

# Affine Arithmetic

Fabian Immler

March 17, 2025

## Abstract

We give a formalization of affine forms [1, 2] as abstract representations of zonotopes. We provide affine operations as well as overapproximations of some non-affine operations like multiplication and division. Expressions involving those operations can automatically be turned into (executable) functions approximating the original expression in affine arithmetic.

Moreover we give a verified implementation of a functional algorithm to compute the intersection of a zonotope with a hyperplane, as described in the paper [3].

## Contents

0.1	<i>sum-list</i>	5
0.2	Radiant and Degree	5
<b>1</b>	<b>Euclidean Space: Executability</b>	<b>6</b>
1.1	Ordered representation of Basis and Rounding of Components	6
1.2	Instantiations	7
1.3	Representation as list	9
1.4	Bounded Linear Functions	17
1.5	bounded linear functions	17
<b>2</b>	<b>Affine Form</b>	<b>20</b>
2.1	Auxiliary developments	21
2.2	Partial Deviations	22
2.3	Affine Forms	23
2.4	Evaluation, Range, Joint Range	23
2.5	Domain	25
2.6	Least Fresh Index	25
2.7	Total Deviation	27
2.8	Binary Pointwise Operations	27
2.9	Addition	27
2.10	Total Deviation	28

2.11	Unary Operations	28
2.12	Pointwise Scaling of Partial Deviations	28
2.13	Partial Deviations Scale Pointwise	29
2.14	Pointwise Unary Minus	29
2.15	Constant	30
2.16	Inner Product	30
2.17	Inner Product Pair	30
2.18	Update	31
2.19	Inf/Sup	31
2.20	Minkowski Sum	32
2.21	Splitting	35
2.22	From List of Generators	36
2.22.1	(reverse) ordered coefficients as list	41
2.23	2d zonotopes	44
2.24	Intervals	45
<b>3</b>	<b>Operations on Expressions</b>	<b>47</b>
3.1	Approximating Expression*s*	47
3.2	Syntax	48
3.3	Derived symbols	49
3.4	Constant Folding	50
3.5	Free Variables	51
3.6	Derivatives	56
3.7	Definition of Approximating Function using Affine Arithmetic	64
<b>4</b>	<b>Straight Line Programs</b>	<b>80</b>
4.1	Definition	80
4.2	Reification as straight line program (with common subexpression elimination)	80
4.3	better code equations for construction of large programs	85
<b>5</b>	<b>Approximation with Affine Forms</b>	<b>88</b>
5.1	Approximate Operations	90
5.1.1	set of generated endpoints	90
5.1.2	Approximate total deviation	90
5.1.3	truncate partial deviations	91
5.1.4	truncation with error bound	92
5.1.5	general affine operation	94
5.1.6	Inf/Sup	95
5.2	Min Range approximation	96
5.2.1	Addition	97
5.2.2	Scaling	98
5.2.3	Multiplication	98
5.2.4	Inverse	100

5.3	Reduction (Summarization of Coefficients) . . . . .	103
5.4	Splitting with heuristics . . . . .	106
5.5	Approximate Min Range - Kind Of Trigonometric Functions . . . . .	109
5.6	Power, TODO: compare with Min-range approximation?! . . . . .	114
5.7	Generic operations on Affine Forms in Euclidean Space . . . . .	125
<b>6</b>	<b>Counterclockwise</b>	<b>126</b>
6.1	Auxiliary Lemmas . . . . .	126
6.2	Sort Elements of a List . . . . .	127
6.3	Abstract CCW Systems . . . . .	130
<b>7</b>	<b>CCW Vector Space</b>	<b>131</b>
<b>8</b>	<b>CCW for Nonaligned Points in the Plane</b>	<b>133</b>
8.1	Determinant . . . . .	134
8.2	Strict CCW Predicate . . . . .	137
8.3	Collinearity . . . . .	137
8.4	Polygonal chains . . . . .	139
8.5	Dirvec: Inverse of Polychain . . . . .	142
8.6	Polychain of Sorted ( <i>polychain-of, ccw'.sortedP</i> ) . . . . .	142
<b>9</b>	<b>CCW for Arbitrary Points in the Plane</b>	<b>145</b>
9.1	Interpretation of Knuth's axioms in the plane . . . . .	145
9.2	Order prover setup . . . . .	148
9.3	Contradictions . . . . .	148
<b>10</b>	<b>Intersection</b>	<b>152</b>
10.1	Polygons and <i>ccw, Counterclockwise-2D-Arbitrary.lex, psi, coll</i> . . . . .	152
10.2	Orient all entries . . . . .	153
10.3	Lowest Vertex . . . . .	154
10.4	Collinear Generators . . . . .	155
10.5	Independent Generators . . . . .	155
10.6	Independent Oriented Generators . . . . .	157
10.7	Half Segments . . . . .	158
10.8	Mirror . . . . .	162
10.9	Full Segments . . . . .	164
10.10	Continuous Generalization . . . . .	166
10.11	Intersection of Vertical Line with Segment . . . . .	166
10.12	Bounds on Vertical Intersection with Oriented List of Segments . . . . .	169
10.13	Bounds on Vertical Intersection with General List of Segments . . . . .	170
10.14	Approximation from Orthogonal Directions . . . . .	171
10.15	"Completeness" of Intersection . . . . .	172

<b>11 Implementation</b>	<b>173</b>
11.1 Reverse Sorted, Distinct Association Lists . . . . .	173
11.2 Degree . . . . .	174
11.3 Auxiliary Definitions . . . . .	174
11.4 Pointwise Addition . . . . .	175
11.5 prod of pdevs . . . . .	176
11.6 Set of Coefficients . . . . .	176
11.7 Domain . . . . .	176
11.8 Application . . . . .	177
11.9 Total Deviation . . . . .	177
11.10 Minkowski Sum . . . . .	177
11.11 Unary Operations . . . . .	178
11.12 Filter . . . . .	179
11.13 Constant . . . . .	179
11.14 Update . . . . .	180
11.15 Approximate Total Deviation . . . . .	181
11.16 Equality . . . . .	181
11.17 From List of Generators . . . . .	181
<b>12 Optimizations for Code Integer</b>	<b>182</b>
<b>13 Optimizations for Code Float</b>	<b>183</b>
<b>14 Target Language debug messages</b>	<b>184</b>
14.1 Printing . . . . .	185
14.2 Write to File . . . . .	185
14.3 Show for Floats . . . . .	185
14.4 Convert Float to Decimal number . . . . .	186
14.4.1 Version that should be easy to prove correct, but slow!	186
14.5 Trusted, but faster version . . . . .	186
14.6 gnuplot output . . . . .	188
14.6.1 vector output of 2D zonotope . . . . .	188
<b>15 Dyadic Rational Representation of Real</b>	<b>189</b>
<b>16 Examples</b>	<b>192</b>
<b>17 Examples on Proving Inequalities</b>	<b>192</b>
<b>18 Examples: Intersection of Zonotopes with Hyperplanes</b>	<b>195</b>
18.1 Example . . . . .	196
<b>theory</b> <i>Affine-Arithmetic-Auxiliarities</i>	
<b>imports</b> <i>HOL-Analysis.Multivariate-Analysis</i>	
<b>begin</b>	

## 0.1 *sum-list*

**lemma** *sum-list-nth-eqI*:

**fixes** *xs ys::'a::monoid-add list*

**shows**

$length\ xs = length\ ys \implies (\bigwedge x\ y. (x, y) \in set\ (zip\ xs\ ys) \implies x = y) \implies$   
 $sum-list\ xs = sum-list\ ys$

*<proof>*

**lemma** *fst-sum-list*:  $fst\ (sum-list\ xs) = sum-list\ (map\ fst\ xs)$

*<proof>*

**lemma** *snd-sum-list*:  $snd\ (sum-list\ xs) = sum-list\ (map\ snd\ xs)$

*<proof>*

**lemma** *take-greater-eqI*:  $take\ c\ xs = take\ c\ ys \implies c \geq a \implies take\ a\ xs = take\ a\ ys$

*<proof>*

**lemma** *take-max-eqD*:

$take\ (max\ a\ b)\ xs = take\ (max\ a\ b)\ ys \implies take\ a\ xs = take\ a\ ys \wedge take\ b\ xs = take\ b\ ys$

*<proof>*

**lemma** *take-Suc-eq*:  $take\ (Suc\ n)\ xs = (if\ n < length\ xs\ then\ take\ n\ xs\ @\ [xs\ !\ n]\ else\ xs)$

*<proof>*

## 0.2 Radiant and Degree

**definition** *rad-of*  $w = w * pi / 180$

**definition** *deg-of*  $w = 180 * w / pi$

**lemma** *rad-of-inverse[simp]*:  $deg-of\ (rad-of\ w) = w$

**and** *deg-of-inverse[simp]*:  $rad-of\ (deg-of\ w) = w$

*<proof>*

**lemma** *deg-of-monoI*:  $x \leq y \implies deg-of\ x \leq deg-of\ y$

*<proof>*

**lemma** *rad-of-monoI*:  $x \leq y \implies rad-of\ x \leq rad-of\ y$

*<proof>*

**lemma** *deg-of-strict-monoI*:  $x < y \implies deg-of\ x < deg-of\ y$

*<proof>*

**lemma** *rad-of-strict-monoI*:  $x < y \implies rad-of\ x < rad-of\ y$

*<proof>*

**lemma** *deg-of-mono*[simp]:  $\text{deg-of } x \leq \text{deg-of } y \longleftrightarrow x \leq y$   
(proof)

**lemma** *rad-of-mono*[simp]:  $\text{rad-of } x \leq \text{rad-of } y \longleftrightarrow x \leq y$   
(proof)

**lemma** *deg-of-strict-mono*[simp]:  $\text{deg-of } x < \text{deg-of } y \longleftrightarrow x < y$   
(proof)

**lemma** *rad-of-strict-mono*[simp]:  $\text{rad-of } x < \text{rad-of } y \longleftrightarrow x < y$   
(proof)

**lemma** *rad-of-lt-iff*:  $\text{rad-of } d < r \longleftrightarrow d < \text{deg-of } r$   
**and** *rad-of-gt-iff*:  $\text{rad-of } d > r \longleftrightarrow d > \text{deg-of } r$   
**and** *rad-of-le-iff*:  $\text{rad-of } d \leq r \longleftrightarrow d \leq \text{deg-of } r$   
**and** *rad-of-ge-iff*:  $\text{rad-of } d \geq r \longleftrightarrow d \geq \text{deg-of } r$   
(proof)

**end**

## 1 Euclidean Space: Executability

**theory** *Executable-Euclidean-Space*

**imports**

*HOL-Analysis.Multivariate-Analysis*

*List-Index.List-Index*

*HOL-Library.Float*

*HOL-Library.Code-Cardinality*

*Affine-Arithmetic-Auxiliarities*

**begin**

### 1.1 Ordered representation of Basis and Rounding of Components

**class** *executable-euclidean-space* = *ordered-euclidean-space* +

**fixes** *Basis-list* *eucl-down* *eucl-truncate-down* *eucl-truncate-up*

**assumes** *eucl-down-def*:

$\text{eucl-down } p \ b = (\sum i \in \text{Basis}. \text{round-down } p \ (b \cdot i) *_R i)$

**assumes** *eucl-truncate-down-def*:

$\text{eucl-truncate-down } q \ b = (\sum i \in \text{Basis}. \text{truncate-down } q \ (b \cdot i) *_R i)$

**assumes** *eucl-truncate-up-def*:

$\text{eucl-truncate-up } q \ b = (\sum i \in \text{Basis}. \text{truncate-up } q \ (b \cdot i) *_R i)$

**assumes** *Basis-list*[simp]:  $\text{set } \text{Basis-list} = \text{Basis}$

**assumes** *distinct-Basis-list*[simp]:  $\text{distinct } \text{Basis-list}$

**begin**

**lemma** *length-Basis-list*:

$\text{length } \text{Basis-list} = \text{card } \text{Basis}$

(proof)

**end**

**lemma** *eucl-truncate-down-zero*[simp]: *eucl-truncate-down*  $p$   $0 = 0$   
⟨*proof*⟩

**lemma** *eucl-truncate-up-zero*[simp]: *eucl-truncate-up*  $p$   $0 = 0$   
⟨*proof*⟩

## 1.2 Instantiations

**instantiation** *real::executable-euclidean-space*  
**begin**

**definition** *Basis-list-real* :: *real list* **where**  
*Basis-list-real* = [1]

**definition** *eucl-down*  $prec$   $b = round-down$   $prec$   $b$

**definition** *eucl-truncate-down*  $prec$   $b = truncate-down$   $prec$   $b$

**definition** *eucl-truncate-up*  $prec$   $b = truncate-up$   $prec$   $b$

**instance** ⟨*proof*⟩

**end**

**instantiation** *prod::(executable-euclidean-space, executable-euclidean-space)*  
*executable-euclidean-space*

**begin**

**definition** *Basis-list-prod* :: (*'a* × *'b*) *list* **where**

*Basis-list-prod* =  
zip *Basis-list* (replicate (length (*Basis-list::'a list*)) 0) @  
zip (replicate (length (*Basis-list::'b list*)) 0) *Basis-list*

**definition** *eucl-down*  $p$   $a = (eucl-down$   $p$  (*fst*  $a$ ), *eucl-down*  $p$  (*snd*  $a$ ))

**definition** *eucl-truncate-down*  $p$   $a = (eucl-truncate-down$   $p$  (*fst*  $a$ ), *eucl-truncate-down*  $p$  (*snd*  $a$ ))

**definition** *eucl-truncate-up*  $p$   $a = (eucl-truncate-up$   $p$  (*fst*  $a$ ), *eucl-truncate-up*  $p$  (*snd*  $a$ ))

**instance**

⟨*proof*⟩

**end**

**lemma** *eucl-truncate-down-Basis*[simp]:

$i \in \text{Basis} \implies eucl-truncate-down$   $e$   $x \cdot i = truncate-down$   $e$  ( $x \cdot i$ )  
⟨*proof*⟩

**lemma** *eucl-truncate-down-correct*:

*dist* ( $x::'a::\text{executable-euclidean-space}$ ) (*eucl-down*  $e$   $x$ )  $\in$   
 $\{0..sqrt (DIM('a)) * 2 \text{ powr of-int } (- e)\}$   
*<proof>*

**lemma** *eucl-down*: *eucl-down*  $e$  ( $x::'a::\text{executable-euclidean-space}$ )  $\leq x$   
*<proof>*

**lemma** *eucl-truncate-down*: *eucl-truncate-down*  $e$  ( $x::'a::\text{executable-euclidean-space}$ )  
 $\leq x$   
*<proof>*

**lemma** *eucl-truncate-down-le*:  
 $x \leq y \implies \text{eucl-truncate-down } w \ x \leq (y::'a::\text{executable-euclidean-space})$   
*<proof>*

**lemma** *eucl-truncate-up-Basis[simp]*:  $i \in \text{Basis} \implies \text{eucl-truncate-up } e \ x \cdot i =$   
*truncate-up*  $e$  ( $x \cdot i$ )  
*<proof>*

**lemma** *eucl-truncate-up*:  $x \leq \text{eucl-truncate-up } e$  ( $x::'a::\text{executable-euclidean-space}$ )  
*<proof>*

**lemma** *eucl-truncate-up-le*:  $x \leq y \implies x \leq \text{eucl-truncate-up } e$  ( $y::'a::\text{executable-euclidean-space}$ )  
*<proof>*

**lemma** *eucl-truncate-down-mono*:

**fixes**  $x::'a::\text{executable-euclidean-space}$   
**shows**  $x \leq y \implies \text{eucl-truncate-down } p \ x \leq \text{eucl-truncate-down } p \ y$   
*<proof>*

**lemma** *eucl-truncate-up-mono*:

**fixes**  $x::'a::\text{executable-euclidean-space}$   
**shows**  $x \leq y \implies \text{eucl-truncate-up } p \ x \leq \text{eucl-truncate-up } p \ y$   
*<proof>*

**lemma** *infnorm[code]*:

**fixes**  $x::'a::\text{executable-euclidean-space}$   
**shows**  $\text{infnorm } x = \text{fold max } (\text{map } (\lambda i. \text{abs } (x \cdot i)) \text{Basis-list}) \ 0$   
*<proof>*

**declare** *Inf-real-def*[code del]

**declare** *Sup-real-def*[code del]

**declare** *Inf-prod-def*[code del]

**declare** *Sup-prod-def*[code del]

**declare** [[code abort: *Inf::real set*  $\Rightarrow$  *real*]]

**declare** [[code abort: *Sup::real set*  $\Rightarrow$  *real*]]

**declare** [[code abort: *Inf::('a::Inf \* 'b::Inf) set*  $\Rightarrow$  *'a \* 'b*]]

**declare** [[code abort: *Sup::('a::Sup \* 'b::Sup) set*  $\Rightarrow$  *'a \* 'b*]]

**lemma** *nth-Basis-list-in-Basis*[simp]:

$n < \text{length } (\text{Basis-list}::'a::\text{executable-euclidean-space list}) \implies \text{Basis-list} ! n \in (\text{Basis}::'a \text{ set})$   
 ⟨proof⟩

### 1.3 Representation as list

**lemma** *nth-eq-iff-index*:

$\text{distinct } xs \implies n < \text{length } xs \implies xs ! n = i \iff n = \text{index } xs \ i$   
 ⟨proof⟩

**lemma** *in-Basis-index-Basis-list*:  $i \in \text{Basis} \implies i = \text{Basis-list} ! \text{index } \text{Basis-list} \ i$   
 ⟨proof⟩

**lemmas** [simp] = *length-Basis-list*

**lemma** *sum-Basis-sum-nth-Basis-list*:

$(\sum_{i \in \text{Basis}} f \ i) = (\sum_{i < \text{DIM}('a::\text{executable-euclidean-space})} f \ ((\text{Basis-list}::'a \ \text{list}) ! i))$   
 ⟨proof⟩

**definition** *eucl-of-list*  $xs = (\sum (x, i) \leftarrow \text{zip } xs \ \text{Basis-list}. x *_{\mathbb{R}} i)$

**lemma** *eucl-of-list-nth*:

**assumes**  $\text{length } xs = \text{DIM}('a)$   
**shows**  $\text{eucl-of-list } xs = (\sum_{i < \text{DIM}('a::\text{executable-euclidean-space})} (xs ! i) *_{\mathbb{R}} ((\text{Basis-list}::'a \ \text{list}) ! i))$   
 ⟨proof⟩

**lemma** *eucl-of-list-inner*:

**fixes**  $i::'a::\text{executable-euclidean-space}$   
**assumes**  $i \in \text{Basis}$   
**assumes**  $l: \text{length } xs = \text{DIM}('a)$   
**shows**  $\text{eucl-of-list } xs \cdot i = xs ! (\text{index } \text{Basis-list} \ i)$   
 ⟨proof⟩

**lemma** *inner-eucl-of-list*:

**fixes**  $i::'a::\text{executable-euclidean-space}$   
**assumes**  $i \in \text{Basis}$   
**assumes**  $l: \text{length } xs = \text{DIM}('a)$   
**shows**  $i \cdot \text{eucl-of-list } xs = xs ! (\text{index } \text{Basis-list} \ i)$   
 ⟨proof⟩

**definition** *list-of-eucl*  $x = \text{map } ((\cdot) \ x) \ \text{Basis-list}$

**lemma** *index-Basis-list-nth*[simp]:

$i < \text{DIM}('a::\text{executable-euclidean-space}) \implies \text{index } \text{Basis-list} \ ((\text{Basis-list}::'a \ \text{list})$

$! i) = i$   
 ⟨proof⟩

**lemma** *list-of-eucl-eucl-of-list*[simp]:  
 $length\ xs = DIM('a::executable-euclidean-space) \implies list-of-eucl\ (eucl-of-list\ xs::'a)$   
 $= xs$   
 ⟨proof⟩

**lemma** *eucl-of-list-list-of-eucl*[simp]:  
 $eucl-of-list\ (list-of-eucl\ x) = x$   
 ⟨proof⟩

**lemma** *length-list-of-eucl*[simp]:  $length\ (list-of-eucl\ (x::'a::executable-euclidean-space))$   
 $= DIM('a)$   
 ⟨proof⟩

**lemma** *list-of-eucl-nth*[simp]:  $n < DIM('a::executable-euclidean-space) \implies list-of-eucl$   
 $x\ !\ n = x \cdot (Basis-list\ !\ n::'a)$   
 ⟨proof⟩

**lemma** *nth-ge-len*:  $n \geq length\ xs \implies xs\ !\ n = []\ !\ (n - length\ xs)$   
 ⟨proof⟩

**lemma** *list-of-eucl-nth-if*:  $list-of-eucl\ x\ !\ n = (if\ n < DIM('a::executable-euclidean-space)$   
 $then\ x \cdot (Basis-list\ !\ n::'a)\ else\ []\ !\ (n - DIM('a)))$   
 ⟨proof⟩

**lemma** *list-of-eucl-eq-iff*:  
 $list-of-eucl\ (x::'a::executable-euclidean-space) = list-of-eucl\ (y::'b::executable-euclidean-space)$   
 $\longleftrightarrow$   
 $(DIM('a) = DIM('b) \wedge (\forall i < DIM('b). x \cdot Basis-list\ !\ i = y \cdot Basis-list\ !\ i))$   
 ⟨proof⟩

**lemma** *eucl-le-Basis-list-iff*:  
 $(x::'a::executable-euclidean-space) \leq y \longleftrightarrow$   
 $(\forall i < DIM('a). x \cdot Basis-list\ !\ i \leq y \cdot Basis-list\ !\ i)$   
 ⟨proof⟩

**lemma** *eucl-of-list-inj*:  $length\ xs = DIM('a::executable-euclidean-space) \implies length$   
 $ys = DIM('a) \implies$   
 $(eucl-of-list\ xs::'a) = eucl-of-list\ (ys) \implies xs = ys$   
 ⟨proof⟩

**lemma** *eucl-of-list-map-plus*[simp]:  
**assumes** [simp]:  $length\ xs = DIM('a::executable-euclidean-space)$   
**shows**  $(eucl-of-list\ (map\ (\lambda x. f\ x + g\ x)\ xs)::'a) =$   
 $eucl-of-list\ (map\ f\ xs) + eucl-of-list\ (map\ g\ xs)$   
 ⟨proof⟩

**lemma** *eucl-of-list-map-uminus*[simp]:  
**assumes** [simp]:  $\text{length } xs = \text{DIM}('a::\text{executable-euclidean-space})$   
**shows**  $(\text{eucl-of-list } (\text{map } (\lambda x. - f x) xs)::'a) = - \text{eucl-of-list } (\text{map } f xs)$   
 $\langle \text{proof} \rangle$

**lemma** *eucl-of-list-map-mult-left*[simp]:  
**assumes** [simp]:  $\text{length } xs = \text{DIM}('a::\text{executable-euclidean-space})$   
**shows**  $(\text{eucl-of-list } (\text{map } (\lambda x. r * f x) xs)::'a) = r *_R \text{eucl-of-list } (\text{map } f xs)$   
 $\langle \text{proof} \rangle$

**lemma** *eucl-of-list-map-mult-right*[simp]:  
**assumes** [simp]:  $\text{length } xs = \text{DIM}('a::\text{executable-euclidean-space})$   
**shows**  $(\text{eucl-of-list } (\text{map } (\lambda x. f x * r) xs)::'a) = r *_R \text{eucl-of-list } (\text{map } f xs)$   
 $\langle \text{proof} \rangle$

**lemma** *eucl-of-list-map-divide-right*[simp]:  
**assumes** [simp]:  $\text{length } xs = \text{DIM}('a::\text{executable-euclidean-space})$   
**shows**  $(\text{eucl-of-list } (\text{map } (\lambda x. f x / r) xs)::'a) = \text{eucl-of-list } (\text{map } f xs) /_R r$   
 $\langle \text{proof} \rangle$

**lemma** *eucl-of-list-map-const*[simp]:  
**assumes** [simp]:  $\text{length } xs = \text{DIM}('a::\text{executable-euclidean-space})$   
**shows**  $(\text{eucl-of-list } (\text{map } (\lambda x. c) xs)::'a) = c *_R \text{One}$   
 $\langle \text{proof} \rangle$

**lemma** *replicate-eq-list-of-eucl-zero*:  $\text{replicate } \text{DIM}('a::\text{executable-euclidean-space})$   
 $0 = \text{list-of-eucl } (0::'a)$   
 $\langle \text{proof} \rangle$

**lemma** *eucl-of-list-append-zeroes*[simp]:  $\text{eucl-of-list } (xs @ \text{replicate } n \ 0) = \text{eucl-of-list}$   
 $xs$   
 $\langle \text{proof} \rangle$

**lemma** *Basis-prodD*:  
**assumes**  $(i, j) \in \text{Basis}$   
**shows**  $i \in \text{Basis} \wedge j = 0 \vee i = 0 \wedge j \in \text{Basis}$   
 $\langle \text{proof} \rangle$

**lemma** *eucl-of-list-take-DIM*[simp]:  
**assumes**  $d = \text{DIM}('b::\text{executable-euclidean-space})$   
**shows**  $(\text{eucl-of-list } (\text{take } d \ xs)::'b) = (\text{eucl-of-list } xs)$   
 $\langle \text{proof} \rangle$

**lemma** *eucl-of-list-eqI*:  
**assumes**  $\text{take } \text{DIM}('a) (xs @ \text{replicate } (\text{DIM}('a) - \text{length } xs) \ 0) =$   
 $\text{take } \text{DIM}('a) (ys @ \text{replicate } (\text{DIM}('a) - \text{length } ys) \ 0)$   
**shows**  $\text{eucl-of-list } xs = (\text{eucl-of-list } ys::'a::\text{executable-euclidean-space})$   
 $\langle \text{proof} \rangle$

**lemma** *eucl-of-list-replicate-zero*[simp]: *eucl-of-list (replicate E 0) = 0*  
 ⟨proof⟩

**lemma** *eucl-of-list-Nil*[simp]: *eucl-of-list [] = 0*  
 ⟨proof⟩

**lemma** *fst-eucl-of-list-prod*:  
**shows** *fst (eucl-of-list xs::'b::executable-euclidean-space × -) = (eucl-of-list (take DIM('b) xs)::'b)*  
 ⟨proof⟩

**lemma** *index-zip-replicate1*[simp]: *index (zip (replicate d a) bs) (a, b) = index bs b*  
**if** *d = length bs*  
 ⟨proof⟩

**lemma** *index-zip-replicate2*[simp]: *index (zip as (replicate d b)) (a, b) = index as a*  
**if** *d = length as*  
 ⟨proof⟩

**lemma** *index-Basis-list-prod*[simp]:  
**fixes** *a::'a::executable-euclidean-space* **and** *b::'b::executable-euclidean-space*  
**shows** *a ∈ Basis ⇒ index Basis-list (a, 0::'b) = index Basis-list a*  
*b ∈ Basis ⇒ index Basis-list (0::'a, b) = DIM('a) + index Basis-list b*  
 ⟨proof⟩

**lemma** *eucl-of-list-eq-takeI*:  
**assumes** *(eucl-of-list (take DIM('a::executable-euclidean-space) xs)::'a) = x*  
**shows** *eucl-of-list xs = x*  
 ⟨proof⟩

**lemma** *eucl-of-list-inner-le*:  
**fixes** *i::'a::executable-euclidean-space*  
**assumes** *i ∈ Basis*  
**assumes** *l: length xs ≥ DIM('a)*  
**shows** *eucl-of-list xs · i = xs ! (index Basis-list i)*  
 ⟨proof⟩

**lemma** *eucl-of-list-prod-if*:  
**assumes** *length xs = DIM('a::executable-euclidean-space) + DIM('b::executable-euclidean-space)*  
**shows** *eucl-of-list xs = (eucl-of-list (take DIM('a) xs)::'a, eucl-of-list (drop DIM('a) xs)::'b)*  
 ⟨proof⟩

**lemma** *snd-eucl-of-list-prod*:

**shows**  $snd (eucl\text{-of-list } xs :: 'b :: executable\text{-euclidean-space} \times 'c :: executable\text{-euclidean-space})$   
 $=$   
 $(eucl\text{-of-list } (drop\ DIM('b) xs) :: 'c)$   
 $\langle proof \rangle$

**lemma** *eucl-of-list-prod*:

**shows**  $eucl\text{-of-list } xs = (eucl\text{-of-list } (take\ DIM('b) xs) :: 'b :: executable\text{-euclidean-space},$   
 $eucl\text{-of-list } (drop\ DIM('b) xs) :: 'c :: executable\text{-euclidean-space})$   
 $\langle proof \rangle$

**lemma** *eucl-of-list-real[simp]*:  $eucl\text{-of-list } [x] = (x :: real)$   
 $\langle proof \rangle$

**lemma** *eucl-of-list-append[simp]*:

**assumes**  $length\ xs = DIM('i :: executable\text{-euclidean-space})$   
**assumes**  $length\ ys = DIM('j :: executable\text{-euclidean-space})$   
**shows**  $eucl\text{-of-list } (xs @ ys) = (eucl\text{-of-list } xs :: 'i, eucl\text{-of-list } ys :: 'j)$   
 $\langle proof \rangle$

**lemma** *list-allI*:  $(\bigwedge x. x \in set\ xs \implies P\ x) \implies list\text{-all } P\ xs$   
 $\langle proof \rangle$

**lemma**

*concat-map-nthI*:

**assumes**  $\bigwedge x\ y. x \in set\ xs \implies y \in set\ (f\ x) \implies P\ y$   
**assumes**  $j < length\ (concat\ (map\ f\ xs))$   
**shows**  $P\ (concat\ (map\ f\ xs) ! j)$   
 $\langle proof \rangle$

**lemma** *map-nth-append1*:

**assumes**  $length\ xs = d$   
**shows**  $map\ (!) (xs @ ys) [0..<d] = xs$   
 $\langle proof \rangle$

**lemma** *map-nth-append2*:

**assumes**  $length\ ys = d$   
**shows**  $map\ (!) (xs @ ys) [length\ xs..<length\ xs + d] = ys$   
 $\langle proof \rangle$

**lemma** *length-map2 [simp]*:  $length\ (map2\ f\ xs\ ys) = min\ (length\ xs)\ (length\ ys)$   
 $\langle proof \rangle$

**lemma** *map2-nth [simp]*:  $map2\ f\ xs\ ys\ !\ n = f\ (xs\ !\ n)\ (ys\ !\ n)$   
**if**  $n < length\ xs\ n < length\ ys$   
 $\langle proof \rangle$

**lemma** *list-of-eucl-add*:  $list\text{-of-eucl } (x + y) = map2\ (+)\ (list\text{-of-eucl } x)\ (list\text{-of-eucl } y)$   
 $\langle proof \rangle$

**lemma** *list-of-eucl-inj*:

*list-of-eucl z = list-of-eucl y  $\implies$  y = z*  
*<proof>*

**lemma** *length-Basis-list-pos[simp]*: *length Basis-list > 0*

*<proof>*

**lemma** *Basis-list-nth-nonzero*:

*i < length (Basis-list::'a::executable-euclidean-space list)  $\implies$  (Basis-list::'a list)*  
*! i  $\neq$  0*  
*<proof>*

**lemma** *nth-Basis-list-prod*:

*i < DIM('a) + DIM('b)  $\implies$  (Basis-list::('a::executable-euclidean-space  $\times$  'b::executable-euclidean-space)*  
*list) ! i =*  
*(if i < DIM('a) then (Basis-list ! i, 0) else (0, Basis-list ! (i - DIM('a))))*  
*<proof>*

**lemma** *eucl-of-list-if*:

**assumes** [simp]: *length xs = DIM('a::executable-euclidean-space) distinct xs*  
**shows** *eucl-of-list (map ( $\lambda xa. \text{if } xa = x \text{ then } 1 \text{ else } 0$ ) (xs::nat list)) =*  
*(if x  $\in$  set xs then Basis-list ! index xs x else 0::'a)*  
*<proof>*

**lemma** *take-append-take-minus-idem*: *take n XS @ map (!) XS [n..<length XS]*  
*= XS*

*<proof>*

**lemma** *sum-list-Basis-list[simp]*: *sum-list (map f Basis-list) = ( $\sum b \in \text{Basis}. f b$ )*

*<proof>*

**lemma** *hd-Basis-list[simp]*: *hd Basis-list  $\in$  Basis*

*<proof>*

**definition** *inner-lv-rel a b = sum-list (map2 (\*) a b)*

**lemma** *eucl-of-list-inner-eq*: *(eucl-of-list xs::'a)  $\cdot$  eucl-of-list ys = inner-lv-rel xs ys*

**if** *length xs = DIM('a::executable-euclidean-space) length ys = DIM('a)*  
*<proof>*

**lemma** *euclidean-vec-componentwise*:

*( $\sum (xa::'a::euclidean-space \hat{\ } b::finite) \in \text{Basis}. f xa) = (\sum a \in \text{Basis}. (\sum b::'b \in \text{UNIV}. f (axis b a)))$*   
*<proof>*

**lemma** *vec-nth-inner-scaleR-craziness*:

$f (x \$ i \cdot j) *_R j = (\sum_{xa \in UNIV}. f (x \$ xa \cdot j) *_R axis xa j) \$ i$   
 <proof>

**instantiation** *vec* :: (*executable-euclidean-space*, *enum*) *executable-euclidean-space*  
**begin**

**definition** *Basis-list-vec* :: ('a, 'b) *vec list* **where**  
*Basis-list-vec* = *concat* (*map* ( $\lambda n. \text{map } (axis\ n) \text{ Basis-list}$ ) *enum-class.enum*)

**definition** *eucl-down-vec* :: *int*  $\Rightarrow$  ('a, 'b) *vec*  $\Rightarrow$  ('a, 'b) *vec* **where**  
*eucl-down-vec* *p* *x* = ( $\chi\ i. \text{eucl-down } p (x \$ i)$ )

**definition** *eucl-truncate-down-vec* :: *nat*  $\Rightarrow$  ('a, 'b) *vec*  $\Rightarrow$  ('a, 'b) *vec* **where**  
*eucl-truncate-down-vec* *p* *x* = ( $\chi\ i. \text{eucl-truncate-down } p (x \$ i)$ )

**definition** *eucl-truncate-up-vec* :: *nat*  $\Rightarrow$  ('a, 'b) *vec*  $\Rightarrow$  ('a, 'b) *vec* **where**  
*eucl-truncate-up-vec* *p* *x* = ( $\chi\ i. \text{eucl-truncate-up } p (x \$ i)$ )

**instance**  
 <proof>  
**end**

**lemma** *concat-same-lengths-nth*:  
**assumes**  $\bigwedge xs. xs \in \text{set } XS \implies \text{length } xs = N$   
**assumes**  $i < \text{length } XS * N \wedge N > 0$   
**shows**  $\text{concat } XS ! i = XS ! (i \text{ div } N) ! (i \text{ mod } N)$   
 <proof>

**lemma** *concat-map-map-index*:  
**shows**  $\text{concat } (\text{map } (\lambda n. \text{map } (f\ n) xs) ys) =$   
 $\text{map } (\lambda i. f (ys ! (i \text{ div } \text{length } xs)) (xs ! (i \text{ mod } \text{length } xs))) [0..<\text{length } xs * \text{length } ys]$   
 <proof>

**lemma**  
*sum-list-zip-map*:  
**assumes** *distinct* *xs*  
**shows**  $(\sum (x, y) \leftarrow \text{zip } xs (\text{map } g\ xs). f\ x\ y) = (\sum_{x \in \text{set } xs}. f\ x\ (g\ x))$   
 <proof>

**lemma**  
*sum-list-zip-map-of*:  
**assumes** *distinct* *bs*  
**assumes**  $\text{length } xs = \text{length } bs$   
**shows**  $(\sum (x, y) \leftarrow \text{zip } xs\ bs. f\ x\ y) = (\sum_{x \in \text{set } bs}. f\ (\text{the } (\text{map-of } (\text{zip } bs\ xs) x)))$   
 <proof>

**lemma** *vec-nth-matrix*:

$vec\text{-}nth (vec\text{-}nth (matrix\ y)\ i)\ j = vec\text{-}nth (y\ (axis\ j\ 1))\ i$   
*<proof>*

**lemma** *matrix-eqI*:

**assumes**  $\bigwedge x. x \in Basis \implies A * v\ x = B * v\ x$   
**shows**  $(A::real^{n \times n}) = B$   
*<proof>*

**lemma** *matrix-columnI*:

**assumes**  $\bigwedge i. column\ i\ A = column\ i\ B$   
**shows**  $(A::real^{n \times n}) = B$   
*<proof>*

**lemma**

*vec-nth-Basis*:

**fixes**  $x::real^n$

**shows**  $x \in Basis \implies vec\text{-}nth\ x\ i = (if\ x = axis\ i\ 1\ then\ 1\ else\ 0)$   
*<proof>*

**lemma** *vec-nth-eucl-of-list-eq*:  $length\ M = CARD(n) \implies$

$vec\text{-}nth\ (eucl\text{-}of\text{-}list\ M::real^n::enum)\ i = M\ !\ index\ Basis\text{-}list\ (axis\ i\ (1::real))$   
*<proof>*

**lemma** *index-Basis-list-axis1*:  $index\ Basis\text{-}list\ (axis\ i\ (1::real)) = index\ enum\text{-}class.\ enum\ i$

*<proof>*

**lemma** *vec-nth-eq-list-of-eucl1*:

$(vec\text{-}nth\ (M::real^n::enum)\ i) = list\text{-}of\text{-}eucl\ M\ !\ (index\ enum\text{-}class.\ enum\ i)$   
*<proof>*

**lemma** *enum-3[simp]*:  $(enum\text{-}class.\ enum::3\ list) = [0, 1, 2]$

*<proof>*

**lemma** *three-eq-zero*:  $(3::3) = 0$  *<proof>*

**lemma** *forall-3'*:  $(\forall i::3. P\ i) \iff P\ 0 \wedge P\ 1 \wedge P\ 2$

*<proof>*

**lemma** *euclidean-eq-list-of-euclI*:  $x = y$  **if**  $list\text{-}of\text{-}eucl\ x = list\text{-}of\text{-}eucl\ y$

*<proof>*

**lemma** *axis-one-neq-zero[simp]*:  $axis\ xa\ (1::'a::zero\text{-}neq\text{-}one) \neq 0$

*<proof>*

**lemma** *eucl-of-list-vec-nth3*[simp]:  
 (eucl-of-list [g, h, i]::real^3) \$ 0 = g  
 (eucl-of-list [g, h, i]::real^3) \$ 1 = h  
 (eucl-of-list [g, h, i]::real^3) \$ 2 = i  
 (eucl-of-list [g, h, i]::real^3) \$ 3 = g  
 <proof>

**type-synonym** *R3* = real\*real\*real

**lemma** *Basis-list-R3*: *Basis-list* = [(1,0,0), (0, 1, 0), (0, 0, 1)::R3]  
 <proof>

**lemma** *Basis-list-vec3*: *Basis-list* = [axis 0 1::real^3, axis 1 1, axis 2 1]  
 <proof>

**lemma** *eucl-of-list3*[simp]: *eucl-of-list* [a, b, c] = (a, b, c)  
 <proof>

## 1.4 Bounded Linear Functions

### 1.5 bounded linear functions

**locale** *blinfun-syntax*

**begin**

**no-notation** *vec-nth* (infixl <\$> 90)

**notation** *blinfun-apply* (infixl <\$> 999)

**end**

**lemma** *bounded-linear-via-derivative*:

**fixes** *f*::'a::real-normed-vector  $\Rightarrow$  'b::euclidean-space  $\Rightarrow_L$  'c::real-normed-vector

— TODO: generalize?

**assumes**  $\bigwedge i. ((\lambda x. \text{blinfun-apply } (f \ x) \ i) \text{ has-derivative } (\lambda x. f' \ y \ x \ i)) \text{ (at } y)$

**shows** *bounded-linear* (f' y x)

<proof>

**definition** *blinfun-scaleR*::('a::real-normed-vector  $\Rightarrow_L$  real)  $\Rightarrow$  'b::real-normed-vector  
 $\Rightarrow$  ('a  $\Rightarrow_L$  'b)

**where** *blinfun-scaleR* a b = *blinfun-scaleR-left* b o<sub>L</sub> a

**lemma** *blinfun-scaleR-transfer*[transfer-rule]:

*rel-fun* (*pcr-blinfun* (=) (=)) (*rel-fun* (=) (*pcr-blinfun* (=) (=)))

( $\lambda a \ b \ c. a \ c \ *_R \ b$ ) *blinfun-scaleR*

<proof>

**lemma** *blinfun-scaleR-rep-eq*[simp]:

*blinfun-scaleR* a b c = a c \*\_R b

<proof>

**lemma** *bounded-linear-blinfun-scaleR*: *bounded-linear* (*blinfun-scaleR* a)

<proof>

**lemma** *blinfun-scaleR-has-derivative*[*derivative-intros*]:  
**assumes** (*f has-derivative f'*) (*at x within s*)  
**shows**  $((\lambda x. \text{blinfun-scaleR } a (f x)) \text{ has-derivative } (\lambda x. \text{blinfun-scaleR } a (f' x)))$   
(*at x within s*)  
 $\langle \text{proof} \rangle$

**lemma** *blinfun-componentwise*:  
**fixes**  $f::'a::\text{real-normed-vector} \Rightarrow 'b::\text{euclidean-space} \Rightarrow_L 'c::\text{real-normed-vector}$   
**shows**  $f = (\lambda x. \sum_{i \in \text{Basis}} \text{blinfun-scaleR } (\text{blinfun-inner-left } i) (f x i))$   
 $\langle \text{proof} \rangle$

**lemma**  
*blinfun-has-derivative-componentwiseI*:  
**fixes**  $f::'a::\text{real-normed-vector} \Rightarrow 'b::\text{euclidean-space} \Rightarrow_L 'c::\text{real-normed-vector}$   
**assumes**  $\bigwedge i. i \in \text{Basis} \Rightarrow ((\lambda x. f x i) \text{ has-derivative } \text{blinfun-apply } (f' i))$  (*at x*)  
**shows** (*f has-derivative*  $(\lambda x. \sum_{i \in \text{Basis}} \text{blinfun-scaleR } (\text{blinfun-inner-left } i) (f' i x))$ ) (*at x*)  
 $\langle \text{proof} \rangle$

**lemma**  
*has-derivative-BlinfunI*:  
**fixes**  $f::'a::\text{real-normed-vector} \Rightarrow 'b::\text{euclidean-space} \Rightarrow_L 'c::\text{real-normed-vector}$   
**assumes**  $\bigwedge i. ((\lambda x. f x i) \text{ has-derivative } (\lambda x. f' y x i))$  (*at y*)  
**shows** (*f has-derivative*  $(\lambda x. \text{Blinfun } (f' y x))$ ) (*at y*)  
 $\langle \text{proof} \rangle$

**lemma**  
*has-derivative-Blinfun*:  
**assumes** (*f has-derivative f'*) *F*  
**shows** (*f has-derivative Blinfun f'*) *F*  
 $\langle \text{proof} \rangle$

**lift-definition** *flip-blinfun*::  
 $( 'a::\text{real-normed-vector} \Rightarrow_L 'b::\text{real-normed-vector} \Rightarrow_L 'c::\text{real-normed-vector} ) \Rightarrow$   
 $'b \Rightarrow_L 'a \Rightarrow_L 'c$  **is**  
 $\lambda f x y. f y x$   
 $\langle \text{proof} \rangle$

**lemma** *flip-blinfun-apply*[*simp*]: *flip-blinfun f a b = f b a*  
 $\langle \text{proof} \rangle$

**lemma** *le-norm-blinfun*:  
**shows**  $\text{norm } (\text{blinfun-apply } f x) / \text{norm } x \leq \text{norm } f$   
 $\langle \text{proof} \rangle$

**lemma** *norm-flip-blinfun*[*simp*]:  $\text{norm } (\text{flip-blinfun } x) = \text{norm } x$  (**is** *?l = ?r*)  
 $\langle \text{proof} \rangle$

**lemma** *bounded-linear-flip-blinfun*[*bounded-linear*]: *bounded-linear flip-blinfun*  
 ⟨*proof*⟩

**lemma** *dist-swap2-swap2*[*simp*]: *dist (flip-blinfun f) (flip-blinfun g) = dist f g*  
 ⟨*proof*⟩

**context includes** *blinfun.lifting* **begin**

**lift-definition** *blinfun-of-vmatrix*::(*real*^'*m*^'*n*) ⇒ ((*real*^(''*m*::*finite*)) ⇒<sub>L</sub> (*real*^(''*n*::*finite*)))  
**is**

*matrix-vector-mult*:: ((*real*, '*m*) *vec*, '*n*) *vec* ⇒ ((*real*, '*m*) *vec* ⇒ (*real*, '*n*) *vec*)  
 ⟨*proof*⟩

**lemma** *matrix-blinfun-of-vmatrix*[*simp*]: *matrix (blinfun-of-vmatrix M) = M*  
 ⟨*proof*⟩

**end**

**lemma** *blinfun-apply-componentwise*:

$B = (\sum_{i \in \text{Basis}} \text{blinfun-scaleR } (\text{blinfun-inner-left } i) (\text{blinfun-apply } B \ i))$   
 ⟨*proof*⟩

**lemma** *blinfun-apply-eq-sum*:

**assumes** [*simp*]: *length v = CARD('n)*  
**shows** *blinfun-apply (B::(real^'*n*::enum)⇒<sub>L</sub>(real^'*m*::enum)) (eucl-of-list v) =*  
 $(\sum_{i < \text{CARD}('m)} \sum_{j < \text{CARD}('n)} ((B (\text{Basis-list } ! j) \cdot \text{Basis-list } ! i) * v ! j)$   
 $*_R (\text{Basis-list } ! i))$   
 ⟨*proof*⟩

**lemma** *in-square-lemma*[*intro, simp*]:  $x * C + y < D * C$  **if**  $x < D$   $y < C$  **for**  
 $x::\text{nat}$   
 ⟨*proof*⟩

**lemma** *less-square-imp-div-less*[*intro, simp*]:  $i < E * D \implies i \text{ div } E < D$  **for**  $i::\text{nat}$   
 ⟨*proof*⟩

**lemma** *in-square-lemma'*[*intro, simp*]:  $i < L \implies n < N \implies i * N + n < N * L$   
**for**  $i n::\text{nat}$   
 ⟨*proof*⟩

**lemma**

*distinct-nth-eq-iff*:

$\text{distinct } xs \implies x < \text{length } xs \implies y < \text{length } xs \implies xs ! x = xs ! y \iff x = y$   
 ⟨*proof*⟩

**lemma** *index-Basis-list-axis2*:

$\text{index Basis-list } (\text{axis } (j::'j::\text{enum}) (\text{axis } (i::'i::\text{enum}) (1::\text{real}))) =$   
 $(\text{index enum-class.enum } j) * \text{CARD}('i) + \text{index enum-class.enum } i$

*<proof>*

**lemma**

*vec-nth-Basis2:*

**fixes**  $x::\text{real}^n{}^m$

**shows**  $x \in \text{Basis} \implies \text{vec-nth} (\text{vec-nth } x \ i) \ j = ((\text{if } x = \text{axis } i \ (\text{axis } j \ 1) \ \text{then } 1 \ \text{else } 0))$

*<proof>*

**lemma** *vec-nth-eucl-of-list-eq2:*  $\text{length } M = \text{CARD}'n * \text{CARD}'m \implies$

$\text{vec-nth} (\text{vec-nth} (\text{eucl-of-list } M::\text{real}'n::\text{enum}'m::\text{enum}) \ i) \ j = M \ ! \ \text{index } \text{Basis-list} \ (\text{axis } i \ (\text{axis } j \ (1::\text{real})))$

*<proof>*

**lemma** *vec-nth-eq-list-of-eucl2:*

$\text{vec-nth} (\text{vec-nth} (M::\text{real}'n::\text{enum}'m::\text{enum}) \ i) \ j =$

$\text{list-of-eucl } M \ ! \ (\text{index } \text{enum-class.enum } i * \text{CARD}'n + \text{index } \text{enum-class.enum}$

$j)$

*<proof>*

**theorem**

*eucl-of-list-matrix-vector-mult-eq-sum-nth-Basis-list:*

**assumes**  $\text{length } M = \text{CARD}'n * \text{CARD}'m$

**assumes**  $\text{length } v = \text{CARD}'n$

**shows**  $(\text{eucl-of-list } M::\text{real}'n::\text{enum}'m::\text{enum}) * v \ \text{eucl-of-list } v =$

$(\sum_{i < \text{CARD}'m}.$

$(\sum_{j < \text{CARD}'n}. M \ ! \ (i * \text{CARD}'n + j) * v \ ! \ j) *_{\mathbb{R}} \text{Basis-list} \ ! \ i)$

*<proof>*

**lemma** *index-enum-less[intro, simp]:*  $\text{index } \text{enum-class.enum} \ (i::'n::\text{enum}) < \text{CARD}'n$

*<proof>*

**lemmas**  $[\text{intro}, \text{simp}] = \text{enum-distinct}$

**lemmas**  $[\text{simp}] = \text{card-UNIV-length-enum}[\text{symmetric}] \ \text{enum-UNIV}$

**lemma** *sum-index-enum-eq:*

$(\sum_{(k::'n::\text{enum}) \in \text{UNIV}. f \ (\text{index } \text{enum-class.enum } k)}) = (\sum_{i < \text{CARD}'n}. f \ i)$

*<proof>*

**end**

## 2 Affine Form

**theory** *Affine-Form*

**imports**

*HOL-Analysis.Multivariate-Analysis*

*HOL-Combinatorics.List-Permutation*

*Affine-Arithmetic-Auxiliarities*

*Executable-Euclidean-Space*

**begin**

## 2.1 Auxiliary developments

**lemma** *sum-list-mono*:

**fixes**  $xs\ ys::'a::ordered-ab-group-add\ list$

**shows**

$length\ xs = length\ ys \implies (\bigwedge x\ y. (x, y) \in set\ (zip\ xs\ ys) \implies x \leq y) \implies$   
 $sum-list\ xs \leq sum-list\ ys$

*<proof>*

**lemma**

**fixes**  $xs::'a::ordered-comm-monoid-add\ list$

**shows** *sum-list-nonneg*:  $(\bigwedge x. x \in set\ xs \implies x \geq 0) \implies sum-list\ xs \geq 0$

*<proof>*

**lemma** *map-filter*:

$map\ f\ (filter\ (\lambda x. P\ (f\ x))\ xs) = filter\ P\ (map\ f\ xs)$

*<proof>*

**lemma**

*map-of-zip-upto2-length-eq-nth*:

**assumes** *distinct B*

**assumes**  $i < length\ B$

**shows**  $(map-of\ (zip\ B\ [0..<length\ B])\ (B\ !\ i)) = Some\ i$

*<proof>*

**lemma** *distinct-map-fst-snd-eqD*:

$distinct\ (map\ fst\ xs) \implies (i, a) \in set\ xs \implies (i, b) \in set\ xs \implies a = b$

*<proof>*

**lemma** *length-filter-snd-zip*:

$length\ ys = length\ xs \implies length\ (filter\ (p \circ snd)\ (zip\ ys\ xs)) = length\ (filter\ p\ xs)$

*<proof>*

**lemma** *filter-snd-nth*:

$length\ ys = length\ xs \implies n < length\ (filter\ p\ xs) \implies$

$snd\ (filter\ (p \circ snd)\ (zip\ ys\ xs))\ !\ n = filter\ p\ xs\ !\ n$

*<proof>*

**lemma** *distinct-map-snd-fst-eqD*:

$distinct\ (map\ snd\ xs) \implies (i, a) \in set\ xs \implies (j, a) \in set\ xs \implies i = j$

*<proof>*

**lemma** *map-of-mapk-inj-on-SomeI*:

$inj-on\ f\ (fst\ ` (set\ t)) \implies map-of\ t\ k = Some\ x \implies$

$map-of\ (map\ (case-prod\ (\lambda k. Pair\ (f\ k)))\ t)\ (f\ k) = Some\ x$

*<proof>*

**lemma** *map-abs-nonneg*[simp]:

**fixes**  $xs::'a::\text{ordered-ab-group-add-abs}$  *list*

**shows**  $\text{list-all } (\lambda x. x \geq 0) \text{ } xs \implies \text{map abs } xs = xs$

*<proof>*

**lemma** *the-inv-into-image-eq*:  $\text{inj-on } f \ A \implies Y \subseteq f \ ' \ A \implies \text{the-inv-into } A \ f \ ' \ Y = f \ - \ ' \ Y \cap A$

*<proof>*

**lemma** *image-fst-zip*:  $\text{length } ys = \text{length } xs \implies \text{fst} \ ' \ \text{set } (\text{zip } ys \ xs) = \text{set } ys$

*<proof>*

**lemma** *inj-on-fst-set-zip-distinct*[simp]:

$\text{distinct } xs \implies \text{length } xs = \text{length } ys \implies \text{inj-on } \text{fst} \ (\text{set } (\text{zip } xs \ ys))$

*<proof>*

**lemma** *mem-greaterThanLessThan-absI*:

**fixes**  $x::\text{real}$

**assumes**  $\text{abs } x < 1$

**shows**  $x \in \{-1 < .. < 1\}$

*<proof>*

**lemma** *minus-one-less-divideI*:  $b > 0 \implies -b < a \implies -1 < a / (b::\text{real})$

*<proof>*

**lemma** *divide-less-oneI*:  $b > 0 \implies b > a \implies a / (b::\text{real}) < 1$

*<proof>*

**lemma** *closed-segment-real*:

**fixes**  $a \ b::\text{real}$

**shows**  $\text{closed-segment } a \ b = (\text{if } a \leq b \text{ then } \{a .. b\} \text{ else } \{b .. a\}) \ (\text{is } - = \text{?if})$

*<proof>*

## 2.2 Partial Deviations

**typedef** (**overloaded**)  $'a \ \text{pdevs} = \{x::\text{nat} \Rightarrow 'a::\text{zero. finite } \{i. x \ i \neq 0\}\}$

— TODO: unify with polynomials

**morphisms**  $\text{pdevs-apply } \text{Abs-pdev}$

*<proof>*

**setup-lifting**  $\text{type-definition-pdevs}$

**lemma** *pdevs-eqI*:  $(\bigwedge i. \text{pdevs-apply } x \ i = \text{pdevs-apply } y \ i) \implies x = y$

*<proof>*

**definition**  $\text{pdevs-val} :: (\text{nat} \Rightarrow \text{real}) \Rightarrow 'a::\text{real-normed-vector } \text{pdevs} \Rightarrow 'a$

**where**  $\text{pdevs-val } e \ x = (\sum i. e \ i \ *_{\mathbb{R}} \ \text{pdevs-apply } x \ i)$

**definition** *valuate*::  $((nat \Rightarrow real) \Rightarrow 'a) \Rightarrow 'a \text{ set}$   
**where** *valuate*  $x = x \text{ ' (UNIV } \rightarrow \{-1 .. 1\})$

**lemma** *valuate-ex*:  $x \in \text{valuate } f \longleftrightarrow (\exists e. (\forall i. e \ i \in \{-1 .. 1\}) \wedge x = f \ e)$   
 $\langle \text{proof} \rangle$

**instantiation** *pdevs* ::  $(\text{equal}) \ \text{equal}$   
**begin**

**definition** *equal-pdevs*::  $'a \ \text{pdevs} \Rightarrow 'a \ \text{pdevs} \Rightarrow \text{bool}$   
**where** *equal-pdevs*  $a \ b \longleftrightarrow a = b$

**instance**  $\langle \text{proof} \rangle$   
**end**

## 2.3 Affine Forms

The data structure of affine forms represents particular sets, zonotopes

**type-synonym**  $'a \ \text{aform} = 'a \times 'a \ \text{pdevs}$

## 2.4 Evaluation, Range, Joint Range

**definition** *aform-val* ::  $(nat \Rightarrow real) \Rightarrow 'a::\text{real-normed-vector} \ \text{aform} \Rightarrow 'a$   
**where** *aform-val*  $e \ X = \text{fst } X + \text{pdevs-val } e \ (\text{snd } X)$

**definition** *Affine* ::  
 $'a::\text{real-normed-vector} \ \text{aform} \Rightarrow 'a \ \text{set}$   
**where** *Affine*  $X = \text{valuate } (\lambda e. \ \text{aform-val } e \ X)$

**definition** *Joints* ::  
 $'a::\text{real-normed-vector} \ \text{aform list} \Rightarrow 'a \ \text{list set}$   
**where** *Joints*  $XS = \text{valuate } (\lambda e. \ \text{map } (\text{aform-val } e) \ XS)$

**lemma** *Joints-nthE*:  
**assumes**  $zs \in \text{Joints } ZS$   
**obtains**  $e$  **where**  
 $\bigwedge i. \ i < \text{length } zs \implies zs \ ! \ i = \text{aform-val } e \ (ZS \ ! \ i)$   
 $\bigwedge i. \ e \ i \in \{-1 .. 1\}$   
 $\langle \text{proof} \rangle$

**lemma** *Joints-mapE*:  
**assumes**  $ys \in \text{Joints } YS$   
**obtains**  $e$  **where**  
 $ys = \text{map } (\lambda x. \ \text{aform-val } e \ x) \ YS$   
 $\bigwedge i. \ e \ i \in \{-1 .. 1\}$   
 $\langle \text{proof} \rangle$

**lemma**  
*zipped-subset-mapped-Elem*:

**assumes**  $xs = \text{map } (\text{aform-val } e) \text{ } XS$   
**assumes**  $e: \bigwedge i. e \ i \in \{-1 .. 1\}$   
**assumes**  $[\text{simp}]: \text{length } xs = \text{length } XS$   
**assumes**  $[\text{simp}]: \text{length } ys = \text{length } YS$   
**assumes**  $\text{set } (\text{zip } ys \ YS) \subseteq \text{set } (\text{zip } xs \ XS)$   
**shows**  $ys = \text{map } (\text{aform-val } e) \ YS$   
 $\langle \text{proof} \rangle$

**lemma** *Joints-set-zip-subset*:  
**assumes**  $xs \in \text{Joints } XS$   
**assumes**  $\text{length } xs = \text{length } XS$   
**assumes**  $\text{length } ys = \text{length } YS$   
**assumes**  $\text{set } (\text{zip } ys \ YS) \subseteq \text{set } (\text{zip } xs \ XS)$   
**shows**  $ys \in \text{Joints } YS$   
 $\langle \text{proof} \rangle$

**lemma** *Joints-set-zip*:  
**assumes**  $ys \in \text{Joints } YS$   
**assumes**  $\text{length } xs = \text{length } XS$   
**assumes**  $\text{length } YS = \text{length } XS$   
**assumes**  $\text{sets-eq}: \text{set } (\text{zip } xs \ XS) = \text{set } (\text{zip } ys \ YS)$   
**shows**  $xs \in \text{Joints } XS$   
 $\langle \text{proof} \rangle$

**definition** *Joints2* ::  
 $'a::\text{real-normed-vector aform list} \Rightarrow 'b::\text{real-normed-vector aform} \Rightarrow ('a \ \text{list} \times 'b)$   
 $\text{set}$   
**where**  $\text{Joints2 } XS \ Y = \text{valuate } (\lambda e. (\text{map } (\text{aform-val } e) \ XS, \text{aform-val } e \ Y))$

**lemma** *Joints2E*:  
**assumes**  $zs-y \in \text{Joints2 } ZS \ Y$   
**obtains**  $e$  **where**  
 $\bigwedge i. i < \text{length } (\text{fst } zs-y) \implies (\text{fst } zs-y) \ ! \ i = \text{aform-val } e \ (ZS \ ! \ i)$   
 $\text{snd } (zs-y) = \text{aform-val } e \ Y$   
 $\bigwedge i. e \ i \in \{-1..1\}$   
 $\langle \text{proof} \rangle$

**lemma** *nth-in-AffineI*:  
**assumes**  $xs \in \text{Joints } XS$   
**assumes**  $i < \text{length } XS$   
**shows**  $xs \ ! \ i \in \text{Affine } (XS \ ! \ i)$   
 $\langle \text{proof} \rangle$

**lemma** *Cons-nth-in-Joints1*:  
**assumes**  $xs \in \text{Joints } XS$   
**assumes**  $i < \text{length } XS$   
**shows**  $((xs \ ! \ i) \ # \ xs) \in \text{Joints } ((XS \ ! \ i) \ # \ XS)$   
 $\langle \text{proof} \rangle$

**lemma** *Cons-nth-in-Joints2*:

**assumes**  $xs \in \text{Joints } XS$

**assumes**  $i < \text{length } XS$

**assumes**  $j < \text{length } XS$

**shows**  $((xs ! i) \# (xs ! j) \# xs) \in \text{Joints } ((XS ! i) \# (XS ! j) \# XS)$

*<proof>*

**lemma** *Joints-swap*:

$x \# y \# xs \in \text{Joints } (X \# Y \# XS) \longleftrightarrow y \# x \# xs \in \text{Joints } (Y \# X \# XS)$

*<proof>*

**lemma** *Joints-swap-Cons-append*:

$\text{length } xs = \text{length } XS \implies x \# ys @ xs \in \text{Joints } (X \# YS @ XS) \longleftrightarrow ys @ x \# xs \in \text{Joints } (YS @ X \# XS)$

*<proof>*

**lemma** *Joints-ConsD*:

$x \# xs \in \text{Joints } (X \# XS) \implies xs \in \text{Joints } XS$

*<proof>*

**lemma** *Joints-appendD1*:

$ys @ xs \in \text{Joints } (YS @ XS) \implies \text{length } xs = \text{length } XS \implies xs \in \text{Joints } XS$

*<proof>*

**lemma** *Joints-appendD2*:

$ys @ xs \in \text{Joints } (YS @ XS) \implies \text{length } ys = \text{length } YS \implies ys \in \text{Joints } YS$

*<proof>*

**lemma** *Joints-imp-length-eq*:  $xs \in \text{Joints } XS \implies \text{length } xs = \text{length } XS$

*<proof>*

**lemma** *Joints-rotate[simp]*:  $xs @ [x] \in \text{Joints } (XS @ [X]) \longleftrightarrow x \# xs \in \text{Joints } (X \# XS)$

*<proof>*

## 2.5 Domain

**definition** *pdevs-domain*  $x = \{i. \text{pdevs-apply } x \ i \neq 0\}$

**lemma** *finite-pdevs-domain[intro, simp]*: *finite* (*pdevs-domain*  $x$ )

*<proof>*

**lemma** *in-pdevs-domain[simp]*:  $i \in \text{pdevs-domain } x \longleftrightarrow \text{pdevs-apply } x \ i \neq 0$

*<proof>*

## 2.6 Least Fresh Index

**definition** *degree*:  $'a::\text{real-vector } pdevs \Rightarrow \text{nat}$

**where**  $\text{degree } x = (\text{LEAST } i. \forall j \geq i. \text{pdevs-apply } x \ j = 0)$

**lemma** *degree[rule-format, intro, simp]*:

**shows**  $\forall j \geq \text{degree } x. \text{pdevs-apply } x \ j = 0$   
 ⟨proof⟩

**lemma** *degree-le*:  
**assumes**  $d: \forall j \geq d. \text{pdevs-apply } x \ j = 0$   
**shows**  $\text{degree } x \leq d$   
 ⟨proof⟩

**lemma** *degree-gt*:  $\text{pdevs-apply } x \ j \neq 0 \implies \text{degree } x > j$   
 ⟨proof⟩

**lemma** *pdevs-val-pdevs-domain*:  $\text{pdevs-val } e \ X = (\sum_{i \in \text{pdevs-domain } X} e \ i \ *_{\mathbb{R}} \text{pdevs-apply } X \ i)$   
 ⟨proof⟩

**lemma** *pdevs-val-sum-le*:  $\text{degree } X \leq d \implies \text{pdevs-val } e \ X = (\sum_{i < d} e \ i \ *_{\mathbb{R}} \text{pdevs-apply } X \ i)$   
 ⟨proof⟩

**lemmas**  $\text{pdevs-val-sum} = \text{pdevs-val-sum-le}[\text{OF order-refl}]$

**lemma** *pdevs-val-zero[simp]*:  $\text{pdevs-val } (\lambda \cdot. 0) \ x = 0$   
 ⟨proof⟩

**lemma** *degree-eqI*:  
**assumes**  $\text{pdevs-apply } x \ d \neq 0$   
**assumes**  $\bigwedge j. j > d \implies \text{pdevs-apply } x \ j = 0$   
**shows**  $\text{degree } x = \text{Suc } d$   
 ⟨proof⟩

**lemma** *finite-degree-nonzero[intro, simp]*:  $\text{finite } \{i. \text{pdevs-apply } x \ i \neq 0\}$   
 ⟨proof⟩

**lemma** *degree-eq-Suc-max*:  
 $\text{degree } x = (\text{if } (\forall i. \text{pdevs-apply } x \ i = 0) \ \text{then } 0 \ \text{else } \text{Suc } (\text{Max } \{i. \text{pdevs-apply } x \ i \neq 0\}))$   
 ⟨proof⟩

**lemma** *pdevs-val-degree-cong*:  
**assumes**  $b = d$   
**assumes**  $\bigwedge i. i < \text{degree } b \implies a \ i = c \ i$   
**shows**  $\text{pdevs-val } a \ b = \text{pdevs-val } c \ d$   
 ⟨proof⟩

**abbreviation** *degree-aform*:: $'a::\text{real-vector aform} \Rightarrow \text{nat}$   
**where**  $\text{degree-aform } X \equiv \text{degree } (\text{snd } X)$

**lemma** *degree-cong*:  $(\bigwedge i. (\text{pdevs-apply } x \ i = 0) = (\text{pdevs-apply } y \ i = 0)) \implies \text{degree } x = \text{degree } y$

*<proof>*

**lemma** *Least-True-nat*[*intro, simp*]: (*LEAST i::nat. True*) = 0  
*<proof>*

**lemma** *sorted-list-of-pdevs-domain-eq*:  
*sorted-list-of-set (pdevs-domain X) = filter (( $\neq$ ) 0 o pdevs-apply X) [0..*degree X*]*  
*<proof>*

## 2.7 Total Deviation

**definition** *tdev::'a::ordered-euclidean-space pdevs  $\Rightarrow$  'a where*  
*tdev x = ( $\sum$  *i*<*degree x*. |pdevs-apply x i|)*

**lemma** *abs-pdevs-val-le-tdev*:  $e \in UNIV \rightarrow \{-1 .. 1\} \Longrightarrow |pdevs-val e x| \leq tdev x$   
*<proof>*

## 2.8 Binary Pointwise Operations

**definition** *binop-pdevs-raw::('a::zero  $\Rightarrow$  'b::zero  $\Rightarrow$  'c::zero)  $\Rightarrow$*   
*(nat  $\Rightarrow$  'a)  $\Rightarrow$  (nat  $\Rightarrow$  'b)  $\Rightarrow$  nat  $\Rightarrow$  'c*  
**where** *binop-pdevs-raw f x y i = (if x i = 0  $\wedge$  y i = 0 then 0 else f (x i) (y i))*

**lemma** *nonzeros-binop-pdevs-subset*:  
*{i. binop-pdevs-raw f x y i  $\neq$  0}  $\subseteq$  {i. x i  $\neq$  0}  $\cup$  {i. y i  $\neq$  0}*  
*<proof>*

**lift-definition** *binop-pdevs::*  
*('a  $\Rightarrow$  'b  $\Rightarrow$  'c)  $\Rightarrow$  'a::zero pdevs  $\Rightarrow$  'b::zero pdevs  $\Rightarrow$  'c::zero pdevs*  
**is** *binop-pdevs-raw*  
*<proof>*

**lemma** *pdevs-apply-binop-pdevs*[*simp*]: *pdevs-apply (binop-pdevs f x y) i =*  
*(if pdevs-apply x i = 0  $\wedge$  pdevs-apply y i = 0 then 0 else f (pdevs-apply x i)*  
*(pdevs-apply y i))*  
*<proof>*

## 2.9 Addition

**definition** *add-pdevs::'a::real-vector pdevs  $\Rightarrow$  'a pdevs  $\Rightarrow$  'a pdevs*  
**where** *add-pdevs = binop-pdevs (+)*

**lemma** *pdevs-apply-add-pdevs*[*simp*]:  
*pdevs-apply (add-pdevs X Y) n = pdevs-apply X n + pdevs-apply Y n*  
*<proof>*

**lemma** *pdevs-val-add-pdevs*[*simp*]:  
**fixes** *x y::'a::euclidean-space*  
**shows** *pdevs-val e (add-pdevs X Y) = pdevs-val e X + pdevs-val e Y*

*<proof>*

## 2.10 Total Deviation

**lemma** *tdev-eq-zero-iff*: **fixes**  $X::\text{real}$  *pdevs* **shows**  $tdev\ X = 0 \iff (\forall e. pdevs\text{-val}\ e\ X = 0)$   
*<proof>*

**lemma** *tdev-nonneg*[*intro, simp*]:  $tdev\ X \geq 0$   
*<proof>*

**lemma** *tdev-nonpos-iff*[*simp*]:  $tdev\ X \leq 0 \iff tdev\ X = 0$   
*<proof>*

## 2.11 Unary Operations

**definition** *unop-pdevs-raw*::  
 $(a::\text{zero} \Rightarrow b::\text{zero}) \Rightarrow (\text{nat} \Rightarrow 'a) \Rightarrow \text{nat} \Rightarrow 'b$   
**where** *unop-pdevs-raw*  $f\ x\ i = (\text{if } x\ i = 0 \text{ then } 0 \text{ else } f\ (x\ i))$

**lemma** *nonzeros-unop-pdevs-subset*:  $\{i. \text{unop-pdevs-raw } f\ x\ i \neq 0\} \subseteq \{i. x\ i \neq 0\}$   
*<proof>*

**lift-definition** *unop-pdevs*::  
 $(a \Rightarrow b) \Rightarrow a::\text{zero } pdevs \Rightarrow b::\text{zero } pdevs$   
**is** *unop-pdevs-raw*  
*<proof>*

**lemma** *pdevs-apply-unop-pdevs*[*simp*]:  $pdevs\text{-apply } (\text{unop-pdevs } f\ x)\ i =$   
 $(\text{if } pdevs\text{-apply } x\ i = 0 \text{ then } 0 \text{ else } f\ (pdevs\text{-apply } x\ i))$   
*<proof>*

**lemma** *pdevs-domain-unop-linear*:  
**assumes** *linear*  $f$   
**shows**  $pdevs\text{-domain } (\text{unop-pdevs } f\ x) \subseteq pdevs\text{-domain } x$   
*<proof>*

**lemma**  
*pdevs-val-unop-linear*:  
**assumes** *linear*  $f$   
**shows**  $pdevs\text{-val } e\ (\text{unop-pdevs } f\ x) = f\ (pdevs\text{-val } e\ x)$   
*<proof>*

## 2.12 Pointwise Scaling of Partial Deviations

**definition** *scaleR-pdevs*:: $\text{real} \Rightarrow 'a::\text{real-vector } pdevs \Rightarrow 'a\ pdevs$   
**where** *scaleR-pdevs*  $r\ x = \text{unop-pdevs } ((*_R)\ r)\ x$

**lemma** *pdevs-apply-scaleR-pdevs*[*simp*]:  
 $pdevs\text{-apply } (\text{scaleR-pdevs } x\ Y)\ n = x *_R pdevs\text{-apply } Y\ n$

*<proof>*

**lemma** *degree-scaleR-pdevs[simp]*:  $\text{degree} (\text{scaleR-pdevs } r \ x) = (\text{if } r = 0 \text{ then } 0 \text{ else } \text{degree } x)$   
*<proof>*

**lemma** *pdevs-val-scaleR-pdevs[simp]*:  
**fixes**  $x::\text{real}$  **and**  $Y::'a::\text{real-normed-vector}$  *pdevs*  
**shows**  $\text{pdevs-val } e (\text{scaleR-pdevs } x \ Y) = x *_R \text{pdevs-val } e \ Y$   
*<proof>*

## 2.13 Partial Deviations Scale Pointwise

**definition** *pdevs-scaleR::real pdevs  $\Rightarrow$  'a::real-vector  $\Rightarrow$  'a pdevs*  
**where**  $\text{pdevs-scaleR } r \ x = \text{unop-pdevs } (\lambda r. r *_R x) \ r$

**lemma** *pdevs-apply-pdevs-scaleR[simp]*:  
 $\text{pdevs-apply } (\text{pdevs-scaleR } X \ y) \ n = \text{pdevs-apply } X \ n *_R y$   
*<proof>*

**lemma** *degree-pdevs-scaleR[simp]*:  $\text{degree} (\text{pdevs-scaleR } r \ x) = (\text{if } x = 0 \text{ then } 0 \text{ else } \text{degree } r)$   
*<proof>*

**lemma** *pdevs-val-pdevs-scaleR[simp]*:  
**fixes**  $X::\text{real pdevs}$  **and**  $y::'a::\text{real-normed-vector}$   
**shows**  $\text{pdevs-val } e (\text{pdevs-scaleR } X \ y) = \text{pdevs-val } e \ X *_R y$   
*<proof>*

## 2.14 Pointwise Unary Minus

**definition** *uminus-pdevs::'a::real-vector pdevs  $\Rightarrow$  'a pdevs*  
**where**  $\text{uminus-pdevs} = \text{unop-pdevs } \text{uminus}$

**lemma** *pdevs-apply-uminus-pdevs[simp]*:  $\text{pdevs-apply } (\text{uminus-pdevs } x) = - \text{pdevs-apply } x$   
*<proof>*

**lemma** *degree-uminus-pdevs[simp]*:  $\text{degree} (\text{uminus-pdevs } x) = \text{degree } x$   
*<proof>*

**lemma** *pdevs-val-uminus-pdevs[simp]*:  $\text{pdevs-val } e (\text{uminus-pdevs } x) = - \text{pdevs-val } e \ x$   
*<proof>*

**definition**  $\text{uminus-aform } X = (- \ \text{fst } X, \ \text{uminus-pdevs } (\text{snd } X))$

**lemma** *fst-uminus-aform[simp]*:  $\text{fst} (\text{uminus-aform } Y) = - \ \text{fst } Y$   
*<proof>*

**lemma** *aform-val-uminus-aform*[simp]: *aform-val*  $e$  (*uminus-aform*  $X$ ) = - *aform-val*  $e$   $X$   
 ⟨*proof*⟩

## 2.15 Constant

**lift-definition** *zero-pdevs*::'a::zero *pdevs* **is**  $\lambda\cdot. 0$  ⟨*proof*⟩

**lemma** *pdevs-apply-zero-pdevs*[simp]: *pdevs-apply* *zero-pdevs*  $i = 0$   
 ⟨*proof*⟩

**lemma** *pdevs-val-zero-pdevs*[simp]: *pdevs-val*  $e$  *zero-pdevs* = 0  
 ⟨*proof*⟩

**definition** *num-aform*  $f = (f, \text{zero-pdevs})$

## 2.16 Inner Product

**definition** *pdevs-inner*::'a::euclidean-space *pdevs*  $\Rightarrow$  'a  $\Rightarrow$  real *pdevs*  
**where** *pdevs-inner*  $x$   $b = \text{unop-pdevs } (\lambda x. x \cdot b)$   $x$

**lemma** *pdevs-apply-pdevs-inner*[simp]: *pdevs-apply* (*pdevs-inner*  $p$   $a$ )  $i = \text{pdevs-apply}$   
 $p$   $i \cdot a$   
 ⟨*proof*⟩

**lemma** *pdevs-val-pdevs-inner*[simp]: *pdevs-val*  $e$  (*pdevs-inner*  $p$   $a$ ) = *pdevs-val*  $e$   $p$   
 $\cdot a$   
 ⟨*proof*⟩

**definition** *inner-aform*::'a::euclidean-space *aform*  $\Rightarrow$  'a  $\Rightarrow$  real *aform*  
**where** *inner-aform*  $X$   $b = (\text{fst } X \cdot b, \text{pdevs-inner } (\text{snd } X) b)$

## 2.17 Inner Product Pair

**definition** *inner2*::'a::euclidean-space  $\Rightarrow$  'a  $\Rightarrow$  'a  $\Rightarrow$  real\*real  
**where** *inner2*  $x$   $n$   $l = (x \cdot n, x \cdot l)$

**definition** *pdevs-inner2*::'a::euclidean-space *pdevs*  $\Rightarrow$  'a  $\Rightarrow$  'a  $\Rightarrow$  (real\*real) *pdevs*  
**where** *pdevs-inner2*  $X$   $n$   $l = \text{unop-pdevs } (\lambda x. \text{inner2 } x n l)$   $X$

**lemma** *pdevs-apply-pdevs-inner2*[simp]: *pdevs-apply* (*pdevs-inner2*  $p$   $a$   $b$ )  $i = (\text{pdevs-apply}$   
 $p$   $i \cdot a, \text{pdevs-apply } p$   $i \cdot b)$   
 ⟨*proof*⟩

**definition** *inner2-aform*::'a::euclidean-space *aform*  $\Rightarrow$  'a  $\Rightarrow$  'a  $\Rightarrow$  (real\*real) *aform*  
**where** *inner2-aform*  $X$   $a$   $b = (\text{inner2 } (\text{fst } X) a b, \text{pdevs-inner2 } (\text{snd } X) a b)$

**lemma** *linear-inner2*[intro, simp]: *linear*  $(\lambda x. \text{inner2 } x n i)$   
 ⟨*proof*⟩

**lemma** *aform-val-inner2-aform*[simp]:  $aform\text{-}val\ e\ (inner2\text{-}aform\ Z\ n\ i) = inner2\ (aform\text{-}val\ e\ Z)\ n\ i$   
 ⟨proof⟩

## 2.18 Update

**lemma** *pdevs-val-upd*[simp]:  
 $pdevs\text{-}val\ (e(n := e'))\ X = pdevs\text{-}val\ e\ X - e\ n * pdevs\text{-}apply\ X\ n + e' * pdevs\text{-}apply\ X\ n$   
 ⟨proof⟩

**lemma** *nonzeros-fun-upd*:  
 $\{i. (f(n := a))\ i \neq 0\} \subseteq \{i. f\ i \neq 0\} \cup \{n\}$   
 ⟨proof⟩

**lift-definition** *pdev-upd*:: $'a::real\text{-}vector\ pdevs \Rightarrow nat \Rightarrow 'a \Rightarrow 'a\ pdevs$   
**is**  $\lambda x\ n\ a. x(n:=a)$   
 ⟨proof⟩

**lemma** *pdevs-apply-pdev-upd*[simp]:  
 $pdevs\text{-}apply\ (pdev\text{-}upd\ X\ n\ x) = (pdevs\text{-}apply\ X)(n:=x)$   
 ⟨proof⟩

**lemma** *pdevs-val-pdev-upd*[simp]:  
 $pdevs\text{-}val\ e\ (pdev\text{-}upd\ X\ n\ x) = pdevs\text{-}val\ e\ X + e\ n *_{R}\ x - e\ n *_{R}\ pdevs\text{-}apply\ X\ n$   
 ⟨proof⟩

**lemma** *degree-pdev-upd*:  
**assumes**  $x = 0 \longleftrightarrow pdevs\text{-}apply\ X\ n = 0$   
**shows**  $degree\ (pdev\text{-}upd\ X\ n\ x) = degree\ X$   
 ⟨proof⟩

**lemma** *degree-pdev-upd-le*:  
**assumes**  $degree\ X \leq n$   
**shows**  $degree\ (pdev\text{-}upd\ X\ n\ x) \leq Suc\ n$   
 ⟨proof⟩

## 2.19 Inf/Sup

**definition** *Inf-aform*  $X = fst\ X - tdev\ (snd\ X)$

**definition** *Sup-aform*  $X = fst\ X + tdev\ (snd\ X)$

**lemma** *Inf-aform*:  
**assumes**  $e \in UNIV \rightarrow \{-1 .. 1\}$   
**shows**  $Inf\text{-}aform\ X \leq aform\text{-}val\ e\ X$   
 ⟨proof⟩

**lemma** *Sup-aform*:

**assumes**  $e \in UNIV \rightarrow \{-1 .. 1\}$   
**shows**  $aform\text{-}val\ e\ X \leq Sup\text{-}aform\ X$   
 $\langle proof \rangle$

## 2.20 Minkowski Sum

**definition**  $msum\text{-}pdevs\text{-}raw::nat \Rightarrow (nat \Rightarrow 'a::real\text{-}vector) \Rightarrow (nat \Rightarrow 'a) \Rightarrow (nat \Rightarrow 'a)$   
**where**

$msum\text{-}pdevs\text{-}raw\ n\ x\ y\ i = (if\ i < n\ then\ x\ i\ else\ y\ (i - n))$

**lemma**  $nonzeros\text{-}msum\text{-}pdevs\text{-}raw$ :

$\{i. msum\text{-}pdevs\text{-}raw\ n\ f\ g\ i \neq 0\} = (\{0..<n\} \cap \{i. f\ i \neq 0\}) \cup (+)\ n\ '(\{i. g\ i \neq 0\})$   
 $\langle proof \rangle$

**lift-definition**  $msum\text{-}pdevs::nat \Rightarrow 'a::real\text{-}vector\ pdevs \Rightarrow 'a\ pdevs \Rightarrow 'a\ pdevs$  **is**  $msum\text{-}pdevs\text{-}raw$   
 $\langle proof \rangle$

**lemma**  $pdevs\text{-}apply\text{-}msum\text{-}pdevs$ :  $pdevs\text{-}apply\ (msum\text{-}pdevs\ n\ f\ g)\ i =$   
 $(if\ i < n\ then\ pdevs\text{-}apply\ f\ i\ else\ pdevs\text{-}apply\ g\ (i - n))$   
 $\langle proof \rangle$

**lemma**  $degree\text{-}least\text{-}nonzero$ :

**assumes**  $degree\ f \neq 0$   
**shows**  $pdevs\text{-}apply\ f\ (degree\ f - 1) \neq 0$   
 $\langle proof \rangle$

**lemma**  $degree\text{-}leI$ :

**assumes**  $(\bigwedge i. pdevs\text{-}apply\ y\ i = 0 \implies pdevs\text{-}apply\ x\ i = 0)$   
**shows**  $degree\ x \leq degree\ y$   
 $\langle proof \rangle$

**lemma**  $degree\text{-}msum\text{-}pdevs\text{-}ge1$ :

**shows**  $degree\ f \leq n \implies degree\ f \leq degree\ (msum\text{-}pdevs\ n\ f\ g)$   
 $\langle proof \rangle$

**lemma**  $degree\text{-}msum\text{-}pdevs\text{-}ge2$ :

**assumes**  $degree\ f \leq n$   
**shows**  $degree\ g \leq degree\ (msum\text{-}pdevs\ n\ f\ g) - n$   
 $\langle proof \rangle$

**lemma**  $degree\text{-}msum\text{-}pdevs\text{-}le$ :

**shows**  $degree\ (msum\text{-}pdevs\ n\ f\ g) \leq n + degree\ g$   
 $\langle proof \rangle$

**lemma**

$sum\text{-}msum\text{-}pdevs\text{-}cases$ :

**assumes**  $degree\ f \leq n$

**assumes**  $[simp]: \bigwedge i. e\ i\ 0 = 0$

**shows**

$(\sum i < \text{degree } (\text{msum-pdevs } n f g).$   
 $e i \text{ (if } i < n \text{ then } \text{pdevs-apply } f i \text{ else } \text{pdevs-apply } g (i - n))) =$   
 $(\sum i < \text{degree } f. e i (\text{pdevs-apply } f i)) + (\sum i < \text{degree } g. e (i + n) (\text{pdevs-apply}$   
 $g i))$   
**(is ?lhs = ?rhs)**  
*<proof>*

**lemma** *tdev-msum-pdevs*:  $\text{degree } f \leq n \implies \text{tdev } (\text{msum-pdevs } n f g) = \text{tdev } f + \text{tdev } g$   
*<proof>*

**lemma** *pdevs-val-msum-pdevs*:  
 $\text{degree } f \leq n \implies \text{pdevs-val } e (\text{msum-pdevs } n f g) = \text{pdevs-val } e f + \text{pdevs-val } (\lambda i. e (i + n)) g$   
*<proof>*

**definition** *msum-aform::nat*  $\Rightarrow$  *'a::real-vector aform*  $\Rightarrow$  *'a aform*  $\Rightarrow$  *'a aform*  
**where** *msum-aform*  $n f g = (\text{fst } f + \text{fst } g, \text{msum-pdevs } n (\text{snd } f) (\text{snd } g))$

**lemma** *fst-msum-aform[simp]*:  $\text{fst } (\text{msum-aform } n f g) = \text{fst } f + \text{fst } g$   
*<proof>*

**lemma** *snd-msum-aform[simp]*:  $\text{snd } (\text{msum-aform } n f g) = \text{msum-pdevs } n (\text{snd } f) (\text{snd } g)$   
*<proof>*

**lemma** *finite-nonzero-summable*:  $\text{finite } \{i. f i \neq 0\} \implies \text{summable } f$   
*<proof>*

**lemma** *aform-val-msum-aform*:  
**assumes**  $\text{degree-aform } f \leq n$   
**shows**  $\text{aform-val } e (\text{msum-aform } n f g) = \text{aform-val } e f + \text{aform-val } (\lambda i. e (i + n)) g$   
*<proof>*

**lemma** *Inf-aform-msum-aform*:  
 $\text{degree-aform } X \leq n \implies \text{Inf-aform } (\text{msum-aform } n X Y) = \text{Inf-aform } X + \text{Inf-aform } Y$   
*<proof>*

**lemma** *Sup-aform-msum-aform*:  
 $\text{degree-aform } X \leq n \implies \text{Sup-aform } (\text{msum-aform } n X Y) = \text{Sup-aform } X + \text{Sup-aform } Y$   
*<proof>*

**definition** *independent-from*  $d Y = \text{msum-aform } d (0, \text{zero-pdevs}) Y$

**definition** *independent-aform*  $X Y = \text{independent-from } (\text{degree-aform } X) Y$

**lemma** *degree-zero-pdevs[simp]*: *degree zero-pdevs = 0*  
⟨*proof*⟩

**lemma** *independent-aform-Joints*:  
assumes  $x \in \text{Affine } X$   
assumes  $y \in \text{Affine } Y$   
shows  $[x, y] \in \text{Joints } [X, \text{independent-aform } X Y]$   
⟨*proof*⟩

**lemma** *msum-aform-Joints*:  
assumes  $d \geq \text{degree-aform } X$   
assumes  $\bigwedge X. X \in \text{set } XS \implies d \geq \text{degree-aform } X$   
assumes  $(x\#xs) \in \text{Joints } (X\#XS)$   
assumes  $y \in \text{Affine } Y$   
shows  $((x + y)\#x\#xs) \in \text{Joints } (\text{msum-aform } d X Y\#X\#XS)$   
⟨*proof*⟩

**lemma** *Joints-msum-aform*:  
assumes  $d \geq \text{degree-aform } X$   
assumes  $\bigwedge Y. Y \in \text{set } YS \implies d \geq \text{degree-aform } Y$   
shows  $\text{Joints } (\text{msum-aform } d X Y\#YS) = \{((x + y)\#ys) \mid x y ys. y \in \text{Affine } Y$   
 $\wedge x\#ys \in \text{Joints } (X\#YS)\}$   
⟨*proof*⟩

**lemma** *Joints-singleton-image*:  $\text{Joints } [x] = (\lambda x. [x]) \text{ ' Affine } x$   
⟨*proof*⟩

**lemma** *Collect-extract-image*:  $\{g (f x y) \mid x y. P x y\} = g \text{ ' } \{f x y \mid x y. P x y\}$   
⟨*proof*⟩

**lemma** *inj-Cons*:  $\text{inj } (\lambda x. x\#xs)$   
⟨*proof*⟩

**lemma** *Joints-Nil[simp]*:  $\text{Joints } [] = \{[]\}$   
⟨*proof*⟩

**lemma** *msum-pdevs-zero-ident[simp]*:  $\text{msum-pdevs } 0 \text{ zero-pdevs } x = x$   
⟨*proof*⟩

**lemma** *msum-aform-zero-ident[simp]*:  $\text{msum-aform } 0 (0, \text{zero-pdevs}) x = x$   
⟨*proof*⟩

**lemma** *mem-Joints-singleton*:  $(x \in \text{Joints } [X]) = (\exists y. x = [y] \wedge y \in \text{Affine } X)$   
⟨*proof*⟩

**lemma** *singleton-mem-Joints[simp]*:  $[x] \in \text{Joints } [X] \longleftrightarrow x \in \text{Affine } X$   
⟨*proof*⟩

**lemma** *msum-aform-Joints-without-first*:

**assumes**  $d \geq \text{degree-aform } X$   
**assumes**  $\bigwedge X. X \in \text{set } XS \implies d \geq \text{degree-aform } X$   
**assumes**  $(x\#xs) \in \text{Joints } (X\#XS)$   
**assumes**  $y \in \text{Affine } Y$   
**assumes**  $z = x + y$   
**shows**  $z\#xs \in \text{Joints } (\text{msum-aform } d \ X \ Y\#XS)$   
*<proof>*

**lemma** *Affine-msum-aform*:

**assumes**  $d \geq \text{degree-aform } X$   
**shows**  $\text{Affine } (\text{msum-aform } d \ X \ Y) = \{x + y \mid x \in \text{Affine } X \wedge y \in \text{Affine } Y\}$   
*<proof>*

**lemma** *Affine-zero-pdevs[simp]*:  $\text{Affine } (0, \text{zero-pdevs}) = \{0\}$   
*<proof>*

**lemma** *Affine-independent-aform*:

$\text{Affine } (\text{independent-aform } X \ Y) = \text{Affine } Y$   
*<proof>*

**lemma**

*abs-diff-eq1*:  
**fixes**  $l \ u :: 'a :: \text{ordered-euclidean-space}$   
**shows**  $l \leq u \implies |u - l| = u - l$   
*<proof>*

**lemma** *compact-sum*:

**fixes**  $f :: 'a \Rightarrow 'b :: \text{topological-space} \Rightarrow 'c :: \text{real-normed-vector}$   
**assumes** *finite*  $I$   
**assumes**  $\bigwedge i. i \in I \implies \text{compact } (S \ i)$   
**assumes**  $\bigwedge i. i \in I \implies \text{continuous-on } (S \ i) \ (f \ i)$   
**assumes**  $I \subseteq J$   
**shows**  $\text{compact } \{\sum_{i \in I} f \ i \ (x \ i) \mid x. x \in \text{Pi } J \ S\}$   
*<proof>*

**lemma** *compact-Affine*:

**fixes**  $X :: 'a :: \text{ordered-euclidean-space}$  *aform*  
**shows**  $\text{compact } (\text{Affine } X)$   
*<proof>*

**lemma** *Joints2-JointsI*:

$(xs, x) \in \text{Joints2 } XS \ X \implies x\#xs \in \text{Joints } (X\#XS)$   
*<proof>*

## 2.21 Splitting

**definition** *split-aform*  $X \ i =$

(let  $xi = pdevs\text{-}apply (snd X) i /_R 2$   
in  $((fst X - xi, pdev\text{-}upd (snd X) i xi), (fst X + xi, pdev\text{-}upd (snd X) i xi))$ )

**lemma** *split-aformE*:

**assumes**  $e \in UNIV \rightarrow \{-1 .. 1\}$

**assumes**  $x = aform\text{-}val e X$

**obtains**  $err$  **where**  $x = aform\text{-}val (e(i:=err)) (fst (split\text{-}aform X i))$   $err \in \{-1 .. 1\}$

|  $err$  **where**  $x = aform\text{-}val (e(i:=err)) (snd (split\text{-}aform X i))$   $err \in \{-1 .. 1\}$   
 $\langle proof \rangle$

**lemma** *pdevs-val-add*:  $pdevs\text{-}val (\lambda i. e i + f i) xs = pdevs\text{-}val e xs + pdevs\text{-}val f xs$

$\langle proof \rangle$

**lemma** *pdevs-val-minus*:  $pdevs\text{-}val (\lambda i. e i - f i) xs = pdevs\text{-}val e xs - pdevs\text{-}val f xs$

$\langle proof \rangle$

**lemma** *pdevs-val-cmul*:  $pdevs\text{-}val (\lambda i. u * e i) xs = u *_R pdevs\text{-}val e xs$

$\langle proof \rangle$

**lemma** *atLeastAtMost-absI*:  $- a \leq a \implies |x::real| \leq |a| \implies x \in atLeastAtMost (- a) a$

$\langle proof \rangle$

**lemma** *divide-atLeastAtMost-1-absI*:  $|x::real| \leq |a| \implies x/a \in \{-1 .. 1\}$

$\langle proof \rangle$

**lemma** *convex-scaleR-aux*:  $u + v = 1 \implies u *_R x + v *_R x = (x::'a::real\text{-}vector)$

$\langle proof \rangle$

**lemma** *convex-mult-aux*:  $u + v = 1 \implies u * x + v * x = (x::real)$

$\langle proof \rangle$

**lemma** *convex-Affine*:  $convex (Affine X)$

$\langle proof \rangle$

**lemma** *segment-in-aform-val*:

**assumes**  $e \in UNIV \rightarrow \{-1 .. 1\}$

**assumes**  $f \in UNIV \rightarrow \{-1 .. 1\}$

**shows**  $closed\text{-}segment (aform\text{-}val e X) (aform\text{-}val f X) \subseteq Affine X$

$\langle proof \rangle$

## 2.22 From List of Generators

**lift-definition** *pdevs-of-list*:: $'a::zero list \Rightarrow 'a pdevs$

**is**  $\lambda xs i. \text{if } i < \text{length } xs \text{ then } xs ! i \text{ else } 0$

$\langle proof \rangle$

**lemma** *pdevs-apply-pdevs-of-list*:

*pdevs-apply (pdevs-of-list xs) i = (if i < length xs then xs ! i else 0)*  
*<proof>*

**lemma** *pdevs-apply-pdevs-of-list-Nil[simp]*:

*pdevs-apply (pdevs-of-list []) i = 0*  
*<proof>*

**lemma** *pdevs-apply-pdevs-of-list-Cons*:

*pdevs-apply (pdevs-of-list (x # xs)) i =*  
*(if i = 0 then x else pdevs-apply (pdevs-of-list xs) (i - 1))*  
*<proof>*

**lemma** *pdevs-domain-pdevs-of-list-Cons[simp]*: *pdevs-domain (pdevs-of-list (x #*

*xs)) =*  
*(if x = 0 then {} else {0}) ∪ (+) 1 ‘ pdevs-domain (pdevs-of-list xs)*  
*<proof>*

**lemma** *pdevs-val-pdevs-of-list-eq[simp]*:

*pdevs-val e (pdevs-of-list (x # xs)) = e 0 \*<sub>R</sub> x + pdevs-val (e o (+) 1) (pdevs-of-list*  
*xs)*  
*<proof>*

**lemma**

*less-degree-pdevs-of-list-imp-less-length*:

**assumes** *i < degree (pdevs-of-list xs)*

**shows** *i < length xs*

*<proof>*

**lemma** *tdev-pdevs-of-list[simp]*: *tdev (pdevs-of-list xs) = sum-list (map abs xs)*

*<proof>*

**lemma** *pdevs-of-list-Nil[simp]*: *pdevs-of-list [] = zero-pdevs*

*<proof>*

**lemma** *pdevs-val-inj-sumI*:

**fixes** *K::'a set* **and** *g::'a ⇒ nat*

**assumes** *finite K*

**assumes** *inj-on g K*

**assumes** *pdevs-domain x ⊆ g ‘ K*

**assumes**  $\bigwedge i. i \in K \implies g\ i \notin \text{pdevs-domain } x \implies f\ i = 0$

**assumes**  $\bigwedge i. i \in K \implies g\ i \in \text{pdevs-domain } x \implies f\ i = e\ (g\ i) *_{\mathbb{R}} \text{pdevs-apply}$   
*x (g i)*

**shows** *pdevs-val e x = (∑ i∈K. f i)*

*<proof>*

**lemma** *pdevs-domain-pdevs-of-list-le*: *pdevs-domain (pdevs-of-list xs) ⊆ {0..<length*  
*xs}*

$\langle \text{proof} \rangle$

**lemma** *pdevs-val-zip*:  $\text{pdevs-val } e \ (\text{pdevs-of-list } xs) = (\sum (i,x) \leftarrow \text{zip } [0..<\text{length } xs]$

$xs. e \ i \ *_R \ x)$

$\langle \text{proof} \rangle$

**lemma** *pdevs-val-map*:

$\langle \text{pdevs-val } e \ (\text{pdevs-of-list } xs)$

$= (\sum n \leftarrow [0..<\text{length } xs]. e \ n \ *_R \ xs \ ! \ n)$

$\langle \text{proof} \rangle$

**lemma** *scaleR-sum-list*:

**fixes**  $xs::'a::\text{real-vector list}$

**shows**  $a \ *_R \ \text{sum-list } xs = \text{sum-list } (\text{map } (\text{scaleR } a) \ xs)$

$\langle \text{proof} \rangle$

**lemma** *pdevs-val-const-pdevs-of-list*:  $\text{pdevs-val } (\lambda-. \ c) \ (\text{pdevs-of-list } xs) = c \ *_R \ \text{sum-list } xs$

$\langle \text{proof} \rangle$

**lemma** *pdevs-val-partition*:

**assumes**  $e \in \text{UNIV} \rightarrow I$

**obtains**  $f \ g$  **where**  $\text{pdevs-val } e \ (\text{pdevs-of-list } xs) =$

$\text{pdevs-val } f \ (\text{pdevs-of-list } (\text{filter } p \ xs)) +$

$\text{pdevs-val } g \ (\text{pdevs-of-list } (\text{filter } (\text{Not } o \ p) \ xs))$

$f \in \text{UNIV} \rightarrow I$

$g \in \text{UNIV} \rightarrow I$

$\langle \text{proof} \rangle$

**lemma** *pdevs-apply-pdevs-of-list-append*:

$\text{pdevs-apply } (\text{pdevs-of-list } (xs \ @ \ zs)) \ i =$

$(\text{if } i < \text{length } xs$

$\text{then } \text{pdevs-apply } (\text{pdevs-of-list } xs) \ i \ \text{else } \text{pdevs-apply } (\text{pdevs-of-list } zs) \ (i - \text{length } xs))$

$\langle \text{proof} \rangle$

**lemma** *degree-pdevs-of-list-le-length*[*intro, simp*]:  $\text{degree } (\text{pdevs-of-list } xs) \leq \text{length } xs$

$\langle \text{proof} \rangle$

**lemma** *degree-pdevs-of-list-append*:

$\text{degree } (\text{pdevs-of-list } (xs \ @ \ ys)) \leq \text{length } xs + \text{degree } (\text{pdevs-of-list } ys)$

$\langle \text{proof} \rangle$

**lemma** *pdevs-val-pdevs-of-list-append*:

**assumes**  $f \in \text{UNIV} \rightarrow I$

**assumes**  $g \in \text{UNIV} \rightarrow I$

**obtains**  $e$  **where**

$\text{pdevs-val } f \ (\text{pdevs-of-list } xs) + \text{pdevs-val } g \ (\text{pdevs-of-list } ys) =$

$pdevs\text{-}val\ e\ (pdevs\text{-}of\text{-}list\ (xs\ @\ ys))$   
 $e \in UNIV \rightarrow I$   
 <proof>

**lemma**

*sum-general-mono:*  
**fixes**  $f::'a \Rightarrow ('b::ordered\text{-}ab\text{-}group\text{-}add)$   
**assumes**  $[simp,intro]:\ finite\ s\ finite\ t$   
**assumes**  $f: \bigwedge x. x \in s - t \implies f\ x \leq 0$   
**assumes**  $g: \bigwedge x. x \in t - s \implies g\ x \geq 0$   
**assumes**  $fg: \bigwedge x. x \in s \cap t \implies f\ x \leq g\ x$   
**shows**  $(\sum x \in s. f\ x) \leq (\sum x \in t. g\ x)$   
 <proof>

**lemma** *degree-pdevs-of-list-eq'*:

$\langle degree\ (pdevs\text{-}of\text{-}list\ xs) = Min\ \{n. n \leq length\ xs \wedge (\forall m. n \leq m \longrightarrow m < length\ xs \longrightarrow xs\ !\ m = 0)\} \rangle$   
 <proof>

**lemma** *pdevs-val-permuted:*

$\langle pdevs\text{-}val\ (e \circ p)\ (pdevs\text{-}of\text{-}list\ (permute\text{-}list\ p\ xs)) = pdevs\text{-}val\ e\ (pdevs\text{-}of\text{-}list\ xs) \rangle$  (is  $\langle ?r = ?s \rangle$ )  
**if**  $perm: \langle p\ permutes\ \{..<length\ xs\} \rangle$   
 <proof>

**lemma** *pdevs-val-perm-ex:*

**assumes**  $xs < \sim \sim > ys$   
**assumes**  $mem: e \in UNIV \rightarrow I$   
**shows**  $\exists e'. e' \in UNIV \rightarrow I \wedge pdevs\text{-}val\ e\ (pdevs\text{-}of\text{-}list\ xs) = pdevs\text{-}val\ e'\ (pdevs\text{-}of\text{-}list\ ys)$   
 <proof>

**lemma** *pdevs-val-perm:*

**assumes**  $xs < \sim \sim > ys$   
**assumes**  $mem: e \in UNIV \rightarrow I$   
**obtains**  $e'$  **where**  $e' \in UNIV \rightarrow I$   
 $pdevs\text{-}val\ e\ (pdevs\text{-}of\text{-}list\ xs) = pdevs\text{-}val\ e'\ (pdevs\text{-}of\text{-}list\ ys)$   
 <proof>

**lemma** *set-distinct-permI:*  $set\ xs = set\ ys \implies distinct\ xs \implies distinct\ ys \implies xs < \sim \sim > ys$   
 <proof>

**lemmas**  $pdevs\text{-}val\ permute = pdevs\text{-}val\ perm[OF\ set\text{-}distinct\text{-}permI]$

**lemma** *partition-permI:*

$filter\ p\ xs\ @\ filter\ (Not\ o\ p)\ xs < \sim \sim > xs$   
 <proof>

**lemma** *pdevs-val-eqI*:

**assumes**  $\bigwedge i. i \in \text{pdevs-domain } y \implies i \in \text{pdevs-domain } x \implies$   
 $e \ i \ *_R \ \text{pdevs-apply } x \ i = f \ i \ *_R \ \text{pdevs-apply } y \ i$   
**assumes**  $\bigwedge i. i \in \text{pdevs-domain } y \implies i \notin \text{pdevs-domain } x \implies f \ i \ *_R \ \text{pdevs-apply}$   
 $y \ i = 0$   
**assumes**  $\bigwedge i. i \in \text{pdevs-domain } x \implies i \notin \text{pdevs-domain } y \implies e \ i \ *_R \ \text{pdevs-apply}$   
 $x \ i = 0$   
**shows**  $\text{pdevs-val } e \ x = \text{pdevs-val } f \ y$   
 $\langle \text{proof} \rangle$

**definition**

*filter-pdevs-raw*:: $(\text{nat} \Rightarrow 'a \Rightarrow \text{bool}) \Rightarrow (\text{nat} \Rightarrow 'a::\text{real-vector}) \Rightarrow (\text{nat} \Rightarrow 'a)$   
**where**  $\text{filter-pdevs-raw } I \ X = (\lambda i. \text{if } I \ i \ (X \ i) \ \text{then } X \ i \ \text{else } 0)$

**lemma** *filter-pdevs-raw-nonzeros*:  $\{i. \text{filter-pdevs-raw } s \ f \ i \neq 0\} = \{i. f \ i \neq 0\} \cap$   
 $\{x. s \ x \ (f \ x)\}$   
 $\langle \text{proof} \rangle$

**lift-definition** *filter-pdevs*:: $(\text{nat} \Rightarrow 'a \Rightarrow \text{bool}) \Rightarrow 'a::\text{real-vector} \ \text{pdevs} \Rightarrow 'a \ \text{pdevs}$   
**is** *filter-pdevs-raw*  
 $\langle \text{proof} \rangle$

**lemma** *pdevs-apply-filter-pdevs[simp]*:

$\text{pdevs-apply } (\text{filter-pdevs } I \ x) \ i = (\text{if } I \ i \ (\text{pdevs-apply } x \ i) \ \text{then } \text{pdevs-apply } x \ i \ \text{else}$   
 $0)$   
 $\langle \text{proof} \rangle$

**lemma** *degree-filter-pdevs-le*:  $\text{degree } (\text{filter-pdevs } I \ x) \leq \text{degree } x$   
 $\langle \text{proof} \rangle$

**lemma** *pdevs-val-filter-pdevs*:

$\text{pdevs-val } e \ (\text{filter-pdevs } I \ x) =$   
 $(\sum i \in \{..<\text{degree } x\} \cap \{i. I \ i \ (\text{pdevs-apply } x \ i)\}. e \ i \ *_R \ \text{pdevs-apply } x \ i)$   
 $\langle \text{proof} \rangle$

**lemma** *pdevs-val-filter-pdevs-dom*:

$\text{pdevs-val } e \ (\text{filter-pdevs } I \ x) =$   
 $(\sum i \in \text{pdevs-domain } x \cap \{i. I \ i \ (\text{pdevs-apply } x \ i)\}. e \ i \ *_R \ \text{pdevs-apply } x \ i)$   
 $\langle \text{proof} \rangle$

**lemma** *pdevs-val-filter-pdevs-eval*:

$\text{pdevs-val } e \ (\text{filter-pdevs } p \ x) = \text{pdevs-val } (\lambda i. \text{if } p \ i \ (\text{pdevs-apply } x \ i) \ \text{then } e \ i \ \text{else}$   
 $0) \ x$   
 $\langle \text{proof} \rangle$

**definition** *pdevs-applys*  $X \ i = \text{map } (\lambda x. \text{pdevs-apply } x \ i) \ X$

**definition** *pdevs-vals*  $e \ X = \text{map } (\text{pdevs-val } e) \ X$

**definition** *aform-vals*  $e \ X = \text{map } (\text{aform-val } e) \ X$

**definition** *filter-pdevs-list*  $I \ X = \text{map } (\text{filter-pdevs } (\lambda i \ -. \ I \ i \ (\text{pdevs-applys } X \ i)))$

$X$

**lemma** *pdevs-applys-filter-pdevs-list[simp]*:

*pdevs-applys (filter-pdevs-list I X) i = (if I i (pdevs-applys X i) then pdevs-applys X i else map ( $\lambda$ -. 0) X)*  
*<proof>*

**definition** *degrees X = Max (insert 0 (degree ' set X))*

**abbreviation** *degree-aforms X  $\equiv$  degrees (map snd X)*

**lemma** *degrees-leI*:

**assumes**  $\bigwedge x. x \in \text{set } X \implies \text{degree } x \leq K$   
**shows** *degrees X  $\leq$  K*  
*<proof>*

**lemma** *degrees-leD*:

**assumes** *degrees X  $\leq$  K*  
**shows**  $\bigwedge x. x \in \text{set } X \implies \text{degree } x \leq K$   
*<proof>*

**lemma** *degree-filter-pdevs-list-le*: *degrees (filter-pdevs-list I x)  $\leq$  degrees x*  
*<proof>*

**definition** *dense-list-of-pdevs x = map ( $\lambda$ i. pdevs-apply x i) [0.. $\text{degree } x$ ]*

### 2.22.1 (reverse) ordered coefficients as list

**definition** *list-of-pdevs x =*

*map ( $\lambda$ i. (i, pdevs-apply x i)) (rev (sorted-list-of-set (pdevs-domain x)))*

**lemma** *list-of-pdevs-zero-pdevs[simp]*: *list-of-pdevs zero-pdevs = []*  
*<proof>*

**lemma** *sum-list-list-of-pdevs*: *sum-list (map snd (list-of-pdevs x)) = sum-list (dense-list-of-pdevs x)*  
*<proof>*

**lemma** *sum-list-filter-dense-list-of-pdevs[symmetric]*:

*sum-list (map snd (filter (p o snd) (list-of-pdevs x))) =*  
*sum-list (filter p (dense-list-of-pdevs x))*  
*<proof>*

**lemma** *pdevs-of-list-dense-list-of-pdevs*: *pdevs-of-list (dense-list-of-pdevs x) = x*  
*<proof>*

**lemma** *pdevs-val-sum-list*: *pdevs-val ( $\lambda$ -. c) X = c  $*_R$  sum-list (map snd (list-of-pdevs*

$X$ )  
 $\langle$ proof $\rangle$

**lemma** *list-of-pdevs-all-nonzero*:  $list\text{-}all\ (\lambda x. x \neq 0)\ (map\ snd\ (list\text{-}of\text{-}pdevs\ xs))$   
 $\langle$ proof $\rangle$

**lemma** *list-of-pdevs-nonzero*:  $x \in set\ (map\ snd\ (list\text{-}of\text{-}pdevs\ xs)) \implies x \neq 0$   
 $\langle$ proof $\rangle$

**lemma** *pdevs-of-list-scaleR-0*[simp]:  
**fixes**  $xs::'a::real\text{-}vector\ list$   
**shows**  $pdevs\text{-}of\text{-}list\ (map\ ((*_R)\ 0)\ xs) = zero\text{-}pdevs$   
 $\langle$ proof $\rangle$

**lemma** *degree-pdevs-of-list-scaleR*:  
 $degree\ (pdevs\text{-}of\text{-}list\ (map\ ((*_R)\ c)\ xs)) = (if\ c \neq 0\ then\ degree\ (pdevs\text{-}of\text{-}list\ xs)$   
 $else\ 0)$   
 $\langle$ proof $\rangle$

**lemma** *list-of-pdevs-eq*:  
 $rev\ (list\text{-}of\text{-}pdevs\ X) = (filter\ ((\neq)\ 0\ o\ snd)\ (map\ (\lambda i. (i,\ pdevs\text{-}apply\ X\ i))$   
 $[0..<degree\ X]))$   
**(is**  $- = filter\ ?P\ (map\ ?f\ ?xs)$   
 $\langle$ proof $\rangle$

**lemma** *sum-list-take-pdevs-val-eq*:  
 $sum\text{-}list\ (take\ d\ xs) = pdevs\text{-}val\ (\lambda i. if\ i < d\ then\ 1\ else\ 0)\ (pdevs\text{-}of\text{-}list\ xs)$   
 $\langle$ proof $\rangle$

**lemma** *zero-in-range-pdevs-apply*[intro, simp]:  
**fixes**  $X::'a::real\text{-}vector\ pdevs$  **shows**  $0 \in range\ (pdevs\text{-}apply\ X)$   
 $\langle$ proof $\rangle$

**lemma** *dense-list-in-range*:  $x \in set\ (dense\text{-}list\text{-}of\text{-}pdevs\ X) \implies x \in range\ (pdevs\text{-}apply\ X)$   
 $\langle$ proof $\rangle$

**lemma** *not-in-dense-list-zeroD*:  
**assumes**  $pdevs\text{-}apply\ X\ i \notin set\ (dense\text{-}list\text{-}of\text{-}pdevs\ X)$   
**shows**  $pdevs\text{-}apply\ X\ i = 0$   
 $\langle$ proof $\rangle$

**lemma** *list-all-list-of-pdevsI*:  
**assumes**  $\bigwedge i. i \in pdevs\text{-}domain\ X \implies P\ (pdevs\text{-}apply\ X\ i)$   
**shows**  $list\text{-}all\ (\lambda x. P\ x)\ (map\ snd\ (list\text{-}of\text{-}pdevs\ X))$   
 $\langle$ proof $\rangle$

**lemma** *pdevs-of-list-map-scaleR*:  
 $pdevs\text{-}of\text{-}list\ (map\ (scaleR\ r)\ xs) = scaleR\text{-}pdevs\ r\ (pdevs\text{-}of\text{-}list\ xs)$

*<proof>*

**lemma**

*map-permI:*

**assumes**  $xs < \sim \sim > ys$

**shows**  $map\ f\ xs < \sim \sim > map\ f\ ys$

*<proof>*

**lemma** *rev-perm:*  $rev\ xs < \sim \sim > ys \longleftrightarrow xs < \sim \sim > ys$

*<proof>*

**lemma** *list-of-pdevs-perm-filter-nonzero:*

$map\ snd\ (list-of-pdevs\ X) < \sim \sim > (filter\ ((\neq)\ 0)\ (dense-list-of-pdevs\ X))$

*<proof>*

**lemma** *pdevs-val-filter:*

**assumes**  $mem: e \in UNIV \rightarrow I$

**assumes**  $0 \in I$

**obtains**  $e'$  **where**

$pdevs-val\ e\ (pdevs-of-list\ (filter\ p\ xs)) = pdevs-val\ e'\ (pdevs-of-list\ xs)$

$e' \in UNIV \rightarrow I$

*<proof>*

**lemma**

*pdevs-val-of-list-of-pdevs:*

**assumes**  $e \in UNIV \rightarrow I$

**assumes**  $0 \in I$

**obtains**  $e'$  **where**

$pdevs-val\ e\ (pdevs-of-list\ (map\ snd\ (list-of-pdevs\ X))) = pdevs-val\ e'\ X$

$e' \in UNIV \rightarrow I$

*<proof>*

**lemma**

*pdevs-val-of-list-of-pdevs2:*

**assumes**  $e \in UNIV \rightarrow I$

**obtains**  $e'$  **where**

$pdevs-val\ e\ X = pdevs-val\ e'\ (pdevs-of-list\ (map\ snd\ (list-of-pdevs\ X)))$

$e' \in UNIV \rightarrow I$

*<proof>*

**lemma** *dense-list-of-pdevs-scaleR:*

$r \neq 0 \implies map\ ((*_R)\ r)\ (dense-list-of-pdevs\ x) = dense-list-of-pdevs\ (scaleR-pdevs\ r\ x)$

*<proof>*

**lemma** *degree-pdevs-of-list-eq:*

$(\bigwedge x. x \in set\ xs \implies x \neq 0) \implies degree\ (pdevs-of-list\ xs) = length\ xs$

*<proof>*

**lemma** *dense-list-of-pdevs-pdevs-of-list*:

$(\bigwedge x. x \in \text{set } xs \implies x \neq 0) \implies \text{dense-list-of-pdevs } (\text{pdevs-of-list } xs) = xs$   
(proof)

**lemma** *pdevs-of-list-sum*:

**assumes** *distinct* *xs*

**assumes**  $e \in \text{UNIV} \rightarrow I$

**obtains** *f* **where**  $f \in \text{UNIV} \rightarrow I$   $\text{pdevs-val } e (\text{pdevs-of-list } xs) = (\sum P \in \text{set } xs. f P *_R P)$   
(proof)

**lemma** *pdevs-domain-eq-pdevs-of-list*:

**assumes** *nz*:  $\bigwedge x. x \in \text{set } (xs) \implies x \neq 0$

**shows**  $\text{pdevs-domain } (\text{pdevs-of-list } xs) = \{0..<\text{length } xs\}$

(proof)

**lemma** *length-list-of-pdevs-pdevs-of-list*:

**assumes** *nz*:  $\bigwedge x. x \in \text{set } xs \implies x \neq 0$

**shows**  $\text{length } (\text{list-of-pdevs } (\text{pdevs-of-list } xs)) = \text{length } xs$

(proof)

**lemma** *nth-list-of-pdevs-pdevs-of-list*:

**assumes** *nz*:  $\bigwedge x. x \in \text{set } xs \implies x \neq 0$

**assumes** *l*:  $n < \text{length } xs$

**shows**  $\text{list-of-pdevs } (\text{pdevs-of-list } xs) ! n = ((\text{length } xs - \text{Suc } n), xs ! (\text{length } xs - \text{Suc } n))$

(proof)

**lemma** *list-of-pdevs-pdevs-of-list-eq*:

$(\bigwedge x. x \in \text{set } xs \implies x \neq 0) \implies$

$\text{list-of-pdevs } (\text{pdevs-of-list } xs) = \text{zip } (\text{rev } [0..<\text{length } xs]) (\text{rev } xs)$

(proof)

**lemma** *sum-list-filter-list-of-pdevs-of-list*:

**fixes** *xs*:*'a::comm-monoid-add list*

**assumes**  $\bigwedge x. x \in \text{set } xs \implies x \neq 0$

**shows**  $\text{sum-list } (\text{filter } p (\text{map } \text{snd } (\text{list-of-pdevs } (\text{pdevs-of-list } xs)))) = \text{sum-list } (\text{filter } p xs)$

(proof)

**lemma**

*sum-list-partition*:

**fixes** *xs*:*'a::comm-monoid-add list*

**shows**  $\text{sum-list } (\text{filter } p xs) + \text{sum-list } (\text{filter } (\text{Not } o p) xs) = \text{sum-list } xs$

(proof)

## 2.23 2d zonotopes

**definition** *prod-of-pdevs*  $x y = \text{binop-pdevs } \text{Pair } x y$

**lemma** *apply-pdevs-prod-of-pdevs*[simp]:  
 $pdevs\text{-}apply\ (prod\text{-}of\text{-}pdevs\ x\ y)\ i = (pdevs\text{-}apply\ x\ i,\ pdevs\text{-}apply\ y\ i)$   
 ⟨proof⟩

**lemma** *pdevs-domain-prod-of-pdevs*[simp]:  
 $pdevs\text{-}domain\ (prod\text{-}of\text{-}pdevs\ x\ y) = pdevs\text{-}domain\ x \cup pdevs\text{-}domain\ y$   
 ⟨proof⟩

**lemma** *pdevs-val-prod-of-pdevs*[simp]:  
 $pdevs\text{-}val\ e\ (prod\text{-}of\text{-}pdevs\ x\ y) = (pdevs\text{-}val\ e\ x,\ pdevs\text{-}val\ e\ y)$   
 ⟨proof⟩

**definition** *prod-of-aforms* (**infixr**  $\langle \times_a \rangle$  80)  
 where  $prod\text{-}of\text{-}aforms\ x\ y = ((fst\ x,\ fst\ y),\ prod\text{-}of\text{-}pdevs\ (snd\ x)\ (snd\ y))$

## 2.24 Intervals

**definition** *One-pdevs-raw*:: $nat \Rightarrow 'a::executable\text{-}euclidean\text{-}space$   
 where  $One\text{-}pdevs\text{-}raw\ i = (if\ i < length\ (Basis\text{-}list::'a\ list)\ then\ Basis\text{-}list\ !\ i\ else\ 0)$

**lemma** *zeros-One-pdevs-raw*:  
 $One\text{-}pdevs\text{-}raw\ -'\ \{0::'a::executable\text{-}euclidean\text{-}space\} = \{length\ (Basis\text{-}list::'a\ list)..'\}$   
 ⟨proof⟩

**lemma** *nonzeros-One-pdevs-raw*:  
 $\{i.\ One\text{-}pdevs\text{-}raw\ i \neq (0::'a::executable\text{-}euclidean\text{-}space)\} = -\ \{length\ (Basis\text{-}list::'a\ list)..'\}$   
 ⟨proof⟩

**lift-definition** *One-pdevs*:: $'a::executable\text{-}euclidean\text{-}space$  **is** *One-pdevs-raw*  
 ⟨proof⟩

**lemma** *pdevs-apply-One-pdevs*[simp]:  $pdevs\text{-}apply\ One\text{-}pdevs\ i =$   
 (if  $i < length\ (Basis\text{-}list::'a::executable\text{-}euclidean\text{-}space\ list)$  then  $Basis\text{-}list\ !\ i$  else  $0::'a$ )  
 ⟨proof⟩

**lemma** *Max-Collect-less-nat*:  $Max\ \{i::nat.\ i < k\} = (if\ k = 0\ then\ Max\ \{\}\ else\ k - 1)$   
 ⟨proof⟩

**lemma** *degree-One-pdevs*[simp]:  $degree\ (One\text{-}pdevs::'a\ pdevs) =$   
 $length\ (Basis\text{-}list::'a::executable\text{-}euclidean\text{-}space\ list)$   
 ⟨proof⟩

**definition** *inner-scaleR-pdevs*:: $'a::euclidean\text{-}space \Rightarrow 'a\ pdevs \Rightarrow 'a\ pdevs$

**where**  $inner-scaleR-pdevs\ b\ x = unop-pdevs\ (\lambda x. (b \cdot x) *_R x)\ x$

**lemma**  $pdevs-apply-inner-scaleR-pdevs[simp]$ :

$pdevs-apply\ (inner-scaleR-pdevs\ a\ x)\ i = (a \cdot (pdevs-apply\ x\ i)) *_R (pdevs-apply\ x\ i)$   
 $\langle proof \rangle$

**lemma**  $degree-inner-scaleR-pdevs-le$ :

$degree\ (inner-scaleR-pdevs\ (l::'a::executable-euclidean-space)\ One-pdevs) \leq$   
 $degree\ (One-pdevs::'a\ pdevs)$   
 $\langle proof \rangle$

**definition**  $pdevs-of-ivl\ l\ u = scaleR-pdevs\ (1/2)\ (inner-scaleR-pdevs\ (u - l)\ One-pdevs)$

**lemma**  $degree-pdevs-of-ivl-le$ :

$degree\ (pdevs-of-ivl\ l\ u::'a::executable-euclidean-space\ pdevs) \leq DIM('a)$   
 $\langle proof \rangle$

**lemma**  $pdevs-apply-pdevs-of-ivl$ :

**defines**  $B \equiv Basis-list::'a::executable-euclidean-space\ list$   
**shows**  $pdevs-apply\ (pdevs-of-ivl\ l\ u)\ i = (if\ i < length\ B\ then\ ((u - l) \cdot (B!i) / 2) *_R (B!i)$   
 $else\ 0)$   
 $\langle proof \rangle$

**lemma**  $deg-length-less-imp[simp]$ :

$k < degree\ (pdevs-of-ivl\ l\ u::'a::executable-euclidean-space\ pdevs) \implies$   
 $k < length\ (Basis-list::'a\ list)$   
 $\langle proof \rangle$

**lemma**  $tdev-pdevs-of-ivl$ :  $tdev\ (pdevs-of-ivl\ l\ u) = |u - l| /_R 2$

$\langle proof \rangle$

**definition**  $aform-of-ivl\ l\ u = ((l + u) /_R 2, pdevs-of-ivl\ l\ u)$

**definition**  $aform-of-point\ x = aform-of-ivl\ x\ x$

**lemma**  $Elem-affine-of-ivl-le$ :

**assumes**  $e \in UNIV \rightarrow \{-1 .. 1\}$   
**assumes**  $l \leq u$   
**shows**  $l \leq aform-val\ e\ (aform-of-ivl\ l\ u)$   
 $\langle proof \rangle$

**lemma**  $Elem-affine-of-ivl-ge$ :

**assumes**  $e \in UNIV \rightarrow \{-1 .. 1\}$   
**assumes**  $l \leq u$   
**shows**  $aform-val\ e\ (aform-of-ivl\ l\ u) \leq u$   
 $\langle proof \rangle$

**lemma**

*map-of-zip-upto-length-eq-nth*:  
**assumes**  $i < \text{length } B$   
**assumes**  $d = \text{length } B$   
**shows**  $(\text{map-of } (\text{zip } [0..<d] B) i) = \text{Some } (B ! i)$   
 $\langle \text{proof} \rangle$

**lemma** *in-ivl-affine-of-ivlE*:  
**assumes**  $k \in \{l .. u\}$   
**obtains**  $e$  **where**  $e \in UNIV \rightarrow \{-1 .. 1\} k = \text{aform-val } e (\text{aform-of-ivl } l u)$   
 $\langle \text{proof} \rangle$

**lemma** *Inf-aform-aform-of-ivl*:  
**assumes**  $l \leq u$   
**shows**  $\text{Inf-aform } (\text{aform-of-ivl } l u) = l$   
 $\langle \text{proof} \rangle$

**lemma** *Sup-aform-aform-of-ivl*:  
**assumes**  $l \leq u$   
**shows**  $\text{Sup-aform } (\text{aform-of-ivl } l u) = u$   
 $\langle \text{proof} \rangle$

**lemma** *Affine-aform-of-ivl*:  
 $a \leq b \implies \text{Affine } (\text{aform-of-ivl } a b) = \{a .. b\}$   
 $\langle \text{proof} \rangle$

**end**

### 3 Operations on Expressions

**theory** *Floatarith-Expression*  
**imports**  
*HOL-Decision-Procs.Approximation*  
*Affine-Arithmetic-Auxiliarities*  
*Executable-Euclidean-Space*  
**begin**

Much of this could move to the distribution...

#### 3.1 Approximating Expression\*s\*

**unbundle** *floatarith-syntax*

**primrec** *interpret-floatariths* :: *floatarith list*  $\Rightarrow$  *real list*  $\Rightarrow$  *real list*

**where**

$\text{interpret-floatariths } [] \text{ vs} = []$   
 $| \text{interpret-floatariths } (a\#bs) \text{ vs} = \text{interpret-floatarith } a \text{ vs}\#\text{interpret-floatariths } bs \text{ vs}$

**lemma** *length-interpret-floatariths[simp]*:  $\text{length } (\text{interpret-floatariths } fas \ xs) = \text{length } fas$

*<proof>*

**lemma** *interpret-floatariths-nth[simp]*:

$\text{interpret-floatariths } fas \ xs \ ! \ n = \text{interpret-floatarith } (fas \ ! \ n) \ xs$

**if**  $n < \text{length } fas$

*<proof>*

**abbreviation** *einterpret*  $\equiv \lambda fas \ vs. \ \text{eucl-of-list } (\text{interpret-floatariths } fas \ vs)$

## 3.2 Syntax

**syntax** *interpret-floatarith::floatarith*  $\Rightarrow \text{real list} \Rightarrow \text{real}$

**instantiation** *floatarith*  $:: \{plus, minus, uminus, times, inverse, zero, one\}$

**begin**

**definition**  $- f = \text{Minus } f$

**lemma** *interpret-floatarith-uminus[simp]*:

$\text{interpret-floatarith } (- f) \ xs = - \text{interpret-floatarith } f \ xs$

*<proof>*

**definition**  $f + g = \text{Add } f \ g$

**lemma** *interpret-floatarith-plus[simp]*:

$\text{interpret-floatarith } (f + g) \ xs = \text{interpret-floatarith } f \ xs + \text{interpret-floatarith } g \ xs$

*<proof>*

**definition**  $f - g = \text{Add } f \ (\text{Minus } g)$

**lemma** *interpret-floatarith-minus[simp]*:

$\text{interpret-floatarith } (f - g) \ xs = \text{interpret-floatarith } f \ xs - \text{interpret-floatarith } g \ xs$

*<proof>*

**definition**  $\text{inverse } f = \text{Inverse } f$

**lemma** *interpret-floatarith-inverse[simp]*:

$\text{interpret-floatarith } (\text{inverse } f) \ xs = \text{inverse } (\text{interpret-floatarith } f \ xs)$

*<proof>*

**definition**  $f * g = \text{Mult } f \ g$

**lemma** *interpret-floatarith-times[simp]*:

$\text{interpret-floatarith } (f * g) \ xs = \text{interpret-floatarith } f \ xs * \text{interpret-floatarith } g \ xs$

*<proof>*

**definition**  $f \ \text{div} \ g = f * \text{Inverse } g$

**lemma** *interpret-floatarith-divide[simp]*:

$\text{interpret-floatarith } (f \ \text{div} \ g) \ xs = \text{interpret-floatarith } f \ xs / \text{interpret-floatarith } g \ xs$

*<proof>*

**definition**  $1 = \text{Num } 1$

**lemma** *interpret-floatarith-one*[simp]:

*interpret-floatarith 1 xs = 1*

*<proof>*

**definition**  $0 = \text{Num } 0$

**lemma** *interpret-floatarith-zero*[simp]:

*interpret-floatarith 0 xs = 0*

*<proof>*

**instance** *<proof>*

**end**

### 3.3 Derived symbols

**definition**  $R_e r = (\text{case quotient-of } r \text{ of } (n, d) \Rightarrow \text{Num } (\text{of-int } n) / \text{Num } (\text{of-int } d))$

**declare** [[*coercion*  $R_e$  ]]

**lemma** *interpret-Re*[simp]: *interpret-floatarith (R<sub>e</sub> x) xs = real-of-rat x*

*<proof>*

**definition**  $\text{Sin } x = \text{Cos } ((\text{Pi} * (\text{Num } (\text{Float } 1 (-1)))) - x)$

**lemma** *interpret-floatarith-Sin*[simp]:

*interpret-floatarith (Sin x) vs = sin (interpret-floatarith x vs)*

*<proof>*

**definition**  $\text{Half } x = \text{Mult } (\text{Num } (\text{Float } 1 (-1))) x$

**lemma** *interpret-Half*[simp]: *interpret-floatarith (Half x) xs = interpret-floatarith x xs / 2*

*<proof>*

**definition**  $\text{Tan } x = (\text{Sin } x) / (\text{Cos } x)$

**lemma** *interpret-floatarith-Tan*[simp]:

*interpret-floatarith (Tan x) vs = tan (interpret-floatarith x vs)*

*<proof>*

**primrec**  $\text{Sum}_e$  **where**

$\text{Sum}_e f [] = 0$

|  $\text{Sum}_e f (x\#xs) = f x + \text{Sum}_e f xs$

**lemma** *interpret-floatarith-Sum<sub>e</sub>*[simp]:

*interpret-floatarith (Sum<sub>e</sub> f x) vs = ( $\sum i \leftarrow x.$  interpret-floatarith (f i) vs)*

*<proof>*

**definition** *Norm* where  $Norm\ is = Sqrt\ (Sum_e\ (\lambda i.\ i * i)\ is)$

**lemma** *interpret-floatarith-norm*[simp]:

**assumes** [simp]:  $length\ x = DIM('a)$

**shows** *interpret-floatarith*  $(Norm\ x)\ vs = norm\ (einterpret\ x\ vs::'a::executable-euclidean-space)$

*<proof>*

**notation** *floatarith.Power* (**infixr**  $\langle \hat{\ }_e \rangle\ 80$ )

### 3.4 Constant Folding

**fun** *dest-Num-fa* where

*dest-Num-fa*  $(floatarith.Num\ x) = Some\ x$

| *dest-Num-fa*  $- = None$

**fun-cases** *dest-Num-fa-None*: *dest-Num-fa*  $fa = None$

**and** *dest-Num-fa-Some*: *dest-Num-fa*  $fa = Some\ x$

**fun** *fold-const-fa* where

*fold-const-fa*  $(Add\ fa1\ fa2) =$

$(let\ (ffa1,\ ffa2) = (fold-const-fa\ fa1,\ fold-const-fa\ fa2)$

$in\ case\ (dest-Num-fa\ ffa1,\ dest-Num-fa\ (ffa2))\ of$

$(Some\ a,\ Some\ b) \Rightarrow Num\ (a + b)$

|  $(Some\ a,\ None) \Rightarrow (if\ a = 0\ then\ ffa2\ else\ Add\ (Num\ a)\ ffa2)$

|  $(None,\ Some\ a) \Rightarrow (if\ a = 0\ then\ ffa1\ else\ Add\ ffa1\ (Num\ a))$

|  $(None,\ None) \Rightarrow Add\ ffa1\ ffa2)$

| *fold-const-fa*  $(Minus\ a) =$

$(case\ (fold-const-fa\ a)\ of$

$(Num\ x) \Rightarrow Num\ (-x)$

|  $x \Rightarrow Minus\ x)$

| *fold-const-fa*  $(Mult\ fa1\ fa2) =$

$(let\ (ffa1,\ ffa2) = (fold-const-fa\ fa1,\ fold-const-fa\ fa2)$

$in\ case\ (dest-Num-fa\ ffa1,\ dest-Num-fa\ (ffa2))\ of$

$(Some\ a,\ Some\ b) \Rightarrow Num\ (a * b)$

|  $(Some\ a,\ None) \Rightarrow (if\ a = 0\ then\ Num\ 0\ else\ if\ a = 1\ then\ ffa2\ else\ Mult\ (Num\ a)\ ffa2)$

|  $(None,\ Some\ a) \Rightarrow (if\ a = 0\ then\ Num\ 0\ else\ if\ a = 1\ then\ ffa1\ else\ Mult\ ffa1\ (Num\ a))$

|  $(None,\ None) \Rightarrow Mult\ ffa1\ ffa2)$

| *fold-const-fa*  $(Inverse\ a) = Inverse\ (fold-const-fa\ a)$

| *fold-const-fa*  $(Abs\ a) =$

$(case\ (fold-const-fa\ a)\ of$

$(Num\ x) \Rightarrow Num\ (abs\ x)$

|  $x \Rightarrow Abs\ x)$

| *fold-const-fa*  $(Max\ a\ b) =$

$(case\ (fold-const-fa\ a,\ fold-const-fa\ b)\ of$

$(Num\ x,\ Num\ y) \Rightarrow Num\ (max\ x\ y)$

|  $(x,\ y) \Rightarrow Max\ x\ y)$

| *fold-const-fa*  $(Min\ a\ b) =$

```

    (case (fold-const-fa a, fold-const-fa b) of
      (Num x, Num y) ⇒ Num (min x y)
    | (x, y) ⇒ Min x y)
| fold-const-fa (Floor a) =
  (case (fold-const-fa a) of
    (Num x) ⇒ Num (floor-fl x)
  | x ⇒ Floor x)
| fold-const-fa (Power a b) =
  (case (fold-const-fa a) of
    (Num x) ⇒ Num (x ^ b)
  | x ⇒ Power x b)
| fold-const-fa (Cos a) = Cos (fold-const-fa a)
| fold-const-fa (Arctan a) = Arctan (fold-const-fa a)
| fold-const-fa (Sqrt a) = Sqrt (fold-const-fa a)
| fold-const-fa (Exp a) = Exp (fold-const-fa a)
| fold-const-fa (Ln a) = Ln (fold-const-fa a)
| fold-const-fa (Powr a b) = Powr (fold-const-fa a) (fold-const-fa b)
| fold-const-fa Pi = Pi
| fold-const-fa (Var v) = Var v
| fold-const-fa (Num x) = Num x

```

**fun-cases** *fold-const-fa-Num*: *fold-const-fa fa = Num y*  
**and** *fold-const-fa-Add*: *fold-const-fa fa = Add x y*  
**and** *fold-const-fa-Minus*: *fold-const-fa fa = Minus y*

**lemma** *fold-const-fa[simp]*: *interpret-floatarith (fold-const-fa fa) xs = interpret-floatarith fa xs*  
 ⟨*proof*⟩

### 3.5 Free Variables

**primrec** *max-Var-floatarith* **where**— TODO: include bound in predicate

```

  max-Var-floatarith (Add a b) = max (max-Var-floatarith a) (max-Var-floatarith
b)
| max-Var-floatarith (Mult a b) = max (max-Var-floatarith a) (max-Var-floatarith
b)
| max-Var-floatarith (Inverse a) = max-Var-floatarith a
| max-Var-floatarith (Minus a) = max-Var-floatarith a
| max-Var-floatarith (Num a) = 0
| max-Var-floatarith (Var i) = Suc i
| max-Var-floatarith (Cos a) = max-Var-floatarith a
| max-Var-floatarith (floatarith.Arctan a) = max-Var-floatarith a
| max-Var-floatarith (Abs a) = max-Var-floatarith a
| max-Var-floatarith (floatarith.Max a b) = max (max-Var-floatarith a) (max-Var-floatarith
b)
| max-Var-floatarith (floatarith.Min a b) = max (max-Var-floatarith a) (max-Var-floatarith
b)
| max-Var-floatarith (floatarith.Pi) = 0
| max-Var-floatarith (Sqrt a) = max-Var-floatarith a

```

|  $\text{max-Var-floatarith (Exp } a) = \text{max-Var-floatarith } a$   
|  $\text{max-Var-floatarith (Powr } a \ b) = \text{max (max-Var-floatarith } a) (\text{max-Var-floatarith } b)$   
|  $\text{max-Var-floatarith (floatarith.Ln } a) = \text{max-Var-floatarith } a$   
|  $\text{max-Var-floatarith (Power } a \ n) = \text{max-Var-floatarith } a$   
|  $\text{max-Var-floatarith (Floor } a) = \text{max-Var-floatarith } a$

**primrec** *max-Var-floatariths* **where**

$\text{max-Var-floatariths } [] = 0$   
|  $\text{max-Var-floatariths (} x\#xs) = \text{max (max-Var-floatarith } x) (\text{max-Var-floatariths } xs)$

**primrec** *max-Var-form* **where**

$\text{max-Var-form (Conj } a \ b) = \text{max (max-Var-form } a) (\text{max-Var-form } b)$   
|  $\text{max-Var-form (Disj } a \ b) = \text{max (max-Var-form } a) (\text{max-Var-form } b)$   
|  $\text{max-Var-form (Less } a \ b) = \text{max (max-Var-floatarith } a) (\text{max-Var-floatarith } b)$   
|  $\text{max-Var-form (LessEqual } a \ b) = \text{max (max-Var-floatarith } a) (\text{max-Var-floatarith } b)$   
|  $\text{max-Var-form (Bound } a \ b \ c \ d) = \text{linorder-class.Max } \{ \text{max-Var-floatarith } a, \text{max-Var-floatarith } b, \text{max-Var-floatarith } c, \text{max-Var-form } d \}$   
|  $\text{max-Var-form (AtLeastAtMost } a \ b \ c) = \text{linorder-class.Max } \{ \text{max-Var-floatarith } a, \text{max-Var-floatarith } b, \text{max-Var-floatarith } c \}$   
|  $\text{max-Var-form (Assign } a \ b \ c) = \text{linorder-class.Max } \{ \text{max-Var-floatarith } a, \text{max-Var-floatarith } b, \text{max-Var-form } c \}$

**lemma**

*interpret-floatarith-eq-take-max-VarI:*  
**assumes**  $\text{take (max-Var-floatarith } ra) \ ys = \text{take (max-Var-floatarith } ra) \ zs$   
**shows**  $\text{interpret-floatarith } ra \ ys = \text{interpret-floatarith } ra \ zs$   
*<proof>*

**lemma**

*interpret-floatariths-eq-take-max-VarI:*  
**assumes**  $\text{take (max-Var-floatariths } ea) \ ys = \text{take (max-Var-floatariths } ea) \ zs$   
**shows**  $\text{interpret-floatariths } ea \ ys = \text{interpret-floatariths } ea \ zs$   
*<proof>*

**lemma** *Max-Image-distrib:*

**includes** *no floatarith-syntax*  
**assumes**  $\text{finite } X \ X \neq \{ \}$   
**shows**  $\text{Max } ((\lambda x. \text{max (} f1 \ x) \ (f2 \ x)) \ 'X) = \text{max (Max (} f1 \ 'X) \ (Max (f2 \ 'X)))$   
*<proof>*

**lemma** *max-Var-floatarith-simps[simp]:*

$\text{max-Var-floatarith (} a / b) = \text{max (max-Var-floatarith } a) (\text{max-Var-floatarith } b)$   
 $\text{max-Var-floatarith (} a * b) = \text{max (max-Var-floatarith } a) (\text{max-Var-floatarith } b)$   
 $\text{max-Var-floatarith (} a + b) = \text{max (max-Var-floatarith } a) (\text{max-Var-floatarith } b)$   
 $\text{max-Var-floatarith (} a - b) = \text{max (max-Var-floatarith } a) (\text{max-Var-floatarith } b)$

$max\text{-}Var\text{-}floatarith (- b) = (max\text{-}Var\text{-}floatarith b)$   
(proof)

**lemma**  $max\text{-}Var\text{-}floatariths\text{-}Max$ :

$max\text{-}Var\text{-}floatariths xs = (if set xs = \{\} then 0 else linorder\text{-}class.Max (max\text{-}Var\text{-}floatarith ' set xs))$   
(proof)

**lemma**  $max\text{-}Var\text{-}floatariths\text{-}map\text{-}plus[simp]$ :

$max\text{-}Var\text{-}floatariths (map (\lambda i. fa1 i + fa2 i) xs) = max (max\text{-}Var\text{-}floatariths (map fa1 xs)) (max\text{-}Var\text{-}floatariths (map fa2 xs))$   
(proof)

**lemma**  $max\text{-}Var\text{-}floatariths\text{-}map\text{-}times[simp]$ :

$max\text{-}Var\text{-}floatariths (map (\lambda i. fa1 i * fa2 i) xs) = max (max\text{-}Var\text{-}floatariths (map fa1 xs)) (max\text{-}Var\text{-}floatariths (map fa2 xs))$   
(proof)

**lemma**  $max\text{-}Var\text{-}floatariths\text{-}map\text{-}divide[simp]$ :

$max\text{-}Var\text{-}floatariths (map (\lambda i. fa1 i / fa2 i) xs) = max (max\text{-}Var\text{-}floatariths (map fa1 xs)) (max\text{-}Var\text{-}floatariths (map fa2 xs))$   
(proof)

**lemma**  $max\text{-}Var\text{-}floatariths\text{-}map\text{-}uminus[simp]$ :

$max\text{-}Var\text{-}floatariths (map (\lambda i. - fa1 i) xs) = max\text{-}Var\text{-}floatariths (map fa1 xs)$   
(proof)

**lemma**  $max\text{-}Var\text{-}floatariths\text{-}map\text{-}const[simp]$ :

$max\text{-}Var\text{-}floatariths (map (\lambda i. fa) xs) = (if xs = [] then 0 else max\text{-}Var\text{-}floatarith fa)$   
(proof)

**lemma**  $max\text{-}Var\text{-}floatariths\text{-}map\text{-}minus[simp]$ :

$max\text{-}Var\text{-}floatariths (map (\lambda i. fa1 i - fa2 i) xs) = max (max\text{-}Var\text{-}floatariths (map fa1 xs)) (max\text{-}Var\text{-}floatariths (map fa2 xs))$   
(proof)

**primrec**  $fresh\text{-}floatarith$  where

$fresh\text{-}floatarith (Var y) x \longleftrightarrow (x \neq y)$   
|  $fresh\text{-}floatarith (Num a) x \longleftrightarrow True$   
|  $fresh\text{-}floatarith Pi x \longleftrightarrow True$   
|  $fresh\text{-}floatarith (Cos a) x \longleftrightarrow fresh\text{-}floatarith a x$   
|  $fresh\text{-}floatarith (Abs a) x \longleftrightarrow fresh\text{-}floatarith a x$   
|  $fresh\text{-}floatarith (Arctan a) x \longleftrightarrow fresh\text{-}floatarith a x$   
|  $fresh\text{-}floatarith (Sqrt a) x \longleftrightarrow fresh\text{-}floatarith a x$   
|  $fresh\text{-}floatarith (Exp a) x \longleftrightarrow fresh\text{-}floatarith a x$   
|  $fresh\text{-}floatarith (Floor a) x \longleftrightarrow fresh\text{-}floatarith a x$

| *fresh-floatarith (Power a n) x*  $\longleftrightarrow$  *fresh-floatarith a x*  
 | *fresh-floatarith (Minus a) x*  $\longleftrightarrow$  *fresh-floatarith a x*  
 | *fresh-floatarith (Ln a) x*  $\longleftrightarrow$  *fresh-floatarith a x*  
 | *fresh-floatarith (Inverse a) x*  $\longleftrightarrow$  *fresh-floatarith a x*  
 | *fresh-floatarith (Add a b) x*  $\longleftrightarrow$  *fresh-floatarith a x*  $\wedge$  *fresh-floatarith b x*  
 | *fresh-floatarith (Mult a b) x*  $\longleftrightarrow$  *fresh-floatarith a x*  $\wedge$  *fresh-floatarith b x*  
 | *fresh-floatarith (Max a b) x*  $\longleftrightarrow$  *fresh-floatarith a x*  $\wedge$  *fresh-floatarith b x*  
 | *fresh-floatarith (Min a b) x*  $\longleftrightarrow$  *fresh-floatarith a x*  $\wedge$  *fresh-floatarith b x*  
 | *fresh-floatarith (Powr a b) x*  $\longleftrightarrow$  *fresh-floatarith a x*  $\wedge$  *fresh-floatarith b x*

**lemma** *fresh-floatarith-subst*:

**fixes** *v::float*

**assumes** *fresh-floatarith e x*

**assumes**  $x < \text{length } vs$

**shows**  $\text{interpret-floatarith } e (vs[x:=v]) = \text{interpret-floatarith } e vs$

*<proof>*

**lemma** *fresh-floatarith-max-Var*:

**assumes** *max-Var-floatarith ea*  $\leq i$

**shows** *fresh-floatarith ea i*

*<proof>*

**primrec** *fresh-floatariths where*

*fresh-floatariths [] x*  $\longleftrightarrow$  *True*

| *fresh-floatariths (a#as) x*  $\longleftrightarrow$  *fresh-floatarith a x*  $\wedge$  *fresh-floatariths as x*

**lemma** *fresh-floatariths-max-Var*:

**assumes** *max-Var-floatariths ea*  $\leq i$

**shows** *fresh-floatariths ea i*

*<proof>*

**lemma**

*interpret-floatariths-take-eqI*:

**assumes**  $\text{take } n \text{ } ys = \text{take } n \text{ } zs$

**assumes** *max-Var-floatariths ea*  $\leq n$

**shows**  $\text{interpret-floatariths } ea \text{ } ys = \text{interpret-floatariths } ea \text{ } zs$

*<proof>*

**lemma**

*interpret-floatarith-fresh-eqI*:

**assumes**  $\bigwedge i. \text{fresh-floatarith } ea \text{ } i \vee (i < \text{length } ys \wedge i < \text{length } zs \wedge ys ! i = zs ! i)$

**shows**  $\text{interpret-floatarith } ea \text{ } ys = \text{interpret-floatarith } ea \text{ } zs$

*<proof>*

**lemma**

*interpret-floatariths-fresh-eqI*:

**assumes**  $\bigwedge i. \text{fresh-floatariths } ea \text{ } i \vee (i < \text{length } ys \wedge i < \text{length } zs \wedge ys ! i = zs ! i)$

**shows**  $\text{interpret-floatariths } ea \ ys = \text{interpret-floatariths } ea \ zs$   
(proof)

**lemma**

*interpret-floatarith-max-Var-cong:*

**assumes**  $\bigwedge i. i < \text{max-Var-floatarith } f \implies xs \ ! \ i = ys \ ! \ i$

**shows**  $\text{interpret-floatarith } f \ ys = \text{interpret-floatarith } f \ xs$

(proof)

**lemma**

*interpret-floatarith-fresh-cong:*

**assumes**  $\bigwedge i. \neg \text{fresh-floatarith } f \ i \implies xs \ ! \ i = ys \ ! \ i$

**shows**  $\text{interpret-floatarith } f \ ys = \text{interpret-floatarith } f \ xs$

(proof)

**lemma** *max-Var-floatarith-le-max-Var-floatariths:*

$fa \in \text{set } fas \implies \text{max-Var-floatarith } fa \leq \text{max-Var-floatariths } fas$

(proof)

**lemma** *max-Var-floatarith-le-max-Var-floatariths-nth:*

$n < \text{length } fas \implies \text{max-Var-floatarith } (fas \ ! \ n) \leq \text{max-Var-floatariths } fas$

(proof)

**lemma** *max-Var-floatariths-leI:*

**assumes**  $\bigwedge i. i < \text{length } xs \implies \text{max-Var-floatarith } (xs \ ! \ i) \leq F$

**shows**  $\text{max-Var-floatariths } xs \leq F$

(proof)

**lemma** *fresh-floatariths-map-Var[simp]:*

$\text{fresh-floatariths } (\text{map } \text{floatarith. Var } xs) \ i \longleftrightarrow i \notin \text{set } xs$

(proof)

**lemma** *max-Var-floatarith-fold-const-fa:*

$\text{max-Var-floatarith } (\text{fold-const-fa } fa) \leq \text{max-Var-floatarith } fa$

(proof)

**lemma** *max-Var-floatariths-fold-const-fa:*

$\text{max-Var-floatariths } (\text{map fold-const-fa } xs) \leq \text{max-Var-floatariths } xs$

(proof)

**lemma** *interpret-form-max-Var-cong:*

**assumes**  $\bigwedge i. i < \text{max-Var-form } f \implies xs \ ! \ i = ys \ ! \ i$

**shows**  $\text{interpret-form } f \ xs = \text{interpret-form } f \ ys$

(proof)

**lemma** *max-Var-floatariths-lessI:*  $i < \text{max-Var-floatarith } (fas \ ! \ j) \implies j < \text{length } fas \implies i < \text{max-Var-floatariths } fas$

(proof)

**lemma** *interpret-floatariths-max-Var-cong*:  
**assumes**  $\bigwedge i. i < \text{max-Var-floatariths } f \implies xs ! i = ys ! i$   
**shows**  $\text{interpret-floatariths } f \text{ } ys = \text{interpret-floatariths } f \text{ } xs$   
*<proof>*

**lemma** *max-Var-floatarithimage-Var[simp]*:  $\text{max-Var-floatarith } ' \text{Var } ' X = \text{Suc } ' X$  *<proof>*

**lemma** *max-Var-floatariths-map-Var[simp]*:  
 $\text{max-Var-floatariths } (\text{map } \text{Var } xs) = (\text{if } xs = [] \text{ then } 0 \text{ else } \text{Suc } (\text{linorder-class.Max } (\text{set } xs)))$   
*<proof>*

**lemma** *Max-atLeastLessThan-nat[simp]*:  $a < b \implies \text{linorder-class.Max } \{a..<b\} = b - 1$  **for**  $a \ b :: \text{nat}$   
*<proof>*

### 3.6 Derivatives

**lemma** *isDERIV-Power-iff*:  $\text{isDERIV } j \text{ } (\text{Power } fa \ n) \text{ } xs = (\text{if } n = 0 \text{ then } \text{True} \text{ else } \text{isDERIV } j \text{ } fa \text{ } xs)$   
*<proof>*

**lemma** *isDERIV-max-Var-floatarithI*:  
**assumes**  $\text{isDERIV } n \text{ } f \text{ } ys$   
**assumes**  $\bigwedge i. i < \text{max-Var-floatarith } f \implies xs ! i = ys ! i$   
**shows**  $\text{isDERIV } n \text{ } f \text{ } xs$   
*<proof>*

**definition** *isFDERIV* **where**  $\text{isFDERIV } n \text{ } xs \text{ } fas \text{ } vs \longleftrightarrow (\forall i < n. \forall j < n. \text{isDERIV } (xs ! i) \text{ } (fas ! j) \text{ } vs) \wedge \text{length } fas = n \wedge \text{length } xs = n$

**lemma** *isFDERIV-I*:  $(\bigwedge i \ j. i < n \implies j < n \implies \text{isDERIV } (xs ! i) \text{ } (fas ! j) \text{ } vs) \implies \text{length } fas = n \implies \text{length } xs = n \implies \text{isFDERIV } n \text{ } xs \text{ } fas \text{ } vs$   
*<proof>*

**lemma** *isFDERIV-isDERIV-D*:  $\text{isFDERIV } n \text{ } xs \text{ } fas \text{ } vs \implies i < n \implies j < n \implies \text{isDERIV } (xs ! i) \text{ } (fas ! j) \text{ } vs$   
*<proof>*

**lemma** *isFDERIV-lengthD*:  $\text{length } fas = n \ \text{length } xs = n \ \mathbf{if} \ \text{isFDERIV } n \text{ } xs \text{ } fas \text{ } vs$   
*<proof>*

**lemma** *isFDERIV-uptD*:  $\text{isFDERIV } n \text{ } [0..<n] \text{ } fas \text{ } vs \implies i < n \implies j < n \implies \text{isDERIV } i \text{ } (fas ! j) \text{ } vs$   
*<proof>*

**lemma** *isFDERIV-max-Var-congI*: *isFDERIV n xs fas ws*  
**if** *f*: *isFDERIV n xs fas vs* **and** *c*:  $(\bigwedge i. i < \text{max-Var-floatariths fas} \implies \text{vs ! } i = \text{ws ! } i)$   
 ⟨*proof*⟩

**lemma** *isFDERIV-max-Var-cong*: *isFDERIV n xs fas ws*  $\longleftrightarrow$  *isFDERIV n xs fas vs*  
**if** *c*:  $(\bigwedge i. i < \text{max-Var-floatariths fas} \implies \text{vs ! } i = \text{ws ! } i)$   
 ⟨*proof*⟩

**lemma** *isDERIV-max-VarI*:  
 $i \geq \text{max-Var-floatarith fa} \implies \text{isDERIV } j \text{ fa } xs \implies \text{isDERIV } i \text{ fa } xs$   
 ⟨*proof*⟩

**lemmas** *max-Var-floatarith-le-max-Var-floatariths-nthI* =  
*max-Var-floatarith-le-max-Var-floatariths-nth*[*THEN order-trans*]

**lemma**  
*isFDERIV-appendD1*:  
**assumes** *isFDERIV (J + K) [0..*J + K*] (es @ rs) xs*  
**assumes** *length es = J*  
**assumes** *length rs = K*  
**assumes** *max-Var-floatariths es*  $\leq$  *J*  
**shows** *isFDERIV J [0..*J*] (es) xs*  
 ⟨*proof*⟩

**lemma** *interpret-floatariths-Var[simp]*:  
*interpret-floatariths (map floatarith.Var xs) vs* = *(map (nth vs) xs)*  
 ⟨*proof*⟩

**lemma** *max-Var-floatariths-append[simp]*: *max-Var-floatariths (xs @ ys)* = *max (max-Var-floatariths xs) (max-Var-floatariths ys)*  
 ⟨*proof*⟩

**lemma** *map-nth-append-upt[simp]*:  
**assumes**  $a \geq \text{length } xs$   
**shows** *map (!) (xs @ ys) [a..*b*] = map (!) ys [a - length xs..*b* - length xs]*  
 ⟨*proof*⟩

**lemma** *map-nth-Cons-upt[simp]*:  
**assumes**  $a > 0$   
**shows** *map (!) (x # ys) [a..*b*] = map (!) ys [a - Suc 0..*b* - Suc 0]*  
 ⟨*proof*⟩

**lemma** *map-nth-eq-self[simp]*:  
**shows**  $\text{length } fas = l \implies (\text{map (!) } fas) [0..*l*] = fas$   
 ⟨*proof*⟩

**lemma**

*isFDERIV-appendI1:*

**assumes** *isFDERIV*  $J$   $[0..<J]$   $(es)$   $xs$

**assumes**  $\bigwedge i j. i < J + K \implies j < K \implies isDERIV\ i\ (rs\ !\ j)\ xs$

**assumes**  $length\ es = J$

**assumes**  $length\ rs = K$

**assumes** *max-Var-floatariths*  $es \leq J$

**shows** *isFDERIV*  $(J + K)$   $[0..<J + K]$   $(es\ @\ rs)$   $xs$

*<proof>*

**lemma** *matrix-matrix-mult-zero[simp]:*

$a\ **\ 0 = 0\ 0\ **\ a = 0$

*<proof>*

**lemma** *scaleR-blinfun-compose-left:*  $i\ *_R\ (A\ o_L\ B) = i\ *_R\ A\ o_L\ B$

**and** *scaleR-blinfun-compose-right:*  $i\ *_R\ (A\ o_L\ B) = A\ o_L\ i\ *_R\ B$

*<proof>*

**lemma**

*matrix-blinfun-compose:*

**fixes**  $A\ B::(real\ ^\ n) \Rightarrow_L\ (real\ ^\ n)$

**shows** *matrix*  $(A\ o_L\ B) = (matrix\ A)\ **\ (matrix\ B)$

*<proof>*

**lemma** *matrix-add-rdistrib:*  $((B + C)\ **\ A) = (B\ **\ A) + (C\ **\ A)$

*<proof>*

**lemma** *matrix-scaleR-right:*  $r\ *_R\ (a::'a::real-algebra-1^\ n^\ m)\ **\ b = r\ *_R\ (a\ **\ b)$

*<proof>*

**lemma** *matrix-scaleR-left:*  $(a::'a::real-algebra-1^\ n^\ m)\ **\ r\ *_R\ b = r\ *_R\ (a\ **\ b)$

*<proof>*

**lemma** *bounded-bilinear-matrix-matrix-mult[bounded-bilinear]:*

*bounded-bilinear*  $((**))::$

$(a::\{euclidean-space,\ real-normed-algebra-1\}^\ n^\ m) \Rightarrow$

$(a::\{euclidean-space,\ real-normed-algebra-1\}^\ p^\ n) \Rightarrow$

$(a::\{euclidean-space,\ real-normed-algebra-1\}^\ p^\ m))$

*<proof>*

**lemma** *norm-axis:*  $norm\ (axis\ ia\ 1::'a::\{real-normed-algebra-1\}^\ n) = 1$

*<proof>*

**lemma** *abs-vec-nth-blinfun-apply-lemma:*

**fixes**  $x::(real\ ^\ n) \Rightarrow_L\ (real\ ^\ m)$

**shows**  $abs (vec\text{-}nth (blinfun\text{-}apply\ x\ (axis\ ia\ 1))\ i) \leq norm\ x$   
 ⟨proof⟩

**lemma** *bounded-linear-matrix-blinfun-apply*: *bounded-linear*  $(\lambda x::(real^{\wedge}n) \Rightarrow_L (real^{\wedge}m).$   
*matrix*  $(blinfun\text{-}apply\ x))$   
 ⟨proof⟩

**lemma** *matrix-has-derivative*:  
**shows**  $((\lambda x::(real^{\wedge}n) \Rightarrow_L (real^{\wedge}n). matrix\ (blinfun\text{-}apply\ x))\ has\text{-}derivative\ (\lambda h.$   
*matrix*  $(blinfun\text{-}apply\ h)))\ (at\ x)$   
 ⟨proof⟩

**lemma**  
*matrix-comp-has-derivative*[*derivative-intros*]:  
**fixes**  $f::'a::real\text{-}normed\text{-}vector \Rightarrow ((real^{\wedge}n) \Rightarrow_L (real^{\wedge}n))$   
**assumes**  $(f\ has\text{-}derivative\ f')\ (at\ x\ within\ S)$   
**shows**  $((\lambda x. matrix\ (blinfun\text{-}apply\ (f\ x)))\ has\text{-}derivative\ (\lambda x. matrix\ (f'\ x)))\ (at\ x\ within\ S)$   
 ⟨proof⟩

**fun** *inner-floatariths* **where**  
*inner-floatariths*  $[] = Num\ 0$   
 $| inner\text{-}floatariths\ - [] = Num\ 0$   
 $| inner\text{-}floatariths\ (x\#\ xs)\ (y\#\ ys) = Add\ (Mult\ x\ y)\ (inner\text{-}floatariths\ xs\ ys)$

**lemma** *interpret-floatarith-inner-eq*:  
**assumes**  $length\ xs = length\ ys$   
**shows**  $interpret\text{-}floatarith\ (inner\text{-}floatariths\ xs\ ys)\ vs =$   
 $(\sum\ i < length\ ys. (interpret\text{-}floatariths\ xs\ vs\ !\ i) * (interpret\text{-}floatariths\ ys\ vs\ !\ i))$   
 ⟨proof⟩

**lemma**  
*interpret-floatarith-inner-floatariths*:  
**assumes**  $length\ xs = DIM('a::executable\text{-}euclidean\text{-}space)$   
**assumes**  $length\ ys = DIM('a)$   
**assumes**  $eucl\text{-}of\text{-}list\ (interpret\text{-}floatariths\ xs\ vs) = (x::'a)$   
**assumes**  $eucl\text{-}of\text{-}list\ (interpret\text{-}floatariths\ ys\ vs) = y$   
**shows**  $interpret\text{-}floatarith\ (inner\text{-}floatariths\ xs\ ys)\ vs = x \cdot y$   
 ⟨proof⟩

**lemma** *max-Var-floatarith-inner-floatariths*[*simp*]:  
**assumes**  $length\ f = length\ g$   
**shows**  $max\text{-}Var\text{-}floatarith\ (inner\text{-}floatariths\ f\ g) = max\ (max\text{-}Var\text{-}floatariths\ f)$   
 $(max\text{-}Var\text{-}floatariths\ g)$   
 ⟨proof⟩

**definition** *FDERIV-floatarith* **where**  
*FDERIV-floatarith*  $fa\ xs\ d = inner\text{-}floatariths\ (map\ (\lambda x. fold\text{-}const\text{-}fa\ (DERIV\text{-}floatarith$

$x \text{ fa}) \text{ xs} \text{ d}$

— TODO: specialize to  $FDERIV\text{-floatarith } fa [0..<n] [m..<m + n]$  and do the rest with  $subst\text{-floatarith}$ ? TODO: introduce approximation on type  $((real, 'i) \text{ vec}, 'j) \text{ vec}$  and use  $jacobian$ ?

**lemma**  $interpret\text{-floatariths-map}$ :  $interpret\text{-floatariths } (map \ f \ xs) \ vs = map \ (\lambda x. interpret\text{-floatarith } (f \ x) \ vs) \ xs$   
 $\langle proof \rangle$

**lemma**  $max\text{-Var}\text{-floatarith-}DERIV\text{-floatarith}$ :  
 $max\text{-Var}\text{-floatarith } (DERIV\text{-floatarith } x \ fa) \leq max\text{-Var}\text{-floatarith } fa$   
 $\langle proof \rangle$

**lemma**  $max\text{-Var}\text{-floatarith-F}DERIV\text{-floatarith}$ :  
 $length \ xs = length \ d \implies$   
 $max\text{-Var}\text{-floatarith } (FDERIV\text{-floatarith } fa \ xs \ d) \leq max \ (max\text{-Var}\text{-floatarith } fa)$   
 $(max\text{-Var}\text{-floatariths } d)$   
 $\langle proof \rangle$

**definition**  $FDERIV\text{-floatariths where}$   
 $FDERIV\text{-floatariths } fas \ xs \ d = map \ (\lambda fa. FDERIV\text{-floatarith } fa \ xs \ d) \ fas$

**lemma**  $max\text{-Var}\text{-floatarith-F}DERIV\text{-floatariths}$ :  
 $length \ xs = length \ d \implies max\text{-Var}\text{-floatariths } (FDERIV\text{-floatariths } fa \ xs \ d) \leq max$   
 $(max\text{-Var}\text{-floatariths } fa) \ (max\text{-Var}\text{-floatariths } d)$   
 $\langle proof \rangle$

**lemma**  $length\text{-F}DERIV\text{-floatariths}[simp]$ :  
 $length \ (FDERIV\text{-floatariths } fas \ xs \ ds) = length \ fas$   
 $\langle proof \rangle$

**lemma**  $FDERIV\text{-floatariths-nth}[simp]$ :  
 $i < length \ fas \implies FDERIV\text{-floatariths } fas \ xs \ ds ! i = FDERIV\text{-floatarith } (fas !$   
 $i) \ xs \ ds$   
 $\langle proof \rangle$

**definition**  $FDERIV\text{-n}\text{-floatariths } fas \ xs \ ds \ n = ((\lambda x. FDERIV\text{-floatariths } x \ xs$   
 $ds) \ \overset{\sim}{n}) \ fas$

**lemma**  $FDERIV\text{-n}\text{-floatariths-Suc}[simp]$ :  
 $FDERIV\text{-n}\text{-floatariths } fa \ xs \ ds \ 0 = fa$   
 $FDERIV\text{-n}\text{-floatariths } fa \ xs \ ds \ (Suc \ n) = FDERIV\text{-floatariths } (FDERIV\text{-n}\text{-floatariths}$   
 $fa \ xs \ ds \ n) \ xs \ ds$   
 $\langle proof \rangle$

**lemma**  $length\text{-F}DERIV\text{-n}\text{-floatariths}[simp]$ :  $length \ (FDERIV\text{-n}\text{-floatariths } fa \ xs \ ds$   
 $n) = length \ fa$   
 $\langle proof \rangle$

**lemma** *max-Var-floatarith-FDERIV-n-floatariths:*

*length xs = length d  $\implies$  max-Var-floatariths (FDERIV-n-floatariths fa xs d n)*  
 $\leq$  *max (max-Var-floatariths fa) (max-Var-floatariths d)*  
 ⟨proof⟩

**lemma** *interpret-floatarith-FDERIV-floatarith-cong:*

**assumes** *rq:  $\bigwedge i. i < \text{max-Var-floatarith } f \implies rs ! i = qs ! i$*   
**assumes** [*simp*]: *length ds = length xs length es = length xs*  
**assumes** *interpret-floatariths ds qs = interpret-floatariths es rs*  
**shows** *interpret-floatarith (FDERIV-floatarith f xs ds) qs =*  
*interpret-floatarith (FDERIV-floatarith f xs es) rs*  
 ⟨proof⟩

**theorem**

*matrix-vector-mult-eq-list-of-eucl-nth:*  
 $(M::\text{real}^{\wedge n}::\text{enum}^{\wedge m}::\text{enum}) * v =$   
 $(\sum i < \text{CARD}('m).$   
 $(\sum j < \text{CARD}('n). \text{list-of-eucl } M ! (i * \text{CARD}('n) + j) * \text{list-of-eucl } v ! j) *_{\mathbb{R}}$   
*Basis-list ! i)*  
 ⟨proof⟩

**definition** *mmult-fa l m n AS BS =*

*concat (map ( $\lambda i. \text{map } (\lambda k. \text{inner-floatariths}$*   
 $(\text{map } (\lambda j. \text{AS} ! (i * m + j)) [0..<m]) (\text{map } (\lambda j. \text{BS} ! (j * n + k)) [0..<m]))$   
 $[0..<n]) [0..<l])$

**lemma** *length-mmult-fa[simp]: length (mmult-fa l m n AS BS) = l \* n*  
 ⟨proof⟩

**lemma** *einterpret-mmult-fa:*

**assumes** [*simp*]: *Dn = CARD('n::enum) Dm = CARD('m::enum) Dl = CARD('l::enum)*  
*length A = CARD('l)\*CARD('m) length B = CARD('m)\*CARD('n)*  
**shows** *einterpret (mmult-fa Dl Dm Dn A B) vs = (einterpret A vs::((real,*  
 $'m::\text{enum}) \text{vec}, 'l) \text{vec}) ** (einterpret B vs::((\text{real}, 'n::\text{enum}) \text{vec}, 'm) \text{vec})$   
 ⟨proof⟩

**lemma** *max-Var-floatariths-mmult-fa:*

**assumes** [*simp*]: *length A = D \* E length B = E \* F*  
**shows** *max-Var-floatariths (mmult-fa D E F A B)  $\leq$  max (max-Var-floatariths*  
*A) (max-Var-floatariths B)*  
 ⟨proof⟩

**lemma** *isDERIV-inner-iff:*

**assumes** *length xs = length ys*  
**shows** *isDERIV i (inner-floatariths xs ys) vs  $\longleftrightarrow$*   
 $(\forall k < \text{length } xs. \text{isDERIV } i (xs ! k) vs) \wedge (\forall k < \text{length } ys. \text{isDERIV } i (ys ! k)$   
 $vs)$   
 ⟨proof⟩

**lemma** *isDERIV-Power*:  $isDERIV\ x\ (fa)\ vs \Longrightarrow isDERIV\ x\ (fa\ \widehat{e}\ n)\ vs$   
 ⟨proof⟩

**lemma** *isDERIV-mmult-fa-nth*:

**assumes**  $\bigwedge j. j < D * E \Longrightarrow isDERIV\ i\ (A\ !\ j)\ xs$   
**assumes**  $\bigwedge j. j < E * F \Longrightarrow isDERIV\ i\ (B\ !\ j)\ xs$   
**assumes** [*simp*]:  $length\ A = D * E\ length\ B = E * F\ j < D * F$   
**shows**  $isDERIV\ i\ (mmult-fa\ D\ E\ F\ A\ B\ !\ j)\ xs$   
 ⟨proof⟩

**definition** *mvmult-fa n m AS B =*

*map* ( $\lambda i. inner-floatariths\ (map\ (\lambda j. AS\ !\ (i * m + j))\ [0..<m])\ (map\ (\lambda j. B\ !\ j)\ [0..<m]))\ [0..<n]$ )

**lemma** *einterpret-mvmult-fa*:

**assumes** [*simp*]:  $Dn = CARD('n::enum)\ Dm = CARD('m::enum)$   
 $length\ A = CARD('n)*CARD('m)\ length\ B = CARD('m)$   
**shows**  $einterpret\ (mvmult-fa\ Dn\ Dm\ A\ B)\ vs = (einterpret\ A\ vs::((real,\ 'm::enum)\ vec,\ 'n)\ vec) * v\ (einterpret\ B\ vs::(real,\ 'm)\ vec)$   
 ⟨proof⟩

**lemma** *max-Var-floatariths-mvult-fa*:

**assumes** [*simp*]:  $length\ A = D * E\ length\ B = E$   
**shows**  $max-Var-floatariths\ (mvmult-fa\ D\ E\ A\ B) \leq max\ (max-Var-floatariths\ A)\ (max-Var-floatariths\ B)$   
 ⟨proof⟩

**lemma** *isDERIV-mvmult-fa-nth*:

**assumes**  $\bigwedge j. j < D * E \Longrightarrow isDERIV\ i\ (A\ !\ j)\ xs$   
**assumes**  $\bigwedge j. j < E \Longrightarrow isDERIV\ i\ (B\ !\ j)\ xs$   
**assumes** [*simp*]:  $length\ A = D * E\ length\ B = E\ j < D$   
**shows**  $isDERIV\ i\ (mvmult-fa\ D\ E\ A\ B\ !\ j)\ xs$   
 ⟨proof⟩

**lemma** *max-Var-floatariths-mapI*:

**assumes**  $\bigwedge x. x \in set\ xs \Longrightarrow max-Var-floatarith\ (f\ x) \leq m$   
**shows**  $max-Var-floatariths\ (map\ f\ xs) \leq m$   
 ⟨proof⟩

**lemma**

*max-Var-floatariths-list-updateI*:  
**assumes**  $max-Var-floatariths\ xs \leq m$   
**assumes**  $max-Var-floatarith\ v \leq m$   
**assumes**  $i < length\ xs$   
**shows**  $max-Var-floatariths\ (xs[i := v]) \leq m$   
 ⟨proof⟩

**lemma**

*max-Var-floatariths-replicateI*:  
**assumes** *max-Var-floatarith*  $v \leq m$   
**shows** *max-Var-floatariths* (*replicate*  $n$   $v$ )  $\leq m$   
 $\langle$ *proof* $\rangle$

**definition** *FDERIV-n-floatarith*  $fa$   $xs$   $ds$   $n = ((\lambda x. FDERIV-floatarith$   $x$   $xs$   $ds) \sim^n)$   
 $fa$

**lemma** *FDERIV-n-floatariths-nth*:  $i < \text{length } fas \implies FDERIV-n-floatariths$   $fas$   
 $xs$   $ds$   $n$   $! i = FDERIV-n-floatarith$  ( $fas$   $! i$ )  $xs$   $ds$   $n$   
 $\langle$ *proof* $\rangle$

**lemma** *einterpret-fold-const-fa*[*simp*]:  
 $(einterpret$  ( $\text{map } (\lambda i. \text{fold-const-fa } (fa$   $i))$   $xs$ )  $vs::'a::executable-euclidean-space$ )  
 $=$   
 $einterpret$  ( $\text{map } fa$   $xs$ )  $vs$  **if**  $\text{length } xs = DIM('a)$   
 $\langle$ *proof* $\rangle$

**lemma** *einterpret-plus*[*simp*]:  
**shows**  $(einterpret$  ( $\text{map } (\lambda i. fa1$   $i + fa2$   $i)$   $[0..<DIM('a)])$   $vs::'a$ )  $=$   
 $einterpret$  ( $\text{map } fa1$   $[0..<DIM('a::executable-euclidean-space)])$   $vs + einterpret$   
 $(\text{map } fa2$   $[0..<DIM('a)])$   $vs$   
 $\langle$ *proof* $\rangle$

**lemma** *einterpret-uminus*[*simp*]:  
**shows**  $(einterpret$  ( $\text{map } (\lambda i. -$   $fa1$   $i)$   $[0..<DIM('a)])$   $vs::'a::executable-euclidean-space$ )  
 $=$   
 $- einterpret$  ( $\text{map } fa1$   $[0..<DIM('a)])$   $vs$   
 $\langle$ *proof* $\rangle$

**lemma** *diff-floatarith-conv-add-uminus*:  $a - b = a + - b$  **for**  $a$   $b::floatarith$   
 $\langle$ *proof* $\rangle$

**lemma** *einterpret-minus*[*simp*]:  
**shows**  $(einterpret$  ( $\text{map } (\lambda i. fa1$   $i - fa2$   $i)$   $[0..<DIM('a)])$   $vs::'a::executable-euclidean-space$ )  
 $=$   
 $einterpret$  ( $\text{map } fa1$   $[0..<DIM('a)])$   $vs - einterpret$  ( $\text{map } fa2$   $[0..<DIM('a)])$   
 $vs$   
 $\langle$ *proof* $\rangle$

**lemma** *einterpret-scaleR*[*simp*]:  
**shows**  $(einterpret$  ( $\text{map } (\lambda i. fa1$   $* fa2$   $i)$   $[0..<DIM('a)])$   $vs::'a::executable-euclidean-space$ )  
 $=$   
 $interpret-floatarith$  ( $fa1$ )  $vs *_R einterpret$  ( $\text{map } fa2$   $[0..<DIM('a)])$   $vs$   
 $\langle$ *proof* $\rangle$

**lemma** *einterpret-nth*[*simp*]:  
**assumes** [*simp*]:  $\text{length } xs = DIM('a)$   
**shows**  $(einterpret$  ( $\text{map } (!)$   $xs$ )  $[0..<DIM('a)])$   $vs::'a::executable-euclidean-space$ )

= *einterpret xs vs*  
 ⟨*proof*⟩

**type-synonym** 'n rvec = (real, 'n) vec

**lemma** *length-mvmult-fa[simp]*: *length (mvmult-fa D E xs ys) = D*  
 ⟨*proof*⟩

**lemma** *interpret-mvmult-nth*:

**assumes** *D = CARD('n::enum)*

**assumes** *E = CARD('m::enum)*

**assumes** *length xs = D \* E*

**assumes** *length ys = E*

**assumes** *n < CARD('n)*

**shows** *interpret-floatarith (mvmult-fa D E xs ys ! n) vs =*

*((einterpret xs vs::(real, 'm) vec, 'n) vec) \*v einterpret ys vs) · (Basis-list ! n)*

⟨*proof*⟩

**lemmas** [*simp del*] = *fold-const-fa.simps*

**lemma** *take-eq-map-nth*: *n < length xs ⇒ take n xs = map (!) xs [0..*n*]*  
 ⟨*proof*⟩

**lemmas** [*simp del*] = *upt-rec-numeral*

**lemmas** *map-nth-eq-take = take-eq-map-nth[symmetric]*

### 3.7 Definition of Approximating Function using Affine Arithmetic

**lemma** *interpret-Floatreal*: *interpret-floatarith (floatarith.Num f) vs = (real-of-float f)*

⟨*proof*⟩

⟨*ML*⟩

**schematic-goal** *reify-example*:

*[xs!i \* xs!j, xs!i + xs!j powr (sin (xs!0)), xs!k + (2 / 3 \* xs!i \* xs!j)] = interpret-floatariths ?fas xs*

⟨*proof*⟩

⟨*ML*⟩

**lemma** *eucl-of-list-interpret-floatariths-cong*:

**fixes** *y::'a::executable-euclidean-space*

**assumes**  $\bigwedge b. b \in \text{Basis} \Rightarrow \text{interpret-floatarith (fa (index Basis-list b)) vs} = y$

$\cdot b$

**assumes** *length xs = DIM('a)*

**shows** *eucl-of-list (interpret-floatariths (map fa [0..*DIM('a)*]) vs) = y*

*<proof>*

**lemma** *interpret-floatariths-fold-const-fa*[*simp*]:

*interpret-floatariths (map fold-const-fa ds) = interpret-floatariths ds*

*<proof>*

**fun** *subst-floatarith* **where**

*subst-floatarith s (Add a b) = Add (subst-floatarith s a) (subst-floatarith s b) |*  
*subst-floatarith s (Mult a b) = Mult (subst-floatarith s a) (subst-floatarith s b) |*  
*subst-floatarith s (Minus a) = Minus (subst-floatarith s a) |*  
*subst-floatarith s (Inverse a) = Inverse (subst-floatarith s a) |*  
*subst-floatarith s (Cos a) = Cos (subst-floatarith s a) |*  
*subst-floatarith s (Arctan a) = Arctan (subst-floatarith s a) |*  
*subst-floatarith s (Min a b) = Min (subst-floatarith s a) (subst-floatarith s b) |*  
*subst-floatarith s (Max a b) = Max (subst-floatarith s a) (subst-floatarith s b) |*  
*subst-floatarith s (Abs a) = Abs (subst-floatarith s a) |*  
*subst-floatarith s Pi = Pi |*  
*subst-floatarith s (Sqrt a) = Sqrt (subst-floatarith s a) |*  
*subst-floatarith s (Exp a) = Exp (subst-floatarith s a) |*  
*subst-floatarith s (Powr a b) = Powr (subst-floatarith s a) (subst-floatarith s b) |*  
*subst-floatarith s (Ln a) = Ln (subst-floatarith s a) |*  
*subst-floatarith s (Power a i) = Power (subst-floatarith s a) i |*  
*subst-floatarith s (Floor a) = Floor (subst-floatarith s a) |*  
*subst-floatarith s (Num f) = Num f |*  
*subst-floatarith s (Var n) = s n*

**lemma** *interpret-floatarith-subst-floatarith*:

**assumes** *max-Var-floatarith fa ≤ D*

**shows** *interpret-floatarith (subst-floatarith s fa) vs =*

*interpret-floatarith fa (map (λi. interpret-floatarith (s i) vs) [0..<D])*

*<proof>*

**lemma** *max-Var-floatarith-subst-floatarith-le*[*THEN order-trans*]:

**assumes** *length xs ≥ max-Var-floatarith fa*

**shows** *max-Var-floatarith (subst-floatarith (!) xs) fa ≤ max-Var-floatariths xs*

*<proof>*

**lemma** *max-Var-floatariths-subst-floatarith-le*[*THEN order-trans*]:

**assumes** *length xs ≥ max-Var-floatariths fas*

**shows** *max-Var-floatariths (map (subst-floatarith (!) xs) fas) ≤ max-Var-floatariths xs*

*<proof>*

*<proof>*

**fun** *continuous-on-floatarith* :: *floatarith ⇒ bool* **where**

$\text{continuous-on-floatarith (Add } a \ b) = (\text{continuous-on-floatarith } a \wedge \text{continuous-on-floatarith } b) \mid$   
 $\text{continuous-on-floatarith (Mult } a \ b) = (\text{continuous-on-floatarith } a \wedge \text{continuous-on-floatarith } b) \mid$   
 $\text{continuous-on-floatarith (Minus } a) = \text{continuous-on-floatarith } a \mid$   
 $\text{continuous-on-floatarith (Inverse } a) = \text{False} \mid$   
 $\text{continuous-on-floatarith (Cos } a) = \text{continuous-on-floatarith } a \mid$   
 $\text{continuous-on-floatarith (Arctan } a) = \text{continuous-on-floatarith } a \mid$   
 $\text{continuous-on-floatarith (Min } a \ b) = (\text{continuous-on-floatarith } a \wedge \text{continuous-on-floatarith } b) \mid$   
 $\text{continuous-on-floatarith (Max } a \ b) = (\text{continuous-on-floatarith } a \wedge \text{continuous-on-floatarith } b) \mid$   
 $\text{continuous-on-floatarith (Abs } a) = \text{continuous-on-floatarith } a \mid$   
 $\text{continuous-on-floatarith Pi} = \text{True} \mid$   
 $\text{continuous-on-floatarith (Sqrt } a) = \text{False} \mid$   
 $\text{continuous-on-floatarith (Exp } a) = \text{continuous-on-floatarith } a \mid$   
 $\text{continuous-on-floatarith (Powr } a \ b) = \text{False} \mid$   
 $\text{continuous-on-floatarith (Ln } a) = \text{False} \mid$   
 $\text{continuous-on-floatarith (Floor } a) = \text{False} \mid$   
 $\text{continuous-on-floatarith (Power } a \ n) = (\text{if } n = 0 \text{ then True else continuous-on-floatarith } a) \mid$   
 $\text{continuous-on-floatarith (Num } f) = \text{True} \mid$   
 $\text{continuous-on-floatarith (Var } n) = \text{True}$

**definition**  $\text{Maxs}_e \ xs = \text{fold } (\lambda a \ b. \text{floatarith.Max } a \ b) \ xs$

**definition**  $\text{norm2}_e \ n = \text{Maxs}_e \ (\text{map } (\lambda j. \text{Norm } (\text{map } (\lambda i. \text{Var } (\text{Suc } j * n + i)) [0..<n]))) [0..<n]) \ (\text{Num } 0)$

**definition**  $N_r \ l = \text{Num } (\text{float-of } l)$

**lemma** *interpret-floatarith-Norm*:

$\text{interpret-floatarith (Norm } xs) \ vs = \text{L2-set } (\lambda i. \text{interpret-floatarith } (xs \ ! \ i) \ vs)$   
 $\{0..<\text{length } xs\}$   
 $\langle \text{proof} \rangle$

**lemma** *interpret-floatarith-Nr[simp]*:  $\text{interpret-floatarith } (N_r \ U) \ vs = \text{real-of-float } (\text{float-of } U)$

$\langle \text{proof} \rangle$

**fun** *list-updates where*

$\text{list-updates } [] \ - \ xs = xs$   
 $\mid \text{list-updates } - \ [] \ xs = xs$   
 $\mid \text{list-updates } (i\#is) \ (u\#us) \ xs = \text{list-updates } is \ us \ (xs[i:=u])$

**lemma** *list-updates-nth-notmem*:

**assumes**  $\text{length } xs = \text{length } ys$   
**assumes**  $i \notin \text{set } xs$

**shows**  $list\_updates\ xs\ ys\ vs\ !\ i = vs\ !\ i$   
 ⟨proof⟩

**lemma**  $list\_updates\_nth\_less$ :

**assumes**  $length\ xs = length\ ys$   $distinct\ xs$

**assumes**  $i < length\ vs$

**shows**  $list\_updates\ xs\ ys\ vs\ !\ i = (if\ i \in set\ xs\ then\ ys\ !\ (index\ xs\ i)\ else\ vs\ !\ i)$

⟨proof⟩

**lemma**  $length\_list\_updates[simp]$ :  $length\ (list\_updates\ xs\ ys\ vs) = length\ vs$

⟨proof⟩

**lemma**  $list\_updates\_nth\_ge[simp]$ :

$x \geq length\ vs \implies length\ xs = length\ ys \implies list\_updates\ xs\ ys\ vs\ !\ x = vs\ !\ x$

⟨proof⟩

**lemma**

$list\_updates\_nth$ :

**assumes**  $[simp]$ :  $length\ xs = length\ ys$   $distinct\ xs$

**shows**  $list\_updates\ xs\ ys\ vs\ !\ i = (if\ i < length\ vs \wedge i \in set\ xs\ then\ ys\ !\ index\ xs\ i\ else\ vs\ !\ i)$

⟨proof⟩

**lemma**  $list\_of\_eucl\_coord\_update$ :

**assumes**  $[simp]$ :  $length\ xs = DIM('a::executable\_euclidean\_space)$

**assumes**  $[simp]$ :  $distinct\ xs$

**assumes**  $[simp]$ :  $i \in Basis$

**assumes**  $[simp]$ :  $\bigwedge n. n \in set\ xs \implies n < length\ vs$

**shows**  $list\_updates\ xs\ (list\_of\_eucl\ (x + (p - x \cdot i) *_{\mathbb{R}} i::'a))\ vs =$   
 $(list\_updates\ xs\ (list\_of\_eucl\ x)\ vs)[xs\ !\ index\ Basis\_list\ i := p]$

⟨proof⟩

**definition**  $eucl\_of\_env\ is\ vs = eucl\_of\_list\ (map\ (nth\ vs)\ is)$

**lemma**  $list\_updates\_list\_of\_eucl\_of\_env[simp]$ :

**assumes**  $[simp]$ :  $length\ xs = DIM('a::executable\_euclidean\_space)$   $distinct\ xs$

**shows**  $list\_updates\ xs\ (list\_of\_eucl\ (eucl\_of\_env\ xs\ vs::'a))\ vs = vs$

⟨proof⟩

**lemma**  $nth\_nth\_eucl\_of\_env\_inner$ :

$b \in Basis \implies length\ is = DIM('a) \implies vs\ !\ (is\ !\ index\ Basis\_list\ b) = eucl\_of\_env\ is\ vs \cdot b$

**for**  $b::'a::executable\_euclidean\_space$

⟨proof⟩

**lemma**  $list\_updates\_idem[simp]$ :

**assumes**  $(\bigwedge i. i \in set\ X0 \implies i < length\ vs)$

**shows**  $(list\_updates\ X0\ (map\ (!)\ vs)\ X0)\ vs = vs$

⟨proof⟩

**lemma** *pairwise-orthogonal-Basis*[*intro, simp*]: *pairwise orthogonal Basis*  
 ⟨*proof*⟩

**primrec** *freshs-floatarith* **where**

$\text{freshs-floatarith } (\text{Var } y) \ x \longleftrightarrow (y \notin \text{set } x)$   
 $\text{freshs-floatarith } (\text{Num } a) \ x \longleftrightarrow \text{True}$   
 $\text{freshs-floatarith } \text{Pi } x \longleftrightarrow \text{True}$   
 $\text{freshs-floatarith } (\text{Cos } a) \ x \longleftrightarrow \text{freshs-floatarith } a \ x$   
 $\text{freshs-floatarith } (\text{Abs } a) \ x \longleftrightarrow \text{freshs-floatarith } a \ x$   
 $\text{freshs-floatarith } (\text{Arctan } a) \ x \longleftrightarrow \text{freshs-floatarith } a \ x$   
 $\text{freshs-floatarith } (\text{Sqrt } a) \ x \longleftrightarrow \text{freshs-floatarith } a \ x$   
 $\text{freshs-floatarith } (\text{Exp } a) \ x \longleftrightarrow \text{freshs-floatarith } a \ x$   
 $\text{freshs-floatarith } (\text{Floor } a) \ x \longleftrightarrow \text{freshs-floatarith } a \ x$   
 $\text{freshs-floatarith } (\text{Power } a \ n) \ x \longleftrightarrow \text{freshs-floatarith } a \ x$   
 $\text{freshs-floatarith } (\text{Minus } a) \ x \longleftrightarrow \text{freshs-floatarith } a \ x$   
 $\text{freshs-floatarith } (\text{Ln } a) \ x \longleftrightarrow \text{freshs-floatarith } a \ x$   
 $\text{freshs-floatarith } (\text{Inverse } a) \ x \longleftrightarrow \text{freshs-floatarith } a \ x$   
 $\text{freshs-floatarith } (\text{Add } a \ b) \ x \longleftrightarrow \text{freshs-floatarith } a \ x \wedge \text{freshs-floatarith } b \ x$   
 $\text{freshs-floatarith } (\text{Mult } a \ b) \ x \longleftrightarrow \text{freshs-floatarith } a \ x \wedge \text{freshs-floatarith } b \ x$   
 $\text{freshs-floatarith } (\text{floatarith.Max } a \ b) \ x \longleftrightarrow \text{freshs-floatarith } a \ x \wedge \text{freshs-floatarith } b \ x$   
 $\text{freshs-floatarith } (\text{floatarith.Min } a \ b) \ x \longleftrightarrow \text{freshs-floatarith } a \ x \wedge \text{freshs-floatarith } b \ x$   
 $\text{freshs-floatarith } (\text{Powr } a \ b) \ x \longleftrightarrow \text{freshs-floatarith } a \ x \wedge \text{freshs-floatarith } b \ x$

**lemma** *freshs-floatarith*[*simp*]:

**assumes** *freshs-floatarith*  $fa \ ds \ \text{length } ds = \text{length } xs$   
**shows** *interpret-floatarith*  $fa \ (\text{list-updates } ds \ xs \ vs) = \text{interpret-floatarith } fa \ vs$   
 ⟨*proof*⟩

**lemma** *freshs-floatarith-max-Var-floatarithI*:

**assumes**  $\bigwedge x. x \in \text{set } xs \implies \text{max-Var-floatarith } f \leq x$   
**shows** *freshs-floatarith*  $f \ xs$   
 ⟨*proof*⟩

**definition** *freshs-floatariths*  $fas \ xs = (\forall fa \in \text{set } fas. \text{freshs-floatarith } fa \ xs)$

**lemma** *freshs-floatariths-max-Var-floatarithsI*:

**assumes**  $\bigwedge x. x \in \text{set } xs \implies \text{max-Var-floatariths } f \leq x$   
**shows** *freshs-floatariths*  $f \ xs$   
 ⟨*proof*⟩

**lemma** *freshs-floatariths-freshs-floatarithI*:

**assumes**  $\bigwedge fa. fa \in \text{set } fas \implies \text{freshs-floatarith } fa \ xs$   
**shows** *freshs-floatariths*  $fas \ xs$   
 ⟨*proof*⟩

**lemma** *fresh-floatariths-fresh-floatarithI*:

**assumes** *freshs-floatariths fas xs*

**assumes**  $fa \in \text{set } fas$

**shows** *freshs-floatarith fa xs*

*<proof>*

**lemma** *fresh-floatariths-fresh-floatarith[simp]*:

*fresh-floatariths (fas) i  $\implies$  fa  $\in$  set fas  $\implies$  fresh-floatarith fa i*

*<proof>*

**lemma** *interpret-floatariths-fresh-cong*:

**assumes**  $\bigwedge i. \neg \text{fresh-floatariths } f \ i \implies xs \ ! \ i = ys \ ! \ i$

**shows** *interpret-floatariths f ys = interpret-floatariths f xs*

*<proof>*

**fun** *subterms* :: *floatarith  $\Rightarrow$  floatarith set* **where**

*subterms (Add a b) = insert (Add a b) (subterms a  $\cup$  subterms b) |*

*subterms (Mult a b) = insert (Mult a b) (subterms a  $\cup$  subterms b) |*

*subterms (Min a b) = insert (Min a b) (subterms a  $\cup$  subterms b) |*

*subterms (floatarith.Max a b) = insert (floatarith.Max a b) (subterms a  $\cup$  subterms b) |*

*subterms (Powr a b) = insert (Powr a b) (subterms a  $\cup$  subterms b) |*

*subterms (Inverse a) = insert (Inverse a) (subterms a) |*

*subterms (Cos a) = insert (Cos a) (subterms a) |*

*subterms (Arctan a) = insert (Arctan a) (subterms a) |*

*subterms (Abs a) = insert (Abs a) (subterms a) |*

*subterms (Sqrt a) = insert (Sqrt a) (subterms a) |*

*subterms (Exp a) = insert (Exp a) (subterms a) |*

*subterms (Ln a) = insert (Ln a) (subterms a) |*

*subterms (Power a n) = insert (Power a n) (subterms a) |*

*subterms (Floor a) = insert (Floor a) (subterms a) |*

*subterms (Minus a) = insert (Minus a) (subterms a) |*

*subterms Pi = {Pi} |*

*subterms (Var v) = {Var v} |*

*subterms (Num n) = {Num n}*

**lemma** *subterms-self[simp]*:  $fa2 \in \text{subterms } fa2$

*<proof>*

**lemma** *interpret-floatarith-FDERIV-floatarith-eucl-of-env*:— TODO: cleanup, reduce to DERIV?!

**assumes**  $iD: \bigwedge i. i < DIM('a) \implies isDERIV (xs \ ! \ i) \ fa \ vs$

**assumes** *ds-fresh: freshs-floatarith fa ds*

**assumes** [simp]:  $length \ xs = DIM ('a) \ length \ ds = DIM ('a)$

$\bigwedge i. i \in \text{set } xs \implies i < \text{length } vs \ \text{distinct } xs$

$\bigwedge i. i \in \text{set } ds \implies i < \text{length } vs \ \text{distinct } ds$

**shows**  $((\lambda x::'a::\text{executable-euclidean-space}.$

$(\text{interpret-floatarith } fa \ (\text{list-updates } xs \ (\text{list-of-eucl } x) \ vs))) \ \text{has-derivative}$

$(\lambda d. \text{interpret-floatarith } (FDERIV-floatarith \ fa \ xs \ (\text{map } \text{Var } ds)) \ (\text{list-updates}$

*ds* (*list-of-eucl d vs*) )  
 ) (*at* (*eucl-of-env xs vs*))  
 ⟨*proof*⟩

**lemma** *interpret-floatarith-FDERIV-floatarith-append*:

**assumes** *iD*:  $\bigwedge i j. i < DIM('a) \implies isDERIV\ i\ (fa)\ (list-of-eucl\ x\ @\ params)$   
**assumes** *m*: *max-Var-floatarith fa*  $\leq DIM('a) + length\ params$   
**shows** ( $\lambda x::'a::executable-euclidean-space.$   
*interpret-floatarith fa* (*list-of-eucl x @ params*) *has-derivative*  
 $(\lambda d. interpret-floatarith$   
 $(FDERIV-floatarith\ fa\ [0..<DIM('a)]\ (map\ Var\ [length\ params + DIM('a)..<length$   
 $params + 2*DIM('a)]))$   
 $(list-of-eucl\ x\ @\ params\ @\ list-of-eucl\ d))$ ) (*at x*)  
 ⟨*proof*⟩

**lemma** *interpret-floatarith-FDERIV-floatarith*:

**assumes** *iD*:  $\bigwedge i j. i < DIM('a) \implies isDERIV\ i\ (fa)\ (list-of-eucl\ x)$   
**assumes** *m*: *max-Var-floatarith fa*  $\leq DIM('a)$   
**shows** ( $\lambda x::'a::executable-euclidean-space.$   
*interpret-floatarith fa* (*list-of-eucl x*) *has-derivative*  
 $(\lambda d. interpret-floatarith$   
 $(FDERIV-floatarith\ fa\ [0..<DIM('a)]\ (map\ Var\ [DIM('a)..<2*DIM('a)]))$   
 $(list-of-eucl\ x\ @\ list-of-eucl\ d))$ ) (*at x*)  
 ⟨*proof*⟩

**lemma** *interpret-floatarith-eventually-isDERIV*:

**assumes** *iD*:  $\bigwedge i j. i < DIM('a) \implies isDERIV\ i\ fa\ (list-of-eucl\ x\ @\ params)$   
**assumes** *m*: *max-Var-floatarith fa*  $\leq DIM('a::executable-euclidean-space) + length$   
*params*  
**shows**  $\forall i < DIM('a). \forall_F (x::'a)\ in\ at\ x. isDERIV\ i\ fa\ (list-of-eucl\ x\ @\ params)$   
 ⟨*proof*⟩

**lemma** *eventually-isFDERIV*:

**assumes** *iD*: *isFDERIV DIM('a) [0..<DIM('a)] fas* (*list-of-eucl x@params*)  
**assumes** *m*: *max-Var-floatariths fas*  $\leq DIM('a::executable-euclidean-space) +$   
*length params*  
**shows**  $\forall_F (x::'a)\ in\ at\ x. isFDERIV\ DIM('a)\ [0..<DIM('a)]\ fas\ (list-of-eucl\ x$   
 $@\ params)$   
 ⟨*proof*⟩

**lemma** *isFDERIV-eventually-isFDERIV*:

**assumes** *iD*: *isFDERIV DIM('a) [0..<DIM('a)] fas* (*list-of-eucl x@params*)  
**assumes** *m*: *max-Var-floatariths fas*  $\leq DIM('a::executable-euclidean-space) +$   
*length params*  
**shows**  $\forall_F (x::'a)\ in\ at\ x. isFDERIV\ DIM('a)\ [0..<DIM('a)]\ fas\ (list-of-eucl\ x$   
 $@\ params)$   
 ⟨*proof*⟩

**lemma** *interpret-floatarith-FDERIV-floatariths-eucl-of-env*:

**assumes**  $iD$ :  $isFDERIV\ DIM('a)\ xs\ fas\ vs$   
**assumes**  $fresh$ :  $freshs-floatariths\ (fas)\ ds$   
**assumes**  $[simp]$ :  $length\ ds = DIM\ ('a)$   
 $\bigwedge i. i \in set\ xs \implies i < length\ vs\ distinct\ xs$   
 $\bigwedge i. i \in set\ ds \implies i < length\ vs\ distinct\ ds$   
**shows**  $((\lambda x::'a::executable-euclidean-space.$   
 $eucl-of-list$   
 $(interpret-floatariths\ fas\ (list-updates\ xs\ (list-of-eucl\ x\ vs))::'a)\ has-derivative$   
 $(\lambda d. eucl-of-list\ (interpret-floatariths$   
 $(FDERIV-floatariths\ fas\ xs\ (map\ Var\ ds))$   
 $(list-updates\ ds\ (list-of-eucl\ d\ vs))))\ (at\ (eucl-of-env\ xs\ vs))$   
 $\langle proof \rangle$

**lemma**  $interpret-floatarith-FDERIV-floatariths-append$ :  
**assumes**  $iD$ :  $isFDERIV\ DIM('a)\ [0..<DIM('a)]\ fas\ (list-of-eucl\ x\ @\ ramsch)$   
**assumes**  $m$ :  $max-Var-floatariths\ fas \leq DIM('a) + length\ ramsch$   
**assumes**  $[simp]$ :  $length\ fas = DIM('a)$   
**shows**  $((\lambda x::'a::executable-euclidean-space.$   
 $eucl-of-list$   
 $(interpret-floatariths\ fas\ (list-of-eucl\ x@ramsch))::'a)\ has-derivative$   
 $(\lambda d. eucl-of-list\ (interpret-floatariths$   
 $(FDERIV-floatariths\ fas\ [0..<DIM('a)]\ (map\ Var\ [DIM('a)+length\ ram-$   
 $sch..<2*DIM('a) + length\ ramsch]))$   
 $(list-of-eucl\ x\ @\ ramsch\ @\ list-of-eucl\ d))))\ (at\ x)$   
 $\langle proof \rangle$

**lemma**  $interpret-floatarith-FDERIV-floatariths$ :  
**assumes**  $iD$ :  $isFDERIV\ DIM('a)\ [0..<DIM('a)]\ fas\ (list-of-eucl\ x)$   
**assumes**  $m$ :  $max-Var-floatariths\ fas \leq DIM('a)$   
**assumes**  $[simp]$ :  $length\ fas = DIM('a)$   
**shows**  $((\lambda x::'a::executable-euclidean-space.$   
 $eucl-of-list$   
 $(interpret-floatariths\ fas\ (list-of-eucl\ x))::'a)\ has-derivative$   
 $(\lambda d. eucl-of-list\ (interpret-floatariths$   
 $(FDERIV-floatariths\ fas\ [0..<DIM('a)]\ (map\ Var\ [DIM('a)..<2*DIM('a)]))$   
 $(list-of-eucl\ x\ @\ list-of-eucl\ d))))\ (at\ x)$   
 $\langle proof \rangle$

**lemma**  $continuous-on-min[continuous-intros]$ :  
**fixes**  $f\ g :: 'a::topological-space \Rightarrow 'b::linorder-topology$   
**shows**  $continuous-on\ A\ f \implies continuous-on\ A\ g \implies continuous-on\ A\ (\lambda x. min$   
 $(f\ x)\ (g\ x))$   
 $\langle proof \rangle$

**lemmas**  $[continuous-intros] = continuous-on-max$

**lemma**  $continuous-on-if-const[continuous-intros]$ :  
 $continuous-on\ s\ f \implies continuous-on\ s\ g \implies continuous-on\ s\ (\lambda x. if\ p\ then\ f\ x$   
 $else\ g\ x)$   
 $\langle proof \rangle$

**lemma** *continuous-on-floatarith*:

**assumes** *continuous-on-floatarith* *fa* *length xs = DIM('a)* *distinct xs*  
**shows** *continuous-on UNIV* ( $\lambda x. \text{interpret-floatarith } fa \text{ (list-updates } xs \text{ (list-of-eucl } (x::'a)::\text{executable-euclidean-space})) } vs)$ )  
(*proof*)

**fun** *open-form* :: *form*  $\Rightarrow$  *bool* **where**

*open-form* (*Bound* *x a b f*) = *False* |  
*open-form* (*Assign* *x a f*) = *False* |  
*open-form* (*Less* *a b*)  $\longleftrightarrow$  *continuous-on-floatarith a*  $\wedge$  *continuous-on-floatarith b* |  
*open-form* (*LessEqual* *a b*) = *False* |  
*open-form* (*AtLeastAtMost* *x a b*) = *False* |  
*open-form* (*Conj* *f g*)  $\longleftrightarrow$  *open-form f*  $\wedge$  *open-form g* |  
*open-form* (*Disj* *f g*)  $\longleftrightarrow$  *open-form f*  $\wedge$  *open-form g*

**lemma** *open-form*:

**assumes** *open-form f* *length xs = DIM('a)::executable-euclidean-space* *distinct xs*  
**shows** *open* (*Collect* ( $\lambda x::'a. \text{interpret-form } f \text{ (list-updates } xs \text{ (list-of-eucl } x \text{ } vs))$ ))  
(*proof*)

**primrec** *isnFDERIV* **where**

*isnFDERIV N fas xs ds vs 0* = *True*  
| *isnFDERIV N fas xs ds vs (Suc n)*  $\longleftrightarrow$   
*isFDERIV N xs (FDERIV-n-floatariths fas xs (map Var ds) n) vs*  $\wedge$   
*isnFDERIV N fas xs ds vs n*

**lemma** *one-add-square-eq-0*:  $1 + (x)^2 \neq (0::\text{real})$

(*proof*)

**lemma** *isDERIV-fold-const-fa*[*intro*]:

**assumes** *isDERIV x fa vs*  
**shows** *isDERIV x (fold-const-fa fa) vs*  
(*proof*)

**lemma** *isDERIV-fold-const-fa-minus*[*intro!*]:

**assumes** *isDERIV x (fold-const-fa fa) vs*  
**shows** *isDERIV x (fold-const-fa (Minus fa)) vs*  
(*proof*)

**lemma** *isDERIV-fold-const-fa-plus*[*intro!*]:

**assumes** *isDERIV x (fold-const-fa fa) vs*  
**assumes** *isDERIV x (fold-const-fa fb) vs*  
**shows** *isDERIV x (fold-const-fa (Add fa fb)) vs*  
(*proof*)

**lemma** *isDERIV-fold-const-fa-mult*[*intro!*]:

**assumes** *isDERIV x (fold-const-fa fa) vs*  
**assumes** *isDERIV x (fold-const-fa fb) vs*

**shows**  $isDERIV\ x\ (fold-const-fa\ (Mult\ fa\ fb))\ vs$   
 $\langle proof \rangle$

**lemma**  $isDERIV-fold-const-fa-power[intro!]$ :  
**assumes**  $isDERIV\ x\ (fold-const-fa\ fa)\ vs$   
**shows**  $isDERIV\ x\ (fold-const-fa\ (fa\ \hat{e}\ n))\ vs$   
 $\langle proof \rangle$

**lemma**  $isDERIV-fold-const-fa-inverse[intro!]$ :  
**assumes**  $isDERIV\ x\ (fold-const-fa\ fa)\ vs$   
**assumes**  $interpret-floatarith\ fa\ vs\ \neq\ 0$   
**shows**  $isDERIV\ x\ (fold-const-fa\ (Inverse\ fa))\ vs$   
 $\langle proof \rangle$

**lemma**  $add-square-ne-zero[simp]$ :  $(y::'a::linordered-idom) > 0 \implies y + x^2 \neq 0$   
 $\langle proof \rangle$

**lemma**  $isDERIV-FDERIV-floatarith$ :  
**assumes**  $isDERIV\ x\ fa\ vs\ \wedge\ i.\ i < length\ ds \implies isDERIV\ x\ (ds\ !\ i)\ vs$   
**assumes**  $[simp]:\ length\ xs = length\ ds$   
**shows**  $isDERIV\ x\ (FDERIV-floatarith\ fa\ xs\ ds)\ vs$   
 $\langle proof \rangle$

**lemma**  $isDERIV-FDERIV-floatariths$ :  
**assumes**  $isFDERIV\ N\ xs\ fas\ vs\ isFDERIV\ N\ xs\ ds\ vs\ \mathbf{and}\ [simp]:\ length\ fas =$   
 $length\ ds$   
**shows**  $isFDERIV\ N\ xs\ (FDERIV-floatariths\ fas\ xs\ ds)\ vs$   
 $\langle proof \rangle$

**lemma**  $isFDERIV-imp-isFDERIV-FDERIV-n$ :  
**assumes**  $length\ fas = length\ ds$   
**shows**  $isFDERIV\ N\ xs\ fas\ vs \implies isFDERIV\ N\ xs\ ds\ vs \implies$   
 $isFDERIV\ N\ xs\ (FDERIV-n-floatariths\ fas\ xs\ ds\ n)\ vs$   
 $\langle proof \rangle$

**lemma**  $isFDERIV-map-Var$ :  
**assumes**  $[simp]:\ length\ ds = N\ length\ xs = N$   
**shows**  $isFDERIV\ N\ xs\ (map\ Var\ ds)\ vs$   
 $\langle proof \rangle$

**theorem**  $isFDERIV-imp-isnFDERIV$ :  
**assumes**  $isFDERIV\ N\ xs\ fas\ vs\ \mathbf{and}\ [simp]:\ length\ fas = N\ length\ xs = N\ length$   
 $ds = N$   
**shows**  $isnFDERIV\ N\ fas\ xs\ ds\ vs\ n$   
 $\langle proof \rangle$

**lemma**  $eventually-isnFDERIV$ :  
**assumes**  $iD:\ isnFDERIV\ DIM('a)\ fas\ [0..<DIM('a)]\ [DIM('a)..<2*DIM('a)]$   
 $(list-of-eucl\ x\ @\ list-of-eucl\ (d::'a))\ n$

**assumes**  $m$ : *max-Var-floatariths*  $fas \leq 2 * DIM('a::executable-euclidean-space)$   
**shows**  $\forall_F (x::'a)$  in at  $x$ . *isnFDERIV*  $DIM('a)$   $fas$   $[0..<DIM('a)]$   $[DIM('a)..<2*DIM('a)]$   
*(list-of-eucl x @ list-of-eucl d)*  $n$   
 $\langle proof \rangle$

**lemma** *isFDERIV-open*:

**assumes** *max-Var-floatariths*  $fas \leq DIM('a)$   
**shows** *open*  $\{x::'a. isFDERIV DIM('a::executable-euclidean-space) [0..<DIM('a)]$   
 $fas$  *(list-of-eucl x)* $\}$   
*(is open (Collect ?s))*  
 $\langle proof \rangle$

**lemma** *interpret-floatarith-FDERIV-floatarith-eq*:

**assumes**  $[simp]$ : *length*  $xs = DIM('a::executable-euclidean-space)$  *length*  $ds =$   
 $DIM('a)$   
**shows** *interpret-floatarith*  $(FDERIV-floatarith fa xs ds)$   $vs =$   
*einterpret*  $(map (\lambda x. DERIV-floatarith x fa) xs)$   $vs \cdot (einterpret ds vs::'a)$   
 $\langle proof \rangle$

**lemma**

*interpret-floatariths-FDERIV-floatariths-cong*:  
**assumes**  $[simp]$ : *length*  $d1s = DIM('a::executable-euclidean-space)$  *length*  $d2s =$   
 $DIM('a)$  *length*  $fas1 = \text{length } fas2$   
**assumes** *fresh1*: *freshs-floatariths*  $fas1$   $d1s$   
**assumes** *fresh2*: *freshs-floatariths*  $fas2$   $d2s$   
**assumes**  $eq1$ :  $\bigwedge i. i < \text{length } fas1 \implies \text{interpret-floatariths } (map (\lambda x. DE-$   
 $RIV-floatarith x (fas1 ! i)) [0..<DIM('a)]) xs1 =$   
*interpret-floatariths*  $(map (\lambda x. DERIV-floatarith x (fas2 ! i)) [0..<DIM('a)])$   
 $xs2$   
**assumes**  $eq2$ :  $\bigwedge i. i < DIM('a) \implies xs1 ! (d1s ! i) = xs2 ! (d2s ! i)$   
**shows** *interpret-floatariths*  $(FDERIV-floatariths fas1 [0..<DIM('a)] (map floatarith. Var$   
 $d1s)) xs1 =$   
*interpret-floatariths*  $(FDERIV-floatariths fas2 [0..<DIM('a)] (map floatarith. Var$   
 $d2s)) xs2$   
 $\langle proof \rangle$

**lemma** *subst-floatarith-Var-DERIV-floatarith*:

**assumes**  $\bigwedge x. x = n \longleftrightarrow s x = n$   
**shows** *subst-floatarith*  $(\lambda x. Var (s x)) (DERIV-floatarith n fa) =$   
 $DERIV-floatarith n (subst-floatarith (\lambda x. Var (s x)) fa)$   
 $\langle proof \rangle$

**lemma** *subst-floatarith-inner-floatariths* $[simp]$ :

**assumes** *length*  $fs = \text{length } gs$   
**shows** *subst-floatarith*  $s$  *(inner-floatariths fs gs)*  $=$   
*inner-floatariths*  $(map (subst-floatarith s) fs)$   $(map (subst-floatarith s) gs)$   
 $\langle proof \rangle$

**fun-cases** *subst-floatarith-Num*: *subst-floatarith*  $s fa = Num y$

**and** *subst-floatarith-Add*:  $\text{subst-floatarith } s \text{ fa} = \text{Add } x \ y$   
**and** *subst-floatarith-Minus*:  $\text{subst-floatarith } s \text{ fa} = \text{Minus } y$

**lemma** *Num-eq-subst-Var[simp]*:  $\text{Num } x = \text{subst-floatarith } (\lambda x. \text{Var } (s \ x)) \ \text{fa} \longleftrightarrow \text{fa} = \text{Num } x$   
 ⟨*proof*⟩

**lemma** *Add-eq-subst-VarE*:  
**assumes**  $\text{Add } \text{fa1 } \text{fa2} = \text{subst-floatarith } (\lambda x. \text{Var } (s \ x)) \ \text{fa}$   
**obtains**  $a1 \ a2$  **where**  $\text{fa} = \text{Add } a1 \ a2$   $\text{fa1} = \text{subst-floatarith } (\lambda x. \text{Var } (s \ x)) \ a1$   
 $\text{fa2} = \text{subst-floatarith } (\lambda x. \text{Var } (s \ x)) \ a2$   
 ⟨*proof*⟩

**lemma** *subst-floatarith-eq-self[simp]*:  $\text{subst-floatarith } s \ f = f$  **if**  $\text{max-Var-floatarith } f = 0$   
 ⟨*proof*⟩

**lemma** *fold-const-fa-unique*:  $\text{False}$  **if**  $(\bigwedge x. f = \text{Num } x)$   
 ⟨*proof*⟩

**lemma** *zero-unique*:  $\text{False}$  **if**  $(\bigwedge x::\text{float}. x = 0)$   
 ⟨*proof*⟩

**lemma** *fold-const-fa-Mult-eq-NumE*:  
**assumes**  $\text{fold-const-fa } (\text{Mult } a \ b) = \text{Num } x$   
**obtains**  $y \ z$  **where**  $\text{fold-const-fa } a = \text{Num } y$   $\text{fold-const-fa } b = \text{Num } z$   $x = y * z$   
 |  $y$  **where**  $\text{fold-const-fa } a = \text{Num } 0$   $x = 0$   
 |  $y$  **where**  $\text{fold-const-fa } b = \text{Num } 0$   $x = 0$   
 ⟨*proof*⟩

**lemma** *fold-const-fa-Add-eq-NumE*:  
**assumes**  $\text{fold-const-fa } (\text{Add } a \ b) = \text{Num } x$   
**obtains**  $y \ z$  **where**  $\text{fold-const-fa } a = \text{Num } y$   $\text{fold-const-fa } b = \text{Num } z$   $x = y + z$   
 ⟨*proof*⟩

**lemma** *subst-floatarith-Var-fold-const-fa[symmetric]*:  
 $\text{fold-const-fa } (\text{subst-floatarith } (\lambda x. \text{Var } (s \ x)) \ \text{fa}) =$   
 $\text{subst-floatarith } (\lambda x. \text{Var } (s \ x)) \ (\text{fold-const-fa } \text{fa})$   
 ⟨*proof*⟩

**lemma** *subst-floatarith-eq-Num[simp]*:  $(\text{subst-floatarith } (\lambda x. \text{Var } (s \ x)) \ \text{fa} = \text{Num } x) \longleftrightarrow \text{fa} = \text{Num } x$   
 ⟨*proof*⟩

**lemma** *fold-const-fa-subst-eq-Num0-iff[simp]*:  
 $\text{fold-const-fa } (\text{subst-floatarith } (\lambda x. \text{Var } (s \ x)) \ \text{fa}) = \text{Num } x \longleftrightarrow \text{fold-const-fa } \text{fa} = \text{Num } x$   
 ⟨*proof*⟩

**lemma** *subst-floatarith-Var-FDERIV-floatarith*:

**assumes** *len*:  $\text{length } xs = \text{DIM}('a::\text{executable-euclidean-space})$  **and** [*simp*]:  $\text{length } ds = \text{DIM}('a)$

**assumes** *eq*:  $\bigwedge x y. x \in \text{set } xs \implies (y = x) = (s y = x)$

**shows**  $\text{subst-floatarith } (\lambda x. \text{Var } (s x)) (\text{FDERIV-floatarith } fa \text{ } xs \text{ } ds) =$   
 $(\text{FDERIV-floatarith } (\text{subst-floatarith } (\lambda x. \text{Var } (s x)) \text{ } fa) \text{ } xs \text{ } (\text{map } (\text{subst-floatarith } (\lambda x. \text{Var } (s x))) \text{ } ds))$

*<proof>*

**lemma** *subst-floatarith-Var-FDERIV-n-nth*:

**assumes** *len*:  $\text{length } xs = \text{DIM}('a::\text{executable-euclidean-space})$  **and** [*simp*]:  $\text{length } ds = \text{DIM}('a)$

**assumes** *eq*:  $\bigwedge x y. x \in \text{set } xs \implies (y = x) = (s y = x)$

**assumes** [*simp*]:  $i < \text{length } fas$

**shows**  $\text{subst-floatarith } (\lambda x. \text{Var } (s x)) (\text{FDERIV-n-floatariths } fas \text{ } xs \text{ } ds \text{ } n ! i) =$   
 $(\text{FDERIV-n-floatariths } (\text{map } (\text{subst-floatarith } (\lambda x. \text{Var } (s x))) \text{ } fas) \text{ } xs \text{ } (\text{map } (\text{subst-floatarith } (\lambda x. \text{Var } (s x))) \text{ } ds) \text{ } n ! i)$

*<proof>*

**lemma** *subst-floatarith-Var-max-Var-floatarith*:

**assumes**  $\bigwedge i. i < \text{max-Var-floatarith } fa \implies s i = i$

**shows**  $\text{subst-floatarith } (\lambda i. \text{Var } (s i)) \text{ } fa = fa$

*<proof>*

**lemma** *interpret-floatarith-subst-floatarith-idem*:

**assumes** *mv*:  $\text{max-Var-floatarith } fa \leq \text{length } vs$

**assumes** *idem*:  $\bigwedge j. j < \text{max-Var-floatarith } fa \implies vs ! s j = vs ! j$

**shows**  $\text{interpret-floatarith } (\text{subst-floatarith } (\lambda i. \text{Var } (s i)) \text{ } fa) \text{ } vs = \text{interpret-floatarith } fa \text{ } vs$

*<proof>*

**lemma** *isDERIV-subst-Var-floatarith*:

**assumes** *mv*:  $\text{max-Var-floatarith } fa \leq \text{length } vs$

**assumes** *idem*:  $\bigwedge j. j < \text{max-Var-floatarith } fa \implies vs ! s j = vs ! j$

**assumes**  $\bigwedge j. s j = i \iff j = i$

**shows**  $\text{isDERIV } i \text{ } (\text{subst-floatarith } (\lambda i. \text{Var } (s i)) \text{ } fa) \text{ } vs = \text{isDERIV } i \text{ } fa \text{ } vs$

*<proof>*

**lemma** *isFDERIV-subst-Var-floatarith*:

**assumes** *mv*:  $\text{max-Var-floatariths } fas \leq \text{length } vs$

**assumes** *idem*:  $\bigwedge j. j < \text{max-Var-floatariths } fas \implies vs ! (s j) = vs ! j$

**assumes**  $\bigwedge i j. i \in \text{set } xs \implies s j = i \iff j = i$

**shows**  $\text{isFDERIV } n \text{ } xs \text{ } (\text{map } (\text{subst-floatarith } (\lambda i. \text{Var } (s i))) \text{ } fas) \text{ } vs = \text{isFDERIV } n \text{ } xs \text{ } fas \text{ } vs$

*<proof>*

**lemma** *interpret-floatariths-append*[*simp*]:

$\text{interpret-floatariths } (xs @ ys) \text{ } vs = \text{interpret-floatariths } xs \text{ } vs @ \text{interpret-floatariths } ys \text{ } vs$

*<proof>*

**lemma** *not-fresh-inner-floatariths*:

**assumes**  $\text{length } xs = \text{length } ys$

**shows**  $\neg \text{fresh-floatarith } (\text{inner-floatariths } xs \ ys) \ i \longleftrightarrow \neg \text{fresh-floatariths } xs \ i \vee \neg \text{fresh-floatariths } ys \ i$

*<proof>*

**lemma** *fresh-inner-floatariths*:

**assumes**  $\text{length } xs = \text{length } ys$

**shows**  $\text{fresh-floatarith } (\text{inner-floatariths } xs \ ys) \ i \longleftrightarrow \text{fresh-floatariths } xs \ i \wedge \text{fresh-floatariths } ys \ i$

*<proof>*

**lemma** *not-fresh-floatariths-map*:

$\neg \text{fresh-floatariths } (\text{map } f \ xs) \ i \longleftrightarrow (\exists x \in \text{set } xs. \neg \text{fresh-floatarith } (f \ x) \ i)$

*<proof>*

**lemma** *fresh-floatariths-map*:

$\text{fresh-floatariths } (\text{map } f \ xs) \ i \longleftrightarrow (\forall x \in \text{set } xs. \text{fresh-floatarith } (f \ x) \ i)$

*<proof>*

**lemma** *fresh-floatarith-fold-const-fa*:  $\text{fresh-floatarith } fa \ i \implies \text{fresh-floatarith } (\text{fold-const-fa } fa) \ i$

*<proof>*

**lemma** *fresh-floatarith-fold-const-fa-Add[intro!]*:

**assumes**  $\text{fresh-floatarith } (\text{fold-const-fa } a) \ i$   $\text{fresh-floatarith } (\text{fold-const-fa } b) \ i$

**shows**  $\text{fresh-floatarith } (\text{fold-const-fa } (\text{Add } a \ b)) \ i$

*<proof>*

**lemma** *fresh-floatarith-fold-const-fa-Mult[intro!]*:

**assumes**  $\text{fresh-floatarith } (\text{fold-const-fa } a) \ i$   $\text{fresh-floatarith } (\text{fold-const-fa } b) \ i$

**shows**  $\text{fresh-floatarith } (\text{fold-const-fa } (\text{Mult } a \ b)) \ i$

*<proof>*

**lemma** *fresh-floatarith-fold-const-fa-Minus[intro!]*:

**assumes**  $\text{fresh-floatarith } (\text{fold-const-fa } b) \ i$

**shows**  $\text{fresh-floatarith } (\text{fold-const-fa } (\text{Minus } b)) \ i$

*<proof>*

**lemma** *fresh-FDERIV-floatarith*:

$\text{fresh-floatarith } \text{ode-e } i \implies \text{fresh-floatariths } ds \ i$

$\implies \text{length } ds = \text{DIM}('a)$

$\implies \text{fresh-floatarith } (\text{FDERIV-floatarith } \text{ode-e } [0..<\text{DIM}('a)::\text{executable-euclidean-space}])$

$ds) \ i$

*<proof>*

**lemma** *not-fresh-FDERIV-floatarith*:

$\neg \text{fresh-floatarith} (\text{FDERIV-floatarith ode-e } [0..<\text{DIM}('a)::\text{executable-euclidean-space}])$   
 $ds) i$   
 $\implies \text{length } ds = \text{DIM}('a)$   
 $\implies \neg \text{fresh-floatarith ode-e } i \vee \neg \text{fresh-floatariths } ds i$   
 $\langle \text{proof} \rangle$

**lemma** *not-fresh-FDERIV-floatariths:*

$\neg \text{fresh-floatariths} (\text{FDERIV-floatariths ode-e } [0..<\text{DIM}('a)::\text{executable-euclidean-space}])$   
 $ds) i \implies$   
 $\text{length } ds = \text{DIM}('a) \implies \neg \text{fresh-floatariths ode-e } i \vee \neg \text{fresh-floatariths } ds i$   
 $\langle \text{proof} \rangle$

**lemma** *isDERIV-FDERIV-floatarith-linear:*

**fixes**  $x h::'a::\text{executable-euclidean-space}$   
**assumes**  $\bigwedge k. k < \text{DIM}('a) \implies \text{isDERIV } i (\text{DERIV-floatarith } k \text{ } fa) \text{ } xs$   
**assumes**  $\text{max-Var-floatarith } fa \leq \text{DIM}('a)$   
**assumes**  $[\text{simp}]: \text{length } xs = \text{DIM}('a) \text{ length } hs = \text{DIM}('a)$   
**shows**  $\text{isDERIV } i (\text{FDERIV-floatarith } fa [0..<\text{DIM}('a)]) (\text{map Var } [\text{DIM}('a)..<2$   
 $* \text{DIM}('a)])$   
 $(xs @ hs)$   
 $\langle \text{proof} \rangle$

**lemma**

*isFDERIV-FDERIV-floatariths-linear:*  
**fixes**  $x h::'a::\text{executable-euclidean-space}$   
**assumes**  $\bigwedge i j k.$   
 $i < \text{DIM}('a) \implies$   
 $j < \text{DIM}('a) \implies k < \text{DIM}('a) \implies \text{isDERIV } i (\text{DERIV-floatarith } k (fas !$   
 $j)) (xs)$   
**assumes**  $[\text{simp}]: \text{length } fas = \text{DIM}('a::\text{executable-euclidean-space})$   
**assumes**  $[\text{simp}]: \text{length } xs = \text{DIM}('a) \text{ length } hs = \text{DIM}('a)$   
**assumes**  $\text{max-Var-floatariths } fas \leq \text{DIM}('a)$   
**shows**  $\text{isFDERIV } \text{DIM}('a) [0..<\text{DIM}('a)::\text{executable-euclidean-space}]$   
 $(\text{FDERIV-floatariths } fas [0..<\text{DIM}('a)]) (\text{map floatarith.Var } [\text{DIM}('a)..<2 *$   
 $\text{DIM}('a)])$   
 $(xs @ hs)$   
 $\langle \text{proof} \rangle$

**definition** *isFDERIV-approx where*

$\text{isFDERIV-approx } p \ n \ xs \ fas \ vs =$   
 $((\forall i < n. \forall j < n. \text{isDERIV-approx } p (xs ! i) (fas ! j) vs) \wedge \text{length } fas = n \wedge \text{length}$   
 $xs = n)$

**lemma** *isFDERIV-approx:*

$\text{bounded-by } vs \ VS \implies \text{isFDERIV-approx } \text{prec } n \ xs \ fas \ VS \implies \text{isFDERIV } n \ xs$   
 $fas \ vs$   
 $\langle \text{proof} \rangle$

**primrec** *isnFDERIV-approx where*

$isnFDERIV\text{-}approx\ p\ N\ fas\ xs\ ds\ vs\ 0 = True$   
 $| isnFDERIV\text{-}approx\ p\ N\ fas\ xs\ ds\ vs\ (Suc\ n) \longleftrightarrow$   
 $\quad isnFDERIV\text{-}approx\ p\ N\ xs\ (FDERIV\text{-}n\text{-}floatariths\ fas\ xs\ (map\ Var\ ds)\ n)\ vs \wedge$   
 $\quad isnFDERIV\text{-}approx\ p\ N\ fas\ xs\ ds\ vs\ n$

**lemma**  $isnFDERIV\text{-}approx$ :

$bounded\text{-}by\ vs\ VS \implies isnFDERIV\text{-}approx\ prec\ N\ fas\ xs\ ds\ VS\ n \implies isnFDERIV$   
 $N\ fas\ xs\ ds\ vs\ n$   
 $\langle proof \rangle$

**fun**  $plain\text{-}floatarith::nat \Rightarrow floatarith \Rightarrow bool$  **where**

$plain\text{-}floatarith\ N\ (floatarith.Add\ a\ b) \longleftrightarrow plain\text{-}floatarith\ N\ a \wedge plain\text{-}floatarith$   
 $N\ b$   
 $| plain\text{-}floatarith\ N\ (floatarith.Mult\ a\ b) \longleftrightarrow plain\text{-}floatarith\ N\ a \wedge plain\text{-}floatarith$   
 $N\ b$   
 $| plain\text{-}floatarith\ N\ (floatarith.Minus\ a) \longleftrightarrow plain\text{-}floatarith\ N\ a$   
 $| plain\text{-}floatarith\ N\ (floatarith.Pi) \longleftrightarrow True$   
 $| plain\text{-}floatarith\ N\ (floatarith.Num\ n) \longleftrightarrow True$   
 $| plain\text{-}floatarith\ N\ (floatarith.Var\ i) \longleftrightarrow i < N$   
 $| plain\text{-}floatarith\ N\ (floatarith.Max\ a\ b) \longleftrightarrow plain\text{-}floatarith\ N\ a \wedge plain\text{-}floatarith$   
 $N\ b$   
 $| plain\text{-}floatarith\ N\ (floatarith.Min\ a\ b) \longleftrightarrow plain\text{-}floatarith\ N\ a \wedge plain\text{-}floatarith$   
 $N\ b$   
 $| plain\text{-}floatarith\ N\ (floatarith.Power\ a\ n) \longleftrightarrow plain\text{-}floatarith\ N\ a$   
 $| plain\text{-}floatarith\ N\ (floatarith.Cos\ a) \longleftrightarrow False$  — TODO: should be plain!  
 $| plain\text{-}floatarith\ N\ (floatarith.Arctan\ a) \longleftrightarrow False$  — TODO: should be plain!  
 $| plain\text{-}floatarith\ N\ (floatarith.Abs\ a) \longleftrightarrow plain\text{-}floatarith\ N\ a$   
 $| plain\text{-}floatarith\ N\ (floatarith.Exp\ a) \longleftrightarrow False$  — TODO: should be plain!  
 $| plain\text{-}floatarith\ N\ (floatarith.Sqrt\ a) \longleftrightarrow False$  — TODO: should be plain!  
 $| plain\text{-}floatarith\ N\ (floatarith.Floor\ a) \longleftrightarrow plain\text{-}floatarith\ N\ a$   
  
 $| plain\text{-}floatarith\ N\ (floatarith.Powr\ a\ b) \longleftrightarrow False$   
 $| plain\text{-}floatarith\ N\ (floatarith.Inverse\ a) \longleftrightarrow False$   
 $| plain\text{-}floatarith\ N\ (floatarith.Ln\ a) \longleftrightarrow False$

**lemma**  $plain\text{-}floatarith\text{-}approx\text{-}not\text{-}None$ :

$assumes\ plain\text{-}floatarith\ N\ fa\ N \leq length\ XS \wedge i. i < N \implies XS\ !\ i \neq None$   
 $shows\ approx\ p\ fa\ XS \neq None$   
 $\langle proof \rangle$

**definition**  $Rad\text{-}of\ w = w * (Pi / Num\ 180)$

**lemma**  $interpret\text{-}Rad\text{-}of[simp]$ :  $interpret\text{-}floatarith\ (Rad\text{-}of\ w)\ xs = rad\text{-}of\ (interpret\text{-}floatarith\ w\ xs)$   
 $\langle proof \rangle$

**definition**  $Deg\text{-}of\ w = Num\ 180 * w / Pi$

**lemma**  $interpret\text{-}Deg\text{-}of[simp]$ :  $interpret\text{-}floatarith\ (Deg\text{-}of\ w)\ xs = deg\text{-}of\ (interpret\text{-}floatarith\ w\ xs)$

*<proof>*

**unbundle** *no floatarith-syntax*

**end**

## 4 Straight Line Programs

**theory** *Straight-Line-Program*

**imports**

*Floatarith-Expression*

*Deriving.Derive*

*HOL-Library.Monad-Syntax*

*HOL-Library.RBT-Mapping*

**begin**

**unbundle** *floatarith-syntax*

**derive** (*linorder*) *compare-order float*

**derive** *linorder floatarith*

### 4.1 Definition

**type-synonym** *slp = floatarith list*

**primrec** *interpret-slp::slp  $\Rightarrow$  real list  $\Rightarrow$  real list* **where**

*interpret-slp [] = ( $\lambda xs. xs$ )*

| *interpret-slp (ea # eas) = ( $\lambda xs. interpret-slp eas (interpret-floatarith ea xs#xs)$ )*

### 4.2 Reification as straight line program (with common subexpression elimination)

**definition** *slp-index vs i = (length vs - Suc i)*

**definition** *slp-index-lookup vs M a = slp-index vs (the (Mapping.lookup M a))*

**definition**

*slp-of-fa-bin Binop a b M slp M2 slp2 =*

*(case Mapping.lookup M (Binop a b) of*

*Some i  $\Rightarrow$  (Mapping.update (Binop a b) (length slp) M, slp@[Var (slp-index slp i)])*

*| None  $\Rightarrow$  (Mapping.update (Binop a b) (length slp2) M2,*

*slp2@[Binop (Var (slp-index-lookup slp2 M2 a)) (Var (slp-index-lookup slp2 M2 b))]))*

**definition**

*slp-of-fa-un Unop a M slp M1 slp1 =*

*(case Mapping.lookup M (Unop a) of*

$Some\ i \Rightarrow (Mapping.update\ (Unop\ a)\ (length\ slp)\ M,\ slp@[Var\ (slp-index\ slp\ i)])$   
 $| None \Rightarrow (Mapping.update\ (Unop\ a)\ (length\ slp1)\ M1,\ slp1@[Unop\ (Var\ (slp-index-lookup\ slp1\ M1\ a))])$

**definition**

$slp-of-fa-cnst\ Const\ Const'\ M\ vs =$   
 $(Mapping.update\ Const\ (length\ vs)\ M,$   
 $vs\ @\ [case\ Mapping.lookup\ M\ Const\ of\ Some\ i \Rightarrow Var\ (slp-index\ vs\ i) | None$   
 $\Rightarrow Const'])$

**fun**  $slp-of-fa :: floatarith \Rightarrow (floatarith, nat)\ mapping \Rightarrow floatarith\ list \Rightarrow$   
 $((floatarith, nat)\ mapping \times floatarith\ list)\ \mathbf{where}$   
 $slp-of-fa\ (Add\ a\ b)\ M\ slp =$   
 $(let\ (M1,\ slp1) = slp-of-fa\ a\ M\ slp;\ (M2,\ slp2) = slp-of-fa\ b\ M1\ slp1\ in$   
 $slp-of-fa-bin\ Add\ a\ b\ M\ slp\ M2\ slp2)$   
 $| slp-of-fa\ (Mult\ a\ b)\ M\ slp =$   
 $(let\ (M1,\ slp1) = slp-of-fa\ a\ M\ slp;\ (M2,\ slp2) = slp-of-fa\ b\ M1\ slp1\ in$   
 $slp-of-fa-bin\ Mult\ a\ b\ M\ slp\ M2\ slp2)$   
 $| slp-of-fa\ (Min\ a\ b)\ M\ slp =$   
 $(let\ (M1,\ slp1) = slp-of-fa\ a\ M\ slp;\ (M2,\ slp2) = slp-of-fa\ b\ M1\ slp1\ in$   
 $slp-of-fa-bin\ Min\ a\ b\ M\ slp\ M2\ slp2)$   
 $| slp-of-fa\ (Max\ a\ b)\ M\ slp =$   
 $(let\ (M1,\ slp1) = slp-of-fa\ a\ M\ slp;\ (M2,\ slp2) = slp-of-fa\ b\ M1\ slp1\ in$   
 $slp-of-fa-bin\ Max\ a\ b\ M\ slp\ M2\ slp2)$   
 $| slp-of-fa\ (Powr\ a\ b)\ M\ slp =$   
 $(let\ (M1,\ slp1) = slp-of-fa\ a\ M\ slp;\ (M2,\ slp2) = slp-of-fa\ b\ M1\ slp1\ in$   
 $slp-of-fa-bin\ Powr\ a\ b\ M\ slp\ M2\ slp2)$   
 $| slp-of-fa\ (Inverse\ a)\ M\ slp =$   
 $(let\ (M1,\ slp1) = slp-of-fa\ a\ M\ slp\ in\ slp-of-fa-un\ Inverse\ a\ M\ slp\ M1\ slp1)$   
 $| slp-of-fa\ (Cos\ a)\ M\ slp =$   
 $(let\ (M1,\ slp1) = slp-of-fa\ a\ M\ slp\ in\ slp-of-fa-un\ Cos\ a\ M\ slp\ M1\ slp1)$   
 $| slp-of-fa\ (Arctan\ a)\ M\ slp =$   
 $(let\ (M1,\ slp1) = slp-of-fa\ a\ M\ slp\ in\ slp-of-fa-un\ Arctan\ a\ M\ slp\ M1\ slp1)$   
 $| slp-of-fa\ (Abs\ a)\ M\ slp =$   
 $(let\ (M1,\ slp1) = slp-of-fa\ a\ M\ slp\ in\ slp-of-fa-un\ Abs\ a\ M\ slp\ M1\ slp1)$   
 $| slp-of-fa\ (Sqrt\ a)\ M\ slp =$   
 $(let\ (M1,\ slp1) = slp-of-fa\ a\ M\ slp\ in\ slp-of-fa-un\ Sqrt\ a\ M\ slp\ M1\ slp1)$   
 $| slp-of-fa\ (Exp\ a)\ M\ slp =$   
 $(let\ (M1,\ slp1) = slp-of-fa\ a\ M\ slp\ in\ slp-of-fa-un\ Exp\ a\ M\ slp\ M1\ slp1)$   
 $| slp-of-fa\ (Ln\ a)\ M\ slp =$   
 $(let\ (M1,\ slp1) = slp-of-fa\ a\ M\ slp\ in\ slp-of-fa-un\ Ln\ a\ M\ slp\ M1\ slp1)$   
 $| slp-of-fa\ (Minus\ a)\ M\ slp =$   
 $(let\ (M1,\ slp1) = slp-of-fa\ a\ M\ slp\ in\ slp-of-fa-un\ Minus\ a\ M\ slp\ M1\ slp1)$   
 $| slp-of-fa\ (Floor\ a)\ M\ slp =$   
 $(let\ (M1,\ slp1) = slp-of-fa\ a\ M\ slp\ in\ slp-of-fa-un\ Floor\ a\ M\ slp\ M1\ slp1)$   
 $| slp-of-fa\ (Power\ a\ n)\ M\ slp =$   
 $(let\ (M1,\ slp1) = slp-of-fa\ a\ M\ slp\ in\ slp-of-fa-un\ (\lambda a.\ Power\ a\ n)\ a\ M\ slp\ M1\ slp1)$

|  $slp\text{-of-fa } Pi \ M \ slp = slp\text{-of-fa-cnst } Pi \ Pi \ M \ slp$   
|  $slp\text{-of-fa } (Var \ v) \ M \ slp = slp\text{-of-fa-cnst } (Var \ v) \ (Var \ (v + length \ slp)) \ M \ slp$   
|  $slp\text{-of-fa } (Num \ n) \ M \ slp = slp\text{-of-fa-cnst } (Num \ n) \ (Num \ n) \ M \ slp$

**lemma** *interpret-slp-snoc[simp]*:

$interpret\text{-slp } (slp \ @ \ [fa]) \ xs = interpret\text{-floatarith } fa \ (interpret\text{-slp } slp \ xs) \# interpret\text{-slp } slp \ xs$   
⟨proof⟩

**lemma**

*binop-slp-of-fa-induction-step*:

**assumes**

*Binop-IH1*:

$\bigwedge M \ slp \ M' \ slp'. \ slp\text{-of-fa } fa1 \ M \ slp = (M', \ slp') \implies$   
 $(\bigwedge f. f \in Mapping.keys \ M \implies subterms \ f \subseteq Mapping.keys \ M) \implies$   
 $(\bigwedge f. f \in Mapping.keys \ M \implies the \ (Mapping.lookup \ M \ f) < length \ slp) \implies$   
 $(\bigwedge f. f \in Mapping.keys \ M \implies interpret\text{-slp } slp \ xs \ ! \ slp\text{-index-lookup } slp \ M \ f =$   
 $interpret\text{-floatarith } f \ xs) \implies$   
 $subterms \ fa1 \subseteq Mapping.keys \ M' \wedge$   
 $Mapping.keys \ M \subseteq Mapping.keys \ M' \wedge$   
 $(\forall f \in Mapping.keys \ M'. subterms \ f \subseteq Mapping.keys \ M' \wedge$   
 $the \ (Mapping.lookup \ M' \ f) < length \ slp' \wedge$   
 $interpret\text{-slp } slp' \ xs \ ! \ slp\text{-index-lookup } slp' \ M' \ f = interpret\text{-floatarith } f \ xs)$

**and**

*Binop-IH2*:

$\bigwedge M \ slp \ M' \ slp'. \ slp\text{-of-fa } fa2 \ M \ slp = (M', \ slp') \implies$   
 $(\bigwedge f. f \in Mapping.keys \ M \implies subterms \ f \subseteq Mapping.keys \ M) \implies$   
 $(\bigwedge f. f \in Mapping.keys \ M \implies the \ (Mapping.lookup \ M \ f) < length \ slp) \implies$   
 $(\bigwedge f. f \in Mapping.keys \ M \implies interpret\text{-slp } slp \ xs \ ! \ slp\text{-index-lookup } slp \ M \ f =$   
 $interpret\text{-floatarith } f \ xs) \implies$   
 $subterms \ fa2 \subseteq Mapping.keys \ M' \wedge$   
 $Mapping.keys \ M \subseteq Mapping.keys \ M' \wedge$   
 $(\forall f \in Mapping.keys \ M'. subterms \ f \subseteq Mapping.keys \ M' \wedge$   
 $the \ (Mapping.lookup \ M' \ f) < length \ slp' \wedge$   
 $interpret\text{-slp } slp' \ xs \ ! \ slp\text{-index-lookup } slp' \ M' \ f = interpret\text{-floatarith } f \ xs)$

**and** *Binop-prems*:

$(case \ slp\text{-of-fa } fa1 \ M \ slp \ of$   
 $(M1, \ slp1) \Rightarrow$   
 $case \ slp\text{-of-fa } fa2 \ M1 \ slp1 \ of \ (x, \ xa) \Rightarrow slp\text{-of-fa-bin } Binop \ fa1 \ fa2 \ M \ slp \ x$   
 $xa) = (M', \ slp')$   
 $\bigwedge f. f \in Mapping.keys \ M \implies subterms \ f \subseteq Mapping.keys \ M$   
 $\bigwedge f. f \in Mapping.keys \ M \implies the \ (Mapping.lookup \ M \ f) < length \ slp$   
 $\bigwedge f. f \in Mapping.keys \ M \implies interpret\text{-slp } slp \ xs \ ! \ slp\text{-index-lookup } slp \ M \ f =$   
 $interpret\text{-floatarith } f \ xs$

**assumes** *subterms-Binop[simp]*:

$\bigwedge a \ b. subterms \ (Binop \ a \ b) = insert \ (Binop \ a \ b) \ (subterms \ a \cup subterms \ b)$

**assumes** *interpret-Binop[simp]*:

$\bigwedge a \ b \ xs. interpret\text{-floatarith } (Binop \ a \ b) \ xs = binop \ (interpret\text{-floatarith } a \ xs)$   
 $(interpret\text{-floatarith } b \ xs)$

**shows**  $\text{insert } (\text{Binop } fa1 \ fa2) \ (\text{subterms } fa1 \cup \text{subterms } fa2) \subseteq \text{Mapping.keys } M'$   
 $\wedge$   
 $\text{Mapping.keys } M \subseteq \text{Mapping.keys } M' \wedge$   
 $(\forall f \in \text{Mapping.keys } M'. \text{subterms } f \subseteq \text{Mapping.keys } M' \wedge$   
 $\text{the } (\text{Mapping.lookup } M' \ f) < \text{length } slp' \wedge$   
 $\text{interpret-slp } slp' \ xs \ ! \ \text{slp-index-lookup } slp' \ M' \ f = \text{interpret-floatarith } f \ xs)$   
 $\langle \text{proof} \rangle$

**lemma**

*unop-slp-of-fa-induction-step:*

**assumes**

*Unop-IH1:*

$\wedge M \ \text{slp} \ M' \ \text{slp}'. \ \text{slp-of-fa } fa1 \ M \ \text{slp} = (M', \ \text{slp}') \implies$

$(\wedge f. \ f \in \text{Mapping.keys } M \implies \text{subterms } f \subseteq \text{Mapping.keys } M) \implies$

$(\wedge f. \ f \in \text{Mapping.keys } M \implies \text{the } (\text{Mapping.lookup } M \ f) < \text{length } \text{slp}) \implies$

$(\wedge f. \ f \in \text{Mapping.keys } M \implies \text{interpret-slp } \text{slp} \ xs \ ! \ \text{slp-index-lookup } \text{slp} \ M \ f =$   
 $\text{interpret-floatarith } f \ xs) \implies$

$\text{subterms } fa1 \subseteq \text{Mapping.keys } M' \wedge$

$\text{Mapping.keys } M \subseteq \text{Mapping.keys } M' \wedge$

$(\forall f \in \text{Mapping.keys } M'. \text{subterms } f \subseteq \text{Mapping.keys } M' \wedge$

$\text{the } (\text{Mapping.lookup } M' \ f) < \text{length } \text{slp}' \wedge$

$\text{interpret-slp } \text{slp}' \ xs \ ! \ \text{slp-index-lookup } \text{slp}' \ M' \ f = \text{interpret-floatarith } f \ xs)$

**and** *Unop-prems:*

$(\text{case } \text{slp-of-fa } fa1 \ M \ \text{slp} \ \text{of } (M1, \ \text{slp1}) \Rightarrow \text{slp-of-fa-un } \text{Unop } fa1 \ M \ \text{slp} \ M1 \ \text{slp1})$

$= (M', \ \text{slp}')$

$\wedge f. \ f \in \text{Mapping.keys } M \implies \text{subterms } f \subseteq \text{Mapping.keys } M$

$\wedge f. \ f \in \text{Mapping.keys } M \implies \text{the } (\text{Mapping.lookup } M \ f) < \text{length } \text{slp}$

$\wedge f. \ f \in \text{Mapping.keys } M \implies \text{interpret-slp } \text{slp} \ xs \ ! \ \text{slp-index-lookup } \text{slp} \ M \ f =$

$\text{interpret-floatarith } f \ xs$

**assumes** *subterms-Unop[simp]:*

$\wedge a \ b. \ \text{subterms } (\text{Unop } a) = \text{insert } (\text{Unop } a) \ (\text{subterms } a)$

**assumes** *interpret-Unop[simp]:*

$\wedge a \ b \ xs. \ \text{interpret-floatarith } (\text{Unop } a) \ xs = \text{unop } (\text{interpret-floatarith } a \ xs)$

**shows**  $\text{insert } (\text{Unop } fa1) \ (\text{subterms } fa1) \subseteq \text{Mapping.keys } M' \wedge$

$\text{Mapping.keys } M \subseteq \text{Mapping.keys } M' \wedge$

$(\forall f \in \text{Mapping.keys } M'. \text{subterms } f \subseteq \text{Mapping.keys } M' \wedge$

$\text{the } (\text{Mapping.lookup } M' \ f) < \text{length } \text{slp}' \wedge$

$\text{interpret-slp } \text{slp}' \ xs \ ! \ \text{slp-index-lookup } \text{slp}' \ M' \ f = \text{interpret-floatarith } f \ xs)$

$\langle \text{proof} \rangle$

**lemma**

*cnst-slp-of-fa-induction-step:*

**assumes** *\**:

$\text{slp-of-fa-cnst } \text{Unop } \text{Unop}' \ M \ \text{slp} = (M', \ \text{slp}')$

$\wedge f. \ f \in \text{Mapping.keys } M \implies \text{subterms } f \subseteq \text{Mapping.keys } M$

$\wedge f. \ f \in \text{Mapping.keys } M \implies \text{the } (\text{Mapping.lookup } M \ f) < \text{length } \text{slp}$

$\wedge f. \ f \in \text{Mapping.keys } M \implies \text{interpret-slp } \text{slp} \ xs \ ! \ \text{slp-index-lookup } \text{slp} \ M \ f =$

$\text{interpret-floatarith } f \ xs$

**assumes** *subterms-Unop[simp]:*

$\wedge a b. \text{subterms } (Unop) = \{Unop\}$   
**assumes** *interpret-Unop[simp]*:  
 $\text{interpret-floatarith } Unop \ xs = unop \ xs$   
 $\text{interpret-floatarith } Unop' \ (\text{interpret-slp } slp \ xs) = unop \ xs$   
**assumes** *ui*:  $unop \ (\text{interpret-slp } slp \ xs) = unop \ xs$   
**shows**  $\{Unop\} \subseteq \text{Mapping.keys } M' \wedge$   
 $\text{Mapping.keys } M \subseteq \text{Mapping.keys } M' \wedge$   
 $(\forall f \in \text{Mapping.keys } M'. \text{subterms } f \subseteq \text{Mapping.keys } M' \wedge$   
 $\text{the } (\text{Mapping.lookup } M' \ f) < \text{length } slp' \wedge$   
 $\text{interpret-slp } slp' \ xs \ ! \ \text{slp-index-lookup } slp' \ M' \ f = \text{interpret-floatarith } f \ xs)$   
 <proof>

**lemma** *interpret-slp-nth*:  
 $n \geq \text{length } slp \implies \text{interpret-slp } slp \ xs \ ! \ n = xs \ ! \ (n - \text{length } slp)$   
 <proof>

**theorem**  
*interpret-slp-of-fa*:  
**assumes**  $slp\text{-of-fa } fa \ M \ slp = (M', \ slp')$   
**assumes**  $\wedge f. f \in \text{Mapping.keys } M \implies \text{subterms } f \subseteq \text{Mapping.keys } M$   
**assumes**  $\wedge f. f \in \text{Mapping.keys } M \implies (\text{the } (\text{Mapping.lookup } M \ f)) < \text{length } slp$   
**assumes**  $\wedge f. f \in \text{Mapping.keys } M \implies \text{interpret-slp } slp \ xs \ ! \ \text{slp-index-lookup } slp$   
 $M \ f = \text{interpret-floatarith } f \ xs$   
**shows**  $\text{subterms } fa \subseteq \text{Mapping.keys } M' \wedge \text{Mapping.keys } M \subseteq \text{Mapping.keys } M'$   
 $\wedge$   
 $(\forall f \in \text{Mapping.keys } M'. \text{subterms } f \subseteq \text{Mapping.keys } M' \wedge$   
 $\text{the } (\text{Mapping.lookup } M' \ f) < \text{length } slp' \wedge$   
 $(\text{interpret-slp } slp' \ xs \ ! \ \text{slp-index-lookup } slp' \ M' \ f = \text{interpret-floatarith } f \ xs))$   
 <proof>

**primrec** *slp-of-fas'* **where**  
 $slp\text{-of-fas}' \ [] \ M \ slp = (M, \ slp)$   
 $| \ slp\text{-of-fas}' \ (fa \ \#\ fas) \ M \ slp = (\text{let } (M, \ slp) = \text{slp-of-fa } fa \ M \ slp \ \text{in } \text{slp-of-fas}' \ fas \ M \ slp)$

**theorem**  
*interpret-slp-of-fas'*:  
**assumes**  $slp\text{-of-fas}' \ fas \ M \ slp = (M', \ slp')$   
**assumes**  $\wedge f. f \in \text{Mapping.keys } M \implies \text{subterms } f \subseteq \text{Mapping.keys } M$   
**assumes**  $\wedge f. f \in \text{Mapping.keys } M \implies \text{the } (\text{Mapping.lookup } M \ f) < \text{length } slp$   
**assumes**  $\wedge f. f \in \text{Mapping.keys } M \implies \text{interpret-slp } slp \ xs \ ! \ \text{slp-index-lookup } slp$   
 $M \ f = \text{interpret-floatarith } f \ xs$   
**shows**  $\bigcup (\text{subterms } \text{' set } fas) \subseteq \text{Mapping.keys } M' \wedge \text{Mapping.keys } M \subseteq \text{Mapping.keys } M' \wedge$   
 $(\forall f \in \text{Mapping.keys } M'. \text{subterms } f \subseteq \text{Mapping.keys } M' \wedge$   
 $(\text{the } (\text{Mapping.lookup } M' \ f) < \text{length } slp') \wedge$   
 $(\text{interpret-slp } slp' \ xs \ ! \ \text{slp-index-lookup } slp' \ M' \ f = \text{interpret-floatarith } f \ xs))$   
 <proof>

**definition** *slp-of-fas* *fas* =  
 (let  
   (*M*, *slp*) = *slp-of-fas'* *fas* *Mapping.empty* [];  
   *fasi* = map (the o *Mapping.lookup* *M*) *fas*;  
   *fasi'* = map ( $\lambda(a, b). \text{Var } (\text{length } \textit{slp} + a - \text{Suc } b)$ ) (zip [0..*length* *fasi*] (rev *fasi*))  
 in *slp* @ *fasi'*)

**lemma** *length-interpret-slp[simp]*:  
*length* (*interpret-slp* *slp* *xs*) = *length* *slp* + *length* *xs*  
 <proof>

**lemma** *length-interpret-floatariths[simp]*:  
*length* (*interpret-floatariths* *slp* *xs*) = *length* *slp*  
 <proof>

**lemma** *interpret-slp-append[simp]*:  
*interpret-slp* (*slp1* @ *slp2*) *xs* =  
*interpret-slp* *slp2* (*interpret-slp* *slp1* *xs*)  
 <proof>

**lemma** *interpret-slp* (map *Var* [*a* + 0, *b* + 1, *c* + 2, *d* + 3]) *xs* =  
 (rev (map ( $\lambda(i, e). \text{xs } ! (e - i)$ ) (zip [0..*length* [0, 1, 2, 3]]))) @ *xs*  
 <proof>

**lemma** *aC-eq-aa*: *xs* @ *y* # *zs* = (*xs* @ [*y*]) @ *zs*  
 <proof>

**lemma**  
*interpret-slp-map-Var*:  
**assumes**  $\bigwedge i. i < \text{length } \textit{is} \implies \textit{is } ! i \geq i$   
**assumes**  $\bigwedge i. i < \text{length } \textit{is} \implies (\textit{is } ! i - i) < \text{length } \textit{xs}$   
**shows** *interpret-slp* (map *Var* *is*) *xs* =  
 (rev (map ( $\lambda(i, e). \text{xs } ! (e - i)$ ) (zip [0..*length* *is*] *is*)))  
 @  
*xs*  
 <proof>

**theorem** *slp-of-fas*:  
 take (*length* *fas*) (*interpret-slp* (*slp-of-fas* *fas*) *xs*) = *interpret-floatariths* *fas* *xs*  
 <proof>

### 4.3 better code equations for construction of large programs

**definition** *slp-indexl* *slpl* *i* = *slpl* - *Suc* *i*

**definition** *slp-indexl-lookup* *vsl* *M* *a* = *slp-indexl* *vsl* (the (*Mapping.lookup* *M* *a*))

**definition**

$slp\text{-of-fa-rev-bin } Binop\ a\ b\ M\ slp\ slpl\ M2\ slp2\ slpl2 =$   
 $(case\ Mapping.lookup\ M\ (Binop\ a\ b)\ of$   
 $\quad Some\ i \Rightarrow (Mapping.update\ (Binop\ a\ b)\ (slpl)\ M,\ Var\ (slp\text{-indexl}\ slpl\ i)\#slp,$   
 $\quad Suc\ slpl)$   
 $\quad | None \Rightarrow (Mapping.update\ (Binop\ a\ b)\ (slpl2)\ M2,$   
 $\quad\quad Binop\ (Var\ (slp\text{-indexl-lookup}\ slpl2\ M2\ a))\ (Var\ (slp\text{-indexl-lookup}$   
 $\quad\quad slpl2\ M2\ b))\#slp2,$   
 $\quad\quad Suc\ slpl2))$

**definition**

$slp\text{-of-fa-rev-un } Unop\ a\ M\ slp\ slpl\ M1\ slp1\ slpl1 =$   
 $(case\ Mapping.lookup\ M\ (Unop\ a)\ of$   
 $\quad Some\ i \Rightarrow (Mapping.update\ (Unop\ a)\ (slpl)\ M,\ Var\ (slp\text{-indexl}\ slpl\ i)\#slp,$   
 $\quad Suc\ slpl)$   
 $\quad | None \Rightarrow (Mapping.update\ (Unop\ a)\ (slpl1)\ M1,$   
 $\quad\quad Unop\ (Var\ (slp\text{-indexl-lookup}\ slpl1\ M1\ a))\#slp1,\ Suc\ slpl1))$

**definition**

$slp\text{-of-fa-rev-cnst } Const\ Const'\ M\ vs\ vsl =$   
 $(Mapping.update\ Const\ vsl\ M,$   
 $\quad (case\ Mapping.lookup\ M\ Const\ of\ Some\ i \Rightarrow Var\ (slp\text{-indexl}\ vsl\ i)\ | None \Rightarrow$   
 $\quad Const')\#vs,\ Suc\ vsl)$

**fun**  $slp\text{-of-fa-rev} :: floatarith \Rightarrow (floatarith, nat) mapping \Rightarrow floatarith list \Rightarrow nat \Rightarrow$

$(floatarith, nat) mapping \times floatarith list \times nat) \mathbf{where}$   
 $slp\text{-of-fa-rev } (Add\ a\ b)\ M\ slp\ slpl =$   
 $(let\ (M1,\ slp1,\ slpl1) = slp\text{-of-fa-rev } a\ M\ slp\ slpl;\ (M2,\ slp2,\ slpl2) = slp\text{-of-fa-rev}$   
 $b\ M1\ slp1\ slpl1\ in$   
 $\quad slp\text{-of-fa-rev-bin } Add\ a\ b\ M\ slp\ slpl\ M2\ slp2\ slpl2)$   
 $| slp\text{-of-fa-rev } (Mult\ a\ b)\ M\ slp\ slpl =$   
 $(let\ (M1,\ slp1,\ slpl1) = slp\text{-of-fa-rev } a\ M\ slp\ slpl;\ (M2,\ slp2,\ slpl2) = slp\text{-of-fa-rev}$   
 $b\ M1\ slp1\ slpl1\ in$   
 $\quad slp\text{-of-fa-rev-bin } Mult\ a\ b\ M\ slp\ slpl\ M2\ slp2\ slpl2)$   
 $| slp\text{-of-fa-rev } (Min\ a\ b)\ M\ slp\ slpl =$   
 $(let\ (M1,\ slp1,\ slpl1) = slp\text{-of-fa-rev } a\ M\ slp\ slpl;\ (M2,\ slp2,\ slpl2) = slp\text{-of-fa-rev}$   
 $b\ M1\ slp1\ slpl1\ in$   
 $\quad slp\text{-of-fa-rev-bin } Min\ a\ b\ M\ slp\ slpl\ M2\ slp2\ slpl2)$   
 $| slp\text{-of-fa-rev } (Max\ a\ b)\ M\ slp\ slpl =$   
 $(let\ (M1,\ slp1,\ slpl1) = slp\text{-of-fa-rev } a\ M\ slp\ slpl;\ (M2,\ slp2,\ slpl2) = slp\text{-of-fa-rev}$   
 $b\ M1\ slp1\ slpl1\ in$   
 $\quad slp\text{-of-fa-rev-bin } Max\ a\ b\ M\ slp\ slpl\ M2\ slp2\ slpl2)$   
 $| slp\text{-of-fa-rev } (Powr\ a\ b)\ M\ slp\ slpl =$   
 $(let\ (M1,\ slp1,\ slpl1) = slp\text{-of-fa-rev } a\ M\ slp\ slpl;\ (M2,\ slp2,\ slpl2) = slp\text{-of-fa-rev}$   
 $b\ M1\ slp1\ slpl1\ in$   
 $\quad slp\text{-of-fa-rev-bin } Powr\ a\ b\ M\ slp\ slpl\ M2\ slp2\ slpl2)$   
 $| slp\text{-of-fa-rev } (Inverse\ a)\ M\ slp\ slpl =$   
 $(let\ (M1,\ slp1,\ slpl1) = slp\text{-of-fa-rev } a\ M\ slp\ slpl\ in\ slp\text{-of-fa-rev-un } Inverse\ a\ M$   
 $slp\ slpl\ M1\ slp1\ slpl1)$

$| \text{slp-of-fa-rev } (\text{Cos } a) M \text{ slp slpl} =$   
 $(\text{let } (M1, \text{slp1}, \text{slpl1}) = \text{slp-of-fa-rev } a M \text{ slp slpl} \text{ in } \text{slp-of-fa-rev-un } \text{Cos } a M \text{ slp}$   
 $\text{slpl } M1 \text{ slp1 slpl1})$   
 $| \text{slp-of-fa-rev } (\text{Arctan } a) M \text{ slp slpl} =$   
 $(\text{let } (M1, \text{slp1}, \text{slpl1}) = \text{slp-of-fa-rev } a M \text{ slp slpl} \text{ in } \text{slp-of-fa-rev-un } \text{Arctan } a M$   
 $\text{slp slpl } M1 \text{ slp1 slpl1})$   
 $| \text{slp-of-fa-rev } (\text{Abs } a) M \text{ slp slpl} =$   
 $(\text{let } (M1, \text{slp1}, \text{slpl1}) = \text{slp-of-fa-rev } a M \text{ slp slpl} \text{ in } \text{slp-of-fa-rev-un } \text{Abs } a M \text{ slp}$   
 $\text{slpl } M1 \text{ slp1 slpl1})$   
 $| \text{slp-of-fa-rev } (\text{Sqrt } a) M \text{ slp slpl} =$   
 $(\text{let } (M1, \text{slp1}, \text{slpl1}) = \text{slp-of-fa-rev } a M \text{ slp slpl} \text{ in } \text{slp-of-fa-rev-un } \text{Sqrt } a M \text{ slp}$   
 $\text{slpl } M1 \text{ slp1 slpl1})$   
 $| \text{slp-of-fa-rev } (\text{Exp } a) M \text{ slp slpl} =$   
 $(\text{let } (M1, \text{slp1}, \text{slpl1}) = \text{slp-of-fa-rev } a M \text{ slp slpl} \text{ in } \text{slp-of-fa-rev-un } \text{Exp } a M \text{ slp}$   
 $\text{slpl } M1 \text{ slp1 slpl1})$   
 $| \text{slp-of-fa-rev } (\text{Ln } a) M \text{ slp slpl} =$   
 $(\text{let } (M1, \text{slp1}, \text{slpl1}) = \text{slp-of-fa-rev } a M \text{ slp slpl} \text{ in } \text{slp-of-fa-rev-un } \text{Ln } a M \text{ slp}$   
 $\text{slpl } M1 \text{ slp1 slpl1})$   
 $| \text{slp-of-fa-rev } (\text{Minus } a) M \text{ slp slpl} =$   
 $(\text{let } (M1, \text{slp1}, \text{slpl1}) = \text{slp-of-fa-rev } a M \text{ slp slpl} \text{ in } \text{slp-of-fa-rev-un } \text{Minus } a M$   
 $\text{slp slpl } M1 \text{ slp1 slpl1})$   
 $| \text{slp-of-fa-rev } (\text{Floor } a) M \text{ slp slpl} =$   
 $(\text{let } (M1, \text{slp1}, \text{slpl1}) = \text{slp-of-fa-rev } a M \text{ slp slpl} \text{ in } \text{slp-of-fa-rev-un } \text{Floor } a M$   
 $\text{slp slpl } M1 \text{ slp1 slpl1})$   
 $| \text{slp-of-fa-rev } (\text{Power } a n) M \text{ slp slpl} =$   
 $(\text{let } (M1, \text{slp1}, \text{slpl1}) = \text{slp-of-fa-rev } a M \text{ slp slpl} \text{ in } \text{slp-of-fa-rev-un } (\lambda a. \text{Power}$   
 $a n) a M \text{ slp slpl } M1 \text{ slp1 slpl1})$   
 $| \text{slp-of-fa-rev } \text{Pi } M \text{ slp slpl} = \text{slp-of-fa-rev-cnst } \text{Pi } \text{Pi } M \text{ slp slpl}$   
 $| \text{slp-of-fa-rev } (\text{Var } v) M \text{ slp slpl} = \text{slp-of-fa-rev-cnst } (\text{Var } v) (\text{Var } (v + \text{slpl})) M$   
 $\text{slp slpl}$   
 $| \text{slp-of-fa-rev } (\text{Num } n) M \text{ slp slpl} = \text{slp-of-fa-rev-cnst } (\text{Num } n) (\text{Num } n) M \text{ slp slpl}$

**lemma** *slp-index-length[simp]*:  $\text{slp-indexl } (\text{length } xs) i = \text{slp-index } xs i$   
 $\langle \text{proof} \rangle$

**lemma** *slp-indexl-lookup-length[simp]*:  $\text{slp-indexl-lookup } (\text{length } xs) i = \text{slp-index-lookup}$   
 $xs i$   
 $\langle \text{proof} \rangle$

**lemma** *slp-index-rev[simp]*:  $\text{slp-index } (\text{rev } xs) i = \text{slp-index } xs i$   
 $\langle \text{proof} \rangle$

**lemma** *slp-index-lookup-rev[simp]*:  $\text{slp-index-lookup } (\text{rev } xs) i = \text{slp-index-lookup}$   
 $xs i$   
 $\langle \text{proof} \rangle$

**lemma** *slp-of-fa-bin-slp-of-fa-rev-bin*:  
 $\text{slp-of-fa-rev-bin } \text{Binop } a b M \text{ slp } (\text{length } \text{slp}) M2 \text{ slp2 } (\text{length } \text{slp2}) =$   
 $(\text{let } (M, \text{slp}') = \text{slp-of-fa-bin } \text{Binop } a b M (\text{rev } \text{slp}) M2 (\text{rev } \text{slp2}) \text{ in } (M, \text{rev}$

*slp'*, *length slp'*)  
 ⟨*proof*⟩

**lemma** *slp-of-fa-un-slp-of-fa-rev-un*:

*slp-of-fa-rev-un Binop a M slp (length slp) M2 slp2 (length slp2) =*  
*(let (M, slp') = slp-of-fa-un Binop a M (rev slp) M2 (rev slp2) in (M, rev slp'*,  
*length slp'))*  
 ⟨*proof*⟩

**lemma** *slp-of-fa-cnst-slp-of-fa-rev-cnst*:

*slp-of-fa-rev-cnst Cnst Cnst' M slp (length slp) =*  
*(let (M, slp') = slp-of-fa-cnst Cnst Cnst' M (rev slp) in (M, rev slp'*, *length*  
*slp'))*  
 ⟨*proof*⟩

**lemma** *slp-of-fa-rev*:

*slp-of-fa-rev fa M slp (length slp) = (let (M, slp') = slp-of-fa fa M (rev slp) in*  
*(M, rev slp', length slp'))*  
 ⟨*proof*⟩

**lemma** *slp-of-fa-code*[*code*]:

*slp-of-fa fa M slp = (let (M, slp', -) = slp-of-fa-rev fa M (rev slp) (length slp) in*  
*(M, rev slp'))*  
 ⟨*proof*⟩

**definition** *norm2-slp n = slp-of-fas [floatarith.Inverse (norm2<sub>e</sub> n)]*

**unbundle** *no floatarith-syntax*

**end**

## 5 Approximation with Affine Forms

**theory** *Affine-Approximation*

**imports**

*HOL-Decision-Procs.Approximation*

*HOL-Library.Monad-Syntax*

*HOL-Library.Mapping*

*Executable-Euclidean-Space*

*Affine-Form*

*Straight-Line-Program*

**begin**

**lemma** *convex-on-imp-above-tangent*:— **TODO**: generalizes  $\llbracket \text{convex-on } ?A \text{ } ?f; \text{ connected } ?A; ?c \in \text{interior } ?A; ?x \in ?A; (?f \text{ has-real-derivative } ?f') \text{ (at } ?c \text{ within } ?A) \rrbracket$   
 $\implies ?f' * (?x - ?c) \leq ?f ?x - ?f ?c$

**assumes** *convex*: *convex-on A f* **and** *connected*: *connected A*

**assumes** *c*: *c ∈ A* **and** *x*: *x ∈ A*

**assumes** *deriv*: (*f has-field-derivative f'*) (at *c* within *A*)

**shows**  $f x - f c \geq f' * (x - c)$   
*<proof>*

Approximate operations on affine forms.

**lemma** *Affine-notempty*[*intro, simp*]: *Affine*  $X \neq \{\}$   
*<proof>*

**lemma** *truncate-up-lt*:  $x < y \implies x < \text{truncate-up } \text{prec } y$   
*<proof>*

**lemma** *truncate-up-pos-eq*[*simp*]:  $0 < \text{truncate-up } p \ x \iff 0 < x$   
*<proof>*

**lemma** *inner-scaleR-pdevs-0*: *inner-scaleR-pdevs* 0 *One-pdevs* = *zero-pdevs*  
*<proof>*

**lemma** *Affine-aform-of-point-eq*[*simp*]: *Affine* (*aform-of-point*  $p$ ) =  $\{p\}$   
*<proof>*

**lemma** *mem-Affine-aform-of-point*:  $x \in \text{Affine } (\text{aform-of-point } x)$   
*<proof>*

**lemma**  
*aform-val-aform-of-ivl-innerE*:  
**assumes**  $e \in \text{UNIV} \rightarrow \{-1 .. 1\}$   
**assumes**  $a \leq b \ c \in \text{Basis}$   
**obtains**  $f$  **where** *aform-val*  $e$  (*aform-of-ivl*  $a \ b$ )  $\cdot c = \text{aform-val } f$  (*aform-of-ivl*  
 $(a \cdot c) (b \cdot c)$ )  
 $f \in \text{UNIV} \rightarrow \{-1 .. 1\}$   
*<proof>*

**lift-definition** *coord-pdevs::nat*  $\Rightarrow$  *real pdevs* **is**  $\lambda n \ i.$  *if*  $i = n$  *then* 1 *else* 0 *<proof>*

**lemma** *pdevs-apply-coord-pdevs* [*simp*]: *pdevs-apply* (*coord-pdevs*  $i$ )  $x =$  (*if*  $x = i$   
*then* 1 *else* 0)  
*<proof>*

**lemma** *degree-coord-pdevs*[*simp*]: *degree* (*coord-pdevs*  $i$ ) = *Suc*  $i$   
*<proof>*

**lemma** *pdevs-val-coord-pdevs*[*simp*]: *pdevs-val*  $e$  (*coord-pdevs*  $i$ ) =  $e \ i$   
*<proof>*

**definition** *aforms-of-ivls*  $ls \ us = \text{map}$   
 $(\lambda(i, (l, u)). ((l + u)/2, \text{scaleR-pdevs } ((u - l)/2) (\text{coord-pdevs } i)))$   
 $(\text{zip } [0..<\text{length } ls] (\text{zip } ls \ us))$

**lemma**  
*aforms-of-ivls*:

**assumes**  $\text{length } ls = \text{length } us \text{ length } xs = \text{length } ls$   
**assumes**  $\bigwedge i. i < \text{length } xs \implies xs ! i \in \{ls ! i .. us ! i\}$   
**shows**  $xs \in \text{Joints } (\text{aforms-of-ivls } ls \ us)$   
 <proof>

## 5.1 Approximate Operations

**definition**  $\text{max-pdev } x = \text{fold } (\lambda x y. \text{if } \text{infnorm } (\text{snd } x) \geq \text{infnorm } (\text{snd } y) \text{ then } x \text{ else } y) (\text{list-of-pdevs } x) (0, 0)$

### 5.1.1 set of generated endpoints

**fun**  $\text{points-of-list}$  **where**

$\text{points-of-list } x0 \ [] = [x0]$   
 $|\ \text{points-of-list } x0 \ ((i, x)\#xs) = (\text{points-of-list } (x0 + x) \ xs \ @ \ \text{points-of-list } (x0 - x) \ xs)$

**primrec**  $\text{points-of-aform}$  **where**

$\text{points-of-aform } (x, xs) = \text{points-of-list } x \ (\text{list-of-pdevs } xs)$

### 5.1.2 Approximate total deviation

**definition**  $\text{sum-list}'::\text{nat} \Rightarrow 'a \ \text{list} \Rightarrow 'a::\text{executable-euclidean-space}$   
**where**  $\text{sum-list}' \ p \ xs = \text{fold } (\lambda a \ b. \text{eucl-truncate-up } p \ (a + b)) \ xs \ 0$

**definition**  $\text{tdev}' \ p \ x = \text{sum-list}' \ p \ (\text{map } (\text{abs } o \ \text{snd}) \ (\text{list-of-pdevs } x))$

**lemma**

*eucl-fold-mono:*

**fixes**  $f::'a::\text{ordered-euclidean-space} \Rightarrow 'a \Rightarrow 'a$

**assumes**  $\text{mono}: \bigwedge w \ x \ y \ z. w \leq x \implies y \leq z \implies f \ w \ y \leq f \ x \ z$

**shows**  $x \leq y \implies \text{fold } f \ xs \ x \leq \text{fold } f \ xs \ y$

<proof>

**lemma** *sum-list-add-le-fold-eucl-truncate-up:*

**fixes**  $z::'a::\text{executable-euclidean-space}$

**shows**  $\text{sum-list } xs + z \leq \text{fold } (\lambda x \ y. \text{eucl-truncate-up } p \ (x + y)) \ xs \ z$

<proof>

**lemma** *sum-list-le-sum-list':*

$\text{sum-list } xs \leq \text{sum-list}' \ p \ xs$

<proof>

**lemma** *sum-list'-sum-list-le:*

$y \leq \text{sum-list } xs \implies y \leq \text{sum-list}' \ p \ xs$

<proof>

**lemma** *tdev':*  $\text{tdev } x \leq \text{tdev}' \ p \ x$

<proof>

**lemma** *tdev'-le*:  $x \leq tdev\ y \implies x \leq tdev'\ p\ y$   
*<proof>*

**lemmas** *abs-pdevs-val-le-tdev'* = *tdev'-le*[*OF abs-pdevs-val-le-tdev'*]

**lemma** *tdev'-uminus-pdevs[simp]*:  $tdev'\ p\ (uminus\ pdevs\ x) = tdev'\ p\ x$   
*<proof>*

**abbreviation** *Radius::'a::ordered-euclidean-space aform*  $\Rightarrow 'a$   
**where** *Radius*  $X \equiv tdev\ (snd\ X)$

**abbreviation** *Radius'::nat $\Rightarrow$ 'a::executable-euclidean-space aform*  $\Rightarrow 'a$   
**where** *Radius'*  $p\ X \equiv tdev'\ p\ (snd\ X)$

**lemma** *Radius'-uminus-aform[simp]*:  $Radius'\ p\ (uminus\ aform\ X) = Radius'\ p\ X$   
*<proof>*

### 5.1.3 truncate partial deviations

**definition** *trunc-pdevs-raw::nat*  $\Rightarrow (nat \Rightarrow 'a) \Rightarrow nat \Rightarrow 'a::executable-euclidean-space$   
**where** *trunc-pdevs-raw*  $p\ x\ i = eucl\ truncate\ down\ p\ (x\ i)$

**lemma** *nonzeros-trunc-pdevs-raw*:  
 $\{i. trunc\ pdevs\ raw\ r\ x\ i \neq 0\} \subseteq \{i. x\ i \neq 0\}$   
*<proof>*

**lift-definition** *trunc-pdevs::nat*  $\Rightarrow 'a::executable-euclidean-space\ pdevs \Rightarrow 'a\ pdevs$   
**is** *trunc-pdevs-raw*  
*<proof>*

**definition** *trunc-err-pdevs-raw::nat*  $\Rightarrow (nat \Rightarrow 'a) \Rightarrow nat \Rightarrow 'a::executable-euclidean-space$   
**where** *trunc-err-pdevs-raw*  $p\ x\ i = trunc\ pdevs\ raw\ p\ x\ i - x\ i$

**lemma** *nonzeros-trunc-err-pdevs-raw*:  
 $\{i. trunc\ err\ pdevs\ raw\ r\ x\ i \neq 0\} \subseteq \{i. x\ i \neq 0\}$   
*<proof>*

**lift-definition** *trunc-err-pdevs::nat*  $\Rightarrow 'a::executable-euclidean-space\ pdevs \Rightarrow 'a\ pdevs$   
**is** *trunc-err-pdevs-raw*  
*<proof>*

**term** *float-plus-down*

**lemma** *pdevs-apply-trunc-pdevs[simp]*:  
**fixes**  $x\ y::'a::euclidean-space$   
**shows**  $pdevs\ apply\ (trunc\ pdevs\ p\ X)\ n = eucl\ truncate\ down\ p\ (pdevs\ apply\ X\ n)$   
*<proof>*

**lemma** *pdevs-apply-trunc-err-pdevs[simp]*:  
**fixes**  $x y::'a::\text{euclidean-space}$   
**shows**  $\text{pdevs-apply } (\text{trunc-err-pdevs } p \ X) \ n =$   
 $\text{eucl-truncate-down } p \ (\text{pdevs-apply } X \ n) - (\text{pdevs-apply } X \ n)$   
 $\langle \text{proof} \rangle$

**lemma** *pdevs-val-trunc-pdevs*:  
**fixes**  $x y::'a::\text{euclidean-space}$   
**shows**  $\text{pdevs-val } e \ (\text{trunc-pdevs } p \ X) = \text{pdevs-val } e \ X + \text{pdevs-val } e \ (\text{trunc-err-pdevs } p \ X)$   
 $\langle \text{proof} \rangle$

**lemma** *pdevs-val-trunc-err-pdevs*:  
**fixes**  $x y::'a::\text{euclidean-space}$   
**shows**  $\text{pdevs-val } e \ (\text{trunc-err-pdevs } p \ X) = \text{pdevs-val } e \ (\text{trunc-pdevs } p \ X) - \text{pdevs-val } e \ X$   
 $\langle \text{proof} \rangle$

**definition** *truncate-aform::nat  $\Rightarrow$  'a aform  $\Rightarrow$  'a::executable-euclidean-space aform*  
**where**  $\text{truncate-aform } p \ x = (\text{eucl-truncate-down } p \ (\text{fst } x), \text{trunc-pdevs } p \ (\text{snd } x))$

**definition** *truncate-error-aform::nat  $\Rightarrow$  'a aform  $\Rightarrow$  'a::executable-euclidean-space aform*  
**where**  $\text{truncate-error-aform } p \ x =$   
 $(\text{eucl-truncate-down } p \ (\text{fst } x) - \text{fst } x, \text{trunc-err-pdevs } p \ (\text{snd } x))$

**lemma**  
*abs-aform-val-le*:  
**assumes**  $e \in \text{UNIV} \rightarrow \{-1..1\}$   
**shows**  $\text{abs } (\text{aform-val } e \ X) \leq \text{eucl-truncate-up } p \ (|\text{fst } X| + \text{tdev}' p \ (\text{snd } X))$   
 $\langle \text{proof} \rangle$

#### 5.1.4 truncation with error bound

**definition** *trunc-bound-eucl*  $p \ s =$   
 $(\text{let}$   
 $\quad d = \text{eucl-truncate-down } p \ s;$   
 $\quad ed = \text{abs } (d - s) \text{ in}$   
 $(d, \text{eucl-truncate-up } p \ ed))$

**lemma** *trunc-bound-euclE*:  
**obtains** *err* **where**  
 $|\text{err}| \leq \text{snd } (\text{trunc-bound-eucl } p \ x)$   
 $\text{fst } (\text{trunc-bound-eucl } p \ x) = x + \text{err}$   
 $\langle \text{proof} \rangle$

**definition** *trunc-bound-pdevs*  $p \ x = (\text{trunc-pdevs } p \ x, \text{tdev}' p \ (\text{trunc-err-pdevs } p \ x))$

$x$ )

**lemma** *pdevs-apply-fst-trunc-bound-pdevs[simp]*:  $pdevs\text{-}apply\ (fst\ (trunc\text{-}bound\text{-}pdevs\ p\ x)) = pdevs\text{-}apply\ (trunc\text{-}pdevs\ p\ x)$   
*<proof>*

**lemma** *trunc-bound-pdevsE*:  
**assumes**  $e \in UNIV \rightarrow \{-1..1\}$   
**obtains**  $err$  **where**  
 $|err| \leq snd\ (trunc\text{-}bound\text{-}pdevs\ p\ x)$   
 $pdevs\text{-}val\ e\ (fst\ ((trunc\text{-}bound\text{-}pdevs\ p\ x))) = pdevs\text{-}val\ e\ x + err$   
*<proof>*

**lemma**  
*degree-add-pdevs-le*:  
**assumes**  $degree\ X \leq n$   
**assumes**  $degree\ Y \leq n$   
**shows**  $degree\ (add\text{-}pdevs\ X\ Y) \leq n$   
*<proof>*

**lemma** *truncate-aform-error-aform-cancel*:  
 $aform\text{-}val\ e\ (truncate\text{-}aform\ p\ z) = aform\text{-}val\ e\ z + aform\text{-}val\ e\ (truncate\text{-}error\text{-}aform\ p\ z)$   
*<proof>*

**lemma** *error-absE*:  
**assumes**  $abs\ err \leq k$   
**obtains**  $e::real$  **where**  $err = e * k$   $e \in \{-1..1\}$   
*<proof>*

**lemma** *eucl-truncate-up-nonneg-eq-zero-iff*:  
 $x \geq 0 \implies eucl\text{-}truncate\text{-}up\ p\ x = 0 \iff x = 0$   
*<proof>*

**lemma**  
*aform-val-consume-error*:  
**assumes**  $abs\ err \leq abs\ (pdevs\text{-}apply\ (snd\ X)\ n)$   
**shows**  $aform\text{-}val\ (e(n := 0))\ X + err = aform\text{-}val\ (e(n := err/pdevs\text{-}apply\ (snd\ X)\ n))\ X$   
*<proof>*

**lemma**  
*aform-val-consume-errorE*:  
**fixes**  $X::real\ aform$   
**assumes**  $abs\ err \leq abs\ (pdevs\text{-}apply\ (snd\ X)\ n)$   
**obtains**  $err'$  **where**  $aform\text{-}val\ (e(n := 0))\ X + err = aform\text{-}val\ (e(n := err'))$   
 $X\ err' \in \{-1..1\}$



**assumes**  $deg: degree\text{-}aforms [X, Y] \leq k$   
**assumes**  $e: e \in UNIV \rightarrow \{-1..1\}$   
**assumes**  $d: abs\ u \leq d$   
**obtains**  $ek$  **where**  
 $a * aform\text{-}val\ e\ X + b * aform\text{-}val\ e\ Y + c + u = aform\text{-}val\ (e(k:=ek))$   
 $(affine\text{-}binop'\ p\ X\ Y\ a\ b\ c\ d\ k)$   
 $ek \in \{-1 .. 1\}$   
 $\langle proof \rangle$

### 5.1.6 Inf/Sup

**definition**  $Inf\text{-}aform'\ p\ X = eucl\text{-}truncate\text{-}down\ p\ (fst\ X - tdev'\ p\ (snd\ X))$

**definition**  $Sup\text{-}aform'\ p\ X = eucl\text{-}truncate\text{-}up\ p\ (fst\ X + tdev'\ p\ (snd\ X))$

**lemma**  $Inf\text{-}aform'$ :

**shows**  $Inf\text{-}aform'\ p\ X \leq Inf\text{-}aform\ X$   
 $\langle proof \rangle$

**lemma**  $Sup\text{-}aform'$ :

**shows**  $Sup\text{-}aform\ X \leq Sup\text{-}aform'\ p\ X$   
 $\langle proof \rangle$

**lemma**  $Inf\text{-}aform\text{-}le\text{-}Sup\text{-}aform[intro]$ :

$Inf\text{-}aform\ X \leq Sup\text{-}aform\ X$   
 $\langle proof \rangle$

**lemma**  $Inf\text{-}aform'\text{-}le\text{-}Sup\text{-}aform'[intro]$ :

$Inf\text{-}aform'\ p\ X \leq Sup\text{-}aform'\ p\ X$   
 $\langle proof \rangle$

**definition**

$iws\text{-}of\text{-}aforms\ prec = map\ (\lambda a. Interval'\ (float\text{-}of\ (Inf\text{-}aform'\ prec\ a))\ (float\text{-}of\ (Sup\text{-}aform'\ prec\ a)))$

**lemma**

**assumes**  $\bigwedge i. e''\ i \leq 1$

**assumes**  $\bigwedge i. -1 \leq e''\ i$

**shows**  $Inf\text{-}aform'\text{-}le: Inf\text{-}aform'\ p\ r \leq aform\text{-}val\ e''\ r$   
**and**  $Sup\text{-}aform'\text{-}le: aform\text{-}val\ e''\ r \leq Sup\text{-}aform'\ p\ r$   
 $\langle proof \rangle$

**lemma**  $InfSup\text{-}aform'\text{-}in\text{-}float[intro, simp]$ :

$Inf\text{-}aform'\ p\ X \in float\ Sup\text{-}aform'\ p\ X \in float$   
 $\langle proof \rangle$

**theorem**  $iws\text{-}of\text{-}aforms: xs \in Joints\ XS \implies bounded\text{-}by\ xs\ (iws\text{-}of\text{-}aforms\ prec\ XS)$

*<proof>*

**definition** *isFDERIV-aform prec N xs fas AS = isFDERIV-approx prec N xs fas (ivs-of-aforms prec AS)*

**theorem** *isFDERIV-aform:*

**assumes** *isFDERIV-aform prec N xs fas AS*

**assumes** *vs ∈ Joints AS*

**shows** *isFDERIV N xs fas vs*

*<proof>*

**definition** *env-len env l = (∀ xs ∈ env. length xs = l)*

**lemma** *env-len-takeI: env-len xs d1 ⇒ d1 ≥ d ⇒ env-len (take d ' xs) d*

*<proof>*

## 5.2 Min Range approximation

**lemma**

*linear-lower:*

**fixes** *x::real*

**assumes**  $\bigwedge x. x \in \{a .. b\} \implies (f \text{ has-field-derivative } f' x) \text{ (at } x \text{ within } \{a .. b\})$

**assumes**  $\bigwedge x. x \in \{a .. b\} \implies f' x \leq u$

**assumes**  $x \in \{a .. b\}$

**shows**  $f b + u * (x - b) \leq f x$

*<proof>*

**lemma**

*linear-lower2:*

**fixes** *x::real*

**assumes**  $\bigwedge x. x \in \{a .. b\} \implies (f \text{ has-field-derivative } f' x) \text{ (at } x \text{ within } \{a .. b\})$

**assumes**  $\bigwedge x. x \in \{a .. b\} \implies l \leq f' x$

**assumes**  $x \in \{a .. b\}$

**shows**  $f x \geq f a + l * (x - a)$

*<proof>*

**lemma**

*linear-upper:*

**fixes** *x::real*

**assumes**  $\bigwedge x. x \in \{a .. b\} \implies (f \text{ has-field-derivative } f' x) \text{ (at } x \text{ within } \{a .. b\})$

**assumes**  $\bigwedge x. x \in \{a .. b\} \implies f' x \leq u$

**assumes**  $x \in \{a .. b\}$

**shows**  $f x \leq f a + u * (x - a)$

*<proof>*

**lemma**

*linear-upper2:*

**fixes** *x::real*

**assumes**  $\bigwedge x. x \in \{a .. b\} \implies (f \text{ has-field-derivative } f' x) \text{ (at } x \text{ within } \{a .. b\})$

**assumes**  $\bigwedge x. x \in \{a .. b\} \implies l \leq f' x$   
**assumes**  $x \in \{a .. b\}$   
**shows**  $f x \leq f b + l * (x - b)$   
 <proof>

**lemma** *linear-enclosure*:

**fixes**  $x::real$   
**assumes**  $\bigwedge x. x \in \{a .. b\} \implies (f \text{ has-field-derivative } f' x) \text{ (at } x \text{ within } \{a .. b\})$   
**assumes**  $\bigwedge x. x \in \{a .. b\} \implies f' x \leq u$   
**assumes**  $x \in \{a .. b\}$   
**shows**  $f x \in \{f b + u * (x - b) .. f a + u * (x - a)\}$   
 <proof>

**definition** *mid-err ivl* =  $((\text{lower ivl} + \text{upper ivl}::\text{float})/2, (\text{upper ivl} - \text{lower ivl})/2)$

**lemma** *degree-aform-uminus-aform[simp]*: *degree-aform (uminus-aform X) = degree-aform X*  
 <proof>

### 5.2.1 Addition

**definition** *add-aform*:: $'a::real\text{-vector } aform \Rightarrow 'a aform \Rightarrow 'a aform$   
**where** *add-aform*  $x y = (fst x + fst y, add-pdevs (snd x) (snd y))$

**lemma** *aform-val-add-aform*:

**shows** *aform-val*  $e (add-aform X Y) = aform-val e X + aform-val e Y$   
 <proof>

**type-synonym** *aform-err* =  $real aform \times real$

**definition** *add-aform'*:: $nat \Rightarrow aform-err \Rightarrow aform-err \Rightarrow aform-err$

**where** *add-aform'*  $p x y =$   
 (let  
 $z0 = trunc-bound-eucl p (fst (fst x) + fst (fst y));$   
 $z = trunc-bound-pdevs p (add-pdevs (snd (fst x)) (snd (fst y)))$   
 in  $((fst z0, fst z), (sum-list' p [snd z0, snd z, abs (snd x), abs (snd y)]))$ )

**abbreviation** *degree-aform-err*:: $aform-err \Rightarrow nat$

**where** *degree-aform-err*  $X \equiv degree-aform (fst X)$

**lemma** *degree-aform-err-add-aform'*:

**assumes** *degree-aform-err*  $x \leq n$   
**assumes** *degree-aform-err*  $y \leq n$   
**shows** *degree-aform-err*  $(add-aform' p x y) \leq n$   
 <proof>

**definition** *aform-err e Xe* =  $\{aform-val e (fst Xe) - snd Xe .. aform-val e (fst Xe) + snd Xe::real\}$

**lemma** *aform-errI*:  $x \in \text{aform-err } e \ Xe$   
**if**  $\text{abs } (x - \text{aform-val } e \ (\text{fst } Xe)) \leq \text{snd } Xe$   
 $\langle \text{proof} \rangle$

**lemma** *add-aform'*:  
**assumes**  $e: e \in \text{UNIV} \rightarrow \{-1..1\}$   
**assumes**  $x: x \in \text{aform-err } e \ X$   
**assumes**  $y: y \in \text{aform-err } e \ Y$   
**shows**  $x + y \in \text{aform-err } e \ (\text{add-aform}' \ p \ X \ Y)$   
 $\langle \text{proof} \rangle$

### 5.2.2 Scaling

**definition** *aform-scaleR::real aform  $\Rightarrow$  'a::real-vector  $\Rightarrow$  'a aform*  
**where**  $\text{aform-scaleR } x \ y = (\text{fst } x \ *_R \ y, \ \text{pdevs-scaleR } (\text{snd } x) \ y)$

**lemma** *aform-val-scaleR-aform[simp]*:  
**shows**  $\text{aform-val } e \ (\text{aform-scaleR } X \ y) = \text{aform-val } e \ X \ *_R \ y$   
 $\langle \text{proof} \rangle$

### 5.2.3 Multiplication

**lemma** *aform-val-mult-exact*:  
 $\text{aform-val } e \ x \ * \ \text{aform-val } e \ y =$   
 $\text{fst } x \ * \ \text{fst } y +$   
 $\text{pdevs-val } e \ (\text{add-pdevs } (\text{scaleR-pdevs } (\text{fst } y) \ (\text{snd } x)) \ (\text{scaleR-pdevs } (\text{fst } x) \ (\text{snd } y))) +$   
 $(\sum i < d. e \ i \ *_R \ \text{pdevs-apply } (\text{snd } x) \ i) * (\sum i < d. e \ i \ *_R \ \text{pdevs-apply } (\text{snd } y) \ i)$   
**if**  $\text{degree } (\text{snd } x) \leq d \ \text{degree } (\text{snd } y) \leq d$   
 $\langle \text{proof} \rangle$

**lemma** *sum-times-bound*:— TODO: this gives better bounds for the remainder of multiplication

$(\sum i < d. e \ i \ * \ f \ i :: \text{real}) \ * \ (\sum i < d. e \ i \ * \ g \ i) =$   
 $(\sum i < d. (e \ i)^2 \ * \ (f \ i \ * \ g \ i)) +$   
 $(\sum (i, j) \mid i < j \wedge j < d. (e \ i \ * \ e \ j) \ * \ (f \ j \ * \ g \ i + f \ i \ * \ g \ j))$  **for**  $d :: \text{nat}$   
 $\langle \text{proof} \rangle$

**definition** *mult-aform::aform-err  $\Rightarrow$  aform-err  $\Rightarrow$  aform-err*  
**where**  $\text{mult-aform } x \ y = ((\text{fst } (\text{fst } x) \ * \ \text{fst } (\text{fst } y),$   
 $(\text{add-pdevs } (\text{scaleR-pdevs } (\text{fst } (\text{fst } y)) \ (\text{snd } (\text{fst } x))) \ (\text{scaleR-pdevs } (\text{fst } (\text{fst } x))$   
 $(\text{snd } (\text{fst } y))))),$   
 $(\text{tdev } (\text{snd } (\text{fst } x)) \ * \ \text{tdev } (\text{snd } (\text{fst } y)) +$   
 $\text{abs } (\text{snd } x) \ * \ (\text{abs } (\text{fst } (\text{fst } y)) + \text{Radius } (\text{fst } y)) +$   
 $\text{abs } (\text{snd } y) \ * \ (\text{abs } (\text{fst } (\text{fst } x)) + \text{Radius } (\text{fst } x)) + \text{abs } (\text{snd } x) \ * \ \text{abs } (\text{snd } y)$   
 $))$

**lemma** *mult-aformE*:  
**fixes**  $X \ Y :: \text{aform-err}$

**assumes**  $e: e \in UNIV \rightarrow \{-1..1\}$   
**assumes**  $x: x \in \text{aform-err } e \ X$   
**assumes**  $y: y \in \text{aform-err } e \ Y$   
**shows**  $x * y \in \text{aform-err } e \ (\text{mult-aform } X \ Y)$   
 <proof>

**definition**  $\text{mult-aform}'::\text{nat} \Rightarrow \text{aform-err} \Rightarrow \text{aform-err} \Rightarrow \text{aform-err}$

**where**  $\text{mult-aform}' \ p \ x \ y =$  (  
 let  
 $(fx, sx) = x;$   
 $(fy, sy) = y;$   
 $ex = \text{abs } sx;$   
 $ey = \text{abs } sy;$   
 $z0 = \text{trunc-bound-eucl } p \ (\text{fst } fx * \text{fst } fy);$   
 $u = \text{trunc-bound-pdevs } p \ (\text{scaleR-pdevs } (\text{fst } fy) \ (\text{snd } fx));$   
 $v = \text{trunc-bound-pdevs } p \ (\text{scaleR-pdevs } (\text{fst } fx) \ (\text{snd } fy));$   
 $w = \text{trunc-bound-pdevs } p \ (\text{add-pdevs } (\text{fst } u) \ (\text{fst } v));$   
 $tx = \text{tdev}' \ p \ (\text{snd } fx);$   
 $ty = \text{tdev}' \ p \ (\text{snd } fy);$   
 $l = \text{truncate-up } p \ (tx * ty);$   
 $ee = \text{truncate-up } p \ (ex * ey);$   
 $e1 = \text{truncate-up } p \ (ex * \text{truncate-up } p \ (\text{abs } (\text{fst } fy) + ty));$   
 $e2 = \text{truncate-up } p \ (ey * \text{truncate-up } p \ (\text{abs } (\text{fst } fx) + tx))$   
 in  
 $((\text{fst } z0, (\text{fst } w)), (\text{sum-list}' \ p \ [ee, e1, e2, l, \text{snd } z0, \text{snd } u, \text{snd } v, \text{snd } w]))$

**lemma**  $\text{aform-err}E:$

$\text{abs } (x - \text{aform-val } e \ (\text{fst } X)) \leq \text{snd } X$   
**if**  $x \in \text{aform-err } e \ X$   
 <proof>

**lemma**  $\text{mult-aform}'E:$

**fixes**  $X \ Y::\text{aform-err}$   
**assumes**  $e: e \in UNIV \rightarrow \{-1..1\}$   
**assumes**  $x: x \in \text{aform-err } e \ X$   
**assumes**  $y: y \in \text{aform-err } e \ Y$   
**shows**  $x * y \in \text{aform-err } e \ (\text{mult-aform}' \ p \ X \ Y)$   
 <proof>

**lemma**  $\text{degree-aform-mult-aform}':$

**assumes**  $\text{degree-aform-err } x \leq n$   
**assumes**  $\text{degree-aform-err } y \leq n$   
**shows**  $\text{degree-aform-err } (\text{mult-aform}' \ p \ x \ y) \leq n$   
 <proof>

**lemma**

**fixes**  $x \ a \ b::\text{real}$   
**assumes**  $a > 0$   
**assumes**  $x \in \{a ..b\}$

**assumes**  $- \text{inverse } (b * b) \leq \text{alpha}$   
**shows**  $\text{inverse-linear-lower: } \text{inverse } b + \text{alpha} * (x - b) \leq \text{inverse } x$  (**is** ?lower)  
**and**  $\text{inverse-linear-upper: } \text{inverse } x \leq \text{inverse } a + \text{alpha} * (x - a)$  (**is** ?upper)  
 <proof>

## 5.2.4 Inverse

**definition**  $\text{inverse-aform}'::\text{nat} \Rightarrow \text{real aform} \Rightarrow \text{real aform} \times \text{real}$  **where**

$\text{inverse-aform}' p X =$  (  
 let  $l = \text{Inf-aform}' p X$  in  
 let  $u = \text{Sup-aform}' p X$  in  
 let  $a = \min (\text{abs } l) (\text{abs } u)$  in  
 let  $b = \max (\text{abs } l) (\text{abs } u)$  in  
 let  $\text{sq} = \text{truncate-up } p (b * b)$  in  
 let  $\text{alpha} = - \text{real-divl } p 1 \text{ sq}$  in  
 let  $\text{dmax} = \text{truncate-up } p (\text{real-divr } p 1 a - \text{alpha} * a)$  in  
 let  $\text{dmin} = \text{truncate-down } p (\text{real-divl } p 1 b - \text{alpha} * b)$  in  
 let  $\text{zeta}' = \text{truncate-up } p ((\text{dmin} + \text{dmax}) / 2)$  in  
 let  $\text{zeta} = \text{if } l < 0 \text{ then } - \text{zeta}' \text{ else } \text{zeta}'$  in  
 let  $\text{delta} = \text{truncate-up } p (\text{zeta} - \text{dmin})$  in  
 let  $\text{res1} = \text{trunc-bound-eucl } p (\text{alpha} * \text{fst } X)$  in  
 let  $\text{res2} = \text{trunc-bound-eucl } p (\text{fst } \text{res1} + \text{zeta})$  in  
 let  $\text{zs} = \text{trunc-bound-pdevs } p (\text{scaleR-pdevs } \text{alpha} (\text{snd } X))$  in  
 $((\text{fst } \text{res2}, \text{fst } \text{zs}), (\text{sum-list}' p [\text{delta}, \text{snd } \text{res1}, \text{snd } \text{res2}, \text{snd } \text{zs}])))$

**lemma**  $\text{inverse-aform}'E:$

**fixes**  $X::\text{real aform}$

**assumes**  $e: e \in \text{UNIV} \rightarrow \{-1 .. 1\}$

**assumes**  $\text{Inf-pos: } \text{Inf-aform}' p X > 0$

**assumes**  $x = \text{aform-val } e X$

**shows**  $\text{inverse } x \in \text{aform-err } e (\text{inverse-aform}' p X)$

<proof>

**definition**  $\text{inverse-aform } p a =$

do {  
 let  $l = \text{Inf-aform}' p a;$   
 let  $u = \text{Sup-aform}' p a;$   
 if  $(l \leq 0 \wedge 0 \leq u)$  then  $\text{None}$   
 else if  $(l \leq 0)$  then  $(\text{Some } (\text{apfst } \text{uminus-aform } (\text{inverse-aform}' p (\text{uminus-aform } a))))$   
 else  $\text{Some } (\text{inverse-aform}' p a)$   
 }

**lemma**  $\text{eucl-truncate-up-eq-eucl-truncate-down:}$

$\text{eucl-truncate-up } p x = - (\text{eucl-truncate-down } p (- x))$

<proof>

**lemma**  $\text{inverse-aform}E:$

**fixes**  $X::\text{real aform}$

**assumes**  $e: e \in UNIV \rightarrow \{-1 .. 1\}$   
**and**  $disj: Inf\text{-}aform' p X > 0 \vee Sup\text{-}aform' p X < 0$   
**obtains**  $Y$  **where**  
 $inverse\text{-}aform p X = Some Y$   
 $inverse (aform\text{-}val e X) \in aform\text{-}err e Y$   
 $\langle proof \rangle$

**definition**  $aform\text{-}err\text{-}to\text{-}aform::aform\text{-}err \Rightarrow nat \Rightarrow real\ aform$   
**where**  $aform\text{-}err\text{-}to\text{-}aform X n = (fst (fst X), pdev\text{-}upd (snd (fst X)) n (snd X))$

**lemma**  $aform\text{-}err\text{-}to\text{-}aformE$ :  
**assumes**  $x \in aform\text{-}err e X$   
**assumes**  $deg: degree\text{-}aform\text{-}err X \leq n$   
**obtains**  $err$  **where**  $x = aform\text{-}val (e(n:=err)) (aform\text{-}err\text{-}to\text{-}aform X n)$   
 $-1 \leq err \leq 1$   
 $\langle proof \rangle$

**definition**  $aform\text{-}to\text{-}aform\text{-}err::real\ aform \Rightarrow nat \Rightarrow aform\text{-}err$   
**where**  $aform\text{-}to\text{-}aform\text{-}err X n = ((fst X, pdev\text{-}upd (snd X) n 0), abs (pdevs\text{-}apply (snd X) n))$

**lemma**  $aform\text{-}to\text{-}aform\text{-}err: aform\text{-}val e X \in aform\text{-}err e (aform\text{-}to\text{-}aform\text{-}err X n)$   
**if**  $e \in UNIV \rightarrow \{-1 .. 1\}$   
 $\langle proof \rangle$

**definition**  $acc\text{-}err p x e \equiv (fst x, truncate\text{-}up p (snd x + e))$

**definition**  $ivl\text{-}err :: real\ interval \Rightarrow (real \times real\ pdevs) \times real$   
**where**  $ivl\text{-}err ivl \equiv (((upper\ ivl + lower\ ivl)/2, zero\text{-}pdevs::real\ pdevs), (upper\ ivl - lower\ ivl) / 2)$

**lemma**  $inverse\text{-}aform$ :  
**fixes**  $X::real\ aform$   
**assumes**  $e: e \in UNIV \rightarrow \{-1 .. 1\}$   
**assumes**  $inverse\text{-}aform p X = Some Y$   
**shows**  $inverse (aform\text{-}val e X) \in aform\text{-}err e Y$   
 $\langle proof \rangle$

**lemma**  $aform\text{-}err\text{-}acc\text{-}err\text{-}leI$ :  
 $fx \in aform\text{-}err e (acc\text{-}err p X err)$   
**if**  $aform\text{-}val e (fst X) - (snd X + err) \leq fx \leq aform\text{-}val e (fst X) + (snd X + err)$   
 $\langle proof \rangle$

**lemma**  $aform\text{-}err\text{-}acc\text{-}errI$ :  
 $fx \in aform\text{-}err e (acc\text{-}err p X err)$   
**if**  $fx \in aform\text{-}err e (fst X, snd X + err)$

*<proof>*

**lemma** *minus-times-le-abs*:  $-(err * B) \leq |B|$  **if**  $-1 \leq err$   $err \leq 1$  **for**  $err::real$   
*<proof>*

**lemma** *times-le-abs*:  $err * B \leq |B|$  **if**  $-1 \leq err$   $err \leq 1$  **for**  $err::real$   
*<proof>*

**lemma** *aform-err-lemma1*:  $-1 \leq err \implies err \leq 1 \implies$   
 $X1 + (A - e d * B + err * B) - e1 \leq x \implies$   
 $X1 + (A - e d * B) - truncate-up p (|B| + e1) \leq x$   
*<proof>*

**lemma** *aform-err-lemma2*:  $-1 \leq err \implies err \leq 1 \implies$   
 $x \leq X1 + (A - e d * B + err * B) + e1 \implies$   
 $x \leq X1 + (A - e d * B) + truncate-up p (|B| + e1)$   
*<proof>*

**lemma** *aform-err-acc-err-aform-to-aform-errI*:  
 $x \in aform-err e (acc-err p (aform-to-aform-err X1 d) e1)$   
**if**  $-1 \leq err$   $err \leq 1$   $x \in aform-err (e(d := err)) (X1, e1)$   
*<proof>*

**definition** *map-aform-err*  $I p X =$   
(do {  
 let  $X0 = aform-err-to-aform X (degree-aform-err X)$ ;  
  $(X1, e1) \leftarrow I X0$ ;  
 Some (acc-err p (aform-to-aform-err X1 (degree-aform-err X)) e1)  
})

**lemma** *map-aform-err*:  
 $i x \in aform-err e Y$   
**if**  $I: \bigwedge e X Y. e \in UNIV \rightarrow \{-1 .. 1\} \implies I X = Some Y \implies i (aform-val e X) \in aform-err e Y$   
**and**  $e: e \in UNIV \rightarrow \{-1 .. 1\}$   
**and**  $Y: map-aform-err I p X = Some Y$   
**and**  $x: x \in aform-err e X$   
*<proof>*

**definition** *inverse-aform-err*  $p X = map-aform-err (inverse-aform p) p X$

**lemma** *inverse-aform-err*:  
 $inverse x \in aform-err e Y$   
**if**  $e: e \in UNIV \rightarrow \{-1 .. 1\}$   
**and**  $Y: inverse-aform-err p X = Some Y$   
**and**  $x: x \in aform-err e X$   
*<proof>*

### 5.3 Reduction (Summarization of Coefficients)

**definition** *pdevs-of-centered-ivl*  $r = (\text{inner-scaleR-pdevs } r \text{ One-pdevs})$

**lemma** *pdevs-of-centered-ivl-eq-pdevs-of-ivl[simp]*:  $\text{pdevs-of-centered-ivl } r = \text{pdevs-of-ivl } (-r) \text{ } r$   
 $\langle \text{proof} \rangle$

**lemma** *filter-pdevs-raw-nonzeros*:  $\{i. \text{filter-pdevs-raw } s \text{ } f \text{ } i \neq 0\} = \{i. f \text{ } i \neq 0\} \cap \{x. s \text{ } x (f \text{ } x)\}$   
 $\langle \text{proof} \rangle$

**definition** *summarize-pdevs*::

$\text{nat} \Rightarrow (\text{nat} \Rightarrow 'a \Rightarrow \text{bool}) \Rightarrow \text{nat} \Rightarrow 'a::\text{executable-euclidean-space } \text{pdevs} \Rightarrow 'a \text{ } \text{pdevs}$

**where** *summarize-pdevs*  $p \text{ } I \text{ } d \text{ } x =$   
 $(\text{let } t = \text{tdev}' p (\text{filter-pdevs } (-I) \text{ } x)$   
 $\text{in } \text{msum-pdevs } d (\text{filter-pdevs } I \text{ } x) (\text{pdevs-of-centered-ivl } t))$

**definition** *summarize-threshold*

**where** *summarize-threshold*  $p \text{ } t \text{ } x \text{ } y \longleftrightarrow \text{infnorm } y \geq t * \text{infnorm } (\text{eucl-truncate-up } p (\text{tdev}' p \text{ } x))$

**lemma** *error-abs-euclE*:

**fixes** *err*:: $'a::\text{ordered-euclidean-space}$   
**assumes**  $\text{abs } \text{err} \leq k$   
**obtains**  $e::'a \Rightarrow \text{real}$  **where**  $\text{err} = (\sum i \in \text{Basis}. (e \text{ } i * (k * i)) *_R i) \text{ } e \in \text{UNIV} \rightarrow \{-1 .. 1\}$   
 $\langle \text{proof} \rangle$

**lemma** *summarize-pdevsE*:

**fixes**  $x::'a::\text{executable-euclidean-space } \text{pdevs}$   
**assumes**  $e: e \in \text{UNIV} \rightarrow \{-1 .. 1\}$   
**assumes**  $d: \text{degree } x \leq d$   
**obtains**  $e'$  **where**  $\text{pdevs-val } e \text{ } x = \text{pdevs-val } e' (\text{summarize-pdevs } p \text{ } I \text{ } d \text{ } x)$   
 $\bigwedge i. i < d \implies e \text{ } i = e' \text{ } i$   
 $e' \in \text{UNIV} \rightarrow \{-1 .. 1\}$   
 $\langle \text{proof} \rangle$

**definition** *summarize-pdevs-list*  $p \text{ } I \text{ } d \text{ } xs =$

$\text{map } (\lambda(d, x). \text{summarize-pdevs } p (\lambda i -. I \text{ } i (\text{pdevs-applys } xs \text{ } i)) \text{ } d \text{ } x) (\text{zip } [d..<d + \text{length } xs] \text{ } xs)$

**lemma** *filter-pdevs-cong[cong]*:

**assumes**  $x = y$   
**assumes**  $\bigwedge i. i \in \text{pdevs-domain } y \implies P \text{ } i (\text{pdevs-apply } x \text{ } i) = Q \text{ } i (\text{pdevs-apply } y \text{ } i)$   
**shows**  $\text{filter-pdevs } P \text{ } x = \text{filter-pdevs } Q \text{ } y$   
 $\langle \text{proof} \rangle$

**lemma** *summarize-pdevs-cong*[*cong*]:  
**assumes**  $p = q \ a = c \ b = d$   
**assumes**  $PQ: \bigwedge i. i \in \text{pdevs-domain } d \implies P \ i \ (\text{pdevs-apply } b \ i) = Q \ i \ (\text{pdevs-apply } d \ i)$   
**shows**  $\text{summarize-pdevs } p \ P \ a \ b = \text{summarize-pdevs } q \ Q \ c \ d$   
 $\langle \text{proof} \rangle$

**lemma** *lookup-eq-None-iff*:  $(\text{Mapping.lookup } M \ x = \text{None}) = (x \notin \text{Mapping.keys } M)$   
 $\langle \text{proof} \rangle$

**lemma** *lookup-eq-SomeD*:  
 $(\text{Mapping.lookup } M \ x = \text{Some } y) \implies (x \in \text{Mapping.keys } M)$   
 $\langle \text{proof} \rangle$

**definition** *domain-pdevs*  $xs = (\bigcup (\text{pdevs-domain } `(\text{set } xs)))$

**definition** *pdevs-mapping*  $xs =$   
 $(\text{let}$   
 $\ D = \text{sorted-list-of-set } (\text{domain-pdevs } xs);$   
 $\ M = \text{Mapping.tabulate } D \ (\text{pdevs-applys } xs);$   
 $\ \text{zeroes} = \text{replicate } (\text{length } xs) \ 0$   
 $\ \text{in } \text{Mapping.lookup-default } \text{zeroes } M)$

**lemma** *pdevs-mapping-eq*[*simp*]:  $\text{pdevs-mapping } xs = \text{pdevs-applys } xs$   
 $\langle \text{proof} \rangle$

**lemma** *compute-summarize-pdevs-list*[*code*]:  
 $\text{summarize-pdevs-list } p \ I \ d \ xs =$   
 $(\text{let } M = \text{pdevs-mapping } xs$   
 $\ \text{in } \text{map } (\lambda(x, y). \text{summarize-pdevs } p \ (\lambda i -. I \ i \ (M \ i)) \ x \ y) \ (\text{zip } [d..<d + \text{length } xs] \ xs))$   
 $\langle \text{proof} \rangle$

**lemma**  
*in-centered-ivlE*:  
**fixes**  $r \ t::\text{real}$   
**assumes**  $r \in \{-t .. t\}$   
**obtains**  $e$  **where**  $e \in \{-1 .. 1\} \ r = e * t$   
 $\langle \text{proof} \rangle$

**lift-definition** *singleton-pdevs*:: $'a \Rightarrow 'a::\text{real-normed-vector pdevs}$  **is**  $\lambda x \ i. \text{if } i = 0 \text{ then } x \text{ else } 0$   
 $\langle \text{proof} \rangle$

**lemmas** [*simp*] = *singleton-pdevs.rep-eq*

**lemma** *singleton-0*[*simp*]:  $\text{singleton-pdevs } 0 = \text{zero-pdevs}$   
 $\langle \text{proof} \rangle$

**lemma** *degree-singleton-pdevs[simp]*:  $\text{degree} (\text{singleton-pdevs } x) = (\text{if } x = 0 \text{ then } 0 \text{ else } \text{Suc } 0)$   
 ⟨proof⟩

**lemma** *pdevs-val-singleton-pdevs[simp]*:  $\text{pdevs-val } e (\text{singleton-pdevs } x) = e \ 0 \ *_{\mathbb{R}} \ x$   
 ⟨proof⟩

**lemma** *pdevs-of-ivl-real*:  
**fixes**  $a \ b :: \text{real}$   
**shows**  $\text{pdevs-of-ivl } a \ b = \text{singleton-pdevs } ((b - a) / 2)$   
 ⟨proof⟩

**lemma** *summarize-pdevs-listE*:  
**fixes**  $X :: \text{real pdevs list}$   
**assumes**  $e: e \in \text{UNIV} \rightarrow \{-1 .. 1\}$   
**assumes**  $d: \text{degrees } X \leq d$   
**obtains**  $e'$  **where**  $\text{pdevs-vals } e \ X = \text{pdevs-vals } e' (\text{summarize-pdevs-list } p \ I \ d \ X)$   
 $\bigwedge i. i < d \implies e \ i = e' \ i$   
 $e' \in \text{UNIV} \rightarrow \{-1 .. 1\}$   
 ⟨proof⟩

**fun** *list-ex2* **where**  
 $\text{list-ex2 } P \ [] \ xs = \text{False}$   
 $|\ \text{list-ex2 } P \ xs \ [] = \text{False}$   
 $|\ \text{list-ex2 } P \ (x\#\xs) \ (y\#ys) = (P \ x \ y \ \vee \ \text{list-ex2 } P \ xs \ ys)$

**lemma** *list-ex2-iff*:  
 $\text{list-ex2 } P \ xs \ ys \longleftrightarrow (\neg \text{list-all2 } (\neg P) (\text{take } (\text{length } ys) \ xs) (\text{take } (\text{length } xs) \ ys))$   
 ⟨proof⟩

**definition** *summarize-aforms*  $p \ C \ d \ (X :: \text{real aform list}) =$   
 $(\text{zip } (\text{map } \text{fst } X) (\text{summarize-pdevs-list } p \ (C \ X) \ d \ (\text{map } \text{snd } X)))$

**lemma** *aform-vals-pdevs-vals*:  
 $\text{aform-vals } e \ X = \text{map } (\lambda(x, y). x + y) (\text{zip } (\text{map } \text{fst } X) (\text{pdevs-vals } e \ (\text{map } \text{snd } X)))$   
 ⟨proof⟩

**lemma** *summarize-aformsE*:  
**fixes**  $X :: \text{real aform list}$   
**assumes**  $e: e \in \text{UNIV} \rightarrow \{-1 .. 1\}$   
**assumes**  $d: \text{degree-aforms } X \leq d$   
**obtains**  $e'$  **where**  $\text{aform-vals } e \ X = \text{aform-vals } e' (\text{summarize-aforms } p \ C \ d \ X)$   
 $\bigwedge i. i < d \implies e \ i = e' \ i$   
 $e' \in \text{UNIV} \rightarrow \{-1 .. 1\}$   
 ⟨proof⟩

Different reduction strategies:

**definition** *collect-threshold*  $p$   $ta$   $t$  ( $X::real$  *aform list*) =  
 (let  
    $Xs = map\ snd\ X$ ;  
    $as = map\ (\lambda X. max\ ta\ (t * tdev'\ p\ X))\ Xs$   
 in ( $\lambda(i::nat)\ xs. list-ex2\ (\leq)\ as\ (map\ abs\ xs)$ ))

**definition** *collect-girard*  $p$   $m$  ( $X::real$  *aform list*) =  
 (let  
    $Xs = map\ snd\ X$ ;  
    $M = pdevs-mapping\ Xs$ ;  
    $D = domain-pdevs\ Xs$ ;  
    $N = length\ X$   
 in if  $card\ D \leq m$  then ( $\lambda-.\ True$ ) else  
 let  
    $Ds = sorted-list-of-set\ D$ ;  
    $ortho-indices = map\ fst\ (take\ (2 * N)\ (sort-key\ (\lambda(i, r). r)\ (map\ (\lambda i. let\ xs$   
 =  $M\ i\ in\ (i, sum-list'\ p\ xs - fold\ max\ xs\ 0))\ Ds)))$ ;  
    $- = ()$   
 in ( $\lambda i\ (xs::real\ list). i \in set\ ortho-indices$ ))

## 5.4 Splitting with heuristics

**definition** *abs-pdevs* = *unop-pdevs abs*

**definition** *abssum-of-pdevs-list*  $X = fold\ (\lambda a\ b. (add-pdevs\ (abs-pdevs\ a)\ b))\ X\ zero-pdevs$

**definition** *split-aforms*  $xs\ i = (let\ splits = map\ (\lambda x. split-aform\ x\ i)\ xs\ in\ (map\ fst\ splits, map\ snd\ splits))$

**definition** *split-aforms-largest-uncond*  $X =$   
 (let ( $i, x$ ) = *max-pdev* (*abssum-of-pdevs-list* (*map snd X*)) in *split-aforms X i*)

**definition** *Inf-aform-err*  $p\ Rd = (float-of\ (truncate-down\ p\ (Inf-aform'\ p\ (fst\ Rd) - abs(snd\ Rd))))$

**definition** *Sup-aform-err*  $p\ Rd = (float-of\ (truncate-up\ p\ (Sup-aform'\ p\ (fst\ Rd) + abs(snd\ Rd))))$

**context includes** *interval.lifting begin*

**lift-definition** *ivl-of-aform-err::nat*  $\Rightarrow$  *aform-err*  $\Rightarrow$  *float interval*

is  $\lambda p\ Rd. (Inf-aform-err\ p\ Rd, Sup-aform-err\ p\ Rd)$

*<proof>*

**lemma** *lower-ivl-of-aform-err*: *lower* (*ivl-of-aform-err p Rd*) = *Inf-aform-err p Rd*

**and** *upper-ivl-of-aform-err*: *upper* (*ivl-of-aform-err p Rd*) = *Sup-aform-err p Rd*

*<proof>*

**end**

**definition** *approx-un::nat*

$\Rightarrow$  (*float interval*  $\Rightarrow$  *float interval option*)

$\Rightarrow ((\text{real} \times \text{real pdevs}) \times \text{real}) \text{ option}$   
 $\Rightarrow ((\text{real} \times \text{real pdevs}) \times \text{real}) \text{ option}$   
**where** *approx-un p f a* = do {  
*rd*  $\leftarrow$  *a*;  
*ivl*  $\leftarrow$  *f (ivl-of-aform-err p rd)*;  
*Some (ivl-err (real-interval ivl))*  
}

**definition** *interval-extension1*::(*float interval*  $\Rightarrow$  (*float interval*) *option*)  $\Rightarrow$  (*real*  
 $\Rightarrow$  *real*)  $\Rightarrow$  *bool*  
**where** *interval-extension1 F f*  $\longleftrightarrow$  ( $\forall$  *ivl ivl'*. *F ivl* = *Some ivl'*  $\longrightarrow$  ( $\forall$  *x*. *x*  $\in_r$   
*ivl*  $\longrightarrow$  *f x*  $\in_r$  *ivl'*))

**lemma** *interval-extension1D*:  
**assumes** *interval-extension1 F f*  
**assumes** *F ivl* = *Some ivl'*  
**assumes** *x*  $\in_r$  *ivl*  
**shows** *f x*  $\in_r$  *ivl'*  
*<proof>*

**lemma** *approx-un-argE*:  
**assumes** *au: approx-un p F X* = *Some Y*  
**obtains** *X'* **where** *X* = *Some X'*  
*<proof>*

**lemma** *degree-aform-independent-from*:  
*degree-aform (independent-from d1 X)*  $\leq$  *d1* + *degree-aform X*  
*<proof>*

**lemma** *degree-aform-of-ivl*:  
**fixes** *a b::'a::executable-euclidean-space*  
**shows** *degree-aform (aform-of-ivl a b)*  $\leq$  *length (Basis-list::'a list)*  
*<proof>*

**lemma** *aform-err-ivl-err[simp]*: *aform-err e (ivl-err ivl')* = *set-of ivl'*  
*<proof>*

**lemma** *Inf-Sup-aform-err*:  
**fixes** *X*  
**assumes** *e: e*  $\in$  *UNIV*  $\rightarrow$   $\{-1 .. 1\}$   
**defines** *X'*  $\equiv$  *fst X*  
**shows** *aform-err e X*  $\subseteq$   $\{\text{Inf-aform-err } p \text{ } X .. \text{Sup-aform-err } p \text{ } X\}$   
*<proof>*

**lemma** *ivl-of-aform-err*:  
**fixes** *X*  
**assumes** *e: e*  $\in$  *UNIV*  $\rightarrow$   $\{-1 .. 1\}$   
**shows** *x*  $\in$  *aform-err e X*  $\implies$  *x*  $\in_r$  *ivl-of-aform-err p X*  
*<proof>*

**lemma** *approx-unE*:

**assumes** *ie*: *interval-extension1* *F f*

**assumes** *e*:  $e \in UNIV \rightarrow \{-1 .. 1\}$

**assumes** *au*: *approx-un* *p F X'err = Some Ye*

**assumes** *x*: *case X'err of None  $\Rightarrow$  True | Some X'err  $\Rightarrow x \in aform-err e X'err$*

**shows**  $f x \in aform-err e Ye$

*<proof>*

**definition** *approx-bin p f rd sd = do* {

*ivl*  $\leftarrow f$  (*ivl-of-aform-err p rd*)

(*ivl-of-aform-err p sd*);

*Some* (*ivl-err* (*real-interval ivl*))

}

**definition** *interval-extension2::(float interval  $\Rightarrow$  float interval  $\Rightarrow$  float interval option)  $\Rightarrow$  (real  $\Rightarrow$  real  $\Rightarrow$  real)  $\Rightarrow$  bool*

**where** *interval-extension2 F f*  $\longleftrightarrow (\forall ivl1 ivl2 ivl. F ivl1 ivl2 = Some ivl \longrightarrow (\forall x y. x \in_r ivl1 \longrightarrow y \in_r ivl2 \longrightarrow f x y \in_r ivl))$

**lemma** *interval-extension2D*:

**assumes** *interval-extension2 F f*

**assumes**  $F ivl1 ivl2 = Some ivl$

**shows**  $x \in_r ivl1 \Longrightarrow y \in_r ivl2 \Longrightarrow f x y \in_r ivl$

*<proof>*

**lemma** *approx-binE*:

**assumes** *ie*: *interval-extension2 F f*

**assumes** *w*:  $w \in aform-err e (W', errw)$

**assumes** *x*:  $x \in aform-err e (X', errx)$

**assumes** *ab*: *approx-bin p F ((W', errw)) ((X', errx)) = Some Ye*

**assumes** *e*:  $e \in UNIV \rightarrow \{-1 .. 1\}$

**shows**  $f w x \in aform-err e Ye$

*<proof>*

**definition** *min-aform-err p a1 (a2::aform-err) =*

(*let*

*ivl1* = *ivl-of-aform-err p a1*;

*ivl2* = *ivl-of-aform-err p a2*

*in* if upper *ivl1* < lower *ivl2* then *a1*

else if upper *ivl2* < lower *ivl1* then *a2*

else *ivl-err* (*real-interval* (*min-interval ivl1 ivl2*)))

**definition** *max-aform-err p a1 (a2::aform-err) =*

(*let*

*ivl1* = *ivl-of-aform-err p a1*;

*ivl2* = *ivl-of-aform-err p a2*

*in* if upper *ivl1* < lower *ivl2* then *a2*

else if upper *ivl2* < lower *ivl1* then *a1*

else ivl-err (real-interval (max-interval ivl1 ivl2)))

## 5.5 Approximate Min Range - Kind Of Trigonometric Functions

**definition** *affine-unop* :: nat  $\Rightarrow$  real  $\Rightarrow$  real  $\Rightarrow$  real  $\Rightarrow$  aform-err  $\Rightarrow$  aform-err  
**where**

*affine-unop* p a b d X = (let  
 ((x, xs), xe) = X;  
 (ax, axe) = trunc-bound-eucl p (a \* x);  
 (y, ye) = trunc-bound-eucl p (ax + b);  
 (ys, yse) = trunc-bound-pdevs p (scaleR-pdevs a xs)  
 in ((y, ys), sum-list' p [truncate-up p (|a| \* xe), axe, ye, yse, d]))  
 — TODO: also do binop

**lemma** *aform-err-leI*:

y  $\in$  aform-err e (c, d)  
**if** y  $\in$  aform-err e (c, d') d'  $\leq$  d  
 <proof>

**lemma** *aform-err-eqI*:

y  $\in$  aform-err e (c, d)  
**if** y  $\in$  aform-err e (c, d') d' = d  
 <proof>

**lemma** *sum-list'-append[simp]*: sum-list' p (ds@[d]) = truncate-up p (d + sum-list' p ds)

<proof>

**lemma** *aform-err-sum-list'*:

y  $\in$  aform-err e (c, sum-list' p ds)  
**if** y  $\in$  aform-err e (c, sum-list ds)  
 <proof>

**lemma** *aform-err-trunc-bound-eucl*:

y  $\in$  aform-err e ((fst (trunc-bound-eucl p X), xs), snd (trunc-bound-eucl p X) + d)

**if** y: y  $\in$  aform-err e ((X, xs), d)  
 <proof>

**lemma** *trunc-err-pdevsE*:

**assumes** e  $\in$  UNIV  $\rightarrow$   $\{-1 .. 1\}$

**obtains** err **where**

|err|  $\leq$  tdev' p (trunc-err-pdevs p xs)  
 pdevs-val e (trunc-pdevs p xs) = pdevs-val e xs + err  
 <proof>

**lemma** *aform-err-trunc-bound-pdevsI*:

y  $\in$  aform-err e ((c, fst (trunc-bound-pdevs p xs)), snd (trunc-bound-pdevs p xs))

+ d)  
**if**  $y: y \in \text{aform-err } e ((c, xs), d)$   
**and**  $e: e \in \text{UNIV} \rightarrow \{-1 .. 1\}$   
 ⟨proof⟩

**lemma** *aform-err-addI*:  
 $y \in \text{aform-err } e ((a + b, xs), d)$   
**if**  $y - b \in \text{aform-err } e ((a, xs), d)$   
 ⟨proof⟩

**theorem** *affine-unop*:  
**assumes**  $x: x \in \text{aform-err } e X$   
**assumes**  $f: |f x - (a * x + b)| \leq d$   
**and**  $e: e \in \text{UNIV} \rightarrow \{-1 .. 1\}$   
**shows**  $f x \in \text{aform-err } e (\text{affine-unop } p a b d X)$   
 ⟨proof⟩

**lemma** *min-range-coeffs-ge*:  
 $|f x - (a * x + b)| \leq d$   
**if**  $l: l \leq x$  **and**  $u: x \leq u$   
**and**  $f': \bigwedge y. y \in \{l .. u\} \implies (f \text{ has-real-derivative } f' y) (\text{at } y)$   
**and**  $a: \bigwedge y. y \in \{l..u\} \implies a \leq f' y$   
**and**  $d: d \geq (f u - f l - a * (u - l)) / 2 + |(f l + f u - a * (l + u)) / 2 - b|$   
**for**  $a b d::\text{real}$   
 ⟨proof⟩

**lemma** *min-range-coeffs-le*:  
 $|f x - (a * x + b)| \leq d$   
**if**  $l: l \leq x$  **and**  $u: x \leq u$   
**and**  $f': \bigwedge y. y \in \{l .. u\} \implies (f \text{ has-real-derivative } f' y) (\text{at } y)$   
**and**  $a: \bigwedge y. y \in \{l .. u\} \implies f' y \leq a$   
**and**  $d: d \geq (f l - f u + a * (u - l)) / 2 + |(f l + f u - a * (l + u)) / 2 - b|$   
**for**  $a b d::\text{real}$   
 ⟨proof⟩

**context includes** *floatarith-syntax begin*

**definition** *range-reducer*  $p l =$   
 (if  $l < 0 \vee l > 2 * \text{lb-pi } p$   
 then  $\text{approx } p (Pi * (\text{Num } (-2)) * (\text{Floor } (\text{Num } (l * \text{Float } 1 (-1)) / Pi))) \square$   
 else  $\text{Some } 0$ )

**lemmas** *approx-emptyD = approx[OF bounded-by-None[of Nil], simplified]*

**lemma** *range-reducerE*:  
**assumes** *range-reducer*  $p l = \text{Some } ivl$   
**obtains**  $n::\text{int}$  **where**  $n * (2 * pi) \in_r ivl$   
 ⟨proof⟩

**definition** *range-reduce-aform-err*  $p X = do \{$   
 $r \leftarrow range-reducer\ p\ (lower\ (ivl-of-aform-err\ p\ X));$   
 $Some\ (add-aform'\ p\ X\ (ivl-err\ (real-interval\ r)))$   
 $\}$

**lemma** *range-reduce-aform-errE*:  
**assumes**  $e: e \in UNIV \rightarrow \{-1 .. 1\}$   
**assumes**  $x: x \in aform-err\ e\ X$   
**assumes** *range-reduce-aform-err*  $p\ X = Some\ Y$   
**obtains**  $n::int$  **where**  $x + n * (2 * pi) \in aform-err\ e\ Y$   
 $\langle proof \rangle$

**definition** *min-range-mono*  $p\ F\ DF\ l\ u\ X = do \{$   
 $let\ L = Num\ l;$   
 $let\ U = Num\ u;$   
 $aivl \leftarrow approx\ p\ (Min\ (DF\ L)\ (DF\ U))\ [];$   
 $let\ a = lower\ aivl;$   
 $let\ A = Num\ a;$   
 $bivl \leftarrow approx\ p\ (Half\ (F\ L + F\ U - A * (L + U)))\ [];$   
 $let\ (b, be) = mid-err\ bivl;$   
 $let\ (B, Be) = (Num\ (float-of\ b), Num\ (float-of\ be));$   
 $divl \leftarrow approx\ p\ ((Half\ (F\ U - F\ L - A * (U - L))) + Be)\ [];$   
 $Some\ (affine-unop\ p\ a\ b\ (real-of-float\ (upper\ divl))\ X)$   
 $\}$

**lemma** *min-range-mono*:  
**assumes**  $x: x \in aform-err\ e\ X$   
**assumes**  $l \leq x\ x \leq u$   
**assumes** *min-range-mono*  $p\ F\ DF\ l\ u\ X = Some\ Y$   
**assumes**  $e: e \in UNIV \rightarrow \{-1 .. 1\}$   
**assumes**  $F: \bigwedge x. x \in \{real-of-float\ l .. u\} \implies interpret-floatarith\ (F\ (Num\ x))\ []$   
 $= f\ x$   
**assumes**  $DF: \bigwedge x. x \in \{real-of-float\ l .. u\} \implies interpret-floatarith\ (DF\ (Num\ x))\ [] = f'\ x$   
**assumes**  $f': \bigwedge x. x \in \{real-of-float\ l .. u\} \implies (f\ has-real-derivative\ f'\ x)\ (at\ x)$   
**assumes**  $f'-le: \bigwedge x. x \in \{real-of-float\ l .. u\} \implies min\ (f'\ l)\ (f'\ u) \leq f'\ x$   
**shows**  $f\ x \in aform-err\ e\ Y$   
 $\langle proof \rangle$

**definition** *min-range-antimono*  $p\ F\ DF\ l\ u\ X = do \{$   
 $let\ L = Num\ l;$   
 $let\ U = Num\ u;$   
 $aivl \leftarrow approx\ p\ (Max\ (DF\ L)\ (DF\ U))\ [];$   
 $let\ a = upper\ aivl;$   
 $let\ A = Num\ a;$   
 $bivl \leftarrow approx\ p\ (Half\ (F\ L + F\ U - A * (L + U)))\ [];$   
 $let\ (b, be) = mid-err\ bivl;$   
 $let\ (B, Be) = (Num\ (float-of\ b), Num\ (float-of\ be));$   
 $divl \leftarrow approx\ p\ (Add\ (Half\ (F\ L - F\ U + A * (U - L)))\ Be)\ [];$   
 $\}$

Some (affine-unop p a b (real-of-float (upper divl)) X)  
}

**lemma** *min-range-antimono*:

**assumes**  $x: x \in \text{aform-err } e \ X$   
**assumes**  $l \leq x \ x \leq u$   
**assumes** *min-range-antimono* p F DF l u X = Some Y  
**assumes**  $e: e \in \text{UNIV} \rightarrow \{-1 .. 1\}$   
**assumes**  $F: \bigwedge x. x \in \{\text{real-of-float } l .. u\} \implies \text{interpret-floatarith } (F \ (\text{Num } x)) \ \square$   
 $= f \ x$   
**assumes**  $DF: \bigwedge x. x \in \{\text{real-of-float } l .. u\} \implies \text{interpret-floatarith } (DF \ (\text{Num } x)) \ \square = f' \ x$   
**assumes**  $f': \bigwedge x. x \in \{\text{real-of-float } l .. u\} \implies (f \ \text{has-real-derivative } f' \ x) \ (\text{at } x)$   
**assumes**  $f'\text{-le}: \bigwedge x. x \in \{\text{real-of-float } l .. u\} \implies f' \ x \leq \max (f' \ l) (f' \ u)$   
**shows**  $f \ x \in \text{aform-err } e \ Y$   
<proof>

**definition** *cos-aform-err* p X = do {

$X \leftarrow \text{range-reduce-aform-err } p \ X;$   
let  $ivl = \text{ivl-of-aform-err } p \ X;$   
let  $l = \text{lower } ivl;$   
let  $u = \text{upper } ivl;$   
let  $L = \text{Num } l;$   
let  $U = \text{Num } u;$   
if  $l \geq 0 \wedge u \leq \text{lb-pi } p$  then  
  *min-range-antimono* p Cos ( $\lambda x. (\text{Minus } (\text{Sin } x))) \ l \ u \ X$   
else if  $l \geq \text{ub-pi } p \wedge u \leq 2 * \text{lb-pi } p$  then  
  *min-range-mono* p Cos ( $\lambda x. (\text{Minus } (\text{Sin } x))) \ l \ u \ X$   
else do {  
  Some (*ivl-err* (real-interval (cos-float-interval p ivl)))  
}  
}

**lemma** *abs-half-enclosure*:

**fixes**  $r::\text{real}$   
**assumes**  $bl \leq r \ r \leq bu$   
**shows**  $|r - (bl + bu) / 2| \leq (bu - bl) / 2$   
<proof>

**lemma** *cos-aform-err*:

**assumes**  $x: x \in \text{aform-err } e \ X0$   
**assumes** *cos-aform-err* p X0 = Some Y  
**assumes**  $e: e \in \text{UNIV} \rightarrow \{-1 .. 1\}$   
**shows**  $\text{cos } x \in \text{aform-err } e \ Y$   
<proof>

**definition** *sqrt-aform-err* p X = do {

let  $ivl = \text{ivl-of-aform-err } p \ X;$   
let  $l = \text{lower } ivl;$

```

let u = upper ivl;
if 0 < l then min-range-mono p Sqrt ( $\lambda x. \text{Half } (\text{Inverse } (\text{Sqrt } x))$ ) l u X
else Some (ivl-err (real-interval (sqrt-float-interval p ivl)))
}

```

**lemma** *sqrt-aform-err*:

```

assumes x: x  $\in$  aform-err e X
assumes sqrt-aform-err p X = Some Y
assumes e: e  $\in$  UNIV  $\rightarrow$   $\{-1 .. 1\}$ 
shows sqrt x  $\in$  aform-err e Y
<proof>

```

**definition** *ln-aform-err* p X = do {

```

let ivl = ivl-of-aform-err p X;
let l = lower ivl;
if 0 < l then min-range-mono p Ln inverse l (upper ivl) X
else None
}

```

**lemma** *ln-aform-err*:

```

assumes x: x  $\in$  aform-err e X
assumes ln-aform-err p X = Some Y
assumes e: e  $\in$  UNIV  $\rightarrow$   $\{-1 .. 1\}$ 
shows ln x  $\in$  aform-err e Y
<proof>

```

**definition** *exp-aform-err* p X = do {

```

let ivl = ivl-of-aform-err p X;
min-range-mono p Exp Exp (lower ivl) (upper ivl) X
}

```

**lemma** *exp-aform-err*:

```

assumes x: x  $\in$  aform-err e X
assumes exp-aform-err p X = Some Y
assumes e: e  $\in$  UNIV  $\rightarrow$   $\{-1 .. 1\}$ 
shows exp x  $\in$  aform-err e Y
<proof>

```

**definition** *arctan-aform-err* p X = do {

```

let l = Inf-aform-err p X;
let u = Sup-aform-err p X;
min-range-mono p Arctan ( $\lambda x. 1 / (\text{Num } 1 + x * x)$ ) l u X
}

```

**lemma** *pos-add-nonneg-ne-zero*:  $a > 0 \implies b \geq 0 \implies a + b \neq 0$

```

for a b::real
<proof>

```

**lemma** *arctan-aform-err*:

**assumes**  $x: x \in \text{aform-err } e \ X$   
**assumes**  $\text{arctan-aform-err } p \ X = \text{Some } Y$   
**assumes**  $e: e \in \text{UNIV} \rightarrow \{-1 .. 1\}$   
**shows**  $\text{arctan } x \in \text{aform-err } e \ Y$   
 <proof>

## 5.6 Power, TODO: compare with Min-range approximation?!

**definition**  $\text{power-aform-err } p \ (X::\text{aform-err}) \ n =$   
 (if  $n = 0$  then  $((1, \text{zero-pdevs}), 0)$   
 else if  $n = 1$  then  $X$   
 else  
 let  $x0 = \text{float-of } (\text{fst } (\text{fst } X));$   
 $xs = \text{snd } (\text{fst } X);$   
 $xe = \text{float-of } (\text{snd } X);$   
 $C = \text{the } (\text{approx } p \ (\text{Num } x0 \ \widehat{e} \ n) \ []);$   
 $(c, ce) = \text{mid-err } C;$   
 $NX = \text{the } (\text{approx } p \ (\text{Num } (\text{of-nat } n) * (\text{Num } x0 \ \widehat{e} \ (n - 1))) \ []);$   
 $(nx, nxe) = \text{mid-err } NX;$   
 $Y = \text{scaleR-pdevs } nx \ xs;$   
 $(Y', Y\text{-err}) = \text{trunc-bound-pdevs } p \ Y;$   
 $t = \text{tdev}' \ p \ xs;$   
 $Ye = \text{truncate-up } p \ (nxe * t);$   
 $\text{err} = \text{the } (\text{approx } p$   
 $(\text{Num } (\text{of-nat } n) * \text{Num } xe * \text{Abs } (\text{Num } x0) \ \widehat{e} \ (n - 1) +$   
 $(\text{Sum}_e (\lambda k. \text{Num } (\text{of-nat } (n \ \text{choose } k)) * \text{Abs } (\text{Num } x0) \ \widehat{e} \ (n - k)) * (\text{Num}$   
 $xe + \text{Num } (\text{float-of } t)) \ \widehat{e} \ k$   
 $[2..<\text{Suc } n])) \ []);$   
 $\text{ERR} = \text{upper err}$   
 in  $((c, Y'), \text{sum-list}' \ p \ [ce, Y\text{-err}, Ye, \text{real-of-float } \text{ERR}]))$

**lemma**  $\text{bounded-by-Nil: bounded-by } [] \ []$   
 <proof>

**lemma**  $\text{plain-floatarith-approx:}$   
**assumes**  $\text{plain-floatarith } 0 \ f$   
**shows**  $\text{interpret-floatarith } f \ [] \ \in_r \ (\text{the } (\text{approx } p \ f \ []))$   
 <proof>

**lemma**  $\text{plain-floatarith-Sum}_e:$   
 $\text{plain-floatarith } n \ (\text{Sum}_e \ f \ xs) \longleftrightarrow \text{list-all } (\lambda i. \text{plain-floatarith } n \ (f \ i)) \ xs$   
 <proof>

**lemma**  $\text{sum-list}'\text{-float[simp]: } \text{sum-list}' \ p \ xs \in \text{float}$   
 <proof>

**lemma**  $\text{tdev}'\text{-float[simp]: } \text{tdev}' \ p \ xs \in \text{float}$   
 <proof>

**lemma**

**fixes**  $x\ y::real$   
**assumes**  $abs\ (x - y) \leq e$   
**obtains**  $err$  **where**  $x = y + err$   $abs\ err \leq e$   
 $\langle proof \rangle$

**theorem** *power-aform-err*:

**assumes**  $x \in aform\text{-}err\ e\ X$   
**assumes**  $floats[simp]: fst\ (fst\ X) \in float\ snd\ X \in float$   
**assumes**  $e: e \in UNIV \rightarrow \{-1 .. 1\}$   
**shows**  $x \wedge n \in aform\text{-}err\ e\ (power\text{-}aform\text{-}err\ p\ X\ n)$   
 $\langle proof \rangle$

**definition** [*code-abbrev*]:  $is\text{-}float\ r \longleftrightarrow r \in float$

**lemma** [*code*]:  $is\text{-}float\ (real\text{-}of\text{-}float\ f) = True$   
 $\langle proof \rangle$

**definition** *powr-aform-err*  $p\ X\ A =$  (  
  **if**  $Inf\text{-}aform\text{-}err\ p\ X > 0$  **then do** {  
     $L \leftarrow ln\text{-}aform\text{-}err\ p\ X;$   
     $exp\text{-}aform\text{-}err\ p\ (mult\text{-}aform'\ p\ A\ L)$   
  }  
  **else**  $approx\text{-}bin\ p\ (powr\text{-}float\text{-}interval\ p)\ X\ A$ )

**lemma** *interval-extension-powr*:  $interval\text{-}extension2\ (powr\text{-}float\text{-}interval\ p)\ (powr)$   
 $\langle proof \rangle$

**theorem** *powr-aform-err*:

**assumes**  $x: x \in aform\text{-}err\ e\ X$   
**assumes**  $a: a \in aform\text{-}err\ e\ A$   
**assumes**  $e: e \in UNIV \rightarrow \{-1 .. 1\}$   
**assumes**  $Y: powr\text{-}aform\text{-}err\ p\ X\ A = Some\ Y$   
**shows**  $x\ powr\ a \in aform\text{-}err\ e\ Y$   
 $\langle proof \rangle$

**fun**

$approx\text{-}floatarith :: nat \Rightarrow floatarith \Rightarrow aform\text{-}err\ list \Rightarrow (aform\text{-}err)\ option$

**where**

$approx\text{-}floatarith\ p\ (Add\ a\ b)\ vs =$   
  **do** {  
     $a1 \leftarrow approx\text{-}floatarith\ p\ a\ vs;$   
     $a2 \leftarrow approx\text{-}floatarith\ p\ b\ vs;$   
     $Some\ (add\text{-}aform'\ p\ a1\ a2)$   
  }  
|  $approx\text{-}floatarith\ p\ (Mult\ a\ b)\ vs =$   
  **do** {  
     $a1 \leftarrow approx\text{-}floatarith\ p\ a\ vs;$   
     $a2 \leftarrow approx\text{-}floatarith\ p\ b\ vs;$   
     $Some\ (mult\text{-}aform'\ p\ a1\ a2)$   
  }

```

    }
| approx-floatarith p (Inverse a) vs =
  do {
    a ← approx-floatarith p a vs;
    inverse-aform-err p a
  }
| approx-floatarith p (Minus a) vs =
  map-option (apfst uminus-aform) (approx-floatarith p a vs)
| approx-floatarith p (Num f) vs =
  Some (num-aform (real-of-float f), 0)
| approx-floatarith p (Var i) vs =
  (if i < length vs then Some (vs ! i) else None)
| approx-floatarith p (Abs a) vs =
  do {
    r ← approx-floatarith p a vs;
    let ivl = ivl-of-aform-err p r;
    let i = lower ivl;
    let s = upper ivl;
    if i > 0 then Some r
    else if s < 0 then Some (apfst uminus-aform r)
    else do {
      Some (ivl-err (real-interval (abs-interval ivl)))
    }
  }
}
| approx-floatarith p (Min a b) vs =
  do {
    a1 ← approx-floatarith p a vs;
    a2 ← approx-floatarith p b vs;
    Some (min-aform-err p a1 a2)
  }
| approx-floatarith p (Max a b) vs =
  do {
    a1 ← approx-floatarith p a vs;
    a2 ← approx-floatarith p b vs;
    Some (max-aform-err p a1 a2)
  }
| approx-floatarith p (Floor a) vs =
  approx-un p (λivl. Some (floor-float-interval ivl)) (approx-floatarith p a vs)
| approx-floatarith p (Cos a) vs =
  do {
    a ← approx-floatarith p a vs;
    cos-aform-err p a
  }
| approx-floatarith p Pi vs = Some (ivl-err (real-interval (pi-float-interval p)))
| approx-floatarith p (Sqrt a) vs =
  do {
    a ← approx-floatarith p a vs;
    sqrt-aform-err p a
  }
}

```

```

| approx-floatarith p (Ln a) vs =
  do {
    a ← approx-floatarith p a vs;
    ln-aform-err p a
  }
| approx-floatarith p (Arctan a) vs =
  do {
    a ← approx-floatarith p a vs;
    arctan-aform-err p a
  }
| approx-floatarith p (Exp a) vs =
  do {
    a ← approx-floatarith p a vs;
    exp-aform-err p a
  }
| approx-floatarith p (Power a n) vs =
  do {
    ((a, as), e) ← approx-floatarith p a vs;
    if is-float a ∧ is-float e then Some (power-aform-err p ((a, as), e) n)
    else None
  }
| approx-floatarith p (Powr a b) vs =
  do {
    ae1 ← approx-floatarith p a vs;
    ae2 ← approx-floatarith p b vs;
    powr-aform-err p ae1 ae2
  }

```

**lemma** *uminus-aform-uminus-aform[simp]*:  $uminus-aform (uminus-aform z) = (z::'a::real-vector\ aform)$   
 ⟨proof⟩

**lemma** *degree-aform-inverse-aform'*:  
 $degree-aform X \leq n \implies degree-aform (fst (inverse-aform' p X)) \leq n$   
 ⟨proof⟩

**lemma** *degree-aform-inverse-aform*:  
**assumes**  $inverse-aform p X = Some Y$   
**assumes**  $degree-aform X \leq n$   
**shows**  $degree-aform (fst Y) \leq n$   
 ⟨proof⟩

**lemma** *degree-aform-ivl-err[simp]*:  $degree-aform (fst (ivl-err a)) = 0$   
 ⟨proof⟩

**lemma** *degree-aform-approx-bin*:  
**assumes**  $approx-bin p ivl X Y = Some Z$   
**assumes**  $degree-aform (fst X) \leq m$   
**assumes**  $degree-aform (fst Y) \leq m$

**shows**  $\text{degree-aform } (\text{fst } Z) \leq m$   
*<proof>*

**lemma** *degree-aform-approx-un*:  
**assumes**  $\text{approx-un } p \text{ ivl } X = \text{Some } Y$   
**assumes**  $\text{case } X \text{ of None} \Rightarrow \text{True} \mid \text{Some } X \Rightarrow \text{degree-aform } (\text{fst } X) \leq d1$   
**shows**  $\text{degree-aform } (\text{fst } Y) \leq d1$   
*<proof>*

**lemma** *degree-aform-num-aform[simp]*:  $\text{degree-aform } (\text{num-aform } x) = 0$   
*<proof>*

**lemma** *degree-max-aform*:  
**assumes**  $\text{degree-aform-err } x \leq d$   
**assumes**  $\text{degree-aform-err } y \leq d$   
**shows**  $\text{degree-aform-err } (\text{max-aform-err } p \ x \ y) \leq d$   
*<proof>*

**lemma** *degree-min-aform*:  
**assumes**  $\text{degree-aform-err } x \leq d$   
**assumes**  $\text{degree-aform-err } y \leq d$   
**shows**  $\text{degree-aform-err } ((\text{min-aform-err } p \ x \ y)) \leq d$   
*<proof>*

**lemma** *degree-aform-acc-err*:  
 $\text{degree-aform } (\text{fst } (\text{acc-err } p \ X \ e)) \leq d$   
**if**  $\text{degree-aform } (\text{fst } X) \leq d$   
*<proof>*

**lemma** *degree-pdev-upd-degree*:  
**assumes**  $\text{degree } b \leq \text{Suc } n$   
**assumes**  $\text{degree } b \leq \text{Suc } (\text{degree-aform-err } X)$   
**assumes**  $\text{degree-aform-err } X \leq n$   
**shows**  $\text{degree } (\text{pdev-upd } b \ (\text{degree-aform-err } X) \ 0) \leq n$   
*<proof>*

**lemma** *degree-aform-err-inverse-aform-err*:  
**assumes**  $\text{inverse-aform-err } p \ X = \text{Some } Y$   
**assumes**  $\text{degree-aform-err } X \leq n$   
**shows**  $\text{degree-aform-err } Y \leq n$   
*<proof>*

**lemma** *degree-aform-err-affine-unop*:  
 $\text{degree-aform-err } (\text{affine-unop } p \ a \ b \ d \ X) \leq n$   
**if**  $\text{degree-aform-err } X \leq n$   
*<proof>*

**lemma** *degree-aform-err-min-range-mono*:

**assumes** *min-range-mono*  $p$   $F D l u X = \text{Some } Y$   
**assumes** *degree-aform-err*  $X \leq n$   
**shows** *degree-aform-err*  $Y \leq n$   
*<proof>*

**lemma** *degree-aform-err-min-range-antimono*:  
**assumes** *min-range-antimono*  $p$   $F D l u X = \text{Some } Y$   
**assumes** *degree-aform-err*  $X \leq n$   
**shows** *degree-aform-err*  $Y \leq n$   
*<proof>*

**lemma** *degree-aform-err-cos-aform-err*:  
**assumes** *cos-aform-err*  $p$   $X = \text{Some } Y$   
**assumes** *degree-aform-err*  $X \leq n$   
**shows** *degree-aform-err*  $Y \leq n$   
*<proof>*

**lemma** *degree-aform-err-sqrt-aform-err*:  
**assumes** *sqrt-aform-err*  $p$   $X = \text{Some } Y$   
**assumes** *degree-aform-err*  $X \leq n$   
**shows** *degree-aform-err*  $Y \leq n$   
*<proof>*

**lemma** *degree-aform-err-arctan-aform-err*:  
**assumes** *arctan-aform-err*  $p$   $X = \text{Some } Y$   
**assumes** *degree-aform-err*  $X \leq n$   
**shows** *degree-aform-err*  $Y \leq n$   
*<proof>*

**lemma** *degree-aform-err-exp-aform-err*:  
**assumes** *exp-aform-err*  $p$   $X = \text{Some } Y$   
**assumes** *degree-aform-err*  $X \leq n$   
**shows** *degree-aform-err*  $Y \leq n$   
*<proof>*

**lemma** *degree-aform-err-ln-aform-err*:  
**assumes** *ln-aform-err*  $p$   $X = \text{Some } Y$   
**assumes** *degree-aform-err*  $X \leq n$   
**shows** *degree-aform-err*  $Y \leq n$   
*<proof>*

**lemma** *degree-aform-err-power-aform-err*:  
**assumes** *degree-aform-err*  $X \leq n$   
**shows** *degree-aform-err* (*power-aform-err*  $p$   $X$   $m$ )  $\leq n$   
*<proof>*

**lemma** *degree-aform-err-powr-aform-err*:  
**assumes** *powr-aform-err*  $p$   $X$   $Z = \text{Some } Y$   
**assumes** *degree-aform-err*  $X \leq n$

**assumes** *degree-aform-err*  $Z \leq n$   
**shows** *degree-aform-err*  $Y \leq n$   
 ⟨*proof*⟩

**lemma** *approx-floatarith-degree*:  
**assumes** *approx-floatarith*  $p \text{ ra } VS = \text{Some } X$   
**assumes**  $\bigwedge V. V \in \text{set } VS \implies \text{degree-aform-err } V \leq d$   
**shows** *degree-aform-err*  $X \leq d$   
 ⟨*proof*⟩

**definition** *affine-extension2* **where**  
*affine-extension2* *fctn-aff* *fctn*  $\longleftrightarrow$  (  
 $\forall d \ a1 \ a2 \ X \ e2.$   
*fctn-aff*  $d \ a1 \ a2 = \text{Some } X \longrightarrow$   
 $e2 \in UNIV \rightarrow \{-1..1\} \longrightarrow$   
 $d \geq \text{degree-aform } a1 \longrightarrow$   
 $d \geq \text{degree-aform } a2 \longrightarrow$   
 $(\exists e3 \in UNIV \rightarrow \{-1..1\}.$   
 $(\text{fctn } (\text{aform-val } e2 \ a1) (\text{aform-val } e2 \ a2) = \text{aform-val } e3 \ X \wedge$   
 $(\forall n. n < d \longrightarrow e3 \ n = e2 \ n) \wedge$   
 $\text{aform-val } e2 \ a1 = \text{aform-val } e3 \ a1 \wedge \text{aform-val } e2 \ a2 = \text{aform-val } e3 \ a2)))$

**lemma** *affine-extension2E*:  
**assumes** *affine-extension2* *fctn-aff* *fctn*  
**assumes** *fctn-aff*  $d \ a1 \ a2 = \text{Some } X$   
 $e \in UNIV \rightarrow \{-1..1\}$   
 $d \geq \text{degree-aform } a1$   
 $d \geq \text{degree-aform } a2$   
**obtains**  $e'$  **where**  $e' \in UNIV \rightarrow \{-1..1\}$   
 $\text{fctn } (\text{aform-val } e \ a1) (\text{aform-val } e \ a2) = \text{aform-val } e' \ X$   
 $\bigwedge n. n < d \implies e' \ n = e \ n$   
 $\text{aform-val } e \ a1 = \text{aform-val } e' \ a1$   
 $\text{aform-val } e \ a2 = \text{aform-val } e' \ a2$   
 ⟨*proof*⟩

**lemma** *aform-err-uminus-aform*:  
 $- x \in \text{aform-err } e \ (\text{uminus-aform } X, \text{ba})$   
**if**  $e \in UNIV \rightarrow \{-1 .. 1\} \ x \in \text{aform-err } e \ (X, \text{ba})$   
 ⟨*proof*⟩

**definition** *aforms-err*  $e \ (xs::\text{aform-err list}) = \text{listset } (\text{map } (\text{aform-err } e) \ xs)$

**lemma** *aforms-err-Nil[simp]*: *aforms-err*  $e \ [] = \{[]\}$   
**and** *aforms-err-Cons*: *aforms-err*  $e \ (x\#\text{xs}) = \text{set-Cons } (\text{aform-err } e \ x) \ (\text{aforms-err } e \ \text{xs})$   
 ⟨*proof*⟩

**lemma** *in-set-ConsI*:  $a\#b \in \text{set-Cons } A \ B$   
**if**  $a \in A$  **and**  $b \in B$

*<proof>*

**lemma** *mem-aforms-err-Cons-iff[simp]*:  $x\#xs \in \text{aforms-err } e \ (X\#XS) \longleftrightarrow x \in \text{aform-err } e \ X \wedge xs \in \text{aforms-err } e \ XS$

*<proof>*

**lemma** *mem-aforms-err-Cons-iff-Ex-conv*:  $x \in \text{aforms-err } e \ (X\#XS) \longleftrightarrow (\exists y \ ys. x = y\#ys \wedge y \in \text{aform-err } e \ X \wedge ys \in \text{aforms-err } e \ XS)$

*<proof>*

**lemma** *listset-Cons-mem-conv*:

$a \# vs \in \text{listset } AVS \longleftrightarrow (\exists A \ VS. AVS = A \# VS \wedge a \in A \wedge vs \in \text{listset } VS)$

*<proof>*

**lemma** *listset-Nil-mem-conv[simp]*:

$[] \in \text{listset } AVS \longleftrightarrow AVS = []$

*<proof>*

**lemma** *listset-nthD*:  $vs \in \text{listset } VS \Longrightarrow i < \text{length } vs \Longrightarrow vs ! i \in VS ! i$

*<proof>*

**lemma** *length-listsetD*:

$vs \in \text{listset } VS \Longrightarrow \text{length } vs = \text{length } VS$

*<proof>*

**lemma** *length-aforms-errD*:

$vs \in \text{aforms-err } e \ VS \Longrightarrow \text{length } vs = \text{length } VS$

*<proof>*

**lemma** *nth-aforms-errI*:

$vs ! i \in \text{aform-err } e \ (VS ! i)$

**if**  $vs \in \text{aforms-err } e \ VS \ i < \text{length } vs$

*<proof>*

**lemma** *eucl-truncate-down-float[simp]*:  $\text{eucl-truncate-down } p \ x \in \text{float}$

*<proof>*

**lemma** *eucl-truncate-up-float[simp]*:  $\text{eucl-truncate-up } p \ x \in \text{float}$

*<proof>*

**lemma** *trunc-bound-eucl-float[simp]*:  $\text{fst } (\text{trunc-bound-eucl } p \ x) \in \text{float}$

$\text{snd } (\text{trunc-bound-eucl } p \ x) \in \text{float}$

*<proof>*

**lemma** *add-aform'-float*:

$\text{add-aform}' \ p \ x \ y = ((a, b), ba) \Longrightarrow a \in \text{float}$

$\text{add-aform}' \ p \ x \ y = ((a, b), ba) \Longrightarrow ba \in \text{float}$

*<proof>*

**lemma** *uminus-aform-float*:  $uminus\text{-}aform\ (aa, bb) = (a, b) \implies aa \in float \implies a \in float$   
 ⟨proof⟩

**lemma** *mult-aform'-float*:  $mult\text{-}aform'\ p\ x\ y = ((a, b), ba) \implies a \in float$   
 $mult\text{-}aform'\ p\ x\ y = ((a, b), ba) \implies ba \in float$   
 ⟨proof⟩

**lemma** *inverse-aform'-float*:  $inverse\text{-}aform'\ p\ x = ((a, bb), baa) \implies a \in float$   
 ⟨proof⟩

**lemma** *inverse-aform-float*:  
 $inverse\text{-}aform\ p\ x = Some\ ((a, bb), baa) \implies a \in float$   
 ⟨proof⟩

**lemma** *inverse-aform-err-float*:  $inverse\text{-}aform\text{-}err\ p\ x = Some\ ((a, b), ba) \implies a \in float$   
 $inverse\text{-}aform\text{-}err\ p\ x = Some\ ((a, b), ba) \implies ba \in float$   
 ⟨proof⟩

**lemma** *affine-unop-float*:  
 $affine\text{-}unop\ p\ asdf\ aaa\ bba\ h = ((a, b), ba) \implies a \in float$   
 $affine\text{-}unop\ p\ asdf\ aaa\ bba\ h = ((a, b), ba) \implies ba \in float$   
 ⟨proof⟩

**lemma** *min-range-antimono-float*:  
 $min\text{-}range\text{-}antimono\ p\ f\ f'\ i\ g\ h = Some\ ((a, b), ba) \implies a \in float$   
 $min\text{-}range\text{-}antimono\ p\ f\ f'\ i\ g\ h = Some\ ((a, b), ba) \implies ba \in float$   
 ⟨proof⟩

**lemma** *min-range-mono-float*:  
 $min\text{-}range\text{-}mono\ p\ f\ f'\ i\ g\ h = Some\ ((a, b), ba) \implies a \in float$   
 $min\text{-}range\text{-}mono\ p\ f\ f'\ i\ g\ h = Some\ ((a, b), ba) \implies ba \in float$   
 ⟨proof⟩

**lemma** *in-float-timesI*:  $a \in float$  **if**  $b = a * 2$   $b \in float$   
 ⟨proof⟩

**lemma** *interval-extension-floor*:  $interval\text{-}extension1\ (\lambda ivl.\ Some\ (floor\text{-}float\text{-}interval\ ivl))\ floor$   
 ⟨proof⟩

**lemma** *approx-floatarith-Elem*:  
**assumes**  $approx\text{-}floatarith\ p\ ra\ VS = Some\ X$   
**assumes**  $e: e \in UNIV \rightarrow \{-1 .. 1\}$   
**assumes**  $vs \in aforms\text{-}err\ e\ VS$   
**shows**  $interpret\text{-}floatarith\ ra\ vs \in aform\text{-}err\ e\ X$   
 ⟨proof⟩

**primrec** *approx-floatariths-aformerr* ::  
*nat*  $\Rightarrow$  *floatarith list*  $\Rightarrow$  *aform-err list*  $\Rightarrow$  *aform-err list option*  
**where**  
*approx-floatariths-aformerr* - [] - = *Some* []  
| *approx-floatariths-aformerr* *p* (*a#bs*) *vs* =  
  *do* {  
    *a*  $\leftarrow$  *approx-floatarith* *p* *a* *vs*;  
    *r*  $\leftarrow$  *approx-floatariths-aformerr* *p* *bs* *vs*;  
    *Some* (*a#r*)  
  }

**lemma** *approx-floatariths-Elem*:  
**assumes** *e*  $\in$  *UNIV*  $\rightarrow$   $\{-1 .. 1\}$   
**assumes** *approx-floatariths-aformerr* *p* *ra* *VS* = *Some* *X*  
**assumes** *vs*  $\in$  *aforms-err* *e* *VS*  
**shows** *interpret-floatariths* *ra* *vs*  $\in$  *aforms-err* *e* *X*  
*<proof>*

**lemma** *fold-max-mono*:  
**fixes** *x::'a::linorder*  
**shows**  $x \leq y \implies \text{fold max } zs \ x \leq \text{fold max } zs \ y$   
*<proof>*

**lemma** *fold-max-le-self*:  
**fixes** *y::'a::linorder*  
**shows**  $y \leq \text{fold max } ys \ y$   
*<proof>*

**lemma** *fold-max-le*:  
**fixes** *x::'a::linorder*  
**shows**  $x \in \text{set } xs \implies x \leq \text{fold max } xs \ z$   
*<proof>*

**abbreviation** *degree-aforms-err*  $\equiv$  *degrees o map (snd o fst)*

**definition** *aforms-err-to-aforms* *d* *xs* =  
(*map* ( $\lambda(d, x). \text{aform-err-to-aform } x \ d$ ) (*zip* [*d..<d + length xs*] *xs*))

**lemma** *aform-vals-empty[simp]*: *aform-vals* *e'* [] = []  
*<proof>*

**lemma** *aforms-err-to-aforms-Nil[simp]*: (*aforms-err-to-aforms* *n* []) = []  
*<proof>*

**lemma** *aforms-err-to-aforms-Cons[simp]*:  
*aforms-err-to-aforms* *n* (*X # XS*) = *aform-err-to-aform* *X* *n* # *aforms-err-to-aforms*  
(*Suc* *n*) *XS*  
*<proof>*

**lemma** *degree-aform-err-to-aform-le*:

$\text{degree-aform (aform-err-to-aform } X \ n) \leq \max (\text{degree-aform-err } X) (\text{Suc } n)$

*<proof>*

**lemma** *less-degree-aform-aform-err-to-aformD*:  $i < \text{degree-aform (aform-err-to-aform } X \ n) \implies i < \max (\text{Suc } n) (\text{degree-aform-err } X)$

*<proof>*

**lemma** *pdevs-domain-aform-err-to-aform*:

$\text{pdevs-domain (snd (aform-err-to-aform } X \ n)) = \text{pdevs-domain (snd (fst } X)) \cup$   
 $(\text{if snd } X = 0 \text{ then } \{\} \text{ else } \{n\})$

**if**  $n \geq \text{degree-aform-err } X$

*<proof>*

**lemma** *length-aforms-err-to-aforms[simp]*:  $\text{length (aforms-err-to-aforms } i \ XS) = \text{length } XS$

*<proof>*

**lemma** *aforms-err-to-aforms-ex*:

**assumes**  $X: x \in \text{aforms-err } e \ X$

**assumes**  $\text{deg: degree-aforms-err } X \leq n$

**assumes**  $e: e \in \text{UNIV} \rightarrow \{-1 .. 1\}$

**shows**  $\exists e' \in \text{UNIV} \rightarrow \{-1 .. 1\}. x = \text{aform-vals } e' (\text{aforms-err-to-aforms } n \ X)$

$\wedge$

$(\forall i < n. e' \ i = e \ i)$

*<proof>*

**lemma** *aforms-err-to-aformsE*:

**assumes**  $X: x \in \text{aforms-err } e \ X$

**assumes**  $\text{deg: degree-aforms-err } X \leq n$

**assumes**  $e: e \in \text{UNIV} \rightarrow \{-1 .. 1\}$

**obtains**  $e'$  **where**  $x = \text{aform-vals } e' (\text{aforms-err-to-aforms } n \ X) \ e' \in \text{UNIV} \rightarrow \{-1 .. 1\}$

$\wedge i. i < n \implies e' \ i = e \ i$

*<proof>*

**definition** *approx-floatariths*  $p \ ea \ as =$

*do* {

*let*  $da = (\text{degree-aforms } as);$

*let*  $aes = (\text{map } (\lambda x. (x, 0)) \ as);$

$rs \leftarrow \text{approx-floatariths-aformerr } p \ ea \ aes;$

*let*  $d = \max \ da \ (\text{degree-aforms-err } (rs));$

*Some*  $(\text{aforms-err-to-aforms } d \ rs)$

}

**lemma** *listset-sings[simp]*:

$\text{listset (map } (\lambda x. \{f \ x\}) \ as) = \{\text{map } f \ as\}$

*<proof>*

**lemma** *approx-floatariths-outer*:  
**assumes** *approx-floatariths p ea as = Some XS*  
**assumes** *vs ∈ Joints as*  
**shows** *(interpret-floatariths ea vs @ vs) ∈ Joints (XS @ as)*  
*<proof>*

**lemma** *length-eq-NilI*: *length [] = length []*  
**and** *length-eq-ConsI*: *length xs = length ys ⇒ length (x#xs) = length (y#ys)*  
*<proof>*

## 5.7 Generic operations on Affine Forms in Euclidean Space

**lemma** *pdevs-val-domain-cong*:  
**assumes** *b = d*  
**assumes**  $\bigwedge i. i \in \text{pdevs-domain } b \Rightarrow a \ i = c \ i$   
**shows** *pdevs-val a b = pdevs-val c d*  
*<proof>*

**lemma** *fresh-JointsI*:  
**assumes** *xs ∈ Joints XS*  
**assumes** *list-all (λY. pdevs-domain (snd X) ∩ pdevs-domain (snd Y) = {}) XS*  
**assumes** *x ∈ Affine X*  
**shows** *x#xs ∈ Joints (X#XS)*  
*<proof>*

**primrec** *approx-slp::nat ⇒ slp ⇒ aform-err list ⇒ aform-err list option*

**where**

*approx-slp p [] xs = Some xs*  
| *approx-slp p (ea # eas) xs =*  
  *do {*  
    *r ← approx-floatarith p ea xs;*  
    *approx-slp p eas (r#xs)*  
  *}*

**lemma** *Nil-mem-Joints[intro, simp]*: *[] ∈ Joints []*  
*<proof>*

**lemma** *map-nth-Joints*: *xs ∈ Joints XS ⇒ (∧i. i ∈ set is ⇒ i < length XS)*  
 $\Rightarrow \text{map } (nth \ xs) \ is \ @ \ xs \in \text{Joints } (\text{map } (nth \ XS) \ is \ @ \ XS)$   
*<proof>*

**lemma** *map-nth-Joints'*: *xs ∈ Joints XS ⇒ (∧i. i ∈ set is ⇒ i < length XS)*  
 $\Rightarrow \text{map } (nth \ xs) \ is \in \text{Joints } (\text{map } (nth \ XS) \ is)$   
*<proof>*

**lemma** *approx-slp-Elem*:  
**assumes** *e: e ∈ UNIV → {-1 .. 1}*  
**assumes** *vs ∈ aforms-err e VS*

**assumes** *approx-slp*  $p$   $ra$   $VS = Some\ X$   
**shows** *interpret-slp*  $ra$   $vs \in aforms\ err\ e\ X$   
*<proof>*

**definition** *approx-slp-outer*  $p$   $n$   $slp$   $XS =$   
*do* {  
  *let*  $d = degree\ aforms\ XS$ ;  
  *let*  $XSe = (map\ (\lambda x. (x, 0))\ XS)$ ;  
   $rs \leftarrow approx\ slp\ p\ slp\ XSe$ ;  
  *let*  $rs' = take\ n\ rs$ ;  
  *let*  $d' = max\ d\ (degree\ aforms\ err\ rs')$ ;  
  *Some* (*aforms-err-to-aforms*  $d'\ rs'$ )  
}

**lemma** *take-in-listsetI*:  $xs \in listset\ XS \implies take\ n\ xs \in listset\ (take\ n\ XS)$   
*<proof>*

**lemma** *take-in-aforms-errI*:  $take\ n\ xs \in aforms\ err\ e\ (take\ n\ XS)$   
**if**  $xs \in aforms\ err\ e\ XS$   
*<proof>*

**theorem** *approx-slp-outer*:  
**assumes** *approx-slp-outer*  $p$   $n$   $slp$   $XS = Some\ RS$   
**assumes** *slp*:  $slp = slp\ of\ fas\ fas\ n = length\ fas$   
**assumes**  $xs \in Joints\ XS$   
**shows** *interpret-floatariths*  $fas\ xs @ xs \in Joints\ (RS @ XS)$   
*<proof>*

**theorem** *approx-slp-outer-plain*:  
**assumes** *approx-slp-outer*  $p$   $n$   $slp$   $XS = Some\ RS$   
**assumes** *slp*:  $slp = slp\ of\ fas\ fas\ n = length\ fas$   
**assumes**  $xs \in Joints\ XS$   
**shows** *interpret-floatariths*  $fas\ xs \in Joints\ RS$   
*<proof>*

**end**

**end**

## 6 Counterclockwise

**theory** *Counterclockwise*  
**imports** *HOL-Analysis.Multivariate-Analysis*  
**begin**

### 6.1 Auxiliary Lemmas

**lemma** *convex3-alt*:

**fixes**  $x\ y\ z :: 'a :: \text{real-vector}$   
**assumes**  $0 \leq a\ 0 \leq b\ 0 \leq c\ a + b + c = 1$   
**obtains**  $u\ v$  **where**  $a *_R x + b *_R y + c *_R z = x + u *_R (y - x) + v *_R (z - x)$   
**and**  $0 \leq u\ 0 \leq v\ u + v \leq 1$   
 $\langle \text{proof} \rangle$

**lemma** (in *ordered-ab-group-add*) *add-nonpos-eq-0-iff*:  
**assumes**  $x: 0 \geq x$  **and**  $y: 0 \geq y$   
**shows**  $x + y = 0 \longleftrightarrow x = 0 \wedge y = 0$   
 $\langle \text{proof} \rangle$

**lemma** *sum-nonpos-eq-0-iff*:  
**fixes**  $f :: 'a \Rightarrow 'b :: \text{ordered-ab-group-add}$   
**shows**  $\llbracket \text{finite } A; \forall x \in A. f\ x \leq 0 \rrbracket \Longrightarrow \text{sum } f\ A = 0 \longleftrightarrow (\forall x \in A. f\ x = 0)$   
 $\langle \text{proof} \rangle$

**lemma** *fold-if-in-set*:  
*fold*  $(\lambda x\ m. \text{if } P\ x\ m\ \text{then } x\ \text{else } m)\ xs\ x \in \text{set } (x \# xs)$   
 $\langle \text{proof} \rangle$

## 6.2 Sort Elements of a List

**locale** *linorder-list0* = **fixes**  $le :: 'a \Rightarrow 'a \Rightarrow \text{bool}$   
**begin**

**definition** *min-for*  $a\ b = (\text{if } le\ a\ b\ \text{then } a\ \text{else } b)$

**lemma** *min-for-in[simp]*:  $x \in S \Longrightarrow y \in S \Longrightarrow \text{min-for } x\ y \in S$   
 $\langle \text{proof} \rangle$

**lemma** *fold-min-eqI1*:  $\text{fold } \text{min-for } ys\ y \notin \text{set } ys \Longrightarrow \text{fold } \text{min-for } ys\ y = y$   
 $\langle \text{proof} \rangle$

**function** *selsort* **where**  
 $\text{selsort } [] = []$   
 $| \text{selsort } (y \# ys) = (\text{let}$   
 $\quad xm = \text{fold } \text{min-for } ys\ y;$   
 $\quad xs' = \text{List.remove1 } xm\ (y \# ys)$   
 $\quad \text{in } (xm \# \text{selsort } xs'))$   
 $\langle \text{proof} \rangle$

**termination**  
 $\langle \text{proof} \rangle$

**lemma** *in-set-selsort-eq*:  $x \in \text{set } (\text{selsort } xs) \longleftrightarrow x \in (\text{set } xs)$   
 $\langle \text{proof} \rangle$

**lemma** *set-selsort[simp]*:  $\text{set } (\text{selsort } xs) = \text{set } xs$   
 $\langle \text{proof} \rangle$

**lemma** *length-selsort[simp]*:  $\text{length } (\text{selsort } xs) = \text{length } xs$   
(proof)

**lemma** *distinct-selsort[simp]*:  $\text{distinct } (\text{selsort } xs) = \text{distinct } xs$   
(proof)

**lemma** *selsort-eq-empty-iff[simp]*:  $\text{selsort } xs = [] \longleftrightarrow xs = []$   
(proof)

**inductive** *sortedP* :: 'a list  $\Rightarrow$  bool **where**  
  *Nil*: *sortedP* []  
  | *Cons*:  $\forall y \in \text{set } ys. \text{le } x \ y \Longrightarrow \text{sortedP } ys \Longrightarrow \text{sortedP } (x \# \text{ys})$

**inductive-cases**  
  *sortedP-Nil*: *sortedP* [] **and**  
  *sortedP-Cons*: *sortedP* (x#xs)

**inductive-simps**  
  *sortedP-Nil-iff*: *sortedP* Nil **and**  
  *sortedP-Cons-iff*: *sortedP* (Cons x xs)

**lemma** *sortedP-append-iff*:  
   $\text{sortedP } (xs \ @ \ ys) = (\text{sortedP } xs \ \& \ \text{sortedP } ys \ \& \ (\forall x \in \text{set } xs. \forall y \in \text{set } ys. \text{le } x \ y))$   
(proof)

**lemma** *sortedP-appendI*:  
   $\text{sortedP } xs \Longrightarrow \text{sortedP } ys \Longrightarrow (\bigwedge x \ y. x \in \text{set } xs \Longrightarrow y \in \text{set } ys \Longrightarrow \text{le } x \ y) \Longrightarrow \text{sortedP } (xs \ @ \ ys)$   
(proof)

**lemma** *sorted-nth-less*:  $\text{sortedP } xs \Longrightarrow i < j \Longrightarrow j < \text{length } xs \Longrightarrow \text{le } (xs \ ! \ i) \ (xs \ ! \ j)$   
(proof)

**lemma** *sorted-butlastI[intro, simp]*:  $\text{sortedP } xs \Longrightarrow \text{sortedP } (\text{butlast } xs)$   
(proof)

**lemma** *sortedP-right-of-append1*:  
  **assumes** *sortedP* (zs@[z])  
  **assumes**  $y \in \text{set } zs$   
  **shows**  $\text{le } y \ z$   
(proof)

**lemma** *sortedP-right-of-last*:  
  **assumes** *sortedP* zs  
  **assumes**  $y \in \text{set } zs \ y \neq \text{last } zs$   
  **shows**  $\text{le } y \ (\text{last } zs)$

$\langle proof \rangle$

**lemma** *selsort-singleton-iff*:  $selsort\ xs = [x] \longleftrightarrow xs = [x]$   
 $\langle proof \rangle$

**lemma** *hd-last-sorted*:  
**assumes** *sortedP xs length xs > 1*  
**shows**  $le\ (hd\ xs)\ (last\ xs)$   
 $\langle proof \rangle$

**end**

**lemma** (in *comm-monoid-add*) *sum-list-distinct-selsort*:  
**assumes** *distinct xs*  
**shows**  $sum\ list\ (linorder\ list0.\ selsort\ le\ xs) = sum\ list\ xs$   
 $\langle proof \rangle$

**declare** *linorder-list0.sortedP-Nil-iff*[code]  
*linorder-list0.sortedP-Cons-iff*[code]  
*linorder-list0.selsort.simps*[code]  
*linorder-list0.min-for-def*[code]

**locale** *linorder-list* = *linorder-list0 le* **for**  $le::'a::ab\ group\ add \Rightarrow - +$   
**fixes**  $S$   
**assumes** *order-refl*:  $a \in S \Longrightarrow le\ a\ a$   
**assumes** *trans'*:  $a \in S \Longrightarrow b \in S \Longrightarrow c \in S \Longrightarrow a \neq b \Longrightarrow b \neq c \Longrightarrow a \neq c$   
 $\Longrightarrow$   
 $le\ a\ b \Longrightarrow le\ b\ c \Longrightarrow le\ a\ c$   
**assumes** *antisym*:  $a \in S \Longrightarrow b \in S \Longrightarrow le\ a\ b \Longrightarrow le\ b\ a \Longrightarrow a = b$   
**assumes** *linear'*:  $a \in S \Longrightarrow b \in S \Longrightarrow a \neq b \Longrightarrow le\ a\ b \vee le\ b\ a$   
**begin**

**lemma** *trans*:  $a \in S \Longrightarrow b \in S \Longrightarrow c \in S \Longrightarrow le\ a\ b \Longrightarrow le\ b\ c \Longrightarrow le\ a\ c$   
 $\langle proof \rangle$

**lemma** *linear*:  $a \in S \Longrightarrow b \in S \Longrightarrow le\ a\ b \vee le\ b\ a$   
 $\langle proof \rangle$

**lemma** *min-le1*:  $w \in S \Longrightarrow y \in S \Longrightarrow le\ (min\ for\ w\ y)\ y$   
**and** *min-le2*:  $w \in S \Longrightarrow y \in S \Longrightarrow le\ (min\ for\ w\ y)\ w$   
 $\langle proof \rangle$

**lemma** *fold-min*:  
**assumes**  $set\ xs \subseteq S$   
**shows**  $list\ all\ (\lambda y.\ le\ (fold\ min\ for\ (tl\ xs)\ (hd\ xs))\ y)\ xs$   
 $\langle proof \rangle$

**lemma**  
*sortedP-selsort*:

**assumes**  $set\ xs \subseteq S$   
**shows**  $sortedP\ (selsort\ xs)$   
 $\langle proof \rangle$

**end**

### 6.3 Abstract CCW Systems

**locale**  $ccw\text{-}system0 =$   
**fixes**  $ccw::'a \Rightarrow 'a \Rightarrow bool$   
**and**  $S::'a\ set$   
**begin**

**abbreviation**  $indelta\ t\ p\ q\ r \equiv ccw\ t\ q\ r \wedge ccw\ p\ t\ r \wedge ccw\ p\ q\ t$   
**abbreviation**  $insquare\ p\ q\ r\ s \equiv ccw\ p\ q\ r \wedge ccw\ q\ r\ s \wedge ccw\ r\ s\ p \wedge ccw\ s\ p\ q$

**end**

**abbreviation**  $distinct3\ p\ q\ r \equiv \neg(p = q \vee p = r \vee q = r)$   
**abbreviation**  $distinct4\ p\ q\ r\ s \equiv \neg(p = q \vee p = r \vee p = s \vee \neg distinct3\ q\ r\ s)$   
**abbreviation**  $distinct5\ p\ q\ r\ s\ t \equiv \neg(p = q \vee p = r \vee p = s \vee p = t \vee \neg distinct4\ q\ r\ s\ t)$

**abbreviation**  $in3\ S\ p\ q\ r \equiv p \in S \wedge q \in S \wedge r \in S$   
**abbreviation**  $in4\ S\ p\ q\ r\ s \equiv in3\ S\ p\ q\ r \wedge s \in S$   
**abbreviation**  $in5\ S\ p\ q\ r\ s\ t \equiv in4\ S\ p\ q\ r\ s \wedge t \in S$

**locale**  $ccw\text{-}system12 = ccw\text{-}system0 +$   
**assumes**  $cyclic: ccw\ p\ q\ r \Longrightarrow ccw\ q\ r\ p$   
**assumes**  $ccw\text{-}antisym: distinct3\ p\ q\ r \Longrightarrow in3\ S\ p\ q\ r \Longrightarrow ccw\ p\ q\ r \Longrightarrow \neg ccw\ p\ r\ q$

**locale**  $ccw\text{-}system123 = ccw\text{-}system12 +$   
**assumes**  $nondegenerate: distinct3\ p\ q\ r \Longrightarrow in3\ S\ p\ q\ r \Longrightarrow ccw\ p\ q\ r \vee ccw\ p\ r\ q$   
**begin**

**lemma**  $not\text{-}ccw\text{-}eq: distinct3\ p\ q\ r \Longrightarrow in3\ S\ p\ q\ r \Longrightarrow \neg ccw\ p\ q\ r \longleftrightarrow ccw\ p\ r\ q$   
 $\langle proof \rangle$

**end**

**locale**  $ccw\text{-}system4 = ccw\text{-}system123 +$   
**assumes**  $interior:$   
 $distinct4\ p\ q\ r\ t \Longrightarrow in4\ S\ p\ q\ r\ t \Longrightarrow ccw\ t\ q\ r \Longrightarrow ccw\ p\ t\ r \Longrightarrow ccw\ p\ q\ t$   
 $\Longrightarrow ccw\ p\ q\ r$   
**begin**

**lemma**  $interior'$ :

*distinct4*  $p q r t \implies in4 S p q r t \implies ccw p q t \implies ccw q r t \implies ccw r p t \implies$   
 $ccw p q r$   
 ⟨*proof*⟩

**end**

**locale** *ccw-system1235'* = *ccw-system123* +  
**assumes** *dual-transitive*:  
 $distinct5 p q r s t \implies in5 S p q r s t \implies$   
 $ccw s t p \implies ccw s t q \implies ccw s t r \implies ccw t p q \implies ccw t q r \implies ccw t p r$

**locale** *ccw-system1235* = *ccw-system123* +  
**assumes** *transitive*:  $distinct5 p q r s t \implies in5 S p q r s t \implies$   
 $ccw t s p \implies ccw t s q \implies ccw t s r \implies ccw t p q \implies ccw t q r \implies ccw t p r$   
**begin**

**lemmas** *ccw-axioms* = *cyclic nondegenerate ccw-antisym transitive*

**sublocale** *ccw-system1235'*  
 ⟨*proof*⟩

**end**

**locale** *ccw-system* = *ccw-system1235* + *ccw-system4*

**end**

## 7 CCW Vector Space

**theory** *Counterclockwise-Vector*

**imports** *Counterclockwise*

**begin**

**locale** *ccw-vector-space* = *ccw-system12* *ccw S* **for** *ccw::'a::real-vector*  $\implies 'a \implies 'a$   
 $\implies bool$  **and** *S* +

**assumes** *translate-plus[simp]*:  $ccw (a + x) (b + x) (c + x) \longleftrightarrow ccw a b c$

**assumes** *scaleR1-eq[simp]*:  $0 < e \implies ccw 0 (e *_{R} a) b = ccw 0 a b$

**assumes** *uminus1[simp]*:  $ccw 0 (-a) b = ccw 0 b a$

**assumes** *add1*:  $ccw 0 a b \implies ccw 0 c b \implies ccw 0 (a + c) b$

**begin**

**lemma** *translate-plus'[simp]*:

$ccw (x + a) (x + b) (x + c) \longleftrightarrow ccw a b c$

⟨*proof*⟩

**lemma** *uminus2[simp]*:  $ccw 0 a (-b) = ccw 0 b a$

⟨*proof*⟩

**lemma** *uminus-all[simp]*:  $ccw (-a) (-b) (-c) \longleftrightarrow ccw a b c$

*<proof>*

**lemma** *translate-origin: NO-MATCH*  $0 p \implies ccw\ p\ q\ r \longleftrightarrow ccw\ 0\ (q - p)\ (r - p)$   
*<proof>*

**lemma** *translate[simp]*:  $ccw\ a\ (a + b)\ (a + c) \longleftrightarrow ccw\ 0\ b\ c$   
*<proof>*

**lemma** *translate-plus3*:  $ccw\ (a - x)\ (b - x)\ c \longleftrightarrow ccw\ a\ b\ (c + x)$   
*<proof>*

**lemma** *renormalize*:  
 $ccw\ 0\ (a - b)\ (c - a) \implies ccw\ b\ a\ c$   
*<proof>*

**lemma** *cyclicI*:  $ccw\ p\ q\ r \implies ccw\ q\ r\ p$   
*<proof>*

**lemma**  
*scaleR2-eq[simp]*:  
 $0 < e \implies ccw\ 0\ xr\ (e *R P) \longleftrightarrow ccw\ 0\ xr\ P$   
*<proof>*

**lemma** *scaleR1-nonzero-eq*:  
 $e \neq 0 \implies ccw\ 0\ (e *R a)\ b = (if\ e > 0\ then\ ccw\ 0\ a\ b\ else\ ccw\ 0\ b\ a)$   
*<proof>*

**lemma** *neg-scaleR[simp]*:  $x < 0 \implies ccw\ 0\ (x *R b)\ c \longleftrightarrow ccw\ 0\ c\ b$   
*<proof>*

**lemma**  
*scaleR1*:  
 $0 < e \implies ccw\ 0\ xr\ P \implies ccw\ 0\ (e *R xr)\ P$   
*<proof>*

**lemma**  
*add3*:  $ccw\ 0\ a\ b \wedge ccw\ 0\ a\ c \implies ccw\ 0\ a\ (b + c)$   
*<proof>*

**lemma** *add3-self[simp]*:  $ccw\ 0\ p\ (p + q) \longleftrightarrow ccw\ 0\ p\ q$   
*<proof>*

**lemma** *add2-self[simp]*:  $ccw\ 0\ (p + q)\ p \longleftrightarrow ccw\ 0\ q\ p$   
*<proof>*

**lemma** *scale-add3[simp]*:  $ccw\ 0\ a\ (x *R a + b) \longleftrightarrow ccw\ 0\ a\ b$   
*<proof>*

**lemma** *scale-add3'*[simp]:  $ccw\ 0\ a\ (b + x *_{R}\ a) \longleftrightarrow ccw\ 0\ a\ b$   
**and** *scale-minus3'*[simp]:  $ccw\ 0\ a\ (x *_{R}\ a - b) \longleftrightarrow ccw\ 0\ b\ a$   
**and** *scale-minus3'*[simp]:  $ccw\ 0\ a\ (b - x *_{R}\ a) \longleftrightarrow ccw\ 0\ a\ b$   
 ⟨*proof*⟩

**lemma** *sum*:  
**assumes** *fin*: *finite* *X*  
**assumes** *ne*:  $X \neq \{\}$   
**assumes** *ncoll*:  $(\bigwedge x. x \in X \implies ccw\ 0\ a\ (f\ x))$   
**shows**  $ccw\ 0\ a\ (sum\ f\ X)$   
 ⟨*proof*⟩

**lemma** *sum2*:  
**assumes** *fin*: *finite* *X*  
**assumes** *ne*:  $X \neq \{\}$   
**assumes** *ncoll*:  $(\bigwedge x. x \in X \implies ccw\ 0\ (f\ x)\ a)$   
**shows**  $ccw\ 0\ (sum\ f\ X)\ a$   
 ⟨*proof*⟩

**lemma** *translate-minus*[simp]:  
 $ccw\ (x - a)\ (x - b)\ (x - c) = ccw\ (-a)\ (-b)\ (-c)$   
 ⟨*proof*⟩

**end**

**locale** *ccw-convex* = *ccw-system* *ccw* *S* **for** *ccw* **and** *S*::'*a*::*real-vector set* +  
**fixes** *oriented*  
**assumes** *convex2*:  
 $u \geq 0 \implies v \geq 0 \implies u + v = 1 \implies ccw\ a\ b\ c \implies ccw\ a\ b\ d \implies oriented\ a$   
 $b \implies$   
 $ccw\ a\ b\ (u *_{R}\ c + v *_{R}\ d)$

**begin**

**lemma** *convex-hull*:  
**assumes** [*intro*, *simp*]: *finite* *C*  
**assumes** *ccw*:  $\bigwedge c. c \in C \implies ccw\ a\ b\ c$   
**assumes** *ch*:  $x \in convex\ hull\ C$   
**assumes** *oriented*: *oriented* *a* *b*  
**shows**  $ccw\ a\ b\ x$   
 ⟨*proof*⟩

**end**

**end**

## 8 CCW for Nonaligned Points in the Plane

**theory** *Counterclockwise-2D-Strict*  
**imports**

*Counterclockwise-Vector*  
*Affine-Arithmetic-Auxiliarities*

**begin**

## 8.1 Determinant

**type-synonym** *point* = *real*\**real*

**fun** *det3*::*point* ⇒ *point* ⇒ *point* ⇒ *real* **where** *det3* (*xp*, *yp*) (*xq*, *yq*) (*xr*, *yr*) =  
*xp* \* *yq* + *yp* \* *xr* + *xq* \* *yr* - *yq* \* *xr* - *yp* \* *xq* - *xp* \* *yr*

**lemma** *det3-def'*:

*det3* *p q r* = *fst p* \* *snd q* + *snd p* \* *fst r* + *fst q* \* *snd r* -  
*snd q* \* *fst r* - *snd p* \* *fst q* - *fst p* \* *snd r*  
⟨*proof*⟩

**lemma** *det3-eq-det*: *det3* (*xa*, *ya*) (*xb*, *yb*) (*xc*, *yc*) =

*det* (*vector* [*vector* [*xa*, *ya*, 1], *vector* [*xb*, *yb*, 1], *vector* [*xc*, *yc*, 1]]::*real*<sup>3</sup>)  
⟨*proof*⟩

**declare** *det3.simps*[*simp det*]

**lemma** *det3-self23*[*simp*]: *det3* *a b b* = 0

**and** *det3-self12*[*simp*]: *det3* *b b a* = 0

⟨*proof*⟩

**lemma**

*coll-ex-scaling*:

**assumes** *b* ≠ *c*

**assumes** *d*: *det3* *a b c* = 0

**shows** ∃ *r*. *a* = *b* + *r* \*<sub>R</sub> (*c* - *b*)

⟨*proof*⟩

**lemma** *cramer*: ¬*det3* *s t q* = 0 ⇒

(*det3* *t p r*) = ((*det3* *t q r*) \* (*det3* *s t p*) + (*det3* *t p q*) \* (*det3* *s t r*)) / (*det3* *s t*

*q*)

⟨*proof*⟩

**lemma** *convex-comb-dets*:

**assumes** *det3* *p q r* > 0

**shows** *s* = (*det3* *s q r* / *det3* *p q r*) \*<sub>R</sub> *p* + (*det3* *p s r* / *det3* *p q r*) \*<sub>R</sub> *q* +

(*det3* *p q s* / *det3* *p q r*) \*<sub>R</sub> *r*

(**is** ?*lhs* = ?*rhs*)

⟨*proof*⟩

**lemma** *four-points-aligned*:

**assumes** *c*: *det3* *t p q* = 0 *det3* *t q r* = 0

**assumes** *distinct*: *distinct5* *t s p q r*

**shows** *det3* *t r p* = 0 *det3* *p q r* = 0

*<proof>*

**lemma** *det-identity*:

$$\det3\ t\ p\ q * \det3\ t\ s\ r + \det3\ t\ q\ r * \det3\ t\ s\ p + \det3\ t\ r\ p * \det3\ t\ s\ q = 0$$

*<proof>*

**lemma** *det3-eq-zeroI*:

$$\text{assumes } p = q + x *_{\mathbb{R}} (t - q)$$

$$\text{shows } \det3\ q\ t\ p = 0$$

*<proof>*

**lemma** *det3-rotate*:  $\det3\ a\ b\ c = \det3\ c\ a\ b$

*<proof>*

**lemma** *det3-switch*:  $\det3\ a\ b\ c = -\det3\ a\ c\ b$

*<proof>*

**lemma** *det3-switch'*:  $\det3\ a\ b\ c = -\det3\ b\ a\ c$

*<proof>*

**lemma** *det3-pos-transitive-coll*:

$$\det3\ t\ s\ p > 0 \implies \det3\ t\ s\ r \geq 0 \implies \det3\ t\ p\ q \geq 0 \implies$$

$$\det3\ t\ q\ r > 0 \implies \det3\ t\ s\ q = 0 \implies \det3\ t\ p\ r > 0$$

*<proof>*

**lemma** *det3-pos-transitive*:

$$\det3\ t\ s\ p > 0 \implies \det3\ t\ s\ q \geq 0 \implies \det3\ t\ s\ r \geq 0 \implies \det3\ t\ p\ q \geq 0 \implies$$

$$\det3\ t\ q\ r > 0 \implies \det3\ t\ p\ r > 0$$

*<proof>*

**lemma** *det3-zero-translate-plus[simp]*:  $\det3\ (a + x)\ (b + x)\ (c + x) = 0 \iff \det3\ a\ b\ c = 0$

*<proof>*

**lemma** *det3-zero-translate-plus'[simp]*:  $\det3\ (a)\ (a + b)\ (a + c) = 0 \iff \det3\ 0\ b\ c = 0$

*<proof>*

**lemma**

*det30-zero-scaleR1*:

$$0 < e \implies \det3\ 0\ x\ r\ P = 0 \implies \det3\ 0\ (e *_{\mathbb{R}} x\ r)\ P = 0$$

*<proof>*

**lemma** *det3-same[simp]*:  $\det3\ a\ x\ x = 0$

*<proof>*

**lemma**

*det30-zero-scaleR2*:

$$0 < e \implies \det3\ 0\ P\ x\ r = 0 \implies \det3\ 0\ P\ (e *_{\mathbb{R}} x\ r) = 0$$

*<proof>*

**lemma** *det3-eq-zero*:  $e \neq 0 \implies \det3\ 0\ xr\ (e *_{\mathbb{R}} Q) = 0 \iff \det3\ 0\ xr\ Q = 0$   
*<proof>*

**lemma** *det30-plus-scaled3[simp]*:  $\det3\ 0\ a\ (b + x *_{\mathbb{R}} a) = 0 \iff \det3\ 0\ a\ b = 0$   
*<proof>*

**lemma** *det30-plus-scaled2[simp]*:  
**shows**  $\det3\ 0\ (a + x *_{\mathbb{R}} a)\ b = 0 \iff (\text{if } x = -1 \text{ then True else } \det3\ 0\ a\ b = 0)$   
**(is ?lhs = ?rhs)**  
*<proof>*

**lemma** *det30-uminus2[simp]*:  $\det3\ 0\ (-a)\ (b) = 0 \iff \det3\ 0\ a\ b = 0$   
**and** *det30-uminus3[simp]*:  $\det3\ 0\ a\ (-b) = 0 \iff \det3\ 0\ a\ b = 0$   
*<proof>*

**lemma** *det30-minus-scaled3[simp]*:  $\det3\ 0\ a\ (b - x *_{\mathbb{R}} a) = 0 \iff \det3\ 0\ a\ b = 0$   
*<proof>*

**lemma** *det30-scaled-minus3[simp]*:  $\det3\ 0\ a\ (e *_{\mathbb{R}} a - b) = 0 \iff \det3\ 0\ a\ b = 0$   
*<proof>*

**lemma** *det30-minus-scaled2[simp]*:  
 $\det3\ 0\ (a - x *_{\mathbb{R}} a)\ b = 0 \iff (\text{if } x = 1 \text{ then True else } \det3\ 0\ a\ b = 0)$   
*<proof>*

**lemma** *det3-nonneg-scaleR1*:  
 $0 < e \implies \det3\ 0\ xr\ P \geq 0 \implies \det3\ 0\ (e *_{\mathbb{R}} xr)\ P \geq 0$   
*<proof>*

**lemma** *det3-nonneg-scaleR1-eq*:  
 $0 < e \implies \det3\ 0\ (e *_{\mathbb{R}} xr)\ P \geq 0 \iff \det3\ 0\ xr\ P \geq 0$   
*<proof>*

**lemma** *det3-translate-origin*: *NO-MATCH*  $0\ p \implies \det3\ p\ q\ r = \det3\ 0\ (q - p)\ (r - p)$   
*<proof>*

**lemma** *det3-nonneg-scaleR-segment2*:  
**assumes**  $\det3\ x\ y\ z \geq 0$   
**assumes**  $a > 0$   
**shows**  $\det3\ x\ ((1 - a) *_{\mathbb{R}} x + a *_{\mathbb{R}} y)\ z \geq 0$   
*<proof>*

**lemma** *det3-nonneg-scaleR-segment1*:  
**assumes**  $\det3\ x\ y\ z \geq 0$   
**assumes**  $0 \leq a < 1$

**shows**  $\det3 ((1 - a) *_R x + a *_R y) y z \geq 0$   
*<proof>*

## 8.2 Strict CCW Predicate

**definition**  $ccw' p q r \longleftrightarrow 0 < \det3 p q r$

**interpretation**  $ccw'$ : *ccw-vector-space*  $ccw'$   
*<proof>*

**interpretation**  $ccw'$ : *linorder-list0*  $ccw' x$  **for**  $x$  *<proof>*

**lemma**  $ccw'$ -*contra*:  $ccw' t r q \implies ccw' t q r = False$   
*<proof>*

**lemma**  $not\text{-}ccw'\text{-}eq$ :  $\neg ccw' t p s \longleftrightarrow ccw' t s p \vee \det3 t s p = 0$   
*<proof>*

**lemma**  $neq\text{-}left\text{-}right\text{-}of$ :  $ccw' a b c \implies ccw' a c d \implies b \neq d$   
*<proof>*

**lemma**  $ccw'\text{-}subst\text{-}collinear$ :  
**assumes**  $\det3 t r s = 0$   
**assumes**  $s \neq t$   
**assumes**  $ccw' t r p$   
**shows**  $ccw' t s p \vee ccw' t p s$   
*<proof>*

**lemma**  $ccw'\text{-}sorted\text{-}scaleR$ :  $ccw'.sortedP 0 xs \implies r > 0 \implies ccw'.sortedP 0 (map ((*_R) r) xs)$   
*<proof>*

## 8.3 Collinearity

**abbreviation**  $coll a b c \equiv \det3 a b c = 0$

**lemma**  $coll\text{-}zero$ [*intro, simp*]:  $coll 0 z 0$   
*<proof>*

**lemma**  $coll\text{-}zero1$ [*intro, simp*]:  $coll 0 0 z$   
*<proof>*

**lemma**  $coll\text{-}self$ [*intro, simp*]:  $coll 0 z z$   
*<proof>*

**lemma**  $ccw'\text{-}not\text{-}coll$ :  
 $ccw' a b c \implies \neg coll a b c$   
 $ccw' a b c \implies \neg coll a c b$   
 $ccw' a b c \implies \neg coll b a c$   
 $ccw' a b c \implies \neg coll b c a$

$ccw' a b c \implies \neg coll\ c\ a\ b$   
 $ccw' a b c \implies \neg coll\ c\ b\ a$   
 <proof>

**lemma** *coll-add*:  $coll\ 0\ x\ y \implies coll\ 0\ x\ z \implies coll\ 0\ x\ (y + z)$   
 <proof>

**lemma** *coll-scaleR-left-eq[simp]*:  $coll\ 0\ (r *_{\mathbb{R}} x)\ y \longleftrightarrow r = 0 \vee coll\ 0\ x\ y$   
 <proof>

**lemma** *coll-scaleR-right-eq[simp]*:  $coll\ 0\ y\ (r *_{\mathbb{R}} x) \longleftrightarrow r = 0 \vee coll\ 0\ y\ x$   
 <proof>

**lemma** *coll-scaleR*:  $coll\ 0\ x\ y \implies coll\ 0\ (r *_{\mathbb{R}} x)\ y$   
 <proof>

**lemma** *coll-sum-list*:  $(\bigwedge y. y \in set\ ys \implies coll\ 0\ x\ y) \implies coll\ 0\ x\ (sum-list\ ys)$   
 <proof>

**lemma** *scaleR-left-normalize*:  
**fixes**  $a :: real$  **and**  $b\ c :: 'a :: real-vector$   
**shows**  $a *_{\mathbb{R}} b = c \longleftrightarrow (if\ a = 0\ then\ c = 0\ else\ b = c /_{\mathbb{R}} a)$   
 <proof>

**lemma** *coll-scale-pair*:  $coll\ 0\ (a, b)\ (c, d) \implies (a, b) \neq 0 \implies (\exists x. (c, d) = x *_{\mathbb{R}} (a, b))$   
 <proof>

**lemma** *coll-scale*:  $coll\ 0\ r\ q \implies r \neq 0 \implies (\exists x. q = x *_{\mathbb{R}} r)$   
 <proof>

**lemma** *coll-add-trans*:  
**assumes**  $coll\ 0\ x\ (y + z)$   
**assumes**  $coll\ 0\ y\ z$   
**assumes**  $x \neq 0$   
**assumes**  $y \neq 0$   
**assumes**  $z \neq 0$   
**assumes**  $y + z \neq 0$   
**shows**  $coll\ 0\ x\ z$   
 <proof>

**lemma** *coll-commute*:  $coll\ 0\ a\ b \longleftrightarrow coll\ 0\ b\ a$   
 <proof>

**lemma** *coll-add-cancel*:  $coll\ 0\ a\ (a + b) \implies coll\ 0\ a\ b$   
 <proof>

**lemma** *coll-trans*:  
 $coll\ 0\ a\ b \implies coll\ 0\ a\ c \implies a \neq 0 \implies coll\ 0\ b\ c$

*<proof>*

**lemma** *sum-list-posI*:

**fixes** *xs::'a::ordered-comm-monoid-add list*

**shows**  $(\bigwedge x. x \in \text{set } xs \implies x > 0) \implies xs \neq [] \implies \text{sum-list } xs > 0$

*<proof>*

**lemma** *nonzero-fstI*[*intro, simp*]:  $\text{fst } x \neq 0 \implies x \neq 0$

**and** *nonzero-sndI*[*intro, simp*]:  $\text{snd } x \neq 0 \implies x \neq 0$

*<proof>*

**lemma** *coll-sum-list-trans*:

$xs \neq [] \implies \text{coll } 0 \ a \ (\text{sum-list } xs) \implies (\bigwedge x. x \in \text{set } xs \implies \text{coll } 0 \ x \ y) \implies$

$(\bigwedge x. x \in \text{set } xs \implies \text{coll } 0 \ x \ (\text{sum-list } xs)) \implies$

$(\bigwedge x. x \in \text{set } xs \implies \text{snd } x > 0) \implies a \neq 0 \implies \text{coll } 0 \ a \ y$

*<proof>*

**lemma** *sum-list-coll-ex-scale*:

**assumes** *coll*:  $\bigwedge x. x \in \text{set } xs \implies \text{coll } 0 \ z \ x$

**assumes** *nz*:  $z \neq 0$

**shows**  $\exists r. \text{sum-list } xs = r *_{\mathbb{R}} z$

*<proof>*

**lemma** *sum-list-filter-coll-ex-scale*:  $z \neq 0 \implies \exists r. \text{sum-list } (\text{filter } (\text{coll } 0 \ z) \ zs) = r *_{\mathbb{R}} z$

*<proof>*

**end**

**theory** *Polygon*

**imports** *Counterclockwise-2D-Strict*

**begin**

## 8.4 Polygonal chains

**definition** *polychain*  $xs = (\forall i. \text{Suc } i < \text{length } xs \longrightarrow \text{snd } (xs ! i) = \text{fst } (xs ! \text{Suc } i))$

**lemma** *polychainI*:

**assumes**  $\bigwedge i. \text{Suc } i < \text{length } xs \implies \text{snd } (xs ! i) = \text{fst } (xs ! \text{Suc } i)$

**shows** *polychain*  $xs$

*<proof>*

**lemma** *polychain-Nil*[*simp*]: *polychain*  $[] = \text{True}$

**and** *polychain-singleton*[*simp*]: *polychain*  $[x] = \text{True}$

*<proof>*

**lemma** *polychain-Cons*:

*polychain*  $(y \# ys) = (\text{if } ys = [] \text{ then } \text{True} \text{ else } \text{snd } y = \text{fst } (ys ! 0) \wedge \text{polychain } ys)$

*<proof>*

**lemma** *polychain-appendI*:

$polychain\ xs \implies polychain\ ys \implies (xs \neq [] \implies ys \neq [] \implies snd\ (last\ xs) = fst\ (hd\ ys)) \implies$   
 $polychain\ (xs\ @\ ys)$   
*<proof>*

**fun** *pairself* **where** *pairself*  $f\ (x, y) = (f\ x, f\ y)$

**lemma** *pairself-apply*: *pairself*  $f\ x = (f\ (fst\ x), f\ (snd\ x))$

*<proof>*

**lemma** *polychain-map-pairself*:  $polychain\ xs \implies polychain\ (map\ (pairself\ f)\ xs)$

*<proof>*

**definition** *convex-polychain*  $xs \longleftrightarrow$

$(polychain\ xs \wedge$   
 $(\forall i. Suc\ i < length\ xs \longrightarrow det3\ (fst\ (xs\ !\ i))\ (snd\ (xs\ !\ i))\ (snd\ (xs\ !\ Suc\ i)) > 0))$

**lemma** *convex-polychain-Cons2[simp]*:

$convex-polychain\ (x\ \#\ y\ \#\ zs) \longleftrightarrow$   
 $snd\ x = fst\ y \wedge det3\ (fst\ x)\ (fst\ y)\ (snd\ y) > 0 \wedge convex-polychain\ (y\ \#\ zs)$   
*<proof>*

**lemma** *convex-polychain-ConsD*:

**assumes**  $convex-polychain\ (x\ \#\ xs)$

**shows**  $convex-polychain\ xs$

*<proof>*

**definition**

$convex-polygon\ xs \longleftrightarrow (convex-polychain\ xs \wedge (xs \neq [] \longrightarrow fst\ (hd\ xs) = snd\ (last\ xs)))$

**lemma** *convex-polychain-Nil[simp]*:  $convex-polychain\ [] = True$

**and** *convex-polychain-Cons[simp]*:  $convex-polychain\ [x] = True$

*<proof>*

**lemma** *convex-polygon-Cons2[simp]*:

$convex-polygon\ (x\ \#\ y\ \#\ zs) \longleftrightarrow fst\ x = snd\ (last\ (y\ \#\ zs)) \wedge convex-polychain\ (x\ \#\ y\ \#\ zs)$   
*<proof>*

**lemma** *polychain-append-connected*:

$polychain\ (xs\ @\ ys) \implies xs \neq [] \implies ys \neq [] \implies fst\ (hd\ ys) = snd\ (last\ xs)$

*<proof>*

**lemma** *convex-polychain-appendI*:

**assumes** *cxs*: *convex-polychain xs*  
**assumes** *cys*: *convex-polychain ys*  
**assumes** *pxy*: *polychain (xs @ ys)*  
**assumes**  $xs \neq [] \implies ys \neq [] \implies \text{det3 } (\text{fst } (\text{last } xs)) (\text{snd } (\text{last } xs)) (\text{snd } (\text{hd } ys))$   
 $> 0$   
**shows** *convex-polychain (xs @ ys)*  
*<proof>*

**lemma** *convex-polychainI*:  
**assumes** *polychain xs*  
**assumes**  $\bigwedge i. \text{Suc } i < \text{length } xs \implies \text{det3 } (\text{fst } (xs ! i)) (\text{snd } (xs ! i)) (\text{snd } (xs ! \text{Suc } i)) > 0$   
**shows** *convex-polychain xs*  
*<proof>*

**lemma** *convex-polygon-skip*:  
**assumes** *convex-polygon (x # y # z # w # ws)*  
**assumes**  $\text{ccw'.sortedP } (\text{fst } x) (\text{map } \text{snd } (\text{butlast } (x \# y \# z \# w \# ws)))$   
**shows** *convex-polygon ((fst x, snd y) # z # w # ws)*  
*<proof>*

**primrec** *polychain-of::'a::ab-group-add  $\Rightarrow$  'a list  $\Rightarrow$  ('a\*'a) list where*  
*polychain-of xc [] = []*  
*| polychain-of xc (xm#xs) = (xc, xc + xm)#polychain-of (xc + xm) xs*

**lemma** *in-set-polychain-ofD*:  $ab \in \text{set } (\text{polychain-of } x \text{ } xs) \implies (\text{snd } ab - \text{fst } ab) \in \text{set } xs$   
*<proof>*

**lemma** *fst-polychain-of-nth-0[simp]*:  $xs \neq [] \implies \text{fst } ((\text{polychain-of } p \text{ } xs) ! 0) = p$   
*<proof>*

**lemma** *fst-hd-polychain-of*:  $xs \neq [] \implies \text{fst } (\text{hd } (\text{polychain-of } x \text{ } xs)) = x$   
*<proof>*

**lemma** *length-polychain-of-eq[simp]*:  
**shows**  $\text{length } (\text{polychain-of } p \text{ } qs) = \text{length } qs$   
*<proof>*

**lemma**  
*polychain-of-subsequent-eq*:  
**assumes**  $\text{Suc } i < \text{length } qs$   
**shows**  $\text{snd } (\text{polychain-of } p \text{ } qs ! i) = \text{fst } (\text{polychain-of } p \text{ } qs ! \text{Suc } i)$   
*<proof>*

**lemma** *polychain-of-eq-empty-iff[simp]*:  $\text{polychain-of } p \text{ } xs = [] \iff xs = []$   
*<proof>*

**lemma** *in-set-polychain-of-imp-sum-list*:  
**assumes**  $z \in \text{set } (\text{polychain-of } Pc \ Ps)$   
**obtains**  $d$  **where**  $z = (Pc + \text{sum-list } (\text{take } d \ Ps), Pc + \text{sum-list } (\text{take } (\text{Suc } d) \ Ps))$   
 ⟨*proof*⟩

**lemma** *last-polychain-of*:  $\text{length } xs > 0 \implies \text{snd } (\text{last } (\text{polychain-of } p \ xs)) = p + \text{sum-list } xs$   
 ⟨*proof*⟩

**lemma** *polychain-of-singleton-iff*:  $\text{polychain-of } p \ xs = [a] \longleftrightarrow \text{fst } a = p \wedge xs = [(\text{snd } a - p)]$   
 ⟨*proof*⟩

**lemma** *polychain-of-add*:  $\text{polychain-of } (x + y) \ xs = \text{map } (((+) (y, y))) (\text{polychain-of } x \ xs)$   
 ⟨*proof*⟩

## 8.5 Dirvec: Inverse of Polychain

**primrec** *dirvec* **where**  $\text{dirvec } (x, y) = (y - x)$

**lemma** *dirvec-minus*:  $\text{dirvec } x = \text{snd } x - \text{fst } x$   
 ⟨*proof*⟩

**lemma** *dirvec-nth-polychain-of*:  $n < \text{length } xs \implies \text{dirvec } ((\text{polychain-of } p \ xs) ! n) = (xs ! n)$   
 ⟨*proof*⟩

**lemma** *dirvec-hd-polychain-of*:  $xs \neq [] \implies \text{dirvec } (\text{hd } (\text{polychain-of } p \ xs)) = (\text{hd } xs)$   
 ⟨*proof*⟩

**lemma** *dirvec-last-polychain-of*:  $xs \neq [] \implies \text{dirvec } (\text{last } (\text{polychain-of } p \ xs)) = (\text{last } xs)$   
 ⟨*proof*⟩

**lemma** *map-dirvec-polychain-of[simp]*:  $\text{map } \text{dirvec } (\text{polychain-of } x \ xs) = xs$   
 ⟨*proof*⟩

## 8.6 Polychain of Sorted (*polychain-of, ccw'.sortedP*)

**lemma** *ccw'-sortedP-translateD*:  
 $\text{linorder-list0.sortedP } (ccw' \ x0) (\text{map } ((+) \ x \circ \ g) \ xs) \implies$   
 $\text{linorder-list0.sortedP } (ccw' (x0 - x)) (\text{map } g \ xs)$   
 ⟨*proof*⟩

**lemma** *ccw'-sortedP-translateI*:  
 $\text{linorder-list0.sortedP } (ccw' (x0 - x)) (\text{map } g \ xs) \implies$   
 $\text{linorder-list0.sortedP } (ccw' \ x0) (\text{map } ((+) \ x \circ \ g) \ xs)$

*<proof>*

**lemma** *ccw'-sortedP-translate-comp[simp]*:  
*linorder-list0.sortedP (ccw' x0) (map ((+) x o g) xs)  $\longleftrightarrow$*   
*linorder-list0.sortedP (ccw' (x0 - x)) (map g xs)*  
*<proof>*

**lemma** *snd-plus-commute*: *snd o (+) (x0, x0) = (+) x0 o snd*  
*<proof>*

**lemma** *ccw'-sortedP-renormalize*:  
*ccw'.sortedP a (map snd (polychain-of (x0 + x) xs))  $\longleftrightarrow$*   
*ccw'.sortedP (a - x0) (map snd (polychain-of x xs))*  
*<proof>*

**lemma** *ccw'-sortedP-polychain-of01*:  
**shows** *ccw'.sortedP 0 [u]  $\implies$  ccw'.sortedP x0 (map snd (polychain-of x0 [u]))*  
**and** *ccw'.sortedP 0 []  $\implies$  ccw'.sortedP x0 (map snd (polychain-of x0 []))*  
*<proof>*

**lemma** *ccw'-sortedP-polychain-of2*:  
**assumes** *ccw'.sortedP 0 [u, v]*  
**shows** *ccw'.sortedP x0 (map snd (polychain-of x0 [u, v]))*  
*<proof>*

**lemma** *ccw'-sortedP-polychain-of3*:  
**assumes** *ccw'.sortedP 0 (u#v#w#xs)*  
**shows** *ccw'.sortedP x0 (map snd (polychain-of x0 (u#v#w#xs)))*  
*<proof>*

**lemma** *ccw'-sortedP-polychain-of-snd*:  
**assumes** *ccw'.sortedP 0 xs*  
**shows** *ccw'.sortedP x0 (map snd (polychain-of x0 xs))*  
*<proof>*

**lemma** *ccw'-sortedP-implies-distinct*:  
**assumes** *ccw'.sortedP x qs*  
**shows** *distinct qs*  
*<proof>*

**lemma** *ccw'-sortedP-implies-nonaligned*:  
**assumes** *ccw'.sortedP x qs*  
**assumes** *y  $\in$  set qs z  $\in$  set qs y  $\neq$  z*  
**shows**  $\neg$  *coll x y z*  
*<proof>*

**lemma** *list-all-mp*: *list-all P xs  $\implies$  ( $\bigwedge x. x \in$  set xs  $\implies$  P x  $\implies$  Q x)  $\implies$  list-all Q xs*  
*<proof>*

**lemma**

*ccw'-scale-origin:*

**assumes**  $e \in UNIV \rightarrow \{0 < .. < 1\}$

**assumes**  $x \in \text{set } (\text{polychain-of } Pc \ (P \# \text{QRRs}))$

**assumes**  $ccw'.\text{sortedP } 0 \ (P \# \text{QRRs})$

**assumes**  $ccw' \ (fst \ x) \ (snd \ x) \ (P + (Pc + (\sum_{P \in \text{set } \text{QRRs}} e \ P \ *_R \ P)))$

**shows**  $ccw' \ (fst \ x) \ (snd \ x) \ (e \ P \ *_R \ P + (Pc + (\sum_{P \in \text{set } \text{QRRs}} e \ P \ *_R \ P)))$

*<proof>*

**lemma** *polychain-of-ccw-convex:*

**assumes**  $e \in UNIV \rightarrow \{0 < .. < 1\}$

**assumes** *sorted: linorder-list0.sortedP*  $(ccw' \ 0) \ (P \# \text{Q} \# \text{Ps})$

**shows** *list-all*

$(\lambda(x_i, x_j). \text{ccw}' \ x_i \ x_j \ (Pc + (\sum_{P \in \text{set } (P \# \text{Q} \# \text{Ps})} e \ P \ *_R \ P)))$

$(\text{polychain-of } Pc \ (P \# \text{Q} \# \text{Ps}))$

*<proof>*

**lemma** *polychain-of-ccw:*

**assumes**  $e \in UNIV \rightarrow \{0 < .. < 1\}$

**assumes** *sorted: ccw'.sortedP*  $0 \ qs$

**assumes** *qs: length qs*  $\neq 1$

**shows** *list-all*  $(\lambda(x_i, x_j). \text{ccw}' \ x_i \ x_j \ (Pc + (\sum_{P \in \text{set } qs} e \ P \ *_R \ P))) \ (\text{polychain-of } Pc \ qs)$

*<proof>*

**lemma** *in-polychain-of-ccw:*

**assumes**  $e \in UNIV \rightarrow \{0 < .. < 1\}$

**assumes**  $ccw'.\text{sortedP } 0 \ qs$

**assumes** *length qs*  $\neq 1$

**assumes** *seg*  $\in \text{set } (\text{polychain-of } Pc \ qs)$

**shows**  $ccw' \ (fst \ \text{seg}) \ (snd \ \text{seg}) \ (Pc + (\sum_{P \in \text{set } qs} e \ P \ *_R \ P))$

*<proof>*

**lemma** *distinct-butlast-ne-last: distinct xs*  $\implies x \in \text{set } (\text{butlast } xs) \implies x \neq \text{last } xs$

*<proof>*

**lemma**

*ccw'-sortedP-convex-rotate-aux:*

**assumes**  $ccw'.\text{sortedP } 0 \ (zs) \ ccw'.\text{sortedP } x \ (\text{map } \text{snd} \ (\text{polychain-of } x \ (zs)))$

**shows**  $ccw'.\text{sortedP} \ (\text{snd} \ (\text{last} \ (\text{polychain-of } x \ (zs)))) \ (\text{map } \text{snd} \ (\text{butlast} \ (\text{polychain-of } x \ (zs))))$

*<proof>*

**lemma** *ccw'-polychain-of-sorted-center-last:*

**assumes** *set-butlast: (c, d) ∈ set (butlast (polychain-of x0 xs))*

**assumes** *sorted: ccw'.sortedP*  $0 \ xs$

**assumes** *ne: xs*  $\neq []$

**shows**  $ccw' \ x0 \ d \ (\text{snd} \ (\text{last} \ (\text{polychain-of } x0 \ xs)))$

*<proof>*

**end**

## 9 CCW for Arbitrary Points in the Plane

**theory** *Counterclockwise-2D-Arbitrary*  
**imports** *Counterclockwise-2D-Strict*  
**begin**

### 9.1 Interpretation of Knuth's axioms in the plane

**definition** *lex::point  $\Rightarrow$  point  $\Rightarrow$  bool* **where**

$$\text{lex } p \ q \longleftrightarrow (\text{fst } p < \text{fst } q \vee \text{fst } p = \text{fst } q \wedge \text{snd } p < \text{snd } q \vee p = q)$$

**definition** *psi::point  $\Rightarrow$  point  $\Rightarrow$  point  $\Rightarrow$  bool* **where**

$$\text{psi } p \ q \ r \longleftrightarrow (\text{lex } p \ q \wedge \text{lex } q \ r)$$

**definition** *ccw::point  $\Rightarrow$  point  $\Rightarrow$  point  $\Rightarrow$  bool* **where**

$$\text{ccw } p \ q \ r \longleftrightarrow \text{ccw}' \ p \ q \ r \vee (\text{det3 } p \ q \ r = 0 \wedge (\text{psi } p \ q \ r \vee \text{psi } q \ r \ p \vee \text{psi } r \ p \ q))$$

**interpretation** *ccw: linorder-list0 ccw x* **for** *x* *<proof>*

**lemma** *ccw'-imp-ccw: ccw' a b c  $\Longrightarrow$  ccw a b c*

*<proof>*

**lemma** *ccw-ncoll-imp-ccw: ccw a b c  $\Longrightarrow$   $\neg$ coll a b c  $\Longrightarrow$  ccw' a b c*

*<proof>*

**lemma** *ccw-translate: ccw p (p + q) (p + r) = ccw 0 q r*

*<proof>*

**lemma** *ccw-translate-origin: NO-MATCH 0 p  $\Longrightarrow$  ccw p q r = ccw 0 (q - p) (r - p)*

*<proof>*

**lemma** *psi-scale:*

*psi (r \*<sub>R</sub> a) (r \*<sub>R</sub> b) 0 = (if r > 0 then psi a b 0 else if r < 0 then psi 0 b a else True)*

*psi (r \*<sub>R</sub> a) 0 (r \*<sub>R</sub> b) = (if r > 0 then psi a 0 b else if r < 0 then psi b 0 a else True)*

*psi 0 (r \*<sub>R</sub> a) (r \*<sub>R</sub> b) = (if r > 0 then psi 0 a b else if r < 0 then psi b a 0 else True)*

*<proof>*

**lemma** *ccw-scale23: ccw 0 a b  $\Longrightarrow$  r > 0  $\Longrightarrow$  ccw 0 (r \*<sub>R</sub> a) (r \*<sub>R</sub> b)*

*<proof>*

**lemma** *psi-notI: distinct3 p q r  $\Longrightarrow$  psi p q r  $\Longrightarrow$   $\neg$  psi q p r*

*<proof>*

**lemma** *not-lex-eq*:  $\neg \text{lex } a \ b \longleftrightarrow \text{lex } b \ a \wedge b \neq a$   
*<proof>*

**lemma** *lex-trans*:  $\text{lex } a \ b \implies \text{lex } b \ c \implies \text{lex } a \ c$   
*<proof>*

**lemma** *lex-sym-eqI*:  $\text{lex } a \ b \implies \text{lex } b \ a \implies a = b$   
**and** *lex-sym-eq-iff*:  $\text{lex } a \ b \implies \text{lex } b \ a \longleftrightarrow a = b$   
*<proof>*

**lemma** *lex-refl[simp]*:  $\text{lex } p \ p$   
*<proof>*

**lemma** *psi-disjuncts*:  
 $\text{distinct3 } p \ q \ r \implies \text{psi } p \ q \ r \vee \text{psi } p \ r \ q \vee \text{psi } q \ r \ p \vee \text{psi } q \ p \ r \vee \text{psi } r \ p \ q \vee \text{psi } r \ q \ p$   
*<proof>*

**lemma** *nlex-ccw-left*:  $\text{lex } x \ 0 \implies \text{ccw } 0 \ (0, 1) \ x$   
*<proof>*

**interpretation** *ccw-system123* *ccw*  
*<proof>*

**lemma** *lex-scaleR-nonneg*:  $\text{lex } a \ b \implies r \geq 0 \implies \text{lex } a \ (a + r *_{\mathbb{R}} (b - a))$   
*<proof>*

**lemma** *lex-scale1-zero*:  
 $\text{lex } (v *_{\mathbb{R}} u) \ 0 = (\text{if } v > 0 \text{ then } \text{lex } u \ 0 \text{ else if } v < 0 \text{ then } \text{lex } 0 \ u \text{ else True})$   
**and** *lex-scale2-zero*:  
 $\text{lex } 0 \ (v *_{\mathbb{R}} u) = (\text{if } v > 0 \text{ then } \text{lex } 0 \ u \text{ else if } v < 0 \text{ then } \text{lex } u \ 0 \text{ else True})$   
*<proof>*

**lemma** *nlex-add*:  
**assumes**  $\text{lex } a \ 0 \ \text{lex } b \ 0$   
**shows**  $\text{lex } (a + b) \ 0$   
*<proof>*

**lemma** *nlex-sum*:  
**assumes** *finite*  $X$   
**assumes**  $\bigwedge x. x \in X \implies \text{lex } (f \ x) \ 0$   
**shows**  $\text{lex } (\text{sum } f \ X) \ 0$   
*<proof>*

**lemma** *abs-add-nlex*:  
**assumes** *coll*  $0 \ a \ b$   
**assumes**  $\text{lex } a \ 0$

**assumes**  $lex\ b\ 0$   
**shows**  $abs\ (a + b) = abs\ a + abs\ b$   
 $\langle proof \rangle$

**lemma**  $lex\text{-}sum\text{-}list$ :  $(\bigwedge x. x \in set\ xs \implies lex\ x\ 0) \implies lex\ (sum\text{-}list\ xs)\ 0$   
 $\langle proof \rangle$

**lemma**  
*abs-sum-list-coll*:  
**assumes**  $coll$ :  $list\text{-}all\ (coll\ 0\ x)\ xs$   
**assumes**  $x \neq 0$   
**assumes**  $up$ :  $list\text{-}all\ (\lambda x. lex\ x\ 0)\ xs$   
**shows**  $abs\ (sum\text{-}list\ xs) = sum\text{-}list\ (map\ abs\ xs)$   
 $\langle proof \rangle$

**lemma**  $lex\text{-}diff1$ :  $lex\ (a - b)\ c = lex\ a\ (c + b)$   
**and**  $lex\text{-}diff2$ :  $lex\ c\ (a - b) = lex\ (c + b)\ a$   
 $\langle proof \rangle$

**lemma**  $sum\text{-}list\text{-}eq\ 0\text{-}iff\text{-}nonpos$ :  
**fixes**  $xs::'a::ordered\text{-}ab\text{-}group\text{-}add\ list$   
**shows**  $list\text{-}all\ (\lambda x. x \leq 0)\ xs \implies sum\text{-}list\ xs = 0 \longleftrightarrow (\forall n \in set\ xs. n = 0)$   
 $\langle proof \rangle$

**lemma**  $sum\text{-}list\text{-}nlex\text{-}eq\text{-}zeroI$ :  
**assumes**  $nlex$ :  $list\text{-}all\ (\lambda x. lex\ x\ 0)\ xs$   
**assumes**  $sum\text{-}list\ xs = 0$   
**assumes**  $x \in set\ xs$   
**shows**  $x = 0$   
 $\langle proof \rangle$

**lemma**  $sum\text{-}list\text{-}eq0I$ :  $(\forall x \in set\ xs. x = 0) \implies sum\text{-}list\ xs = 0$   
 $\langle proof \rangle$

**lemma**  $sum\text{-}list\text{-}nlex\text{-}eq\text{-}zero\text{-}iff$ :  
**assumes**  $nlex$ :  $list\text{-}all\ (\lambda x. lex\ x\ 0)\ xs$   
**shows**  $sum\text{-}list\ xs = 0 \longleftrightarrow list\text{-}all\ ((=)\ 0)\ xs$   
 $\langle proof \rangle$

**lemma**  
**assumes**  $lex\ p\ q\ lex\ q\ r\ 0 \leq a\ 0 \leq b\ 0 \leq c\ a + b + c = 1$   
**assumes**  $comb\text{-}def$ :  $comb = a *_{\mathbb{R}} p + b *_{\mathbb{R}} q + c *_{\mathbb{R}} r$   
**shows**  $lex\text{-}convex3$ :  $lex\ p\ comb\ lex\ comb\ r$   
 $\langle proof \rangle$

**lemma**  $lex\text{-}convex\text{-}self2$ :  
**assumes**  $lex\ p\ q\ 0 \leq a\ a \leq 1$   
**defines**  $r \equiv a *_{\mathbb{R}} p + (1 - a) *_{\mathbb{R}} q$   
**shows**  $lex\ p\ r$  (**is** ?th1)

**and**  $\text{lex } r \ q$  (**is** ?th2)  
 ⟨proof⟩

**lemma**  $\text{lex-uminus0}[simp]$ :  $\text{lex } (-a) \ 0 = \text{lex } 0 \ a$   
 ⟨proof⟩

**lemma**  
 $\text{lex-fst-zero-imp}$ :  
 $\text{fst } x = 0 \implies \text{lex } x \ 0 \implies \text{lex } y \ 0 \implies \neg \text{coll } 0 \ x \ y \implies \text{ccw}' \ 0 \ y \ x$   
 ⟨proof⟩

**lemma**  $\text{lex-ccw-left}$ :  $\text{lex } x \ y \implies r > 0 \implies \text{ccw } y \ (y + (0, r)) \ x$   
 ⟨proof⟩

**lemma**  $\text{lex-translate-origin}$ :  $\text{NO-MATCH } 0 \ a \implies \text{lex } a \ b = \text{lex } 0 \ (b - a)$   
 ⟨proof⟩

## 9.2 Order prover setup

**definition**  $\text{lexs } p \ q \longleftrightarrow (\text{lex } p \ q \wedge p \neq q)$

**lemma**  $\text{lexs-irrefl}$ :  $\neg \text{lexs } p \ p$   
**and**  $\text{lexs-imp-lex}$ :  $\text{lexs } x \ y \implies \text{lex } x \ y$   
**and**  $\text{not-lexs}$ :  $(\neg \text{lexs } x \ y) = (\text{lex } y \ x)$   
**and**  $\text{not-lex}$ :  $(\neg \text{lex } x \ y) = (\text{lexs } y \ x)$   
**and**  $\text{eq-lex-refl}$ :  $x = y \implies \text{lex } x \ y$   
 ⟨proof⟩

**lemma**  $\text{lexs-trans}$ :  $\text{lexs } x \ y \implies \text{lexs } y \ z \implies \text{lexs } x \ z$   
**and**  $\text{lexs-lex-trans}$ :  $\text{lexs } x \ y \implies \text{lex } y \ z \implies \text{lexs } x \ z$   
**and**  $\text{lex-lexs-trans}$ :  $\text{lex } x \ y \implies \text{lexs } y \ z \implies \text{lexs } x \ z$   
**and**  $\text{lex-neq-trans}$ :  $\text{lex } a \ b \implies a \neq b \implies \text{lexs } a \ b$   
**and**  $\text{neq-lex-trans}$ :  $a \neq b \implies \text{lex } a \ b \implies \text{lexs } a \ b$   
**and**  $\text{lexs-imp-neq}$ :  $\text{lexs } a \ b \implies a \neq b$   
 ⟨proof⟩

⟨ML⟩

## 9.3 Contradictions

**lemma**  
**assumes**  $d$ :  $\text{distinct4 } s \ p \ q \ r$   
**shows**  $\text{contra1}$ :  $\neg(\text{lex } p \ q \wedge \text{lex } q \ r \wedge \text{lex } r \ s \wedge \text{indelta } s \ p \ q \ r)$  (**is** ?th1)  
**and**  $\text{contra2}$ :  $\neg(\text{lex } s \ p \wedge \text{lex } p \ q \wedge \text{lex } q \ r \wedge \text{indelta } s \ p \ q \ r)$  (**is** ?th2)  
**and**  $\text{contra3}$ :  $\neg(\text{lex } p \ r \wedge \text{lex } p \ s \wedge \text{lex } q \ r \wedge \text{lex } q \ s \wedge \text{insquare } p \ r \ q \ s)$  (**is** ?th3)  
 ⟨proof⟩

**lemma**  $\text{ccw}'\text{-subst-psi-disj}$ :  
**assumes**  $\text{det3 } t \ r \ s = 0$   
**assumes**  $\text{psi } t \ r \ s \vee \text{psi } t \ s \ r \vee \text{psi } s \ r \ t$

**assumes**  $s \neq t$   
**assumes**  $ccw' t r p$   
**shows**  $ccw' t s p$   
 ⟨*proof*⟩

**lemma** *lex-contr*:  
**assumes**  $distinct_4 t s q r$   
**assumes**  $lex t s lex s r$   
**assumes**  $det_3 t s r = 0$   
**assumes**  $ccw' t s q$   
**assumes**  $ccw' t q r$   
**shows** *False*  
 ⟨*proof*⟩

**lemma** *contra\_4*:  
**assumes**  $distinct_4 s r q p$   
**assumes**  $lex: lex q p lex p r lex r s$   
**assumes**  $ccw: ccw r q s ccw r s p ccw r q p$   
**shows** *False*  
 ⟨*proof*⟩

**lemma** *not-coll-ordered-lexI*:  
**assumes**  $l \neq x_0$   
**and**  $lex x_1 r$   
**and**  $lex x_1 l$   
**and**  $lex r x_0$   
**and**  $lex l x_0$   
**and**  $ccw' x_0 l x_1$   
**and**  $ccw' x_0 x_1 r$   
**shows**  $det_3 x_0 l r \neq 0$   
 ⟨*proof*⟩

**interpretation** *ccw-system\_4 ccw*  
 ⟨*proof*⟩

**lemma** *lex-total*:  $lex t q \wedge t \neq q \vee lex q t \wedge t \neq q \vee t = q$   
 ⟨*proof*⟩

**lemma**  
*ccw-two-up-contr*:  
**assumes**  $c: ccw' t p q ccw' t q r$   
**assumes**  $ccws: ccw t s p ccw t s q ccw t s r ccw t p q ccw t q r ccw t r p$   
**assumes** *distinct*:  $distinct_5 t s p q r$   
**shows** *False*  
 ⟨*proof*⟩

**lemma**  
*ccw-transitive-contr*:  
**fixes**  $t s p q r$

**assumes** *ccws*:  $ccw\ t\ s\ p\ ccw\ t\ s\ q\ ccw\ t\ s\ r\ ccw\ t\ p\ q\ ccw\ t\ q\ r\ ccw\ t\ r\ p$   
**assumes** *distinct*:  $distinct5\ t\ s\ p\ q\ r$   
**shows** *False*  
 <proof>

**interpretation** *ccw*: *ccw-system ccw*  
 <proof>

**lemma** *ccw-scaleR1*:  
 $det3\ 0\ xr\ P \neq 0 \implies 0 < e \implies ccw\ 0\ xr\ P \implies ccw\ 0\ (e*_R xr)\ P$   
 <proof>

**lemma** *ccw-scaleR2*:  
 $det3\ 0\ xr\ P \neq 0 \implies 0 < e \implies ccw\ 0\ xr\ P \implies ccw\ 0\ xr\ (e*_R P)$   
 <proof>

**lemma** *ccw-translate3-aux*:  
**assumes**  $\neg coll\ 0\ a\ b$   
**assumes**  $x < 1$   
**assumes**  $ccw\ 0\ (a - x*_R a)\ (b - x*_R a)$   
**shows**  $ccw\ 0\ a\ b$   
 <proof>

**lemma** *ccw-translate3-minus*:  $det3\ 0\ a\ b \neq 0 \implies x < 1 \implies ccw\ 0\ a\ (b - x*_R a) \implies ccw\ 0\ a\ b$   
 <proof>

**lemma** *ccw-translate3*:  $det3\ 0\ a\ b \neq 0 \implies x < 1 \implies ccw\ 0\ a\ b \implies ccw\ 0\ a\ (x*_R a + b)$   
 <proof>

**lemma** *ccw-switch23*:  $det3\ 0\ P\ Q \neq 0 \implies (\neg ccw\ 0\ Q\ P \longleftrightarrow ccw\ 0\ P\ Q)$   
 <proof>

**lemma** *ccw0-upward*:  $fst\ a > 0 \implies snd\ a = 0 \implies snd\ b > snd\ a \implies ccw\ 0\ a\ b$   
 <proof>

**lemma** *ccw-uminus3[simp]*:  $det3\ a\ b\ c \neq 0 \implies ccw\ (-a)\ (-b)\ (-c) = ccw\ a\ b\ c$   
 <proof>

**lemma** *coll-minus-eq*:  $coll\ (x - a)\ (x - b)\ (x - c) = coll\ a\ b\ c$   
 <proof>

**lemma** *ccw-minus3*:  $\neg coll\ a\ b\ c \implies ccw\ (x - a)\ (x - b)\ (x - c) \longleftrightarrow ccw\ a\ b\ c$   
 <proof>

**lemma** *ccw0-uminus[simp]*:  $\neg coll\ 0\ a\ b \implies ccw\ 0\ (-a)\ (-b) \longleftrightarrow ccw\ 0\ a\ b$   
 <proof>

**lemma** *lex-convex2*:

**assumes**  $lex\ p\ q\ lex\ p\ r\ 0 \leq u\ u \leq 1$

**shows**  $lex\ p\ (u *_{R} q + (1 - u) *_{R} r)$

*<proof>*

**lemma** *lex-convex2'*:

**assumes**  $lex\ q\ p\ lex\ r\ p\ 0 \leq u\ u \leq 1$

**shows**  $lex\ (u *_{R} q + (1 - u) *_{R} r)\ p$

*<proof>*

**lemma** *psi-convex1*:

**assumes**  $psi\ c\ a\ b$

**assumes**  $psi\ d\ a\ b$

**assumes**  $0 \leq u\ 0 \leq v\ u + v = 1$

**shows**  $psi\ (u *_{R} c + v *_{R} d)\ a\ b$

*<proof>*

**lemma** *psi-convex2*:

**assumes**  $psi\ a\ c\ b$

**assumes**  $psi\ a\ d\ b$

**assumes**  $0 \leq u\ 0 \leq v\ u + v = 1$

**shows**  $psi\ a\ (u *_{R} c + v *_{R} d)\ b$

*<proof>*

**lemma** *psi-convex3*:

**assumes**  $psi\ a\ b\ c$

**assumes**  $psi\ a\ b\ d$

**assumes**  $0 \leq u\ 0 \leq v\ u + v = 1$

**shows**  $psi\ a\ b\ (u *_{R} c + v *_{R} d)$

*<proof>*

**lemma** *coll-convex*:

**assumes**  $coll\ a\ b\ c\ coll\ a\ b\ d$

**assumes**  $0 \leq u\ 0 \leq v\ u + v = 1$

**shows**  $coll\ a\ b\ (u *_{R} c + v *_{R} d)$

*<proof>*

**lemma** (*in ccw-vector-space*) *convex3*:

**assumes**  $u \geq 0\ v \geq 0\ u + v = 1\ ccw\ a\ b\ d\ ccw\ a\ b\ c$

**shows**  $ccw\ a\ b\ (u *_{R} c + v *_{R} d)$

*<proof>*

**lemma** *ccw-self[simp]*:  $ccw\ a\ a\ b\ ccw\ b\ a\ a$

*<proof>*

**lemma** *ccw-seft'[simp]*:  $ccw\ a\ b\ a$

*<proof>*

**lemma** *ccw-convex'*:

**assumes**  $uv: u \geq 0 \ v \geq 0 \ u + v = 1$   
**assumes**  $ccw \ a \ b \ c$  **and**  $1: coll \ a \ b \ c$   
**assumes**  $ccw \ a \ b \ d$  **and**  $2: \neg \ coll \ a \ b \ d$   
**shows**  $ccw \ a \ b \ (u \ *_R \ c + v \ *_R \ d)$   
 $\langle proof \rangle$

**lemma** *ccw-convex*:  
**assumes**  $uv: u \geq 0 \ v \geq 0 \ u + v = 1$   
**assumes**  $ccw \ a \ b \ c$   
**assumes**  $ccw \ a \ b \ d$   
**assumes**  $lex: coll \ a \ b \ c \implies coll \ a \ b \ d \implies lex \ b \ a$   
**shows**  $ccw \ a \ b \ (u \ *_R \ c + v \ *_R \ d)$   
 $\langle proof \rangle$

**interpretation** *ccw*: *ccw-convex ccw S*  $\lambda a \ b. lex \ b \ a$  **for** *S*  
 $\langle proof \rangle$

**lemma** *ccw-sorted-scaleR*:  $ccw.sortedP \ 0 \ xs \implies r > 0 \implies ccw.sortedP \ 0 \ (map \ ((*_R) \ r) \ xs)$   
 $\langle proof \rangle$

**lemma** *ccw-sorted-implies-ccw'-sortedP*:  
**assumes**  $nonaligned: \bigwedge y \ z. y \in set \ Ps \implies z \in set \ Ps \implies y \neq z \implies \neg \ coll \ 0 \ y \ z$   
**assumes**  $sorted: linorder-list0.sortedP \ (ccw \ 0) \ Ps$   
**assumes**  $distinct \ Ps$   
**shows**  $linorder-list0.sortedP \ (ccw' \ 0) \ Ps$   
 $\langle proof \rangle$

end

## 10 Intersection

**theory** *Intersection*  
**imports**  
*HOL-Library.Monad-Syntax*  
*Polygon*  
*Counterclockwise-2D-Arbitrary*  
*Affine-Form*  
**begin**

### 10.1 Polygons and *ccw*, *Counterclockwise-2D-Arbitrary.lex*, *psi*, *coll*

**lemma** *polychain-of-ccw-conjunction*:  
**assumes**  $sorted: ccw'.sortedP \ 0 \ Ps$   
**assumes**  $z: z \in set \ (polychain-of \ Pc \ Ps)$

**shows**  $list\text{-}all (\lambda(xi, xj). ccw\ xi\ xj\ (fst\ z) \wedge ccw\ xi\ xj\ (snd\ z))\ (polychain\text{-}of\ Pc\ Ps)$   
 ⟨proof⟩

**lemma** *lex-polychain-of-center*:

$d \in set\ (polychain\text{-}of\ x0\ xs) \implies list\text{-}all\ (\lambda x. lex\ x\ 0)\ xs \implies lex\ (snd\ d)\ x0$   
 ⟨proof⟩

**lemma** *lex-polychain-of-last*:

$(c, d) \in set\ (polychain\text{-}of\ x0\ xs) \implies list\text{-}all\ (\lambda x. lex\ x\ 0)\ xs \implies$   
 $lex\ (snd\ (last\ (polychain\text{-}of\ x0\ xs)))\ d$   
 ⟨proof⟩

**lemma** *distinct-fst-polychain-of*:

**assumes**  $list\text{-}all\ (\lambda x. x \neq 0)\ xs$   
**assumes**  $list\text{-}all\ (\lambda x. lex\ x\ 0)\ xs$   
**shows**  $distinct\ (map\ fst\ (polychain\text{-}of\ x0\ xs))$   
 ⟨proof⟩

**lemma** *distinct-snd-polychain-of*:

**assumes**  $list\text{-}all\ (\lambda x. x \neq 0)\ xs$   
**assumes**  $list\text{-}all\ (\lambda x. lex\ x\ 0)\ xs$   
**shows**  $distinct\ (map\ snd\ (polychain\text{-}of\ x0\ xs))$   
 ⟨proof⟩

## 10.2 Orient all entries

**lift-definition** *nlex-pdevs::point pdevs  $\Rightarrow$  point pdevs*

**is**  $\lambda x\ n. if\ lex\ 0\ (x\ n)\ then\ -\ x\ n\ else\ x\ n$  ⟨proof⟩

**lemma** *pdevs-apply-nlex-pdevs[simp]*:  $pdevs\text{-}apply\ (nlex\text{-}pdevs\ x)\ n =$

$(if\ lex\ 0\ (pdevs\text{-}apply\ x\ n)\ then\ -\ pdevs\text{-}apply\ x\ n\ else\ pdevs\text{-}apply\ x\ n)$   
 ⟨proof⟩

**lemma** *nlex-pdevs-zero-pdevs[simp]*:  $nlex\text{-}pdevs\ zero\text{-}pdevs = zero\text{-}pdevs$

⟨proof⟩

**lemma** *pdevs-domain-nlex-pdevs[simp]*:  $pdevs\text{-}domain\ (nlex\text{-}pdevs\ x) = pdevs\text{-}domain$

$x$   
 ⟨proof⟩

**lemma** *degree-nlex-pdevs[simp]*:  $degree\ (nlex\text{-}pdevs\ x) = degree\ x$

⟨proof⟩

**lemma**

*pdevs-val-nlex-pdevs*:

**assumes**  $e \in UNIV \rightarrow I\ minus\ 'I = I$

**obtains**  $e'$  **where**  $e' \in UNIV \rightarrow I\ pdevs\text{-}val\ e\ x = pdevs\text{-}val\ e'\ (nlex\text{-}pdevs\ x)$

⟨proof⟩

**lemma**

*pdevs-val-nlex-pdevs2:*

**assumes**  $e \in UNIV \rightarrow I$  *uminus* '  $I = I$

**obtains**  $e'$  **where**  $e' \in UNIV \rightarrow I$   $pdevs\text{-}val\ e\ (nlex\text{-}pdevs\ x) = pdevs\text{-}val\ e'\ x$

*<proof>*

**lemma**

*pdevs-val-selsort-ccw:*

**assumes** *distinct xs*

**assumes**  $e \in UNIV \rightarrow I$

**obtains**  $e'$  **where**  $e' \in UNIV \rightarrow I$

$pdevs\text{-}val\ e\ (pdevs\text{-}of\text{-}list\ xs) = pdevs\text{-}val\ e'\ (pdevs\text{-}of\text{-}list\ (ccw.\textit{selsort}\ 0\ xs))$

*<proof>*

**lemma**

*pdevs-val-selsort-ccw2:*

**assumes** *distinct xs*

**assumes**  $e \in UNIV \rightarrow I$

**obtains**  $e'$  **where**  $e' \in UNIV \rightarrow I$

$pdevs\text{-}val\ e\ (pdevs\text{-}of\text{-}list\ (ccw.\textit{selsort}\ 0\ xs)) = pdevs\text{-}val\ e'\ (pdevs\text{-}of\text{-}list\ xs)$

*<proof>*

**lemma** *lex-nlex-pdevs:*  $lex\ (pdevs\text{-}apply\ (nlex\text{-}pdevs\ x)\ i)\ 0$

*<proof>*

### 10.3 Lowest Vertex

**definition** *lowest-vertex::'a::ordered-euclidean-space aform  $\Rightarrow$  'a where*

*lowest-vertex X = fst X - sum-list (map snd (list-of-pdevs (snd X)))*

**lemma** *snd-abs:*  $snd\ (abs\ x) = abs\ (snd\ x)$

*<proof>*

**lemma** *lowest-vertex:*

**fixes**  $X\ Y::(real*real)\ aform$

**assumes**  $e \in UNIV \rightarrow \{-1 .. 1\}$

**assumes**  $\bigwedge i. snd\ (pdevs\text{-}apply\ (snd\ X)\ i) \geq 0$

**assumes**  $\bigwedge i. abs\ (snd\ (pdevs\text{-}apply\ (snd\ Y)\ i)) = abs\ (snd\ (pdevs\text{-}apply\ (snd\ X)\ i))$

**assumes**  $degree\text{-}aform\ Y = degree\text{-}aform\ X$

**assumes**  $fst\ Y = fst\ X$

**shows**  $snd\ (lowest\text{-}vertex\ X) \leq snd\ (aform\text{-}val\ e\ Y)$

*<proof>*

**lemma** *sum-list-nonposI:*

**fixes**  $xs::'a::ordered\text{-}comm\text{-}monoid\text{-}add\ list$

**shows**  $list\text{-}all\ (\lambda x. x \leq 0)\ xs \implies sum\text{-}list\ xs \leq 0$

*<proof>*

**lemma** *center-le-lowest*:

*fst (fst X) ≤ fst (lowest-vertex (fst X, nlex-pdevs (snd X)))*  
 ⟨proof⟩

**lemma** *lowest-vertex-eq-center-iff*:

*lowest-vertex (x0, nlex-pdevs (snd X)) = x0 ↔ snd X = zero-pdevs*  
 ⟨proof⟩

## 10.4 Collinear Generators

**lemma** *scaleR-le-self-cancel*:

**fixes** *c::'a::ordered-real-vector*

**shows**  $a *_R c \leq c \longleftrightarrow (1 < a \wedge c \leq 0 \vee a < 1 \wedge 0 \leq c \vee a = 1)$

⟨proof⟩

**lemma** *pdevs-val-coll*:

**assumes** *coll*: *list-all (coll 0 x) xs*

**assumes** *nlex*: *list-all (λx. lex x 0) xs*

**assumes**  $x \neq 0$

**assumes**  $f \in UNIV \rightarrow \{-1 .. 1\}$

**obtains** *e* **where**  $e \in \{-1 .. 1\}$  *pdevs-val f (pdevs-of-list xs) = e \*\_R (sum-list xs)*

⟨proof⟩

**lemma** *scaleR-eq-self-cancel*:

**fixes** *x::'a::real-vector*

**shows**  $a *_R x = x \longleftrightarrow a = 1 \vee x = 0$

⟨proof⟩

**lemma** *abs-pdevs-val-less-tdev*:

**assumes**  $e \in UNIV \rightarrow \{-1 < .. < 1\}$  *degree x > 0*

**shows**  $|pdevs-val e x| < tdev x$

⟨proof⟩

**lemma** *pdevs-val-coll-strict*:

**assumes** *coll*: *list-all (coll 0 x) xs*

**assumes** *nlex*: *list-all (λx. lex x 0) xs*

**assumes**  $x \neq 0$

**assumes**  $f \in UNIV \rightarrow \{-1 < .. < 1\}$

**obtains** *e* **where**  $e \in \{-1 < .. < 1\}$  *pdevs-val f (pdevs-of-list xs) = e \*\_R (sum-list xs)*

⟨proof⟩

## 10.5 Independent Generators

**fun** *independent-pdevs::point list ⇒ point list*

**where**

*independent-pdevs [] = []*

| *independent-pdevs (x#xs) =*

(let  
 (cs, is) = List.partition (coll 0 x) xs;  
 s = x + sum-list cs  
 in (if s = 0 then [] else [s]) @ independent-pdevs is)

**lemma** *in-set-independent-pdevsE*:  
**assumes**  $y \in \text{set } (\text{independent-pdevs } xs)$   
**obtains**  $x$  **where**  $x \in \text{set } xs \text{ coll } 0 \ x \ y$   
 <proof>

**lemma** *in-set-independent-pdevs-nonzero*:  $x \in \text{set } (\text{independent-pdevs } xs) \implies x \neq 0$   
 <proof>

**lemma** *independent-pdevs-pairwise-non-coll*:  
**assumes**  $x \in \text{set } (\text{independent-pdevs } xs)$   
**assumes**  $y \in \text{set } (\text{independent-pdevs } xs)$   
**assumes**  $\bigwedge x. x \in \text{set } xs \implies x \neq 0$   
**assumes**  $x \neq y$   
**shows**  $\neg \text{coll } 0 \ x \ y$   
 <proof>

**lemma** *distinct-independent-pdevs[simp]*:  
**shows** *distinct* (*independent-pdevs*  $xs$ )  
 <proof>

**lemma** *in-set-independent-pdevs-invariant-nlex*:  
 $x \in \text{set } (\text{independent-pdevs } xs) \implies (\bigwedge x. x \in \text{set } xs \implies \text{lex } x \ 0) \implies$   
 $(\bigwedge x. x \in \text{set } xs \implies x \neq 0) \implies \text{Counterclockwise-2D-Arbitrary.lex } x \ 0$   
 <proof>

**lemma**  
*pdevs-val-independent-pdevs2*:  
**assumes**  $e \in \text{UNIV} \rightarrow I$   
**shows**  $\exists e'. e' \in \text{UNIV} \rightarrow I \wedge$   
 $\text{pdevs-val } e \ (\text{pdevs-of-list } (\text{independent-pdevs } xs)) = \text{pdevs-val } e' \ (\text{pdevs-of-list } xs)$   
 <proof>

**lemma** *list-all-filter*: *list-all*  $p$  (*filter*  $p$   $xs$ )  
 <proof>

**lemma** *pdevs-val-independent-pdevs*:  
**assumes** *list-all*  $(\lambda x. \text{lex } x \ 0) \ xs$   
**assumes** *list-all*  $(\lambda x. x \neq 0) \ xs$   
**assumes**  $e \in \text{UNIV} \rightarrow \{-1 .. 1\}$   
**shows**  $\exists e'. e' \in \text{UNIV} \rightarrow \{-1 .. 1\} \wedge \text{pdevs-val } e \ (\text{pdevs-of-list } xs) =$   
 $\text{pdevs-val } e' \ (\text{pdevs-of-list } (\text{independent-pdevs } xs))$   
 <proof>

**lemma**

*pdevs-val-independent-pdevs-strict:*

**assumes** *list-all* ( $\lambda x. \text{lex } x \ 0$ ) *xs*

**assumes** *list-all* ( $\lambda x. x \neq 0$ ) *xs*

**assumes**  $e \in UNIV \rightarrow \{-1 < .. < 1\}$

**shows**  $\exists e'. e' \in UNIV \rightarrow \{-1 < .. < 1\} \wedge \text{pdevs-val } e (\text{pdevs-of-list } xs) =$   
 $\text{pdevs-val } e' (\text{pdevs-of-list } (\text{independent-pdevs } xs))$

*<proof>*

**lemma** *sum-list-independent-pdevs:*  $\text{sum-list } (\text{independent-pdevs } xs) = \text{sum-list } xs$

*<proof>*

**lemma** *independent-pdevs-eq-Nil-iff:*

$\text{list-all } (\lambda x. \text{lex } x \ 0) \ xs \implies \text{list-all } (\lambda x. x \neq 0) \ xs \implies \text{independent-pdevs } xs = []$   
 $\longleftrightarrow xs = []$

*<proof>*

## 10.6 Independent Oriented Generators

**definition**  $\text{inl } p = \text{independent-pdevs } (\text{map } \text{snd } (\text{list-of-pdevs } (\text{nlex-pdevs } p)))$

**lemma** *distinct-inl[simp]:*  $\text{distinct } (\text{inl } (\text{snd } X))$

*<proof>*

**lemma** *in-set-inl-nonzero:*  $x \in \text{set } (\text{inl } xs) \implies x \neq 0$

*<proof>*

**lemma**

*inl-ncoll:*  $y \in \text{set } (\text{inl } (\text{snd } X)) \implies z \in \text{set } (\text{inl } (\text{snd } X)) \implies y \neq z \implies \neg \text{coll } 0$   
 $y \ z$

*<proof>*

**lemma** *in-set-inl-lex:*  $x \in \text{set } (\text{inl } xs) \implies \text{lex } x \ 0$

*<proof>*

**interpretation** *ccw0:* *linorder-list*  $\text{ccw } 0 \ \text{set } (\text{inl } (\text{snd } X))$

*<proof>*

**lemma** *sorted-inl:*  $\text{ccw.sortedP } 0 \ (\text{ccw.selsort } 0 \ (\text{inl } (\text{snd } X)))$

*<proof>*

**lemma** *sorted-scaled-inl:*  $\text{ccw.sortedP } 0 \ (\text{map } ((*_R) \ 2) \ (\text{ccw.selsort } 0 \ (\text{inl } (\text{snd } X))))$

*<proof>*

**lemma** *distinct-selsort-inl:*  $\text{distinct } (\text{ccw.selsort } 0 \ (\text{inl } (\text{snd } X)))$

*<proof>*

**lemma** *distinct-map-scaleRI*:

**fixes**  $xs::'a::\text{real-vector list}$

**shows**  $\text{distinct } xs \implies c \neq 0 \implies \text{distinct } (\text{map } ((*_R) c) xs)$

$\langle \text{proof} \rangle$

**lemma** *distinct-scaled-inl*:  $\text{distinct } (\text{map } ((*_R) 2) (\text{ccw.selsort } 0 (\text{inl } (\text{snd } X))))$

$\langle \text{proof} \rangle$

**lemma** *ccw'-sortedP-scaled-inl*:

$\text{ccw'.sortedP } 0 (\text{map } ((*_R) 2) (\text{ccw.selsort } 0 (\text{inl } (\text{snd } X))))$

$\langle \text{proof} \rangle$

**lemma** *pdevs-val-pdevs-of-list-inl2E*:

**assumes**  $e \in \text{UNIV} \rightarrow \{-1 .. 1\}$

**obtains**  $e'$  **where**  $\text{pdevs-val } e X = \text{pdevs-val } e' (\text{pdevs-of-list } (\text{inl } X))$   $e' \in \text{UNIV} \rightarrow \{-1 .. 1\}$

$\langle \text{proof} \rangle$

**lemma** *pdevs-val-pdevs-of-list-inlE*:

**assumes**  $e \in \text{UNIV} \rightarrow I \text{ uminus } 'I = I \ 0 \in I$

**obtains**  $e'$  **where**  $\text{pdevs-val } e (\text{pdevs-of-list } (\text{inl } X)) = \text{pdevs-val } e' X$   $e' \in \text{UNIV} \rightarrow I$

$\langle \text{proof} \rangle$

**lemma** *sum-list-nlex-eq-sum-list-inl*:

$\text{sum-list } (\text{map } \text{snd } (\text{list-of-pdevs } (\text{nlex-pdevs } X))) = \text{sum-list } (\text{inl } X)$

$\langle \text{proof} \rangle$

**lemma** *Affine-inl*:  $\text{Affine } (\text{fst } X, \text{pdevs-of-list } (\text{inl } (\text{snd } X))) = \text{Affine } X$

$\langle \text{proof} \rangle$

## 10.7 Half Segments

**definition** *half-segments-of-aform::point aform*  $\Rightarrow (\text{point*point}) \text{ list}$

**where** *half-segments-of-aform*  $X =$

(*let*

$x0 = \text{lowest-vertex } (\text{fst } X, \text{nlex-pdevs } (\text{snd } X))$

*in*

$\text{polychain-of } x0 (\text{map } ((*_R) 2) (\text{ccw.selsort } 0 (\text{inl } (\text{snd } X))))$ )

**lemma** *subsequent-half-segments*:

**fixes**  $X$

**assumes**  $\text{Suc } i < \text{length } (\text{half-segments-of-aform } X)$

**shows**  $\text{snd } (\text{half-segments-of-aform } X ! i) = \text{fst } (\text{half-segments-of-aform } X ! \text{Suc } i)$

$\langle \text{proof} \rangle$

**lemma** *polychain-half-segments-of-aform*:  $\text{polychain } (\text{half-segments-of-aform } X)$

$\langle \text{proof} \rangle$

**lemma** *fst-half-segments*:

*half-segments-of-aform*  $X \neq [] \implies$

*fst* (*half-segments-of-aform*  $X$  ! 0) = *lowest-vertex* (*fst*  $X$ , *nlex-pdevs* (*snd*  $X$ ))

*<proof>*

**lemma** *nlex-half-segments-of-aform*:  $(a, b) \in \text{set } (\text{half-segments-of-aform } X) \implies$

*lex*  $b$   $a$

*<proof>*

**lemma** *ccw-half-segments-of-aform-all*:

**assumes** *cd*:  $(c, d) \in \text{set } (\text{half-segments-of-aform } X)$

**shows** *list-all*  $(\lambda(xi, xj). \text{ccw } xi \ xj \ c \wedge \text{ccw } xi \ xj \ d)$  (*half-segments-of-aform*  $X$ )

*<proof>*

**lemma** *ccw-half-segments-of-aform*:

**assumes** *ij*:  $(xi, xj) \in \text{set } (\text{half-segments-of-aform } X)$

**assumes** *c*:  $(c, d) \in \text{set } (\text{half-segments-of-aform } X)$

**shows** *ccw*  $xi \ xj \ c \ \text{ccw } xi \ xj \ d$

*<proof>*

**lemma** *half-segments-of-aform1*:

**assumes** *ch*:  $x \in \text{convex hull set } (\text{map } \text{fst } (\text{half-segments-of-aform } X))$

**assumes** *ab*:  $(a, b) \in \text{set } (\text{half-segments-of-aform } X)$

**shows** *ccw*  $a \ b \ x$

*<proof>*

**lemma** *half-segments-of-aform2*:

**assumes** *ch*:  $x \in \text{convex hull set } (\text{map } \text{snd } (\text{half-segments-of-aform } X))$

**assumes** *ab*:  $(a, b) \in \text{set } (\text{half-segments-of-aform } X)$

**shows** *ccw*  $a \ b \ x$

*<proof>*

**lemma**

*in-set-half-segments-of-aform-aform-valE*:

**assumes**  $(x2, y2) \in \text{set } (\text{half-segments-of-aform } X)$

**obtains** *e* **where**  $y2 = \text{aform-val } e \ X \ e \in \text{UNIV} \rightarrow \{-1 .. 1\}$

*<proof>*

**lemma** *fst-hd-half-segments-of-aform*:

**assumes** *half-segments-of-aform*  $X \neq []$

**shows** *fst* (*hd* (*half-segments-of-aform*  $X$ )) = *lowest-vertex* (*fst*  $X$ , (*nlex-pdevs* (*snd*  $X$ )))

*<proof>*

**lemma**

*linorder-list0.sortedP* (*ccw'* (*lowest-vertex* (*fst*  $X$ , *nlex-pdevs* (*snd*  $X$ ))))

(*map* *snd* (*half-segments-of-aform*  $X$ ))

(**is** *linorder-list0.sortedP* (*ccw'* ?*x0*) ?*ms*)

*<proof>*

**lemma** *rev-zip*:  $length\ xs = length\ ys \implies rev\ (zip\ xs\ ys) = zip\ (rev\ xs)\ (rev\ ys)$   
*<proof>*

**lemma** *zip-upt-self-aux*:  $zip\ [0..<length\ xs]\ xs = map\ (\lambda i. (i, xs\ !\ i))\ [0..<length\ xs]$   
*<proof>*

**lemma** *half-segments-of-aform-strict*:  
**assumes**  $e \in UNIV \rightarrow \{-1 < .. < 1\}$   
**assumes**  $seg \in set\ (half-segments-of-aform\ X)$   
**assumes**  $length\ (half-segments-of-aform\ X) \neq 1$   
**shows**  $ccw'\ (fst\ seg)\ (snd\ seg)\ (aform-val\ e\ X)$   
*<proof>*

**lemma** *half-segments-of-aform-strict-all*:  
**assumes**  $e \in UNIV \rightarrow \{-1 < .. < 1\}$   
**assumes**  $length\ (half-segments-of-aform\ X) \neq 1$   
**shows**  $list-all\ (\lambda seg. ccw'\ (fst\ seg)\ (snd\ seg)\ (aform-val\ e\ X))\ (half-segments-of-aform\ X)$   
*<proof>*

**lemma** *list-allD*:  $list-all\ P\ xs \implies x \in set\ xs \implies P\ x$   
*<proof>*

**lemma** *minus-one-less-multI*:  
**fixes**  $e\ x::real$   
**shows**  $-1 \leq e \implies 0 < x \implies x < 1 \implies -1 < e * x$   
*<proof>*

**lemma** *less-one-multI*:  
**fixes**  $e\ x::real$   
**shows**  $e \leq 1 \implies 0 < x \implies x < 1 \implies e * x < 1$   
*<proof>*

**lemma** *ccw-half-segments-of-aform-strictI*:  
**assumes**  $e \in UNIV \rightarrow \{-1 < .. < 1\}$   
**assumes**  $(s1, s2) \in set\ (half-segments-of-aform\ X)$   
**assumes**  $length\ (half-segments-of-aform\ X) \neq 1$   
**assumes**  $x = (aform-val\ e\ X)$   
**shows**  $ccw'\ s1\ s2\ x$   
*<proof>*

**lemma**  
*ccw'-sortedP-subsequent*:  
**assumes**  $Suc\ i < length\ xs\ ccw'.sortedP\ 0\ (map\ dirvec\ xs)\ fst\ (xs\ !\ Suc\ i) = snd\ (xs\ !\ i)$   
**shows**  $ccw'\ (fst\ (xs\ !\ i))\ (snd\ (xs\ !\ i))\ (snd\ (xs\ !\ Suc\ i))$

*<proof>*

**lemma** *ccw'-sortedP-uminus*:  $ccw'.sortedP\ 0\ xs \implies ccw'.sortedP\ 0\ (map\ uminus\ xs)$

*<proof>*

**lemma** *subsequent-half-segments-ccw*:

**fixes**  $X$

**assumes**  $Suc\ i < length\ (half-segments-of-aform\ X)$

**shows**  $ccw'\ (fst\ (half-segments-of-aform\ X\ !\ i))\ (snd\ (half-segments-of-aform\ X\ !\ i))$

$(snd\ (half-segments-of-aform\ X\ !\ Suc\ i))$

*<proof>*

**lemma** *convex-polychain-half-segments-of-aform*:  $convex-polychain\ (half-segments-of-aform\ X)$

*<proof>*

**lemma** *hd-distinct-neq-last*:  $distinct\ xs \implies length\ xs > 1 \implies hd\ xs \neq last\ xs$

*<proof>*

**lemma** *ccw-hd-last-half-segments-dirvec*:

**assumes**  $length\ (half-segments-of-aform\ X) > 1$

**shows**  $ccw'\ 0\ (dirvec\ (hd\ (half-segments-of-aform\ X)))\ (dirvec\ (last\ (half-segments-of-aform\ X)))$

*<proof>*

**lemma** *map-fst-half-segments-aux-eq*:  $\square \neq half-segments-of-aform\ X \implies$

$map\ fst\ (half-segments-of-aform\ X) =$

$fst\ (hd\ (half-segments-of-aform\ X))\ \#butlast\ (map\ snd\ (half-segments-of-aform\ X))$

*<proof>*

**lemma** *le-less-Suc-eq*:  $x - Suc\ 0 \leq (i::nat) \implies i < x \implies x - Suc\ 0 = i$

*<proof>*

**lemma** *map-snd-half-segments-aux-eq*:  $half-segments-of-aform\ X \neq \square \implies$

$map\ snd\ (half-segments-of-aform\ X) =$

$tl\ (map\ fst\ (half-segments-of-aform\ X))\ @\ [snd\ (last\ (half-segments-of-aform\ X))]$

*<proof>*

**lemma** *ccw'-sortedP-snd-half-segments-of-aform*:

$ccw'.sortedP\ (lowest-vertex\ (fst\ X,\ nlex-pdevs\ (snd\ X)))\ (map\ snd\ (half-segments-of-aform\ X))$

*<proof>*

**lemma**

*lex-half-segments-lowest-vertex*:

**assumes**  $(c, d) \in \text{set } (\text{half-segments-of-aform } X)$   
**shows**  $\text{lex } d \text{ (lowest-vertex (fst } X, \text{ nlex-pdevs (snd } X)))}$   
 $\langle \text{proof} \rangle$

**lemma**

*lex-half-segments-lowest-vertex'*:  
**assumes**  $d \in \text{set } (\text{map snd } (\text{half-segments-of-aform } X))$   
**shows**  $\text{lex } d \text{ (lowest-vertex (fst } X, \text{ nlex-pdevs (snd } X)))}$   
 $\langle \text{proof} \rangle$

**lemma**

*lex-half-segments-last*:  
**assumes**  $(c, d) \in \text{set } (\text{half-segments-of-aform } X)$   
**shows**  $\text{lex } (\text{snd } (\text{last } (\text{half-segments-of-aform } X))) \text{ } d$   
 $\langle \text{proof} \rangle$

**lemma**

*lex-half-segments-last'*:  
**assumes**  $d \in \text{set } (\text{map snd } (\text{half-segments-of-aform } X))$   
**shows**  $\text{lex } (\text{snd } (\text{last } (\text{half-segments-of-aform } X))) \text{ } d$   
 $\langle \text{proof} \rangle$

**lemma**

*ccw'-half-segments-lowest-last*:  
**assumes**  $\text{set-butlast: } (c, d) \in \text{set } (\text{butlast } (\text{half-segments-of-aform } X))$   
**assumes**  $\text{ne: } \text{inl } (\text{snd } X) \neq []$   
**shows**  $\text{ccw}' \text{ (lowest-vertex (fst } X, \text{ nlex-pdevs (snd } X))) \text{ } d \text{ (snd } (\text{last } (\text{half-segments-of-aform } X)))}$   
 $\langle \text{proof} \rangle$

**lemma** *distinct-fst-half-segments*:

$\text{distinct } (\text{map fst } (\text{half-segments-of-aform } X))$   
 $\langle \text{proof} \rangle$

**lemma** *distinct-snd-half-segments*:

$\text{distinct } (\text{map snd } (\text{half-segments-of-aform } X))$   
 $\langle \text{proof} \rangle$

## 10.8 Mirror

**definition**  $\text{mirror-point } x \ y = 2 *_{\mathbb{R}} x - y$

**lemma** *ccw'-mirror-point3[simp]*:

$\text{ccw}' \text{ (mirror-point } x \ y) \text{ (mirror-point } x \ z) \text{ (mirror-point } x \ w) \longleftrightarrow \text{ccw}' \ y \ z \ w$   
 $\langle \text{proof} \rangle$

**lemma** *mirror-point-self-inverse[simp]*:

**fixes**  $x::'a::\text{real-vector}$   
**shows**  $\text{mirror-point } p \text{ (mirror-point } p \ x) = x$

*<proof>*

**lemma** *mirror-half-segments-of-aform:*

**assumes**  $e \in UNIV \rightarrow \{-1 < .. < 1\}$

**assumes**  $length\ (half-segments-of-aform\ X) \neq 1$

**shows**  $list-all\ (\lambda seg.\ ccw'\ (fst\ seg)\ (snd\ seg)\ (aform-val\ e\ X))$

$(map\ (pairsel\ (mirror-point\ (fst\ X)))\ (half-segments-of-aform\ X))$

*<proof>*

**lemma** *last-half-segments:*

**assumes**  $half-segments-of-aform\ X \neq []$

**shows**  $snd\ (last\ (half-segments-of-aform\ X)) =$

$mirror-point\ (fst\ X)\ (lowest-vertex\ (fst\ X,\ nlex-pdevs\ (snd\ X)))$

*<proof>*

**lemma** *convex-polychain-map-mirror:*

**assumes**  $convex-polychain\ hs$

**shows**  $convex-polychain\ (map\ (pairsel\ (mirror-point\ x))\ hs)$

*<proof>*

**lemma** *ccw'-mirror-point0:*

$ccw'\ (mirror-point\ x\ y)\ z\ w \longleftrightarrow ccw'\ y\ (mirror-point\ x\ z)\ (mirror-point\ x\ w)$

*<proof>*

**lemma** *ccw'-sortedP-mirror:*

$ccw'.sortedP\ x0\ (map\ (mirror-point\ p0)\ xs) \longleftrightarrow ccw'.sortedP\ (mirror-point\ p0\ x0)\ xs$

*<proof>*

**lemma** *ccw'-sortedP-mirror2:*

$ccw'.sortedP\ (mirror-point\ p0\ x0)\ (map\ (mirror-point\ p0)\ xs) \longleftrightarrow ccw'.sortedP\ x0\ xs$

*<proof>*

**lemma** *map-mirror-o-snd-polychain-of-eq:*  $map\ (mirror-point\ x0\ o\ snd)\ (polychain-of\ y\ xs) =$

$map\ snd\ (polychain-of\ (mirror-point\ x0\ y)\ (map\ uminus\ xs))$

*<proof>*

**lemma** *lowest-vertex-eq-mirror-last:*

$half-segments-of-aform\ X \neq [] \implies$

$(lowest-vertex\ (fst\ X,\ nlex-pdevs\ (snd\ X))) =$

$mirror-point\ (fst\ X)\ (snd\ (last\ (half-segments-of-aform\ X)))$

*<proof>*

**lemma** *snd-last:*  $xs \neq [] \implies snd\ (last\ xs) = last\ (map\ snd\ xs)$

*<proof>*

**lemma** *mirror-point-snd-last-eq-lowest:*

$half\text{-}segments\text{-}of\text{-}aform\ X \neq [] \implies$   
 $mirror\text{-}point\ (fst\ X)\ (last\ (map\ snd\ (half\text{-}segments\text{-}of\text{-}aform\ X))) =$   
 $lowest\text{-}vertex\ (fst\ X,\ nlex\text{-}pdevs\ (snd\ X))$   
 $\langle proof \rangle$

**lemma** *lex-mirror-point*:  $lex\ (mirror\text{-}point\ x0\ a)\ (mirror\text{-}point\ x0\ b) \implies lex\ b\ a$   
 $\langle proof \rangle$

**lemma** *ccw'-mirror-point*:  
 $ccw'\ (mirror\text{-}point\ x0\ a)\ (mirror\text{-}point\ x0\ b)\ (mirror\text{-}point\ x0\ c) \implies ccw'\ a\ b\ c$   
 $\langle proof \rangle$

**lemma** *inj-mirror-point*:  $inj\ (mirror\text{-}point\ (x::'a::real\text{-}vector))$   
 $\langle proof \rangle$

**lemma**  
*distinct-map-mirror-point-eq*:  
 $distinct\ (map\ (mirror\text{-}point\ (x::'a::real\text{-}vector))\ xs) = distinct\ xs$   
 $\langle proof \rangle$

**lemma** *eq-self-mirror-iff*: **fixes**  $x::'a::real\text{-}vector$  **shows**  $x = mirror\text{-}point\ y\ x \longleftrightarrow$   
 $x = y$   
 $\langle proof \rangle$

## 10.9 Full Segments

**definition** *segments-of-aform::point aform*  $\Rightarrow (point * point)\ list$   
**where** *segments-of-aform X* =  
 $(let\ hs = half\text{-}segments\text{-}of\text{-}aform\ X\ in\ hs\ @\ map\ (pairs\ self\ (mirror\text{-}point\ (fst\ X)))\ hs)$

**lemma** *segments-of-aform-strict*:  
**assumes**  $e \in UNIV \rightarrow \{-1 <..< 1\}$   
**assumes**  $length\ (half\text{-}segments\text{-}of\text{-}aform\ X) \neq 1$   
**shows**  $list\text{-}all\ (\lambda seg.\ ccw'\ (fst\ seg)\ (snd\ seg)\ (aform\text{-}val\ e\ X))\ (segments\text{-}of\text{-}aform\ X)$   
 $\langle proof \rangle$

**lemma** *mirror-point-aform-val[simp]*:  $mirror\text{-}point\ (fst\ X)\ (aform\text{-}val\ e\ X) = aform\text{-}val$   
 $(- e)\ X$   
 $\langle proof \rangle$

**lemma**  
*in-set-segments-of-aform-aform-valE*:  
**assumes**  $(x2,\ y2) \in set\ (segments\text{-}of\text{-}aform\ X)$   
**obtains**  $e$  **where**  $y2 = aform\text{-}val\ e\ X\ e \in UNIV \rightarrow \{-1 .. 1\}$   
 $\langle proof \rangle$

**lemma**

*last-half-segments-eq-mirror-hd:*

**assumes**  $\text{half-segments-of-aform } X \neq []$

**shows**  $\text{snd } (\text{last } (\text{half-segments-of-aform } X)) = \text{mirror-point } (\text{fst } X) (\text{fst } (\text{hd } (\text{half-segments-of-aform } X)))$

*<proof>*

**lemma** *polychain-segments-of-aform:*  $\text{polychain } (\text{segments-of-aform } X)$

*<proof>*

**lemma** *segments-of-aform-closed:*

**assumes**  $\text{segments-of-aform } X \neq []$

**shows**  $\text{fst } (\text{hd } (\text{segments-of-aform } X)) = \text{snd } (\text{last } (\text{segments-of-aform } X))$

*<proof>*

**lemma** *convex-polychain-segments-of-aform:*

**assumes**  $1 < \text{length } (\text{half-segments-of-aform } X)$

**shows**  $\text{convex-polychain } (\text{segments-of-aform } X)$

*<proof>*

**lemma** *convex-polygon-segments-of-aform:*

**assumes**  $1 < \text{length } (\text{half-segments-of-aform } X)$

**shows**  $\text{convex-polygon } (\text{segments-of-aform } X)$

*<proof>*

**lemma**

*previous-segments-of-aformE:*

**assumes**  $(y, z) \in \text{set } (\text{segments-of-aform } X)$

**obtains**  $x$  **where**  $(x, y) \in \text{set } (\text{segments-of-aform } X)$

*<proof>*

**lemma** *fst-compose-pairself:*  $\text{fst } o \text{ pairself } f = f o \text{ fst}$

**and** *snd-compose-pairself:*  $\text{snd } o \text{ pairself } f = f o \text{ snd}$

*<proof>*

**lemma** *in-set-butlastI:*  $xs \neq [] \implies x \in \text{set } xs \implies x \neq \text{last } xs \implies x \in \text{set } (\text{butlast } xs)$

*<proof>*

**lemma** *distinct-in-set-butlastD:*

$x \in \text{set } (\text{butlast } xs) \implies \text{distinct } xs \implies x \neq \text{last } xs$

*<proof>*

**lemma** *distinct-in-set-tlD:*

$x \in \text{set } (\text{tl } xs) \implies \text{distinct } xs \implies x \neq \text{hd } xs$

*<proof>*

**lemma** *ccw'-sortedP-snd-segments-of-aform:*

**assumes**  $\text{length } (\text{inl } (\text{snd } X)) > 1$

**shows**  
 $ccw'.sortedP$  ( $lowest\text{-}vertex$  ( $fst$   $X$ ,  $nlex\text{-}pdevs$  ( $snd$   $X$ )))  
( $butlast$  ( $map$   $snd$  ( $segments\text{-}of\text{-}aform$   $X$ )))  
 $\langle proof \rangle$

**lemma**  $polychain\text{-}of\text{-}segments\text{-}of\text{-}aform1$ :  
**assumes**  $length$  ( $segments\text{-}of\text{-}aform$   $X$ ) = 1  
**shows**  $False$   
 $\langle proof \rangle$

**lemma**  $polychain\text{-}of\text{-}segments\text{-}of\text{-}aform2$ :  
**assumes**  $segments\text{-}of\text{-}aform$   $X$  = [ $x$ ,  $y$ ]  
**assumes**  $between$  ( $fst$   $x$ ,  $snd$   $x$ )  $p$   
**shows**  $p \in convex\ hull\ set$  ( $map$   $fst$  ( $segments\text{-}of\text{-}aform$   $X$ ))  
 $\langle proof \rangle$

**lemma**  $append\text{-}eq\text{-}2$ :  
**assumes**  $length$   $xs$  =  $length$   $ys$   
**shows**  $xs @ ys$  = [ $x$ ,  $y$ ]  $\longleftrightarrow$   $xs$  = [ $x$ ]  $\wedge$   $ys$  = [ $y$ ]  
 $\langle proof \rangle$

**lemma**  $segments\text{-}of\text{-}aform\text{-}line\text{-}segment$ :  
**assumes**  $segments\text{-}of\text{-}aform$   $X$  = [ $x$ ,  $y$ ]  
**assumes**  $e \in UNIV \rightarrow \{-1 .. 1\}$   
**shows**  $aform\text{-}val$   $e$   $X \in closed\text{-}segment$  ( $fst$   $x$ ) ( $snd$   $x$ )  
 $\langle proof \rangle$

## 10.10 Continuous Generalization

**lemma**  $LIMSEQ\text{-}minus\text{-}fract\text{-}mult$ :  
 $(\lambda n. r * (1 - 1 / real (Suc (Suc n)))) \longrightarrow r$   
 $\langle proof \rangle$

**lemma**  $det3\text{-}nonneg\text{-}segments\text{-}of\text{-}aform$ :  
**assumes**  $e \in UNIV \rightarrow \{-1 .. 1\}$   
**assumes**  $length$  ( $half\text{-}segments\text{-}of\text{-}aform$   $X$ )  $\neq 1$   
**shows**  $list\text{-}all$  ( $\lambda seg. det3$  ( $fst$   $seg$ ) ( $snd$   $seg$ ) ( $aform\text{-}val$   $e$   $X$ )  $\geq 0$ ) ( $segments\text{-}of\text{-}aform$   $X$ )  
 $\langle proof \rangle$

**lemma**  $det3\text{-}nonneg\text{-}segments\text{-}of\text{-}aformI$ :  
**assumes**  $e \in UNIV \rightarrow \{-1 .. 1\}$   
**assumes**  $length$  ( $half\text{-}segments\text{-}of\text{-}aform$   $X$ )  $\neq 1$   
**assumes**  $seg \in set$  ( $segments\text{-}of\text{-}aform$   $X$ )  
**shows**  $det3$  ( $fst$   $seg$ ) ( $snd$   $seg$ ) ( $aform\text{-}val$   $e$   $X$ )  $\geq 0$   
 $\langle proof \rangle$

## 10.11 Intersection of Vertical Line with Segment

**fun**  $intersect\text{-}segment\text{-}xline'$ :: $nat \Rightarrow point * point \Rightarrow real \Rightarrow point\ option$

**where** *intersect-segment-xline'*  $p ((x0, y0), (x1, y1)) xl =$   
 (if  $x0 \leq xl \wedge xl \leq x1$  then  
 if  $x0 = x1$  then *Some* ((*min*  $y0 y1$ ), (*max*  $y0 y1$ ))  
 else  
 let  
 $yl = \text{truncate-down } p (\text{truncate-down } p (\text{real-divl } p (y1 - y0) (x1 - x0)$   
 $* (xl - x0)) + y0$ );  
 $yr = \text{truncate-up } p (\text{truncate-up } p (\text{real-divr } p (y1 - y0) (x1 - x0) * (xl$   
 $- x0)) + y0$ )  
 in *Some* ( $yl, yr$ )  
 else *None*)

**lemma** *norm-pair-fst0[simp]*:  $\text{norm } (0, x) = \text{norm } x$   
 ⟨*proof*⟩

**lemmas** *add-right-mono-le* = *order-trans[OF add-right-mono]*  
**lemmas** *mult-right-mono-le* = *order-trans[OF mult-right-mono]*

**lemmas** *add-right-mono-ge* = *order-trans[OF - add-right-mono]*  
**lemmas** *mult-right-mono-ge* = *order-trans[OF - mult-right-mono]*

**lemma** *dest-segment*:

**fixes**  $x b::\text{real}$   
**assumes**  $(x, b) \in \text{closed-segment } (x0, y0) (x1, y1)$   
**assumes**  $x0 \neq x1$   
**shows**  $b = (y1 - y0) * (x - x0) / (x1 - x0) + y0$   
 ⟨*proof*⟩

**lemma** *intersect-segment-xline'*:

**assumes** *intersect-segment-xline'* *prec* ( $p0, p1$ )  $x = \text{Some } (m, M)$   
**shows**  $\text{closed-segment } p0 p1 \cap \{p. \text{fst } p = x\} \subseteq \{(x, m) .. (x, M)\}$   
 ⟨*proof*⟩

**lemma**

*in-segment-fst-le*:  
**fixes**  $x0 x1 b::\text{real}$   
**assumes**  $x0 \leq x1$   $(x, b) \in \text{closed-segment } (x0, y0) (x1, y1)$   
**shows**  $x \leq x1$   
 ⟨*proof*⟩

**lemma**

*in-segment-fst-ge*:  
**fixes**  $x0 x1 b::\text{real}$   
**assumes**  $x0 \leq x1$   $(x, b) \in \text{closed-segment } (x0, y0) (x1, y1)$   
**shows**  $x0 \leq x$   
 ⟨*proof*⟩

**lemma** *intersect-segment-xline'-eq-None*:

**assumes** *intersect-segment-xline'* *prec* ( $p0, p1$ )  $x = \text{None}$

**assumes**  $\text{fst } p0 \leq \text{fst } p1$   
**shows**  $\text{closed-segment } p0 \ p1 \cap \{p. \text{fst } p = x\} = \{\}$   
 <proof>

**fun** *intersect-segment-xline*  
**where**  $\text{intersect-segment-xline } \text{prec } ((a, b), (c, d)) \ x =$   
 (if  $a \leq c$  then  $\text{intersect-segment-xline}' \ \text{prec } ((a, b), (c, d)) \ x$   
 else  $\text{intersect-segment-xline}' \ \text{prec } ((c, d), (a, b)) \ x$ )

**lemma** *closed-segment-commute*:  $\text{closed-segment } a \ b = \text{closed-segment } b \ a$   
 <proof>

**lemma** *intersect-segment-xline*:  
**assumes**  $\text{intersect-segment-xline } \text{prec } (p0, p1) \ x = \text{Some } (m, M)$   
**shows**  $\text{closed-segment } p0 \ p1 \cap \{p. \text{fst } p = x\} \subseteq \{(x, m) .. (x, M)\}$   
 <proof>

**lemma** *intersect-segment-xline-fst-snd*:  
**assumes**  $\text{intersect-segment-xline } \text{prec } \text{seg } x = \text{Some } (m, M)$   
**shows**  $\text{closed-segment } (\text{fst } \text{seg}) \ (\text{snd } \text{seg}) \cap \{p. \text{fst } p = x\} \subseteq \{(x, m) .. (x, M)\}$   
 <proof>

**lemma** *intersect-segment-xline-eq-None*:  
**assumes**  $\text{intersect-segment-xline } \text{prec } (p0, p1) \ x = \text{None}$   
**shows**  $\text{closed-segment } p0 \ p1 \cap \{p. \text{fst } p = x\} = \{\}$   
 <proof>

**lemma** *inter-image-empty-iff*:  $(X \cap \{p. \text{fst } p = x\} = \{\}) \longleftrightarrow (x \notin \text{fst } X)$   
 <proof>

**lemma** *fst-closed-segment[simp]*:  $\text{fst } \text{'closed-segment } a \ b = \text{closed-segment } (\text{fst } a)$   
 ( $\text{fst } b$ )  
 <proof>

**lemma** *intersect-segment-xline-eq-empty*:  
**fixes**  $p0 \ p1 :: \text{real} * \text{real}$   
**assumes**  $\text{closed-segment } p0 \ p1 \cap \{p. \text{fst } p = x\} = \{\}$   
**shows**  $\text{intersect-segment-xline } \text{prec } (p0, p1) \ x = \text{None}$   
 <proof>

**lemma** *intersect-segment-xline-le*:  
**assumes**  $\text{intersect-segment-xline } \text{prec } y \ xl = \text{Some } (m0, M0)$   
**shows**  $m0 \leq M0$   
 <proof>

**lemma** *intersect-segment-xline-None-iff*:  
**fixes**  $p0 \ p1 :: \text{real} * \text{real}$   
**shows**  $\text{intersect-segment-xline } \text{prec } (p0, p1) \ x = \text{None} \longleftrightarrow \text{closed-segment } p0 \ p1$   
 $\cap \{p. \text{fst } p = x\} = \{\}$

*<proof>*

## 10.12 Bounds on Vertical Intersection with Oriented List of Segments

**primrec** *bound-intersect-2d* **where**

*bound-intersect-2d prec [] x = None*  
| *bound-intersect-2d prec (X # Xs) xl =*  
  (*case bound-intersect-2d prec Xs xl of*  
    *None ⇒ intersect-segment-xline prec X xl*  
  | *Some (m, M) ⇒*  
    (*case intersect-segment-xline prec X xl of*  
      *None ⇒ Some (m, M)*  
    | *Some (m', M') ⇒ Some (min m' m, max M' M)))*

**lemma**

*bound-intersect-2d-eq-None:*  
**assumes** *bound-intersect-2d prec Xs x = None*  
**assumes**  $X \in \text{set } Xs$   
**shows** *intersect-segment-xline prec X x = None*  
*<proof>*

**lemma** *bound-intersect-2d-upper:*

**assumes** *bound-intersect-2d prec Xs x = Some (m, M)*  
**obtains**  $X m'$  **where**  $X \in \text{set } Xs$  *intersect-segment-xline prec X x = Some (m', M)*  
 $\bigwedge X m' M'. X \in \text{set } Xs \implies \text{intersect-segment-xline prec X x = Some (m', M')}$   
 $\implies M' \leq M$   
*<proof>*

**lemma** *bound-intersect-2d-lower:*

**assumes** *bound-intersect-2d prec Xs x = Some (m, M)*  
**obtains**  $X M'$  **where**  $X \in \text{set } Xs$  *intersect-segment-xline prec X x = Some (m, M')*  
 $\bigwedge X m' M'. X \in \text{set } Xs \implies \text{intersect-segment-xline prec X x = Some (m', M')}$   
 $\implies m \leq m'$   
*<proof>*

**lemma** *bound-intersect-2d:*

**assumes** *bound-intersect-2d prec Xs x = Some (m, M)*  
**shows**  $(\bigcup (p1, p2) \in \text{set } Xs. \text{closed-segment } p1 \ p2) \cap \{p. \text{fst } p = x\} \subseteq \{(x, m) .. (x, M)\}$   
*<proof>*

**lemma** *bound-intersect-2d-eq-None-iff:*

*bound-intersect-2d prec Xs x = None*  $\longleftrightarrow (\forall X \in \text{set } Xs. \text{intersect-segment-xline prec X x = None})$   
*<proof>*

**lemma** *bound-intersect-2d-nonempty*:

**assumes** *bound-intersect-2d* prec  $Xs$   $x = \text{Some } (m, M)$

**shows**  $(\bigcup (p1, p2) \in \text{set } Xs. \text{closed-segment } p1 \ p2) \cap \{p. \text{fst } p = x\} \neq \{\}$   
 $\langle \text{proof} \rangle$

**lemma** *bound-intersect-2d-le*:

**assumes** *bound-intersect-2d* prec  $Xs$   $x = \text{Some } (m, M)$  **shows**  $m \leq M$

$\langle \text{proof} \rangle$

### 10.13 Bounds on Vertical Intersection with General List of Segments

**definition** *bound-intersect-2d-ud* prec  $X$   $xl =$

(*case segments-of-aform*  $X$  of

$\square \Rightarrow$  if  $\text{fst } (\text{fst } X) = xl$  then let  $m = \text{snd } (\text{fst } X)$  in  $\text{Some } (m, m)$  else  $\text{None}$

|  $[x, y] \Rightarrow$  *intersect-segment-xline* prec  $x$   $xl$

|  $xs \Rightarrow$

(*case bound-intersect-2d* prec (*filter*  $(\lambda((x1, y1), x2, y2). x1 < x2)$   $xs$ )  $xl$  of  
 $\text{Some } (m, M') \Rightarrow$

(*case bound-intersect-2d* prec (*filter*  $(\lambda((x1, y1), x2, y2). x1 > x2)$   $xs$ )  $xl$  of  
 $\text{Some } (m', M) \Rightarrow \text{Some } (\min m \ m', \max M \ M')$

|  $\text{None} \Rightarrow \text{None}$ )

|  $\text{None} \Rightarrow \text{None}$ )

**lemma** *list-cases4*:

$\bigwedge x P. (x = \square \Longrightarrow P) \Longrightarrow (\bigwedge y. x = [y] \Longrightarrow P) \Longrightarrow$

$(\bigwedge y z. x = [y, z] \Longrightarrow P) \Longrightarrow$

$(\bigwedge w y z zs. x = w \# y \# z \# zs \Longrightarrow P) \Longrightarrow P$

$\langle \text{proof} \rangle$

**lemma** *bound-intersect-2d-ud-segments-of-aform-le*:

*bound-intersect-2d-ud* prec  $X$   $xl = \text{Some } (m0, M0) \Longrightarrow m0 \leq M0$

$\langle \text{proof} \rangle$

**lemma** *pdevs-domain-eq-empty-iff[simp]*:  $\text{pdevs-domain } (\text{snd } X) = \{\} \longleftrightarrow \text{snd } X$

$= \text{zero-pdevs}$

$\langle \text{proof} \rangle$

**lemma** *ccw-contr-on-line-left*:

**assumes**  $\text{det3 } (a, b) (x, c) (x, d) \geq 0$   $a > x$

**shows**  $d \leq c$

$\langle \text{proof} \rangle$

**lemma** *ccw-contr-on-line-right*:

**assumes**  $\text{det3 } (a, b) (x, c) (x, d) \geq 0$   $a < x$

**shows**  $d \geq c$

$\langle \text{proof} \rangle$

**lemma** *singleton-contrE*:

**assumes**  $\bigwedge x y. x \neq y \implies x \in X \implies y \in X \implies \text{False}$   
**assumes**  $X \neq \{\}$   
**obtains**  $x$  **where**  $X = \{x\}$   
 $\langle \text{proof} \rangle$

**lemma** *segment-intersection-singleton*:  
**fixes**  $xl$  **and**  $a b::\text{real} * \text{real}$   
**defines**  $i \equiv \text{closed-segment } a b \cap \{p. \text{fst } p = xl\}$   
**assumes**  $ne1: \text{fst } a \neq \text{fst } b$   
**assumes**  $upper: i \neq \{\}$   
**obtains**  $p1$  **where**  $i = \{p1\}$   
 $\langle \text{proof} \rangle$

**lemma** *bound-intersect-2d-ud-segments-of-aform*:  
**assumes**  $\text{bound-intersect-2d-ud } prec \ X \ xl = \text{Some } (m0, M0)$   
**assumes**  $e \in UNIV \rightarrow \{-1 .. 1\}$   
**shows**  $\{\text{aform-val } e \ X\} \cap \{x. \text{fst } x = xl\} \subseteq \{(xl, m0) .. (xl, M0)\}$   
 $\langle \text{proof} \rangle$

## 10.14 Approximation from Orthogonal Directions

**definition** *inter-aform-plane-ortho*:  
 $\text{nat} \Rightarrow 'a::\text{executable-euclidean-space } \text{aform} \Rightarrow 'a \Rightarrow \text{real} \Rightarrow 'a \ \text{aform } \text{option}$   
**where**  
 $\text{inter-aform-plane-ortho } p \ Z \ n \ g = \text{do } \{$   
 $\quad mMs \leftarrow \text{those } (\lambda b. \text{bound-intersect-2d-ud } p \ (\text{inner2-aform } Z \ n \ b) \ g)$   
 $\quad \text{Basis-list};$   
 $\quad \text{let } l = (\sum (b, m) \leftarrow \text{zip } \text{Basis-list } (\text{map } \text{fst } mMs). m *_{\mathbb{R}} b);$   
 $\quad \text{let } u = (\sum (b, M) \leftarrow \text{zip } \text{Basis-list } (\text{map } \text{snd } mMs). M *_{\mathbb{R}} b);$   
 $\quad \text{Some } (\text{aform-of-ivl } l \ u)$   
 $\}$

**lemma**  
 $\text{those-eq-SomeD}$ :  
**assumes**  $\text{those } (\text{map } f \ xs) = \text{Some } ys$   
**shows**  $ys = \text{map } (\text{the } o \ f) \ xs \wedge (\forall i. \exists y. i < \text{length } xs \longrightarrow f \ (xs \ ! \ i) = \text{Some } y)$   
 $\langle \text{proof} \rangle$

**lemma**  
 $\text{sum-list-zip-map}$ :  
**assumes**  $\text{distinct } xs$   
**shows**  $(\sum (x, y) \leftarrow \text{zip } xs \ (\text{map } g \ xs). f \ x \ y) = (\sum x \in \text{set } xs. f \ x \ (g \ x))$   
 $\langle \text{proof} \rangle$

**lemma**  
 $\text{inter-aform-plane-ortho-overappr}$ :  
**assumes**  $\text{inter-aform-plane-ortho } p \ Z \ n \ g = \text{Some } X$   
**shows**  $\{x. \forall i \in \text{Basis}. x \cdot i \in \{y. (g, y) \in (\lambda x. (x \cdot n, x \cdot i)) \ \text{Affine } Z\}\} \subseteq \text{Affine } X$

*<proof>*

**lemma** *inter-proj-eq*:

**fixes**  $n\ g\ l$

**defines**  $G \equiv \{x. x \cdot n = g\}$

**shows**  $(\lambda x. x \cdot l) \cdot (Z \cap G) =$   
 $\{y. (g, y) \in (\lambda x. (x \cdot n, x \cdot l)) \cdot Z\}$

*<proof>*

**lemma**

*inter-overappr*:

**fixes**  $n\ \gamma\ l$

**shows**  $Z \cap \{x. x \cdot n = g\} \subseteq \{x. \forall i \in \text{Basis}. x \cdot i \in \{y. (g, y) \in (\lambda x. (x \cdot n, x \cdot i)) \cdot Z\}\}$

*<proof>*

**lemma** *inter-inter-aform-plane-ortho*:

**assumes** *inter-aform-plane-ortho*  $p\ Z\ n\ g = \text{Some } X$

**shows**  $\text{Affine } Z \cap \{x. x \cdot n = g\} \subseteq \text{Affine } X$

*<proof>*

## 10.15 “Completeness” of Intersection

**abbreviation** *encompasses*  $x\ \text{seg} \equiv \text{det3 } (\text{fst } \text{seg})\ (\text{snd } \text{seg})\ x \geq 0$

**lemma** *encompasses-cases*:

*encompasses*  $x\ \text{seg} \vee \text{encompasses } x\ (\text{snd } \text{seg}, \text{fst } \text{seg})$

*<proof>*

**lemma** *list-all-encompasses-cases*:

**assumes** *list-all* (*encompasses*  $p$ )  $(x\ \# y\ \# zs)$

**obtains** *list-all* (*encompasses*  $p$ )  $[x, y, (\text{snd } y, \text{fst } x)]$

| *list-all* (*encompasses*  $p$ )  $((\text{fst } x, \text{snd } y)\ \# zs)$

*<proof>*

**lemma** *triangle-encompassing-polychain-of*:

**assumes**  $\text{det3 } p\ a\ b \geq 0\ \text{det3 } p\ b\ c \geq 0\ \text{det3 } p\ c\ a \geq 0$

**assumes**  $\text{ccw}'\ a\ b\ c$

**shows**  $p \in \text{convex hull } \{a, b, c\}$

*<proof>*

**lemma** *encompasses-convex-polygon3*:

**assumes** *list-all* (*encompasses*  $p$ )  $(x\ \# y\ \# z\ \# zs)$

**assumes** *convex-polygon*  $(x\ \# y\ \# z\ \# zs)$

**assumes**  $\text{ccw}'.\text{sortedP } (\text{fst } x)\ (\text{map } \text{snd } (\text{butlast } (x\ \# y\ \# z\ \# zs)))$

**shows**  $p \in \text{convex hull } (\text{set } (\text{map } \text{fst } (x\ \# y\ \# z\ \# zs)))$

*<proof>*

**lemma** *segments-of-aform-empty-Affine-eq*:

**assumes** *segments-of-aform*  $X = []$   
**shows** *Affine*  $X = \{fst\ X\}$   
 $\langle proof \rangle$

**lemma** *not-segments-of-aform-singleton*: *segments-of-aform*  $X \neq [x]$   
 $\langle proof \rangle$

**lemma** *encompasses-segments-of-aform-in-AffineI*:  
**assumes** *length* (*segments-of-aform*  $X$ )  $> 2$   
**assumes** *list-all* (*encompasses*  $p$ ) (*segments-of-aform*  $X$ )  
**shows**  $p \in \textit{Affine } X$   
 $\langle proof \rangle$

**end**

## 11 Implementation

**theory** *Affine-Code*  
**imports**  
*Affine-Approximation*  
*Intersection*  
**begin**

Implementing partial deviations as sorted lists of coefficients.

### 11.1 Reverse Sorted, Distinct Association Lists

**typedef** (**overloaded**) ( $'a, 'b$ ) *slist* =  
 $\{xs::('a::\textit{linorder} \times 'b)\ \textit{list}. \textit{distinct} (\textit{map}\ \textit{fst}\ xs) \wedge \textit{sorted} (\textit{rev} (\textit{map}\ \textit{fst}\ xs))\}$   
 $\langle proof \rangle$

**setup-lifting** *type-definition-slist*

**lift-definition** *map-of-slist*::( $\textit{nat}, 'a::\textit{zero}$ ) *slist*  $\Rightarrow$   $\textit{nat} \Rightarrow 'a\ \textit{option}$  **is** *map-of*  
 $\langle proof \rangle$

**lemma** *finite-dom-map-of-slist*[*intro, simp*]: *finite* (*dom* (*map-of-slist*  $xs$ ))  
 $\langle proof \rangle$

**abbreviation** *the-default*  $a\ x \equiv (\textit{case}\ x\ \textit{of}\ \textit{None} \Rightarrow a \mid \textit{Some}\ b \Rightarrow b)$

**definition** *Pdevs-raw*  $xs\ i = \textit{the-default}\ 0\ (\textit{map-of}\ xs\ i)$

**lemma** *nonzeros-Pdevs-raw-subset*:  $\{i. \textit{Pdevs-raw}\ xs\ i \neq 0\} \subseteq \textit{dom} (\textit{map-of}\ xs)$   
 $\langle proof \rangle$

**lift-definition** *Pdevs*::( $\textit{nat}, 'a::\textit{zero}$ ) *slist*  $\Rightarrow 'a\ \textit{pdevs}$   
**is** *Pdevs-raw*  
 $\langle proof \rangle$

**code-datatype** *Pdevs*

## 11.2 Degree

**primrec** *degree-list*::(nat × 'a::zero) list ⇒ nat **where**  
  *degree-list* [] = 0  
| *degree-list* (x#xs) = (if snd x = 0 then *degree-list* xs else Suc (fst x))

**lift-definition** *degree-slist*::(nat, 'a::zero) slist ⇒ nat **is** *degree-list* ⟨proof⟩

**lemma** *degree-list-eq-zeroD*:  
  **assumes** *degree-list* xs = 0  
  **shows** the-default 0 (map-of xs i) = 0  
  ⟨proof⟩

**lemma** *degree-slist-eq-zeroD*: *degree-slist* xs = 0 ⇒ *degree* (Pdevs xs) = 0  
  ⟨proof⟩

**lemma** *degree-slist-eq-SucD*: *degree-slist* xs = Suc n ⇒ *pdevs-apply* (Pdevs xs) n ≠ 0  
  ⟨proof⟩

**lemma** *degree-slist-zero*:  
  *degree-slist* xs = n ⇒ n ≤ j ⇒ *pdevs-apply* (Pdevs xs) j = 0  
  ⟨proof⟩

**lemma** *compute-degree[code]*: *degree* (Pdevs xs) = *degree-slist* xs  
  ⟨proof⟩

## 11.3 Auxiliary Definitions

**fun** *binop* **where**  
  *binop* f z1 z2 [] [] = []  
| *binop* f z1 z2 ((i, x)#xs) [] = (i, f x z2) # *binop* f z1 z2 xs []  
| *binop* f z1 z2 [] ((i, y)#ys) = (i, f z1 y) # *binop* f z1 z2 [] ys  
| *binop* f z1 z2 ((i, x)#xs) ((j, y)#ys) =  
  (if (i = j) then (i, f x y) # *binop* f z1 z2 xs ys  
  else if (i > j) then (i, f x z2) # *binop* f z1 z2 xs ((j, y)#ys)  
  else (j, f z1 y) # *binop* f z1 z2 ((i, x)#xs) ys)

**lemma** *set-binop-elemD1*:  
  (a, b) ∈ set (*binop* f z1 z2 xs ys) ⇒ (a ∈ set (map fst xs) ∨ a ∈ set (map fst ys))  
  ⟨proof⟩

**lemma** *set-binop-elemD2*:  
  (a, b) ∈ set (*binop* f z1 z2 xs ys) ⇒  
  (∃ x ∈ set (map snd xs). b = f x z2) ∨  
  (∃ y ∈ set (map snd ys). b = f z1 y) ∨

$(\exists x \in \text{set } (\text{map } \text{snd } xs). \exists y \in \text{set } (\text{map } \text{snd } ys). b = f x y)$   
 $\langle \text{proof} \rangle$

**abbreviation**  $\text{rsorted} \equiv \lambda x. \text{sorted } (\text{rev } x)$

**lemma** *rsorted-binop*:

**fixes**  $xs::('a::\text{linorder} * 'b) \text{ list}$  **and**  $ys::('a::\text{linorder} * 'c) \text{ list}$   
**assumes**  $\text{rsorted } ((\text{map } \text{fst } xs))$   
**assumes**  $\text{rsorted } ((\text{map } \text{fst } ys))$   
**shows**  $\text{rsorted } ((\text{map } \text{fst } (\text{binop } f z1 z2 xs ys)))$   
 $\langle \text{proof} \rangle$

**lemma** *distinct-binop*:

**fixes**  $xs::('a::\text{linorder} * 'b) \text{ list}$  **and**  $ys::('a::\text{linorder} * 'c) \text{ list}$   
**assumes**  $\text{distinct } (\text{map } \text{fst } xs)$   
**assumes**  $\text{distinct } (\text{map } \text{fst } ys)$   
**assumes**  $\text{rsorted } ((\text{map } \text{fst } xs))$   
**assumes**  $\text{rsorted } ((\text{map } \text{fst } ys))$   
**shows**  $\text{distinct } (\text{map } \text{fst } (\text{binop } f z1 z2 xs ys))$   
 $\langle \text{proof} \rangle$

**lemma** *binop-plus*:

**fixes**  $b::(\text{nat} * 'a::\text{euclidean-space}) \text{ list}$   
**shows**  
 $(\sum (i, y) \leftarrow \text{binop } (+) 0 0 b \text{ ba. } e i *_R y) = (\sum (i, y) \leftarrow b. e i *_R y) + (\sum (i, y) \leftarrow \text{ba. } e i *_R y)$   
 $\langle \text{proof} \rangle$

**lemma** *binop-compose*:

$\text{binop } (\lambda x y. f (g x y)) z1 z2 xs ys = \text{map } (\text{apsnd } f) (\text{binop } g z1 z2 xs ys)$   
 $\langle \text{proof} \rangle$

**lemma** *linear-cmul-left*[*intro, simp*]:  $\text{linear } ((* ) x::\text{real} \Rightarrow -)$   
 $\langle \text{proof} \rangle$

**lemma** *length-merge-sorted-eq*:

$\text{length } (\text{binop } f z1 z2 xs ys) = \text{length } (\text{binop } g y1 y2 xs ys)$   
 $\langle \text{proof} \rangle$

## 11.4 Pointwise Addition

**lift-definition**  $\text{add-slist}::(\text{nat}, 'a::\{\text{plus, zero}\}) \text{ slist} \Rightarrow (\text{nat}, 'a) \text{ slist} \Rightarrow (\text{nat}, 'a) \text{ slist}$  **is**

$\lambda xs ys. \text{binop } (+) 0 0 xs ys$   
 $\langle \text{proof} \rangle$

**lemma** *map-of-binop*[*simp*]:  $\text{rsorted } (\text{map } \text{fst } xs) \Longrightarrow \text{rsorted } (\text{map } \text{fst } ys) \Longrightarrow$   
 $\text{distinct } (\text{map } \text{fst } xs) \Longrightarrow \text{distinct } (\text{map } \text{fst } ys) \Longrightarrow$   
 $\text{map-of } (\text{binop } f z1 z2 xs ys) i =$

(*case map-of xs i of*  
*Some x*  $\Rightarrow$  *Some (f x (case map-of ys i of Some x*  $\Rightarrow$  *x | None*  $\Rightarrow$  *z2))*  
*| None*  $\Rightarrow$  (*case map-of ys i of Some y*  $\Rightarrow$  *Some (f z1 y) | None*  $\Rightarrow$  *None*))  
*<proof>*

**lemma** *pdevs-apply-Pdevs-add-slist[simp]*:  
**fixes** *xs ys::(nat, 'a::monoid-add) slist*  
**shows** *pdevs-apply (Pdevs (add-slist xs ys)) i =*  
*pdevs-apply (Pdevs xs) i + pdevs-apply (Pdevs ys) i*  
*<proof>*

**lemma** *compute-add-pdevs[code]*: *add-pdevs (Pdevs xs) (Pdevs ys) = Pdevs (add-slist xs ys)*  
*<proof>*

## 11.5 prod of pdevs

**lift-definition** *prod-slist::(nat, 'a::zero) slist*  $\Rightarrow$  (*nat, 'b::zero) slist*  $\Rightarrow$  (*nat, ('a*  $\times$  *'b)) slist* **is**  
 $\lambda xs ys. \text{binop Pair } 0 \ 0 \ xs \ ys$   
*<proof>*

**lemma** *pdevs-apply-Pdevs-prod-slist[simp]*:  
*pdevs-apply (Pdevs (prod-slist xs ys)) i = (pdevs-apply (Pdevs xs) i, pdevs-apply (Pdevs ys) i)*  
*<proof>*

**lemma** *compute-prod-of-pdevs[code]*: *prod-of-pdevs (Pdevs xs) (Pdevs ys) = Pdevs (prod-slist xs ys)*  
*<proof>*

## 11.6 Set of Coefficients

**lift-definition** *set-slist::(nat, 'a::real-vector) slist*  $\Rightarrow$  (*nat \* 'a) set* **is** *set* *<proof>*

**lemma** *finite-set-slist[intro, simp]*: *finite (set-slist xs)*  
*<proof>*

## 11.7 Domain

**lift-definition** *list-of-slist::('a::linorder, 'b::zero) slist*  $\Rightarrow$  (*'a \* 'b) list*  
**is**  $\lambda xs. \text{filter } (\lambda x. \text{snd } x \neq 0) \ xs$  *<proof>*

**lemma** *compute-pdevs-domain[code]*: *pdevs-domain (Pdevs xs) = set (map fst (list-of-slist xs))*  
*<proof>*

**lemma** *sort-rev-eq-sort*: *distinct xs*  $\implies$  *sort (rev xs) = sort xs*  
*<proof>*

**lemma** *compute-list-of-pdevs*[code]:  $list\text{-of-pdevs } (Pdevs\ xs) = list\text{-of-slist } xs$   
 ⟨proof⟩

**lift-definition** *slist-of-pdevs*:: $'a\ pdevs \Rightarrow (nat, 'a::real\text{-vector})\ slist$  **is**  $list\text{-of-pdevs}$   
 ⟨proof⟩

## 11.8 Application

**lift-definition** *slist-apply*:: $( 'a::linorder, 'b::zero)\ slist \Rightarrow 'a \Rightarrow 'b$  **is**  
 $\lambda xs\ i.\ the\text{-default } 0\ (map\text{-of } xs\ i)$  ⟨proof⟩

**lemma** *compute-pdevs-apply*[code]:  $pdevs\text{-apply } (Pdevs\ x)\ i = slist\text{-apply } x\ i$   
 ⟨proof⟩

## 11.9 Total Deviation

**lift-definition** *tdev-slist*:: $(nat, 'a::ordered\text{-euclidean-space})\ slist \Rightarrow 'a$  **is**  
 $sum\text{-list } o\ map\ (abs\ o\ snd)$  ⟨proof⟩

**lemma** *tdev-slist-sum*:  $tdev\text{-slist } xs = sum\ (abs\ o\ snd)\ (set\text{-slist } xs)$   
 ⟨proof⟩

**lemma** *pdevs-apply-set-slist*:  $x \in set\text{-slist } xs \Longrightarrow snd\ x = pdevs\text{-apply } (Pdevs\ xs)$   
 (fst  $x$ )  
 ⟨proof⟩

**lemma**  
*tdev-list-eq-zeroI*:  
**shows**  $(\bigwedge i.\ pdevs\text{-apply } (Pdevs\ xs)\ i = 0) \Longrightarrow tdev\text{-slist } xs = 0$   
 ⟨proof⟩

**lemma** *inj-on-fst-set-slist*:  $inj\text{-on } fst\ (set\text{-slist } xs)$   
 ⟨proof⟩

**lemma** *pdevs-apply-Pdevs-eq-0*:  
 $pdevs\text{-apply } (Pdevs\ xs)\ i = 0 \longleftrightarrow ((\forall x.\ (i, x) \in set\text{-slist } xs \longrightarrow x = 0))$   
 ⟨proof⟩

**lemma** *compute-tdev*[code]:  $tdev\ (Pdevs\ xs) = tdev\text{-slist } xs$   
 ⟨proof⟩

## 11.10 Minkowski Sum

**lemma** *dropWhile-rsorted-eq-filter*:  
 $rsorted\ (map\ fst\ xs) \Longrightarrow drop\ While\ (\lambda(i, x).\ i \geq (m::nat))\ xs = filter\ (\lambda(i, x).\ i < m)\ xs$   
 (**is**  $\Longrightarrow ?lhs\ xs = ?rhs\ xs$ )  
 ⟨proof⟩

**lift-definition** *msum-slist*:: $nat \Rightarrow (nat, 'a)\ slist \Rightarrow (nat, 'a)\ slist \Rightarrow (nat, 'a)\ slist$

**is**  $\lambda m \ xs \ ys. \text{map} (\text{apfst} (\lambda n. n + m)) \ ys \ @ \ \text{dropWhile} (\lambda(i, x). i \geq m) \ xs$   
 ⟨proof⟩

**lemma** *slist-apply-msum-slist*:

*slist-apply* (*msum-slist*  $m \ xs \ ys$ )  $i =$   
 (if  $i < m$  then *slist-apply*  $xs \ i$  else *slist-apply*  $ys \ (i - m)$ )  
 ⟨proof⟩

**lemma** *pdevs-apply-msum-slist*:

*pdevs-apply* (*Pdevs* (*msum-slist*  $m \ xs \ ys$ ))  $i =$   
 (if  $i < m$  then *pdevs-apply* (*Pdevs*  $xs$ )  $i$  else *pdevs-apply* (*Pdevs*  $ys$ ) ( $i - m$ ))  
 ⟨proof⟩

**lemma** *compute-msum-pdevs*[code]: *msum-pdevs*  $m \ (Pdevs \ xs) \ (Pdevs \ ys) = Pdevs$   
 (*msum-slist*  $m \ xs \ ys$ )  
 ⟨proof⟩

## 11.11 Unary Operations

**lift-definition** *map-slist*::( $'a \Rightarrow 'b$ )  $\Rightarrow (nat, 'a) \text{slist} \Rightarrow (nat, 'b) \text{slist}$  **is**  $\lambda f. \text{map}$   
 (*apsnd*  $f$ )  
 ⟨proof⟩

**lemma** *pdevs-apply-map-slist*:

$f \ 0 = 0 \implies \text{pdevs-apply} \ (Pdevs \ (\text{map-slist} \ f \ xs)) \ i = f \ (\text{pdevs-apply} \ (Pdevs \ xs) \ i)$   
 ⟨proof⟩

**lemma** *compute-scaleR-pdves*[code]: *scaleR-pdevs*  $r \ (Pdevs \ xs) = Pdevs \ (\text{map-slist}$   
 ( $\lambda x. r *_{\mathbb{R}} x$ )  $xs$ )

**and** *compute-pdevs-scaleR*[code]: *pdevs-scaleR* (*Pdevs*  $rs$ )  $x = Pdevs \ (\text{map-slist}$   
 ( $\lambda r. r *_{\mathbb{R}} x$ )  $rs$ )

**and** *compute-uminus-pdevs*[code]: *uminus-pdevs* (*Pdevs*  $xs$ ) = *Pdevs* (*map-slist*  
 ( $\lambda x. - x$ )  $xs$ )

**and** *compute-abs-pdevs*[code]: *abs-pdevs* (*Pdevs*  $xs$ ) = *Pdevs* (*map-slist* *abs*  $xs$ )

**and** *compute-pdevs-inner*[code]: *pdevs-inner* (*Pdevs*  $xs$ )  $b = Pdevs \ (\text{map-slist} \ (\lambda x.$   
 $x \cdot b) \ xs)$

**and** *compute-pdevs-inner2*[code]:

*pdevs-inner2* (*Pdevs*  $xs$ )  $b \ c = Pdevs \ (\text{map-slist} \ (\lambda x. (x \cdot b, x \cdot c)) \ xs)$

**and** *compute-inner-scaleR-pdevs*[code]:

*inner-scaleR-pdevs*  $x \ (Pdevs \ ys) = Pdevs \ (\text{map-slist} \ (\lambda y. (x \cdot y) *_{\mathbb{R}} y) \ ys)$

**and** *compute-trunc-pdevs*[code]:

*trunc-pdevs*  $p \ (Pdevs \ xs) = Pdevs \ (\text{map-slist} \ (\lambda x. \text{eucl-truncate-down} \ p \ x) \ xs)$

**and** *compute-trunc-err-pdevs*[code]:

*trunc-err-pdevs*  $p \ (Pdevs \ xs) = Pdevs \ (\text{map-slist} \ (\lambda x. \text{eucl-truncate-down} \ p \ x -$   
 $x) \ xs)$   
 ⟨proof⟩

## 11.12 Filter

**lift-definition** *filter-slist*::(nat  $\Rightarrow$  'a  $\Rightarrow$  bool)  $\Rightarrow$  (nat, 'a) slist  $\Rightarrow$  (nat, 'a) slist  
is  $\lambda P xs. \text{filter } (\lambda(i, x). (P i x)) xs$   
{proof}

**lemma** *slist-apply-filter-slist*: *slist-apply* (*filter-slist* P xs) i =  
(if P i (*slist-apply* xs i) then *slist-apply* xs i else 0)  
{proof}

**lemma** *pdevs-apply-filter-slist*: *pdevs-apply* (Pdevs (*filter-slist* P xs)) i =  
(if P i (*pdevs-apply* (Pdevs xs) i) then *pdevs-apply* (Pdevs xs) i else 0)  
{proof}

**lemma** *compute-filter-pdevs*[code]: *filter-pdevs* P (Pdevs xs) = Pdevs (*filter-slist* P xs)  
{proof}

## 11.13 Constant

**lift-definition** *zero-slist*::(nat, 'a) slist is [] {proof}

**lemma** *compute-zero-pdevs*[code]: *zero-pdevs* = Pdevs (*zero-slist*)  
{proof}

**lift-definition** *One-slist*::(nat, 'a::executable-euclidean-space) slist  
is *rev* (*zip* [0..*length* (*Basis-list*::'a list)] (*Basis-list*::'a list))  
{proof}

**lemma**  
*map-of-rev-zip-upto-length-eq-nth*:  
**assumes**  $i < \text{length } B$   $d = \text{length } B$   
**shows** (*map-of* (*rev* (*zip* [0..*d*] B)) i) = *Some* (B ! i)  
{proof}

**lemma**  
*map-of-rev-zip-upto-length-eq-None*:  
**assumes**  $\neg i < \text{length } B$   
**assumes**  $d = \text{length } B$   
**shows** (*map-of* (*rev* (*zip* [0..*d*] B)) i) = *None*  
{proof}

**lemma** *pdevs-apply-One-slist*:  
*pdevs-apply* (Pdevs *One-slist*) i =  
(if  $i < \text{length} (Basis-list::'a::executable-euclidean-space \text{ list})$   
then (*Basis-list*::'a list) ! i  
else 0)  
{proof}

**lemma** *compute-One-pdevs*[code]: *One-pdevs* = Pdevs *One-slist*

*<proof>*

**lift-definition** *coord-slist::nat*  $\Rightarrow$  (*nat, real*) *slist* **is**  $\lambda i. [(i, 1)]$  *<proof>*

**lemma** *compute-coord-pdevs[code]*: *coord-pdevs* *i* = *Pdevs* (*coord-slist* *i*)  
*<proof>*

## 11.14 Update

**primrec** *update-list::nat*  $\Rightarrow$  '*a*  $\Rightarrow$  (*nat* \* '*a*) *list*  $\Rightarrow$  (*nat* \* '*a*) *list*

**where**

*update-list* *n* *x* [] = [(*n*, *x*)]  
| *update-list* *n* *x* (*y*#*ys*) =  
  (*if* *n* > *fst* *y* *then* (*n*, *x*)#*y*#*ys*  
  else *if* *n* = *fst* *y* *then* (*n*, *x*)#*ys*  
  else *y*#(*update-list* *n* *x* *ys*))

**lemma** *map-of-update-list[simp]*: *map-of* (*update-list* *n* *x* *ys*) = (*map-of* *ys*)(*n*:=*Some* *x*)  
*<proof>*

**lemma** *in-set-update-listD*:

**assumes** *y*  $\in$  *set* (*update-list* *n* *x* *ys*)

**shows** *y* = (*n*, *x*)  $\vee$  (*y*  $\in$  *set* *ys*)

*<proof>*

**lemma** *in-set-update-listI*:

*y* = (*n*, *x*)  $\vee$  (*fst* *y*  $\neq$  *n*  $\wedge$  *y*  $\in$  *set* *ys*)  $\implies$  *y*  $\in$  *set* (*update-list* *n* *x* *ys*)

*<proof>*

**lemma** *in-set-update-list*: (*n*, *x*)  $\in$  *set* (*update-list* *n* *x* *xs*)

*<proof>*

**lemma** *overwrite-update-list*: (*a*, *b*)  $\in$  *set* *xs*  $\implies$  (*a*, *b*)  $\notin$  *set* (*update-list* *n* *x* *xs*)  
 $\implies$  *a* = *n*

*<proof>*

**lemma** *insert-update-list*:

*distinct* (*map* *fst* *xs*)  $\implies$  *rsorted* (*map* *fst* *xs*)  $\implies$  (*a*, *b*)  $\in$  *set* (*update-list* *a* *x* *xs*)  $\implies$  *b* = *x*

*<proof>*

**lemma** *set-update-list-eq*: *distinct* (*map* *fst* *xs*)  $\implies$  *rsorted* (*map* *fst* *xs*)  $\implies$

*set* (*update-list* *n* *x* *xs*) = *insert* (*n*, *x*) (*set* *xs* - {*x*. *fst* *x* = *n*})

*<proof>*

**lift-definition** *update-slist::nat*  $\Rightarrow$  '*a*  $\Rightarrow$  (*nat*, '*a*) *slist*  $\Rightarrow$  (*nat*, '*a*) *slist* **is** *update-list*

*<proof>*

**lemma** *pdevs-apply-update-slist*:  $pdevs\text{-}apply\ (Pdevs\ (update\text{-}slist\ n\ x\ xs))\ i =$   
*(if*  $i = n$  *then*  $x$  *else*  $pdevs\text{-}apply\ (Pdevs\ xs)\ i$ )  
*<proof>*

**lemma** *compute-pdev-upd[code]*:  $pdev\text{-}upd\ (Pdevs\ xs)\ n\ x = Pdevs\ (update\text{-}slist\ n\ x\ xs)$   
*<proof>*

### 11.15 Approximate Total Deviation

**lift-definition** *fold-slist*:: $'a \Rightarrow 'b \Rightarrow 'b) \Rightarrow (nat, 'a::zero)\ slist \Rightarrow 'b \Rightarrow 'b$   
**is**  $\lambda f\ xs\ z.\ fold\ (f\ o\ snd)\ (filter\ (\lambda x.\ snd\ x \neq 0)\ xs)\ z$  *<proof>*

**lemma** *Pdevs-raw-Cons*:  
 $Pdevs\text{-}raw\ ((a, b)\ \#\ xs) = (\lambda i.\ if\ i = a\ then\ b\ else\ Pdevs\text{-}raw\ xs\ i)$   
*<proof>*

**lemma** *zeros-aux*:  $-\ (\lambda i.\ if\ i = a\ then\ b\ else\ Pdevs\text{-}raw\ xs\ i) - ' \{0\} \subseteq$   
 $-\ Pdevs\text{-}raw\ xs - ' \{0\} \cup \{a\}$   
*<proof>*

**lemma** *compute-tdev'[code]*:  
 $tdev'\ p\ (Pdevs\ xs) = fold\text{-}slist\ (\lambda a\ b.\ eucl\text{-}truncate\text{-}up\ p\ (|a| + b))\ xs\ 0$   
*<proof>*

### 11.16 Equality

**lemma** *slist-apply-list-of-slist-eq*:  $slist\text{-}apply\ a\ i = the\text{-}default\ 0\ (map\text{-}of\ (list\text{-}of\text{-}slist\ a)\ i)$   
*<proof>*

**lemma** *compute-equal-pdevs[code]*:  
 $equal\text{-}class.\ equal\ (Pdevs\ a)\ (Pdevs\ b) \longleftrightarrow (list\text{-}of\text{-}slist\ a) = (list\text{-}of\text{-}slist\ b)$   
*<proof>*

### 11.17 From List of Generators

**lift-definition** *slist-of-list*:: $'a::zero\ list \Rightarrow (nat, 'a)\ slist$   
**is**  $\lambda xs.\ rev\ (zip\ [0..<length\ xs]\ xs)$   
*<proof>*

**lemma** *slist-apply-slist-of-list*:  
 $slist\text{-}apply\ (slist\text{-}of\text{-}list\ xs)\ i = (if\ i < length\ xs\ then\ xs\ !\ i\ else\ 0)$   
*<proof>*

**lemma** *compute-pdevs-of-list[code]*:  $pdevs\text{-}of\text{-}list\ xs = Pdevs\ (slist\text{-}of\text{-}list\ xs)$   
*<proof>*

abstraction function which can be used in code equations

**lift-definition** *abs-slist-if*::('a::linorder×'b) list ⇒ ('a, 'b) slist  
**is** λlist. if distinct (map fst list) ∧ rsorted (map fst list) then list else []  
 ⟨proof⟩

**definition** *slist* = *Abs-slist*

**lemma** [*code-post*]: *Abs-slist* = *slist*  
 ⟨proof⟩

**lemma** [*code*]: *slist* = (λxs.  
 (if distinct (map fst xs) ∧ rsorted (map fst xs) then *abs-slist-if* xs else *Code.abort*  
 (STR "" (λ-. *slist* xs)))  
 ⟨proof⟩

**abbreviation** *pdevs* ≡ (λx. *Pdevs* (*slist* x))

**lift-definition** *nlex-slist*::(nat, point) slist ⇒ (nat, point) slist **is**  
 map (λ(i, x). (i, if lex 0 x then - x else x))  
 ⟨proof⟩

**lemma** *Pdevs-raw-map*: f 0 = 0 ⇒ *Pdevs-raw* (map (λ(i, x). (i, f x)) xs) i = f  
 (*Pdevs-raw* xs i)  
 ⟨proof⟩

**lemma** *compute-nlex-pdevs*[*code*]: *nlex-pdevs* (*Pdevs* x) = *Pdevs* (*nlex-slist* x)  
 ⟨proof⟩

**end**

## 12 Optimizations for Code Integer

**theory** *Optimize-Integer*

**imports**

*Complex-Main*

*HOL-Library.Code-Target-Numeral*

**begin**

shallowly embed log and power

**definition** *log2*::int ⇒ int  
**where** *log2* a = floor (log 2 (of-int a))

**context includes** *integer.lifting* **begin**

**lift-definition** *log2-integer* :: integer ⇒ integer  
**is** *log2* :: int ⇒ int  
 ⟨proof⟩

**end**

**lemma** [code]:  $\log_2 (\text{int-of-integer } a) = \text{int-of-integer } (\log_2\text{-integer } a)$   
 ⟨proof⟩

**code-printing**

**constant**  $\log_2\text{-integer} :: \text{integer} \Rightarrow - \rightarrow$   
 (SML) *IntInf.log2*

**definition**  $\text{power-int} :: \text{int} \Rightarrow \text{int} \Rightarrow \text{int}$   
**where**  $\text{power-int } a \ b = a \wedge (\text{nat } b)$

**context includes** *integer.lifting* **begin**

**lift-definition**  $\text{power-integer} :: \text{integer} \Rightarrow \text{integer} \Rightarrow \text{integer}$   
**is**  $\text{power-int} :: \text{int} \Rightarrow \text{int} \Rightarrow \text{int}$   
 ⟨proof⟩

**end**

**code-printing**

**constant**  $\text{power-integer} :: \text{integer} \Rightarrow - \Rightarrow - \rightarrow$   
 (SML) *IntInf.pow ((-), (-))*

**lemma** [code]:  $\text{power-int } (\text{int-of-integer } a) (\text{int-of-integer } b) = \text{int-of-integer } (\text{power-integer } a \ b)$   
 ⟨proof⟩

**end**

## 13 Optimizations for Code Float

**theory** *Optimize-Float*

**imports**

*HOL-Library.Float*  
*Optimize-Integer*

**begin**

**lemma** *compute-bitlen*[code]:  $\text{bitlen } a = (\text{if } a > 0 \text{ then } \log_2 a + 1 \text{ else } 0)$   
 ⟨proof⟩

**lemma** *compute-float-plus*[code]:  $\text{Float } m1 \ e1 + \text{Float } m2 \ e2 =$   
 (if  $m1 = 0$  then  $\text{Float } m2 \ e2$  else if  $m2 = 0$  then  $\text{Float } m1 \ e1$  else  
 if  $e1 \leq e2$  then  $\text{Float } (m1 + m2 * \text{power-int } 2 \ (e2 - e1)) \ e1$   
 else  $\text{Float } (m2 + m1 * \text{power-int } 2 \ (e1 - e2)) \ e2$ )  
 ⟨proof⟩

**lemma** *compute-real-of-float*[code]:

$\text{real-of-float } (\text{Float } m \ e) = (\text{if } e \geq 0 \text{ then } m * 2 \wedge \text{nat } e \text{ else } m / \text{power-int } 2$   
 $(-e))$   
 ⟨proof⟩

```

lemma compute-float-down[code]:
  float-down p (Float m e) =
    (if p + e < 0 then Float (m div power-int 2 (-(p + e))) (-p) else Float m e)
  ⟨proof⟩

lemma compute-lapprox-posrat[code]:
  fixes prec::nat and x y::nat
  shows lapprox-posrat prec x y =
    (let
      l = rat-precision prec x y;
      d = if 0 ≤ l then int x * power-int 2 l div y else int x div power-int 2 (- l)
    in normfloat (Float d (- l)))
  ⟨proof⟩

lemma compute-rapprox-posrat[code]:
  fixes prec x y
  defines l ≡ rat-precision prec x y
  shows rapprox-posrat prec x y = (let
    l = l ;
    (r, s) = if 0 ≤ l then (int x * power-int 2 l, int y) else (int x, int y * power-int
  2 (-l)) ;
    d = r div s ;
    m = r mod s
  in normfloat (Float (d + (if m = 0 ∨ y = 0 then 0 else 1)) (- l)))
  ⟨proof⟩

lemma compute-float-truncate-down[code]:
  float-round-down prec (Float m e) = (let d = bitlen (abs m) - int prec - 1 in
    if 0 < d then let P = power-int 2 d ; n = m div P in Float n (e + d)
    else Float m e)
  ⟨proof⟩

lemma compute-int-floor-fl[code]:
  int-floor-fl (Float m e) = (if 0 ≤ e then m * power-int 2 e else m div (power-int
  2 (-e)))
  ⟨proof⟩

lemma compute-floor-fl[code]:
  floor-fl (Float m e) = (if 0 ≤ e then Float m e else Float (m div (power-int 2
  ((-e)))) 0)
  ⟨proof⟩

```

end

## 14 Target Language debug messages

theory *Print*

```

imports
  HOL-Decision-Procs.Approximation
  Affine-Code
  Show.Show-Instances
  HOL-Library.Monad-Syntax
  Optimize-Float
begin

```

```

hide-const (open) floatarith.Max

```

## 14.1 Printing

Just for debugging purposes

```

definition print::String.literal  $\Rightarrow$  unit where print x = ()

```

```

context includes integer.lifting begin

```

```

end

```

```

code-printing constant print  $\rightarrow$  (SML) TextIO.print

```

## 14.2 Write to File

```

definition file-output::String.literal  $\Rightarrow$  ((String.literal  $\Rightarrow$  unit)  $\Rightarrow$  'a)  $\Rightarrow$  'a where
  file-output - f = f ( $\lambda$ -. ())

```

```

code-printing constant file-output  $\rightarrow$  (SML) (fn s => fn f => File'-Stream.open'-output
  (fn os => f (File'-Stream.output os)) (Path.explode s))

```

## 14.3 Show for Floats

```

definition showsp-float :: float showsp
where

```

```

  showsp-float p x = (
    let m = mantissa x; e = exponent x in
    if e = 0 then showsp-int p m else showsp-int p m o shows-string "*"2^" o
  showsp-int p e)

```

```

lemma show-law-float [show-law-intros]:
  show-law showsp-float r
  <proof>

```

```

lemma showsp-float-append [show-law-simps]:
  showsp-float p r (x @ y) = showsp-float p r x @ y
  <proof>

```

```

<ML>

```

```

derive show float

```

## 14.4 Convert Float to Decimal number

type for decimal floating point numbers (currently just for printing, TODO?  
generalize theory Float for arbitrary base)

**datatype** *float10* = *Float10* (*mantissa10*: int) (*exponent10*: int)  
**notation** *Float10* (**infix** <e> 999)

**partial-function** (*tailrec*) *normalize-float10*  
**where** [*code*]: *normalize-float10* *f* =  
    (*if* *mantissa10* *f* *mod* 10  $\neq$  0  $\vee$  *mantissa10* *f* = 0 *then* *f*  
    *else* *normalize-float10* (*Float10* (*mantissa10* *f* *div* 20) (*exponent10* *f* + 1)))

### 14.4.1 Version that should be easy to prove correct, but slow!

**context includes** *floatarith-syntax* **begin**

**definition** *float-to-float10-approximation* *f* = *the*  
    (*do* {  
        *let* (*x*, *y*) = (*mantissa* *f* \* 1024, *exponent* *f* - 10);  
        *let* *p* = *nat* (*bitlen* (*abs* *x*) + *bitlen* (*abs* *y*) + 80); — FIXME: are there  
        guarantees?  
        *y-log*  $\leftarrow$  *approx* *p* (*Mult* (*Num* (*of-int* *y*))  
            ((*Mult* (*Ln* (*Num* 2))  
            (*Inverse* (*Ln* (*Num* 10)))))) [];  
        *let* *e-fl* = *floor-fl* (*lower* *y-log*);  
        *let* *e* = *int-floor-fl* *e-fl*;  
        *m*  $\leftarrow$  *approx* *p* (*Mult* (*Num* (*of-int* *x*)) (*Powr* (*Num* 10) (*Add*(*Var* 0) (*Minus*  
        (*Num* *e-fl*)))))) [*Some* *y-log*];  
        *let* *ml* = *lower* *m*;  
        *let* *mu* = *upper* *m*;  
        *Some* (*normalize-float10* (*Float10* (*int-floor-fl* *ml*) *e*), *normalize-float10* (*Float10*  
        ( $-$  *int-floor-fl* ( $-$  *mu*)) *e*)  
    })

**end**

**lemma** *compute-float-of*[*code*]: *float-of* (*real-of-float* *f*) = *f* *<proof>*

## 14.5 Trusted, but faster version

TODO: this is the HOL version of the SML-code in Approximation.thy

**lemma** *prod-case-call-mono*[*partial-function-mono*]:  
    *mono-tailrec* ( $\lambda f. (let (d, e) = a in (\lambda y. f (c d e y))) b$ )  
    *<proof>*

**definition** *divmod-int::int*  $\Rightarrow$  *int*  $\Rightarrow$  *int* \* *int*  
**where** *divmod-int* *a* *b* = (*a* *div* *b*, *a* *mod* *b*)

**partial-function** (*tailrec*) *f2f10-frac* **where**  
*f2f10-frac* *c p r digits cnt e* =  
 (if *r* = 0 then (*digits*, *cnt*, 0)  
 else if *p* = 0 then (*digits*, *cnt*, *r*)  
 else (let  
   (*d*, *r*) = *divmod-int* (*r* \* 10) (*power-int* 2 (-*e*))  
   in *f2f10-frac* (*c* ∨ *d* ≠ 0) (if *d* ≠ 0 ∨ *c* then *p* - 1 else *p*) *r*  
   (*digits* \* 10 + *d*) (*cnt* + 1)) *e*)  
**declare** *f2f10-frac.simps*[code]

**definition** *float2-float10::int* ⇒ *bool* ⇒ *int* ⇒ *int* ⇒ (*int* \* *int*) **where**  
*float2-float10* *prec rd m e* = (  
 let  
   (*m*, *e*) = (if *e* < 0 then (*m*,*e*) else (*m* \* *power-int* 2 *e*, 0));  
   *sgn* = *sgn* *m*;  
   *m* = *abs* *m*;  
  
   *round-down* = (*sgn* = 1 ∧ *rd*) ∨ (*sgn* = -1 ∧ ¬ *rd*);  
  
   (*x*, *r*) = *divmod-int* *m* ((*power-int* 2 (-*e*)));  
  
   *p* = ((if *x* = 0 then *prec* else *prec* - (*log2* *x* + 1)) \* 3) *div* 10 + 1;  
  
   (*digits*, *e10*, *r*) = if *p* > 0 then *f2f10-frac* (*x* ≠ 0) *p r 0 0 e* else (0,0,0);  
  
   *digits* = if *round-down* ∨ *r* = 0 then *digits* else *digits* + 1  
  
 in (*sgn* \* (*digits* + *x* \* (*power-int* 10 *e10*)), -*e10*))

**definition** *lfloat10* *r* = (let *f* = *float-of* *r* in case-prod *Float10* (*float2-float10* 20  
 True (*mantissa* *f*) (*exponent* *f*)))

**definition** *ufloat10* *r* = (let *f* = *float-of* *r* in case-prod *Float10* (*float2-float10* 20  
 False (*mantissa* *f*) (*exponent* *f*)))

**partial-function** (*tailrec*) *digits*

**where** [code]: *digits* *m ds* = (if *m* = 0 then *ds* else *digits* (*m* *div* 10) (*m* *mod* 10  
 # *ds*))

**primrec** *showsp-float10* :: *float10* *showsp*

**where**

*showsp-float10* *p* (*Float10* *m e*) = (  
 let  
   *ds* = *digits* (*nat* (*abs* *m*)) [];  
   *d* = *int* (*length* *ds*);  
   *e* = *e* + *d* - 1;  
   *mp* = *take* 1 *ds*;  
   *ms* = *drop* 1 *ds*;  
   *ms* = *rev* (*dropWhile* ((=) 0) (*rev* *ms*));  
   *show-digits* = *shows-list-gen* (*showsp-nat* *p*) "0" "" "" "" ""

in (if  $m < 0$  then shows-string "--" else  $(\lambda x. x)$ ) o  
 show-digits mp o  
 (if  $ms = []$  then  $(\lambda x. x)$  else shows-string "." o show-digits ms) o  
 (if  $e = 0$  then  $(\lambda x. x)$  else shows-string "e" o showsp-int p e))

**lemma** show-law-float10-aux:  
**fixes** m e  
**shows** show-law showsp-float10 (Float10 m e)  
 ⟨proof⟩

**lemma** show-law-float10 [show-law-intros]: show-law showsp-float10 r  
 ⟨proof⟩

**lemma** showsp-float10-append [show-law-simps]:  
 showsp-float10 p r (x @ y) = showsp-float10 p r x @ y  
 ⟨proof⟩

⟨ML⟩

**derive** show float10

**definition** showsp-real p x = showsp-float10 p (lfloat10 x)

**lemma** show-law-real[show-law-intros]: show-law showsp-real x  
 ⟨proof⟩

⟨ML⟩

**derive** show real

## 14.6 gnuplot output

### 14.6.1 vector output of 2D zonotope

**fun** polychain-of-segments::((real × real) × (real × real)) list ⇒ (real × real) list  
**where**

polychain-of-segments [] = []  
 | polychain-of-segments (((x0, y0), z)#segs) = (x0, y0)#z#map snd segs

**definition** shows-segments-of-aform

**where** shows-segments-of-aform a b xs color =  
 shows-list-gen id "" "" "↔" "↔" (map  $(\lambda(x0, y0).$   
 shows-words (map lfloat10 [x0, y0]) o shows-space o shows-string color)  
 (polychain-of-segments (segments-of-aform (prod-of-aforms (xs ! a) (xs ! b))))))

**abbreviation** show-segments-of-aform a b x c ≡ shows-segments-of-aform a b x c  
 ""

**definition** shows-box-of-aforms— box and some further information

**where** shows-box-of-aforms (XS::real aform list) = (let  
 RS = map (Radius' 20) XS;  
 l = map (Inf-aform' 20) XS;

```

    u = map (Sup-aform' 20) XS
    in shows-words
      (l @ u @ RS) o shows-space o
        shows (card (⋃((λx. pdevs-domain (snd x)) ' (set XS))))
      )
abbreviation show-box-of-aforms x ≡ shows-box-of-aforms x ""
definition pdevs-domains ((XS::real aform list)) = (⋃((λx. pdevs-domain (snd
x)) ' (set XS)))
definition generators XS =
  (let
    is = sorted-list-of-set (pdevs-domains XS);
    rs = map (λi. (i, map (λx. pdevs-apply (snd x) i) XS)) is
  in
    (map fst XS, rs))
definition shows-box-of-aforms-hr— human readable
where shows-box-of-aforms-hr XS = (let
  RS = map (Radius' 20) XS;
  l = map (Inf-aform' 20) XS;
  u = map (Sup-aform' 20) XS
  in shows-paren (shows-words l) o shows-string ".." o shows-paren (shows-words
u) o
  shows-string "; devs: " o shows (card (pdevs-domains XS)) o
  shows-string "; tdev: " o shows-paren (shows-words RS)
  )
abbreviation show-box-of-aforms-hr x ≡ shows-box-of-aforms-hr x ""
definition shows-aforms-hr— human readable
where shows-aforms-hr XS = shows (generators XS)
abbreviation show-aform-hr x ≡ shows-aforms-hr x ""
end

```

## 15 Dyadic Rational Representation of Real

```

theory Float-Real
imports
  HOL-Library.Float
  Optimize-Float
begin

code-datatype real-of-float

abbreviation
  float-of-nat :: nat ⇒ float
where

```

*float-of-nat*  $\equiv$  *of-nat*

**abbreviation**

*float-of-int* :: *int*  $\Rightarrow$  *float*

**where**

*float-of-int*  $\equiv$  *of-int*

Collapse nested embeddings

Operations

Undo code setup for *Ratreal*.

**lemma** *of-rat-numeral-eq* [*code-abbrev*]:

*real-of-float* (*numeral w*) = *Ratreal* (*numeral w*)  
<*proof*>

**lemma** *zero-real-code* [*code*]:

0 = *real-of-float* 0  
<*proof*>

**lemma** *one-real-code* [*code*]:

1 = *real-of-float* 1  
<*proof*>

**lemma** [*code-abbrev*]:

(*real-of-float* (*of-int a*) :: *real*) = (*Ratreal* (*Rat.of-int a*) :: *real*)  
<*proof*>

**lemma** [*code-abbrev*]:

*real-of-float* 0  $\equiv$  *Ratreal* 0  
<*proof*>

**lemma** [*code-abbrev*]:

*real-of-float* 1 = *Ratreal* 1  
<*proof*>

**lemmas** *compute-real-of-float*[*code del*]

**lemmas** [*code del*] =

*real-equal-code*  
*real-less-eq-code*  
*real-less-code*  
*real-plus-code*  
*real-times-code*  
*real-uminus-code*  
*real-minus-code*  
*real-inverse-code*  
*real-divide-code*  
*real-floor-code*  
*Float.compute-truncate-down*

*Float.compute-truncate-up*

**lemma** *real-equal-code* [code]:

*HOL.equal* (real-of-float x) (real-of-float y)  $\longleftrightarrow$  *HOL.equal* x y  
(proof)

**abbreviation** *FloatR::int $\Rightarrow$ int $\Rightarrow$ real* **where**

*FloatR a b*  $\equiv$  *real-of-float* (*Float a b*)

**lemma** *real-less-eq-code'* [code]: *real-of-float* x  $\leq$  *real-of-float* y  $\longleftrightarrow$  x  $\leq$  y

**and** *real-less-code'* [code]: *real-of-float* x < *real-of-float* y  $\longleftrightarrow$  x < y

**and** *real-plus-code'* [code]: *real-of-float* x + *real-of-float* y = *real-of-float* (x + y)

**and** *real-times-code'* [code]: *real-of-float* x \* *real-of-float* y = *real-of-float* (x \* y)

**and** *real-uminus-code'* [code]: - *real-of-float* x = *real-of-float* (- x)

**and** *real-minus-code'* [code]: *real-of-float* x - *real-of-float* y = *real-of-float* (x - y)

**and** *real-inverse-code'* [code]: *inverse* (*FloatR a b*) =

(if *FloatR a b* = 2 then *FloatR 1* (-1) else

if a = 1 then *FloatR 1* (- b) else

*Code.abort* (*STR "inverse not of 2"*) ( $\lambda$ -. *inverse* (*FloatR a b*)))

**and** *real-divide-code'* [code]: *FloatR a b* / *FloatR c d* =

(if *FloatR c d* = 2 then if a mod 2 = 0 then *FloatR* (a div 2) b else *FloatR a*  
(b - 1) else

if c = 1 then *FloatR a* (b - d) else

*Code.abort* (*STR "division not by 2"*) ( $\lambda$ -. *FloatR a b* / *FloatR c d*))

**and** *real-floor-code'* [code]: *floor* (*real-of-float* x) = *int-floor-fl* x

**and** *real-abs-code'* [code]: *abs* (*real-of-float* x) = *real-of-float* (*abs* x)

(proof)

**lemma** *compute-round-down*[code]: *round-down prec* (*real-of-float* f) = *real-of-float* (*float-down prec* f)

(proof)

**lemma** *compute-round-up*[code]: *round-up prec* (*real-of-float* f) = *real-of-float* (*float-up prec* f)

(proof)

**lemma** *compute-truncate-down*[code]:

*truncate-down prec* (*real-of-float* f) = *real-of-float* (*float-round-down prec* f)

(proof)

**lemma** *compute-truncate-up*[code]:

*truncate-up prec* (*real-of-float* f) = *real-of-float* (*float-round-up prec* f)

(proof)

**lemma** [code]: *real-divl p* (*real-of-float* x) (*real-of-float* y) = *real-of-float* (*float-divl p x y*)

(proof)

```

lemma [code]: real-divr p (real-of-float x) (real-of-float y) = real-of-float (float-divr p x y)
  <proof>

lemmas [code] = real-of-float-inverse

end

```

## 16 Examples

```

theory Ex-Affine-Approximation
imports
  Affine-Code
  Print
  Float-Real
begin

context includes floatarith-syntax begin

definition rotate-fas =
  [Cos (Rad-of (Var 2)) * Var 0 - Sin (Rad-of (Var 2)) * Var 1,
   Sin (Rad-of (Var 2)) * Var 0 + Cos (Rad-of (Var 2)) * Var 1]

definition rotate-slp = slp-of-fas rotate-fas
definition approx-rotate p t X = approx-slp-outer p 3 rotate-slp X

fun rotate-aform where
  rotate-aform x i = (let r = (((the o (λx. approx-rotate 30 (FloatR 1 (-3)) x))  $\sim$ i)
x) in
  (r ! 0)  $\times_a$  (r ! 1)  $\times_a$  (r ! 2))

value [code] rotate-aform (aforms-of-ivls [2, 1, 45] [3, 5, 45]) 70

definition translate-slp = slp-of-fas [Var 0 + Var 2, Var 1 + Var 2]
fun translatei where translatei x i = (((the o (λx. approx-slp-outer 7 3 translate-slp
x))  $\sim$ i) x)

value translatei (aforms-of-ivls [2, 1, 512] [3, 5, 512]) 50

end

hide-const rotate-fas rotate-slp approx-rotate rotate-aform translate-slp translatei

end

```

## 17 Examples on Proving Inequalities

```

theory Ex-Ineqs

```

```

imports
  Affine-Code
  Print
  Float-Real
begin

definition plotcolors =
  [[(0, 1, "0x000000"),

    [(0, 2, "0xff0000"),
     (1, 2, "0x7f0000")],

    [(0, 3, "0x00ff00"),
     (1, 3, "0x00aa00"),
     (2, 3, "0x005500")],

    [(1, 4, "0x0000ff"),
     (2, 4, "0x0000c0"),
     (3, 4, "0x00007f"),
     (0, 4, "0x00003f")],

    [(0, 5, "0x00ffff"),
     (1, 5, "0x00cccc"),
     (2, 5, "0x009999"),
     (3, 5, "0x006666"),
     (4, 5, "0x003333")],

    [(0, 6, "0xff00ff"),
     (1, 6, "0xd500d5"),
     (2, 6, "0xaa00aa"),
     (3, 6, "0x800080"),
     (4, 6, "0x550055"),
     (5, 6, "0x2a002a")]]

```

```

primrec prove-pos::(nat * nat * string) list ⇒ nat ⇒ nat ⇒
  (nat ⇒ real aform list ⇒ real aform option) ⇒ real aform list list ⇒ bool where
  prove-pos prnt 0 p F X = (let - = if prnt ≠ [] then print (STR "# depth limit
  exceeded[↔]") else () in False)
| prove-pos prnt (Suc i) p F XXS =
  (case XXS of [] ⇒ True | (X#XS) ⇒
  let
    R = F p X;
    - = if prnt ≠ [] then print (String.implode ((shows "# " o shows-box-of-aforms-hr
  X) "[↔]")) else ();
    - = fold (λ(a, b, c) -. print (String.implode (shows-segments-of-aform a b X
  c "[↔]"))) prnt ()
  in

```

```

    if R ≠ None ∧ 0 < Inf-aform' p (the R)
    then let - = if prnt ≠ [] then print (STR "# Success↔") else () in prove-pos
prnt i p F XS
    else let - = if prnt ≠ [] then print (STR "# Split↔") else () in case
split-aforms-largest-uncond X of (a, b) ⇒
    prove-pos prnt i p F (a#b#XS)

```

**definition** *prove-pos-slp prnt p fa i xs = (let slp = slp-of-fas [fa] in prove-pos prnt i p (λp xs.*

*case approx-slp-outer p 1 slp xs of None ⇒ None | Some [x] ⇒ Some x | Some - ⇒ None) xs)*

**experiment begin**

**unbundle** *floatarith-syntax*

The examples below are taken from [http://link.springer.com/chapter/10.1007/978-3-642-38088-4\\_26](http://link.springer.com/chapter/10.1007/978-3-642-38088-4_26), “Formal Verification of Nonlinear Inequalities with Taylor Interval Approximations”, Alexey Solovyev, Thomas C. Hales, NASA Formal Methods 2013, LNCS 7871

**definition** *schwefel =*

$$(5.8806 / 10 \wedge 10) + (\text{Var } 0 - (\text{Var } 1) \wedge_e 2) \wedge_e 2 + (\text{Var } 1 - 1) \wedge_e 2 + (\text{Var } 0 - (\text{Var } 2) \wedge_e 2) \wedge_e 2 + (\text{Var } 2 - 1) \wedge_e 2$$

**lemma** *schwefel:*

$$5.8806 / 10 \wedge 10 + (x0 - (x1)^2)^2 + (x1 - 1)^2 + (x0 - (x2)^2)^2 + (x2 - 1)^2 =$$

*interpret-floatarith schwefel [x0, x1, x2]*  
*<proof>*

**lemma** *prove-pos-slp [] 30 schwefel 100000 [aforms-of-ivls [-10,-10,-10] [10,10,10]]*  
*<proof>*

**definition** *delta6 = (Var 0 \* Var 3 \* (-Var 0 + Var 1 + Var 2 - Var 3 + Var 4 + Var 5) +*

$$\begin{aligned} & \text{Var } 1 * \text{Var } 4 * (\text{Var } 0 - \text{Var } 1 + \text{Var } 2 + \text{Var } 3 - \text{Var } 4 + \text{Var } 5) + \\ & \text{Var } 2 * \text{Var } 5 * (\text{Var } 0 + \text{Var } 1 - \text{Var } 2 + \text{Var } 3 + \text{Var } 4 - \text{Var } 5) \\ & - \text{Var } 1 * \text{Var } 2 * \text{Var } 3 \\ & - \text{Var } 0 * \text{Var } 2 * \text{Var } 4 \\ & - \text{Var } 0 * \text{Var } 1 * \text{Var } 5 \\ & - \text{Var } 3 * \text{Var } 4 * \text{Var } 5) \end{aligned}$$

**schematic-goal** *delta6:*

$$\begin{aligned} & (x0 * x3 * (-x0 + x1 + x2 - x3 + x4 + x5) + \\ & x1 * x4 * (x0 - x1 + x2 + x3 - x4 + x5) + \\ & x2 * x5 * (x0 + x1 - x2 + x3 + x4 - x5) \\ & - x1 * x2 * x3 \\ & - x0 * x2 * x4 \\ & - x0 * x1 * x5) \end{aligned}$$

$-x3 * x4 * x5) = \text{interpret-floatarith delta6 } [x0, x1, x2, x3, x4, x5]$   
 ⟨proof⟩

**lemma** *prove-pos-slp* [] 20 delta6 10000 [*aforms-of-ivls (replicate 6 4) (replicate 6 (FloatR 104045 (-14))))*]  
 ⟨proof⟩

**definition** *caprasse* = (3.1801 + - Var 0 \* (Var 2) ^e 3 + 4 \* Var 1 \* (Var 2) ^e 2 \* Var 3 +  
 4 \* Var 0 \* Var 2 \* (Var 3) ^e 2 + 2 \* Var 1 \* (Var 3) ^e 3 + 4 \* Var 0 \*  
 Var 2 + 4 \* (Var 2) ^e 2 - 10 \* Var 1 \* Var 3 +  
 -10 \* (Var 3) ^e 2 + 2)

**schematic-goal** *caprasse*:

(3.1801 + - xs!0 \* (xs!2) ^ 3 + 4 \* xs!1 \* (xs!2)<sup>2</sup> \* xs!3 +  
 4 \* xs!0 \* xs!2 \* (xs!3)<sup>2</sup> + 2 \* xs!1 \* (xs!3)<sup>3</sup> + 4 \* xs!0 \* xs!2 + 4 \* (xs!2)<sup>2</sup>  
 - 10 \* xs!1 \* xs!3 +  
 -10 \* (xs!3)<sup>2</sup> + 2) = *interpret-floatarith caprasse xs*  
 ⟨proof⟩

**lemma** *prove-pos-slp* [] 20 caprasse 10000 [*aforms-of-ivls (replicate 4 (1/2)) (replicate 4 (1/2))*]  
 ⟨proof⟩

**definition** *magnetism* =

0.25001 + (Var 0) ^e 2 + 2 \* (Var 1) ^e 2 + 2 \* (Var 2) ^e 2 + 2 \* (Var 3) ^e 2  
 + 2 \* (Var 4) ^e 2 + 2 \* (Var 5) ^e 2 +  
 2 \* (Var 6) ^e 2 - Var 0

**schematic-goal** *magnetism*:

0.25001 + (xs!0)<sup>2</sup> + 2 \* (xs!1)<sup>2</sup> + 2 \* (xs!2)<sup>2</sup> + 2 \* (xs!3)<sup>2</sup> + 2 \* (xs!4)<sup>2</sup> +  
 2 \* (xs!5)<sup>2</sup> +  
 2 \* (xs!6)<sup>2</sup> - xs!0 = *interpret-floatarith magnetism xs*  
 ⟨proof⟩

end

end

## 18 Examples: Intersection of Zonotopes with Hyperplanes

**theory** *Ex-Inter*

**imports**

*Intersection*

*Affine-Code*

*Print*

**begin**

## 18.1 Example

**definition** *zono1*::(real\*real\*real) aform

**where** *zono1* = *msum-aform* 53 (*aform-of-ivl* ((0,0,0)::real\*real\*real) ((1,2,0)::real\*real\*real))  
 (0, *pdevs-of-list* [(5, 10, 20)])

**definition** *interzono1*::(real\*real\*real) aform

**where** *interzono1* = *the* (*inter-aform-plane-ortho* 53 *zono1* (0, 0, 1) 3)

10-dimensional zonotope with 50 generators

**definition** *random-zono*::(real\*real\*real\*real\*real\*real\*real\*real\*real\*real) aform

**where**

*random-zono* =  
 (0, *pdevs-of-list*  
 [(5, 9, 27, 12, 23, 3, 9, 10, 18, 2),  
 (26, 4, 14, 15, 11, 7, 27, 5, 21, 16),  
 (10, 17, 11, 27, 13, 14, 27, 14, 25, 23),  
 (7, 6, 5, 30, 14, 10, 2, 1, 18, 25),  
 (17, 5, 28, 6, 10, 22, 5, 18, 8, 11),  
 (5, 7, 14, 14, 5, 11, 5, 17, 1, 22),  
 (3, 6, 11, 20, 28, 13, 12, 10, 2, 23),  
 (3, 1, 26, 15, 1, 3, 25, 23, 6, 18),  
 (30, 8, 24, 16, 8, 20, 27, 25, 21, 17),  
 (30, 4, 8, 12, 8, 4, 22, 27, 23, 2),  
 (24, 21, 19, 15, 24, 22, 16, 15, 25, 6),  
 (20, 4, 1, 24, 2, 9, 19, 4, 21, 17),  
 (1, 12, 13, 7, 8, 8, 2, 11, 28, 6),  
 (26, 25, 19, 8, 6, 26, 27, 17, 27, 25),  
 (8, 8, 1, 4, 6, 2, 28, 13, 18, 28),  
 (14, 14, 12, 7, 26, 19, 9, 25, 21, 17),  
 (25, 14, 30, 17, 24, 17, 7, 25, 25, 5),  
 (27, 21, 29, 22, 30, 10, 13, 15, 23, 19),  
 (27, 5, 10, 4, 11, 12, 3, 20, 8, 23),  
 (29, 11, 19, 12, 2, 28, 30, 27, 27, 1),  
 (18, 7, 23, 1, 14, 6, 23, 22, 23, 19),  
 (7, 17, 3, 15, 28, 15, 9, 16, 23, 7),  
 (18, 25, 10, 13, 17, 14, 3, 24, 14, 7),  
 (28, 13, 6, 27, 8, 14, 7, 14, 5, 24),  
 (17, 5, 18, 9, 2, 11, 24, 17, 3, 2),  
 (13, 17, 15, 30, 27, 29, 29, 16, 27, 13),  
 (25, 21, 21, 17, 19, 3, 26, 27, 26, 2),  
 (5, 16, 21, 18, 23, 1, 19, 13, 10, 2),  
 (8, 27, 14, 16, 2, 11, 27, 27, 29, 2),  
 (10, 22, 1, 23, 2, 22, 17, 22, 19, 15),  
 (16, 8, 9, 27, 19, 23, 24, 30, 1, 3),  
 (2, 20, 9, 12, 19, 21, 30, 9, 19, 13),  
 (23, 21, 28, 26, 27, 17, 22, 9, 17, 13),  
 (24, 1, 19, 19, 28, 21, 4, 8, 10, 20),  
 (27, 19, 7, 23, 11, 30, 12, 10, 27, 20),  
 (4, 3, 23, 21, 17, 13, 25, 8, 13, 26),

(11, 25, 7, 2, 27, 10, 15, 14, 17, 23),  
 (25, 27, 28, 15, 11, 4, 30, 25, 16, 1),  
 (27, 26, 11, 21, 9, 14, 15, 11, 30, 18),  
 (3, 19, 13, 17, 13, 9, 22, 4, 20, 30),  
 (21, 26, 20, 8, 19, 1, 22, 9, 28, 15),  
 (22, 12, 5, 25, 29, 27, 13, 9, 2, 10),  
 (9, 24, 30, 6, 23, 13, 18, 15, 30, 20),  
 (13, 5, 7, 6, 21, 30, 7, 22, 26, 15),  
 (9, 3, 3, 1, 29, 16, 10, 2, 21, 25),  
 (3, 14, 22, 18, 21, 15, 16, 22, 27, 26),  
 (16, 25, 16, 22, 27, 18, 4, 15, 9, 21),  
 (30, 23, 29, 24, 20, 14, 15, 25, 3, 22),  
 (6, 18, 17, 14, 19, 25, 9, 22, 7, 26),  
 (24, 7, 30, 27, 9, 2, 8, 23, 24, 1)]

10-dimensional zonotope with 100 generators

**definition** *random-zono2::(real\*real\*real\*real\*real\*real\*real\*real\*real\*real) aform*  
**where**

*random-zono2 =*  
 (0, *pdeus-of-list*  
 [(17, 28, 12, 10, 18, 3, 14, 27, 21, 22),  
 (7, 17, 16, 26, 25, 4, 12, 20, 18, 28),  
 (11, 8, 30, 20, 11, 17, 8, 13, 28, 18),  
 (18, 20, 26, 12, 25, 24, 23, 24, 22, 2),  
 (14, 27, 20, 12, 16, 7, 21, 5, 5, 20),  
 (4, 27, 8, 19, 11, 14, 9, 25, 8, 11),  
 (14, 29, 12, 28, 29, 21, 20, 6, 18, 6),  
 (20, 25, 8, 19, 30, 1, 21, 18, 7, 18),  
 (5, 6, 7, 25, 30, 2, 19, 7, 13, 19),  
 (11, 15, 16, 13, 17, 2, 9, 10, 29, 17),  
 (29, 1, 30, 6, 6, 27, 19, 24, 11, 12),  
 (27, 30, 8, 11, 30, 2, 19, 25, 5, 27),  
 (3, 26, 16, 18, 12, 11, 4, 8, 2, 4),  
 (16, 7, 11, 23, 29, 30, 22, 22, 5, 21),  
 (6, 12, 28, 24, 12, 4, 11, 27, 6, 13),  
 (30, 13, 16, 29, 22, 7, 10, 12, 3, 17),  
 (26, 22, 6, 4, 8, 11, 29, 23, 13, 17),  
 (30, 23, 20, 3, 4, 28, 25, 26, 25, 17),  
 (30, 27, 8, 20, 4, 1, 9, 6, 23, 16),  
 (10, 27, 15, 17, 14, 9, 19, 22, 7, 19),  
 (29, 5, 14, 23, 23, 29, 13, 19, 1, 14),  
 (7, 30, 29, 23, 27, 2, 3, 8, 10, 14),  
 (7, 10, 10, 10, 30, 5, 7, 29, 7, 23),  
 (2, 1, 11, 19, 23, 9, 14, 16, 13, 25),  
 (5, 10, 2, 24, 16, 21, 21, 30, 14, 12),  
 (25, 19, 9, 29, 21, 29, 10, 4, 19, 25),  
 (30, 18, 3, 8, 9, 6, 13, 17, 1, 19),  
 (7, 30, 18, 16, 25, 15, 10, 17, 18, 12),  
 (21, 10, 13, 2, 12, 25, 25, 2, 27, 19),
 ]

(17, 7, 18, 22, 24, 10, 8, 3, 26, 3),  
 (3, 22, 19, 23, 30, 20, 1, 25, 18, 27),  
 (8, 2, 15, 23, 28, 18, 4, 20, 7, 7),  
 (4, 8, 29, 22, 20, 8, 18, 29, 13, 2),  
 (20, 5, 8, 8, 20, 17, 2, 17, 29, 2),  
 (4, 27, 8, 20, 18, 2, 18, 21, 6, 16),  
 (8, 11, 24, 10, 20, 6, 16, 17, 13, 23),  
 (22, 8, 21, 25, 17, 13, 9, 21, 4, 19),  
 (18, 23, 22, 22, 2, 15, 25, 18, 30, 7),  
 (2, 5, 5, 21, 18, 6, 27, 5, 30, 6),  
 (28, 4, 17, 15, 27, 7, 27, 5, 9, 19),  
 (8, 7, 4, 28, 22, 1, 28, 10, 14, 8),  
 (6, 7, 30, 26, 5, 15, 21, 28, 1, 21),  
 (20, 11, 8, 18, 17, 1, 24, 11, 22, 6),  
 (23, 5, 29, 8, 10, 8, 28, 6, 5, 3),  
 (8, 8, 17, 23, 23, 10, 9, 27, 10, 20),  
 (3, 7, 29, 26, 1, 16, 1, 30, 5, 4),  
 (23, 22, 17, 2, 15, 16, 17, 7, 20, 13),  
 (1, 14, 3, 21, 14, 5, 24, 29, 5, 4),  
 (6, 14, 26, 18, 29, 7, 2, 19, 19, 24),  
 (24, 24, 10, 14, 22, 6, 17, 13, 3, 6),  
 (5, 17, 2, 30, 26, 6, 21, 13, 11, 7),  
 (11, 20, 15, 29, 20, 2, 23, 6, 28, 9),  
 (27, 10, 3, 16, 21, 22, 8, 5, 19, 14),  
 (21, 25, 23, 24, 7, 3, 30, 8, 21, 19),  
 (10, 9, 17, 15, 14, 2, 5, 19, 28, 9),  
 (1, 4, 3, 1, 22, 27, 15, 26, 1, 9),  
 (8, 19, 18, 12, 26, 18, 1, 5, 19, 16),  
 (6, 30, 11, 8, 22, 1, 24, 10, 30, 5),  
 (10, 11, 12, 14, 24, 27, 22, 8, 11, 27),  
 (8, 29, 17, 19, 20, 17, 4, 9, 3, 1),  
 (17, 15, 1, 17, 22, 30, 1, 22, 3, 23),  
 (1, 11, 15, 8, 6, 22, 4, 24, 18, 3),  
 (23, 21, 24, 2, 17, 14, 14, 7, 18, 27),  
 (30, 3, 25, 17, 25, 3, 5, 8, 4, 24),  
 (4, 29, 30, 7, 14, 27, 25, 11, 18, 19),  
 (2, 26, 15, 13, 16, 8, 7, 11, 21, 23),  
 (9, 22, 28, 29, 18, 9, 22, 25, 26, 20),  
 (21, 15, 29, 18, 24, 29, 20, 17, 2, 29),  
 (12, 17, 11, 9, 4, 6, 2, 4, 22, 25),  
 (17, 9, 9, 19, 3, 8, 6, 22, 12, 15),  
 (28, 19, 25, 28, 1, 15, 8, 7, 6, 4),  
 (17, 17, 22, 7, 1, 21, 25, 23, 22, 14),  
 (19, 1, 7, 3, 11, 9, 7, 24, 2, 4),  
 (17, 27, 18, 29, 8, 2, 17, 17, 13, 30),  
 (8, 14, 14, 11, 26, 20, 28, 25, 13, 17),  
 (10, 17, 7, 26, 24, 4, 10, 17, 2, 15),  
 (21, 9, 29, 7, 13, 10, 13, 17, 2, 2),  
 (16, 10, 18, 27, 26, 26, 3, 30, 14, 1),



(2, 8, 16, 16, 8, 21, 19, 22, 10, 28, 7, 11, 21, 3, 18, 30, 15, 21, 3, 16),  
 (7, 8, 8, 19, 21, 13, 7, 7, 29, 16, 10, 5, 21, 28, 16, 19, 11, 21, 13, 23),  
 (26, 7, 26, 14, 9, 18, 10, 24, 20, 2, 5, 1, 15, 21, 29, 24, 27, 20, 24, 16),  
 (4, 14, 10, 8, 22, 20, 1, 4, 1, 25, 17, 15, 16, 2, 30, 10, 29, 11, 29, 17),  
 (21, 12, 16, 3, 28, 7, 3, 8, 12, 19, 24, 12, 6, 14, 18, 16, 24, 12, 21, 2),  
 (7, 30, 25, 20, 23, 14, 17, 17, 18, 27, 24, 17, 3, 19, 7, 10, 19, 14, 24, 6),  
 (12, 16, 26, 29, 27, 1, 18, 3, 14, 4, 27, 28, 24, 4, 18, 25, 25, 7, 12, 30),  
 (19, 30, 30, 15, 16, 4, 12, 16, 27, 24, 22, 28, 13, 14, 22, 17, 18, 21, 7,  
 19),  
 (9, 9, 23, 5, 1, 23, 9, 26, 23, 13, 19, 14, 29, 27, 23, 25, 2, 13, 18, 11),  
 (12, 8, 20, 14, 14, 23, 24, 11, 8, 6, 25, 27, 28, 3, 4, 15, 1, 22, 19, 22),  
 (19, 23, 28, 13, 2, 5, 17, 1, 17, 19, 30, 7, 6, 29, 7, 12, 11, 20, 30, 23),  
 (27, 10, 21, 19, 24, 17, 10, 22, 22, 26, 2, 25, 8, 1, 5, 9, 22, 18, 28, 6),  
 (9, 22, 9, 13, 20, 10, 6, 23, 7, 10, 29, 5, 28, 30, 22, 23, 8, 10, 14, 11),  
 (14, 16, 20, 4, 25, 1, 10, 20, 13, 29, 17, 14, 21, 30, 21, 16, 10, 19, 6, 16),  
 (25, 3, 6, 20, 18, 23, 3, 12, 14, 9, 2, 2, 30, 19, 12, 29, 23, 20, 29, 22),  
 (20, 15, 11, 23, 5, 17, 13, 2, 4, 20, 16, 7, 7, 24, 7, 10, 13, 22, 9, 15),  
 (8, 12, 30, 22, 11, 26, 25, 16, 27, 2, 9, 15, 15, 13, 30, 21, 4, 3, 1, 5),  
 (23, 26, 23, 29, 26, 24, 8, 15, 22, 5, 26, 6, 2, 3, 17, 5, 14, 25, 28, 10),  
 (20, 28, 25, 20, 9, 22, 1, 5, 24, 8, 10, 19, 3, 26, 21, 1, 13, 15, 3, 3),  
 (9, 24, 1, 5, 22, 11, 11, 22, 25, 25, 16, 25, 24, 28, 15, 26, 22, 1, 23, 9),  
 (13, 1, 11, 16, 6, 12, 11, 8, 29, 21, 23, 21, 21, 20, 5, 26, 2, 23, 2, 16),  
 (12, 13, 5, 24, 25, 19, 26, 4, 17, 5, 18, 6, 2, 29, 21, 3, 10, 20, 7, 5),  
 (26, 10, 13, 17, 29, 22, 3, 3, 28, 11, 5, 8, 11, 11, 17, 27, 19, 17, 23, 8),  
 (2, 4, 11, 17, 18, 23, 14, 22, 4, 29, 2, 29, 25, 3, 4, 13, 2, 14, 5, 15),  
 (12, 6, 16, 4, 25, 22, 29, 21, 2, 27, 17, 4, 11, 22, 2, 2, 5, 9, 28, 8),  
 (3, 26, 17, 3, 29, 17, 16, 24, 10, 9, 16, 4, 23, 14, 10, 12, 16, 28, 28, 28),  
 (7, 15, 28, 6, 25, 24, 11, 26, 22, 3, 28, 17, 10, 17, 19, 12, 20, 18, 29, 23),  
 (24, 7, 7, 26, 17, 23, 19, 29, 1, 28, 11, 30, 23, 25, 30, 2, 6, 21, 1, 16),  
 (6, 27, 22, 25, 9, 1, 16, 2, 12, 30, 23, 19, 12, 29, 20, 16, 16, 16, 6, 21),  
 (25, 12, 5, 28, 19, 9, 25, 12, 10, 27, 10, 26, 27, 15, 2, 4, 23, 12, 20, 27)]

**fun** *random-inter1* **where**  
*random-inter1* () =  
*the (inter-aform-plane-ortho 53 random-zono (1, 15, 26, 8, 15, 23, 5, 14, 8,*  
*8) 12)*

**fun** *random-inter2* **where**  
*random-inter2* () =  
*the (inter-aform-plane-ortho 53 random-zono2 (13, 23, 22, 30, 27, 19, 17, 11,*  
*24, 29) 12)*

**fun** *random-inter3* **where**  
*random-inter3* () =  
*the (inter-aform-plane-ortho 53 random-zono3*  
*(7, 10, 24, 12, 6, 14, 10, 14, 23, 13, 25, 27, 20, 2, 1, 9, 4, 17, 28, 19)*  
*12)*

(ML)

Timings

$\langle ML \rangle$

**end**

**theory** *Affine-Arithmetic*

**imports**

*Affine-Code*

*Intersection*

*Straight-Line-Program*

*Ex-Affine-Approximation*

*Ex-Ineqs*

*Ex-Inter*

**begin**

**end**

## References

- [1] L. H. De Figueiredo and J. Stolfi. Affine arithmetic: concepts and applications. *Numerical Algorithms*, 37(1-4):147–158, 2004.
- [2] A. Girard. Reachability of uncertain linear systems using zonotopes. In *Hybrid systems: computation and control*, pages 291–305. Springer, 2005.
- [3] F. Immler. A verified algorithm for geometric zonotope/hyperplane intersection. In *Proceedings of the 2015 Conference on Certified Programs and Proofs*, CPP '15, pages 129–136, New York, NY, USA, 2015. ACM.