

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)

April 18, 2024

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

1	Sorts	3
1.1	Bodies	3
1.2	Quantities	3
2	Points	7
2.1	Squared norms and separation functions	10
2.2	Topological concepts	10
2.3	Lines	10
2.4	Directions	11
2.5	Slopes and slopers	11
3	WorldView	16
4	Functions	17
4.1	Differentiable approximation	19
4.2	<code>lemApproxEqualAtBase</code>	19
5	WorldLine	20

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

6	Translations	21
7	AXIOM: AxSelfMinus	24
8	TangentLines	25
9	Cones	26
10	AXIOM: AxLightMinus	27
11	Proposition1	28
12	AXIOM: AxEField	29
13	Norms	29
	13.1 axTriangleInequality	29
14	AxTriangleInequality	31
15	Sublemma3	31
16	Vectors	32
17	CauchySchwarz	36
18	Matrices	37
19	LinearMaps	38
20	Affine	40
	20.1 Affine approximation	41
21	Sublemma4	44
22	MainLemma	45
23	AXIOM: AxDiff	46
24	TangentLineLemma	47
25	Proposition2	48
26	AXIOM: AxEventMinus	49
27	Proposition3	49
28	ObserverConeLemma	50
29	Quadratics	51
30	Classification	53
31	ReverseCauchySchwarz	60

32 KeyLemma	61
33 Cardinalities	61
34 AffineConeLemma	63
35 NoFTLGR	63

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 ⇒ 'a ⇒ 'a ⇒ bool (- < - < -)
  where (a < b < c) ≡ (a < b) ∧ (b < c)
```

```
abbreviation inRangeOC :: 'a ⇒ 'a ⇒ 'a ⇒ bool (- < - ≤ -)
  where (a < b ≤ c) ≡ (a < b) ∧ (b ≤ c)
```

```
abbreviation inRangeCO :: 'a ⇒ 'a ⇒ 'a ⇒ bool (- ≤ - < -)
  where (a ≤ b < c) ≡ (a ≤ b) ∧ (b < c)
```

```
abbreviation inRangeCC :: 'a ⇒ 'a ⇒ 'a ⇒ 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 $(r \geq 0) \wedge (x = \text{sqr } r)$

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 $(\text{sqr } u) < (\text{sqr } v)$
<proof>

lemma *lemSqrMono*: $(0 \leq u) \wedge (u \leq v) \longrightarrow (\text{sqr } u) \leq (\text{sqr } v)$
<proof>

lemma *lemSqrOrderedStrict*: $(v > 0) \wedge (\text{sqr } u < \text{sqr } v) \longrightarrow (u < v)$
<proof>

lemma *lemSqrOrdered*: $(v \geq 0) \wedge (\text{sqr } u \leq \text{sqr } v) \longrightarrow (u \leq v)$
<proof>

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

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

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

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

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

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

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

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

lemma *lemSqrt1*: $\text{sqr } 1 = 1$
⟨*proof*⟩

lemma *lemSqrt0*: $\text{sqr } 0 = 0$
⟨*proof*⟩

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

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

<proof>

lemma *lemSquareExistsAbove:*
 shows $\exists x > 0 . (\text{sqr } x) > y$
<proof>

lemma *lemSmallSquares:*
 assumes $x > 0$
 shows $\exists y > 0 . (\text{sqr } y < x)$
<proof>

lemma *lemSqrLT1:*
 assumes $0 < x < 1$
 shows $0 < (\text{sqr } x) < x$
<proof>

lemma *lemReducedBound:*
 assumes $x > 0$
 shows $\exists y > 0 . (y < x) \wedge (\text{sqr } y < y) \wedge (y < 1)$
<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* \Rightarrow *'a* **where**
tComponent p \equiv *tval p*

abbreviation *sComponent* :: *'a Point* \Rightarrow *'a Space* **where**
sComponent p \equiv (\mid *svalx* = *xval p*, *svaly* = *yval p*, *svalz* = *zval p* \mid)

abbreviation *mkPoint* :: *'a* \Rightarrow *'a* \Rightarrow *'a* \Rightarrow *'a* \Rightarrow *'a Point* **where**
mkPoint t x y z \equiv (\mid *tval* = *t*, *xval* = *x*, *yval* = *y*, *zval* = *z* \mid)

abbreviation *stPoint* :: *'a* \Rightarrow *'a Space* \Rightarrow *'a Point* **where**
stPoint t s \equiv *mkPoint t (svalx s) (svaly s) (svalz s)*

abbreviation *mkSpace* :: *'a* \Rightarrow *'a* \Rightarrow *'a* \Rightarrow *'a Space* **where**
mkSpace x y z \equiv (\mid *svalx* = *x*, *svaly* = *y*, *svalz* = *z* \mid)

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* \Rightarrow *'a Point* \Rightarrow *'a Point* ($- \oplus -$)
where
(*p* \oplus *q*) \equiv (\mid *tval* = *tval p* + *tval q*,
xval = *xval p* + *xval q*,
yval = *yval p* + *yval q*,
zval = *zval p* + *zval q* \mid)

abbreviation *movebackBy* :: *'a Point* \Rightarrow *'a Point* \Rightarrow *'a Point* ($- \ominus -$)
where
(*p* \ominus *q*) \equiv (\mid *tval* = *tval p* - *tval q*,
xval = *xval p* - *xval q*,
yval = *yval p* - *yval q*,
zval = *zval p* - *zval q* \mid)

abbreviation *sMoveBy* :: *'a Space* \Rightarrow *'a Space* \Rightarrow *'a Space* ($- \oplus s -$)
where
(*p* $\oplus s$ *q*) \equiv (\mid *svalx* = *svalx p* + *svalx q*,
svaly = *svaly p* + *svaly q*,

$$svalz = svalz p + svalz q \text{)}$$

abbreviation $sMovebackBy :: 'a \text{ Space} \Rightarrow 'a \text{ Space} \Rightarrow 'a \text{ Space} (- \ominus s -)$ **where**

$$(p \ominus s q) \equiv (\text{ | } svalx = svalx p - svalx q, \\ svaly = svaly p - svaly q, \\ svalz = svalz p - svalz q \text{ | })$$

abbreviation $scaleBy :: 'a \Rightarrow 'a \text{ Point} \Rightarrow 'a \text{ Point} (- \otimes -)$ **where**

$$scaleBy a p \equiv (\text{ | } tval = a * tval p, xval = a * xval p, \\ yval = a * yval p, zval = a * zval p \text{ | })$$

abbreviation $sScaleBy :: 'a \Rightarrow 'a \text{ Space} \Rightarrow 'a \text{ Space} (- \otimes s -)$ **where**

$$sScaleBy a p \equiv (\text{ | } svalx = a * svalx p, \\ svaly = a * svaly p, \\ svalz = a * svalz p \text{ | })$$

abbreviation $sOrigin :: 'a \text{ Space}$ **where**

$$sOrigin \equiv (\text{ | } svalx = 0, svaly = 0, svalz = 0 \text{ | })$$

abbreviation $origin :: 'a \text{ Point}$ **where**

$$origin \equiv (\text{ | } tval = 0, xval = 0, yval = 0, zval = 0 \text{ | })$$

abbreviation $tUnit :: 'a \text{ Point}$ **where**

$$tUnit \equiv (\text{ | } tval = 1, xval = 0, yval = 0, zval = 0 \text{ | })$$

abbreviation $xUnit :: 'a \text{ Point}$ **where**

$$xUnit \equiv (\text{ | } tval = 0, xval = 1, yval = 0, zval = 0 \text{ | })$$

abbreviation $yUnit :: 'a \text{ Point}$ **where**

$$yUnit \equiv (\text{ | } tval = 0, xval = 0, yval = 1, zval = 0 \text{ | })$$

abbreviation $zUnit :: 'a \text{ Point}$ **where**

$$zUnit \equiv (\text{ | } tval = 0, xval = 0, yval = 0, zval = 1 \text{ | })$$

abbreviation $timeAxis :: 'a \text{ Point set}$ **where**

$$timeAxis \equiv \{ p . xval p = 0 \wedge yval p = 0 \wedge zval p = 0 \}$$

abbreviation $onTimeAxis :: 'a \text{ Point} \Rightarrow bool$

$$\text{where } onTimeAxis p \equiv (p \in timeAxis)$$

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 $norm2 :: 'a \text{ Point} \Rightarrow 'a \text{ where}$
 $norm2\ p \equiv sqr\ (tval\ p) + sqr\ (xval\ p) + sqr\ (yval\ p) + sqr\ (zval\ p)$

abbreviation $sep2 :: 'a \text{ Point} \Rightarrow 'a \text{ Point} \Rightarrow 'a \text{ where}$
 $sep2\ p\ q \equiv norm2\ (p \ominus q)$

abbreviation $sNorm2 :: 'a \text{ Space} \Rightarrow 'a \text{ where}$
 $sNorm2\ s \equiv sqr\ (svalx\ s)$
 $+ \text{ } sqr\ (svaly\ s)$
 $+ \text{ } sqr\ (svalz\ s)$

abbreviation $sSep2 :: 'a \text{ Point} \Rightarrow 'a \text{ Point} \Rightarrow 'a \text{ where}$
 $sSep2\ p\ q \equiv sqr\ (xval\ p - xval\ q)$
 $+ \text{ } sqr\ (yval\ p - yval\ q)$
 $+ \text{ } sqr\ (zval\ p - zval\ q)$

abbreviation $mNorm2 :: 'a \text{ Point} \Rightarrow 'a\ (\| - \|m)$
where $\| p \|m \equiv sqr\ (tval\ p) - sNorm2\ (sComponent\ p)$

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 $inBall :: 'a \text{ Point} \Rightarrow 'a \Rightarrow 'a \text{ Point} \Rightarrow bool$
(- within - of -)
where $inBall\ q\ \varepsilon\ p \equiv sep2\ q\ p < sqr\ \varepsilon$

abbreviation $ball :: 'a \text{ Point} \Rightarrow 'a \Rightarrow 'a \text{ Point set}$
where $ball\ q\ \varepsilon \equiv \{ p . inBall\ q\ \varepsilon\ p \}$

abbreviation $accPoint :: 'a \text{ Point} \Rightarrow 'a \text{ Point set} \Rightarrow bool$
where $accPoint\ p\ s \equiv \forall\ \varepsilon > 0. \exists\ q \in s. (p \neq q) \wedge (inBall\ q\ \varepsilon\ p)$

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 $line :: 'a \text{ Point} \Rightarrow 'a \text{ Point} \Rightarrow 'a \text{ Point set}$
where $line\ base\ drtn \equiv \{ p . \exists\ \alpha . p = (base \oplus (\alpha \otimes drtn)) \}$

abbreviation $lineJoining :: 'a Point \Rightarrow 'a Point \Rightarrow 'a Point set$
where $lineJoining p q \equiv line p (q \ominus p)$

abbreviation $isLine :: 'a Point set \Rightarrow bool$
where $isLine l \equiv \exists b d . (l = line b d)$

abbreviation $sameLine :: 'a Point set \Rightarrow 'a Point set \Rightarrow bool$
where $sameLine l1 l2 \equiv ((isLine l1) \vee (isLine l2)) \wedge (l1 = l2)$

abbreviation $onLine :: 'a Point \Rightarrow 'a Point set \Rightarrow bool$
where $onLine p l \equiv (isLine l) \wedge (p \in l)$

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 $drtn :: 'a Point set \Rightarrow 'a Point set$
where $drtn l = \{ d . \exists p q . (p \neq q) \wedge (onLine p l) \wedge (onLine q l) \wedge (d = (q \ominus p)) \}$

abbreviation $parallelLines :: 'a Point set \Rightarrow 'a Point set \Rightarrow bool$
where $parallelLines l1 l2 \equiv (drtn l1) \cap (drtn l2) \neq \{\}$

abbreviation $parallel :: 'a Point \Rightarrow 'a Point \Rightarrow bool (- \parallel -)$
where $parallel p q \equiv (\exists \alpha \neq 0 . p = (\alpha \otimes q))$

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

abbreviation $slopeFinite :: 'a Point \Rightarrow 'a Point \Rightarrow bool$
where $slopeFinite p q \equiv (tval p \neq tval q)$

abbreviation $slopeInfinite :: 'a Point \Rightarrow 'a Point \Rightarrow bool$
where $slopeInfinite p q \equiv (tval p = tval q)$

abbreviation $lineSlopeFinite :: 'a Point set \Rightarrow bool$
where $lineSlopeFinite l \equiv (\exists x y . (onLine x l) \wedge (onLine y l) \wedge (x \neq y) \wedge (slopeFinite x y))$

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  $\Rightarrow$  'a Point  $\Rightarrow$  'a Point
  where sloper p q = (if (slopeFinite p q) then ((1 / (tval p - tval
q))  $\otimes$  (p  $\ominus$  q))
                    else origin)
```

```
fun velocityJoining :: 'a Point  $\Rightarrow$  'a Point  $\Rightarrow$  'a Space
  where velocityJoining p q = sComponent (sloper p q)
```

```
fun lineVelocity :: 'a Point set  $\Rightarrow$  'a Space set
  where lineVelocity l = { v .  $\exists$  d  $\in$  drtn l . v = velocityJoining origin
d }
```

```
lemma lemNorm2Decomposition:
  shows norm2 u = sqr (tval u) + sNorm2 (sComponent u)
   $\langle$ proof $\rangle$ 
```

```
lemma lemPointDecomposition:
  shows p = (((tval p)  $\otimes$  tUnit)  $\oplus$  (((xval p)  $\otimes$  xUnit)
 $\oplus$  (((yval p)  $\otimes$  yUnit)  $\oplus$  ((zval p)  $\otimes$  zUnit))))
   $\langle$ proof $\rangle$ 
```

```
lemma lemScaleLeftSumDistrib: ((a + b)  $\otimes$  p) = ((a  $\otimes$  p)  $\oplus$  (b  $\otimes$  p))
   $\langle$ proof $\rangle$ 
```

```
lemma lemScaleLeftDiffDistrib: ((a - b)  $\otimes$  p) = ((a  $\otimes$  p)  $\ominus$  (b  $\otimes$  p))
   $\langle$ proof $\rangle$ 
```

```
lemma lemScaleAssoc: ( $\alpha$   $\otimes$  ( $\beta$   $\otimes$  p)) = (( $\alpha$  *  $\beta$ )  $\otimes$  p)
   $\langle$ proof $\rangle$ 
```

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 $p \in \text{line } b \ d$
shows $\text{sameLine } (\text{line } b \ d) \ (\text{line } p \ d)$
⟨proof⟩

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

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

lemma *lemSpatialNullImpliesSpatialOrigin:*
assumes $s\text{Norm2 } s = 0$
shows $s = s\text{Origin}$
⟨proof⟩

lemma *lemNorm2NonNeg: norm2 p ≥ 0*
⟨proof⟩

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

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

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

lemma *lemBallContainsCentre:*

assumes $\varepsilon > 0$

shows x within ε of x

\langle *proof* \rangle

lemma *lemPointLimit:*

assumes $\forall \varepsilon > 0 . (v \text{ within } \varepsilon \text{ of } u)$

shows $v = u$

\langle *proof* \rangle

lemma *lemBallPopulated:*

assumes $e > 0$

shows $\exists y . (y \text{ within } e \text{ of } x) \wedge (y \neq x)$

\langle *proof* \rangle

lemma *lemBallInBall:*

assumes p within x of q

and $0 < x \leq y$

shows p within y of q

\langle *proof* \rangle

lemma *lemSmallPoints:*

assumes $e > 0$

shows $\exists a > 0 . \text{norm2 } (a \otimes p) < \text{sqr } e$

\langle *proof* \rangle

lemma *lemLineJoiningContainsEndPoints:*

assumes $l = \text{lineJoining } x \ p$

shows $\text{onLine } x \ l \wedge \text{onLine } p \ l$

\langle *proof* \rangle

lemma *lemLineAndPoints:*

assumes $p \neq q$

shows $(\text{onLine } p \ l \wedge \text{onLine } q \ l) \longleftrightarrow (l = \text{lineJoining } p \ q)$

\langle *proof* \rangle

lemma *lemLineDefinedByPair:*

assumes $x \neq p$

and $(\text{onLine } p \ l1) \wedge (\text{onLine } x \ l1)$

```

and      (onLine p l2) ∧ (onLine x l2)
shows l1 = l2
⟨proof⟩

```

```

lemma lemDrtn:
assumes { d1, d2 } ⊆ drtn l
shows ∃ α ≠ 0 . d2 = (α ⊗ d1)
⟨proof⟩

```

```

lemma lemLineDeterminedByPointAndDrtn:
assumes (x ≠ p) ∧ (p ∈ l1) ∧ (onLine x l1) ∧ (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 ⇒ Body ⇒ 'a Point ⇒ bool (- sees - at -)
begin

```

```

abbreviation ev :: Body ⇒ 'a Point ⇒ Body set
where ev h x ≡ { b . h sees b at x }

```

```

fun wvt :: Body ⇒ Body ⇒ 'a Point ⇒ 'a Point set
where wvt m k p = { q. (∃ b . (m sees b at p)) ∧ (ev m p = ev k q)
}

```

```

abbreviation wvtFunc :: Body ⇒ Body ⇒ ('a Point ⇒ 'a Point ⇒
bool)
where wvtFunc m k ≡ (λ p q . q ∈ wvt m k p)

```


abbreviation $wvtLine :: Body \Rightarrow Body \Rightarrow 'a Point set \Rightarrow 'a Point set \Rightarrow bool$

where $wvtLine m k l l' \equiv \exists p q p' q' . ($
 $(wvtFunc m k p p') \wedge (wvtFunc m k q q') \wedge$
 $(l = lineJoining p q) \wedge (l' = lineJoining$
 $p' q'))$

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 $bounded :: ('a Point \Rightarrow 'a Point) \Rightarrow bool$
where $bounded f \equiv \exists bnd > 0 . (\forall p . (norm2 (f p) \leq bnd * (norm2 p)))$

abbreviation $composeRel ::$
 $('a Point \Rightarrow 'a Point \Rightarrow bool)$
 $\Rightarrow ('a Point \Rightarrow 'a Point \Rightarrow bool)$
 $\Rightarrow ('a Point \Rightarrow 'a Point \Rightarrow bool)$
where $(composeRel g f) p r \equiv (\exists q . ((f p q) \wedge (g q r)))$

abbreviation *injective* :: ('a Point \Rightarrow 'a Point \Rightarrow bool) \Rightarrow bool
where *injective* f $\equiv \forall x1\ x2\ y1\ y2.$
 $(f\ x1\ y1 \wedge f\ x2\ y2) \wedge (x1 \neq x2) \longrightarrow (y1 \neq y2)$

abbreviation *definedAt* :: ('a Point \Rightarrow 'a Point \Rightarrow bool) \Rightarrow 'a Point
 \Rightarrow bool
where *definedAt* f x $\equiv \exists y . f\ x\ y$

abbreviation *domain* :: ('a Point \Rightarrow 'a Point \Rightarrow bool) \Rightarrow 'a Point
set
where *domain* f $\equiv \{ x . \text{definedAt}\ f\ x \}$

abbreviation *total* :: ('a Point \Rightarrow 'a Point \Rightarrow bool) \Rightarrow bool
where *total* f $\equiv \forall x . (\text{definedAt}\ f\ x)$

abbreviation *surjective* :: ('a Point \Rightarrow 'a Point \Rightarrow bool) \Rightarrow bool
where *surjective* f $\equiv \forall y . \exists x . f\ x\ y$

abbreviation *bijjective* :: ('a Point \Rightarrow 'a Point \Rightarrow bool) \Rightarrow bool
where *bijjective* f $\equiv (\text{injective}\ f) \wedge (\text{surjective}\ f)$

abbreviation *invertible* :: ('a Point \Rightarrow 'a Point) \Rightarrow bool
where *invertible* f $\equiv \forall q . (\exists p . (f\ p = q) \wedge (\forall x . f\ x = q \longrightarrow x = p))$

fun *applyToSet* :: ('a Point \Rightarrow 'a Point \Rightarrow bool) \Rightarrow 'a Point *set* \Rightarrow 'a
Point *set*
where *applyToSet* f s = { q . $\exists p \in s . f\ p\ q$ }

abbreviation *singleValued* :: ('a Point \Rightarrow 'a Point \Rightarrow bool) \Rightarrow 'a Point
 \Rightarrow bool
where *singleValued* f x $\equiv \forall y\ z . (((f\ x\ y) \wedge (f\ x\ z)) \longrightarrow (y = z))$

abbreviation *isFunction* :: ('a Point \Rightarrow 'a Point \Rightarrow bool) \Rightarrow bool
where *isFunction* f $\equiv \forall x . \text{singleValued}\ f\ x$

abbreviation *isTotalFunction* :: ('a Point \Rightarrow 'a Point \Rightarrow bool) \Rightarrow bool
where *isTotalFunction* f $\equiv (\text{total}\ f) \wedge (\text{isFunction}\ f)$

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

fun *asFunc* :: ('a Point \Rightarrow 'a Point) \Rightarrow ('a Point \Rightarrow 'a Point \Rightarrow 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 $diffApprox :: ('a Point \Rightarrow 'a Point \Rightarrow bool) \Rightarrow ('a Point \Rightarrow 'a Point \Rightarrow bool) \Rightarrow 'a Point \Rightarrow bool$
where $diffApprox\ g\ f\ x \equiv (definedAt\ f\ x) \wedge (\forall\ \varepsilon > 0 . (\exists\ \delta > 0 . (\forall\ y . (y\ within\ \delta\ of\ x) \longrightarrow ((definedAt\ f\ y) \wedge (\forall\ u\ v . (f\ y\ u \wedge g\ y\ v) \longrightarrow (sep2\ v\ u) \leq (sqr\ \varepsilon) * sep2\ y\ x))))))$

abbreviation $cts :: ('a Point \Rightarrow 'a Point \Rightarrow bool) \Rightarrow 'a Point \Rightarrow bool$
where $cts\ f\ x \equiv \forall\ y . (f\ x\ y) \longrightarrow (\forall\ \varepsilon > 0 . \exists\ \delta > 0 . (applyToSet\ f\ (ball\ x\ \delta)) \subseteq ball\ y\ \varepsilon)$

fun $invFunc :: ('a Point \Rightarrow 'a Point \Rightarrow bool) \Rightarrow ('a Point \Rightarrow 'a Point \Rightarrow bool)$
where $(invFunc\ f)\ p\ q = f\ q\ p$

lemma $lemBijInv: bijective\ (asFunc\ f) \longleftrightarrow invertible\ f$
 $\langle proof \rangle$

4.2 lemApproxEqualAtBase

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

lemma $lemApproxEqualAtBase:$
assumes $diffApprox\ g\ f\ x$
shows $(f\ x\ y \wedge g\ x\ z) \longrightarrow (y = z)$
 $\langle proof \rangle$

lemma $lemCtsOfCtsIsCts:$
assumes $cts\ f\ x$
and $\forall\ y . (f\ x\ y) \longrightarrow (cts\ g\ y)$
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*) \subseteq *wline* *k* *b*
 \langle *proof* \rangle

lemma *lemFiniteLineVelocityUnique*:
assumes (*u* \in *lineVelocity* *l*) \wedge (*v* \in *lineVelocity* *l*)
and *lineSlopeFinite* *l*
shows *u* = *v*
 \langle *proof* \rangle

end

end

6 Translations

This theory describes translation maps.

theory *Translations*
imports *Functions*
begin

class *Translations* = *Functions*
begin

abbreviation *mkTranslation* :: '*a* *Point* \Rightarrow ('*a* *Point* \Rightarrow '*a* *Point*)
where (*mkTranslation* *t*) \equiv (λ *p* . (*p* \oplus *t*))

abbreviation *translation* :: ('*a* *Point* \Rightarrow '*a* *Point*) \Rightarrow *bool*
where *translation* *T* \equiv \exists *q* . \forall *p* . (*T* *p*) = (*p* \oplus *q*)

lemma *lemMkTrans*: \forall *t* . *translation* (*mkTranslation* *t*)
 \langle *proof* \rangle

lemma *lemInverseTranslation*:
assumes (*T* = *mkTranslation* *t*) \wedge (*T'* = *mkTranslation* (*origin* \ominus *t*))
shows (*T'* \circ *T* = *id*) \wedge (*T* \circ *T'* = *id*)
 \langle *proof* \rangle

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

lemma *lemIdIsTranslation: translation id*
 $\langle proof \rangle$

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

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

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

lemma *lemTranslationInjective:*
 assumes *translation T*
 shows *injective (asFunc T)*
 $\langle proof \rangle$

lemma *lemTranslationSurjective:*
 assumes *translation T*
 shows *surjective (asFunc T)*
 $\langle proof \rangle$

lemma *lemTranslationTotalFunction:*
 assumes *translation T*
 shows *isTotalFunction (asFunc T)*
 $\langle proof \rangle$

lemma *lemTranslationOfLine:*
assumes *translation T*
shows $(\text{applyToSet } (asFunc\ T) (line\ B\ D)) = line\ (T\ B)\ D$
 $\langle proof \rangle$

lemma *lemOnLineTranslation:*
assumes $(translation\ T) \wedge (onLine\ p\ l)$
shows $onLine\ (T\ p)\ (\text{applyToSet } (asFunc\ T)\ l)$
 $\langle proof \rangle$

lemma *lemLineJoiningTranslation:*
assumes *translation T*
shows $\text{applyToSet } (asFunc\ T) (lineJoining\ p\ q) = lineJoining\ (T\ p)\ (T\ q)$
 $\langle proof \rangle$

lemma *lemBallTranslation:*
assumes *translation T*
and $x\ \text{within}\ e\ \text{of}\ y$
shows $(T\ x)\ \text{within}\ e\ \text{of}\ (T\ y)$
 $\langle proof \rangle$

lemma *lemBallTranslationWithBoundary:*
assumes *translation T*
and $sep2\ x\ y \leq \text{sqr}\ e$
shows $sep2\ (T\ x)\ (T\ y) \leq \text{sqr}\ e$
 $\langle proof \rangle$

lemma *lemTranslationIsCts:*
assumes *translation T*
shows $cts\ (asFunc\ T)\ x$
 $\langle proof \rangle$

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* \Rightarrow *'a Point* \Rightarrow *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  $\Rightarrow$  'a Point set  $\Rightarrow$  'a Point
 $\Rightarrow$  bool
```

```
where tangentLine l s x  $\equiv$ 
  (x  $\in$  s)  $\wedge$  (onLine x l)  $\wedge$  (accPoint x s)
 $\wedge$ 
  ( $\exists$  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  $\Rightarrow$  'a Point set  $\Rightarrow$  'a Point
 $\Rightarrow$  bool
```

```
where tangentLineA l s x  $\equiv$ 
  (x  $\in$  s)  $\wedge$  (onLine x l)  $\wedge$  (accPoint x s)
 $\wedge$ 
  ( $\forall$  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  $\Rightarrow$  'a Point  $\Rightarrow$  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  $\Rightarrow$  'a Point  $\Rightarrow$  'a Space  $\Rightarrow$  bool
  where vel wl p v = (  $\exists$  l . ( tangentLine l wl p )  $\wedge$  ( v  $\in$  lineVelocity l ) )
```

lemma *lemTangentLineTranslation:*

```
  assumes translation T
and      tangentLine l s x
shows    tangentLine (applyToSet (asFunc T) l)
           (applyToSet (asFunc T) s) (T x)
   $\langle$ proof $\rangle$ 
```

lemma *lemTangentLineA:*

```
  assumes tangentLine l s x
shows    tangentLineA l s x
   $\langle$ proof $\rangle$ 
```

lemma *lemTangentLineE:*

```
  assumes tangentLineA l s x
and       $\exists$  p  $\neq$  x . onLine p l
shows    tangentLine l s x
   $\langle$ proof $\rangle$ 
```

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 $tl :: 'a \text{ Point set} \Rightarrow \text{Body} \Rightarrow \text{Body} \Rightarrow 'a \text{ Point} \Rightarrow \text{bool}$
where $tl \ l \ m \ b \ x \equiv \text{tangentLine } l \ (\text{wline } m \ b) \ x$

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 $\text{cone} :: \text{Body} \Rightarrow 'a \text{ Point} \Rightarrow 'a \text{ Point} \Rightarrow \text{bool}$
where $\text{cone } m \ x \ p$
 $\equiv \exists \ l . (\text{onLine } p \ l) \wedge (\text{onLine } x \ l) \wedge (\exists \ ph . \text{Ph } ph \wedge tl \ l \ m \ ph \ x)$

abbreviation $\text{regularCone} :: 'a \text{ Point} \Rightarrow 'a \text{ Point} \Rightarrow \text{bool}$
where $\text{regularCone } x \ p \equiv \exists \ l . (\text{onLine } p \ l) \wedge (\text{onLine } x \ l)$
 $\wedge (\exists \ v \in \text{lineVelocity } l . \text{sNorm2 } v = 1)$

abbreviation $\text{coneSet} :: \text{Body} \Rightarrow 'a \text{ Point} \Rightarrow 'a \text{ Point set}$
where $\text{coneSet } m \ x \equiv \{ p . \text{cone } m \ x \ p \}$

abbreviation $\text{regularConeSet} :: 'a \text{ Point} \Rightarrow 'a \text{ Point set}$
where $\text{regularConeSet } x \equiv \{ p . \text{regularCone } x \ p \}$

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 $axLightMinusOLD :: Body \Rightarrow 'a Point \Rightarrow 'a Space \Rightarrow bool$

where $axLightMinusOLD\ m\ p\ v \equiv (m\ sees\ m\ at\ p) \longrightarrow ($
 $(\exists\ ph.\ (Ph\ ph \wedge (vel\ (wline\ m\ ph)\ p\ v))) \longleftrightarrow (sNorm2\ v = 1)$
 $)$

abbreviation $axLightMinus :: Body \Rightarrow 'a Point \Rightarrow 'a Space \Rightarrow bool$

where $axLightMinus\ m\ p\ v \equiv (m\ sees\ m\ at\ p)$
 $\longrightarrow (\forall\ l.\ \forall\ v \in lineVelocity\ l.$
 $(\exists\ ph.\ (Ph\ ph \wedge (tangentLine\ l\ (wline\ m\ ph)\ p))) \longleftrightarrow$
 $(sNorm2\ v = 1))$

end

class $AxLightMinus = axLightMinus +$
assumes $AxLightMinus: \forall\ m\ p\ v.\ axLightMinus\ m\ p\ v$
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 $x \in wline\ m\ m$
shows $cone\ m\ x\ p = regularCone\ x\ p$
 $\langle proof \rangle$

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  $\Rightarrow$  bool
    where axEField  $x \equiv (x \geq 0) \longrightarrow \text{hasRoot } x$ 
end
```

```
class AxEField = axEField +
  assumes AxEField:  $\forall x . \text{axEField } x$ 
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  $\Rightarrow$  'a ( $\| \cdot \|$ )
  where norm  $p \equiv \text{sqrt } (\text{norm2 } p)$ 
```

```
abbreviation sNorm :: 'a Space  $\Rightarrow$  'a
  where sNorm  $p \equiv \text{sqrt } (\text{sNorm2 } p)$ 
```

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:  $(x > 0) \wedge (y > 0) \wedge (\text{sep2 } p \ q \leq \text{sqr } x) \wedge (\text{sep2}$ 
 $r \ q < \text{sqr } y)$ 
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:  $\forall \ p \ q . \text{axTriangleInequality } p \ q$ 
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 + Tangent-
 $Lines$ 
begin

```

```

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

```

```

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

```

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  $\Rightarrow$  'a Point  $\Rightarrow$  'a (-  $\odot$  -)
where dot u v = (tval u)*(tval v) + (xval u)*(xval v) +
(yval u)*(yval v) + (zval u)*(zval v)

```


fun *sdot* :: 'a Space \Rightarrow 'a Space \Rightarrow 'a (- \odot s -)
where *sdot* u v = (svalx u)*(svalx v) + (svaly u)*(svaly v) + (svalz u)*(svalz v)

fun *mdot* :: 'a Point \Rightarrow 'a Point \Rightarrow 'a (- \odot m -)
where *mdot* u v = (tval u)*(tval v) - ((sComponent u) \odot s (sComponent v))

abbreviation *timelike* :: 'a Point \Rightarrow bool
where *timelike* p \equiv mNorm2 p > 0

abbreviation *lightlike* :: 'a Point \Rightarrow bool
where *lightlike* p \equiv (p \neq origin \wedge mNorm2 p = 0)

abbreviation *spacelike* :: 'a Point \Rightarrow bool
where *spacelike* p \equiv mNorm2 p < 0

abbreviation *causal* :: 'a Point \Rightarrow bool
where *causal* p \equiv *timelike* p \vee *lightlike* p

abbreviation *orthog* :: 'a Point \Rightarrow 'a Point \Rightarrow bool
where *orthog* p q \equiv (p \odot q) = 0

abbreviation *orthogs* :: 'a Space \Rightarrow 'a Space \Rightarrow bool
where *orthogs* p q \equiv (p \odot s q) = 0

abbreviation *orthogm* :: 'a Point \Rightarrow 'a Point \Rightarrow bool
where *orthogm* p q \equiv (p \odot m q) = 0

lemma *lemDotDecomposition*:
shows (u \odot v) = (tval u * tval v) + ((sComponent u) \odot s (sComponent v))
 <proof>

lemma *lemDotCommute*: dot u v = dot v u
 <proof>

lemma *lemDotScaleLeft*: dot (a \otimes u) v = a * (dot u v)
 <proof>

lemma *lemDotScaleRight*: dot u (a \otimes v) = a * (dot u v)
 <proof>

lemma *lemDotSumLeft*: $\text{dot } (u \oplus v) w = (\text{dot } u w) + (\text{dot } v w)$
<proof>

lemma *lemDotSumRight*: $\text{dot } u (v \oplus w) = (\text{dot } u v) + (\text{dot } u w)$
<proof>

lemma *lemDotDiffLeft*: $\text{dot } (u \ominus v) w = (\text{dot } u w) - (\text{dot } v w)$
<proof>

lemma *lemDotDiffRight*: $\text{dot } u (v \ominus w) = (\text{dot } u v) - (\text{dot } u w)$
<proof>

lemma *lemNorm2OfSum*: $\text{norm2 } (u \oplus v) = \text{norm2 } u + 2*(u \odot v) + \text{norm2 } v$
<proof>

lemma *lemSDotCommute*: $\text{sdot } u v = \text{sdot } v u$
<proof>

lemma *lemSDotScaleLeft*: $\text{sdot } (a \otimes s u) v = a * (\text{sdot } u v)$
<proof>

lemma *lemSDotScaleRight*: $\text{sdot } u (a \otimes s v) = a * (\text{sdot } u v)$
<proof>

lemma *lemSDotSumLeft*: $\text{sdot } (u \oplus s v) w = (\text{sdot } u w) + (\text{sdot } v w)$
<proof>

lemma *lemSDotSumRight*: $\text{sdot } u (v \oplus s w) = (\text{sdot } u v) + (\text{sdot } u w)$
<proof>

lemma *lemSDotDiffLeft*: $\text{sdot } (u \ominus s v) w = (\text{sdot } u w) - (\text{sdot } v w)$
<proof>

lemma *lemSDotDiffRight*: $\text{sdot } u (v \ominus s w) = (\text{sdot } u v) - (\text{sdot } u w)$
<proof>

lemma *lemMDotDiffLeft*: $\text{mdot } (u \ominus v) w = (\text{mdot } u w) - (\text{mdot } v w)$
<proof>

lemma *lemMDotSumLeft*: $\text{mdot } (u \oplus v) w = (\text{mdot } u w) + (\text{mdot } v w)$
<proof>

lemma *lemMDotScaleLeft*: $\text{mdot } (a \otimes u) v = a * (\text{mdot } u v)$
<proof>

lemma *lemMDotScaleRight*: $\text{mdot } u (a \otimes v) = a * (\text{mdot } u v)$
<proof>

lemma *lemSNorm2OfSum*: $s\text{Norm2 } (u \oplus s v) = s\text{Norm2 } u + 2*(u \odot s v) + s\text{Norm2 } v$
<proof>

lemma *lemSNormNonNeg*:
shows $s\text{Norm } v \geq 0$
<proof>

lemma *lemMNorm2OfSum*: $m\text{Norm2 } (u \oplus v) = m\text{Norm2 } u + 2*(u \odot m v) + m\text{Norm2 } v$
<proof>

lemma *lemMNorm2OfDiff*: $m\text{Norm2 } (u \ominus v) = m\text{Norm2 } u - 2*(u \odot m v) + m\text{Norm2 } v$
<proof>

lemma *lemMNorm2Decomposition*: $m\text{Norm2 } p = (p \odot m p)$
<proof>

lemma *lemMDecomposition*:
assumes $(u \odot m v) \neq 0$
and $m\text{Norm2 } v \neq 0$
and $a = (u \odot m v) / (m\text{Norm2 } v)$
and $up = (a \otimes v)$
and $uo = (u \ominus up)$
shows $u = (up \oplus uo) \wedge \text{parallel } up v \wedge \text{orthogm } uo v \wedge (up \odot m v) = (u \odot m v)$
<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  $abs (dot\ u\ v) \leq (norm\ u) * (norm\ v)$ 
   $\langle proof \rangle$ 
```

```
lemma lemCauchySchwarzSqr4:
  shows  $sqr(dot\ u\ v) \leq (norm2\ u) * (norm2\ v)$ 
   $\langle proof \rangle$ 
```

```
lemma lemCauchySchwarz:
  shows  $abs (sdot\ u\ v) \leq (sNorm\ u) * (sNorm\ v)$ 
   $\langle proof \rangle$ 
```

```
lemma lemCauchySchwarzSqr:
  shows  $sqr(sdot\ u\ v) \leq (sNorm2\ u) * (sNorm2\ v)$ 
   $\langle proof \rangle$ 
```

```
lemma lemCauchySchwarzEquality:
  assumes  $sqr (sdot\ u\ v) = (sNorm2\ u) * (sNorm2\ v)$ 
  and  $u \neq sOrigin \wedge v \neq sOrigin$ 
```

```

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  $\Rightarrow$  'a Point  $\Rightarrow$  'a Point
  where applyMatrix m p = ( $\mid$  tval = dot (trow m) p, xval = dot (xrow
m) p,
                                     yval = dot (yrow m) p, zval = dot (zrow m) p  $\mid$ )

```

```

fun tcol :: 'a Matrix  $\Rightarrow$  'a Point
  where tcol m = ( $\mid$  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**

$linear\ L \equiv (L\ origin = origin)$
 $\wedge (\forall\ a\ p . L\ (a \otimes p) = (a \otimes (L\ p)))$
 $\wedge (\forall\ p\ q . L\ (p \oplus q) = ((L\ p) \oplus (L\ q)))$
 $\wedge (\forall\ p\ q . L\ (p \ominus q) = ((L\ p) \ominus (L\ q)))$

lemma *lemLinearProps:*
assumes *linear L*
shows $(L\ origin = origin) \wedge (L\ (a \otimes p) = (a \otimes (L\ p)))$
 $\wedge (L\ (p \oplus q) = ((L\ p) \oplus (L\ q)))$
 $\wedge (L\ (p \ominus q) = ((L\ p) \ominus (L\ q)))$
 $\langle proof \rangle$

lemma *lemMatrixApplicationIsLinear:* *linear (applyMatrix m)*
 $\langle proof \rangle$

lemma *lemLinearIsMatrixApplication:*
assumes *linear L*
shows $\exists\ m . L = (applyMatrix\ m)$
 $\langle proof \rangle$

lemma *lemLinearIffMatrix:* *linear L \longleftrightarrow ($\exists\ M . L = applyMatrix\ M$)*
 $\langle proof \rangle$

lemma *lemIdIsLinear:* *linear id*
 $\langle proof \rangle$

lemma *lemLinearIsBounded:*
assumes *linear L*
shows *bounded L*
 $\langle proof \rangle$

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  $\exists A' . (linear\ A') \wedge (\forall p\ q. A\ p = q \longleftrightarrow A'\ q = p)$ 
⟨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 $isLinearPart :: ('a Point \Rightarrow 'a Point) \Rightarrow ('a Point \Rightarrow 'a Point) \Rightarrow bool$

where $isLinearPart A L \equiv (affine A) \wedge (linear L) \wedge (\exists T. (translation T \wedge A = T \circ L))$

abbreviation $isTranslationPart :: ('a Point \Rightarrow 'a Point) \Rightarrow ('a Point \Rightarrow 'a Point) \Rightarrow bool$

where $isTranslationPart A T \equiv (affine A) \wedge (translation T) \wedge (\exists L. (linear L \wedge A = T \circ L))$

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 $affineApprox :: ('a Point \Rightarrow 'a Point) \Rightarrow ('a Point \Rightarrow 'a Point \Rightarrow bool) \Rightarrow 'a Point \Rightarrow bool$

where $affineApprox A f x \equiv (isFunction f) \wedge (affInvertible A) \wedge (diffApprox (asFunc A) f x)$

fun $applyAffineToLine :: ('a Point \Rightarrow 'a Point) \Rightarrow 'a Point set \Rightarrow 'a Point set \Rightarrow bool$

where $applyAffineToLine A l l' \longleftrightarrow (affine A) \wedge (\exists T L b d. ((linear L) \wedge (translation T) \wedge (A = T \circ L) \wedge (l = line b d) \wedge (l' = (line (A b) (L d)))))$

abbreviation $affConstantOn :: ('a Point \Rightarrow 'Point) \Rightarrow 'a Point \Rightarrow 'a Point set \Rightarrow bool$

where $affConstantOn A x s \equiv (\exists \varepsilon > 0. \forall y \in s. (y \text{ within } \varepsilon \text{ of } x) \longrightarrow ((A y) = (A x)))$

lemma $lemTranslationPartIsUnique:$

assumes $isTranslationPart A T1$

and $isTranslationPart A T2$

shows $T1 = T2$

$\langle proof \rangle$

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 $\exists T. ((\text{translation } T) \wedge (A = T \circ L))$
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 $(\text{linear } L) \wedge (\text{translation } T) \wedge (A = T \circ L)$
and $x = (b \oplus (a \otimes d))$
shows $A \ x = ((A \ b) \oplus (a \otimes (L \ d)))$
 ⟨*proof*⟩

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

(*asFunc* A) l
⟨proof⟩

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

lemma *lemAffOfAffIsAff*:
 assumes (*affine* A) ∧ (*affine* B)
shows *affine* (B ∘ 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◦T) (composeRel f (asFunc T)) (T' x)
<proof>

```

lemma *lemAffineApproxRangeTranslation:*

```

assumes translation T
and      affineApprox A f x
shows    affineApprox (T◦A) (composeRel (asFunc T) f) x
<proof>

```

lemma *lemAffineIdentity:*

```

assumes affine A
and       $e > 0$ 
and       $\forall y . (y \text{ within } e \text{ of } x) \longrightarrow (A y = y)$ 
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 $(\exists \delta > 0. \forall p. (p \text{ within } \delta \text{ of } x) \longrightarrow (definedAt f p)) \wedge (cts f x)$

<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 fx: 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  $\Rightarrow$  Body  $\Rightarrow$  'a Point  $\Rightarrow$  bool  
      where axDiff m k p  $\equiv$  (definedAt (wvtFunc m k) p)  
         $\longrightarrow$  ( $\exists$  A . (affineApprox A (wvtFunc m k) p))  
  end
```

```
class AxDiff = axDiff +  
  assumes AxDiff:  $\forall$  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  $\Rightarrow$  Body  $\Rightarrow$  'a Point  $\Rightarrow$  bool  
    where axEventMinus m k p  $\equiv$  (m sees k at p)  
           $\longrightarrow$  ( $\exists$  q .  $\forall$  b . (m sees b at p)  $\longleftrightarrow$  (k sees b at q))
```

end

```
class AxEventMinus = axEventMinus +  
  assumes AxEventMinus:  $\forall$  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  $\exists A y . (wvtFunc\ m\ k\ x\ y)$ 
         $\wedge$  (affineApprox A (wvtFunc m k) x)
         $\wedge$  (applyToSet (asFunc A) (coneSet m x)  $\subseteq$  coneSet k y)
         $\wedge$  (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 ⇒ 'a ⇒ 'a ⇒ ('a ⇒ 'a)
  where quadratic a b c ≡ λ x . a*(sqr x) + b*x + c

abbreviation root :: 'a ⇒ 'a ⇒ 'a ⇒ 'a ⇒ bool
  where root a b c r ≡ (quadratic a b c) r = 0

abbreviation roots :: 'a ⇒ 'a ⇒ 'a ⇒ 'a set
  where roots a b c ≡ { r . root a b c r }

abbreviation discriminant :: 'a ⇒ 'a ⇒ 'a ⇒ 'a
  where discriminant a b c ≡ (sqr b) - 4*a*c

abbreviation qcase1 :: 'a ⇒ 'a ⇒ 'a ⇒ bool
  where qcase1 a b c ≡ (a = 0 ∧ b = 0 ∧ c = 0)
abbreviation qcase2 :: 'a ⇒ 'a ⇒ 'a ⇒ bool
  where qcase2 a b c ≡ (a = 0 ∧ b = 0 ∧ c ≠ 0)
abbreviation qcase3 :: 'a ⇒ 'a ⇒ 'a ⇒ bool
  where qcase3 a b c ≡ (a = 0 ∧ b ≠ 0 ∧ (c = 0 ∨ c ≠ 0))
abbreviation qcase4 :: 'a ⇒ 'a ⇒ 'a ⇒ bool
  where qcase4 a b c ≡ (a ≠ 0 ∧ discriminant a b c < 0)
abbreviation qcase5 :: 'a ⇒ 'a ⇒ 'a ⇒ bool
  where qcase5 a b c ≡ (a ≠ 0 ∧ discriminant a b c = 0)
abbreviation qcase6 :: 'a ⇒ 'a ⇒ 'a ⇒ bool
  where qcase6 a b c ≡ (a ≠ 0 ∧ discriminant a b c > 0)

lemma lemQuadRootCondition:
  assumes a ≠ 0
  shows (sqr (2*a*r + b) = discriminant a b c) ⟷ root a b c r
  ⟨proof⟩

lemma lemQuadraticCasesComplete:
  shows qcase1 a b c ∨ qcase2 a b c ∨ qcase3 a b c ∨ qcase4 a b c ∨
    qcase5 a b c ∨ qcase6 a b c
  ⟨proof⟩

```

lemma *lemQCase1*:
assumes *qcase1 a b c*
shows $\forall r . \text{qroot } a \ b \ c \ r$
 $\langle \text{proof} \rangle$

lemma *lemQCase2*:
assumes *qcase2 a b c*
shows $\neg (\exists r . \text{qroot } a \ b \ c \ r)$
 $\langle \text{proof} \rangle$

lemma *lemQCase3*:
assumes *qcase3 a b c*
shows $\text{qroot } a \ b \ c \ r \longleftrightarrow r = -c/b$
 $\langle \text{proof} \rangle$

lemma *lemQCase4*:
assumes *qcase4 a b c*
shows $\neg (\exists r . \text{qroot } a \ b \ c \ r)$
 $\langle \text{proof} \rangle$

lemma *lemQCase5*:
assumes *qcase5 a b c*
shows $\text{qroot } a \ b \ c \ r \longleftrightarrow r = -b/(2*a)$
 $\langle \text{proof} \rangle$

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

lemma *lemQuadraticRootCount*:
assumes $\neg(\text{qcase1 } a \ b \ c)$
shows $\text{finite } (\text{roots } a \ b \ c) \wedge \text{card } (\text{roots } a \ b \ c) \leq 2$
 $\langle \text{proof} \rangle$

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  $\Rightarrow$  'a Point  $\Rightarrow$  bool
  where vertex x p  $\equiv$  (x = p)
```

```
abbreviation insideRegularCone :: 'a Point  $\Rightarrow$  'a Point  $\Rightarrow$  bool
  where insideRegularCone x p  $\equiv$ 
    (slopeFinite x p)  $\wedge$  ( $\exists$  v  $\in$  lineVelocity (lineJoining x p) . sNorm2
v < 1)
```

```
abbreviation outsideRegularCone :: 'a Point  $\Rightarrow$  'a Point  $\Rightarrow$  bool
  where outsideRegularCone x p  $\equiv$ 
    (x  $\neq$  p)  $\wedge$ 
    ((slopeInfinite x p)  $\vee$  ( $\exists$  v  $\in$  lineVelocity (lineJoining x p) .
sNorm2 v > 1))
```

```
abbreviation onRegularCone :: 'a Point  $\Rightarrow$  'a Point  $\Rightarrow$  bool
  where onRegularCone x p  $\equiv$  (x = p)  $\vee$  ( $\exists$  v  $\in$  lineVelocity (lineJoining
x p) . sNorm2 v = 1)
```

lemma *lemDrtnLineJoining*:
assumes $l = \text{lineJoining } x \ p$
and $x \neq p$
shows $(p \ominus x) \in \text{drtn } l$
 $\langle \text{proof} \rangle$

lemma *lemVelocityLineJoining*:
assumes $l = \text{lineJoining } x \ p$
and $v = \text{velocityJoining origin } (p \ominus x)$
and $x \neq p$
shows $v \in \text{lineVelocity } l$
 $\langle \text{proof} \rangle$

lemma *lemSlopeLineJoining*:
assumes $l = \text{lineJoining } p \ q$
and $p \neq q$
shows $\text{lineSlopeFinite } l \longleftrightarrow \text{slopeFinite } p \ q$
 $\langle \text{proof} \rangle$

lemma *lemVelocityJoiningUsingPoints*:
assumes $p \neq q$
shows $\text{velocityJoining } p \ q = \text{velocityJoining origin } (q \ominus p)$
 $\langle \text{proof} \rangle$

lemma *lemLineVelocityNonZeroImpliesFinite*:
assumes $u \in \text{lineVelocity } l$
and $s\text{Norm2 } u \neq 0$
shows $\text{lineSlopeFinite } l$
 $\langle \text{proof} \rangle$

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

lemma *lemSNorm2VelocityJoining:*
assumes *slopeFinite* x p
and $v = \text{velocityJoining } x \ p$
shows $\text{sqr } (\text{tval } p - \text{tval } x) * \text{sNorm2 } v = \text{sNorm2 } (\text{sComponent } (p \ominus x))$
 $\langle \text{proof} \rangle$

lemma *lemOrthogonalSpaceVectorExists:*
shows $\exists w . (w \neq \text{sOrigin}) \wedge (w \odot_s v) = 0$
 $\langle \text{proof} \rangle$

lemma *lemNonParallelVectorsExist:*
shows $\exists w . ((w \neq \text{origin}) \wedge (\text{tval } v = \text{tval } w)) \wedge (\neg (\exists \alpha . (\alpha \neq 0) \wedge v = (\alpha \otimes w)))$
 $\langle \text{proof} \rangle$

lemma *lemConeContainsVertex:*
shows *regularCone* x x
 $\langle \text{proof} \rangle$

lemma *lemConesExist:*
shows *regularConeSet* $x \neq \{\}$
 $\langle \text{proof} \rangle$

lemma *lemRegularCone:*
shows $((x = p) \vee \text{onRegularCone } x \ p) \longleftrightarrow \text{regularCone } x \ p$
 $\langle \text{proof} \rangle$

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

lemma *lemClassification:*
shows $(\text{insideRegularCone } x \ p) \vee (\text{vertex } x \ p \vee \text{outsideRegularCone } x \ p \vee \text{onRegularCone } x \ p)$
 $\langle \text{proof} \rangle$

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 $sqr\ (tval\ (p\ominus x)) - sNorm2\ (sComponent\ (p\ominus x)) = a*(sqr\ \alpha) + b*\alpha + c$
 $\langle proof \rangle$

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

lemma *lemConeCoordinates1:*
shows $p \in regularConeSet\ x \longleftrightarrow norm2\ (p\ominus x) = 2*sqr\ (tval\ p - tval\ x)$
 $\langle proof \rangle$

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 $groot\ a\ b\ c\ \alpha \longleftrightarrow regularCone\ x\ (B \oplus (\alpha \otimes D))$
 $\langle proof \rangle$

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 $(qcase1\ a\ b\ c \longrightarrow S = \{B\})$

$\langle proof \rangle$

lemma *lemLineMeetsCone2:*

assumes $\neg (x \in l)$
and *isLine* l
and $S = l \cap \text{regularConeSet } x$
and $l: l = \text{line } B D$
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 $qcase2 a b c \longrightarrow S = \{\}$
 $\langle proof \rangle$

lemma *lemLineMeetsCone3:*

assumes $\neg (x \in l)$
and *isLine* l
and $S = l \cap \text{regularConeSet } x$
and $l: l = \text{line } B D$
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 $c: c = \text{sqr } (tval X) - sNorm2 (sComponent X)$
and $y3: y3 = (B \oplus ((-c/b) \otimes D))$
shows $qcase3 a b c \longrightarrow S = \{y3\}$
 $\langle proof \rangle$

lemma *lemLineMeetsCone4:*

assumes $\neg (x \in l)$
and *isLine* l
and $S = l \cap \text{regularConeSet } x$
and $l: l = \text{line } B D$
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 $c: c = \text{sqr } (tval X) - sNorm2 (sComponent X)$
shows $(qcase4 a b c \longrightarrow S = \{\})$
 $\langle proof \rangle$

lemma *lemLineMeetsCone5*:
assumes $\neg (x \in l)$
and *isLine* l
and $S = l \cap \text{regularConeSet } x$
and $l: l = \text{line } B D$
and $X: X = (B \ominus x)$
and $a: a = m\text{Norm2 } D$
and $b: b = 2*(\text{tval } X)*(\text{tval } D) - 2*((\text{sComponent } D) \odot_s (\text{sComponent } X))$
and $c: c = \text{sqr } (\text{tval } X) - s\text{Norm2 } (\text{sComponent } X)$
and $y5: y5 = (B \oplus ((-b/(2*a)) \otimes D))$
shows $(\text{qcase5 } a b c \longrightarrow S = \{y5\})$
 $\langle \text{proof} \rangle$

lemma *lemLineMeetsCone6*:
assumes $\neg (x \in l)$
and *isLine* l
and $S = l \cap \text{regularConeSet } x$
and $l: l = \text{line } B D$
and $X: X = (B \ominus x)$
and $a: a = m\text{Norm2 } D$
and $b: b = 2*(\text{tval } X)*(\text{tval } D) - 2*((\text{sComponent } D) \odot_s (\text{sComponent } X))$
and $c: c = \text{sqr } (\text{tval } X) - s\text{Norm2 } (\text{sComponent } X)$
and $ym: ym = (B \oplus (((-b - (\text{sqr } (\text{discriminant } a b c))) / (2*a)) \otimes D))$
and $yp: yp = (B \oplus (((-b + (\text{sqr } (\text{discriminant } a b c))) / (2*a)) \otimes D))$
shows $(\text{qcase6 } a b c \longrightarrow (ym \neq yp) \wedge S = \{ym, yp\})$
 $\langle \text{proof} \rangle$

lemma *lemConeLemma1*:
assumes $\neg (x \in l)$
and *isLine* l
and $S = l \cap \text{regularConeSet } x$
shows $\text{finite } S \wedge \text{card } S \leq 2$
 $\langle \text{proof} \rangle$

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

⟨proof⟩

lemma *lemLineInsideRegularConeHasFiniteSlope:*

assumes *insideRegularCone* $x\ p$
and $l = \text{lineJoining } x\ p$
shows *lineSlopeFinite* l
⟨proof⟩

lemma *lemInvertibleOnMeet:*

assumes *invertible* f
and $S = A \cap B$
shows $\text{applyToSet } (asFunc\ f)\ S = (\text{applyToSet } (asFunc\ f)\ A) \cap$
 $(\text{applyToSet } (asFunc\ f)\ B)$
⟨proof⟩

lemma *lemInsideCone:*

shows *insideRegularCone* $x\ p \longleftrightarrow$
 $\neg(\text{vertex } x\ p \vee \text{outsideRegularCone } x\ p \vee \text{onRegularCone } x$
 $p)$
⟨proof⟩

lemma *lemOnRegularConeIff:*

assumes $l = \text{lineJoining } x\ p$
shows *onRegularCone* $x\ p \longleftrightarrow (l \cap \text{regularConeSet } x = l)$
⟨proof⟩

lemma *lemOutsideRegularConeImplies:*

shows *outsideRegularCone* $x\ p$
 $\longrightarrow (\exists\ l\ p' . (p' \neq p) \wedge \text{onLine } p'\ l \wedge \text{onLine } p\ l$
 $\wedge (l \cap \text{regularConeSet } x = \{\}))$
⟨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  $s = sComponent ((1/tval v) \otimes v)$ 
shows  $(0 \leq sNorm2 s < 1) \wedge (tval ((1/tval v) \otimes v) = 1)$ 
  <proof>
```

```
lemma lemReverseCauchySchwarz:
```

```
  assumes timelike X ∧ timelike D
shows  $sqr (X \odot_m D) \geq (mNorm2 X) * (mNorm2 D)$ 
  <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 $D \neq \text{origin}$

and $l = \text{line } p \ D$

shows $0 < \text{card } (l \cap \text{regularConeSet } x) \leq 2$

<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 *bijjective f*
and *isFunction f*
and $s \subseteq \text{domain } f$
and $\text{card } s = 2$
shows $\text{card } (\text{applyToSet } f s) = 2$
 ⟨*proof*⟩

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

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

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

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

end
end

34 AffineConeLemma

This theory shows that affine approximations preserve "inside-ness" 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'
   $\langle proof \rangle$ 

lemma lemInsideRegularConeUnderAffInvertible:
  assumes affInvertible A
and      insideRegularCone x p
and      regularConeSet (A x) = applyToSet (asFunc A) (regularConeSet x)
shows    insideRegularCone (A x) (A p)
   $\langle proof \rangle$ 

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.