a tangent line exists).

fun *vel* :: *'a Point set* ⇒ *'a Point* ⇒ *'a Space* ⇒ *bool*
where *vel wl p v* = (∃ *l* . ((*tangentLine l wl p*) ∧ (*v* ∈ *lineVelocity l*)))

lemma *lemTangentLineTranslation*:
assumes *translation T*
and *tangentLine l s x*
shows *tangentLine* (*applyToSet* (*asFunc T*) *l*)
(*applyToSet* (*asFunc T*) *s*) (*T x*)
⟨*proof*⟩

lemma *lemTangentLineA*:
assumes *tangentLine l s x*
shows *tangentLineA l s x*
⟨*proof*⟩

lemma *lemTangentLineE*:
assumes *tangentLineA l s x*
and ∃ *p* ≠ *x* . *onLine p l*
shows *tangentLine l s x*
⟨*proof*⟩

end

end

9 Cones

This theory defines (light)cones, regular cones, and their properties.

theory *Cones*
imports *WorldLine TangentLines*
begin

class *Cones* = *WorldLine* + *TangentLines*
begin

abbreviation *tl* :: *'a Point set* ⇒ *Body* ⇒ *Body* ⇒ *'a Point* ⇒ *bool*
where *tl l m b x* ≡ *tangentLine l* (*wline m b*) *x*

The cone of a body at a point comprises the set of points that lie on tangent lines of photons emitted by the body at that point.

abbreviation *cone* :: *Body* ⇒ *'a Point* ⇒ *'a Point* ⇒ *bool*
where *cone m x p*
≡ ∃ *l* . (*onLine p l*) ∧ (*onLine x l*) ∧ (∃ *ph* . *Ph ph* ∧ *tl l m ph x*)

abbreviation *regularCone* :: *'a Point* ⇒ *'a Point* ⇒ *bool*
where *regularCone x p* ≡ ∃ *l* . (*onLine p l*) ∧ (*onLine x l*)
∧ (∃ *v* ∈ *lineVelocity l* . *sNorm2 v* = *1*)

abbreviation *coneSet* :: *Body* ⇒ *'a Point* ⇒ *'a Point set*
where *coneSet m x* ≡ { *p* . *cone m x p* }

abbreviation *regularConeSet* :: *'a Point* ⇒ *'a Point set*
where *regularConeSet x* ≡ { *p* . *regularCone x p* }

end

end

10 AXIOM: AxLightMinus

This theory declares the axiom AxLightMinus.

theory *AxLightMinus*
imports *WorldLine TangentLines*
begin

AxLightMinus: If an observer sends out a light signal, then the speed of the light signal is 1 according to the observer. Moreover it is possible to send out a light signal in any direction.

class *axLightMinus* = *WorldLine* + *TangentLines*
begin

The definition of AxLightMinus used in this Isabelle proof is slightly different to the one used in the paper-based proof on which it is based. We have established elsewhere, however, that each entails the other in all relevant contexts.

abbreviation *axLightMinusOLD* :: *Body* ⇒ *'a Point* ⇒ *'a Space* ⇒ *bool*
  where *axLightMinusOLD m p v* ≡ (*m sees m at p*) ⟶ (
    (∃ *ph* . (*Ph ph* ∧ (*vel* (*wline m ph*) *p v*))) ⟷ (*sNorm2 v* = *1*)
)

abbreviation *axLightMinus* :: *Body* ⇒ *'a Point* ⇒ *'a Space* ⇒ *bool*
  where *axLightMinus m p v* ≡ (*m sees m at p*)
      ⟶ (∀ *l* . ∀ *v* ∈ *lineVelocity l* .
          (∃ *ph* . (*Ph ph* ∧ (*tangentLine l* (*wline m ph*) *p*))) ⟷ (*sNorm2 v* = *1*))

end

class *AxLightMinus* = *axLightMinus* +
  assumes *AxLightMinus*: ∀ *m p v* . *axLightMinus m p v*
begin
end

end

11 Proposition1

This theory shows that observers consider their own lightcones to be upright.

theory *Proposition1*
  imports *Cones AxLightMinus*
begin

class *Proposition1* = *Cones* + *AxLightMinus*
begin

lemma *lemProposition1*:
  assumes *x* ∈ *wline m m*
  shows *cone m x p* = *regularCone x p*
⟨*proof*⟩

end

end

12 AXIOM: AxEField

This theory defines the axiom AxEField, which states that the linearly ordered field of quantities is Euclidean, i.e. that all non-negative values have square roots in the field.

theory *AxEField*
  imports *Sorts*
begin

class *axEField* = *Quantities*
begin
  abbreviation *axEField* :: *'a* ⇒ *bool*
    where *axEField x* ≡ (*x* ≥ *0*) ⟶ *hasRoot x*
end

class *AxEField* = *axEField* +
  assumes *AxEField*: ∀ *x* . *axEField x*
begin
end

end

13 Norms

This theory defines norms, assuming that roots exist.

theory *Norms*
  imports *Points AxEField*
begin

class *Norms* = *Points* + *AxEField*
begin

abbreviation *norm* :: *'a Point* ⇒ *'a* (‖ - ‖)
  where *norm p* ≡ *sqrt* (*norm2 p*)

abbreviation *sNorm* :: *'a Space* ⇒ *'a*
  where *sNorm p* ≡ *sqrt* (*sNorm2 p*)

13.1 axTriangleInequality

Given that norms exist, we can define the triangle inequality for specific cases. This will be asserted more generally as an axiom later.

abbreviation *axTriangleInequality* :: *'a Point* ⇒ *'a Point* ⇒ *bool*
  where *axTriangleInequality p q* ≡ (*norm* (*p*⊕*q*) ≤ *norm p* + *norm q*)

lemma *lemNormSqrIsNorm2*: *norm2 p* = *sqr* (*norm p*)
⟨*proof*⟩

lemma *lemZeroNorm*:
  shows (*p* = *origin*) ⟷ (*norm p* = *0*)
⟨*proof*⟩

lemma *lemNormNonNegative*: *norm p* ≥ *0*
⟨*proof*⟩

lemma *lemNotOriginImpliesPositiveNorm*:
  assumes *p* ≠ *origin*
  shows (*norm p* > *0*)
⟨*proof*⟩

lemma *lemNormSymmetry*: *norm* (*p*⊖*q*) = *norm* (*q*⊖*p*)
⟨*proof*⟩

lemma *lemNormOfScaled*: *norm* (*α*⊗*p*) = (*abs α*) ∗ (*norm p*)
⟨*proof*⟩

lemma *lemDistancesAdd*:
  assumes *triangle*: *axTriangleInequality* (*q*⊖*p*) (*r*⊖*q*)
and *distances*: (*x* > *0*) ∧ (*y* > *0*) ∧ (*sep2 p q* < *sqr x*) ∧ (*sep2 r q* < *sqr y*)
  shows *r within* (*x*+*y*) *of p*
⟨*proof*⟩

lemma *lemDistancesAddStrictR*:

assumes *triangle*: *axTriangleInequality* $(q \ominus p)$ $(r \ominus q)$
and *distances*: $(x > 0) \wedge (y > 0) \wedge (sep2\ p\ q \leq sqr\ x) \wedge (sep2\ r\ q < sqr\ y)$
shows *r within* $(x+y)$ *of p*
⟨*proof*⟩

end

end

14 AxTriangleInequality

This theory declares the Triangle Inequality as an axiom.

theory *AxTriangleInequality*
imports *Norms*
begin

Although AxTriangleInequality can be proven rather than asserted we have left it as an axiom to illustrate the flexibility of using Isabelle for mathematical physics: well-known mathematical results can be asserted, leaving the researcher free to concentrate on the physics. We can return later to prove the mathematical results when time permits.

class *AxTriangleInequality* = *Norms* +
assumes *AxTriangleInequality*: $\forall\ p\ q\ .\ axTriangleInequality\ p\ q$
begin
end

end

15 Sublemma3

This theory establishes how closely tangent lines approximate world lines.

theory *Sublemma3*
imports *WorldLine AxTriangleInequality TangentLines*
begin

class *Sublemma3* = *WorldLine* + *AxTriangleInequality* + *TangentLines*
begin

lemma *sublemma3*:
assumes *onLine p l*
and *norm2 p = 1*
and *tangentLine l wl origin*
shows
$\forall\ \varepsilon > 0\ .\ \exists\ \delta > 0\ .\ \forall\ y\ ny\ .\ (((y\ within\ \delta\ of\ origin) \wedge (y \neq origin) \wedge (y \in wl) \wedge (norm\ y = ny)) \longrightarrow (((((1/ny)) \otimes y)\ within\ \varepsilon\ of\ p) \vee ((((-1/ny)) \otimes y)\ within\ \varepsilon\ of\ p)))$
⟨*proof*⟩

lemma *sublemma3Translation*:
assumes *onLine p l*
and *norm2 (p⊖x) = 1*
and *tangentLine l wl x*
shows $\forall\ \varepsilon > 0\ .\ \exists\ \delta > 0\ .\ \forall\ y\ nyx\ .\ ((y\ within\ \delta\ of\ x) \wedge (y \neq x) \wedge (y \in wl) \wedge (norm\ (y \ominus x) = nyx)) \longrightarrow (((((1/nyx)) \otimes (y \ominus x))\ within\ \varepsilon\ of\ (p \ominus x)) \vee ((((-1/nyx)) \otimes (y \ominus x))\ within\ \varepsilon\ of\ (p \ominus x)))$
⟨*proof*⟩

end

end

16 Vectors

In this theory we define dot-products, and explain what we mean by timelike, lightlike (null), causal and spacelike vectors.

theory *Vectors*
imports *Norms*
begin

class *Vectors* = *Norms*
begin

fun *dot* :: *'a Point* ⇒ *'a Point* ⇒ *'a* (- ⊙ -)
where *dot u v* = (*tval u*)∗(*tval v*) + (*xval u*)∗(*xval v*) + (*yval u*)∗(*yval v*) + (*zval u*)∗(*zval v*)

fun *sdot* :: *'a Space* ⇒ *'a Space* ⇒ *'a* (- ⊙*s* -)
where *sdot u v* = (*svalx u*)∗(*svalx v*) + (*svaly u*)∗(*svaly v*) + (*svalz u*)∗(*svalz v*)

fun *mdot* :: *'a Point* ⇒ *'a Point* ⇒ *'a* (- ⊙*m* -)
where *mdot u v* = (*tval u*)∗(*tval v*) − ((*sComponent u*) ⊙*s* (*sComponent v*))

abbreviation *timelike* :: *'a Point* ⇒ *bool*
where *timelike p* ≡ *mNorm2 p* > *0*

abbreviation *lightlike* :: *'a Point* ⇒ *bool*
where *lightlike p* ≡ (*p* ≠ *origin* ∧ *mNorm2 p* = *0*)

abbreviation *spacelike* :: *'a Point* ⇒ *bool*
where *spacelike p* ≡ *mNorm2 p* < *0*

abbreviation *causal* :: *'a Point* ⇒ *bool*
where *causal p* ≡ *timelike p* ∨ *lightlike p*

abbreviation *orthog* :: *'a Point* ⇒ *'a Point* ⇒ *bool*
where *orthog p q* ≡ (*p* ⊙ *q*) = *0*

abbreviation *orthogs* :: *'a Space* ⇒ *'a Space* ⇒ *bool*
where *orthogs p q* ≡ (*p* ⊙*s q*) = *0*

abbreviation *orthogm* :: *'a Point* ⇒ *'a Point* ⇒ *bool*
where *orthogm p q* ≡ (*p* ⊙*m q*) = *0*

lemma *lemDotDecomposition*:
shows (*u* ⊙ *v*) = (*tval u* ∗ *tval v*) + ((*sComponent u*) ⊙*s* (*sComponent v*))
⟨*proof*⟩

lemma *lemDotCommute*: *dot u v* = *dot v u*
⟨*proof*⟩

lemma *lemDotScaleLeft*: *dot* (*a*⊗*u*) *v* = *a* ∗ (*dot u v*)
⟨*proof*⟩

lemma *lemDotScaleRight*: *dot u* (*a*⊗*v*) = *a* ∗ (*dot u v*)
⟨*proof*⟩

lemma *lemDotSumLeft*: $dot\ (u \oplus v)\ w = (dot\ u\ w) + (dot\ v\ w)$
⟨*proof*⟩

lemma *lemDotSumRight*: $dot\ u\ (v \oplus w) = (dot\ u\ v) + (dot\ u\ w)$
⟨*proof*⟩

lemma *lemDotDiffLeft*: $dot\ (u \ominus v)\ w = (dot\ u\ w) - (dot\ v\ w)$
⟨*proof*⟩

lemma *lemDotDiffRight*: $dot\ u\ (v \ominus w) = (dot\ u\ v) - (dot\ u\ w)$
⟨*proof*⟩

lemma *lemNorm2OfSum*: $norm2\ (u \oplus v) = norm2\ u + 2 * (u \odot v) + norm2\ v$
⟨*proof*⟩

lemma *lemSDotCommute*: $sdot\ u\ v = sdot\ v\ u$
⟨*proof*⟩

lemma *lemSDotScaleLeft*: $sdot\ (a \otimes s\ u)\ v = a * (sdot\ u\ v)$
⟨*proof*⟩

lemma *lemSDotScaleRight*: $sdot\ u\ (a \otimes s\ v) = a * (sdot\ u\ v)$
⟨*proof*⟩

lemma *lemSDotSumLeft*: $sdot\ (u \oplus s\ v)\ w = (sdot\ u\ w) + (sdot\ v\ w)$
⟨*proof*⟩

lemma *lemSDotSumRight*: $sdot\ u\ (\ v \oplus s\ w) = (sdot\ u\ v) + (sdot\ u\ w)$
⟨*proof*⟩

lemma *lemSDotDiffLeft*: $sdot\ (u \ominus s\ v)\ w = (sdot\ u\ w) - (sdot\ v\ w)$
⟨*proof*⟩

lemma *lemSDotDiffRight*: $sdot\ u\ (\ v \ominus s\ w) = (sdot\ u\ v) - (sdot\ u\ w)$
⟨*proof*⟩

lemma *lemMDotDiffLeft*: $mdot\ (u \ominus v)\ w = (mdot\ u\ w) - (mdot\ v\ w)$
⟨*proof*⟩

lemma *lemMDotSumLeft*: $mdot\ (u \oplus v)\ w = (mdot\ u\ w) + (mdot\ v\ w)$
⟨*proof*⟩

lemma *lemMDotScaleLeft*: $mdot\ (a \otimes u)\ v = a * (mdot\ u\ v)$
⟨*proof*⟩

lemma *lemMDotScaleRight*: $mdot\ u\ (a \otimes v) = a * (mdot\ u\ v)$
⟨*proof*⟩

lemma *lemSNorm2OfSum*: $sNorm2\ (u \oplus s\ v) = sNorm2\ u + 2*(u \odot s\ v) + sNorm2\ v$
⟨*proof*⟩

lemma *lemSNormNonNeg*:
shows $sNorm\ v \geq 0$
⟨*proof*⟩

lemma *lemMNorm2OfSum*: $mNorm2\ (u \oplus v) = mNorm2\ u + 2*(u \odot m\ v) + mNorm2\ v$
⟨*proof*⟩

lemma *lemMNorm2OfDiff*: $mNorm2\ (u \ominus v) = mNorm2\ u - 2*(u \odot m\ v) + mNorm2\ v$
⟨*proof*⟩

lemma *lemMNorm2Decomposition*: $mNorm2\ p = (p \odot m\ p)$
⟨*proof*⟩

lemma *lemMDecomposition*:
assumes $(u \odot m\ v) \neq 0$
and $mNorm2\ v \neq 0$
and $a = (u \odot m\ v)/(mNorm2\ v)$
and $up = (a \otimes v)$
and $uo = (u \ominus up)$
shows $u = (up \oplus uo) \wedge parallel\ up\ v \wedge orthogm\ uo\ v \wedge (up \odot m\ v) = (u \odot m\ v)$
⟨*proof*⟩

end

end

17 CauchySchwarz

This theory defines and proves the Cauchy-Schwarz inequality for both spatial and spacetime vectors.

theory *CauchySchwarz*
imports *Vectors*
begin

We essentially prove the same result twice, once for 3-dimensional spatial points, and once for 4-dimensional spacetime points. While this is clearly inefficient, it keeps things straightforward for non-Isabelle experts.

class *CauchySchwarz* = *Vectors*
begin

lemma *lemCauchySchwarz4*:
shows *abs* (*dot u v*) ≤ (*norm u*)∗(*norm v*)
⟨*proof*⟩

lemma *lemCauchySchwarzSqr4*:
shows *sqr*(*dot u v*) ≤ (*norm2 u*)∗(*norm2 v*)
⟨*proof*⟩

lemma *lemCauchySchwarz*:
shows *abs* (*sdot u v*) ≤ (*sNorm u*)∗(*sNorm v*)
⟨*proof*⟩

lemma *lemCauchySchwarzSqr*:
shows *sqr*(*sdot u v*) ≤ (*sNorm2 u*)∗(*sNorm2 v*)
⟨*proof*⟩

lemma *lemCauchySchwarzEquality*:
assumes *sqr* (*sdot u v*) = (*sNorm2 u*)∗(*sNorm2 v*)
and *u* ≠ *sOrigin* ∧ *v* ≠ *sOrigin*

shows $\exists\ a \neq 0\ .\ u = (a \otimes s\ v)$
⟨*proof*⟩

lemma *lemCauchySchwarzEqualityInUnitSphere*:
assumes $(sNorm2\ u \leq 1) \wedge (sNorm2\ v \leq 1)$
and $sdot\ u\ v = 1$
shows $u = v$
⟨*proof*⟩

lemma *lemCausalOrthogmToLightlikeImpliesParallel*:
assumes *causal p*
and *lightlike q*
and *orthogm p q*
shows *parallel p q*
⟨*proof*⟩

end

end

18 Matrices

This theory defines 4×4 matrices.

theory *Matrices*
imports *Vectors*
begin

record *'a Matrix* =
trow :: *'a Point*
xrow :: *'a Point*
yrow :: *'a Point*
zrow :: *'a Point*

class *Matrices* = *Vectors*
begin

fun *applyMatrix* :: *'a Matrix* ⇒ *'a Point* ⇒ *'a Point*
where *applyMatrix m p* = (| *tval* = *dot* (*trow m*) *p*, *xval* = *dot* (*xrow m*) *p*,
yval = *dot* (*yrow m*) *p*, *zval* = *dot* (*zrow m*) *p* |)

fun *tcol* :: *'a Matrix* ⇒ *'a Point*
where *tcol m* = (| *tval* = *tval* (*trow m*), *xval* = *tval* (*xrow m*),

$yval = tval\ (yrow\ m),\ zval = tval\ (zrow\ m)\ \|)$

fun *xcol* :: *'a Matrix* ⇒ *'a Point*
  where *xcol m* = (| *tval* = *xval* (*trow m*), *xval* = *xval* (*xrow m*),
    yval = *xval* (*yrow m*), *zval* = *xval* (*zrow m*) |)

fun *ycol* :: *'a Matrix* ⇒ *'a Point*
  where *ycol m* = (| *tval* = *yval* (*trow m*), *xval* = *yval* (*xrow m*),
    yval = *yval* (*yrow m*), *zval* = *yval* (*zrow m*) |)

fun *zcol* :: *'a Matrix* ⇒ *'a Point*
  where *zcol m* = (| *tval* = *zval* (*trow m*), *xval* = *zval* (*xrow m*),
    yval = *zval* (*yrow m*), *zval* = *zval* (*zrow m*) |)

fun *transpose* :: *'a Matrix* ⇒ *'a Matrix*
  where *transpose m* = (| *trow* = (*tcol m*), *xrow* = (*xcol m*),
      yrow = (*ycol m*), *zrow* = (*zcol m*) |)

fun *mprod* :: *'a Matrix* ⇒ *'a Matrix* ⇒ *'a Matrix*
  where *mprod m1 m2* =
    transpose (| *trow* = *applyMatrix m1* (*tcol m2*), *xrow* = *applyMatrix m1* (*xcol m2*),
      yrow = *applyMatrix m1* (*ycol m2*), *zrow* = *applyMatrix m1* (*zcol m2*) |)

end

end

19 LinearMaps

This theory defines linear maps and establishes their main properties.

theory *LinearMaps*
  imports *Functions CauchySchwarz Matrices*
begin

class *LinearMaps* = *Functions* + *CauchySchwarz* + *Matrices*
begin

abbreviation *linear* :: (*'a Point* ⇒ *'a Point*) ⇒ *bool* **where**

$$\begin{aligned}
&\mathit{linear}\ L \equiv (L\ \mathit{origin} = \mathit{origin}) \\
&\qquad \wedge (\forall\ a\ p\ .\ \ L\ (a \otimes p) = (a \otimes (L\ p))) \\
&\qquad \wedge (\forall\ p\ q\ .\ L\ (p \oplus q) = ((L\ p) \oplus (L\ q))) \\
&\qquad \wedge (\forall\ p\ q\ .\ L\ (p \ominus q) = ((L\ p) \ominus (L\ q)))
\end{aligned}$$

lemma *lemLinearProps*:
assumes $\mathit{linear}\ L$
shows $(L\ \mathit{origin} = \mathit{origin}) \wedge (L\ (a \otimes p) = (a \otimes (L\ p)))$
$\wedge\ (L\ (p \oplus q) = ((L\ p) \oplus (L\ q)))$
$\wedge\ (L\ (p \ominus q) = ((L\ p) \ominus (L\ q)))$
⟨*proof*⟩

lemma *lemMatrixApplicationIsLinear*: $\mathit{linear}\ (\mathit{applyMatrix}\ m)$
⟨*proof*⟩

lemma *lemLinearIsMatrixApplication*:
assumes $\mathit{linear}\ L$
shows $\exists\ m\ .\ L = (\mathit{applyMatrix}\ m)$
⟨*proof*⟩

lemma *lemLinearIffMatrix*: $\mathit{linear}\ L \longleftrightarrow (\exists\ M.\ L = \mathit{applyMatrix}\ M)$
⟨*proof*⟩

lemma *lemIdIsLinear*: $\mathit{linear}\ \mathit{id}$
⟨*proof*⟩

lemma *lemLinearIsBounded*:
assumes $\mathit{linear}\ L$
shows $\mathit{bounded}\ L$
⟨*proof*⟩

lemma *lemLinearIsCts*:

assumes *linear L*
shows *cts* (*asFunc L*) *x*
⟨*proof*⟩

lemma *lemLinOfLinIsLin*:
assumes (*linear A*) ∧ (*linear B*)
shows *linear* (*B* ∘ *A*)
⟨*proof*⟩

lemma *lemInverseLinear*:
assumes *linear A*
and *invertible A*
shows ∃ *A′* . (*linear A′*) ∧ (∀ *p q*. *A p* = *q* ⟷ *A′ q* = *p*)
⟨*proof*⟩

end

end

20 Affine

This theory defines affine transformations and established their key properties.

theory *Affine*
imports *Translations LinearMaps*
begin

class *Affine* = *Translations* + *LinearMaps*
begin

abbreviation *affine* :: (*′a Point* ⇒ *′a Point*) ⇒ *bool*
where *affine A* ≡ ∃ *L T* . (*linear L*) ∧ (*translation T*) ∧ (*A* = *T* ∘ *L*)

abbreviation *affInvertible* :: (*′a Point* ⇒ *′a Point*) ⇒ *bool*
where *affInvertible A* ≡ *affine A* ∧ *invertible A*

abbreviation *isLinearPart* :: (*'a Point* ⇒ *'a Point*) ⇒ (*'a Point* ⇒ *'a Point*) ⇒ *bool*
  where *isLinearPart A L* ≡ (*affine A*) ∧ (*linear L*) ∧
        (∃ *T*. (*translation T* ∧ *A* = *T* ∘ *L*))

abbreviation *isTranslationPart* :: (*'a Point* ⇒ *'a Point*) ⇒ (*'a Point* ⇒ *'a Point*) ⇒ *bool*
  where *isTranslationPart A T* ≡ (*affine A*) ∧ (*translation T*) ∧
        (∃ *L*. (*linear L* ∧ *A* = *T* ∘ *L*))

20.1 Affine approximation

A key concept in the proof is affine approximation. We will eventually assert that worldview transformation can be approximated by invertible affine transformations.

abbreviation *affineApprox* :: (*'a Point* ⇒ *'a Point*) ⇒
                (*'a Point* ⇒ *'a Point* => *bool*) ⇒
                'a Point ⇒ *bool*
  where *affineApprox A f x* ≡ (*isFunction f*) ∧
        (*affInvertible A*) ∧ (*diffApprox* (*asFunc A*) *f x*)

fun *applyAffineToLine* :: (*'a Point* ⇒ *'a Point*)
          ⇒ *'a Point set* ⇒ *'a Point set* ⇒ *bool*
  where *applyAffineToLine A l l'* ⟷ (*affine A*) ∧
    (∃ *T L b d* . ((*linear L*) ∧ (*translation T*) ∧ (*A* = *T* ∘ *L*) ∧
      (*l* = *line b d*) ∧ (*l'* = (*line* (*A b*) (*L d*)))))

abbreviation *affConstantOn* :: (*'a Point* ⇒ *'Point*) ⇒ *'a Point* ⇒ *'a Point set* ⇒ *bool*
  where *affConstantOn A x s* ≡ (∃ε>*0*. ∀ *y*∈*s*. (*y within* ε *of x*) ⟶ ((*A y*) = (*A x*)))

lemma *lemTranslationPartIsUnique*:
  assumes *isTranslationPart A T1*
and *isTranslationPart A T2*
shows *T1* = *T2*
⟨*proof*⟩

lemma *lemLinearPartIsUnique*:

assumes *isLinearPart A L1*
and *isLinearPart A L2*
shows *L1 = L2*
⟨*proof*⟩

lemma *lemLinearImpliesAffine*:
assumes *linear L*
shows *affine L*
⟨*proof*⟩

lemma *lemTranslationImpliesAffine*:
assumes *translation T*
shows *affine T*
⟨*proof*⟩

lemma *lemAffineDiff*:
assumes *linear L*
and $\exists\ T\ .\ ((translation\ T) \wedge (A = T \circ L))$
shows $((A\ p) \ominus (A\ q)) = L\ (p \ominus q)$
⟨*proof*⟩

lemma *lemAffineImpliesTotalFunction*:
assumes *affine A*
shows *isTotalFunction* (*asFunc A*)
⟨*proof*⟩

lemma *lemAffineEqualAtBase*:
assumes *affineApprox A f x*
shows $\forall\, y.\ (f\ x\ y) \longleftrightarrow (y = A\ x)$
⟨*proof*⟩

lemma *lemAffineOfPointOnLine*:
assumes $(linear\ L) \wedge (translation\ T) \wedge (A = T \circ L)$
and $x = (b \oplus (a \otimes d))$
shows $A\ x = ((A\ b) \oplus (a \otimes (L\ d)))$
⟨*proof*⟩

lemma *lemAffineOfLineIsLine*:
assumes *isLine l*
shows $(applyAffineToLine\ A\ l\ l') \longleftrightarrow (affine\ A \wedge l' = applyToSet$

$(asFunc\ A)\ l)$
⟨*proof*⟩

lemma *lemOnLineUnderAffine*:
assumes $(affine\ A) \wedge (onLine\ p\ l)$
shows $onLine\ (A\ p)\ (applyToSet\ (asFunc\ A)\ l)$
⟨*proof*⟩

lemma *lemLineJoiningUnderAffine*:
assumes *affine A*
shows $applyToSet\ (asFunc\ A)\ (lineJoining\ p\ q) = lineJoining\ (A\ p)\ (A\ q)$
⟨*proof*⟩

lemma *lemAffineIsCts*:
assumes *affine A*
shows $cts\ (asFunc\ A)\ x$
⟨*proof*⟩

lemma *lemAffineContinuity*:
assumes *affine A*
shows $\forall\ x.\ \forall \varepsilon{>}0.\ \exists \delta{>}0\ .\ \forall p.\ (p\ within\ \delta\ of\ x) \longrightarrow ((A\ p)\ within\ \varepsilon\ of\ (A\ x))$
⟨*proof*⟩

lemma *lemAffOfAffIsAff*:
assumes $(affine\ A) \wedge (affine\ B)$
shows $affine\ (B \circ A)$
⟨*proof*⟩

lemma *lemInverseAffine*:
assumes *affInvertible A*
shows $\exists A'\ .\ (affine\ A') \wedge (\forall\ p\ q\ .\ A\ p = q \longleftrightarrow A'\ q = p)$
⟨*proof*⟩

lemma *lemAffineApproxDomainTranslation*:

assumes *translation T*
and *affineApprox A f x*
and $\forall\ p\ q\ .\ T\ p = q \longleftrightarrow T'\ q = p$
shows *affineApprox* $(A \circ T)$ $(composeRel\ f\ (asFunc\ T))$ $(T'\ x)$
⟨*proof*⟩

lemma *lemAffineApproxRangeTranslation*:
assumes *translation T*
and *affineApprox A f x*
shows *affineApprox* $(T \circ A)$ $(composeRel\ (asFunc\ T)\ f)\ x$
⟨*proof*⟩

lemma *lemAffineIdentity*:
assumes *affine A*
and $e > 0$
and $\forall\ y\ .\ (y\ within\ e\ of\ x) \longrightarrow (A\ y = y)$
shows $A = id$
⟨*proof*⟩

end

end

21 Sublemma4

This theory shows that functions with affine approximations are continuous where approximated.

theory *Sublemma4*
imports *Affine AxTriangleInequality*
begin

Our naming of lemmas, propositions, etc., is sometimes counterintuitive. This is because the proof follows a hand-written proof, and we need to maintain the link between the paper-based and Isabelle versions. We will specifically be discussing how we translated from one to the other in a forthcoming paper (under construction). In fact, sublemmas 1 and 2 were eventually found to be unnecessary during construction of the Isabelle proof, and so do not appear in this documentation.

class *Sublemma4* = *Affine* + *AxTriangleInequality*

begin

lemma *sublemma4*:
assumes *affineApprox A f x*
shows $(\exists \delta{>}0.\ \forall p.\ (p\ within\ \delta\ of\ x) \longrightarrow (definedAt\ f\ p)) \wedge (cts\ f\ x)$
⟨*proof*⟩

end

end

22 MainLemma

This theory establishes conditions under which a function maps tangent lines to tangent lines.

theory *MainLemma*
imports *Sublemma3 Sublemma4*
begin

class *MainLemma* = *Sublemma3* + *Sublemma4*
begin

lemma *lemMainLemmaBasic*:
assumes *tgt*: *tangentLine l wl origin*
and *injf*: *injective f*
and *affapp*: *affineApprox A f origin*
and *f00*: *f origin origin*
and *ctsf'0*: *cts* (*invFunc f*) *origin*
and *affline*: *applyAffineToLine A l l'*
shows *tangentLine l'* (*applyToSet f wl*) *origin*
⟨*proof*⟩

lemma *lemMainLemmaOrigin*:
assumes *tgtx*: *tangentLine l wl x*
and *injf*: *injective f*
and *affappx*: *affineApprox A f x*
and *fx0*: *f x origin*
and *ctsf'0*: *cts* (*invFunc f*) *origin*
and *affline*: *applyAffineToLine A l l'*
shows *tangentLine l'* (*applyToSet f wl*) *origin*

⟨*proof*⟩

lemma *lemMainLemma*:
assumes *tgtx*: *tangentLine l wl x*
and *injf*: *injective f*
and *affappx*: *affineApprox A f x*
and *fxy*: *f x y*
and *ctsf'y*: *cts (invFunc f) y*
and *affline*: *applyAffineToLine A l l'*
shows *tangentLine l' (applyToSet f wl) y*
⟨*proof*⟩

end

end

23 AXIOM: AxDiff

This theory declares the axiom AxDiff.

theory *AxDiff*
imports *Affine WorldView*
begin

AxDiff: Worldview transformations are differentiable wherever they are defined - they can be approximated locally by affine transformations.

class *axDiff* = *Affine* + *WorldView*
begin
abbreviation *axDiff* :: *Body* ⇒ *Body* ⇒ *'a Point* ⇒ *bool*
where *axDiff m k p* ≡ (*definedAt* (*wvtFunc m k*) *p*)
⟶ (∃ *A* . (*affineApprox A* (*wvtFunc m k*) *p*))
end

class *AxDiff* = *axDiff* +
assumes *AxDiff*: ∀ *m k p* . *axDiff m k p*
begin
end

end

24 TangentLineLemma

This theory shows that affine approximations map tangent lines to tangent lines.

theory *TangentLineLemma*
imports *MainLemma AxDiff Cones*
begin

class *TangentLineLemma* = *MainLemma* + *AxDiff* + *Cones*
begin

lemma *lemWVTImpliesFunction*: *isFunction* (*wvtFunc k h*)
⟨*proof*⟩

lemma *lemWVTCts*:
assumes *definedAt* (*wvtFunc h k*) *p*
shows *cts* (*wvtFunc h k*) *p*
⟨*proof*⟩

lemma *lemWVTInverse*: *invFunc* (*wvtFunc k h*) = *wvtFunc h k*
⟨*proof*⟩

lemma *lemWVTInverseCts*:
assumes *wvtFunc k h p q*
shows *cts* (*wvtFunc h k*) *q*
⟨*proof*⟩

lemma *lemWVTInjective*: *injective* (*wvtFunc k h*)
⟨*proof*⟩

lemma *lemPresentation*:
assumes *x* ∈ *wline m b*
and *tangentLine l* (*wline m b*) *x*
and *affineApprox A* (*wvtFunc m k*) *x*
and *wvtFunc m k x y*
and *applyAffineToLine A l l′*
shows *tangentLine l′* (*wline k b*) *y*
⟨*proof*⟩

lemma *lemTangentLines*:
assumes *affineApprox A* (*wvtFunc m k*) *x*
and *tl l m b x*
and *applyAffineToLine A l l′*
and *wvtFunc m k x y*
shows *tl l′ k b y*
⟨*proof*⟩

lemma *lemSelfTangentIsTimeAxis*:
assumes *tangentLine l* (*wline k k*) *x*
shows *l* = *timeAxis*
⟨*proof*⟩

lemma *lemTangentLineUnique*:
assumes *tl l1 m k x*
and *tl l2 m k x*
and *affineApprox A* (*wvtFunc m k*) *x*
and *wvtFunc m k x y*
and *x* ∈ *wline m k*
shows *l1* = *l2*
⟨*proof*⟩

end

end

25 Proposition2

This theory shows that affine approximations map surfaces of cones to (subsets of) surfaces of cones.

theory *Proposition2*
imports *TangentLineLemma*
begin

class *Proposition2* = *TangentLineLemma*
begin

lemma *lemProposition2*:
assumes *affineApprox A* (*wvtFunc m k*) *x*
shows *applyToSet* (*asFunc A*) (*coneSet m x*) ⊆ *coneSet k* (*A x*)

⟨*proof*⟩

end

end

26 AXIOM: AxEventMinus

This theory declares the axiom AxEventMinus

theory *AxEventMinus*
  imports *WorldView*
begin

AxEventMinus: An observer encounters the events in which they are observed.

class *axEventMinus* = *WorldView*
begin

  abbreviation *axEventMinus* :: *Body* ⇒ *Body* ⇒ *'a Point* ⇒ *bool*
    where *axEventMinus m k p* ≡ (*m sees k at p*)
                ⟶ (∃ *q* . ∀ *b* . ((*m sees b at p*) ⟷ (*k sees b at q*)))

end

class *AxEventMinus* = *axEventMinus* +
  assumes *AxEventMinus*: ∀ *m k p* . *axEventMinus m k p*
begin
end

end

27 Proposition3

This theory collects together earlier results to show that world-view transformations can be approximated by affine transformations that have various useful properties.

theory *Proposition3*
  imports *Proposition1 Proposition2 AxEventMinus*
begin

class *Proposition3* = *Proposition1* + *Proposition2* + *AxEventMinus*
begin

lemma *lemProposition3*:
assumes *m sees k at x*
shows $\exists\ A\ y\ .\ (wvtFunc\ m\ k\ x\ y)$
$\wedge\quad (affineApprox\ A\ (wvtFunc\ m\ k)\ x)$
$\wedge\quad (applyToSet\ (asFunc\ A)\ (coneSet\ m\ x) \subseteq coneSet\ k\ y)$
$\wedge\quad (coneSet\ k\ y = regularConeSet\ y)$
⟨*proof*⟩

end

end

28 ObserverConeLemma

This theory gives sufficient conditions for an observed observer's cone to appear upright to that observer.

theory *ObserverConeLemma*
imports *Proposition3*
begin

class *ObserverConeLemma* = *Proposition3*
begin

lemma *lemConeOfObserved*:
assumes $affineApprox\ A\ (wvtFunc\ m\ k)\ x$
and *m sees k at x*
shows $coneSet\ k\ (A\ x) = regularConeSet\ (A\ x)$
⟨*proof*⟩

end

end

29 Quadratics

This theory shows how to find the roots of a quadratic, assuming that roots exist (AxEField).

theory *Quadratics*
imports *Functions AxEField*
begin

class *Quadratics* = *Functions* + *AxEField*
begin

abbreviation *quadratic* :: $'a \Rightarrow 'a \Rightarrow 'a \Rightarrow ('a \Rightarrow 'a)$
where *quadratic a b c* $\equiv \lambda\ x\ .\ a*(sqr\ x) + b*x + c$

abbreviation *qroot* :: $'a \Rightarrow 'a \Rightarrow 'a \Rightarrow 'a \Rightarrow bool$
where *qroot a b c r* $\equiv$ (*quadratic a b c*) *r* = *0*

abbreviation *qroots* :: $'a \Rightarrow 'a \Rightarrow 'a \Rightarrow 'a\ set$
where *qroots a b c* $\equiv \{\ r\ .\ qroot\ a\ b\ c\ r\ \}$

abbreviation *discriminant* :: $'a \Rightarrow 'a \Rightarrow 'a \Rightarrow 'a$
where *discriminant a b c* $\equiv (sqr\ b) - 4*a*c$

abbreviation *qcase1* :: $'a \Rightarrow 'a \Rightarrow 'a \Rightarrow bool$
where *qcase1 a b c* $\equiv (a = 0 \wedge b = 0 \wedge c = 0)$
abbreviation *qcase2* :: $'a \Rightarrow 'a \Rightarrow 'a \Rightarrow bool$
where *qcase2 a b c* $\equiv (a = 0 \wedge b = 0 \wedge c \neq 0)$
abbreviation *qcase3* :: $'a \Rightarrow 'a \Rightarrow 'a \Rightarrow bool$
where *qcase3 a b c* $\equiv (a = 0 \wedge b \neq 0 \wedge (c = 0 \vee c \neq 0))$
abbreviation *qcase4* :: $'a \Rightarrow 'a \Rightarrow 'a \Rightarrow bool$
where *qcase4 a b c* $\equiv (a \neq 0 \wedge discriminant\ a\ b\ c < 0)$
abbreviation *qcase5* :: $'a \Rightarrow 'a \Rightarrow 'a \Rightarrow bool$
where *qcase5 a b c* $\equiv (a \neq 0 \wedge discriminant\ a\ b\ c = 0)$
abbreviation *qcase6* :: $'a \Rightarrow 'a \Rightarrow 'a \Rightarrow bool$
where *qcase6 a b c* $\equiv (a \neq 0 \wedge discriminant\ a\ b\ c > 0)$

lemma *lemQuadRootCondition*:
assumes *a* $\neq$ *0*
shows $(sqr\ (2*a*r + b) = discriminant\ a\ b\ c) \longleftrightarrow qroot\ a\ b\ c\ r$
⟨*proof*⟩

lemma *lemQuadraticCasesComplete*:
shows *qcase1 a b c* $\vee$ *qcase2 a b c* $\vee$ *qcase3 a b c* $\vee$ *qcase4 a b c* $\vee$
qcase5 a b c $\vee$ *qcase6 a b c*
⟨*proof*⟩

lemma *lemQCase1*:
assumes *qcase1 a b c*
shows $\forall\ r\ .\ qroot\ a\ b\ c\ r$
⟨*proof*⟩

lemma *lemQCase2*:
assumes *qcase2 a b c*
shows $\neg\ (\exists\ r\ .\ qroot\ a\ b\ c\ r)$
⟨*proof*⟩

lemma *lemQCase3*:
assumes *qcase3 a b c*
shows $qroot\ a\ b\ c\ r \longleftrightarrow r = -c/b$
⟨*proof*⟩

lemma *lemQCase4*:
assumes *qcase4 a b c*
shows $\neg\ (\exists\ r\ .\ qroot\ a\ b\ c\ r)$
⟨*proof*⟩

lemma *lemQCase5*:
assumes *qcase5 a b c*
shows $qroot\ a\ b\ c\ r \longleftrightarrow r = -b/(2*a)$
⟨*proof*⟩

lemma *lemQCase6*:
assumes *qcase6 a b c*
and $rd = sqrt\ (discriminant\ a\ b\ c)$
and $rp = ((-b) + rd)\ /\ (2*a)$
and $rm = ((-b) - rd)\ /\ (2*a)$
shows $(rp \neq rm) \wedge qroots\ a\ b\ c = \{\ rp,\ rm\ \}$
⟨*proof*⟩

lemma *lemQuadraticRootCount*:
assumes $\neg(qcase1\ a\ b\ c)$
shows $finite\ (qroots\ a\ b\ c) \wedge card\ (qroots\ a\ b\ c) \leq 2$
⟨*proof*⟩

end

end

30 Classification

This theory explains how to establish whether a point lies inside, on or outside a cone.

theory *Classification*
imports *Cones Quadratics CauchySchwarz*
begin

We want to establish where a point lies in relation to a cone, and will later show that this relationship is preserved under relevant affine transformations. We therefore need a classification scheme that relies on purely affine concepts. To do this we consider lines that can be drawn through the point, and ask how many points lie in the intersection of such a line and the cone.

class *Classification* = *Cones* + *Quadratics* + *CauchySchwarz*
begin

abbreviation *vertex* :: *'a Point* ⇒ *'a Point* ⇒ *bool*
where *vertex x p* ≡ (*x* = *p*)

abbreviation *insideRegularCone* :: *'a Point* ⇒ *'a Point* ⇒ *bool*
where *insideRegularCone x p* ≡
(*slopeFinite x p*) ∧ (∃ *v* ∈ *lineVelocity* (*lineJoining x p*) . *sNorm2 v* < *1*)

abbreviation *outsideRegularCone* :: *'a Point* ⇒ *'a Point* ⇒ *bool*
where *outsideRegularCone x p* ≡
(*x* ≠ *p*) ∧
((*slopeInfinite x p*) ∨ (∃ *v* ∈ *lineVelocity* (*lineJoining x p*) . *sNorm2 v* > *1*))

abbreviation *onRegularCone* :: *'a Point* ⇒ *'a Point* ⇒ *bool*
where *onRegularCone x p* ≡ (*x* = *p*) ∨ (∃ *v* ∈ *lineVelocity* (*lineJoining x p*) . *sNorm2 v* = *1*)

lemma *lemDrtnLineJoining*:
assumes $l = lineJoining\ x\ p$
and $x \neq p$
shows $(p \ominus x) \in drtn\ l$
⟨*proof*⟩

lemma *lemVelocityLineJoining*:
assumes $l = lineJoining\ x\ p$
and $v = velocityJoining\ origin\ (p \ominus x)$
and $x \neq p$
shows $v \in lineVelocity\ l$
⟨*proof*⟩

lemma *lemSlopeLineJoining*:
assumes $l = lineJoining\ p\ q$
and $p \neq q$
shows $lineSlopeFinite\ l \longleftrightarrow slopeFinite\ p\ q$
⟨*proof*⟩

lemma *lemVelocityJoiningUsingPoints*:
assumes $p \neq q$
shows $velocityJoining\ p\ q = velocityJoining\ origin\ (q \ominus p)$
⟨*proof*⟩

lemma *lemLineVelocityNonZeroImpliesFinite*:
assumes $u \in lineVelocity\ l$
and $sNorm2\ u \neq 0$
shows $lineSlopeFinite\ l$
⟨*proof*⟩

lemma *lemLineVelocityUsingPoints*:
assumes $slopeFinite\ p\ q$
and $onLine\ p\ l \wedge onLine\ q\ l$
shows $lineVelocity\ l = \{\ velocityJoining\ p\ q\ \}$
⟨*proof*⟩

lemma *lemSNorm2VelocityJoining*:
assumes *slopeFinite x p*
and $v = velocityJoining\ x\ p$
shows $sqr\ (tval\ p - tval\ x) * sNorm2\ v = sNorm2\ (sComponent\ (p \ominus x))$
⟨*proof*⟩

lemma *lemOrthogalSpaceVectorExists*:
shows $\exists\ w\ .\ (w \neq sOrigin) \wedge (w \odot s\ v) = 0$
⟨*proof*⟩

lemma *lemNonParallelVectorsExist*:
shows $\exists\ w\ .\ ((w \neq origin) \wedge (tval\ v = tval\ w)) \wedge (\neg\ (\exists\ \alpha\ .\ (\alpha \neq 0) \wedge v = (\alpha \otimes w)))$
⟨*proof*⟩

lemma *lemConeContainsVertex*:
shows *regularCone x x*
⟨*proof*⟩

lemma *lemConesExist*:
shows $regularConeSet\ x \neq \{\}$
⟨*proof*⟩

lemma *lemRegularCone*:
shows $((x = p) \vee onRegularCone\ x\ p) \longleftrightarrow regularCone\ x\ p$
⟨*proof*⟩

lemma *lemSlopeInfiniteImpliesOutside*:
assumes $x \neq p$
and *slopeInfinite x p*
shows $\exists\ l\ p'\ .\ (p' \neq p) \wedge onLine\ p'\ l \wedge onLine\ p\ l \wedge (l \cap regularConeSet\ x = \{\})$
⟨*proof*⟩

lemma *lemClassification*:
shows $(insideRegularCone\ x\ p) \vee (vertex\ x\ p \vee outsideRegularCone\ x\ p \vee onRegularCone\ x\ p)$
⟨*proof*⟩

lemma *lemQuadCoordinates*:
assumes $p = (B \oplus (\alpha \otimes D))$
and $a = mNorm2\ D$
and $b = 2{*}(tval\ (B{\ominus}x)){*}(tval\ D) - 2{*}((sComponent\ D) \odot s\ (sComponent\ (B{\ominus}x)))$
and $c = mNorm2\ (B{\ominus}x)$
shows $sqr\ (tval\ (p{\ominus}x)) - sNorm2\ (sComponent\ (p{\ominus}x)) = a{*}(sqr\ \alpha) + b{*}\alpha + c$
⟨*proof*⟩

lemma *lemConeCoordinates*:
shows $(onRegularCone\ x\ p \longleftrightarrow sqr\ (tval\ p - tval\ x) = sNorm2\ (sComponent\ (p{\ominus}x)))$
$\wedge\ (insideRegularCone\ x\ p \longleftrightarrow sqr\ (tval\ p - tval\ x) > sNorm2\ (sComponent\ (p{\ominus}x)))$
$\wedge\ (outsideRegularCone\ x\ p \longleftrightarrow sqr\ (tval\ p - tval\ x) < sNorm2\ (sComponent\ (p{\ominus}x)))$
⟨*proof*⟩

lemma *lemConeCoordinates1*:
shows $p \in regularConeSet\ x \longleftrightarrow norm2\ (p{\ominus}x) = 2{*}sqr\ (tval\ p - tval\ x)$
⟨*proof*⟩

lemma *lemWhereLineMeetsCone*:
assumes $a = mNorm2\ D$
and $b = 2{*}(tval\ (B{\ominus}x)){*}(tval\ D) - 2{*}((sComponent\ D) \odot s\ (sComponent\ (B{\ominus}x)))$
and $c = mNorm2\ (B{\ominus}x)$
shows $qroot\ a\ b\ c\ \alpha \longleftrightarrow regularCone\ x\ (B \oplus (\alpha{\otimes}D))$
⟨*proof*⟩

lemma *lemLineMeetsCone1*:
assumes $\neg\ (x \in l)$
and $isLine\ l$
and $S = l \cap regularConeSet\ x$
and l: $l = line\ B\ D$
and X: $X = (B \ominus x)$
and a: $a = mNorm2\ D$
and b: $b = 2{*}(tval\ X){*}(tval\ D) - 2{*}((sComponent\ D) \odot s\ (sComponent\ X))$
and c: $c = mNorm2\ X$
shows $(qcase1\ a\ b\ c \longrightarrow S = \{B\})$

⟨*proof*⟩

lemma *lemLineMeetsCone2*:
assumes ¬ $(x \in l)$
and *isLine* l
and $S = l \cap$ *regularConeSet* x
and l: $l =$ *line* B D
and X: $X = (B \ominus x)$
and $a =$ *mNorm2* D
and $b = 2*(tval\ (B \ominus x))*(tval\ D) - 2*((sComponent\ D) \odot s\ (sComponent\ (B \ominus x)))$
and $c =$ *mNorm2* $(B \ominus x)$
shows *qcase2* a b $c \longrightarrow S = \{\}$
⟨*proof*⟩

lemma *lemLineMeetsCone3*:
assumes ¬ $(x \in l)$
and *isLine* l
and $S = l \cap$ *regularConeSet* x
and l: $l =$ *line* B D
and X: $X = (B \ominus x)$
and a: $a =$ *mNorm2* D
and b: $b = 2*(tval\ X)*(tval\ D) - 2*((sComponent\ D) \odot s\ (sComponent\ X))$
and c: $c = sqr\ (tval\ X) - sNorm2\ (sComponent\ X)$
and *y3*: *y3* $= (B \oplus ((-c/b) \otimes D))$
shows *qcase3* a b $c \longrightarrow S = \{y3\}$
⟨*proof*⟩

lemma *lemLineMeetsCone4*:
assumes ¬ $(x \in l)$
and *isLine* l
and $S = l \cap$ *regularConeSet* x
and l: $l =$ *line* B D
and X: $X = (B \ominus x)$
and a: $a =$ *mNorm2* D
and b: $b = 2*(tval\ X)*(tval\ D) - 2*((sComponent\ D) \odot s\ (sComponent\ X))$
and c: $c = sqr\ (tval\ X) - sNorm2\ (sComponent\ X)$
shows (*qcase4* a b $c \longrightarrow S = \{\}$)
⟨*proof*⟩

lemma *lemLineMeetsCone5*:
assumes $\neg\ (x \in l)$
and *isLine l*
and $S = l \cap$ *regularConeSet x*
and *l*: $l =$ *line B D*
and *X*: $X = (B \ominus x)$
and *a*: $a =$ *mNorm2 D*
and *b*: $b = 2{*}(tval\ X){*}(tval\ D) - 2{*}((sComponent\ D) \odot s\ (sComponent\ X))$
and *c*: $c = sqr\ (tval\ X) - sNorm2\ (sComponent\ X)$
and *y5*: $y5 = (B \oplus ((-b/(2{*}a)) \otimes D))$
shows $(qcase5\ a\ b\ c \longrightarrow S = \{y5\})$
⟨*proof*⟩

lemma *lemLineMeetsCone6*:
assumes $\neg\ (x \in l)$
and *isLine l*
and $S = l \cap$ *regularConeSet x*
and *l*: $l =$ *line B D*
and *X*: $X = (B \ominus x)$
and *a*: $a =$ *mNorm2 D*
and *b*: $b = 2{*}(tval\ X){*}(tval\ D) - 2{*}((sComponent\ D) \odot s\ (sComponent\ X))$
and *c*: $c = sqr\ (tval\ X) - sNorm2\ (sComponent\ X)$
and *ym*: $ym = (B \oplus (((-b - (sqrt\ (discriminant\ a\ b\ c))) / (2{*}a)) \otimes D))$
and *yp*: $yp = (B \oplus (((-b + (sqrt\ (discriminant\ a\ b\ c))) / (2{*}a)) \otimes D))$
shows $(qcase6\ a\ b\ c \longrightarrow (ym \neq yp) \wedge S = \{ym,\ yp\})$
⟨*proof*⟩

lemma *lemConeLemma1*:
assumes $\neg\ (x \in l)$
and *isLine l*
and $S = l \cap$ *regularConeSet x*
shows $finite\ S \wedge card\ S \leq 2$
⟨*proof*⟩

lemma *lemConeLemma2*:
assumes $\neg\ (regularCone\ x\ w)$
shows $\exists\ l\ .\ (onLine\ w\ l) \wedge (\neg\ (x \in l)) \wedge (card\ (l \cap (regularConeSet\ x)) = 2)$

⟨*proof*⟩

lemma *lemLineInsideRegularConeHasFiniteSlope*:
assumes *insideRegularCone x p*
and $l = lineJoining\ x\ p$
shows *lineSlopeFinite l*
⟨*proof*⟩

lemma *lemInvertibleOnMeet*:
assumes *invertible f*
and $S = A \cap B$
shows $applyToSet\ (asFunc\ f)\ S = (applyToSet\ (asFunc\ f)\ A) \cap (applyToSet\ (asFunc\ f)\ B)$
⟨*proof*⟩

lemma *lemInsideCone*:
shows $insideRegularCone\ x\ p \longleftrightarrow \neg(vertex\ x\ p \vee outsideRegularCone\ x\ p \vee onRegularCone\ x\ p)$
⟨*proof*⟩

lemma *lemOnRegularConeIff*:
assumes $l = lineJoining\ x\ p$
shows $onRegularCone\ x\ p \longleftrightarrow (l \cap regularConeSet\ x = l)$
⟨*proof*⟩

lemma *lemOutsideRegularConeImplies*:
shows $outsideRegularCone\ x\ p \longrightarrow (\exists\ l\ p'\ .\ (p' \neq p) \wedge onLine\ p'\ l \wedge onLine\ p\ l \wedge (l \cap regularConeSet\ x = \{\}))$
⟨*proof*⟩

lemma *lemTimelikeInsideCone*:
assumes *insideRegularCone x p*
shows $timelike\ (p \ominus x)$
⟨*proof*⟩

end
end

31 ReverseCauchySchwarz

This theory defines and proves the "reverse" Cauchy-Schwarz inequality for timelike vectors in the Minkowski metric.

theory *ReverseCauchySchwarz*
imports *CauchySchwarz*
begin

Rather than construct the proof, one could simply have asserted the claim as an axiom. We did this during development of the main proof, and then returned to this section later. In practice the axiom we chose to assert contained far more information than required, because we eventually found a proof that only required consideration of timelike vectors (our axiom considered lightlike vectors as well).

class *ReverseCauchySchwarz* = *CauchySchwarz*

begin

lemma *lemTimelikeNotZeroTime*:
assumes *timelike v*
shows *tval v* ≠ *0*
⟨*proof*⟩

lemma *lemOrthogmToTimelike*:
assumes *timelike u*
and *orthogm u v*
and *v* ≠ *origin*
shows *spacelike v*
⟨*proof*⟩

lemma *lemNormaliseTimelike*:
assumes *timelike v*
and *s* = *sComponent* ((*1*/*tval v*)⊗*v*)
shows (*0* ≤ *sNorm2 s* < *1*) ∧ (*tval* ((*1*/*tval v*)⊗*v*) = *1*)
⟨*proof*⟩

lemma *lemReverseCauchySchwarz*:
assumes *timelike X* ∧ *timelike D*
shows *sqr* (*X* ⊙*m D*) ≥ (*mNorm2 X*)∗(*mNorm2 D*)
⟨*proof*⟩

end

end

32 KeyLemma

This theory establishes a "key lemma": if you draw a line through a point inside a cone, that line will intersect the cone in no fewer than 1 and no more than 2 points.

theory *KeyLemma*
imports *Classification ReverseCauchySchwarz*
begin

class *KeyLemma* = *Classification* + *ReverseCauchySchwarz*
begin

lemma *lemInsideRegularConeImplies*:
assumes *insideRegularCone x p*
and $D \neq origin$
and $l = line\ p\ D$
shows $0 < card\ (l \cap regularConeSet\ x) \leq 2$
⟨*proof*⟩

end
end

33 Cardinalities

For our purposes the only relevant cardinalities are 0, 1, 2 and more-than-2 (a proxy for "infinite"). We will use these cardinalities when looking at how lines intersect cones, using the size of the intersection set to characterise whether points are inside, on or outside of lightcones.

theory *Cardinalities*
imports *Functions*
begin

class *Cardinalities* = *Functions*
begin

lemma *lemInjectiveValueUnique*:

assumes *injective f*
and *isFunction f*
and *f x y*
shows $\{ q.\ f\ x\ q \} = \{ y \}$
⟨*proof*⟩

lemma *lemBijectionOnTwo*:
assumes *bijective f*
and *isFunction f*
and $s \subseteq domain\ f$
and $card\ s = 2$
shows $card\ (applyToSet\ f\ s) = 2$
⟨*proof*⟩

lemma *lemElementsOfSet2*:
assumes $card\ S = 2$
shows $\exists\ p\ q\ .\ (p \neq q) \wedge p \in S \wedge q \in S$
⟨*proof*⟩

lemma *lemThirdElementOfSet2*:
assumes $(p \neq q) \wedge p \in S \wedge q \in S \wedge (card\ S = 2)$
and $r \in S$
shows $p = r \vee q = r$
⟨*proof*⟩

lemma *lemSmallCardUnderInvertible*:
assumes *invertible f*
and $0 < card\ S \leq 2$
shows $card\ S = card\ (applyToSet\ (asFunc\ f)\ S)$
⟨*proof*⟩

lemma *lemCardOfLineIsBig*:
assumes $x \neq p$
and $onLine\ x\ l \wedge onLine\ p\ l$
shows $\exists\ p1\ p2\ p3\ .\ (onLine\ p1\ l \wedge onLine\ p2\ l \wedge onLine\ p3\ l)$
$\wedge\ (p1 \neq p2 \wedge p2 \neq p3 \wedge p3 \neq p1)$
⟨*proof*⟩

end
end

34 AffineConeLemma

This theory shows that affine approximations preserve "insideness" of points relative to cones.

theory *AffineConeLemma*
imports *KeyLemma TangentLineLemma Cardinalities*
begin

class *AffineConeLemma* = *KeyLemma* + *TangentLineLemma* + *Cardinalities*
begin

lemma *lemInverseOfAffInvertibleIsAffInvertible*:
assumes *affInvertible A*
and $\forall\ x\ y\ .\ A\ x = y \longleftrightarrow A'\ y = x$
shows *affInvertible* A'
⟨*proof*⟩

lemma *lemInsideRegularConeUnderAffInvertible*:
assumes *affInvertible A*
and *insideRegularCone x p*
and *regularConeSet* (*A x*) = *applyToSet* (*asFunc A*) (*regularConeSet x*)
shows *insideRegularCone* (*A x*) (*A p*)
⟨*proof*⟩

end
end

35 NoFTLGR

This theory completes the proof of NoFTLGR.

theory *NoFTLGR*
imports *ObserverConeLemma AffineConeLemma*
begin

class *NoFTLGR* = *ObserverConeLemma* + *AffineConeLemma*
begin

The theorem says: if observer m encounters observer k (so that they are both present at the same spacetime point x), then k

is moving at sub-light speed relative to m. In other words, no observer ever encounters another observer who appears to be moving at or above lightspeed.

theorem *lemNoFTLGR*:
assumes *ass1*: $x \in wline\ m\ m \cap wline\ m\ k$
and *ass2*: $tl\ l\ m\ k\ x$
and *ass3*: $v \in lineVelocity\ l$
and *ass4*: $\exists\ p\ .\ (p \neq x) \wedge (p \in l)$
shows $(lineSlopeFinite\ l) \wedge (sNorm2\ v < 1)$
⟨*proof*⟩

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.

[2] M. Stannett and I. Németi. No faster-than-light observers. *Archive of Formal Proofs*, April 2016. https://isa-afp.org/entries/No_FTL_observers.html, Formal proof development.