

# Verified Quadratic Virtual Substitution for Real Arithmetic

Matias Scharager, Katherine Cordwell, Stefan Mitsch and André Platzer

February 6, 2026

## Abstract

This paper presents a formally verified quantifier elimination (QE) algorithm for first-order real arithmetic by linear and quadratic virtual substitution (VS) in Isabelle/HOL [4, 5]. The Tarski-Seidenberg theorem established that the first-order logic of real arithmetic is decidable by QE. However, in practice, QE algorithms are highly complicated and often combine multiple methods for performance. VS is a practically successful method for QE that targets formulas with low-degree polynomials. To our knowledge, this is the first work to formalize VS for quadratic real arithmetic including inequalities. The proofs necessitate various contributions to the existing multivariate polynomial libraries in Isabelle/HOL. Our framework is modularized and easily expandable (to facilitate integrating future optimizations), and could serve as a basis for developing practical general-purpose QE algorithms. Further, as our formalization is designed with practicality in mind, we export our development to SML and test the resulting code on 378 benchmarks from the literature, comparing to Redlog, Z3, Wolfram Engine, and SMT-RAT. This identified inconsistencies in some tools, underscoring the significance of a verified approach for the intricacies of real arithmetic.

## Contents

<b>1</b>	<b>Related Works</b>	<b>3</b>
<b>2</b>	<b>QE lemmas</b>	<b>4</b>
2.1	Useful Definitions/Setting Up . . . . .	4
2.2	Quadratic polynomial properties . . . . .	5
2.3	Continuity Properties . . . . .	6
2.4	Negative Infinity (Limit) Properties . . . . .	6
2.5	Infinitesimal and Continuity Properties . . . . .	7
2.6	Some Casework . . . . .	8
2.7	More Continuity Properties . . . . .	9
2.8	Setting up and Helper Lemmas . . . . .	10

2.8.1	The les_qe lemma . . . . .	10
2.8.2	equiv_lemma . . . . .	11
2.8.3	The eq_qe lemma . . . . .	13
2.8.4	The qe_forwards lemma . . . . .	15
2.8.5	Some Cases and Misc . . . . .	26
2.8.6	The qe_backwards lemma . . . . .	29
2.9	General QE lemmas . . . . .	33
<b>3</b>	<b>Multivariate Polynomials Extension</b>	<b>41</b>
3.1	Definition Lifting . . . . .	41
3.2	Lemmas with Monomial and Monomials . . . . .	44
3.3	Simplification Lemmas for Const 0 and Const 1 . . . . .	46
3.4	Coefficient Lemmas . . . . .	46
3.5	Miscellaneous . . . . .	46
3.5.1	Keys and vars . . . . .	47
3.6	Degree Lemmas . . . . .	47
3.7	Lemmas on treating a multivariate polynomial as univariate . . . . .	48
3.8	Summation Lemmas . . . . .	49
3.9	Additional Lemmas with Vars . . . . .	50
3.10	Insertion Lemmas . . . . .	51
3.11	Putting Things Together . . . . .	52
3.11.1	More Degree Lemmas . . . . .	52
3.11.2	More isolate_variable_sparse lemmas . . . . .	54
3.11.3	More Miscellaneous . . . . .	55
3.11.4	Yet more Degree Lemmas . . . . .	55
<b>4</b>	<b>Atoms</b>	<b>56</b>
4.1	Definition . . . . .	57
4.2	Evaluation . . . . .	58
4.3	Useful Properties . . . . .	61
4.4	Some eval results . . . . .	61
<b>5</b>	<b>Debruijn Indices Formulation</b>	<b>62</b>
5.1	Lift and Lower Functions . . . . .	62
5.2	Swapping Indices . . . . .	70
<b>6</b>	<b>Optimizations</b>	<b>74</b>
6.1	Simplify Constants . . . . .	74
6.2	Group Quantifiers . . . . .	75
6.3	Clear Quantifiers . . . . .	75
6.4	Push Forall . . . . .	76
6.5	Unpower . . . . .	77
6.6	Optimization Proofs . . . . .	78

<b>7</b>	<b>Algorithms</b>	<b>81</b>
7.1	Equality VS Helper Functions . . . . .	81
7.2	General VS Helper Functions . . . . .	85
7.3	VS Algorithms . . . . .	86
7.4	DNF . . . . .	90
7.5	Repeat QE multiple times . . . . .	92
7.6	Heuristic Algorithms . . . . .	92
7.7	Top-Level Algorithms . . . . .	98
<b>8</b>	<b>Equality VS Proofs</b>	<b>100</b>
8.1	Linear Case . . . . .	100
8.2	Quadratic Case . . . . .	103
8.3	Lemmas of the elimVar function . . . . .	109
8.4	Overall LuckyFind Proofs . . . . .	110
8.5	Overall Equality VS Proofs . . . . .	111
<b>9</b>	<b>General VS Proofs</b>	<b>112</b>
9.1	Univariate Atoms . . . . .	112
9.2	Negative Infinity . . . . .	117
9.3	Infinitesimals . . . . .	120
9.4	Overall General VS Proofs . . . . .	125
<b>10</b>	<b>QE Algorithm Proofs</b>	<b>133</b>
10.1	DNF . . . . .	133
10.2	Recursive QE . . . . .	134
10.3	Heuristic Proofs . . . . .	135
10.4	Top-Level Algorithm Proofs . . . . .	137

## 1 Related Works

There has already been some work on formally verified VS: Nipkow [2] formally verified a VS procedure for *linear* equations and inequalities. The building blocks of  $\text{FOL}_{\mathbb{R}}$  formulas, or “atoms,” in Nipkow’s work only allow for linear polynomials  $\sum_i a_i x_i \sim c$ , where  $\sim \in \{=, <\}$ , the  $x_i$ ’s are quantified variables and  $c$  and the  $a_i$ ’s are real numbers. These restrictions ensure that linear QE can always be performed, and they also simplify the substitution procedure and associated proofs. Nipkow additionally provides a generic framework that can be applied to several different kinds of atoms (each new atom requires implementing several new code theorems in order to create an exportable algorithm). While this is an excellent theoretical framework—we utilize several similar constructs in our formulation—we create an independent formalization that is specific to general  $\text{FOL}_{\mathbb{R}}$  formulas, as our main focus is to provide an efficient algorithm in this domain. Specializing to one type of atom allows us to implement several optimizations, such as our

modified DNF algorithm, which would be unwieldy to develop in a generic setting.

Chaieb [1] extends Nipkow’s work to quadratic equalities. His formalizations are not publicly available, but he generously provided us with the code. While this was helpful for reference, we chose to build on a newer Isabelle/HOL polynomial library, and we focus on VS as an exportable standalone procedure, whereas Chaieb intrinsically links VS with an auxiliary QE procedure.

We also use the Logical Foundations of Cyber-Physical Systems textbook[3] for easy reference for the VS algorithm.

## 2 QE lemmas

**theory** *QE*

**imports** *Polynomials.MPoly-Type-Univariate*

*Polynomials.Polynomials Polynomials.MPoly-Type-Class-FMap*

*HOL-Library.Quadratic-Discriminant*

**begin**

### 2.1 Useful Definitions/Setting Up

**definition** *sign:: real Polynomial.poly  $\Rightarrow$  real  $\Rightarrow$  int*

**where** *sign p x  $\equiv$  (if poly p x = 0 then 0 else (if poly p x > 0 then 1 else -1))*

**definition** *sign-num:: real  $\Rightarrow$  int*

**where** *sign-num x  $\equiv$  (if x = 0 then 0 else (if x > 0 then 1 else -1))*

**definition** *root-list:: real Polynomial.poly  $\Rightarrow$  real set*

**where** *root-list p  $\equiv$  ({(x::real). poly p x = 0}::real set)*

**definition** *root-set:: (real  $\times$  real  $\times$  real) set  $\Rightarrow$  real set*

**where** *root-set les  $\equiv$  ({(x::real). ( $\exists$  (a, b, c)  $\in$  les.  $a*x^2 + b*x + c = 0$ )})::real set)*

**definition** *sorted-root-list-set:: (real  $\times$  real  $\times$  real) set  $\Rightarrow$  real list*

**where** *sorted-root-list-set p  $\equiv$  sorted-list-of-set (root-set p)*

**definition** *nonzero-root-set:: (real  $\times$  real  $\times$  real) set  $\Rightarrow$  real set*

**where** *nonzero-root-set les  $\equiv$  ({(x::real). ( $\exists$  (a, b, c)  $\in$  les. ( $a \neq 0 \vee b \neq 0$ )  $\wedge$   $a*x^2 + b*x + c = 0$ )})::real set)*

**definition** *sorted-nonzero-root-list-set:: (real  $\times$  real  $\times$  real) set  $\Rightarrow$  real list*

**where** *sorted-nonzero-root-list-set p  $\equiv$  sorted-list-of-set (nonzero-root-set p)*

**lemma** *sorted-list-prop:*

**fixes** *l::real list*

**fixes**  $x::\text{real}$   
**assumes**  $\text{sorted}: \text{sorted } l$   
**assumes**  $\text{lengt}: \text{length } l > 0$   
**assumes**  $\text{xgt}: x > l ! 0$   
**assumes**  $\text{xlt}: x \leq l ! (\text{length } l - 1)$   
**shows**  $\exists n. (n+1) < (\text{length } l) \wedge x \geq l ! n \wedge x \leq l ! (n + 1)$   
 $\langle \text{proof} \rangle$

## 2.2 Quadratic polynomial properties

**lemma** *quadratic-poly-eval*:  
**fixes**  $a \ b \ c::\text{real}$   
**fixes**  $x::\text{real}$   
**shows**  $\text{poly } [:c, b, a:] x = a*x^2 + b*x + c$   
 $\langle \text{proof} \rangle$

**lemma** *poly-roots-set-same*:  
**fixes**  $a \ b \ c:: \text{real}$   
**shows**  $\{(x::\text{real}). a * x^2 + b * x + c = 0\} = \{x. \text{poly } [:c, b, a:] x = 0\}$   
 $\langle \text{proof} \rangle$

**lemma** *root-set-finite*:  
**assumes**  $\text{fin}: \text{finite } \text{les}$   
**assumes**  $\text{nex}: \neg(\exists (a, b, c) \in \text{les}. a = 0 \wedge b = 0 \wedge c = 0)$   
**shows**  $\text{finite } (\text{root-set } \text{les})$   
 $\langle \text{proof} \rangle$

**lemma** *nonzero-root-set-finite*:  
**assumes**  $\text{fin}: \text{finite } \text{les}$   
**shows**  $\text{finite } (\text{nonzero-root-set } \text{les})$   
 $\langle \text{proof} \rangle$

**lemma** *discriminant-lemma*:  
**fixes**  $a \ b \ c \ r::\text{real}$   
**assumes**  $\text{aneq}: a \neq 0$   
**assumes**  $\text{beq}: b = 2 * a * r$   
**assumes**  $\text{root}: a * r^2 - 2 * a * r * r + c = 0$   
**shows**  $\forall x. a * x^2 + b * x + c = 0 \longleftrightarrow x = -r$   
 $\langle \text{proof} \rangle$

**lemma** *changes-sign*:  
**fixes**  $p:: \text{real Polynomial.poly}$   
**shows**  $\forall x::\text{real}. \forall y::\text{real}. ((\text{sign } p \ x \neq \text{sign } p \ y \wedge x < y) \longrightarrow (\exists c \in (\text{root-list } p). x \leq c \wedge c \leq y))$   
 $\langle \text{proof} \rangle$

**lemma** *changes-sign-var*:

**fixes**  $a\ b\ c\ x\ y::\text{real}$   
**shows**  $((\text{sign-num } (a*x^2 + b*x + c) \neq \text{sign-num } (a*y^2 + b*y + c) \wedge x < y) \implies (\exists q. (a*q^2 + b*q + c = 0 \wedge x \leq q \wedge q \leq y)))$   
 $\langle\text{proof}\rangle$

## 2.3 Continuity Properties

**lemma** *continuity-lem-eq0*:

**fixes**  $p::\text{real}$   
**shows**  $r < p \implies \forall x \in \{r <..p\}. a * x^2 + b * x + c = 0 \implies (a = 0 \wedge b = 0 \wedge c = 0)$   
 $\langle\text{proof}\rangle$

**lemma** *continuity-lem-lt0*:

**fixes**  $r::\text{real}$   
**fixes**  $a\ b\ c::\text{real}$   
**shows**  $\text{poly } [:c, b, a:]\ r < 0 \implies \exists y' > r. \forall x \in \{r <..y'\}. \text{poly } [:c, b, a:]\ x < 0$   
 $\langle\text{proof}\rangle$

**lemma** *continuity-lem-gt0*:

**fixes**  $r::\text{real}$   
**fixes**  $a\ b\ c::\text{real}$   
**shows**  $\text{poly } [:c, b, a:]\ r > 0 \implies \exists y' > r. \forall x \in \{r <..y'\}. \text{poly } [:c, b, a:]\ x > 0$   
 $\langle\text{proof}\rangle$

**lemma** *continuity-lem-lt0-expanded*:

**fixes**  $r::\text{real}$   
**fixes**  $a\ b\ c::\text{real}$   
**shows**  $a*r^2 + b*r + c < 0 \implies \exists y' > r. \forall x \in \{r <..y'\}. a*x^2 + b*x + c < 0$   
 $\langle\text{proof}\rangle$

**lemma** *continuity-lem-gt0-expanded*:

**fixes**  $r::\text{real}$   
**fixes**  $a\ b\ c::\text{real}$   
**fixes**  $k::\text{real}$   
**assumes**  $kgt: k > r$   
**shows**  $a*r^2 + b*r + c > 0 \implies \exists x \in \{r <..k\}. a*x^2 + b*x + c > 0$   
 $\langle\text{proof}\rangle$

## 2.4 Negative Infinity (Limit) Properties

**lemma** *ysq-dom-y*:

**fixes**  $b::\text{real}$   
**fixes**  $c::\text{real}$   
**shows**  $\exists (w::\text{real}). \forall (y::\text{real}). (y < w \implies y^2 > b*y)$   
 $\langle\text{proof}\rangle$

**lemma** *ysq-dom-y-plus-coeff*:  
**fixes** *b:: real*  
**fixes** *c:: real*  
**shows**  $\exists (w::real). \forall (y::real). (y < w \longrightarrow y^2 > b*y + c)$   
 $\langle proof \rangle$

**lemma** *ysq-dom-y-plus-coeff-2*:  
**fixes** *b:: real*  
**fixes** *c:: real*  
**shows**  $\exists (w::real). \forall (y::real). (y > w \longrightarrow y^2 > b*y + c)$   
 $\langle proof \rangle$

**lemma** *neg-lc-dom-quad*:  
**fixes** *a:: real*  
**fixes** *b:: real*  
**fixes** *c:: real*  
**assumes** *alt: a < 0*  
**shows**  $\exists (w::real). \forall (y::real). (y > w \longrightarrow a*y^2 + b*y + c < 0)$   
 $\langle proof \rangle$

**lemma** *pos-lc-dom-quad*:  
**fixes** *a:: real*  
**fixes** *b:: real*  
**fixes** *c:: real*  
**assumes** *alt: a > 0*  
**shows**  $\exists (w::real). \forall (y::real). (y > w \longrightarrow a*y^2 + b*y + c > 0)$   
 $\langle proof \rangle$

## 2.5 Infinitesimal and Continuity Properties

**lemma** *les-qe-inf-helper*:  
**fixes** *q:: real*  
**shows**  $(\forall (d, e, f) \in set\ les. \exists y' > q. \forall x \in \{q <..y'\}. d * x^2 + e * x + f < 0) \implies$   
 $(\exists y' > q. (\forall (d, e, f) \in set\ les. \forall x \in \{q <..y'\}. d * x^2 + e * x + f < 0))$   
 $\langle proof \rangle$

**lemma** *have-inbetween-point-les*:  
**fixes** *r::real*  
**assumes**  $(\forall (d, e, f) \in set\ les. \exists y' > r. \forall x \in \{r <..y'\}. d * x^2 + e * x + f < 0)$   
**shows**  $(\exists x. (\forall (a, b, c) \in set\ les. a * x^2 + b * x + c < 0))$   
 $\langle proof \rangle$

**lemma** *one-root-a-gt0*:  
**fixes** *a b c r:: real*  
**shows**  $\bigwedge y'. b = 2 * a * r \implies$   
 $\neg a < 0 \implies$   
 $a * r^2 - 2 * a * r * r + c = 0 \implies$   
 $- r < y' \implies$

$\exists x \in \{-r <..y'\}. \neg a * x^2 + 2 * a * r * x + c < 0$   
 <proof>

**lemma** *leq-qe-inf-helper*:

**fixes** *q*:: real

**shows**  $(\forall (d, e, f) \in \text{set leq}. \exists y' > q. \forall x \in \{q <..y'\}. d * x^2 + e * x + f \leq 0) \implies$   
 $(\exists y' > q. (\forall (d, e, f) \in \text{set leq}. \forall x \in \{q <..y'\}. d * x^2 + e * x + f \leq 0))$

<proof>

**lemma** *neq-qe-inf-helper*:

**fixes** *q*:: real

**shows**  $(\forall (d, e, f) \in \text{set neq}. \exists y' > q. \forall x \in \{q <..y'\}. d * x^2 + e * x + f \neq 0) \implies$   
 $(\exists y' > q. (\forall (d, e, f) \in \text{set neq}. \forall x \in \{q <..y'\}. d * x^2 + e * x + f \neq 0))$

<proof>

## 2.6 Some Casework

**lemma** *quadratic-shape1a*:

**fixes** *a b c x y*::real

**assumes** *agt*:  $a > 0$

**assumes** *xyroots*:  $x < y \wedge a * x^2 + b * x + c = 0 \wedge a * y^2 + b * y + c = 0$

**shows**  $\bigwedge z. (z > x \wedge z < y \implies a * z^2 + b * z + c < 0)$

<proof>

**lemma** *quadratic-shape1b*:

**fixes** *a b c x y*::real

**assumes** *agt*:  $a > 0$

**assumes** *xy-roots*:  $x < y \wedge a * x^2 + b * x + c = 0 \wedge a * y^2 + b * y + c = 0$

**shows**  $\bigwedge z. (z > y \implies a * z^2 + b * z + c > 0)$

<proof>

**lemma** *quadratic-shape2a*:

**fixes** *a b c x y*::real

**assumes**  $a < 0$

**assumes**  $x < y \wedge a * x^2 + b * x + c = 0 \wedge a * y^2 + b * y + c = 0$

**shows**  $\bigwedge z. (z > x \wedge z < y \implies a * z^2 + b * z + c > 0)$

<proof>

**lemma** *quadratic-shape2b*:

**fixes** *a b c x y*::real

**assumes**  $a < 0$

**assumes**  $x < y \wedge a * x^2 + b * x + c = 0 \wedge a * y^2 + b * y + c = 0$

**shows**  $\bigwedge z. (z > y \implies a * z^2 + b * z + c < 0)$

<proof>

**lemma** *case-d1*:

**fixes** *a b c r*::real

**shows**  $b < 2 * a * r \implies$

$a * r^2 - b * r + c = 0 \implies$

$\exists y' > -r. \forall x \in \{-r <..y'\}. a * x^2 + b * x + c < 0$   
 <proof>

**lemma case-d4:**

**fixes**  $a\ b\ c\ r::real$

**shows**  $\bigwedge y'. b \neq 2 * a * r \implies$

$\neg b < 2 * a * r \implies$

$a * r^2 - b * r + c = 0 \implies$

$-r < y' \implies \exists x \in \{-r <..y'\}. \neg a * x^2 + b * x + c < 0$

<proof>

**lemma one-root-a-lt0:**

**fixes**  $a\ b\ c\ r\ y'::real$

**assumes**  $alt: a < 0$

**assumes**  $beg: b = 2 * a * r$

**assumes**  $root: a * r^2 - 2 * a * r * r + c = 0$

**shows**  $\exists y' > -r. \forall x \in \{-r <..y'\}. a * x^2 + 2 * a * r * x + c < 0$

<proof>

**lemma one-root-a-lt0-var:**

**fixes**  $a\ b\ c\ r\ y'::real$

**assumes**  $alt: a < 0$

**assumes**  $beg: b = 2 * a * r$

**assumes**  $root: a * r^2 - 2 * a * r * r + c = 0$

**shows**  $\exists y' > -r. \forall x \in \{-r <..y'\}. a * x^2 + 2 * a * r * x + c \leq 0$

<proof>

## 2.7 More Continuity Properties

**lemma continuity-lem-gt0-expanded-var:**

**fixes**  $r::real$

**fixes**  $a\ b\ c::real$

**fixes**  $k::real$

**assumes**  $kgt: k > r$

**shows**  $a * r^2 + b * r + c > 0 \implies$

$\exists x \in \{r <..k\}. a * x^2 + b * x + c \geq 0$

<proof>

**lemma continuity-lem-lt0-expanded-var:**

**fixes**  $r::real$

**fixes**  $a\ b\ c::real$

**shows**  $a * r^2 + b * r + c < 0 \implies$

$\exists y' > r. \forall x \in \{r <..y'\}. a * x^2 + b * x + c \leq 0$

<proof>

**lemma nonzcoeffs:**

**fixes**  $a\ b\ c\ r::real$

**shows**  $a \neq 0 \vee b \neq 0 \vee c \neq 0 \implies \exists y' > r. \forall x \in \{r <..y'\}. a * x^2 + b * x + c \neq 0$

*<proof>*

**lemma** *infzeros* :

**fixes** *y::real*

**assumes**  $\forall x::real < (y::real). a * x^2 + b * x + c = 0$

**shows**  $a = 0 \wedge b = 0 \wedge c = 0$

*<proof>*

**lemma** *have-inbetween-point-leq*:

**fixes** *r::real*

**assumes**  $(\forall ((d::real), (e::real), (f::real)) \in \text{set leq}. \exists y' > r. \forall x \in \{r <..y'\}. d * x^2 + e * x + f \leq 0)$

**shows**  $(\exists x. (\forall (a, b, c) \in \text{set leq}. a * x^2 + b * x + c \leq 0))$

*<proof>*

**lemma** *have-inbetween-point-neq*:

**fixes** *r::real*

**assumes**  $(\forall ((d::real), (e::real), (f::real)) \in \text{set neq}. \exists y' > r. \forall x \in \{r <..y'\}. d * x^2 + e * x + f \neq 0)$

**shows**  $(\exists x. (\forall (a, b, c) \in \text{set neq}. a * x^2 + b * x + c \neq 0))$

*<proof>*

## 2.8 Setting up and Helper Lemmas

### 2.8.1 The `les_qe` lemma

**lemma** *les-qe-forward* :

**shows**  $((\forall (a, b, c) \in \text{set les}. \exists x. \forall y < x. a * y^2 + b * y + c < 0) \vee$

$(\exists (a', b', c') \in \text{set les}.$

$a' = 0 \wedge$

$b' \neq 0 \wedge$

$(\forall (d, e, f) \in \text{set les}.$

$\exists y' > -(c' / b'). \forall x \in \{-(c' / b') <..y'\}. d * x^2 + e * x + f < 0) \vee$

$a' \neq 0 \wedge$

$4 * a' * c' \leq b'^2 \wedge$

$(\forall (d, e, f) \in \text{set les}.$

$\exists y' > (\text{sqrt}(b'^2 - 4 * a' * c') - b') / (2 * a').$

$\forall x \in \{(\text{sqrt}(b'^2 - 4 * a' * c') - b') / (2 * a') <..y'\}.$

$d * x^2 + e * x + f < 0) \vee$

$(\forall (d, e, f) \in \text{set les}.$

$\exists y' > (-b' - \text{sqrt}(b'^2 - 4 * a' * c')) / (2 * a').$

$\forall x \in \{(-b' - \text{sqrt}(b'^2 - 4 * a' * c')) / (2 * a') <..y'\}.$

$d * x^2 + e * x + f < 0)))) \implies ((\exists x. (\forall (a, b, c) \in \text{set les}. a * x^2$

$+ b * x + c < 0))$

*<proof>*

**lemma** *les-qe-backward* :

**shows**  $(\exists x. (\forall (a, b, c) \in \text{set les. } a * x^2 + b * x + c < 0)) \implies$   
 $((\forall (a, b, c) \in \text{set les. } \exists x. \forall y < x. a * y^2 + b * y + c < 0) \vee$   
 $(\exists (a', b', c') \in \text{set les.}$   
 $a' = 0 \wedge$   
 $b' \neq 0 \wedge$   
 $(\forall (d, e, f) \in \text{set les.}$   
 $\exists y' > - (c' / b'). \forall x \in \{- (c' / b') <..y'\}. d * x^2 + e * x + f < 0) \vee$   
 $a' \neq 0 \wedge$   
 $4 * a' * c' \leq b'^2 \wedge$   
 $((\forall (d, e, f) \in \text{set les.}$   
 $\exists y' > (\text{sqrt } (b'^2 - 4 * a' * c') - b') / (2 * a').$   
 $\forall x \in \{(\text{sqrt } (b'^2 - 4 * a' * c') - b') / (2 * a') <..y'\}.$   
 $d * x^2 + e * x + f < 0) \vee$   
 $(\forall (d, e, f) \in \text{set les.}$   
 $\exists y' > (- b' - \text{sqrt } (b'^2 - 4 * a' * c')) / (2 * a').$   
 $\forall x \in \{(- b' - \text{sqrt } (b'^2 - 4 * a' * c')) / (2 * a') <..y'\}.$   
 $d * x^2 + e * x + f < 0))))))$

*<proof>*

**lemma** *les-qe* :

**shows**  $(\exists x. (\forall (a, b, c) \in \text{set les. } a * x^2 + b * x + c < 0)) =$   
 $((\forall (a, b, c) \in \text{set les. } \exists x. \forall y < x. a * y^2 + b * y + c < 0) \vee$   
 $(\exists (a', b', c') \in \text{set les.}$   
 $a' = 0 \wedge$   
 $b' \neq 0 \wedge$   
 $(\forall (d, e, f) \in \text{set les.}$   
 $\exists y' > - (c' / b'). \forall x \in \{- (c' / b') <..y'\}. d * x^2 + e * x + f < 0) \vee$   
 $a' \neq 0 \wedge$   
 $4 * a' * c' \leq b'^2 \wedge$   
 $((\forall (d, e, f) \in \text{set les.}$   
 $\exists y' > (\text{sqrt } (b'^2 - 4 * a' * c') - b') / (2 * a').$   
 $\forall x \in \{(\text{sqrt } (b'^2 - 4 * a' * c') - b') / (2 * a') <..y'\}.$   
 $d * x^2 + e * x + f < 0) \vee$   
 $(\forall (d, e, f) \in \text{set les.}$   
 $\exists y' > (- b' - \text{sqrt } (b'^2 - 4 * a' * c')) / (2 * a').$   
 $\forall x \in \{(- b' - \text{sqrt } (b'^2 - 4 * a' * c')) / (2 * a') <..y'\}.$   
 $d * x^2 + e * x + f < 0))))))$

*<proof>*

## 2.8.2 equiv\_lemma

**lemma** *equiv-lemma*:

**assumes** *big-asm*:  $(\exists (a', b', c') \in \text{set eq.}$   
 $(a' = 0 \wedge b' \neq 0) \wedge$   
 $(\forall (d, e, f) \in \text{set eq. } d * (- c' / b')^2 + e * (- c' / b') + f = 0) \wedge$

$$\begin{aligned}
& (\forall (d, e, f) \in \text{set les. } d * (-c' / b')^2 + e * (-c' / b') + f < 0) \vee \\
& (\exists (a', b', c') \in \text{set eq.} \\
& \quad a' \neq 0 \wedge \\
& \quad -b'^2 + 4 * a' * c' \leq 0 \wedge \\
& \quad ((\forall (d, e, f) \in \text{set eq.} \\
& \quad \quad d * ((-b' + 1 * \text{sqrt}(b'^2 - 4 * a' * c')) / (2 * a'))^2 + \\
& \quad \quad e * ((-b' + 1 * \text{sqrt}(b'^2 - 4 * a' * c')) / (2 * a')) + \\
& \quad \quad f = \\
& \quad \quad 0) \wedge \\
& \quad (\forall (d, e, f) \in \text{set les.} \\
& \quad \quad d * ((-b' + 1 * \text{sqrt}(b'^2 - 4 * a' * c')) / (2 * a'))^2 + \\
& \quad \quad e * ((-b' + 1 * \text{sqrt}(b'^2 - 4 * a' * c')) / (2 * a')) + \\
& \quad \quad f \\
& \quad \quad < 0))) \vee
\end{aligned}$$

$$\begin{aligned}
& (\exists (a', b', c') \in \text{set eq. } a' \neq 0 \wedge \\
& \quad -b'^2 + 4 * a' * c' \leq 0 \wedge (\forall (d, e, f) \in \text{set eq.} \\
& \quad \quad d * ((-b' + -1 * \text{sqrt}(b'^2 - 4 * a' * c')) / (2 * a'))^2 + \\
& \quad \quad e * ((-b' + -1 * \text{sqrt}(b'^2 - 4 * a' * c')) / (2 * a')) + \\
& \quad \quad f = \\
& \quad \quad 0) \wedge \\
& \quad (\forall (d, e, f) \in \text{set les.} \\
& \quad \quad d * ((-b' + -1 * \text{sqrt}(b'^2 - 4 * a' * c')) / (2 * a'))^2 + \\
& \quad \quad e * ((-b' + -1 * \text{sqrt}(b'^2 - 4 * a' * c')) / (2 * a')) + \\
& \quad \quad f \\
& \quad \quad < 0)) \vee \\
& ((\forall (d, e, f) \in \text{set eq. } d = 0 \wedge e = 0 \wedge f = 0) \wedge \\
& (\exists x. \forall (a, b, c) \in \text{set les. } a * x^2 + b * x + c < 0))
\end{aligned}$$

**shows**  $((\exists (a', b', c') \in \text{set eq.}$

$$\begin{aligned}
& (a' = 0 \wedge b' \neq 0) \wedge \\
& (\forall (d, e, f) \in \text{set eq. } d * (-c' / b')^2 + e * (-c' / b') + f = 0) \wedge \\
& (\forall (d, e, f) \in \text{set les. } d * (-c' / b')^2 + e * (-c' / b') + f < 0) \vee \\
& a' \neq 0 \wedge \\
& -b'^2 + 4 * a' * c' \leq 0 \wedge \\
& ((\forall (d, e, f) \in \text{set eq.} \\
& \quad d * ((-b' + 1 * \text{sqrt}(b'^2 - 4 * a' * c')) / (2 * a'))^2 + \\
& \quad e * ((-b' + 1 * \text{sqrt}(b'^2 - 4 * a' * c')) / (2 * a')) + \\
& \quad f = \\
& \quad 0) \wedge \\
& (\forall (d, e, f) \in \text{set les.} \\
& \quad d * ((-b' + 1 * \text{sqrt}(b'^2 - 4 * a' * c')) / (2 * a'))^2 + \\
& \quad e * ((-b' + 1 * \text{sqrt}(b'^2 - 4 * a' * c')) / (2 * a')) + \\
& \quad f \\
& \quad < 0) \vee \\
& (\forall (d, e, f) \in \text{set eq.} \\
& \quad d * ((-b' + -1 * \text{sqrt}(b'^2 - 4 * a' * c')) / (2 * a'))^2 + \\
& \quad e * ((-b' + -1 * \text{sqrt}(b'^2 - 4 * a' * c')) / (2 * a')) + \\
& \quad f = \\
& \quad 0) \wedge
\end{aligned}$$

$$\begin{aligned}
& (\forall (d, e, f) \in \text{set les.} \\
& \quad d * ((-b' + -1 * \text{sqrt}(b'^2 - 4 * a' * c')) / (2 * a'))^2 + \\
& \quad e * ((-b' + -1 * \text{sqrt}(b'^2 - 4 * a' * c')) / (2 * a')) + \\
& \quad f \\
& \quad < 0))) \vee \\
& (\forall (d, e, f) \in \text{set eq. } d = 0 \wedge e = 0 \wedge f = 0) \wedge \\
& (\exists x. \forall (a, b, c) \in \text{set les. } a * x^2 + b * x + c < 0)) \\
\langle \text{proof} \rangle
\end{aligned}$$

### 2.8.3 The eq\_qe lemma

**lemma** *eq-qe-forwards*:

**shows**  $(\exists x. (\forall (a, b, c) \in \text{set eq. } a * x^2 + b * x + c = 0) \wedge$   
 $(\forall (a, b, c) \in \text{set les. } a * x^2 + b * x + c < 0)) \implies$   
 $((\exists (a', b', c') \in \text{set eq.}$   
 $(a' = 0 \wedge b' \neq 0) \wedge$   
 $(\forall (d, e, f) \in \text{set eq. } d * (-c' / b')^2 + e * (-c' / b') + f = 0) \wedge$   
 $(\forall (d, e, f) \in \text{set les. } d * (-c' / b')^2 + e * (-c' / b') + f < 0) \vee$   
 $a' \neq 0 \wedge$   
 $-b'^2 + 4 * a' * c' \leq 0 \wedge$   
 $(\forall (d, e, f) \in \text{set eq.}$   
 $d * ((-b' + 1 * \text{sqrt}(b'^2 - 4 * a' * c')) / (2 * a'))^2 +$   
 $e * ((-b' + 1 * \text{sqrt}(b'^2 - 4 * a' * c')) / (2 * a')) +$   
 $f =$   
 $0) \wedge$   
 $(\forall (d, e, f) \in \text{set les.}$   
 $d * ((-b' + 1 * \text{sqrt}(b'^2 - 4 * a' * c')) / (2 * a'))^2 +$   
 $e * ((-b' + 1 * \text{sqrt}(b'^2 - 4 * a' * c')) / (2 * a')) +$   
 $f$   
 $< 0) \vee$   
 $(\forall (d, e, f) \in \text{set eq.}$   
 $d * ((-b' + -1 * \text{sqrt}(b'^2 - 4 * a' * c')) / (2 * a'))^2 +$   
 $e * ((-b' + -1 * \text{sqrt}(b'^2 - 4 * a' * c')) / (2 * a')) +$   
 $f =$   
 $0) \wedge$   
 $(\forall (d, e, f) \in \text{set les.}$   
 $d * ((-b' + -1 * \text{sqrt}(b'^2 - 4 * a' * c')) / (2 * a'))^2 +$   
 $e * ((-b' + -1 * \text{sqrt}(b'^2 - 4 * a' * c')) / (2 * a')) +$   
 $f$   
 $< 0))) \vee$   
 $(\forall (d, e, f) \in \text{set eq. } d = 0 \wedge e = 0 \wedge f = 0) \wedge$   
 $(\exists x. \forall (a, b, c) \in \text{set les. } a * x^2 + b * x + c < 0))$   
 $\langle \text{proof} \rangle$

**lemma** *eq-qe-backwards*:  $((\exists (a', b', c') \in \text{set eq.}$

$$\begin{aligned}
& (a' = 0 \wedge b' \neq 0) \wedge \\
& (\forall (d, e, f) \in \text{set eq. } d * (-c' / b')^2 + e * (-c' / b') + f = 0) \wedge \\
& (\forall (d, e, f) \in \text{set les. } d * (-c' / b')^2 + e * (-c' / b') + f < 0) \vee \\
& a' \neq 0 \wedge
\end{aligned}$$

$$\begin{aligned}
& -b'^2 + 4 * a' * c' \leq 0 \wedge \\
& ((\forall (d, e, f) \in \text{set eq.} \\
& \quad d * ((-b' + 1 * \text{sqrt}(b'^2 - 4 * a' * c')) / (2 * a'))^2 + \\
& \quad e * ((-b' + 1 * \text{sqrt}(b'^2 - 4 * a' * c')) / (2 * a')) + \\
& \quad f = \\
& \quad 0) \wedge \\
& (\forall (d, e, f) \in \text{set les.} \\
& \quad d * ((-b' + 1 * \text{sqrt}(b'^2 - 4 * a' * c')) / (2 * a'))^2 + \\
& \quad e * ((-b' + 1 * \text{sqrt}(b'^2 - 4 * a' * c')) / (2 * a')) + \\
& \quad f \\
& \quad < 0) \vee \\
& (\forall (d, e, f) \in \text{set eq.} \\
& \quad d * ((-b' + -1 * \text{sqrt}(b'^2 - 4 * a' * c')) / (2 * a'))^2 + \\
& \quad e * ((-b' + -1 * \text{sqrt}(b'^2 - 4 * a' * c')) / (2 * a')) + \\
& \quad f = \\
& \quad 0) \wedge \\
& (\forall (d, e, f) \in \text{set les.} \\
& \quad d * ((-b' + -1 * \text{sqrt}(b'^2 - 4 * a' * c')) / (2 * a'))^2 + \\
& \quad e * ((-b' + -1 * \text{sqrt}(b'^2 - 4 * a' * c')) / (2 * a')) + \\
& \quad f \\
& \quad < 0))) \vee \\
& (\forall (d, e, f) \in \text{set eq. } d = 0 \wedge e = 0 \wedge f = 0) \wedge \\
& (\exists x. \forall (a, b, c) \in \text{set les. } a * x^2 + b * x + c < 0) \implies \\
& (\exists x. ((\forall (a, b, c) \in \text{set eq. } a * x^2 + b * x + c = 0) \wedge \\
& \quad (\forall (a, b, c) \in \text{set les. } a * x^2 + b * x + c < 0)))
\end{aligned}$$

\langle proof \rangle

**lemma** *eq-qe* :  $(\exists x. ((\forall (a, b, c) \in \text{set eq. } a * x^2 + b * x + c = 0) \wedge$   
 $(\forall (a, b, c) \in \text{set les. } a * x^2 + b * x + c < 0))) =$   
 $((\exists (a', b', c') \in \text{set eq.}$   
 $(a' = 0 \wedge b' \neq 0) \wedge$   
 $(\forall (d, e, f) \in \text{set eq. } d * (-c' / b')^2 + e * (-c' / b') + f = 0) \wedge$   
 $(\forall (d, e, f) \in \text{set les. } d * (-c' / b')^2 + e * (-c' / b') + f < 0) \vee$   
 $a' \neq 0 \wedge$   
 $-b'^2 + 4 * a' * c' \leq 0 \wedge$   
 $((\forall (d, e, f) \in \text{set eq.}$   
 $\quad d * ((-b' + 1 * \text{sqrt}(b'^2 - 4 * a' * c')) / (2 * a'))^2 +$   
 $\quad e * ((-b' + 1 * \text{sqrt}(b'^2 - 4 * a' * c')) / (2 * a')) +$   
 $\quad f =$   
 $\quad 0) \wedge$   
 $(\forall (d, e, f) \in \text{set les.}$   
 $\quad d * ((-b' + 1 * \text{sqrt}(b'^2 - 4 * a' * c')) / (2 * a'))^2 +$   
 $\quad e * ((-b' + 1 * \text{sqrt}(b'^2 - 4 * a' * c')) / (2 * a')) +$   
 $\quad f$   
 $\quad < 0) \vee$   
 $(\forall (d, e, f) \in \text{set eq.}$   
 $\quad d * ((-b' + -1 * \text{sqrt}(b'^2 - 4 * a' * c')) / (2 * a'))^2 +$

$$\begin{aligned}
& e * ((- b' + - 1 * \text{sqrt}(b'^2 - 4 * a' * c')) / (2 * a')) + \\
& f = \\
& 0) \wedge \\
& (\forall (d, e, f) \in \text{set les.} \\
& d * ((- b' + - 1 * \text{sqrt}(b'^2 - 4 * a' * c')) / (2 * a'))^2 + \\
& e * ((- b' + - 1 * \text{sqrt}(b'^2 - 4 * a' * c')) / (2 * a')) + \\
& f \\
& < 0))) \vee \\
& (\forall (d, e, f) \in \text{set eq. } d = 0 \wedge e = 0 \wedge f = 0) \wedge \\
& (\exists x. \forall (a, b, c) \in \text{set les. } a * x^2 + b * x + c < 0)) \\
\langle \text{proof} \rangle
\end{aligned}$$

## 2.8.4 The `qe_forwards` lemma

**lemma** *qe-forwards-helper-gen*:

**fixes** *r*:: real

**assumes** *f8*:  $\neg(\exists((a'::\text{real}), (b'::\text{real}), (c'::\text{real})) \in \text{set } c.$

$((a' \neq 0 \vee b' \neq 0) \wedge a' * r^2 + b' * r + c' = 0) \wedge$

$((\forall (d, e, f) \in \text{set } a. d * r^2 + e * r + f = 0) \wedge$

$(\forall (d, e, f) \in \text{set } b. d * r^2 + e * r + f < 0) \wedge$

$(\forall (d, e, f) \in \text{set } c. d * r^2 + e * r + f \leq 0) \wedge$

$(\forall (d, e, f) \in \text{set } d. d * r^2 + e * r + f \neq 0)))$

**assumes** *allegset*:  $\forall x. (\forall (d, e, f) \in \text{set } a. d * x^2 + e * x + f = 0)$

**assumes** *f5*:  $\neg(\exists(a', b', c') \in \text{set } b.$

$(a' = 0 \wedge b' \neq 0) \wedge$

$(\forall (d, e, f) \in \text{set } a.$

$\exists y' > -c' / b'. \forall x \in \{-c' / b' < .. y'\}. d * x^2 + e * x + f = 0) \wedge$

$(\forall (d, e, f) \in \text{set } b.$

$\exists y' > -c' / b'. \forall x \in \{-c' / b' < .. y'\}. d * x^2 + e * x + f < 0) \wedge$

$(\forall (d, e, f) \in \text{set } c.$

$\exists y' > -c' / b'. \forall x \in \{-c' / b' < .. y'\}. d * x^2 + e * x + f \leq 0) \wedge$

$(\forall (d, e, f) \in \text{set } d.$

$\exists y' > -c' / b'. \forall x \in \{-c' / b' < .. y'\}. d * x^2 + e * x + f \neq 0))$

**assumes** *f6*:  $\neg(\exists(a', b', c') \in \text{set } b. a' \neq 0 \wedge$

$-b'^2 + 4 * a' * c' \leq 0 \wedge$

$(\forall (d, e, f) \in \text{set } a.$

$\exists y' > (-b' + 1 * \text{sqrt}(b'^2 - 4 * a' * c')) / (2 * a').$

$\forall x \in \{(-b' + 1 * \text{sqrt}(b'^2 - 4 * a' * c')) / (2 * a') < .. y'\}.$

$d * x^2 + e * x + f = 0) \wedge$

$(\forall (d, e, f) \in \text{set } b.$

$\exists y' > (-b' + 1 * \text{sqrt}(b'^2 - 4 * a' * c')) / (2 * a').$

$\forall x \in \{(-b' + 1 * \text{sqrt}(b'^2 - 4 * a' * c')) / (2 * a') < .. y'\}.$

$d * x^2 + e * x + f < 0) \wedge$

$(\forall (d, e, f) \in \text{set } c.$

$\exists y' > (-b' + 1 * \text{sqrt}(b'^2 - 4 * a' * c')) / (2 * a').$

$\forall x \in \{(-b' + 1 * \text{sqrt}(b'^2 - 4 * a' * c')) / (2 * a') < .. y'\}.$

$d * x^2 + e * x + f \leq 0) \wedge$

$(\forall (d, e, f) \in \text{set } d.$

$\exists y' > (-b' + 1 * \text{sqrt}(b'^2 - 4 * a' * c')) / (2 * a').$

$$\forall x \in \{(-b' + 1 * \text{sqrt}(b^2 - 4 * a' * c')) / (2 * a') < ..y'\}.$$

$$d * x^2 + e * x + f \neq 0))$$

**assumes f7:**  $\neg(\exists(a', b', c') \in \text{set } b. a' \neq 0 \wedge$

$$- b^2 + 4 * a' * c' \leq 0 \wedge (\forall(d, e, f) \in \text{set } a.$$

$$\exists y' > (-b' + -1 * \text{sqrt}(b^2 - 4 * a' * c')) / (2 * a').$$

$$\forall x \in \{(-b' + -1 * \text{sqrt}(b^2 - 4 * a' * c')) / (2 * a') < ..y'\}.$$

$$d * x^2 + e * x + f = 0) \wedge$$

$$(\forall(d, e, f) \in \text{set } b.$$

$$\exists y' > (-b' + -1 * \text{sqrt}(b^2 - 4 * a' * c')) / (2 * a').$$

$$\forall x \in \{(-b' + -1 * \text{sqrt}(b^2 - 4 * a' * c')) / (2 * a') < ..y'\}.$$

$$d * x^2 + e * x + f < 0) \wedge$$

$$(\forall(d, e, f) \in \text{set } c.$$

$$\exists y' > (-b' + -1 * \text{sqrt}(b^2 - 4 * a' * c')) / (2 * a').$$

$$\forall x \in \{(-b' + -1 * \text{sqrt}(b^2 - 4 * a' * c')) / (2 * a') < ..y'\}.$$

$$d * x^2 + e * x + f \leq 0) \wedge$$

$$(\forall(d, e, f) \in \text{set } d.$$

$$\exists y' > (-b' + -1 * \text{sqrt}(b^2 - 4 * a' * c')) / (2 * a').$$

$$\forall x \in \{(-b' + -1 * \text{sqrt}(b^2 - 4 * a' * c')) / (2 * a') < ..y'\}.$$

$$d * x^2 + e * x + f \neq 0))$$

**assumes f10:**  $\neg(\exists(a', b', c') \in \text{set } d.$

$$(a' = 0 \wedge b' \neq 0) \wedge$$

$$(\forall(d, e, f) \in \text{set } a.$$

$$\exists y' > -c' / b'. \forall x \in \{-c' / b' < ..y'\}. d * x^2 + e * x + f = 0) \wedge$$

$$(\forall(d, e, f) \in \text{set } b.$$

$$\exists y' > -c' / b'. \forall x \in \{-c' / b' < ..y'\}. d * x^2 + e * x + f < 0) \wedge$$

$$(\forall(d, e, f) \in \text{set } c.$$

$$\exists y' > -c' / b'. \forall x \in \{-c' / b' < ..y'\}. d * x^2 + e * x + f \leq 0) \wedge$$

$$(\forall(d, e, f) \in \text{set } d.$$

$$\exists y' > -c' / b'. \forall x \in \{-c' / b' < ..y'\}. d * x^2 + e * x + f \neq 0))$$

**assumes f11:**  $\neg(\exists(a', b', c') \in \text{set } d.$

$$a' \neq 0 \wedge$$

$$- b^2 + 4 * a' * c' \leq 0 \wedge$$

$$((\forall(d, e, f) \in \text{set } a.$$

$$\exists y' > (-b' + 1 * \text{sqrt}(b^2 - 4 * a' * c')) / (2 * a').$$

$$\forall x \in \{(-b' + 1 * \text{sqrt}(b^2 - 4 * a' * c')) / (2 * a') < ..y'\}.$$

$$d * x^2 + e * x + f = 0) \wedge$$

$$(\forall(d, e, f) \in \text{set } b.$$

$$\exists y' > (-b' + 1 * \text{sqrt}(b^2 - 4 * a' * c')) / (2 * a').$$

$$\forall x \in \{(-b' + 1 * \text{sqrt}(b^2 - 4 * a' * c')) / (2 * a') < ..y'\}.$$

$$d * x^2 + e * x + f < 0) \wedge$$

$$(\forall(d, e, f) \in \text{set } c.$$

$$\exists y' > (-b' + 1 * \text{sqrt}(b^2 - 4 * a' * c')) / (2 * a').$$

$$\forall x \in \{(-b' + 1 * \text{sqrt}(b^2 - 4 * a' * c')) / (2 * a') < ..y'\}.$$

$$d * x^2 + e * x + f \leq 0) \wedge$$

$$(\forall(d, e, f) \in \text{set } d.$$

$$\exists y' > (-b' + 1 * \text{sqrt}(b^2 - 4 * a' * c')) / (2 * a').$$

$$\forall x \in \{(-b' + 1 * \text{sqrt}(b^2 - 4 * a' * c')) / (2 * a') < ..y'\}.$$

$$d * x^2 + e * x + f \neq 0))$$

**assumes f12:**  $\neg(\exists(a', b', c') \in \text{set } d.$

$$\begin{aligned}
& a' \neq 0 \wedge \\
& -b'^2 + 4 * a' * c' \leq 0 \wedge (\forall (d, e, f) \in \text{set } a. \\
& \quad \exists y' > (-b' + -1 * \text{sqrt}(b'^2 - 4 * a' * c')) / (2 * a'). \\
& \quad \forall x \in \{(-b' + -1 * \text{sqrt}(b'^2 - 4 * a' * c')) / (2 * a') <..y'\}. \\
& \quad d * x^2 + e * x + f = 0) \wedge \\
& (\forall (d, e, f) \in \text{set } b. \\
& \quad \exists y' > (-b' + -1 * \text{sqrt}(b'^2 - 4 * a' * c')) / (2 * a'). \\
& \quad \forall x \in \{(-b' + -1 * \text{sqrt}(b'^2 - 4 * a' * c')) / (2 * a') <..y'\}. \\
& \quad d * x^2 + e * x + f < 0) \wedge \\
& (\forall (d, e, f) \in \text{set } c. \\
& \quad \exists y' > (-b' + -1 * \text{sqrt}(b'^2 - 4 * a' * c')) / (2 * a'). \\
& \quad \forall x \in \{(-b' + -1 * \text{sqrt}(b'^2 - 4 * a' * c')) / (2 * a') <..y'\}. \\
& \quad d * x^2 + e * x + f \leq 0) \wedge \\
& (\forall (d, e, f) \in \text{set } d. \\
& \quad \exists y' > (-b' + -1 * \text{sqrt}(b'^2 - 4 * a' * c')) / (2 * a'). \\
& \quad \forall x \in \{(-b' + -1 * \text{sqrt}(b'^2 - 4 * a' * c')) / (2 * a') <..y'\}. \\
& \quad d * x^2 + e * x + f \neq 0))
\end{aligned}$$

**shows**  $\neg(\exists (a', b', c') \in \text{set } c.$

$$\begin{aligned}
& ((a' \neq 0 \vee b' \neq 0) \wedge a' * r^2 + b' * r + c' = 0) \wedge \\
& (\forall (d, e, f) \in \text{set } a. \\
& \quad \exists y' > r. \forall x \in \{r <..y'\}. d * x^2 + e * x + f = 0) \wedge \\
& (\forall (d, e, f) \in \text{set } b. \\
& \quad \exists y' > r. \forall x \in \{r <..y'\}. d * x^2 + e * x + f < 0) \wedge \\
& (\forall (d, e, f) \in \text{set } c. \\
& \quad \exists y' > r. \forall x \in \{r <..y'\}. d * x^2 + e * x + f \leq 0) \wedge \\
& (\forall (d, e, f) \in \text{set } d. \\
& \quad \exists y' > r. \forall x \in \{r <..y'\}. d * x^2 + e * x + f \neq 0))
\end{aligned}$$

*\langle proof \rangle*

**lemma** *qe-forwards-helper-lin*:

**assumes** *allegset*:  $\forall x. (\forall (d, e, f) \in \text{set } a. d * x^2 + e * x + f = 0)$

**assumes** *f5*:  $\neg(\exists (a', b', c') \in \text{set } b.$

$$\begin{aligned}
& (a' = 0 \wedge b' \neq 0) \wedge \\
& (\forall (d, e, f) \in \text{set } a. \\
& \quad \exists y' > -c' / b'. \forall x \in \{-c' / b' <..y'\}. d * x^2 + e * x + f = 0) \wedge \\
& (\forall (d, e, f) \in \text{set } b. \\
& \quad \exists y' > -c' / b'. \forall x \in \{-c' / b' <..y'\}. d * x^2 + e * x + f < 0) \wedge \\
& (\forall (d, e, f) \in \text{set } c. \\
& \quad \exists y' > -c' / b'. \forall x \in \{-c' / b' <..y'\}. d * x^2 + e * x + f \leq 0) \wedge \\
& (\forall (d, e, f) \in \text{set } d. \\
& \quad \exists y' > -c' / b'. \forall x \in \{-c' / b' <..y'\}. d * x^2 + e * x + f \neq 0))
\end{aligned}$$

**assumes** *f6*:  $\neg(\exists (a', b', c') \in \text{set } b. a' \neq 0 \wedge$

$$\begin{aligned}
& -b'^2 + 4 * a' * c' \leq 0 \wedge \\
& ((\forall (d, e, f) \in \text{set } a. \\
& \quad \exists y' > (-b' + 1 * \text{sqrt}(b'^2 - 4 * a' * c')) / (2 * a'). \\
& \quad \forall x \in \{(-b' + 1 * \text{sqrt}(b'^2 - 4 * a' * c')) / (2 * a') <..y'\}. \\
& \quad d * x^2 + e * x + f = 0) \wedge
\end{aligned}$$

$(\forall (d, e, f) \in \text{set } b.$   
 $\exists y' > (-b' + 1 * \text{sqrt}(b'^2 - 4 * a' * c')) / (2 * a').$   
 $\forall x \in \{(-b' + 1 * \text{sqrt}(b'^2 - 4 * a' * c')) / (2 * a') < ..y'\}.$   
 $d * x^2 + e * x + f < 0) \wedge$   
 $(\forall (d, e, f) \in \text{set } c.$   
 $\exists y' > (-b' + 1 * \text{sqrt}(b'^2 - 4 * a' * c')) / (2 * a').$   
 $\forall x \in \{(-b' + 1 * \text{sqrt}(b'^2 - 4 * a' * c')) / (2 * a') < ..y'\}.$   
 $d * x^2 + e * x + f \leq 0) \wedge$   
 $(\forall (d, e, f) \in \text{set } d.$   
 $\exists y' > (-b' + 1 * \text{sqrt}(b'^2 - 4 * a' * c')) / (2 * a').$   
 $\forall x \in \{(-b' + 1 * \text{sqrt}(b'^2 - 4 * a' * c')) / (2 * a') < ..y'\}.$   
 $d * x^2 + e * x + f \neq 0))$   
**assumes f7:**  $\neg(\exists(a', b', c') \in \text{set } b. a' \neq 0 \wedge$   
 $-b'^2 + 4 * a' * c' \leq 0 \wedge (\forall (d, e, f) \in \text{set } a.$   
 $\exists y' > (-b' + -1 * \text{sqrt}(b'^2 - 4 * a' * c')) / (2 * a').$   
 $\forall x \in \{(-b' + -1 * \text{sqrt}(b'^2 - 4 * a' * c')) / (2 * a') < ..y'\}.$   
 $d * x^2 + e * x + f = 0) \wedge$   
 $(\forall (d, e, f) \in \text{set } b.$   
 $\exists y' > (-b' + -1 * \text{sqrt}(b'^2 - 4 * a' * c')) / (2 * a').$   
 $\forall x \in \{(-b' + -1 * \text{sqrt}(b'^2 - 4 * a' * c')) / (2 * a') < ..y'\}.$   
 $d * x^2 + e * x + f < 0) \wedge$   
 $(\forall (d, e, f) \in \text{set } c.$   
 $\exists y' > (-b' + -1 * \text{sqrt}(b'^2 - 4 * a' * c')) / (2 * a').$   
 $\forall x \in \{(-b' + -1 * \text{sqrt}(b'^2 - 4 * a' * c')) / (2 * a') < ..y'\}.$   
 $d * x^2 + e * x + f \leq 0) \wedge$   
 $(\forall (d, e, f) \in \text{set } d.$   
 $\exists y' > (-b' + -1 * \text{sqrt}(b'^2 - 4 * a' * c')) / (2 * a').$   
 $\forall x \in \{(-b' + -1 * \text{sqrt}(b'^2 - 4 * a' * c')) / (2 * a') < ..y'\}.$   
 $d * x^2 + e * x + f \neq 0))$   
**assumes f8:**  $\neg(\exists(a', b', c') \in \text{set } c. (a' = 0 \wedge b' \neq 0) \wedge$   
 $(\forall (d, e, f) \in \text{set } a. d * (-c' / b')^2 + e * (-c' / b') + f = 0) \wedge$   
 $(\forall (d, e, f) \in \text{set } b. d * (-c' / b')^2 + e * (-c' / b') + f < 0) \wedge$   
 $(\forall (d, e, f) \in \text{set } c. d * (-c' / b')^2 + e * (-c' / b') + f \leq 0) \wedge$   
 $(\forall (d, e, f) \in \text{set } d. d * (-c' / b')^2 + e * (-c' / b') + f \neq 0))$   
**assumes f10:**  $\neg(\exists(a', b', c') \in \text{set } d.$   
 $(a' = 0 \wedge b' \neq 0) \wedge$   
 $(\forall (d, e, f) \in \text{set } a.$   
 $\exists y' > -c' / b'. \forall x \in \{-c' / b' < ..y'\}. d * x^2 + e * x + f = 0) \wedge$   
 $(\forall (d, e, f) \in \text{set } b.$   
 $\exists y' > -c' / b'. \forall x \in \{-c' / b' < ..y'\}. d * x^2 + e * x + f < 0) \wedge$   
 $(\forall (d, e, f) \in \text{set } c.$   
 $\exists y' > -c' / b'. \forall x \in \{-c' / b' < ..y'\}. d * x^2 + e * x + f \leq 0) \wedge$   
 $(\forall (d, e, f) \in \text{set } d.$   
 $\exists y' > -c' / b'. \forall x \in \{-c' / b' < ..y'\}. d * x^2 + e * x + f \neq 0))$   
**assumes f11:**  $\neg(\exists(a', b', c') \in \text{set } d.$   
 $a' \neq 0 \wedge$   
 $-b'^2 + 4 * a' * c' \leq 0 \wedge$   
 $(\forall (d, e, f) \in \text{set } a.$   
 $\exists y' > (-b' + 1 * \text{sqrt}(b'^2 - 4 * a' * c')) / (2 * a').$

$$\begin{aligned}
& \forall x \in \{(-b' + 1 * \text{sqrt}(b'^2 - 4 * a' * c')) / (2 * a') <..y'\}. \\
& \quad d * x^2 + e * x + f = 0) \wedge \\
(\forall (d, e, f) \in \text{set } b. \\
& \quad \exists y' > (-b' + 1 * \text{sqrt}(b'^2 - 4 * a' * c')) / (2 * a'). \\
& \quad \forall x \in \{(-b' + 1 * \text{sqrt}(b'^2 - 4 * a' * c')) / (2 * a') <..y'\}. \\
& \quad \quad d * x^2 + e * x + f < 0) \wedge \\
(\forall (d, e, f) \in \text{set } c. \\
& \quad \exists y' > (-b' + 1 * \text{sqrt}(b'^2 - 4 * a' * c')) / (2 * a'). \\
& \quad \forall x \in \{(-b' + 1 * \text{sqrt}(b'^2 - 4 * a' * c')) / (2 * a') <..y'\}. \\
& \quad \quad d * x^2 + e * x + f \leq 0) \wedge \\
(\forall (d, e, f) \in \text{set } d. \\
& \quad \exists y' > (-b' + 1 * \text{sqrt}(b'^2 - 4 * a' * c')) / (2 * a'). \\
& \quad \forall x \in \{(-b' + 1 * \text{sqrt}(b'^2 - 4 * a' * c')) / (2 * a') <..y'\}. \\
& \quad \quad d * x^2 + e * x + f \neq 0))) \\
\text{assumes } f12: & \neg(\exists (a', b', c') \in \text{set } d. \\
& \quad a' \neq 0 \wedge \\
& \quad -b'^2 + 4 * a' * c' \leq 0 \wedge (\forall (d, e, f) \in \text{set } a. \\
& \quad \quad \exists y' > (-b' + -1 * \text{sqrt}(b'^2 - 4 * a' * c')) / (2 * a'). \\
& \quad \quad \forall x \in \{(-b' + -1 * \text{sqrt}(b'^2 - 4 * a' * c')) / (2 * a') <..y'\}. \\
& \quad \quad \quad d * x^2 + e * x + f = 0) \wedge \\
& \quad \quad (\forall (d, e, f) \in \text{set } b. \\
& \quad \quad \quad \exists y' > (-b' + -1 * \text{sqrt}(b'^2 - 4 * a' * c')) / (2 * a'). \\
& \quad \quad \quad \forall x \in \{(-b' + -1 * \text{sqrt}(b'^2 - 4 * a' * c')) / (2 * a') <..y'\}. \\
& \quad \quad \quad \quad d * x^2 + e * x + f < 0) \wedge \\
& \quad \quad (\forall (d, e, f) \in \text{set } c. \\
& \quad \quad \quad \exists y' > (-b' + -1 * \text{sqrt}(b'^2 - 4 * a' * c')) / (2 * a'). \\
& \quad \quad \quad \forall x \in \{(-b' + -1 * \text{sqrt}(b'^2 - 4 * a' * c')) / (2 * a') <..y'\}. \\
& \quad \quad \quad \quad d * x^2 + e * x + f \leq 0) \wedge \\
& \quad \quad (\forall (d, e, f) \in \text{set } d. \\
& \quad \quad \quad \exists y' > (-b' + -1 * \text{sqrt}(b'^2 - 4 * a' * c')) / (2 * a'). \\
& \quad \quad \quad \forall x \in \{(-b' + -1 * \text{sqrt}(b'^2 - 4 * a' * c')) / (2 * a') <..y'\}. \\
& \quad \quad \quad \quad d * x^2 + e * x + f \neq 0))) \\
\text{shows } & \neg(\exists (a', b', c') \in \text{set } c. \\
& \quad (a' = 0 \wedge b' \neq 0) \wedge \\
& \quad (\forall (d, e, f) \in \text{set } a. \\
& \quad \quad \exists y' > -c' / b'. \forall x \in \{-c' / b' <..y'\}. d * x^2 + e * x + f = 0) \wedge \\
& \quad (\forall (d, e, f) \in \text{set } b. \\
& \quad \quad \exists y' > -c' / b'. \forall x \in \{-c' / b' <..y'\}. d * x^2 + e * x + f < 0) \wedge \\
& \quad (\forall (d, e, f) \in \text{set } c. \\
& \quad \quad \exists y' > -c' / b'. \forall x \in \{-c' / b' <..y'\}. d * x^2 + e * x + f \leq 0) \wedge \\
& \quad (\forall (d, e, f) \in \text{set } d. \\
& \quad \quad \exists y' > -c' / b'. \forall x \in \{-c' / b' <..y'\}. d * x^2 + e * x + f \neq 0))) \\
& \langle \text{proof} \rangle
\end{aligned}$$

**lemma** *qe-forwards-helper*:

**assumes** *allegset*:  $\forall x. (\forall (d, e, f) \in \text{set } a. d * x^2 + e * x + f = 0)$

**assumes** *f1*:  $\neg((\forall (a, b, c) \in \text{set } a. a = 0 \wedge b = 0 \wedge c = 0) \wedge$

$(\forall (a, b, c) \in \text{set } b. \exists x. \forall y < x. a * y^2 + b * y + c < 0) \wedge$   
 $(\forall (a, b, c) \in \text{set } c. \exists x. \forall y < x. a * y^2 + b * y + c \leq 0) \wedge$   
 $(\forall (a, b, c) \in \text{set } d. \exists x. \forall y < x. a * y^2 + b * y + c \neq 0))$

**assumes f5:**  $\neg(\exists (a', b', c') \in \text{set } b.$

$(a' = 0 \wedge b' \neq 0) \wedge$   
 $(\forall (d, e, f) \in \text{set } a.$   
 $\exists y' > -c' / b'. \forall x \in \{-c' / b' <..y'\}. d * x^2 + e * x + f = 0) \wedge$   
 $(\forall (d, e, f) \in \text{set } b.$   
 $\exists y' > -c' / b'. \forall x \in \{-c' / b' <..y'\}. d * x^2 + e * x + f < 0) \wedge$   
 $(\forall (d, e, f) \in \text{set } c.$   
 $\exists y' > -c' / b'. \forall x \in \{-c' / b' <..y'\}. d * x^2 + e * x + f \leq 0) \wedge$   
 $(\forall (d, e, f) \in \text{set } d.$   
 $\exists y' > -c' / b'. \forall x \in \{-c' / b' <..y'\}. d * x^2 + e * x + f \neq 0))$

**assumes f6:**  $\neg(\exists (a', b', c') \in \text{set } b. a' \neq 0 \wedge$

$-b^2 + 4 * a' * c' \leq 0 \wedge$   
 $(\forall (d, e, f) \in \text{set } a.$   
 $\exists y' > (-b' + 1 * \text{sqrt}(b^2 - 4 * a' * c')) / (2 * a').$   
 $\forall x \in \{(-b' + 1 * \text{sqrt}(b^2 - 4 * a' * c')) / (2 * a') <..y'\}.$   
 $d * x^2 + e * x + f = 0) \wedge$   
 $(\forall (d, e, f) \in \text{set } b.$   
 $\exists y' > (-b' + 1 * \text{sqrt}(b^2 - 4 * a' * c')) / (2 * a').$   
 $\forall x \in \{(-b' + 1 * \text{sqrt}(b^2 - 4 * a' * c')) / (2 * a') <..y'\}.$   
 $d * x^2 + e * x + f < 0) \wedge$   
 $(\forall (d, e, f) \in \text{set } c.$   
 $\exists y' > (-b' + 1 * \text{sqrt}(b^2 - 4 * a' * c')) / (2 * a').$   
 $\forall x \in \{(-b' + 1 * \text{sqrt}(b^2 - 4 * a' * c')) / (2 * a') <..y'\}.$   
 $d * x^2 + e * x + f \leq 0) \wedge$   
 $(\forall (d, e, f) \in \text{set } d.$   
 $\exists y' > (-b' + 1 * \text{sqrt}(b^2 - 4 * a' * c')) / (2 * a').$   
 $\forall x \in \{(-b' + 1 * \text{sqrt}(b^2 - 4 * a' * c')) / (2 * a') <..y'\}.$   
 $d * x^2 + e * x + f \neq 0))$

**assumes f7:**  $\neg(\exists (a', b', c') \in \text{set } b. a' \neq 0 \wedge$

$-b^2 + 4 * a' * c' \leq 0 \wedge (\forall (d, e, f) \in \text{set } a.$   
 $\exists y' > (-b' + -1 * \text{sqrt}(b^2 - 4 * a' * c')) / (2 * a').$   
 $\forall x \in \{(-b' + -1 * \text{sqrt}(b^2 - 4 * a' * c')) / (2 * a') <..y'\}.$   
 $d * x^2 + e * x + f = 0) \wedge$   
 $(\forall (d, e, f) \in \text{set } b.$   
 $\exists y' > (-b' + -1 * \text{sqrt}(b^2 - 4 * a' * c')) / (2 * a').$   
 $\forall x \in \{(-b' + -1 * \text{sqrt}(b^2 - 4 * a' * c')) / (2 * a') <..y'\}.$   
 $d * x^2 + e * x + f < 0) \wedge$   
 $(\forall (d, e, f) \in \text{set } c.$   
 $\exists y' > (-b' + -1 * \text{sqrt}(b^2 - 4 * a' * c')) / (2 * a').$   
 $\forall x \in \{(-b' + -1 * \text{sqrt}(b^2 - 4 * a' * c')) / (2 * a') <..y'\}.$   
 $d * x^2 + e * x + f \leq 0) \wedge$   
 $(\forall (d, e, f) \in \text{set } d.$   
 $\exists y' > (-b' + -1 * \text{sqrt}(b^2 - 4 * a' * c')) / (2 * a').$   
 $\forall x \in \{(-b' + -1 * \text{sqrt}(b^2 - 4 * a' * c')) / (2 * a') <..y'\}.$   
 $d * x^2 + e * x + f \neq 0))$

**assumes f8:**  $\neg(\exists (a', b', c') \in \text{set } c. (a' = 0 \wedge b' \neq 0) \wedge$

$$\begin{aligned}
& (\forall (d, e, f) \in \text{set } a. d * (-c' / b')^2 + e * (-c' / b') + f = 0) \wedge \\
& (\forall (d, e, f) \in \text{set } b. d * (-c' / b')^2 + e * (-c' / b') + f < 0) \wedge \\
& (\forall (d, e, f) \in \text{set } c. d * (-c' / b')^2 + e * (-c' / b') + f \leq 0) \wedge \\
& (\forall (d, e, f) \in \text{set } d. d * (-c' / b')^2 + e * (-c' / b') + f \neq 0)) \\
\text{assumes } f13: & \neg(\exists (a', b', c') \in \text{set } c. \\
& a' \neq 0 \wedge \\
& -b'^2 + 4 * a' * c' \leq 0 \wedge \\
& ((\forall (d, e, f) \in \text{set } a. \\
& \quad d * ((-b' + 1 * \text{sqrt}(b'^2 - 4 * a' * c')) / (2 * a'))^2 + \\
& \quad e * ((-b' + 1 * \text{sqrt}(b'^2 - 4 * a' * c')) / (2 * a')) + \\
& \quad f = 0) \wedge \\
& (\forall (d, e, f) \in \text{set } b. \\
& \quad d * ((-b' + 1 * \text{sqrt}(b'^2 - 4 * a' * c')) / (2 * a'))^2 + \\
& \quad e * ((-b' + 1 * \text{sqrt}(b'^2 - 4 * a' * c')) / (2 * a')) + \\
& \quad f < 0) \wedge \\
& (\forall (d, e, f) \in \text{set } c. \\
& \quad d * ((-b' + 1 * \text{sqrt}(b'^2 - 4 * a' * c')) / (2 * a'))^2 + \\
& \quad e * ((-b' + 1 * \text{sqrt}(b'^2 - 4 * a' * c')) / (2 * a')) + \\
& \quad f \leq 0) \wedge \\
& (\forall (d, e, f) \in \text{set } d. \\
& \quad d * ((-b' + 1 * \text{sqrt}(b'^2 - 4 * a' * c')) / (2 * a'))^2 + \\
& \quad e * ((-b' + 1 * \text{sqrt}(b'^2 - 4 * a' * c')) / (2 * a')) + \\
& \quad f \neq 0))) \\
\text{assumes } f9: & \neg(\exists (a', b', c') \in \text{set } c. a' \neq 0 \wedge \\
& -b'^2 + 4 * a' * c' \leq 0 \wedge \\
& (\forall (d, e, f) \in \text{set } a. \\
& \quad d * ((-b' + -1 * \text{sqrt}(b'^2 - 4 * a' * c')) / (2 * a'))^2 + \\
& \quad e * ((-b' + -1 * \text{sqrt}(b'^2 - 4 * a' * c')) / (2 * a')) + \\
& \quad f = 0) \wedge \\
& (\forall (d, e, f) \in \text{set } b. \\
& \quad d * ((-b' + -1 * \text{sqrt}(b'^2 - 4 * a' * c')) / (2 * a'))^2 + \\
& \quad e * ((-b' + -1 * \text{sqrt}(b'^2 - 4 * a' * c')) / (2 * a')) + \\
& \quad f < 0) \wedge \\
& (\forall (d, e, f) \in \text{set } c. \\
& \quad d * ((-b' + -1 * \text{sqrt}(b'^2 - 4 * a' * c')) / (2 * a'))^2 + \\
& \quad e * ((-b' + -1 * \text{sqrt}(b'^2 - 4 * a' * c')) / (2 * a')) + \\
& \quad f \leq 0) \wedge \\
& (\forall (d, e, f) \in \text{set } d. \\
& \quad d * ((-b' + -1 * \text{sqrt}(b'^2 - 4 * a' * c')) / (2 * a'))^2 + \\
& \quad e * ((-b' + -1 * \text{sqrt}(b'^2 - 4 * a' * c')) / (2 * a')) + \\
& \quad f \neq 0)) \\
\text{assumes } f10: & \neg(\exists (a', b', c') \in \text{set } d. \\
& (a' = 0 \wedge b' \neq 0) \wedge \\
& (\forall (d, e, f) \in \text{set } a. \\
& \quad \exists y' > -c' / b'. \forall x \in \{-c' / b' < ..y'\}. d * x^2 + e * x + f = 0) \wedge \\
& (\forall (d, e, f) \in \text{set } b. \\
& \quad \exists y' > -c' / b'. \forall x \in \{-c' / b' < ..y'\}. d * x^2 + e * x + f < 0) \wedge \\
& (\forall (d, e, f) \in \text{set } c. \\
& \quad \exists y' > -c' / b'. \forall x \in \{-c' / b' < ..y'\}. d * x^2 + e * x + f \leq 0) \wedge
\end{aligned}$$

$(\forall (d, e, f) \in \text{set } d.$   
 $\exists y' > -c' / b'. \forall x \in \{-c' / b' < ..y'\}. d * x^2 + e * x + f \neq 0)$   
**assumes**  $f11: \neg(\exists (a', b', c') \in \text{set } d.$   
 $a' \neq 0 \wedge$   
 $-b'^2 + 4 * a' * c' \leq 0 \wedge$   
 $(\forall (d, e, f) \in \text{set } a.$   
 $\exists y' > (-b' + 1 * \text{sqrt}(b'^2 - 4 * a' * c')) / (2 * a').$   
 $\forall x \in \{-b' + 1 * \text{sqrt}(b'^2 - 4 * a' * c') / (2 * a') < ..y'\}.$   
 $d * x^2 + e * x + f = 0) \wedge$   
 $(\forall (d, e, f) \in \text{set } b.$   
 $\exists y' > (-b' + 1 * \text{sqrt}(b'^2 - 4 * a' * c')) / (2 * a').$   
 $\forall x \in \{-b' + 1 * \text{sqrt}(b'^2 - 4 * a' * c') / (2 * a') < ..y'\}.$   
 $d * x^2 + e * x + f < 0) \wedge$   
 $(\forall (d, e, f) \in \text{set } c.$   
 $\exists y' > (-b' + 1 * \text{sqrt}(b'^2 - 4 * a' * c')) / (2 * a').$   
 $\forall x \in \{-b' + 1 * \text{sqrt}(b'^2 - 4 * a' * c') / (2 * a') < ..y'\}.$   
 $d * x^2 + e * x + f \leq 0) \wedge$   
 $(\forall (d, e, f) \in \text{set } d.$   
 $\exists y' > (-b' + 1 * \text{sqrt}(b'^2 - 4 * a' * c')) / (2 * a').$   
 $\forall x \in \{-b' + 1 * \text{sqrt}(b'^2 - 4 * a' * c') / (2 * a') < ..y'\}.$   
 $d * x^2 + e * x + f \neq 0)))$   
**assumes**  $f12: \neg(\exists (a', b', c') \in \text{set } d.$   
 $a' \neq 0 \wedge$   
 $-b'^2 + 4 * a' * c' \leq 0 \wedge (\forall (d, e, f) \in \text{set } a.$   
 $\exists y' > (-b' + -1 * \text{sqrt}(b'^2 - 4 * a' * c')) / (2 * a').$   
 $\forall x \in \{-b' + -1 * \text{sqrt}(b'^2 - 4 * a' * c') / (2 * a') < ..y'\}.$   
 $d * x^2 + e * x + f = 0) \wedge$   
 $(\forall (d, e, f) \in \text{set } b.$   
 $\exists y' > (-b' + -1 * \text{sqrt}(b'^2 - 4 * a' * c')) / (2 * a').$   
 $\forall x \in \{-b' + -1 * \text{sqrt}(b'^2 - 4 * a' * c') / (2 * a') < ..y'\}.$   
 $d * x^2 + e * x + f < 0) \wedge$   
 $(\forall (d, e, f) \in \text{set } c.$   
 $\exists y' > (-b' + -1 * \text{sqrt}(b'^2 - 4 * a' * c')) / (2 * a').$   
 $\forall x \in \{-b' + -1 * \text{sqrt}(b'^2 - 4 * a' * c') / (2 * a') < ..y'\}.$   
 $d * x^2 + e * x + f \leq 0) \wedge$   
 $(\forall (d, e, f) \in \text{set } d.$   
 $\exists y' > (-b' + -1 * \text{sqrt}(b'^2 - 4 * a' * c')) / (2 * a').$   
 $\forall x \in \{-b' + -1 * \text{sqrt}(b'^2 - 4 * a' * c') / (2 * a') < ..y'\}.$   
 $d * x^2 + e * x + f \neq 0))$   
**shows**  $\neg(\exists x. (\forall (a, b, c) \in \text{set } b. a * x^2 + b * x + c < 0) \wedge$   
 $(\forall (a, b, c) \in \text{set } c. a * x^2 + b * x + c \leq 0) \wedge$   
 $(\forall (a, b, c) \in \text{set } d. a * x^2 + b * x + c \neq 0))$   
 $\langle \text{proof} \rangle$

**lemma** *qe-forwards*:

**assumes**  $(\exists x. (\forall (a, b, c) \in \text{set } a. a * x^2 + b * x + c = 0) \wedge$   
 $(\forall (a, b, c) \in \text{set } b. a * x^2 + b * x + c < 0) \wedge$   
 $(\forall (a, b, c) \in \text{set } c. a * x^2 + b * x + c \leq 0) \wedge$

$(\forall (a, b, c) \in \text{set } d. a * x^2 + b * x + c \neq 0)$   
**shows**  $(\forall (a, b, c) \in \text{set } a. a = 0 \wedge b = 0 \wedge c = 0) \wedge$   
 $(\forall (a, b, c) \in \text{set } b. \exists x. \forall y < x. a * y^2 + b * y + c < 0) \wedge$   
 $(\forall (a, b, c) \in \text{set } c. \exists x. \forall y < x. a * y^2 + b * y + c \leq 0) \wedge$   
 $(\forall (a, b, c) \in \text{set } d. \exists x. \forall y < x. a * y^2 + b * y + c \neq 0) \vee$   
 $(\exists (a', b', c') \in \text{set } a.$   
 $(a' = 0 \wedge b' \neq 0) \wedge$   
 $(\forall (d, e, f) \in \text{set } a. d * (-c' / b')^2 + e * (-c' / b') + f = 0) \wedge$   
 $(\forall (d, e, f) \in \text{set } b. d * (-c' / b')^2 + e * (-c' / b') + f < 0) \wedge$   
 $(\forall (d, e, f) \in \text{set } c. d * (-c' / b')^2 + e * (-c' / b') + f \leq 0) \wedge$   
 $(\forall (d, e, f) \in \text{set } d. d * (-c' / b')^2 + e * (-c' / b') + f \neq 0) \vee$   
 $a' \neq 0 \wedge$   
 $-b'^2 + 4 * a' * c' \leq 0 \wedge$   
 $(\forall (d, e, f) \in \text{set } a.$   
 $d * ((-b' + 1 * \text{sqrt}(b'^2 - 4 * a' * c')) / (2 * a'))^2 +$   
 $e * ((-b' + 1 * \text{sqrt}(b'^2 - 4 * a' * c')) / (2 * a')) +$   
 $f =$   
 $0) \wedge$   
 $(\forall (d, e, f) \in \text{set } b.$   
 $d * ((-b' + 1 * \text{sqrt}(b'^2 - 4 * a' * c')) / (2 * a'))^2 +$   
 $e * ((-b' + 1 * \text{sqrt}(b'^2 - 4 * a' * c')) / (2 * a')) +$   
 $f$   
 $< 0) \wedge$   
 $(\forall (d, e, f) \in \text{set } c.$   
 $d * ((-b' + 1 * \text{sqrt}(b'^2 - 4 * a' * c')) / (2 * a'))^2 +$   
 $e * ((-b' + 1 * \text{sqrt}(b'^2 - 4 * a' * c')) / (2 * a')) +$   
 $f$   
 $\leq 0) \wedge$   
 $(\forall (d, e, f) \in \text{set } d.$   
 $d * ((-b' + 1 * \text{sqrt}(b'^2 - 4 * a' * c')) / (2 * a'))^2 +$   
 $e * ((-b' + 1 * \text{sqrt}(b'^2 - 4 * a' * c')) / (2 * a')) +$   
 $f \neq$   
 $0) \vee$   
 $(\forall (d, e, f) \in \text{set } a.$   
 $d * ((-b' + -1 * \text{sqrt}(b'^2 - 4 * a' * c')) / (2 * a'))^2 +$   
 $e * ((-b' + -1 * \text{sqrt}(b'^2 - 4 * a' * c')) / (2 * a')) +$   
 $f =$   
 $0) \wedge$   
 $(\forall (d, e, f) \in \text{set } b.$   
 $d * ((-b' + -1 * \text{sqrt}(b'^2 - 4 * a' * c')) / (2 * a'))^2 +$   
 $e * ((-b' + -1 * \text{sqrt}(b'^2 - 4 * a' * c')) / (2 * a')) +$   
 $f$   
 $< 0) \wedge$   
 $(\forall (d, e, f) \in \text{set } c.$   
 $d * ((-b' + -1 * \text{sqrt}(b'^2 - 4 * a' * c')) / (2 * a'))^2 +$   
 $e * ((-b' + -1 * \text{sqrt}(b'^2 - 4 * a' * c')) / (2 * a')) +$   
 $f$   
 $\leq 0) \wedge$   
 $(\forall (d, e, f) \in \text{set } d.$

$$\begin{aligned}
& d * ((- b' + - 1 * \text{sqrt}(b'^2 - 4 * a' * c')) / (2 * a'))^2 + \\
& e * ((- b' + - 1 * \text{sqrt}(b'^2 - 4 * a' * c')) / (2 * a')) + \\
& f \neq \\
& 0))) \vee \\
(\exists (a', b', c') \in \text{set } b. \\
& (a' = 0 \wedge b' \neq 0) \wedge \\
& (\forall (d, e, f) \in \text{set } a. \\
& \quad \exists y' > -c' / b'. \forall x \in \{-c' / b' <..y'\}. d * x^2 + e * x + f = 0) \wedge \\
& (\forall (d, e, f) \in \text{set } b. \\
& \quad \exists y' > -c' / b'. \forall x \in \{-c' / b' <..y'\}. d * x^2 + e * x + f < 0) \wedge \\
& (\forall (d, e, f) \in \text{set } c. \\
& \quad \exists y' > -c' / b'. \forall x \in \{-c' / b' <..y'\}. d * x^2 + e * x + f \leq 0) \wedge \\
& (\forall (d, e, f) \in \text{set } d. \\
& \quad \exists y' > -c' / b'. \forall x \in \{-c' / b' <..y'\}. d * x^2 + e * x + f \neq 0) \vee \\
& a' \neq 0 \wedge \\
& -b'^2 + 4 * a' * c' \leq 0 \wedge \\
& (\forall (d, e, f) \in \text{set } a. \\
& \quad \exists y' > (-b' + 1 * \text{sqrt}(b'^2 - 4 * a' * c')) / (2 * a'). \\
& \quad \forall x \in \{(-b' + 1 * \text{sqrt}(b'^2 - 4 * a' * c')) / (2 * a') <..y'\}. \\
& \quad \quad d * x^2 + e * x + f = 0) \wedge \\
& (\forall (d, e, f) \in \text{set } b. \\
& \quad \exists y' > (-b' + 1 * \text{sqrt}(b'^2 - 4 * a' * c')) / (2 * a'). \\
& \quad \forall x \in \{(-b' + 1 * \text{sqrt}(b'^2 - 4 * a' * c')) / (2 * a') <..y'\}. \\
& \quad \quad d * x^2 + e * x + f < 0) \wedge \\
& (\forall (d, e, f) \in \text{set } c. \\
& \quad \exists y' > (-b' + 1 * \text{sqrt}(b'^2 - 4 * a' * c')) / (2 * a'). \\
& \quad \forall x \in \{(-b' + 1 * \text{sqrt}(b'^2 - 4 * a' * c')) / (2 * a') <..y'\}. \\
& \quad \quad d * x^2 + e * x + f \leq 0) \wedge \\
& (\forall (d, e, f) \in \text{set } d. \\
& \quad \exists y' > (-b' + 1 * \text{sqrt}(b'^2 - 4 * a' * c')) / (2 * a'). \\
& \quad \forall x \in \{(-b' + 1 * \text{sqrt}(b'^2 - 4 * a' * c')) / (2 * a') <..y'\}. \\
& \quad \quad d * x^2 + e * x + f \neq 0) \vee \\
& (\forall (d, e, f) \in \text{set } a. \\
& \quad \exists y' > (-b' + - 1 * \text{sqrt}(b'^2 - 4 * a' * c')) / (2 * a'). \\
& \quad \forall x \in \{(-b' + - 1 * \text{sqrt}(b'^2 - 4 * a' * c')) / (2 * a') <..y'\}. \\
& \quad \quad d * x^2 + e * x + f = 0) \wedge \\
& (\forall (d, e, f) \in \text{set } b. \\
& \quad \exists y' > (-b' + - 1 * \text{sqrt}(b'^2 - 4 * a' * c')) / (2 * a'). \\
& \quad \forall x \in \{(-b' + - 1 * \text{sqrt}(b'^2 - 4 * a' * c')) / (2 * a') <..y'\}. \\
& \quad \quad d * x^2 + e * x + f < 0) \wedge \\
& (\forall (d, e, f) \in \text{set } c. \\
& \quad \exists y' > (-b' + - 1 * \text{sqrt}(b'^2 - 4 * a' * c')) / (2 * a'). \\
& \quad \forall x \in \{(-b' + - 1 * \text{sqrt}(b'^2 - 4 * a' * c')) / (2 * a') <..y'\}. \\
& \quad \quad d * x^2 + e * x + f \leq 0) \wedge \\
& (\forall (d, e, f) \in \text{set } d. \\
& \quad \exists y' > (-b' + - 1 * \text{sqrt}(b'^2 - 4 * a' * c')) / (2 * a'). \\
& \quad \forall x \in \{(-b' + - 1 * \text{sqrt}(b'^2 - 4 * a' * c')) / (2 * a') <..y'\}. \\
& \quad \quad d * x^2 + e * x + f \neq 0))) \vee \\
(\exists (a', b', c') \in \text{set } c.
\end{aligned}$$

$$\begin{aligned}
& (a' = 0 \wedge b' \neq 0) \wedge \\
& (\forall (d, e, f) \in \text{set } a. d * (-c' / b')^2 + e * (-c' / b') + f = 0) \wedge \\
& (\forall (d, e, f) \in \text{set } b. d * (-c' / b')^2 + e * (-c' / b') + f < 0) \wedge \\
& (\forall (d, e, f) \in \text{set } c. d * (-c' / b')^2 + e * (-c' / b') + f \leq 0) \wedge \\
& (\forall (d, e, f) \in \text{set } d. d * (-c' / b')^2 + e * (-c' / b') + f \neq 0) \vee \\
& a' \neq 0 \wedge \\
& -b'^2 + 4 * a' * c' \leq 0 \wedge \\
& ((\forall (d, e, f) \in \text{set } a. \\
& \quad d * ((-b' + 1 * \text{sqrt}(b'^2 - 4 * a' * c')) / (2 * a'))^2 + \\
& \quad e * ((-b' + 1 * \text{sqrt}(b'^2 - 4 * a' * c')) / (2 * a')) + \\
& \quad f = \\
& \quad 0) \wedge \\
& (\forall (d, e, f) \in \text{set } b. \\
& \quad d * ((-b' + 1 * \text{sqrt}(b'^2 - 4 * a' * c')) / (2 * a'))^2 + \\
& \quad e * ((-b' + 1 * \text{sqrt}(b'^2 - 4 * a' * c')) / (2 * a')) + \\
& \quad f \\
& \quad < 0) \wedge \\
& (\forall (d, e, f) \in \text{set } c. \\
& \quad d * ((-b' + 1 * \text{sqrt}(b'^2 - 4 * a' * c')) / (2 * a'))^2 + \\
& \quad e * ((-b' + 1 * \text{sqrt}(b'^2 - 4 * a' * c')) / (2 * a')) + \\
& \quad f \\
& \quad \leq 0) \wedge \\
& (\forall (d, e, f) \in \text{set } d. \\
& \quad d * ((-b' + 1 * \text{sqrt}(b'^2 - 4 * a' * c')) / (2 * a'))^2 + \\
& \quad e * ((-b' + 1 * \text{sqrt}(b'^2 - 4 * a' * c')) / (2 * a')) + \\
& \quad f \neq \\
& \quad 0) \vee \\
& (\forall (d, e, f) \in \text{set } a. \\
& \quad d * ((-b' + -1 * \text{sqrt}(b'^2 - 4 * a' * c')) / (2 * a'))^2 + \\
& \quad e * ((-b' + -1 * \text{sqrt}(b'^2 - 4 * a' * c')) / (2 * a')) + \\
& \quad f = \\
& \quad 0) \wedge \\
& (\forall (d, e, f) \in \text{set } b. \\
& \quad d * ((-b' + -1 * \text{sqrt}(b'^2 - 4 * a' * c')) / (2 * a'))^2 + \\
& \quad e * ((-b' + -1 * \text{sqrt}(b'^2 - 4 * a' * c')) / (2 * a')) + \\
& \quad f \\
& \quad < 0) \wedge \\
& (\forall (d, e, f) \in \text{set } c. \\
& \quad d * ((-b' + -1 * \text{sqrt}(b'^2 - 4 * a' * c')) / (2 * a'))^2 + \\
& \quad e * ((-b' + -1 * \text{sqrt}(b'^2 - 4 * a' * c')) / (2 * a')) + \\
& \quad f \\
& \quad \leq 0) \wedge \\
& (\forall (d, e, f) \in \text{set } d. \\
& \quad d * ((-b' + -1 * \text{sqrt}(b'^2 - 4 * a' * c')) / (2 * a'))^2 + \\
& \quad e * ((-b' + -1 * \text{sqrt}(b'^2 - 4 * a' * c')) / (2 * a')) + \\
& \quad f \neq \\
& \quad 0))) \vee \\
& (\exists (a', b', c') \in \text{set } d. \\
& \quad (a' = 0 \wedge b' \neq 0) \wedge
\end{aligned}$$

$$\begin{aligned}
& (\forall (d, e, f) \in \text{set } a. \\
& \quad \exists y' > -c' / b'. \forall x \in \{-c' / b' <..y'\}. d * x^2 + e * x + f = 0) \wedge \\
& (\forall (d, e, f) \in \text{set } b. \\
& \quad \exists y' > -c' / b'. \forall x \in \{-c' / b' <..y'\}. d * x^2 + e * x + f < 0) \wedge \\
& (\forall (d, e, f) \in \text{set } c. \\
& \quad \exists y' > -c' / b'. \forall x \in \{-c' / b' <..y'\}. d * x^2 + e * x + f \leq 0) \wedge \\
& (\forall (d, e, f) \in \text{set } d. \\
& \quad \exists y' > -c' / b'. \forall x \in \{-c' / b' <..y'\}. d * x^2 + e * x + f \neq 0) \vee \\
& a' \neq 0 \wedge \\
& -b'^2 + 4 * a' * c' \leq 0 \wedge \\
& ((\forall (d, e, f) \in \text{set } a. \\
& \quad \exists y' > (-b' + 1 * \text{sqrt}(b'^2 - 4 * a' * c')) / (2 * a'). \\
& \quad \forall x \in \{(-b' + 1 * \text{sqrt}(b'^2 - 4 * a' * c')) / (2 * a') <..y'\}. \\
& \quad \quad d * x^2 + e * x + f = 0) \wedge \\
& (\forall (d, e, f) \in \text{set } b. \\
& \quad \exists y' > (-b' + 1 * \text{sqrt}(b'^2 - 4 * a' * c')) / (2 * a'). \\
& \quad \forall x \in \{(-b' + 1 * \text{sqrt}(b'^2 - 4 * a' * c')) / (2 * a') <..y'\}. \\
& \quad \quad d * x^2 + e * x + f < 0) \wedge \\
& (\forall (d, e, f) \in \text{set } c. \\
& \quad \exists y' > (-b' + 1 * \text{sqrt}(b'^2 - 4 * a' * c')) / (2 * a'). \\
& \quad \forall x \in \{(-b' + 1 * \text{sqrt}(b'^2 - 4 * a' * c')) / (2 * a') <..y'\}. \\
& \quad \quad d * x^2 + e * x + f \leq 0) \wedge \\
& (\forall (d, e, f) \in \text{set } d. \\
& \quad \exists y' > (-b' + 1 * \text{sqrt}(b'^2 - 4 * a' * c')) / (2 * a'). \\
& \quad \forall x \in \{(-b' + 1 * \text{sqrt}(b'^2 - 4 * a' * c')) / (2 * a') <..y'\}. \\
& \quad \quad d * x^2 + e * x + f \neq 0) \vee \\
& (\forall (d, e, f) \in \text{set } a. \\
& \quad \exists y' > (-b' + -1 * \text{sqrt}(b'^2 - 4 * a' * c')) / (2 * a'). \\
& \quad \forall x \in \{(-b' + -1 * \text{sqrt}(b'^2 - 4 * a' * c')) / (2 * a') <..y'\}. \\
& \quad \quad d * x^2 + e * x + f = 0) \wedge \\
& (\forall (d, e, f) \in \text{set } b. \\
& \quad \exists y' > (-b' + -1 * \text{sqrt}(b'^2 - 4 * a' * c')) / (2 * a'). \\
& \quad \forall x \in \{(-b' + -1 * \text{sqrt}(b'^2 - 4 * a' * c')) / (2 * a') <..y'\}. \\
& \quad \quad d * x^2 + e * x + f < 0) \wedge \\
& (\forall (d, e, f) \in \text{set } c. \\
& \quad \exists y' > (-b' + -1 * \text{sqrt}(b'^2 - 4 * a' * c')) / (2 * a'). \\
& \quad \forall x \in \{(-b' + -1 * \text{sqrt}(b'^2 - 4 * a' * c')) / (2 * a') <..y'\}. \\
& \quad \quad d * x^2 + e * x + f \leq 0) \wedge \\
& (\forall (d, e, f) \in \text{set } d. \\
& \quad \exists y' > (-b' + -1 * \text{sqrt}(b'^2 - 4 * a' * c')) / (2 * a'). \\
& \quad \forall x \in \{(-b' + -1 * \text{sqrt}(b'^2 - 4 * a' * c')) / (2 * a') <..y'\}. \\
& \quad \quad d * x^2 + e * x + f \neq 0))))))
\end{aligned}$$

*<proof>*

## 2.8.5 Some Cases and Misc

**lemma quadratic-linear :**  
**assumes  $b \neq 0$**

**assumes**  $a \neq 0$   
**assumes**  $4 * a * ba \leq aa^2$   
**assumes**  $b * (\text{sqrt}(aa^2 - 4 * a * ba) - aa) / (2 * a) + c = 0$   
**assumes**  $\forall x \in \text{set eq.}$   
*case x of*  
 $(d, e, f) \Rightarrow$   
 $d * ((\text{sqrt}(aa^2 - 4 * a * ba) - aa) / (2 * a))^2 +$   
 $e * (\text{sqrt}(aa^2 - 4 * a * ba) - aa) / (2 * a) +$   
 $f =$   
 $0$   
**assumes**  $(aaa, aaaa, baa) \in \text{set eq}$   
**shows**  $aaa * (c / b)^2 - aaaa * c / b + baa = 0$   
 <proof>

**lemma quadratic-linear1:**

**assumes**  $b \neq 0$   
**assumes**  $a \neq 0$   
**assumes**  $4 * a * ba \leq aa^2$   
**assumes**  $(b::\text{real}) * (\text{sqrt}((a::\text{real})^2 - 4 * (a::\text{real}) * (ba::\text{real})) - (a::\text{real})) /$   
 $(2 * a) + (c::\text{real}) = 0$   
**assumes**  
 $(\forall x \in \text{set } (les::(\text{real} * \text{real} * \text{real}) \text{list}).$   
*case x of*  
 $(d, e, f) \Rightarrow$   
 $d * ((\text{sqrt}(aa^2 - 4 * a * ba) - aa) / (2 * a))^2 +$   
 $e * (\text{sqrt}(aa^2 - 4 * a * ba) - aa) / (2 * a) +$   
 $f$   
 $< 0)$   
**assumes**  $(aaa, aaaa, baa) \in \text{set les}$   
**shows**  $aaa * (c / b)^2 - aaaa * c / b + baa < 0$   
 <proof>

**lemma quadratic-linear2 :**

**assumes**  $b \neq 0$   
**assumes**  $a \neq 0$   
**assumes**  $4 * a * ba \leq aa^2$   
**assumes**  $b * (-aa - \text{sqrt}(aa^2 - 4 * a * ba)) / (2 * a) + c = 0$   
**assumes**  $\forall x \in \text{set eq.}$   
*case x of*  
 $(d, e, f) \Rightarrow$   
 $d * ((-aa - \text{sqrt}(aa^2 - 4 * a * ba)) / (2 * a))^2 +$   
 $e * (-aa - \text{sqrt}(aa^2 - 4 * a * ba)) / (2 * a) +$   
 $f =$   
 $0$   
**assumes**  $(aaa, aaaa, baa) \in \text{set eq}$   
**shows**  $aaa * (c / b)^2 - aaaa * c / b + baa = 0$   
 <proof>

**lemma quadratic-linear3:**

**assumes**  $b \neq 0$   
**assumes**  $a \neq 0$   
**assumes**  $4 * a * ba \leq aa^2$   
**assumes**  $(b::real) * (- (aa::real) - \text{sqrt} ((aa::real)^2 - 4 * (a::real) * (ba::real))) / (2 * a) + (c::real) = 0$   
**assumes**  $(\forall x \in \text{set } (les::(\text{real} * \text{real} * \text{real}) \text{list}).$   
*case x of*  
 $(d, e, f) \Rightarrow$   
 $d * ((- aa - \text{sqrt} (aa^2 - 4 * a * ba)) / (2 * a))^2 +$   
 $e * (- aa - \text{sqrt} (aa^2 - 4 * a * ba)) / (2 * a) +$   
 $f$   
 $< 0)$   
**assumes**  $(aaa, aaaa, baa) \in \text{set } les$   
**shows**  $aaa * (c / b)^2 - aaaa * c / b + baa < 0$   
*<proof>*

**lemma** *h1b-helper-les:*

$(\forall ((a::real), (b::real), (c::real)) \in \text{set } les. \exists x. \forall y < x. a * y^2 + b * y + c < 0) \Rightarrow$   
 $(\exists y. \forall x < y. (\forall (a, b, c) \in \text{set } les. a * x^2 + b * x + c < 0))$   
*<proof>*

**lemma** *h1b-helper-leq:*

$(\forall ((a::real), (b::real), (c::real)) \in \text{set } leq. \exists x. \forall y < x. a * y^2 + b * y + c \leq 0) \Rightarrow$   
 $(\exists y. \forall x < y. (\forall (a, b, c) \in \text{set } leq. a * x^2 + b * x + c \leq 0))$   
*<proof>*

**lemma** *h1b-helper-neq:*

$(\forall ((a::real), (b::real), (c::real)) \in \text{set } neq. \exists x. \forall y < x. a * y^2 + b * y + c \neq 0) \Rightarrow$   
 $(\exists y. \forall x < y. (\forall (a, b, c) \in \text{set } neq. a * x^2 + b * x + c \neq 0))$   
*<proof>*

**lemma** *min-lem:*

**fixes**  $r::real$   
**assumes**  $a1: (\exists y' > r. (\forall ((d::real), (e::real), (f::real)) \in \text{set } b. \forall x \in \{r < ..y'\}. d * x^2 + e * x + f < 0))$   
**assumes**  $a2: (\exists y' > r. (\forall ((d::real), (e::real), (f::real)) \in \text{set } c. \forall x \in \{r < ..y'\}. d * x^2 + e * x + f \leq 0))$   
**assumes**  $a3: (\exists y' > r. (\forall ((d::real), (e::real), (f::real)) \in \text{set } d. \forall x \in \{r < ..y'\}. d * x^2 + e * x + f \neq 0))$   
**shows**  $(\exists x. (\forall (a, b, c) \in \text{set } b. a * x^2 + b * x + c < 0) \wedge$   
 $(\forall (a, b, c) \in \text{set } c. a * x^2 + b * x + c \leq 0) \wedge$   
 $(\forall (a, b, c) \in \text{set } d. a * x^2 + b * x + c \neq 0))$   
*<proof>*

**lemma** *qe-infinitesimals-helper:*

**fixes**  $k::real$   
**assumes**  $asm: (\forall (d, e, f) \in \text{set } a. \exists y' > k. \forall x \in \{k < ..y'\}. d * x^2 + e * x + f = 0)$

$\wedge$   
 $(\forall (d, e, f) \in \text{set } b. \exists y' > k. \forall x \in \{k <..y'\}. d * x^2 + e * x + f < 0) \wedge$   
 $(\forall (d, e, f) \in \text{set } c. \exists y' > k. \forall x \in \{k <..y'\}. d * x^2 + e * x + f \leq 0) \wedge$   
 $(\forall (d, e, f) \in \text{set } d. \exists y' > k. \forall x \in \{k <..y'\}. d * x^2 + e * x + f \neq 0)$   
**shows**  $(\exists x. (\forall (a, b, c) \in \text{set } a. a * x^2 + b * x + c = 0) \wedge$   
 $(\forall (a, b, c) \in \text{set } b. a * x^2 + b * x + c < 0) \wedge$   
 $(\forall (a, b, c) \in \text{set } c. a * x^2 + b * x + c \leq 0) \wedge$   
 $(\forall (a, b, c) \in \text{set } d. a * x^2 + b * x + c \neq 0))$   
 $\langle \text{proof} \rangle$

## 2.8.6 The `qe_backwards` lemma

**lemma** *qe-backwards*:

**assumes**  $((\forall (a, b, c) \in \text{set } a. a = 0 \wedge b = 0 \wedge c = 0) \wedge$   
 $(\forall (a, b, c) \in \text{set } b. \exists x. \forall y < x. a * y^2 + b * y + c < 0) \wedge$   
 $(\forall (a, b, c) \in \text{set } c. \exists x. \forall y < x. a * y^2 + b * y + c \leq 0) \wedge$   
 $(\forall (a, b, c) \in \text{set } d. \exists x. \forall y < x. a * y^2 + b * y + c \neq 0)$   
 $\vee$   
 $(\exists (a', b', c') \in \text{set } a.$   
 $(a' = 0 \wedge b' \neq 0) \wedge$   
 $(\forall (d, e, f) \in \text{set } a. d * (-c' / b')^2 + e * (-c' / b') + f = 0) \wedge$   
 $(\forall (d, e, f) \in \text{set } b. d * (-c' / b')^2 + e * (-c' / b') + f < 0) \wedge$   
 $(\forall (d, e, f) \in \text{set } c. d * (-c' / b')^2 + e * (-c' / b') + f \leq 0) \wedge$   
 $(\forall (d, e, f) \in \text{set } d. d * (-c' / b')^2 + e * (-c' / b') + f \neq 0)$   
 $\vee$   
 $a' \neq 0 \wedge$   
 $-b'^2 + 4 * a' * c' \leq 0 \wedge$   
 $((\forall (d, e, f) \in \text{set } a.$   
 $d * ((-b' + 1 * \text{sqrt}(b'^2 - 4 * a' * c')) / (2 * a'))^2 +$   
 $e * ((-b' + 1 * \text{sqrt}(b'^2 - 4 * a' * c')) / (2 * a')) +$   
 $f = 0) \wedge$   
 $(\forall (d, e, f) \in \text{set } b.$   
 $d * ((-b' + 1 * \text{sqrt}(b'^2 - 4 * a' * c')) / (2 * a'))^2 +$   
 $e * ((-b' + 1 * \text{sqrt}(b'^2 - 4 * a' * c')) / (2 * a')) +$   
 $f < 0) \wedge$   
 $(\forall (d, e, f) \in \text{set } c.$   
 $d * ((-b' + 1 * \text{sqrt}(b'^2 - 4 * a' * c')) / (2 * a'))^2 +$   
 $e * ((-b' + 1 * \text{sqrt}(b'^2 - 4 * a' * c')) / (2 * a')) +$   
 $f \leq 0) \wedge$   
 $(\forall (d, e, f) \in \text{set } d.$   
 $d * ((-b' + 1 * \text{sqrt}(b'^2 - 4 * a' * c')) / (2 * a'))^2 +$   
 $e * ((-b' + 1 * \text{sqrt}(b'^2 - 4 * a' * c')) / (2 * a')) +$   
 $f \neq 0)$   
 $\vee$   
 $(\forall (d, e, f) \in \text{set } a.$   
 $d * ((-b' + -1 * \text{sqrt}(b'^2 - 4 * a' * c')) / (2 * a'))^2 +$   
 $e * ((-b' + -1 * \text{sqrt}(b'^2 - 4 * a' * c')) / (2 * a')) +$   
 $f = 0) \wedge$   
 $(\forall (d, e, f) \in \text{set } b.$

$$\begin{aligned}
& d * ((-b' + -1 * \text{sqrt}(b'^2 - 4 * a' * c')) / (2 * a'))^2 + \\
& e * ((-b' + -1 * \text{sqrt}(b'^2 - 4 * a' * c')) / (2 * a')) + \\
& f < 0) \wedge \\
& (\forall (d, e, f) \in \text{set } c. \\
& \quad d * ((-b' + -1 * \text{sqrt}(b'^2 - 4 * a' * c')) / (2 * a'))^2 + \\
& \quad e * ((-b' + -1 * \text{sqrt}(b'^2 - 4 * a' * c')) / (2 * a')) + \\
& \quad f \leq 0) \wedge \\
& (\forall (d, e, f) \in \text{set } d. \\
& \quad d * ((-b' + -1 * \text{sqrt}(b'^2 - 4 * a' * c')) / (2 * a'))^2 + \\
& \quad e * ((-b' + -1 * \text{sqrt}(b'^2 - 4 * a' * c')) / (2 * a')) + \\
& \quad f \neq 0))) \\
\vee \\
& (\exists (a', b', c') \in \text{set } b. \\
& \quad (a' = 0 \wedge b' \neq 0) \wedge \\
& \quad (\forall (d, e, f) \in \text{set } a. \\
& \quad \quad \exists y' > -c' / b'. \forall x \in \{-c' / b' < ..y'\}. d * x^2 + e * x + f = 0) \wedge \\
& \quad (\forall (d, e, f) \in \text{set } b. \\
& \quad \quad \exists y' > -c' / b'. \forall x \in \{-c' / b' < ..y'\}. d * x^2 + e * x + f < 0) \wedge \\
& \quad (\forall (d, e, f) \in \text{set } c. \\
& \quad \quad \exists y' > -c' / b'. \forall x \in \{-c' / b' < ..y'\}. d * x^2 + e * x + f \leq 0) \wedge \\
& \quad (\forall (d, e, f) \in \text{set } d. \\
& \quad \quad \exists y' > -c' / b'. \forall x \in \{-c' / b' < ..y'\}. d * x^2 + e * x + f \neq 0) \\
& \quad \vee \\
& \quad a' \neq 0 \wedge \\
& \quad -b'^2 + 4 * a' * c' \leq 0 \wedge \\
& \quad ((\forall (d, e, f) \in \text{set } a. \\
& \quad \quad \exists y' > (-b' + 1 * \text{sqrt}(b'^2 - 4 * a' * c')) / (2 * a'). \\
& \quad \quad \forall x \in \{(-b' + 1 * \text{sqrt}(b'^2 - 4 * a' * c')) / (2 * a') < ..y'\}. \\
& \quad \quad \quad d * x^2 + e * x + f = 0) \wedge \\
& \quad (\forall (d, e, f) \in \text{set } b. \\
& \quad \quad \exists y' > (-b' + 1 * \text{sqrt}(b'^2 - 4 * a' * c')) / (2 * a'). \\
& \quad \quad \forall x \in \{(-b' + 1 * \text{sqrt}(b'^2 - 4 * a' * c')) / (2 * a') < ..y'\}. \\
& \quad \quad \quad d * x^2 + e * x + f < 0) \wedge \\
& \quad (\forall (d, e, f) \in \text{set } c. \\
& \quad \quad \exists y' > (-b' + 1 * \text{sqrt}(b'^2 - 4 * a' * c')) / (2 * a'). \\
& \quad \quad \forall x \in \{(-b' + 1 * \text{sqrt}(b'^2 - 4 * a' * c')) / (2 * a') < ..y'\}. \\
& \quad \quad \quad d * x^2 + e * x + f \leq 0) \wedge \\
& \quad (\forall (d, e, f) \in \text{set } d. \\
& \quad \quad \exists y' > (-b' + 1 * \text{sqrt}(b'^2 - 4 * a' * c')) / (2 * a'). \\
& \quad \quad \forall x \in \{(-b' + 1 * \text{sqrt}(b'^2 - 4 * a' * c')) / (2 * a') < ..y'\}. \\
& \quad \quad \quad d * x^2 + e * x + f \neq 0) \\
& \quad \vee \\
& \quad (\forall (d, e, f) \in \text{set } a. \\
& \quad \quad \exists y' > (-b' + -1 * \text{sqrt}(b'^2 - 4 * a' * c')) / (2 * a'). \\
& \quad \quad \forall x \in \{(-b' + -1 * \text{sqrt}(b'^2 - 4 * a' * c')) / (2 * a') < ..y'\}. \\
& \quad \quad \quad d * x^2 + e * x + f = 0) \wedge \\
& \quad (\forall (d, e, f) \in \text{set } b. \\
& \quad \quad \exists y' > (-b' + -1 * \text{sqrt}(b'^2 - 4 * a' * c')) / (2 * a'). \\
& \quad \quad \forall x \in \{(-b' + -1 * \text{sqrt}(b'^2 - 4 * a' * c')) / (2 * a') < ..y'\}.
\end{aligned}$$

$$\begin{aligned}
& d * x^2 + e * x + f < 0) \wedge \\
& (\forall (d, e, f) \in \text{set } c. \\
& \quad \exists y' > (-b' + -1 * \text{sqrt}(b'^2 - 4 * a' * c')) / (2 * a'). \\
& \quad \forall x \in \{(-b' + -1 * \text{sqrt}(b'^2 - 4 * a' * c')) / (2 * a') < .. y'\}. \\
& \quad d * x^2 + e * x + f \leq 0) \wedge \\
& (\forall (d, e, f) \in \text{set } d. \\
& \quad \exists y' > (-b' + -1 * \text{sqrt}(b'^2 - 4 * a' * c')) / (2 * a'). \\
& \quad \forall x \in \{(-b' + -1 * \text{sqrt}(b'^2 - 4 * a' * c')) / (2 * a') < .. y'\}. \\
& \quad d * x^2 + e * x + f \neq 0))
\end{aligned}$$

∨

$$\begin{aligned}
& (\exists (a', b', c') \in \text{set } c. \\
& \quad (a' = 0 \wedge b' \neq 0) \wedge \\
& \quad (\forall (d, e, f) \in \text{set } a. d * (-c' / b')^2 + e * (-c' / b') + f = 0) \wedge \\
& \quad (\forall (d, e, f) \in \text{set } b. d * (-c' / b')^2 + e * (-c' / b') + f < 0) \wedge \\
& \quad (\forall (d, e, f) \in \text{set } c. d * (-c' / b')^2 + e * (-c' / b') + f \leq 0) \wedge \\
& \quad (\forall (d, e, f) \in \text{set } d. d * (-c' / b')^2 + e * (-c' / b') + f \neq 0) \vee \\
& \quad a' \neq 0 \wedge \\
& \quad -b'^2 + 4 * a' * c' \leq 0 \wedge \\
& \quad ((\forall (d, e, f) \in \text{set } a. \\
& \quad \quad d * ((-b' + 1 * \text{sqrt}(b'^2 - 4 * a' * c')) / (2 * a'))^2 + \\
& \quad \quad e * ((-b' + 1 * \text{sqrt}(b'^2 - 4 * a' * c')) / (2 * a')) + \\
& \quad \quad f = 0) \wedge \\
& \quad (\forall (d, e, f) \in \text{set } b. \\
& \quad \quad d * ((-b' + 1 * \text{sqrt}(b'^2 - 4 * a' * c')) / (2 * a'))^2 + \\
& \quad \quad e * ((-b' + 1 * \text{sqrt}(b'^2 - 4 * a' * c')) / (2 * a')) + \\
& \quad \quad f < 0) \wedge \\
& \quad (\forall (d, e, f) \in \text{set } c. \\
& \quad \quad d * ((-b' + 1 * \text{sqrt}(b'^2 - 4 * a' * c')) / (2 * a'))^2 + \\
& \quad \quad e * ((-b' + 1 * \text{sqrt}(b'^2 - 4 * a' * c')) / (2 * a')) + \\
& \quad \quad f \leq 0) \wedge \\
& \quad (\forall (d, e, f) \in \text{set } d. \\
& \quad \quad d * ((-b' + 1 * \text{sqrt}(b'^2 - 4 * a' * c')) / (2 * a'))^2 + \\
& \quad \quad e * ((-b' + 1 * \text{sqrt}(b'^2 - 4 * a' * c')) / (2 * a')) + \\
& \quad \quad f \neq 0)
\end{aligned}$$

∨

$$\begin{aligned}
& (\forall (d, e, f) \in \text{set } a. \\
& \quad d * ((-b' + -1 * \text{sqrt}(b'^2 - 4 * a' * c')) / (2 * a'))^2 + \\
& \quad e * ((-b' + -1 * \text{sqrt}(b'^2 - 4 * a' * c')) / (2 * a')) + \\
& \quad f = 0) \wedge \\
& (\forall (d, e, f) \in \text{set } b. \\
& \quad d * ((-b' + -1 * \text{sqrt}(b'^2 - 4 * a' * c')) / (2 * a'))^2 + \\
& \quad e * ((-b' + -1 * \text{sqrt}(b'^2 - 4 * a' * c')) / (2 * a')) + \\
& \quad f < 0) \wedge \\
& (\forall (d, e, f) \in \text{set } c. \\
& \quad d * ((-b' + -1 * \text{sqrt}(b'^2 - 4 * a' * c')) / (2 * a'))^2 + \\
& \quad e * ((-b' + -1 * \text{sqrt}(b'^2 - 4 * a' * c')) / (2 * a')) + \\
& \quad f \leq 0) \wedge \\
& (\forall (d, e, f) \in \text{set } d. \\
& \quad d * ((-b' + -1 * \text{sqrt}(b'^2 - 4 * a' * c')) / (2 * a'))^2 +
\end{aligned}$$

$$\begin{aligned}
& e * ((- b' + - 1 * \text{sqrt}(b'^2 - 4 * a' * c')) / (2 * a') + \\
& f \neq 0))) \\
\vee \\
& (\exists (a', b', c') \in \text{set } d. \\
& (a' = 0 \wedge b' \neq 0) \wedge \\
& (\forall (d, e, f) \in \text{set } a. \\
& \exists y' > - c' / b'. \forall x \in \{- c' / b' <.. y'\}. d * x^2 + e * x + f = 0) \wedge \\
& (\forall (d, e, f) \in \text{set } b. \\
& \exists y' > - c' / b'. \forall x \in \{- c' / b' <.. y'\}. d * x^2 + e * x + f < 0) \wedge \\
& (\forall (d, e, f) \in \text{set } c. \\
& \exists y' > - c' / b'. \forall x \in \{- c' / b' <.. y'\}. d * x^2 + e * x + f \leq 0) \wedge \\
& (\forall (d, e, f) \in \text{set } d. \\
& \exists y' > - c' / b'. \forall x \in \{- c' / b' <.. y'\}. d * x^2 + e * x + f \neq 0) \\
\vee \\
& a' \neq 0 \wedge \\
& - b'^2 + 4 * a' * c' \leq 0 \wedge \\
& ((\forall (d, e, f) \in \text{set } a. \\
& \exists y' > (- b' + 1 * \text{sqrt}(b'^2 - 4 * a' * c')) / (2 * a'). \\
& \forall x \in \{(- b' + 1 * \text{sqrt}(b'^2 - 4 * a' * c')) / (2 * a') <.. y'\}. \\
& d * x^2 + e * x + f = 0) \wedge \\
& (\forall (d, e, f) \in \text{set } b. \\
& \exists y' > (- b' + 1 * \text{sqrt}(b'^2 - 4 * a' * c')) / (2 * a'). \\
& \forall x \in \{(- b' + 1 * \text{sqrt}(b'^2 - 4 * a' * c')) / (2 * a') <.. y'\}. \\
& d * x^2 + e * x + f < 0) \wedge \\
& (\forall (d, e, f) \in \text{set } c. \\
& \exists y' > (- b' + 1 * \text{sqrt}(b'^2 - 4 * a' * c')) / (2 * a'). \\
& \forall x \in \{(- b' + 1 * \text{sqrt}(b'^2 - 4 * a' * c')) / (2 * a') <.. y'\}. \\
& d * x^2 + e * x + f \leq 0) \wedge \\
& (\forall (d, e, f) \in \text{set } d. \\
& \exists y' > (- b' + 1 * \text{sqrt}(b'^2 - 4 * a' * c')) / (2 * a'). \\
& \forall x \in \{(- b' + 1 * \text{sqrt}(b'^2 - 4 * a' * c')) / (2 * a') <.. y'\}. \\
& d * x^2 + e * x + f \neq 0) \\
\vee \\
& (\forall (d, e, f) \in \text{set } a. \\
& \exists y' > (- b' + - 1 * \text{sqrt}(b'^2 - 4 * a' * c')) / (2 * a'). \\
& \forall x \in \{(- b' + - 1 * \text{sqrt}(b'^2 - 4 * a' * c')) / (2 * a') <.. y'\}. \\
& d * x^2 + e * x + f = 0) \wedge \\
& (\forall (d, e, f) \in \text{set } b. \\
& \exists y' > (- b' + - 1 * \text{sqrt}(b'^2 - 4 * a' * c')) / (2 * a'). \\
& \forall x \in \{(- b' + - 1 * \text{sqrt}(b'^2 - 4 * a' * c')) / (2 * a') <.. y'\}. \\
& d * x^2 + e * x + f < 0) \wedge \\
& (\forall (d, e, f) \in \text{set } c. \\
& \exists y' > (- b' + - 1 * \text{sqrt}(b'^2 - 4 * a' * c')) / (2 * a'). \\
& \forall x \in \{(- b' + - 1 * \text{sqrt}(b'^2 - 4 * a' * c')) / (2 * a') <.. y'\}. \\
& d * x^2 + e * x + f \leq 0) \wedge \\
& (\forall (d, e, f) \in \text{set } d. \\
& \exists y' > (- b' + - 1 * \text{sqrt}(b'^2 - 4 * a' * c')) / (2 * a'). \\
& \forall x \in \{(- b' + - 1 * \text{sqrt}(b'^2 - 4 * a' * c')) / (2 * a') <.. y'\}. \\
& d * x^2 + e * x + f \neq 0))))))
\end{aligned}$$

shows  $(\exists x. (\forall (a, b, c) \in \text{set } a. a * x^2 + b * x + c = 0) \wedge$   
 $(\forall (a, b, c) \in \text{set } b. a * x^2 + b * x + c < 0) \wedge$   
 $(\forall (a, b, c) \in \text{set } c. a * x^2 + b * x + c \leq 0) \wedge$   
 $(\forall (a, b, c) \in \text{set } d. a * x^2 + b * x + c \neq 0))$

$\langle \text{proof} \rangle$

## 2.9 General QE lemmas

**lemma** *qe*:  $(\exists x. (\forall (a, b, c) \in \text{set } a. a * x^2 + b * x + c = 0) \wedge$   
 $(\forall (a, b, c) \in \text{set } b. a * x^2 + b * x + c < 0) \wedge$   
 $(\forall (a, b, c) \in \text{set } c. a * x^2 + b * x + c \leq 0) \wedge$   
 $(\forall (a, b, c) \in \text{set } d. a * x^2 + b * x + c \neq 0)) =$   
 $((\forall (a, b, c) \in \text{set } a. a = 0 \wedge b = 0 \wedge c = 0) \wedge$   
 $(\forall (a, b, c) \in \text{set } b. \exists x. \forall y < x. a * y^2 + b * y + c < 0) \wedge$   
 $(\forall (a, b, c) \in \text{set } c. \exists x. \forall y < x. a * y^2 + b * y + c \leq 0) \wedge$   
 $(\forall (a, b, c) \in \text{set } d. \exists x. \forall y < x. a * y^2 + b * y + c \neq 0) \vee$   
 $(\exists (a', b', c') \in \text{set } a.$   
 $(a' = 0 \wedge b' \neq 0) \wedge$   
 $(\forall (d, e, f) \in \text{set } a. d * (-c' / b')^2 + e * (-c' / b') + f = 0) \wedge$   
 $(\forall (d, e, f) \in \text{set } b. d * (-c' / b')^2 + e * (-c' / b') + f < 0) \wedge$   
 $(\forall (d, e, f) \in \text{set } c. d * (-c' / b')^2 + e * (-c' / b') + f \leq 0) \wedge$   
 $(\forall (d, e, f) \in \text{set } d. d * (-c' / b')^2 + e * (-c' / b') + f \neq 0) \vee$   
 $a' \neq 0 \wedge$   
 $-b'^2 + 4 * a' * c' \leq 0 \wedge$   
 $(\forall (d, e, f) \in \text{set } a.$   
 $d * ((-b' + 1 * \text{sqrt}(b'^2 - 4 * a' * c')) / (2 * a'))^2 +$   
 $e * ((-b' + 1 * \text{sqrt}(b'^2 - 4 * a' * c')) / (2 * a')) +$   
 $f =$   
 $0) \wedge$   
 $(\forall (d, e, f) \in \text{set } b.$   
 $d * ((-b' + 1 * \text{sqrt}(b'^2 - 4 * a' * c')) / (2 * a'))^2 +$   
 $e * ((-b' + 1 * \text{sqrt}(b'^2 - 4 * a' * c')) / (2 * a')) +$   
 $f$   
 $< 0) \wedge$   
 $(\forall (d, e, f) \in \text{set } c.$   
 $d * ((-b' + 1 * \text{sqrt}(b'^2 - 4 * a' * c')) / (2 * a'))^2 +$   
 $e * ((-b' + 1 * \text{sqrt}(b'^2 - 4 * a' * c')) / (2 * a')) +$   
 $f$   
 $\leq 0) \wedge$   
 $(\forall (d, e, f) \in \text{set } d.$   
 $d * ((-b' + 1 * \text{sqrt}(b'^2 - 4 * a' * c')) / (2 * a'))^2 +$   
 $e * ((-b' + 1 * \text{sqrt}(b'^2 - 4 * a' * c')) / (2 * a')) +$   
 $f \neq$   
 $0) \vee$   
 $(\forall (d, e, f) \in \text{set } a.$   
 $d * ((-b' + -1 * \text{sqrt}(b'^2 - 4 * a' * c')) / (2 * a'))^2 +$   
 $e * ((-b' + -1 * \text{sqrt}(b'^2 - 4 * a' * c')) / (2 * a')) +$   
 $f =$   
 $0) \wedge$

$$\begin{aligned}
& (\forall (d, e, f) \in \text{set } b. \\
& \quad d * ((- b' + - 1 * \text{sqrt} (b'^2 - 4 * a' * c')) / (2 * a'))^2 + \\
& \quad e * ((- b' + - 1 * \text{sqrt} (b'^2 - 4 * a' * c')) / (2 * a')) + \\
& \quad f \\
& \quad < 0) \wedge \\
& (\forall (d, e, f) \in \text{set } c. \\
& \quad d * ((- b' + - 1 * \text{sqrt} (b'^2 - 4 * a' * c')) / (2 * a'))^2 + \\
& \quad e * ((- b' + - 1 * \text{sqrt} (b'^2 - 4 * a' * c')) / (2 * a')) + \\
& \quad f \\
& \quad \leq 0) \wedge \\
& (\forall (d, e, f) \in \text{set } d. \\
& \quad d * ((- b' + - 1 * \text{sqrt} (b'^2 - 4 * a' * c')) / (2 * a'))^2 + \\
& \quad e * ((- b' + - 1 * \text{sqrt} (b'^2 - 4 * a' * c')) / (2 * a')) + \\
& \quad f \neq \\
& \quad 0)) \vee \\
& (\exists (a', b', c') \in \text{set } b. \\
& \quad (a' = 0 \wedge b' \neq 0) \wedge \\
& \quad (\forall (d, e, f) \in \text{set } a. \\
& \quad \quad \exists y' > - c' / b'. \forall x \in \{- c' / b' < .. y'\}. d * x^2 + e * x + f = 0) \wedge \\
& \quad (\forall (d, e, f) \in \text{set } b. \\
& \quad \quad \exists y' > - c' / b'. \forall x \in \{- c' / b' < .. y'\}. d * x^2 + e * x + f < 0) \wedge \\
& \quad (\forall (d, e, f) \in \text{set } c. \\
& \quad \quad \exists y' > - c' / b'. \forall x \in \{- c' / b' < .. y'\}. d * x^2 + e * x + f \leq 0) \wedge \\
& \quad (\forall (d, e, f) \in \text{set } d. \\
& \quad \quad \exists y' > - c' / b'. \forall x \in \{- c' / b' < .. y'\}. d * x^2 + e * x + f \neq 0) \vee \\
& \quad a' \neq 0 \wedge \\
& \quad - b'^2 + 4 * a' * c' \leq 0 \wedge \\
& \quad (\forall (d, e, f) \in \text{set } a. \\
& \quad \quad \exists y' > (- b' + 1 * \text{sqrt} (b'^2 - 4 * a' * c')) / (2 * a'). \\
& \quad \quad \forall x \in \{(- b' + 1 * \text{sqrt} (b'^2 - 4 * a' * c')) / (2 * a') < .. y'\}. \\
& \quad \quad d * x^2 + e * x + f = 0) \wedge \\
& \quad (\forall (d, e, f) \in \text{set } b. \\
& \quad \quad \exists y' > (- b' + 1 * \text{sqrt} (b'^2 - 4 * a' * c')) / (2 * a'). \\
& \quad \quad \forall x \in \{(- b' + 1 * \text{sqrt} (b'^2 - 4 * a' * c')) / (2 * a') < .. y'\}. \\
& \quad \quad d * x^2 + e * x + f < 0) \wedge \\
& \quad (\forall (d, e, f) \in \text{set } c. \\
& \quad \quad \exists y' > (- b' + 1 * \text{sqrt} (b'^2 - 4 * a' * c')) / (2 * a'). \\
& \quad \quad \forall x \in \{(- b' + 1 * \text{sqrt} (b'^2 - 4 * a' * c')) / (2 * a') < .. y'\}. \\
& \quad \quad d * x^2 + e * x + f \leq 0) \wedge \\
& \quad (\forall (d, e, f) \in \text{set } d. \\
& \quad \quad \exists y' > (- b' + 1 * \text{sqrt} (b'^2 - 4 * a' * c')) / (2 * a'). \\
& \quad \quad \forall x \in \{(- b' + 1 * \text{sqrt} (b'^2 - 4 * a' * c')) / (2 * a') < .. y'\}. \\
& \quad \quad d * x^2 + e * x + f \neq 0) \vee \\
& \quad (\forall (d, e, f) \in \text{set } a. \\
& \quad \quad \exists y' > (- b' + - 1 * \text{sqrt} (b'^2 - 4 * a' * c')) / (2 * a'). \\
& \quad \quad \forall x \in \{(- b' + - 1 * \text{sqrt} (b'^2 - 4 * a' * c')) / (2 * a') < .. y'\}. \\
& \quad \quad d * x^2 + e * x + f = 0) \wedge \\
& \quad (\forall (d, e, f) \in \text{set } b. \\
& \quad \quad \exists y' > (- b' + - 1 * \text{sqrt} (b'^2 - 4 * a' * c')) / (2 * a').
\end{aligned}$$

$$\begin{aligned}
& \forall x \in \{(-b' + -1 * \text{sqrt}(b'^2 - 4 * a' * c')) / (2 * a') < ..y'\}. \\
& \quad d * x^2 + e * x + f < 0) \wedge \\
& (\forall (d, e, f) \in \text{set } c. \\
& \quad \exists y' > (-b' + -1 * \text{sqrt}(b'^2 - 4 * a' * c')) / (2 * a'). \\
& \quad \forall x \in \{(-b' + -1 * \text{sqrt}(b'^2 - 4 * a' * c')) / (2 * a') < ..y'\}. \\
& \quad \quad d * x^2 + e * x + f \leq 0) \wedge \\
& (\forall (d, e, f) \in \text{set } d. \\
& \quad \exists y' > (-b' + -1 * \text{sqrt}(b'^2 - 4 * a' * c')) / (2 * a'). \\
& \quad \forall x \in \{(-b' + -1 * \text{sqrt}(b'^2 - 4 * a' * c')) / (2 * a') < ..y'\}. \\
& \quad \quad d * x^2 + e * x + f \neq 0)) \vee \\
& (\exists (a', b', c') \in \text{set } c. \\
& \quad (a' = 0 \wedge b' \neq 0) \wedge \\
& \quad (\forall (d, e, f) \in \text{set } a. d * (-c' / b')^2 + e * (-c' / b') + f = 0) \wedge \\
& \quad (\forall (d, e, f) \in \text{set } b. d * (-c' / b')^2 + e * (-c' / b') + f < 0) \wedge \\
& \quad (\forall (d, e, f) \in \text{set } c. d * (-c' / b')^2 + e * (-c' / b') + f \leq 0) \wedge \\
& \quad (\forall (d, e, f) \in \text{set } d. d * (-c' / b')^2 + e * (-c' / b') + f \neq 0) \vee \\
& \quad a' \neq 0 \wedge \\
& \quad -b'^2 + 4 * a' * c' \leq 0) \wedge \\
& (\forall (d, e, f) \in \text{set } a. \\
& \quad d * ((-b' + 1 * \text{sqrt}(b'^2 - 4 * a' * c')) / (2 * a'))^2 + \\
& \quad e * ((-b' + 1 * \text{sqrt}(b'^2 - 4 * a' * c')) / (2 * a')) + \\
& \quad f = \\
& \quad 0) \wedge \\
& (\forall (d, e, f) \in \text{set } b. \\
& \quad d * ((-b' + 1 * \text{sqrt}(b'^2 - 4 * a' * c')) / (2 * a'))^2 + \\
& \quad e * ((-b' + 1 * \text{sqrt}(b'^2 - 4 * a' * c')) / (2 * a')) + \\
& \quad f \\
& \quad < 0) \wedge \\
& (\forall (d, e, f) \in \text{set } c. \\
& \quad d * ((-b' + 1 * \text{sqrt}(b'^2 - 4 * a' * c')) / (2 * a'))^2 + \\
& \quad e * ((-b' + 1 * \text{sqrt}(b'^2 - 4 * a' * c')) / (2 * a')) + \\
& \quad f \\
& \quad \leq 0) \wedge \\
& (\forall (d, e, f) \in \text{set } d. \\
& \quad d * ((-b' + 1 * \text{sqrt}(b'^2 - 4 * a' * c')) / (2 * a'))^2 + \\
& \quad e * ((-b' + 1 * \text{sqrt}(b'^2 - 4 * a' * c')) / (2 * a')) + \\
& \quad f \neq \\
& \quad 0) \vee \\
& (\forall (d, e, f) \in \text{set } a. \\
& \quad d * ((-b' + -1 * \text{sqrt}(b'^2 - 4 * a' * c')) / (2 * a'))^2 + \\
& \quad e * ((-b' + -1 * \text{sqrt}(b'^2 - 4 * a' * c')) / (2 * a')) + \\
& \quad f = \\
& \quad 0) \wedge \\
& (\forall (d, e, f) \in \text{set } b. \\
& \quad d * ((-b' + -1 * \text{sqrt}(b'^2 - 4 * a' * c')) / (2 * a'))^2 + \\
& \quad e * ((-b' + -1 * \text{sqrt}(b'^2 - 4 * a' * c')) / (2 * a')) + \\
& \quad f \\
& \quad < 0) \wedge \\
& (\forall (d, e, f) \in \text{set } c.
\end{aligned}$$

$$\begin{aligned}
& d * ((-b' + -1 * \text{sqrt}(b'^2 - 4 * a' * c')) / (2 * a'))^2 + \\
& e * ((-b' + -1 * \text{sqrt}(b'^2 - 4 * a' * c')) / (2 * a')) + \\
& f \\
& \leq 0) \wedge \\
& (\forall (d, e, f) \in \text{set } d. \\
& \quad d * ((-b' + -1 * \text{sqrt}(b'^2 - 4 * a' * c')) / (2 * a'))^2 + \\
& \quad e * ((-b' + -1 * \text{sqrt}(b'^2 - 4 * a' * c')) / (2 * a')) + \\
& \quad f \neq \\
& \quad 0)) \vee \\
& (\exists (a', b', c') \in \text{set } d. \\
& \quad (a' = 0 \wedge b' \neq 0) \wedge \\
& \quad (\forall (d, e, f) \in \text{set } a. \\
& \quad \quad \exists y' > -c' / b'. \forall x \in \{-c' / b' < ..y'\}. d * x^2 + e * x + f = 0) \wedge \\
& \quad (\forall (d, e, f) \in \text{set } b. \\
& \quad \quad \exists y' > -c' / b'. \forall x \in \{-c' / b' < ..y'\}. d * x^2 + e * x + f < 0) \wedge \\
& \quad (\forall (d, e, f) \in \text{set } c. \\
& \quad \quad \exists y' > -c' / b'. \forall x \in \{-c' / b' < ..y'\}. d * x^2 + e * x + f \leq 0) \wedge \\
& \quad (\forall (d, e, f) \in \text{set } d. \\
& \quad \quad \exists y' > -c' / b'. \forall x \in \{-c' / b' < ..y'\}. d * x^2 + e * x + f \neq 0) \vee \\
& \quad a' \neq 0 \wedge \\
& \quad -b'^2 + 4 * a' * c' \leq 0 \wedge \\
& \quad (\forall (d, e, f) \in \text{set } a. \\
& \quad \quad \exists y' > (-b' + 1 * \text{sqrt}(b'^2 - 4 * a' * c')) / (2 * a'). \\
& \quad \quad \forall x \in \{(-b' + 1 * \text{sqrt}(b'^2 - 4 * a' * c')) / (2 * a') < ..y'\}. \\
& \quad \quad \quad d * x^2 + e * x + f = 0) \wedge \\
& \quad (\forall (d, e, f) \in \text{set } b. \\
& \quad \quad \exists y' > (-b' + 1 * \text{sqrt}(b'^2 - 4 * a' * c')) / (2 * a'). \\
& \quad \quad \forall x \in \{(-b' + 1 * \text{sqrt}(b'^2 - 4 * a' * c')) / (2 * a') < ..y'\}. \\
& \quad \quad \quad d * x^2 + e * x + f < 0) \wedge \\
& \quad (\forall (d, e, f) \in \text{set } c. \\
& \quad \quad \exists y' > (-b' + 1 * \text{sqrt}(b'^2 - 4 * a' * c')) / (2 * a'). \\
& \quad \quad \forall x \in \{(-b' + 1 * \text{sqrt}(b'^2 - 4 * a' * c')) / (2 * a') < ..y'\}. \\
& \quad \quad \quad d * x^2 + e * x + f \leq 0) \wedge \\
& \quad (\forall (d, e, f) \in \text{set } d. \\
& \quad \quad \exists y' > (-b' + 1 * \text{sqrt}(b'^2 - 4 * a' * c')) / (2 * a'). \\
& \quad \quad \forall x \in \{(-b' + 1 * \text{sqrt}(b'^2 - 4 * a' * c')) / (2 * a') < ..y'\}. \\
& \quad \quad \quad d * x^2 + e * x + f \neq 0) \vee \\
& \quad (\forall (d, e, f) \in \text{set } a. \\
& \quad \quad \exists y' > (-b' + -1 * \text{sqrt}(b'^2 - 4 * a' * c')) / (2 * a'). \\
& \quad \quad \forall x \in \{(-b' + -1 * \text{sqrt}(b'^2 - 4 * a' * c')) / (2 * a') < ..y'\}. \\
& \quad \quad \quad d * x^2 + e * x + f = 0) \wedge \\
& \quad (\forall (d, e, f) \in \text{set } b. \\
& \quad \quad \exists y' > (-b' + -1 * \text{sqrt}(b'^2 - 4 * a' * c')) / (2 * a'). \\
& \quad \quad \forall x \in \{(-b' + -1 * \text{sqrt}(b'^2 - 4 * a' * c')) / (2 * a') < ..y'\}. \\
& \quad \quad \quad d * x^2 + e * x + f < 0) \wedge \\
& \quad (\forall (d, e, f) \in \text{set } c. \\
& \quad \quad \exists y' > (-b' + -1 * \text{sqrt}(b'^2 - 4 * a' * c')) / (2 * a'). \\
& \quad \quad \forall x \in \{(-b' + -1 * \text{sqrt}(b'^2 - 4 * a' * c')) / (2 * a') < ..y'\}. \\
& \quad \quad \quad d * x^2 + e * x + f \leq 0) \wedge
\end{aligned}$$

$(\forall (d, e, f) \in \text{set } d.$   
 $\exists y' > (-b' + -1 * \text{sqrt}(b'^2 - 4 * a' * c')) / (2 * a').$   
 $\forall x \in \{(-b' + -1 * \text{sqrt}(b'^2 - 4 * a' * c')) / (2 * a') < ..y'\}.$   
 $d * x^2 + e * x + f \neq 0))$

*<proof>*

**fun** eq-fun :: real  $\Rightarrow$  real  $\Rightarrow$  real  $\Rightarrow$  (real\*real\*real) list  $\Rightarrow$  (real\*real\*real) list  $\Rightarrow$   
(real\*real\*real) list  $\Rightarrow$  (real\*real\*real) list  $\Rightarrow$  bool **where**

eq-fun a' b' c' eq les leq neq = ((a' = 0  $\wedge$  b'  $\neq$  0)  $\wedge$

( $\forall a \in \text{set eq}.$

case a of (d, e, f)  $\Rightarrow$  d \* (-c' / b')<sup>2</sup> + e \* (-c' / b') + f = 0)  $\wedge$

( $\forall a \in \text{set les}.$

case a of (d, e, f)  $\Rightarrow$  d \* (-c' / b')<sup>2</sup> + e \* (-c' / b') + f < 0)  $\wedge$

( $\forall a \in \text{set leq}.$

case a of (d, e, f)  $\Rightarrow$  d \* (-c' / b')<sup>2</sup> + e \* (-c' / b') + f  $\leq$  0)  $\wedge$

( $\forall a \in \text{set neq}.$

case a of (d, e, f)  $\Rightarrow$  d \* (-c' / b')<sup>2</sup> + e \* (-c' / b') + f  $\neq$  0)  $\vee$

a'  $\neq$  0  $\wedge$

- b'^2 + 4 \* a' \* c'  $\leq$  0  $\wedge$

(( $\forall a \in \text{set eq}.$

case a of

(d, e, f)  $\Rightarrow$

d \* ((-b' + 1 \* sqrt(b'^2 - 4 \* a' \* c')) / (2 \* a'))<sup>2</sup> +

e \* ((-b' + 1 \* sqrt(b'^2 - 4 \* a' \* c')) / (2 \* a')) +

f =

0)  $\wedge$

( $\forall a \in \text{set les}.$

case a of

(d, e, f)  $\Rightarrow$

d \* ((-b' + 1 \* sqrt(b'^2 - 4 \* a' \* c')) / (2 \* a'))<sup>2</sup> +

e \* ((-b' + 1 \* sqrt(b'^2 - 4 \* a' \* c')) / (2 \* a')) +

f

< 0)  $\wedge$

( $\forall a \in \text{set leq}.$

case a of

(d, e, f)  $\Rightarrow$

d \* ((-b' + 1 \* sqrt(b'^2 - 4 \* a' \* c')) / (2 \* a'))<sup>2</sup> +

e \* ((-b' + 1 \* sqrt(b'^2 - 4 \* a' \* c')) / (2 \* a')) +

f

$\leq$  0)  $\wedge$

( $\forall a \in \text{set neq}.$

case a of

(d, e, f)  $\Rightarrow$

d \* ((-b' + 1 \* sqrt(b'^2 - 4 \* a' \* c')) / (2 \* a'))<sup>2</sup> +

e \* ((-b' + 1 \* sqrt(b'^2 - 4 \* a' \* c')) / (2 \* a')) +

f  $\neq$

0)  $\vee$

( $\forall a \in \text{set eq}.$

*case a of*  
 $(d, e, f) \Rightarrow$   
 $d * ((- b' + - 1 * \text{sqrt} (b'^2 - 4 * a' * c')) / (2 * a'))^2 +$   
 $e * ((- b' + - 1 * \text{sqrt} (b'^2 - 4 * a' * c')) / (2 * a')) +$   
 $f =$   
 $0) \wedge$   
 $(\forall a \in \text{set les.}$   
*case a of*  
 $(d, e, f) \Rightarrow$   
 $d * ((- b' + - 1 * \text{sqrt} (b'^2 - 4 * a' * c')) / (2 * a'))^2 +$   
 $e * ((- b' + - 1 * \text{sqrt} (b'^2 - 4 * a' * c')) / (2 * a')) +$   
 $f$   
 $< 0) \wedge$   
 $(\forall a \in \text{set leq.}$   
*case a of*  
 $(d, e, f) \Rightarrow$   
 $d * ((- b' + - 1 * \text{sqrt} (b'^2 - 4 * a' * c')) / (2 * a'))^2 +$   
 $e * ((- b' + - 1 * \text{sqrt} (b'^2 - 4 * a' * c')) / (2 * a')) +$   
 $f$   
 $\leq 0) \wedge$   
 $(\forall a \in \text{set neq.}$   
*case a of*  
 $(d, e, f) \Rightarrow$   
 $d * ((- b' + - 1 * \text{sqrt} (b'^2 - 4 * a' * c')) / (2 * a'))^2 +$   
 $e * ((- b' + - 1 * \text{sqrt} (b'^2 - 4 * a' * c')) / (2 * a')) +$   
 $f \neq$   
 $0)))$

**fun** *les-fun* :: *real*  $\Rightarrow$  *real*  $\Rightarrow$  *real*  $\Rightarrow$  (*real*\**real*\**real*) *list*  $\Rightarrow$  (*real*\**real*\**real*) *list*  $\Rightarrow$   
(*real*\**real*\**real*) *list*  $\Rightarrow$  (*real*\**real*\**real*) *list*  $\Rightarrow$  *bool* **where**

*les-fun* *a' b' c' eq les leq neq* =  $((a' = 0 \wedge b' \neq 0) \wedge$

$(\forall (d, e, f) \in \text{set eq.}$

$\exists y' > -c' / b'. \forall x \in \{-c' / b' <..y'\}. d * x^2 + e * x + f = 0) \wedge$

$(\forall (d, e, f) \in \text{set les.}$

$\exists y' > -c' / b'. \forall x \in \{-c' / b' <..y'\}. d * x^2 + e * x + f < 0) \wedge$

$(\forall (d, e, f) \in \text{set leq.}$

$\exists y' > -c' / b'. \forall x \in \{-c' / b' <..y'\}. d * x^2 + e * x + f \leq 0) \wedge$

$(\forall (d, e, f) \in \text{set neq.}$

$\exists y' > -c' / b'. \forall x \in \{-c' / b' <..y'\}. d * x^2 + e * x + f \neq 0) \vee$

$a' \neq 0 \wedge$

$-b'^2 + 4 * a' * c' \leq 0 \wedge$

$(\forall (d, e, f) \in \text{set eq.}$

$\exists y' > (-b' + 1 * \text{sqrt} (b'^2 - 4 * a' * c')) / (2 * a').$

$\forall x \in \{(-b' + 1 * \text{sqrt} (b'^2 - 4 * a' * c')) / (2 * a') <..y'\}.$

$d * x^2 + e * x + f = 0) \wedge$

$(\forall (d, e, f) \in \text{set les.}$

$\exists y' > (-b' + 1 * \text{sqrt} (b'^2 - 4 * a' * c')) / (2 * a').$

$\forall x \in \{(-b' + 1 * \text{sqrt} (b'^2 - 4 * a' * c')) / (2 * a') <..y'\}.$

$d * x^2 + e * x + f < 0) \wedge$

$$\begin{aligned}
& (\forall (d, e, f) \in \text{set leq.} \\
& \quad \exists y' > (-b' + 1 * \text{sqrt}(b'^2 - 4 * a' * c')) / (2 * a'). \\
& \quad \forall x \in \{(-b' + 1 * \text{sqrt}(b'^2 - 4 * a' * c')) / (2 * a') <..y'\}. \\
& \quad d * x^2 + e * x + f \leq 0) \wedge \\
& (\forall (d, e, f) \in \text{set neq.} \\
& \quad \exists y' > (-b' + 1 * \text{sqrt}(b'^2 - 4 * a' * c')) / (2 * a'). \\
& \quad \forall x \in \{(-b' + 1 * \text{sqrt}(b'^2 - 4 * a' * c')) / (2 * a') <..y'\}. \\
& \quad d * x^2 + e * x + f \neq 0) \vee \\
& (\forall (d, e, f) \in \text{set eq.} \\
& \quad \exists y' > (-b' + -1 * \text{sqrt}(b'^2 - 4 * a' * c')) / (2 * a'). \\
& \quad \forall x \in \{(-b' + -1 * \text{sqrt}(b'^2 - 4 * a' * c')) / (2 * a') <..y'\}. \\
& \quad d * x^2 + e * x + f = 0) \wedge \\
& (\forall (d, e, f) \in \text{set les.} \\
& \quad \exists y' > (-b' + -1 * \text{sqrt}(b'^2 - 4 * a' * c')) / (2 * a'). \\
& \quad \forall x \in \{(-b' + -1 * \text{sqrt}(b'^2 - 4 * a' * c')) / (2 * a') <..y'\}. \\
& \quad d * x^2 + e * x + f < 0) \wedge \\
& (\forall (d, e, f) \in \text{set leq.} \\
& \quad \exists y' > (-b' + -1 * \text{sqrt}(b'^2 - 4 * a' * c')) / (2 * a'). \\
& \quad \forall x \in \{(-b' + -1 * \text{sqrt}(b'^2 - 4 * a' * c')) / (2 * a') <..y'\}. \\
& \quad d * x^2 + e * x + f \leq 0) \wedge \\
& (\forall (d, e, f) \in \text{set neq.} \\
& \quad \exists y' > (-b' + -1 * \text{sqrt}(b'^2 - 4 * a' * c')) / (2 * a'). \\
& \quad \forall x \in \{(-b' + -1 * \text{sqrt}(b'^2 - 4 * a' * c')) / (2 * a') <..y'\}. \\
& \quad d * x^2 + e * x + f \neq 0))
\end{aligned}$$

**lemma general-qe'** :

$$\begin{aligned}
\text{assumes } F &= (\lambda x. \\
& (\forall (a, b, c) \in \text{set eq. } a * x^2 + b * x + c = 0) \wedge \\
& (\forall (a, b, c) \in \text{set les. } a * x^2 + b * x + c < 0) \wedge \\
& (\forall (a, b, c) \in \text{set leq. } a * x^2 + b * x + c \leq 0) \wedge \\
& (\forall (a, b, c) \in \text{set neq. } a * x^2 + b * x + c \neq 0))
\end{aligned}$$

$$\begin{aligned}
\text{assumes } F\varepsilon &= (\lambda r. \\
& (\forall (a, b, c) \in \text{set eq. } \exists y > r. \forall x \in \{r <..y\}. a * x^2 + b * x + c = 0) \wedge \\
& (\forall (a, b, c) \in \text{set les. } \exists y > r. \forall x \in \{r <..y\}. a * x^2 + b * x + c < 0) \wedge \\
& (\forall (a, b, c) \in \text{set leq. } \exists y > r. \forall x \in \{r <..y\}. a * x^2 + b * x + c \leq 0) \wedge \\
& (\forall (a, b, c) \in \text{set neq. } \exists y > r. \forall x \in \{r <..y\}. a * x^2 + b * x + c \neq 0) \\
& )
\end{aligned}$$

$$\begin{aligned}
\text{assumes } F_{inf} &= ( \\
& (\forall (a, b, c) \in \text{set eq. } \exists x. \forall y < x. a * y^2 + b * y + c = 0) \wedge \\
& (\forall (a, b, c) \in \text{set les. } \exists x. \forall y < x. a * y^2 + b * y + c < 0) \wedge \\
& (\forall (a, b, c) \in \text{set leq. } \exists x. \forall y < x. a * y^2 + b * y + c \leq 0) \wedge \\
& (\forall (a, b, c) \in \text{set neq. } \exists x. \forall y < x. a * y^2 + b * y + c \neq 0) \\
& )
\end{aligned}$$

$$\begin{aligned}
\text{assumes roots} &= (\lambda(a, b, c). \\
& \text{if } a=0 \wedge b \neq 0 \text{ then } \{-c/b\}::\text{real set else}
\end{aligned}$$

if  $a \neq 0 \wedge b^2 - 4 * a * c \geq 0$  then  $\{(-b + \text{sqrt}(b^2 - 4 * a * c)) / (2 * a)\} \cup \{(-b - \text{sqrt}(b^2 - 4 * a * c)) / (2 * a)\}$   
else  $\{\}$ )

**assumes**  $A = \bigcup(\text{roots } ' (set \text{ eq}))$   
**assumes**  $B = \bigcup(\text{roots } ' (set \text{ les}))$   
**assumes**  $C = \bigcup(\text{roots } ' (set \text{ leq}))$   
**assumes**  $D = \bigcup(\text{roots } ' (set \text{ neq}))$

**shows**  $(\exists x. F(x)) = (F_{inf} \vee (\exists r \in A. F r) \vee (\exists r \in B. F \varepsilon r) \vee (\exists r \in C. F r) \vee (\exists r \in D. F \varepsilon r))$   
 $\langle \text{proof} \rangle$

**lemma** *general-qe''* :

**assumes**  $S = \{ (=), (<), (\leq), (\neq) \}$   
**assumes**  $\text{finite } (X (=))$   
**assumes**  $\text{finite } (X (<))$   
**assumes**  $\text{finite } (X (\leq))$   
**assumes**  $\text{finite } (X (\neq))$   
**assumes**  $F = (\lambda x. \forall \text{rel} \in S. \forall (a, b, c) \in (X \text{ rel}). \text{rel } (a * x^2 + b * x + c) 0)$

**assumes**  $F \varepsilon = (\lambda r. \forall \text{rel} \in S. \forall (a, b, c) \in (X \text{ rel}). \exists y > r. \forall x \in \{r < .. y\}. \text{rel } (a * x^2 + b * x + c) 0)$

**assumes**  $F_{inf} = (\forall \text{rel} \in S. \forall (a, b, c) \in (X \text{ rel}). \exists x. \forall y < x. \text{rel } (a * y^2 + b * y + c) 0)$

**assumes**  $\text{roots} = (\lambda(a, b, c). \text{if } a = 0 \wedge b \neq 0 \text{ then } \{-c/b\} :: \text{real set else if } a \neq 0 \wedge b^2 - 4 * a * c \geq 0 \text{ then } \{(-b + \text{sqrt}(b^2 - 4 * a * c)) / (2 * a)\} \cup \{(-b - \text{sqrt}(b^2 - 4 * a * c)) / (2 * a)\} \text{ else } \{\})$

**assumes**  $A = \bigcup(\text{roots } ' ((X (=))))$   
**assumes**  $B = \bigcup(\text{roots } ' ((X (<))))$   
**assumes**  $C = \bigcup(\text{roots } ' ((X (\leq))))$   
**assumes**  $D = \bigcup(\text{roots } ' ((X (\neq))))$

**shows**  $(\exists x. F(x)) = (F_{inf} \vee (\exists r \in A. F r) \vee (\exists r \in B. F \varepsilon r) \vee (\exists r \in C. F r) \vee (\exists r \in D. F \varepsilon r))$   
 $\langle \text{proof} \rangle$

**theorem** *general-qe* :

**assumes**  $R = \{ (=), (<), (\leq), (\neq) \}$   
**assumes**  $\forall \text{rel} \in R. \text{finite } (Atoms \text{ rel})$

**assumes**  $F = (\lambda x. \forall \text{rel} \in R. \forall (a, b, c) \in (Atoms \text{ rel}). \text{rel } (a * x^2 + b * x + c) 0)$

**assumes**  $F \varepsilon = (\lambda r. \forall \text{rel} \in R. \forall (a, b, c) \in (Atoms \text{ rel}). \exists y > r. \forall x \in \{r < .. y\}. \text{rel } (a * x^2 + b * x + c) 0)$

0)

**assumes**  $F_{inf} = (\forall rel \in R. \forall (a, b, c) \in (Atoms\ rel). \exists x. \forall y < x. rel\ (a*y^2 + b*y + c)$   
0)

**assumes**  $roots = (\lambda(a, b, c).$   
   $if\ a=0 \wedge b \neq 0\ then\ \{-c/b\}\ else$   
   $if\ a \neq 0 \wedge b^2 - 4*a*c \geq 0\ then\ \{(-b + sqrt(b^2 - 4*a*c))/(2*a)\} \cup \{(-b - sqrt(b^2 - 4*a*c))/(2*a)\}$   
 $else\ \{\})$

**shows**  $(\exists x. F(x)) =$   
   $(F_{inf} \vee$   
   $(\exists r \in \bigcup (roots\ ' (Atoms\ (=) \cup Atoms\ (\leq))). F\ r) \vee$   
   $(\exists r \in \bigcup (roots\ ' (Atoms\ (<) \cup Atoms\ (\neq))). F\ \varepsilon\ r))$   
 $\langle proof \rangle$

**end**

### 3 Multivariate Polynomials Extension

**theory** *MPolyExtension*

**imports** *Polynomials.Polynomials Polynomials.MPoly-Type-Class-FMap*  
**begin**

#### 3.1 Definition Lifting

**lift-definition**  $coeff-code::'a::zero\ mpoly \Rightarrow (nat \Rightarrow_0\ nat) \Rightarrow 'a$  **is**  
 $lookup\ \langle proof \rangle$

**lemma**  $coeff-code[code]:\ coeff = coeff-code$   
 $\langle proof \rangle$

**lemma**  $coeff-transfer[transfer-rule]:$ — **TODO:** coeff should be defined via lifting,  
this gives us the illusion  
 $rel-fun\ cr-mpoly\ (=)\ lookup\ coeff\ \langle proof \rangle$

**lemmas**  $coeff-add = coeff-add[symmetric]$

**lemma**  $plus-monom-zero[simp]:\ p + MPoly-Type.monom\ m\ 0 = p$   
 $\langle proof \rangle$

**lift-definition**  $monomials::'a::zero\ mpoly \Rightarrow (nat \Rightarrow_0\ nat)\ set$  **is**  
 $Poly-Mapping.keys::((nat \Rightarrow_0\ nat) \Rightarrow_0\ 'a) \Rightarrow -\ set\ \langle proof \rangle$

**lemma**  $mpoly-induct\ [case-names\ monom\ sum]:$ — **TODO:** overwrites  $\llbracket \bigwedge m\ a. ?P$   
 $(monom\ m\ a); \bigwedge p1\ p2\ m\ a. \llbracket ?P\ p1; ?P\ p2; p2 = monom\ m\ a; m \notin keys\ (mapping-of$   
 $p1) \rrbracket \implies ?P\ (p1 + p2) \rrbracket \implies ?P\ p$

**assumes**  $monom:\bigwedge m\ a. P\ (MPoly-Type.monom\ m\ a)$   
**and**  $sum:(\bigwedge p1\ p2\ m\ a. P\ p1 \implies P\ p2 \implies p2 = (MPoly-Type.monom\ m\ a)$

$\implies m \notin \text{monomials } p1$   
 $\implies a \neq 0 \implies P (p1+p2)$   
**shows**  $P p$   $\langle \text{proof} \rangle$

**value**  $\text{monomials } ((\text{Var } 0 + \text{Const } (3::\text{int}) * \text{Var } 1) \wedge 2)$

**lemma** *Sum-any-lookup-times-eq*:

$(\sum k. ((\text{lookup } (x::'a \Rightarrow_0 ('b::\text{comm-semiring-1})) (k::'a)) * ((f:: 'a \Rightarrow ('b::\text{comm-semiring-1})) k))) = (\sum k \in \text{keys } x. (\text{lookup } x (k::'a)) * (f k))$   
 $x. (\text{lookup } x (k::'a)) * (f k)$   
 $\langle \text{proof} \rangle$

**lemma** *Prod-any-power-lookup-eq*:

$(\prod k::'a. f k \wedge \text{lookup } (x::'a \Rightarrow_0 \text{nat}) k) = (\prod k \in \text{keys } x. f k \wedge \text{lookup } x k)$   
 $\langle \text{proof} \rangle$

**lemma** *insertion-monom*:  $\text{insertion } i (\text{monom } m a) = a * (\prod k \in \text{keys } m. i k \wedge \text{lookup } m k)$   
 $\langle \text{proof} \rangle$

**lemma** *monomials-monom[simp]*:  $\text{monomials } (\text{monom } m a) = (\text{if } a = 0 \text{ then } \{\} \text{ else } \{m\})$   
 $\langle \text{proof} \rangle$

**lemma** *finite-monomials[simp]*:  $\text{finite } (\text{monomials } m)$   
 $\langle \text{proof} \rangle$

**lemma** *monomials-add-disjoint*:

$\text{monomials } (a + b) = \text{monomials } a \cup \text{monomials } b$  **if**  $\text{monomials } a \cap \text{monomials } b = \{\}$   
 $\langle \text{proof} \rangle$

**lemma** *coeff-monom[simp]*:  $\text{coeff } (\text{monom } m a) m = a$   
 $\langle \text{proof} \rangle$

**lemma** *coeff-not-in-monomials[simp]*:  $\text{coeff } m x = 0$  **if**  $x \notin \text{monomials } m$   
 $\langle \text{proof} \rangle$

**code-thms** *insertion*

**lemma** *insertion-code[code]*:  $\text{insertion } i mp =$

$(\sum m \in \text{monomials } mp. (\text{coeff } mp m) * (\prod k \in \text{keys } m. i k \wedge \text{lookup } m k))$   
 $\langle \text{proof} \rangle$

**code-thms** *insertion*

**value**  $\text{insertion } (\text{nth } [1, 2.3]) ((\text{Var } 0 + \text{Const } (3::\text{int}) * \text{Var } 1) \wedge 2)$

**lift-definition** *isolate-variable-sparse*::'a::comm-monoid-add mpoly  $\Rightarrow$   
 nat  $\Rightarrow$  nat  $\Rightarrow$  'a mpoly **is**  
 $\lambda(mp::'a\ mpoly)\ x\ d.\ \text{sum}$   
 $(\lambda m.\ \text{monomial}\ (\text{coeff}\ mp\ m)\ (\text{Poly-Mapping.update}\ x\ 0\ m))$   
 $\{m \in \text{monomials}\ mp.\ \text{lookup}\ m\ x = d\}$   $\langle\text{proof}\rangle$

**lemma** *Poly-Mapping-update-code*[code]: *Poly-Mapping.update* a b (Pm-fmap  
 fm) = Pm-fmap (fmupd a b fm)  
 $\langle\text{proof}\rangle$

**lemma** *monom-zero* [simp] : *monom* m 0 = 0  
 $\langle\text{proof}\rangle$

**lemma** *remove-key-code*[code]: *remove-key* v (Pm-fmap fm) = Pm-fmap  
 (fmdrop v fm)  
 $\langle\text{proof}\rangle$

**lemma** *extract-var-code*[code]:  
*extract-var* p v =  
 $(\sum_{m \in \text{monomials}\ p} \text{monom}\ (\text{remove-key}\ v\ m)\ (\text{monom}\ (\text{Poly-Mapping.single}\ v\ (\text{lookup}\ m\ v))\ (\text{coeff}\ p\ m))))$   
 $\langle\text{proof}\rangle$   
**value** *extract-var* ((Var 0 + Const (3::real) \* Var 1)^2) 0

**code-thms** *degree*  
**value** *degree* ((Var 0 + Const (3::real) \* Var 1)^2) 0

**lemma** *vars-code*[code]: *vars* p =  $\bigcup$  (*keys* ' *monomials* p)  
 $\langle\text{proof}\rangle$

**value** *vars* ((Var 0 + Const (3::real) \* Var 1)^2)

**fun** *is-constant* :: 'a::zero mpoly  $\Rightarrow$  bool **where**  
*is-constant* p = *Set.is-empty* (*vars* p)

**value** *is-constant* (Const (0::int))

**fun** *get-if-const* :: *real mpoly*  $\Rightarrow$  *real option* **where**  
*get-if-const* *p* = (*if is-constant p then Some (coeff p 0) else None*)

**term** *coeff* *p* 0

**lemma** *insertionNegative* : *insertion f p* = - *insertion f (-p)*  
 ⟨*proof*⟩

**definition** *derivative* :: *nat*  $\Rightarrow$  *real mpoly*  $\Rightarrow$  *real mpoly* **where**  
*derivative* *x p* = ( $\sum i \in \{0..degree\ p\ x - 1\}. isolate\_variable\_sparse\ p\ x\ (i+1) * (Var\ x) \hat{\ }i * (Const\ (i+1))$ )

*get\_coefs* *x p* gets the tuple of coefficients *a b c* of the term  $a * x^2 + b * x + c$  in polynomial *p*

**fun** *get-coeffs* :: *nat*  $\Rightarrow$  *real mpoly*  $\Rightarrow$  *real mpoly* \* *real mpoly* \* *real mpoly* **where**  
*get-coeffs* *var x* = (  
*isolate-variable-sparse* *x var 2*,  
*isolate-variable-sparse* *x var 1*,  
*isolate-variable-sparse* *x var 0*)

**end**

Executable Polynomial Properties

**theory** *ExecutablePolyProps*

**imports**

*Polynomials.MPoly-Type-Univariate*

*MPolyExtension*

**begin**

(Univariate) Polynomial hiding

**lifting-update** *poly.lifting*

**lifting-forget** *poly.lifting*

### 3.2 Lemmas with Monomial and Monomials

**lemma** *of-nat-monomial*: *of-nat p* = *monomial p 0*  
 ⟨*proof*⟩

**lemma** *of-nat-times-monomial*: *of-nat p \* monomial c i* = *monomial (p\*c) i*  
 ⟨*proof*⟩

**lemma** *monomial-adds-nat-iff*: *monomial p i adds c*  $\longleftrightarrow$  *lookup c i*  $\geq$  *p* **for** *p::nat*  
 ⟨*proof*⟩

**lemma** *update-minus-monomial*:  $Poly\text{-Mapping.update } k \ i \ (m - \text{monomial } i \ k) = Poly\text{-Mapping.update } k \ i \ m$   
 ⟨proof⟩

**lemma** *monomials-Var*:  $\text{monomials } (Var \ x :: 'a :: \text{zero-neq-one } mpoly) = \{Poly\text{-Mapping.single } x \ 1\}$   
 ⟨proof⟩

**lemma** *monomials-Const*:  $\text{monomials } (Const \ x) = (\text{if } x = 0 \ \text{then } \{\} \ \text{else } \{0\})$   
 ⟨proof⟩

**lemma** *coeff-eq-zero-iff*:  $MPoly\text{-Type.coeff } c \ p = 0 \iff p \notin \text{monomials } c$   
 ⟨proof⟩

**lemma** *monomials-1[simp]*:  $\text{monomials } 1 = \{0\}$   
 ⟨proof⟩

**lemma** *monomials-and-monom*:  
**shows**  $(k \in \text{monomials } m) = (\exists (a :: \text{nat}). a \neq 0 \wedge (\text{monomials } (MPoly\text{-Type.monom } k \ a)) \subseteq \text{monomials } m)$   
 ⟨proof⟩

**lemma** *mult-monomials-dir-one*:  
**shows**  $\text{monomials } (p * q) \subseteq \{a + b \mid a \ b . a \in \text{monomials } p \wedge b \in \text{monomials } q\}$   
 ⟨proof⟩

**lemma** *monom-eq-zero-iff[simp]*:  $MPoly\text{-Type.monom } a \ b = 0 \iff b = 0$   
 ⟨proof⟩

**lemma** *update-eq-plus-monomial*:  
 $v \geq \text{lookup } m \ k \implies Poly\text{-Mapping.update } k \ v \ m = m + \text{monomial } (v - \text{lookup } m \ k) \ k$   
**for**  $v \ n :: \text{nat}$   
 ⟨proof⟩

**lemma** *coeff-monom-mult'*:  
 $MPoly\text{-Type.coeff } ((MPoly\text{-Type.monom } m' \ a) * q) \ (m' m) = a * MPoly\text{-Type.coeff } q \ (m' m - m')$   
**if**  $*$ :  $m' m = m' + (m' m - m')$   
 ⟨proof⟩

**lemma** *monomials-zero[simp]*:  $\text{monomials } 0 = \{\}$   
 ⟨proof⟩

**lemma** *in-monomials-iff*:  $x \in \text{monomials } m \iff MPoly\text{-Type.coeff } m \ x \neq 0$   
 ⟨proof⟩

**lemma** *monomials-monom-mult*:  $\text{monomials } (MPoly\text{-Type.monom } mon \ a * q) = (\text{if } a = 0 \ \text{then } \{\} \ \text{else } (+) \ mon \ ' \ \text{monomials } q)$

**for**  $q::'a::\text{semiring-no-zero-divisors mpoly}$   
 $\langle\text{proof}\rangle$

### 3.3 Simplification Lemmas for Const 0 and Const 1

**lemma**  $\text{add-zero} : P + \text{Const } 0 = P$   
 $\langle\text{proof}\rangle$

**lemma**  $\text{add-zero-example} : ((\text{Var } 0)^{\wedge 2} - (\text{Const } 1)) + \text{Const } 0 = ((\text{Var } 0)^{\wedge 2} - (\text{Const } 1))$   
 $\langle\text{proof}\rangle$

**lemma**  $\text{mult-zero-left} : \text{Const } 0 * P = \text{Const } 0$   
 $\langle\text{proof}\rangle$

**lemma**  $\text{mult-zero-right} : P * \text{Const } 0 = \text{Const } 0$   
 $\langle\text{proof}\rangle$

**lemma**  $\text{mult-one-left} : \text{Const } 1 * (P :: \text{real mpoly}) = P$   
 $\langle\text{proof}\rangle$

**lemma**  $\text{mult-one-right} : (P :: \text{real mpoly}) * \text{Const } 1 = P$   
 $\langle\text{proof}\rangle$

### 3.4 Coefficient Lemmas

**lemma**  $\text{coeff-zero}[\text{simp}] : \text{MPoly-Type.coeff } 0 \ x = 0$   
 $\langle\text{proof}\rangle$

**lemma**  $\text{coeff-sum} : \text{MPoly-Type.coeff } (\text{sum } f \ S) \ x = \text{sum } (\lambda i. \text{MPoly-Type.coeff } (f \ i) \ x) \ S$   
 $\langle\text{proof}\rangle$

**lemma**  $\text{coeff-mult-Var} : \text{MPoly-Type.coeff } (x * \text{Var } i \ ^{\wedge} p) \ c = (\text{MPoly-Type.coeff } x \ (c - \text{monomial } p \ i) \ \text{when } \text{lookup } c \ i \geq p)$   
 $\langle\text{proof}\rangle$

**lemma**  $\text{lookup-update-self}[\text{simp}] : \text{Poly-Mapping.update } i \ (\text{lookup } m \ i) \ m = m$   
 $\langle\text{proof}\rangle$

**lemma**  $\text{coeff-Const} : \text{MPoly-Type.coeff } (\text{Const } p) \ m = (p \ \text{when } m = 0)$   
 $\langle\text{proof}\rangle$

**lemma**  $\text{coeff-Var} : \text{MPoly-Type.coeff } (\text{Var } p) \ m = (1 \ \text{when } m = \text{monomial } 1 \ p)$   
 $\langle\text{proof}\rangle$

### 3.5 Miscellaneous

**lemma**  $\text{update-0-0}[\text{simp}] : \text{Poly-Mapping.update } x \ 0 \ 0 = 0$

*<proof>*

**lemma** *mpoly-eq-iff*:  $p = q \longleftrightarrow (\forall m. \text{MPoly-Type.coeff } p \ m = \text{MPoly-Type.coeff } q \ m)$   
*<proof>*

**lemma** *power-both-sides* :  
assumes  $Ah : (A::\text{real}) \geq 0$   
assumes  $Bh : (B::\text{real}) \geq 0$   
shows  $(A \leq B) = (A^2 \leq B^2)$   
*<proof>*

**lemma** *in-list-induct-helper*:  
assumes  $\text{set}(xs) \subseteq X$   
assumes  $P []$   
assumes  $(\bigwedge x. x \in X \implies (\bigwedge xs. P \ xs \implies P \ (x \ # \ xs)))$   
shows  $P \ xs$  *<proof>*

**theorem** *in-list-induct* [*case-names Nil Cons*]:  
assumes  $P []$   
assumes  $(\bigwedge x. x \in \text{set}(xs) \implies (\bigwedge xs. P \ xs \implies P \ (x \ # \ xs)))$   
shows  $P \ xs$   
*<proof>*

### 3.5.1 Keys and vars

**lemma** *inKeys-inVars* :  $a \neq 0 \implies x \in \text{keys } m \implies x \in \text{vars}(\text{MPoly-Type.monom } m \ a)$   
*<proof>*

**lemma** *notInKeys-notInVars* :  $x \notin \text{keys } m \implies x \notin \text{vars}(\text{MPoly-Type.monom } m \ a)$   
*<proof>*

**lemma** *lookupNotIn* :  $x \notin \text{keys } m \implies \text{lookup } m \ x = 0$   
*<proof>*

### 3.6 Degree Lemmas

**lemma** *degree-le-iff*:  $\text{MPoly-Type.degree } p \ x \leq k \longleftrightarrow (\forall m \in \text{monomials } p. \text{lookup } m \ x \leq k)$   
*<proof>*

**lemma** *degree-less-iff*:  $\text{MPoly-Type.degree } p \ x < k \longleftrightarrow (k > 0 \wedge (\forall m \in \text{monomials } p. \text{lookup } m \ x < k))$   
*<proof>*

**lemma** *degree-ge-iff*:  $k > 0 \implies \text{MPoly-Type.degree } p \ x \geq k \longleftrightarrow (\exists m \in \text{monomials } p. \text{lookup } m \ x \geq k)$   
*<proof>*

**lemma** *degree-greater-iff*:  $MPoly\text{-Type.degree } p \ x > k \longleftrightarrow (\exists m \in \text{monomials } p. \text{lookup } m \ x > k)$

*<proof>*

**lemma** *degree-eq-iff*:

$MPoly\text{-Type.degree } p \ x = k \longleftrightarrow (\text{if } k = 0$

$\text{then } (\forall m \in \text{monomials } p. \text{lookup } m \ x = 0)$

$\text{else } (\exists m \in \text{monomials } p. \text{lookup } m \ x = k) \wedge (\forall m \in \text{monomials } p. \text{lookup } m \ x \leq k))$

*<proof>*

**declare** *poly-mapping.lookup-inject*[simp del]

**lemma** *lookup-eq-and-update-lemma*:  $\text{lookup } m \ \text{var} = \text{deg} \wedge \text{Poly-Mapping.update } \text{var } 0 \ m = x \longleftrightarrow$

$m = \text{Poly-Mapping.update } \text{var } \text{deg } x \wedge \text{lookup } x \ \text{var} = 0$

*<proof>*

**lemma** *degree-const* :  $MPoly\text{-Type.degree } (\text{Const } (z :: \text{real})) \ (x :: \text{nat}) = 0$

*<proof>*

**lemma** *degree-one* :  $MPoly\text{-Type.degree } (\text{Var } x :: \text{real } \text{mpoly}) \ x = 1$

*<proof>*

### 3.7 Lemmas on treating a multivariate polynomial as univariate

**lemma** *coeff-isolate-variable-sparse*:

$MPoly\text{-Type.coeff } (\text{isolate-variable-sparse } p \ \text{var } \text{deg}) \ x =$

$(\text{if } \text{lookup } x \ \text{var} = 0$

$\text{then } MPoly\text{-Type.coeff } p \ (\text{Poly-Mapping.update } \text{var } \text{deg } x)$

$\text{else } 0)$

*<proof>*

**lemma** *isovarspar-sum*:

$\text{isolate-variable-sparse } (p+q) \ \text{var } \text{deg} =$

$\text{isolate-variable-sparse } (p) \ \text{var } \text{deg}$

$+ \text{isolate-variable-sparse } (q) \ \text{var } \text{deg}$

*<proof>*

**lemma** *isolate-zero*[simp]:  $\text{isolate-variable-sparse } 0 \ \text{var } n = 0$

*<proof>*

**lemma** *coeff-isolate-variable-sparse-minus-monomial*:

$MPoly\text{-Type.coeff } (\text{isolate-variable-sparse } mp \ \text{var } i) \ (m - \text{monomial } i \ \text{var}) =$

$(\text{if } \text{lookup } m \ \text{var} \leq i \text{ then } MPoly\text{-Type.coeff } mp \ (\text{Poly-Mapping.update } \text{var } i \ m)$

$\text{else } 0)$

*<proof>*

**lemma** *sum-over-zero*:  $(mp::\text{real mpoly}) = (\sum i::\text{nat} \leq \text{MPoly-Type.degree } mp \ x. \text{isolate-variable-sparse } mp \ x \ i * \text{Var } x \hat{i})$   
 ⟨proof⟩

**lemma** *const-lookup-zero* :  $\text{isolate-variable-sparse } (\text{Const } p :: \text{real mpoly}) \ (x::\text{nat}) \ (0::\text{nat}) = \text{Const } p$   
 ⟨proof⟩

**lemma** *const-lookup-suc* :  $\text{isolate-variable-sparse } (\text{Const } p :: \text{real mpoly}) \ x \ (\text{Suc } i) = 0$   
 ⟨proof⟩

**lemma** *isovar-greater-degree* :  $\forall i > \text{MPoly-Type.degree } p \ \text{var. } \text{isolate-variable-sparse } p \ \text{var } i = 0$   
 ⟨proof⟩

**lemma** *isolate-var-0* :  $\text{isolate-variable-sparse } (\text{Var } x::\text{real mpoly}) \ x \ 0 = 0$   
 ⟨proof⟩

**lemma** *isolate-var-one* :  $\text{isolate-variable-sparse } (\text{Var } x :: \text{real mpoly}) \ x \ 1 = 1$   
 ⟨proof⟩

**lemma** *isovarspase-monom* :  
**assumes** *notInKeys* :  $x \notin \text{keys } m$   
**assumes** *notZero* :  $a \neq 0$   
**shows**  $\text{isolate-variable-sparse } (\text{MPoly-Type.monom } m \ a) \ x \ 0 = (\text{MPoly-Type.monom } m \ a :: \text{real mpoly})$   
 ⟨proof⟩

**lemma** *isolate-variable-spase-zero* :  $\text{lookup } m \ x = 0 \implies \text{insertion } (\text{nth } L) \ ((\text{MPoly-Type.monom } m \ a)::\text{real mpoly}) = 0 \implies a \neq 0 \implies \text{insertion } (\text{nth } L) \ (\text{isolate-variable-sparse } (\text{MPoly-Type.monom } m \ a) \ x \ 0) = 0$   
 ⟨proof⟩

**lemma** *isovarsparNotIn* :  $x \notin \text{vars } (p::\text{real mpoly}) \implies \text{isolate-variable-sparse } p \ x \ 0 = p$   
 ⟨proof⟩

**lemma** *degree0isovarspar* :  
**assumes** *deg0* :  $\text{MPoly-Type.degree } (p::\text{real mpoly}) \ x = 0$   
**shows**  $\text{isolate-variable-sparse } p \ x \ 0 = p$   
 ⟨proof⟩

### 3.8 Summation Lemmas

**lemma** *summation-normalized* :

**assumes**  $nonzero : (B :: real) \neq 0$   
**shows**  $(\sum i = 0..<(n::nat)+1). (f i :: real) * B ^ (n - i) = (\sum i = 0..<(n+1). (f i) / (B ^ i)) * (B ^ n)$   
 <proof>

**lemma normalize-summation :**  
**assumes**  $nonzero : (B::real)\neq 0$   
**shows**  $(\sum i = 0..<n+1. f i * B ^ (n - i)) = 0$   
 $\longleftrightarrow$   
 $(\sum i = 0..<(n::nat)+1. (f i :: real) / (B ^ i)) = 0$   
 <proof>

**lemma normalize-summation-less :**  
**assumes**  $nonzero : (B::real)\neq 0$   
**shows**  $(\sum i = 0..<(n+1). (f i) * B ^ (n - i)) * B ^ (n \bmod 2) < 0$   
 $\longleftrightarrow$   
 $(\sum i = 0..<((n::nat)+1). (f i :: real) / (B ^ i)) < 0$   
 <proof>

### 3.9 Additional Lemmas with Vars

**lemma not-in-isovarspar :** *isolate-variable-sparse*  $(p::real \text{ mpoly}) \text{ var } x = (q::real \text{ mpoly}) \implies \text{var}\notin(\text{vars } q)$   
 <proof>

**lemma not-in-add :**  $\text{var}\notin(\text{vars } (p::real \text{ mpoly})) \implies \text{var}\notin(\text{vars } (q::real \text{ mpoly})) \implies \text{var}\notin(\text{vars } (p+q))$   
 <proof>

**lemma not-in-mult :**  $\text{var}\notin(\text{vars } (p::real \text{ mpoly})) \implies \text{var}\notin(\text{vars } (q::real \text{ mpoly})) \implies \text{var}\notin(\text{vars } (p*q))$   
 <proof>

**lemma not-in-neg :**  $\text{var}\notin(\text{vars}(p::real \text{ mpoly})) \longleftrightarrow \text{var}\notin(\text{vars}(-p))$   
 <proof>

**lemma not-in-sub :**  $\text{var}\notin(\text{vars } (p::real \text{ mpoly})) \implies \text{var}\notin(\text{vars } (q::real \text{ mpoly})) \implies \text{var}\notin(\text{vars } (p-q))$   
 <proof>

**lemma not-in-pow :**  $\text{var}\notin(\text{vars}(p::real \text{ mpoly})) \implies \text{var}\notin(\text{vars}(p^i))$   
 <proof>

**lemma not-in-sum-var :**  $(\forall i. \text{var}\notin(\text{vars}(f(i)::real \text{ mpoly}))) \implies \text{var}\notin(\text{vars}(\sum i \in \{0..<(n::nat)\}.f(i)))$   
 <proof>

**lemma not-in-sum :**  $(\forall i. \text{var}\notin(\text{vars}(f(i)::real \text{ mpoly}))) \implies \forall (n::nat). \text{var}\notin(\text{vars}(\sum i \in \{0..<n\}.f(i)))$

$\langle \text{proof} \rangle$

**lemma** *not-contains-insertion-helper* :

$\forall x \in \text{keys} (\text{mapping-of } p). \text{ var } \notin \text{keys } x \implies$   
 $(\bigwedge k f. (k \notin \text{keys } f) = (\text{lookup } f \ k = 0)) \implies$   
 $\text{lookup } (\text{mapping-of } p) \ a = 0 \vee$   
 $(\prod aa. (\text{if } aa < \text{length } L \text{ then } L[\text{var} := y] ! aa \text{ else } 0) \wedge \text{lookup } a \ aa) =$   
 $(\prod aa. (\text{if } aa < \text{length } L \text{ then } L[\text{var} := x] ! aa \text{ else } 0) \wedge \text{lookup } a \ aa)$   
 $\langle \text{proof} \rangle$

**lemma** *not-contains-insertion* :

**assumes** *notIn* :  $\text{var} \notin \text{vars } (p :: \text{real mpoly})$   
**assumes** *exists* :  $\text{insertion } (\text{nth-default } 0 \ (\text{list-update } L \ \text{var } x)) \ p = \text{val}$   
**shows**  $\text{insertion } (\text{nth-default } 0 \ (\text{list-update } L \ \text{var } y)) \ p = \text{val}$   
 $\langle \text{proof} \rangle$

### 3.10 Insertion Lemmas

**lemma** *insertion-sum-var* :  $((\text{insertion } f \ (\sum_{i \in \{0..<(n::\text{nat})\}}. g(i))) = (\sum_{i \in \{0..<n\}}. \text{insertion } f \ (g \ i)))$   
 $\langle \text{proof} \rangle$

**lemma** *insertion-sum* :  $\forall (n :: \text{nat}). ((\text{insertion } f \ (\sum_{i \in \{0..<n\}}. g(i))) = (\sum_{i \in \{0..<n\}}. \text{insertion } f \ (g \ i)))$   
 $\langle \text{proof} \rangle$

**lemma** *insertion-sum'* :  $\bigwedge (n :: \text{nat}). ((\text{insertion } f \ (\sum_{i \leq n}. g(i))) = (\sum_{i \leq n}. \text{insertion } f \ (g \ i)))$   
 $\langle \text{proof} \rangle$

**lemma** *insertion-pow* :  $\text{insertion } f \ (p \hat{=} i) = (\text{insertion } f \ p) \hat{=} i$   
 $\langle \text{proof} \rangle$

**lemma** *insertion-neg* :  $\text{insertion } f \ (-p) = -\text{insertion } f \ p$   
 $\langle \text{proof} \rangle$

**lemma** *insertion-var* :

$\text{length } L > \text{var} \implies \text{insertion } (\text{nth-default } 0 \ (\text{list-update } L \ \text{var } x)) \ (\text{Var } \text{var}) = x$   
 $\langle \text{proof} \rangle$

**lemma** *insertion-var-zero* :  $\text{insertion } (\text{nth-default } 0 \ (x \# xs)) \ (\text{Var } 0) = x \ \langle \text{proof} \rangle$

**lemma** *notIn-insertion-sub* :  $x \notin \text{vars}(p :: \text{real mpoly}) \implies x \notin \text{vars}(q :: \text{real mpoly})$   
 $\implies \text{insertion } f \ (p - q) = \text{insertion } f \ p - \text{insertion } f \ q$   
 $\langle \text{proof} \rangle$

**lemma** *insertion-sub* :  $\text{insertion } f \ (A - B :: \text{real mpoly}) = \text{insertion } f \ A - \text{insertion } f \ B$

$f B$   
 $\langle proof \rangle$

**lemma** *insertion-four* :  $insertion ((nth-default 0) xs) 4 = 4$   
 $\langle proof \rangle$

**lemma** *insertion-add-ind-basecase*:  
 $insertion (nth (list-update L var x)) ((\sum i::nat \leq 0. isolate-variable-sparse p var$   
 $i * (Var var) \hat{i}))$   
 $= (\sum i = 0..<(0+1). insertion (nth (list-update L var x)) (isolate-variable-sparse$   
 $p var i * (Var var) \hat{i}))$   
 $\langle proof \rangle$

**lemma** *insertion-add-ind*:  
 $insertion (nth-default 0 (list-update L var x)) ((\sum i::nat \leq d. isolate-variable-sparse$   
 $p var i * (Var var) \hat{i}))$   
 $= (\sum i = 0..<(d+1). insertion (nth-default 0 (list-update L var x)) (isolate-variable-sparse$   
 $p var i * (Var var) \hat{i}))$   
 $\langle proof \rangle$

**lemma** *sum-over-degree-insertion* :  
**assumes**  $lLength : length L > var$   
**assumes**  $deg : MPoly-Type.degree (p::real mpoly) var = d$   
**shows**  $(\sum i = 0..<(d+1). insertion (nth-default 0 (list-update L var x)) (isolate-variable-sparse$   
 $p var i) * (x \hat{i}))$   
 $= insertion (nth-default 0 (list-update L var x)) p$   
 $\langle proof \rangle$

**lemma** *insertion-isovarspars-free* :  
 $insertion (nth-default 0 (list-update L var x)) (isolate-variable-sparse (p::real$   
 $mpoly) var (i::nat))$   
 $= insertion (nth-default 0 (list-update L var y)) (isolate-variable-sparse (p::real$   
 $mpoly) var (i::nat))$   
 $\langle proof \rangle$

**lemma** *insertion-zero* :  $insertion f (Const 0 ::real mpoly) = 0$   
 $\langle proof \rangle$

**lemma** *insertion-one* :  $insertion f (Const 1 ::real mpoly) = 1$   
 $\langle proof \rangle$

**lemma** *insertion-const* :  $insertion f (Const c::real mpoly) = (c::real)$   
 $\langle proof \rangle$

### 3.11 Putting Things Together

#### 3.11.1 More Degree Lemmas

**lemma** *degree-add-leq* :

**assumes**  $h1 : MPoly-Type.degree\ a\ var \leq x$   
**assumes**  $h2 : MPoly-Type.degree\ b\ var \leq x$   
**shows**  $MPoly-Type.degree\ (a+b)\ var \leq x$   
 $\langle proof \rangle$

**lemma** *degree-add-less* :  
**assumes**  $h1 : MPoly-Type.degree\ a\ var < x$   
**assumes**  $h2 : MPoly-Type.degree\ b\ var < x$   
**shows**  $MPoly-Type.degree\ (a+b)\ var < x$   
 $\langle proof \rangle$

**lemma** *degree-sum* :  $(\forall i \in \{0..n::nat\}. MPoly-Type.degree\ (f\ i :: real\ mpoly)\ var \leq x) \implies (MPoly-Type.degree\ (\sum_{x \in \{0..n\}} f\ x)\ var) \leq x$   
 $\langle proof \rangle$

**lemma** *degree-sum-less* :  $(\forall i \in \{0..n::nat\}. MPoly-Type.degree\ (f\ i :: real\ mpoly)\ var < x) \implies (MPoly-Type.degree\ (\sum_{x \in \{0..n\}} f\ x)\ var) < x$   
 $\langle proof \rangle$

**lemma** *varNotIn-degree* :  
**assumes**  $var \notin vars\ p$   
**shows**  $MPoly-Type.degree\ p\ var = 0$   
 $\langle proof \rangle$

**lemma** *degree-mult-leq* :  
**assumes**  $pnonzero : (p :: real\ mpoly) \neq 0$   
**assumes**  $qnonzero : (q :: real\ mpoly) \neq 0$   
**shows**  $MPoly-Type.degree\ ((p :: real\ mpoly) * (q :: real\ mpoly))\ var \leq (MPoly-Type.degree\ p\ var) + (MPoly-Type.degree\ q\ var)$   
 $\langle proof \rangle$

**lemma** *degree-exists-monom*:  
**assumes**  $p \neq 0$   
**shows**  $\exists m \in monomials\ p. lookup\ m\ var = MPoly-Type.degree\ p\ var$   
 $\langle proof \rangle$

**lemma** *degree-not-exists-monom*:  
**assumes**  $p \neq 0$   
**shows**  $\neg (\exists m \in monomials\ p. lookup\ m\ var > MPoly-Type.degree\ p\ var)$   
 $\langle proof \rangle$

**lemma** *degree-monom*:  $MPoly-Type.degree\ (MPoly-Type.monom\ x\ y)\ v = (if\ y = 0\ then\ 0\ else\ lookup\ x\ v)$   
 $\langle proof \rangle$

**lemma** *degree-plus-disjoint*:  
 $MPoly-Type.degree\ (p + MPoly-Type.monom\ m\ c)\ v = max\ (MPoly-Type.degree\ p\ v)\ (MPoly-Type.degree\ (MPoly-Type.monom\ m\ c)\ v)$

**if**  $m \notin \text{monomials } p$   
**for**  $p::\text{real mpoly}$   
 ⟨*proof*⟩

### 3.11.2 More isolate\_variable\_sparse lemmas

**lemma** *isolate-variable-sparse-ne-zeroD*:  
 $\text{isolate-variable-sparse } mp \ v \ x \neq 0 \implies x \leq \text{MPoly-Type.degree } mp \ v$   
 ⟨*proof*⟩

**context includes** *poly.lifting begin*

**lift-definition** *mpoly-to-nested-poly::'a::comm-monoid-add mpoly  $\Rightarrow$  nat  $\Rightarrow$  'a mpoly*  
*Polynomial.poly is*  
 $\lambda(mp::'a \text{ mpoly}) (v::\text{nat}) (p::\text{nat}). \text{isolate-variable-sparse } mp \ v \ p$   
 — note *extract-var* nests the other way around  
 ⟨*proof*⟩

**lemma** *degree-eq-0-mpoly-to-nested-polyI*:  
 $\text{mpoly-to-nested-poly } mp \ v = 0 \implies \text{MPoly-Type.degree } mp \ v = 0$   
 ⟨*proof*⟩

**lemma** *coeff-eq-zero-mpoly-to-nested-polyD*:  $\text{mpoly-to-nested-poly } mp \ v = 0 \implies$   
 $\text{MPoly-Type.coeff } mp \ 0 = 0$   
 ⟨*proof*⟩

**lemma** *mpoly-to-nested-poly-eq-zero-iff[simp]*:  
 $\text{mpoly-to-nested-poly } mp \ v = 0 \iff mp = 0$   
 ⟨*proof*⟩

**lemma** *isolate-variable-sparse-degree-eq-zero-iff*:  $\text{isolate-variable-sparse } p \ v \ (\text{MPoly-Type.degree } p \ v) = 0 \iff p = 0$   
 ⟨*proof*⟩

**lemma** *degree-eq-univariate-degree*:  $\text{MPoly-Type.degree } p \ v =$   
 $(\text{if } p = 0 \text{ then } 0 \text{ else } \text{Polynomial.degree } (\text{mpoly-to-nested-poly } p \ v))$   
 ⟨*proof*⟩

**lemma** *compute-mpoly-to-nested-poly[code]*:  
 $\text{coeffs } (\text{mpoly-to-nested-poly } mp \ v) =$   
 $(\text{if } mp = 0 \text{ then } []$   
 $\text{else } \text{map } (\text{isolate-variable-sparse } mp \ v) [0..<\text{Suc}(\text{MPoly-Type.degree } mp \ v)])$   
 ⟨*proof*⟩

**end**

**lemma** *isolate-variable-sparse-monom*:  $\text{isolate-variable-sparse } (\text{MPoly-Type.monom } m \ a) \ v \ i =$   
 $(\text{if } a = 0 \vee \text{lookup } m \ v \neq i \text{ then } 0 \text{ else } \text{MPoly-Type.monom } (\text{Poly-Mapping.update } v \ 0 \ m) \ a)$

*<proof>*

**lemma** *isolate-variable-sparse-monom-mult:*

*isolate-variable-sparse (MPoly-Type.monom m a \* q) v n =*  
*(if n ≥ lookup m v*  
*then MPoly-Type.monom (Poly-Mapping.update v 0 m) a \* isolate-variable-sparse*  
*q v (n - lookup m v)*  
*else 0)*  
**for** *q::'a::semiring-no-zero-divisors mpoly*  
*<proof>*

**lemma** *isolate-variable-sparse-mult:*

*isolate-variable-sparse (p \* q) v n = (∑ i≤n. isolate-variable-sparse p v i \**  
*isolate-variable-sparse q v (n - i))*  
**for** *p q::'a::semiring-no-zero-divisors mpoly*  
*<proof>*

### 3.11.3 More Miscellaneous

**lemma** *var-not-in-Const : var∉vars (Const x :: real mpoly)*  
*<proof>*

**lemma** *mpoly-to-nested-poly-mult:*

*mpoly-to-nested-poly (p \* q) v = mpoly-to-nested-poly p v \* mpoly-to-nested-poly*  
*q v*  
**for** *p q::'a::{comm-semiring-0, semiring-no-zero-divisors} mpoly*  
*<proof>*

**lemma** *get-if-const-insertion :*

**assumes** *get-if-const (p::real mpoly) = Some r*  
**shows** *insertion f p = r*  
*<proof>*

### 3.11.4 Yet more Degree Lemmas

**lemma** *degree-mult:*

**fixes** *p q::'a::{comm-semiring-0, ring-1-no-zero-divisors} mpoly*  
**assumes** *p ≠ 0*  
**assumes** *q ≠ 0*  
**shows** *MPoly-Type.degree (p \* q) v = MPoly-Type.degree p v + MPoly-Type.degree*  
*q v*  
*<proof>*

**lemma** *degree-eq:*

**assumes** *(p::real mpoly) = (q::real mpoly)*  
**shows** *MPoly-Type.degree p x = MPoly-Type.degree q x*  
*<proof>*

**lemma** *degree-var-i* : *MPoly-Type.degree* (((*Var x*)<sup>i</sup>:: *real mpoly*)) *x = i*  
 ⟨*proof*⟩

**lemma** *degree-less-sum*:  
 assumes *MPoly-Type.degree* (*p*::*real mpoly*) *var = n*  
 assumes *MPoly-Type.degree* (*q*::*real mpoly*) *var = m*  
 assumes *m < n*  
 shows *MPoly-Type.degree* (*p + q*) *var = n*  
 ⟨*proof*⟩

**lemma** *degree-less-sum'*:  
 assumes *MPoly-Type.degree* (*p*::*real mpoly*) *var = n*  
 assumes *MPoly-Type.degree* (*q*::*real mpoly*) *var = m*  
 assumes *n < m*  
 shows *MPoly-Type.degree* (*p + q*) *var = m* ⟨*proof*⟩

**lemma** *nonzero-const-is-nonzero*:  
 assumes (*k*::*real*)  $\neq 0$   
 shows ((*Const k*)::*real mpoly*)  $\neq 0$   
 ⟨*proof*⟩

**lemma** *degree-derivative* :  
 assumes *MPoly-Type.degree* *p x > 0*  
 shows *MPoly-Type.degree* *p x = MPoly-Type.degree* (*derivative x p*) *x + 1*  
 ⟨*proof*⟩

**lemma** *express-poly* :  
 assumes *h* : *MPoly-Type.degree* (*p*::*real mpoly*) *var = 1*  $\vee$  *MPoly-Type.degree* *p*  
*var = 2*  
 shows *p =*  
   (*isolate-variable-sparse p var 2*)\*(*Var var*)<sup>2</sup>  
   +(*isolate-variable-sparse p var 1*)\*(*Var var*)  
   +(*isolate-variable-sparse p var 0*)  
 ⟨*proof*⟩

**lemma** *degree-isovarspar* : *MPoly-Type.degree* (*isolate-variable-sparse* (*p*::*real mpoly*)  
*x i*) *x = 0*  
 ⟨*proof*⟩

end

## 4 Atoms

theory *PolyAtoms*

```

imports ExecutablePolyProps
begin

```

#### 4.1 Definition

```

datatype (atoms: 'a) fm =
  TrueF | FalseF | Atom 'a | And 'a fm 'a fm | Or 'a fm 'a fm |
  Neg 'a fm | ExQ 'a fm | AllQ 'a fm | ExN nat 'a fm | AllN nat 'a fm

```

**definition** *neg* **where**

```

neg  $\varphi = (\text{if } \varphi = \text{TrueF} \text{ then FalseF else if } \varphi = \text{FalseF} \text{ then TrueF else Neg } \varphi)$ 

```

**definition** *and*  $:: 'a \text{ fm} \Rightarrow 'a \text{ fm} \Rightarrow 'a \text{ fm}$  **where**

```

and  $\varphi_1 \varphi_2 =$ 
  (if  $\varphi_1 = \text{TrueF}$  then  $\varphi_2$  else if  $\varphi_2 = \text{TrueF}$  then  $\varphi_1$  else
  if  $\varphi_1 = \text{FalseF} \vee \varphi_2 = \text{FalseF}$  then FalseF else And  $\varphi_1 \varphi_2$ )

```

**definition** *or*  $:: 'a \text{ fm} \Rightarrow 'a \text{ fm} \Rightarrow 'a \text{ fm}$  **where**

```

or  $\varphi_1 \varphi_2 =$ 
  (if  $\varphi_1 = \text{FalseF}$  then  $\varphi_2$  else if  $\varphi_2 = \text{FalseF}$  then  $\varphi_1$  else
  if  $\varphi_1 = \text{TrueF} \vee \varphi_2 = \text{TrueF}$  then TrueF else Or  $\varphi_1 \varphi_2$ )

```

**definition** *list-conj*  $:: 'a \text{ fm list} \Rightarrow 'a \text{ fm}$  **where**

```

list-conj  $fs = \text{foldr and fs TrueF}$ 

```

**definition** *list-disj*  $:: 'a \text{ fm list} \Rightarrow 'a \text{ fm}$  **where**

```

list-disj  $fs = \text{foldr or fs FalseF}$ 

```

The atom datatype corresponds to the defined in Tobias's LinearQuantifierElim.

```

datatype atom = Less real mpoly | Eq real mpoly | Leq real mpoly | Neq real mpoly

```

For each atom, the real mpoly corresponds to a polynomial from the Polynomials library where we allow for real valued coefficients.

The variables in the polynomials are in De Bruijn notation where variable 0 corresponds to the variable of the innermost quantifier, then variable 1 is the next quantifier out from that, and so on. Any variable number greater than the number of quantifiers is a free variable. This means that we have a list of infinite free variables we can pick from and if we want to refer to the  $i$ th free variable (indexed at 0) within an atom that is bound in  $j$  quantifiers, then we would use  $\text{var } (i+j)$ .

The polynomials are all standardized so that they are compared to a 0 on the right. This means the atom Less  $p$  corresponds to  $p \leq 0$  and the atom Eq  $p$  corresponds to  $p = 0$  and so on. This restriction doesn't lose any generality and having all 4 of these kinds of atoms prevents loss of efficiency as the negation of these atoms do not introduce additional logical connectives. The following aNeg function demonstrates this.

**fun** *aNeg* :: *atom*  $\Rightarrow$  *atom* **where**  
*aNeg* (*Less* *p*) = *Leq* ( $-p$ ) |  
*aNeg* (*Eq* *p*) = *Neq* *p* |  
*aNeg* (*Leq* *p*) = *Less* ( $-p$ ) |  
*aNeg* (*Neq* *p*) = *Eq* *p*

## 4.2 Evaluation

In order to do any proofs with these atoms, we need a method of comparing two atoms to check if they are equal. Instead of trying to manipulate the polynomials to a standard form to compare them, it is a lot easier to plug in values for every variable and check if the results are equal. If every single real value input for each variable matches in truth value for both atoms, then they are equal.

*aEval* *a l* corresponds to plugging in the real value list *l* into the variables of atom *a* and then evaluating the truth value of it

**fun** *aEval* :: *atom*  $\Rightarrow$  *real list*  $\Rightarrow$  *bool* **where**  
*aEval* (*Eq* *p*) *L* = (*insertion* (*nth-default* 0 *L*) *p* = 0) |  
*aEval* (*Less* *p*) *L* = (*insertion* (*nth-default* 0 *L*) *p* < 0) |  
*aEval* (*Leq* *p*) *L* = (*insertion* (*nth-default* 0 *L*) *p*  $\leq$  0) |  
*aEval* (*Neq* *p*) *L* = (*insertion* (*nth-default* 0 *L*) *p*  $\neq$  0)

*aNeg\_aEval* shows the general format for how things are proven equal. Plugging in the values to an original atom *a* will results in the opposite truth value if we transformed with the *aNeg* function.

**lemma** *aNeg\_aEval* : *aEval* *a L*  $\longleftrightarrow$  ( $\neg$  *aEval* (*aNeg* *a*) *L*)  
 <proof>

We can extend this to formulas instead of just atoms. Given a formula in prenex normal form, we simply iterate through and apply the quantifiers

**fun** *eval* :: *atom fm*  $\Rightarrow$  *real list*  $\Rightarrow$  *bool* **where**  
*eval* (*Atom* *a*)  $\Gamma$  = *aEval* *a*  $\Gamma$  |  
*eval* (*TrueF*) - = *True* |  
*eval* (*FalseF*) - = *False* |  
*eval* (*And*  $\varphi$   $\psi$ )  $\Gamma$  = ((*eval*  $\varphi$   $\Gamma$ )  $\wedge$  (*eval*  $\psi$   $\Gamma$ )) |  
*eval* (*Or*  $\varphi$   $\psi$ )  $\Gamma$  = ((*eval*  $\varphi$   $\Gamma$ )  $\vee$  (*eval*  $\psi$   $\Gamma$ )) |  
*eval* (*Neg*  $\varphi$ )  $\Gamma$  = ( $\neg$  (*eval*  $\varphi$   $\Gamma$ )) |  
*eval* (*ExQ*  $\varphi$ )  $\Gamma$  = ( $\exists x. (eval \varphi (x\#\Gamma))$ ) |  
*eval* (*AllQ*  $\varphi$ )  $\Gamma$  = ( $\forall x. (eval \varphi (x\#\Gamma))$ ) |  
*eval* (*AllN* *i*  $\varphi$ )  $\Gamma$  = ( $\forall l. length\ l = i \longrightarrow (eval \varphi (l\@\Gamma))$ ) |  
*eval* (*ExN* *i*  $\varphi$ )  $\Gamma$  = ( $\exists l. length\ l = i \wedge (eval \varphi (l\@\Gamma))$ )

**lemma** *eval* (*ExQ* (*Or* (*Atom* *A*) (*Atom* *B*))) *xs* = *eval* (*Or* (*ExQ*(*Atom* *A*))  
 (*ExQ*(*Atom* *B*))) *xs*  
 <proof>

**lemma** *eval-neg-neg* :  $eval (neg (neg f)) L \longleftrightarrow eval f L$   
 ⟨proof⟩

**lemma** *eval-neg* :  $(\neg eval (neg f) L) \longleftrightarrow eval f L$   
 ⟨proof⟩

**lemma** *eval-and* :  $eval (and a b) L \longleftrightarrow (eval a L \wedge eval b L)$   
 ⟨proof⟩

**lemma** *eval-or* :  $eval (or a b) L \longleftrightarrow (eval a L \vee eval b L)$   
 ⟨proof⟩

**lemma** *eval-Or* :  $eval (Or a b) L \longleftrightarrow (eval a L \vee eval b L)$   
 ⟨proof⟩

**lemma** *eval-And* :  $eval (And a b) L \longleftrightarrow (eval a L \wedge eval b L)$   
 ⟨proof⟩

**lemma** *eval-not* :  $eval (neg a) L \longleftrightarrow \neg(eval a L)$   
 ⟨proof⟩

**lemma** *eval-true* :  $eval TrueF L$   
 ⟨proof⟩

**lemma** *eval-false* :  $\neg(eval FalseF L)$   
 ⟨proof⟩

**lemma** *eval-Neg* :  $eval (Neg \varphi) L = eval (neg \varphi) L$   
 ⟨proof⟩

**lemma** *eval-Neg-Neg* :  $eval (Neg (Neg \varphi)) L = eval \varphi L$   
 ⟨proof⟩

**lemma** *eval-Neg-And* :  $eval (Neg (And \varphi \psi)) L = eval (Or (Neg \varphi) (Neg \psi)) L$   
 ⟨proof⟩

**lemma** *aEval-leq* :  $aEval (Leq p) L = (aEval (Less p) L \vee aEval (Eq p) L)$   
 ⟨proof⟩

This function is misleading because it is true iff the variable given doesn't occur as a free variable in the atom fm

**fun** *freeIn* ::  $nat \Rightarrow atom\ fm \Rightarrow bool$  **where**  
*freeIn* var (Atom(Eq p)) = (var  $\notin$  (vars p))|  
*freeIn* var (Atom(Less p)) = (var  $\notin$  (vars p))|  
*freeIn* var (Atom(Leq p)) = (var  $\notin$  (vars p))|  
*freeIn* var (Atom(Neg p)) = (var  $\notin$  (vars p))|  
*freeIn* var (TrueF) = True|

```

freeIn var (FalseF) = True |
freeIn var (And a b) = ((freeIn var a) ∧ (freeIn var b)) |
freeIn var (Or a b) = ((freeIn var a) ∧ (freeIn var b)) |
freeIn var (Neg a) = freeIn var a |
freeIn var (ExQ a) = freeIn (var+1) a |
freeIn var (AllQ a) = freeIn (var+1) a |
freeIn var (AllN i a) = freeIn (var+i) a |
freeIn var (ExN i a) = freeIn (var+i) a

```

**fun** liftmap :: (nat ⇒ atom ⇒ atom fm) ⇒ atom fm ⇒ nat ⇒ atom fm **where**

```

liftmap f TrueF var = TrueF |
liftmap f FalseF var = FalseF |
liftmap f (Atom a) var = f var a |
liftmap f (And φ ψ) var = And (liftmap f φ var) (liftmap f ψ var) |
liftmap f (Or φ ψ) var = Or (liftmap f φ var) (liftmap f ψ var) |
liftmap f (Neg φ) var = Neg (liftmap f φ var) |
liftmap f (ExQ φ) var = ExQ (liftmap f φ (var+1)) |
liftmap f (AllQ φ) var = AllQ (liftmap f φ (var+1)) |
liftmap f (AllN i φ) var = AllN i (liftmap f φ (var+i)) |
liftmap f (ExN i φ) var = ExN i (liftmap f φ (var+i))

```

**fun** depth :: 'a fm ⇒ nat **where**

```

depth TrueF = 1 |
depth FalseF = 1 |
depth (Atom -) = 1 |
depth (And φ ψ) = max (depth φ) (depth ψ) + 1 |
depth (Or φ ψ) = max (depth φ) (depth ψ) + 1 |
depth (Neg φ) = depth φ + 1 |
depth (ExQ φ) = depth φ + 1 |
depth (AllQ φ) = depth φ + 1 |
depth (AllN i φ) = depth φ + 1 |
depth (ExN i φ) = depth φ + 1

```

**value** AllQ (And

```

(ExQ (Atom (Eq (Var 1 * Var 2 - (Var 0)2 * Var 3))))
(Neg (AllQ (Atom (Leq (Const 5 * (Var 1)2 - Var 0))))))

```

)

**fun** negation-free :: atom fm ⇒ bool **where**

```

negation-free TrueF = True |
negation-free FalseF = True |
negation-free (Atom a) = True |
negation-free (And φ1 φ2) = ((negation-free φ1) ∧ (negation-free φ2)) |
negation-free (Or φ1 φ2) = ((negation-free φ1) ∧ (negation-free φ2)) |
negation-free (ExQ φ) = negation-free φ |

```

*negation-free* (*AllQ*  $\varphi$ ) = *negation-free*  $\varphi$  |  
*negation-free* (*AllN*  $i$   $\varphi$ ) = *negation-free*  $\varphi$  |  
*negation-free* (*ExN*  $i$   $\varphi$ ) = *negation-free*  $\varphi$  |  
*negation-free* (*Neg*  $-$ ) = *False*

### 4.3 Useful Properties

**lemma** *sum-eq* : *eval* (*Atom*(*Eq*  $p$ ))  $L \longrightarrow$  *eval* (*Atom*(*Eq*  $q$ ))  $L \longrightarrow$  *eval* (*Atom*(*Eq*( $p + q$ )))  $L$   
 <proof>

**lemma** *freeIn-list-conj* :  $(\forall f \in \text{set}(F). \text{freeIn } \text{var } f) \implies \text{freeIn } \text{var } (\text{list-conj } F)$   
 <proof>

**lemma** *freeIn-list-disj* :  
**assumes**  $\forall f \in \text{set } (L::\text{atom } \text{fm } \text{list}). \text{freeIn } \text{var } f$   
**shows** *freeIn*  $\text{var } (\text{list-disj } L)$   
 <proof>

**lemma** *var-not-in-aEval* : *freeIn*  $\text{var } (\text{Atom } \varphi) \implies (\exists x. \text{aEval } \varphi (\text{list-update } L \text{ var } x)) = (\forall x. \text{aEval } \varphi (\text{list-update } L \text{ var } x))$   
 <proof>

**lemma** *var-not-in-aEval2* : *freeIn*  $0 (\text{Atom } \varphi) \implies (\exists x. \text{aEval } \varphi (x\#L)) = (\forall x. \text{aEval } \varphi (x\#L))$   
 <proof>

**lemma** *plugInLinear* :  
**assumes** *lLength* : *length*  $L > \text{var}$   
**assumes** *nonzero* :  $B \neq 0$   
**assumes** *hb* :  $\forall v. \text{insertion } (\text{nth-default } 0 (\text{list-update } L \text{ var } v)) \text{ } b = B$   
**assumes** *hc* :  $\forall v. \text{insertion } (\text{nth-default } 0 (\text{list-update } L \text{ var } v)) \text{ } c = C$   
**shows** *aEval* (*Eq*( $b * \text{Var } \text{var} + c$ )) (*list-update*  $L \text{ var } (-C/B)$ )  
 <proof>

### 4.4 Some eval results

**lemma** *doubleExist* : *eval* (*ExN*  $2 A$ )  $L =$  *eval* (*ExQ* (*ExQ*  $A$ ))  $L$   
 <proof>

**lemma** *doubleForall* : *eval* (*AllN*  $2 A$ )  $L =$  *eval* (*AllQ* (*AllQ*  $A$ ))  $L$   
 <proof>

**lemma** *unwrapExist* : *eval* (*ExN*  $(j + 1) A$ )  $L =$  *eval* (*ExQ* (*ExN*  $j A$ ))  $L$   
 <proof>

**lemma** *unwrapExist'* : *eval* (*ExN*  $(j + 1) A$ )  $L =$  *eval* (*ExN*  $j$  (*ExQ*  $A$ ))  $L$   
 <proof>

**lemma** *unwrapExist''* : *eval* (*ExN*  $(i + j) A$ )  $L =$  *eval* (*ExN*  $i$  (*ExN*  $j A$ ))  $L$

*<proof>*

**lemma** *unwrapForall* :  $eval (AllN (j + 1) A) L = eval (AllQ (AllN j A)) L$   
*<proof>*

**lemma** *unwrapForall'* :  $eval (AllN (j + 1) A) L = eval (AllN j (AllQ A)) L$   
*<proof>*

**lemma** *unwrapForall''* :  $eval (AllN (i + j) A) L = eval (AllN i (AllN j A)) L$   
*<proof>*

**lemma** *var-not-in-eval* :  $\forall var. \forall L. (freeIn\ var\ \varphi \longrightarrow ((\exists x. eval\ \varphi\ (list-update\ L\ var\ x)) = (\forall x. eval\ \varphi\ (list-update\ L\ var\ x))))$   
*<proof>*

**lemma** *var-not-in-eval2* :  $\forall L. (freeIn\ 0\ \varphi \longrightarrow ((\exists x. eval\ \varphi\ (x\#L)) = (\forall x. eval\ \varphi\ (x\#L))))$   
*<proof>*

**lemma** *var-not-in-eval3* :  
  **assumes** *freeIn var*  $\varphi$   
  **assumes** *length xs' = var*  
  **shows**  $((\exists x. eval\ \varphi\ (xs'\@x\#L)) = (\forall x. eval\ \varphi\ (xs'\@x\#L)))$   
*<proof>*

**lemma** *eval-list-conj* :  $eval (list-conj F) L = (\forall f \in set(F). eval\ f\ L)$   
*<proof>*

**lemma** *eval-list-disj* :  $eval (list-disj F) L = (\exists f \in set(F). eval\ f\ L)$   
*<proof>*  
**end**

## 5 Debruijn Indices Formulation

**theory** *Debruijn*  
  **imports** *PolyAtoms*  
**begin**

### 5.1 Lift and Lower Functions

these functions are required for debruijn notation the (liftPoly n a p) functions increment each variable greater n in polynomial p by a the (lowerPoly n a p) functions lower each variable greater than n by a so variables n through n+a-1 shouldn't exist

**context includes** *poly-mapping.lifting* **begin**

**definition** *inc-above b i x* =  $(if\ x < b\ then\ x\ else\ x + i::nat)$

**definition** *dec-above*  $b\ i\ x = (\text{if } x \leq b \text{ then } x \text{ else } x - i :: \text{nat})$

**lemma** *inc-above-dec-above*:  $x < b \vee b + i \leq x \implies \text{inc-above } b\ i\ (\text{dec-above } b\ i\ x) = x$   
*<proof>*

**lemma** *dec-above-inc-above*:  $\text{dec-above } b\ i\ (\text{inc-above } b\ i\ x) = x$   
*<proof>*

**lemma** *inc-above-dec-above-iff*:  $\text{inc-above } b\ i\ (\text{dec-above } b\ i\ x) = x \iff x < b \vee b + i \leq x$   
*<proof>*

**lemma** *inj-on-dec-above*:  $\text{inj-on } (\text{dec-above } b\ i)\ \{x. x < b \vee b + i \leq x\}$   
*<proof>*

**lemma** *finite-inc-above-ne*:  $\text{finite } \{x. f\ x \neq c\} \implies \text{finite } \{x. f\ (\text{inc-above } b\ i\ x) \neq c\}$   
*<proof>*

**lemma** *finite-dec-above-ne*:  $\text{finite } \{x. f\ x \neq c\} \implies \text{finite } \{x. f\ (\text{dec-above } b\ i\ x) \neq c\}$   
*<proof>*

**lift-definition** *lowerPowers*:  $\text{nat} \Rightarrow \text{nat} \Rightarrow (\text{nat} \Rightarrow_0 'a) \Rightarrow (\text{nat} \Rightarrow_0 'a :: \text{zero})$   
**is**  $\lambda b\ i\ p\ x. \text{if } x \in \{b..<b+i\} \text{ then } 0 \text{ else } p\ (\text{dec-above } b\ i\ x) :: 'a$   
*<proof>*

**lift-definition** *higherPowers*:  $\text{nat} \Rightarrow \text{nat} \Rightarrow (\text{nat} \Rightarrow_0 'a) \Rightarrow (\text{nat} \Rightarrow_0 'a :: \text{zero})$   
**is**  $\lambda b\ i\ p\ x. p\ (\text{inc-above } b\ i\ x) :: 'a$   
*<proof>*

**lemma** *higherPowers-lowerPowers*:  $\text{higherPowers } n\ i\ (\text{lowerPowers } n\ i\ x) = x$   
*<proof>*

**lemma** *inj-lowerPowers*:  $\text{inj } (\text{lowerPowers } b\ i)$   
*<proof>*

**lemma** *lowerPowers-higherPowers*:  
 $(\bigwedge j. n \leq j \implies j < n + i \implies \text{lookup } x\ j = 0) \implies \text{lowerPowers } n\ i\ (\text{higherPowers } n\ i\ x) = x$   
*<proof>*

**lemma** *inj-on-higherPowers*:  $\text{inj-on } (\text{higherPowers } n\ i)\ \{x. \forall j. n \leq j \wedge j < n + i \implies \text{lookup } x\ j = 0\}$   
*<proof>*

**lemma** *higherPowers-eq*:  $\text{lookup } (\text{higherPowers } b\ i\ p)\ x = \text{lookup } p\ (\text{inc-above } b\ i\ x)$   
*<proof>*

**lemma** *lowerPowers-eq*:  $\text{lookup } (\text{lowerPowers } b \ i \ p) \ x = (\text{if } b \leq x \wedge x < b + i \text{ then } 0 \text{ else } \text{lookup } p \ (\text{dec-above } b \ i \ x))$   
 ⟨proof⟩

**lemma** *keys-higherPowers*:  $\text{keys } (\text{higherPowers } b \ i \ m) = \text{dec-above } b \ i \ ' (\text{keys } m \cap \{x. x \notin \{b..<b+i\}\})$   
 ⟨proof⟩

**context includes** *fmap.lifting begin*

**lift-definition** *lowerPowers<sub>f</sub>*:: $\text{nat} \Rightarrow \text{nat} \Rightarrow (\text{nat}, 'a) \text{ fmap} \Rightarrow (\text{nat}, 'a::\text{zero}) \text{ fmap}$   
 is  $\lambda b \ i \ p \ x. \text{if } x \in \{b..<b+i\} \text{ then } \text{None} \text{ else } p \ (\text{dec-above } b \ i \ x)$   
 ⟨proof⟩

**lift-definition** *higherPowers<sub>f</sub>*:: $\text{nat} \Rightarrow \text{nat} \Rightarrow (\text{nat}, 'a) \text{ fmap} \Rightarrow (\text{nat}, 'a) \text{ fmap}$   
 is  $\lambda b \ i \ f \ x. f \ (\text{inc-above } b \ i \ x)$   
 ⟨proof⟩

**lemma** *map-of-map-key-inverse-fun-eq*:  
 $\text{map-of } (\text{map } (\lambda(k, y). (f \ k, y)) \ xs) \ x = \text{map-of } xs \ (g \ x)$   
 if  $\bigwedge x. x \in \text{set } xs \implies g \ (f \ (\text{fst } x)) = \text{fst } x \ f \ (g \ x) = x$   
 for  $f::'a \Rightarrow 'b$   
 ⟨proof⟩

**lemma** *map-of-filter-key-in*:  $P \ x \implies \text{map-of } (\text{filter } (\lambda(k, v). P \ k) \ xs) \ x = \text{map-of } xs \ x$   
 ⟨proof⟩

**lemma** *map-of-eq-NoneI*:  $x \notin \text{fst}' \text{set } xs \implies \text{map-of } xs \ x = \text{None}$   
 ⟨proof⟩

**lemma** *compute-higherPowers<sub>f</sub>[code]*:  $\text{higherPowers}_f \ b \ i \ (\text{fmap-of-list } xs) = \text{fmap-of-list } (\text{map } (\lambda(k, v). (\text{if } k < b \ \text{then } k \ \text{else } k - i, v)) \ (\text{filter } (\lambda(k, v). k \notin \{b..<b+i\}) \ xs))$   
 ⟨proof⟩

**lemma** *compute-lowerPowers<sub>f</sub>[code]*:  $\text{lowerPowers}_f \ b \ i \ (\text{fmap-of-list } xs) = \text{fmap-of-list } (\text{map } (\lambda(k, v). (\text{if } k < b \ \text{then } k \ \text{else } k + i, v)) \ xs)$   
 ⟨proof⟩

**lemma** *compute-higherPowers[code]*:  $\text{higherPowers } n \ i \ (\text{Pm-fmap } xs) = \text{Pm-fmap } (\text{higherPowers}_f \ n \ i \ xs)$   
 ⟨proof⟩

**lemma** *compute-lowerPowers[code]*:  $\text{lowerPowers } n \ i \ (\text{Pm-fmap } xs) = \text{Pm-fmap } (\text{lowerPowers}_f \ n \ i \ xs)$   
 ⟨proof⟩

**lemma** *finite-nonzero-coeff*: *finite* {*x*. *MPoly-Type.coeff* *mpoly* *x* ≠ 0}  
 ⟨*proof*⟩

**lift-definition** *lowerPoly<sub>0</sub>*::*nat* ⇒ *nat* ⇒ ((*nat*⇒<sub>0</sub>*nat*)⇒<sub>0</sub>'*a*::*zero*) ⇒ ((*nat*⇒<sub>0</sub>*nat*)⇒<sub>0</sub>'*a*) **is**  
 λ*b i* (*mp*::(*nat*⇒<sub>0</sub>*nat*)⇒'*a*) *mon*. *mp* (*lowerPowers* *b i mon*)  
 ⟨*proof*⟩

**lemma** *higherPowers-zero[simp]*: *higherPowers* *b i 0* = 0  
 ⟨*proof*⟩

**lemma** *keys-lowerPoly<sub>0</sub>*: *keys* (*lowerPoly<sub>0</sub>* *b i mp*) = *higherPowers* *b i* ‘ (*keys* *mp*  
 ∩ {*x*. ∀*j*∈{*b*..*b+i*}. *lookup* *x j* = 0})  
 ⟨*proof*⟩

**lift-definition** *higherPoly<sub>0</sub>*::*nat* ⇒ *nat* ⇒ ((*nat*⇒<sub>0</sub>*nat*)⇒<sub>0</sub>'*a*::*zero*) ⇒ ((*nat*⇒<sub>0</sub>*nat*)⇒<sub>0</sub>'*a*) **is**  
 λ*b i* (*mp*::(*nat*⇒<sub>0</sub>*nat*)⇒'*a*) *mon*.  
   *if* (∃*j*∈{*b*..*b+i*}. *lookup* *mon j* > 0)  
   *then* 0  
   *else* *mp* (*higherPowers* *b i mon*)  
 ⟨*proof*⟩

**context includes** *fmap.lifting begin*

**lift-definition** *lowerPoly<sub>f</sub>*::*nat* ⇒ *nat* ⇒ ((*nat*⇒<sub>0</sub>*nat*), '*a*::*zero*)*fmap* ⇒ ((*nat*⇒<sub>0</sub>*nat*), '*a*)*fmap* **is**  
 λ*b i* (*mp*::(*nat*⇒<sub>0</sub>*nat*)→'*a*) *mon*::(*nat*⇒<sub>0</sub>*nat*). *mp* (*lowerPowers* *b i mon*)  
 ⟨*proof*⟩

**lift-definition** *higherPoly<sub>f</sub>*::*nat* ⇒ *nat* ⇒ ((*nat*⇒<sub>0</sub>*nat*), '*a*::*zero*)*fmap* ⇒ ((*nat*⇒<sub>0</sub>*nat*), '*a*)*fmap* **is**  
 λ*b i* (*mp*::(*nat*⇒<sub>0</sub>*nat*)→'*a*) *mon*::(*nat*⇒<sub>0</sub>*nat*).  
   *if* (∃*j*∈{*b*..*b+i*}. *lookup* *mon j* > 0)  
   *then* *None*  
   *else* *mp* (*higherPowers* *b i mon*)  
 ⟨*proof*⟩

**lemma** *keys-lowerPowers*: *keys* (*lowerPowers* *b i m*) = *inc-above* *b i* ‘ (*keys* *m*)  
 ⟨*proof*⟩

**lemma** *keys-higherPoly<sub>0</sub>*: *keys* (*higherPoly<sub>0</sub>* *b i mp*) = *lowerPowers* *b i* ‘ (*keys* *mp*)  
 ⟨*proof*⟩

**end**

**lemma** *inc-above-id*[simp]:  $n < m \implies \text{inc-above } m \ i \ n = n$  *<proof>*

**lemma** *inc-above-Suc*[simp]:  $n \geq m \implies \text{inc-above } m \ i \ n = n + i$  *<proof>*

**lemma** *compute-lowerPoly<sub>0</sub>*[code]:  $\text{lowerPoly}_0 \ n \ i \ (Pm\text{-fmap } m) = Pm\text{-fmap } (\text{lowerPoly}_f \ n \ i \ m)$   
*<proof>*

**lemma** *compute-higherPoly<sub>0</sub>*[code]:  $\text{higherPoly}_0 \ n \ i \ (Pm\text{-fmap } m) = Pm\text{-fmap } (\text{higherPoly}_f \ n \ i \ m)$   
*<proof>*

**lemma** *compute-lowerPoly<sub>f</sub>*[code]:  $\text{lowerPoly}_f \ n \ i \ (\text{fmap-of-list } xs) =$   
 $(\text{fmap-of-list } (\text{map } (\lambda(\text{mon}, c). (\text{higherPowers } n \ i \ \text{mon}, c))$   
 $(\text{filter } (\lambda(\text{mon}, v). \forall j \in \{n..<n+i\}. \text{lookup } \text{mon } j = 0) \ xs)))$   
*<proof>*

**lemma** *compute-higherPoly<sub>f</sub>*[code]:  $\text{higherPoly}_f \ n \ i \ (\text{fmap-of-list } xs) =$   
 $\text{fmap-of-list } (\text{filter } (\lambda(\text{mon}, v). \forall j \in \{n..<n+i\}. \text{lookup } \text{mon } j = 0)$   
 $(\text{map } (\lambda(\text{mon}, c). (\text{lowerPowers } n \ i \ \text{mon}, c)) \ xs))$   
*<proof>*

**end**

**lift-definition** *lowerPoly*:: $\text{nat} \Rightarrow \text{nat} \Rightarrow 'a::\text{zero } \text{mpoly} \Rightarrow 'a \ \text{mpoly}$  **is** *lowerPoly<sub>0</sub>*  
*<proof>*

**lift-definition** *liftPoly*:: $\text{nat} \Rightarrow \text{nat} \Rightarrow 'a::\text{zero } \text{mpoly} \Rightarrow 'a \ \text{mpoly}$  **is** *higherPoly<sub>0</sub>*  
*<proof>*

**lemma** *coeff-lowerPoly*:  $MPoly\text{-Type}.coeff \ (\text{lowerPoly } b \ i \ mp) \ x = MPoly\text{-Type}.coeff$   
 $mp \ (\text{lowerPowers } b \ i \ x)$   
*<proof>*

**lemma** *coeff-liftPoly*:  $MPoly\text{-Type}.coeff \ (\text{liftPoly } b \ i \ mp) \ x = (\text{if } (\exists j \in \{b..<b+i\}.$   
 $\text{lookup } x \ j > 0)$   
 $\text{then } 0$   
 $\text{else } MPoly\text{-Type}.coeff \ mp \ (\text{higherPowers } b \ i \ x))$   
*<proof>*

**lemma** *monomials-lowerPoly*:  $\text{monomials } (\text{lowerPoly } b \ i \ mp) = \text{higherPowers } b \ i$   
 $' (\text{monomials } mp \cap \{x. \forall j \in \{b..<b+i\}. \text{lookup } x \ j = 0\})$   
*<proof>*

**lemma** *monomials-liftPoly*:  $\text{monomials } (\text{liftPoly } b \ i \ mp) = \text{lowerPowers } b \ i \ ' (\text{monomials}$   
 $mp)$   
*<proof>*

**value** [code] *lowerPoly* 1 1 (1 \* Var 0 + 2 \* Var 2 ^ 2 + 3 \* Var 3 ^ 4 :: int mpoly) = (Var 0 + 2 \* Var 1 ^ 2 + 3 \* Var 2 ^ 4 :: int mpoly)  
**value** [code] *lowerPoly* 1 3 (1 \* Var 0 + 2 \* Var 4 ^ 2 + 3 \* Var 5 ^ 4 :: int mpoly) = (Var 0 + 2 \* Var 1 ^ 2 + 3 \* Var 2 ^ 4 :: int mpoly)

**value** [code] *liftPoly* 1 3 (1 \* Var 0 + 2 \* Var 4 ^ 2 + 3 \* Var 5 ^ 4 :: int mpoly) = (Var 0 + 2 \* Var 7 ^ 2 + 3 \* Var 8 ^ 4 :: int mpoly)

**fun** *lowerAtom* :: nat ⇒ nat ⇒ atom ⇒ atom **where**  
*lowerAtom* d amount (Eq p) = Eq(*lowerPoly* d amount p)|  
*lowerAtom* d amount (Less p) = Less(*lowerPoly* d amount p)|  
*lowerAtom* d amount (Leq p) = Leq(*lowerPoly* d amount p)|  
*lowerAtom* d amount (Neq p) = Neq(*lowerPoly* d amount p)

**lemma** *lookup-not-in-vars-eq-zero*:  $x \in \text{monomials } p \implies i \notin \text{vars } p \implies \text{lookup } x \ i = 0$   
⟨proof⟩

**lemma** *nth-dec-above*:  
**assumes**  $\text{length } xs = i \ \text{length } ys = j \ k \notin \{i..<i+j\}$   
**shows**  $\text{nth-default } 0 \ (xs \ @ \ zs) \ (\text{dec-above } i \ j \ k) = (\text{nth-default } 0 \ (xs \ @ \ ys \ @ \ zs)) \ k$   
⟨proof⟩

**lemma** *insertion-lowerPoly*:  
**assumes**  $i\text{-notin: } \text{vars } p \cap \{i..<i+j\} = \{\}$   
**and**  $\text{lprfx: } \text{length } \text{prfx} = i$   
**and**  $\text{lhs: } \text{length } xs = j$   
**shows**  $\text{insertion } (\text{nth-default } 0 \ (\text{prfx}@L)) \ (\text{lowerPoly } i \ j \ p) = \text{insertion } (\text{nth-default } 0 \ (\text{prfx}@xs@L)) \ p \ (\text{is } ?lhs = ?rhs)$   
⟨proof⟩

**lemma** *insertion-lowerPoly1*:  
**assumes**  $i\text{-notin: } i \notin \text{vars } p$   
**and**  $\text{lprfx: } \text{length } \text{prfx} = i$   
**shows**  $\text{insertion } (\text{nth-default } 0 \ (\text{prfx}@x\#L)) \ p = \text{insertion } (\text{nth-default } 0 \ (\text{prfx}@L)) \ (\text{lowerPoly } i \ 1 \ p)$   
⟨proof⟩

**lemma** *insertion-lowerPoly01*:  
**assumes**  $i\text{-notin: } 0 \notin \text{vars } p$   
**shows**  $\text{insertion } (\text{nth-default } 0 \ (x\#L)) \ p = \text{insertion } (\text{nth-default } 0 \ L) \ (\text{lowerPoly } 0 \ 1 \ p)$   
⟨proof⟩

**lemma** *aEval-lowerAtom* :  $(\text{freeIn } 0 \ (\text{Atom } A)) \implies (\text{aEval } A \ (x\#L) = \text{aEval } (\text{lowerAtom } 0 \ 1 \ A) \ L)$   
⟨proof⟩

**fun** *map-fm-binders* :: (*atom*  $\Rightarrow$  *nat*  $\Rightarrow$  *atom*)  $\Rightarrow$  *atom fm*  $\Rightarrow$  *nat*  $\Rightarrow$  *atom fm*  
**where**  
*map-fm-binders* *f* *TrueF* *n* = *TrueF*|  
*map-fm-binders* *f* *FalseF* *n* = *FalseF*|  
*map-fm-binders* *f* (*Atom*  $\varphi$ ) *n* = *Atom* (*f*  $\varphi$  *n*)|  
*map-fm-binders* *f* (*And*  $\varphi$   $\psi$ ) *n* = *And* (*map-fm-binders* *f*  $\varphi$  *n*) (*map-fm-binders* *f*  $\psi$  *n*)|  
*map-fm-binders* *f* (*Or*  $\varphi$   $\psi$ ) *n* = *Or* (*map-fm-binders* *f*  $\varphi$  *n*) (*map-fm-binders* *f*  $\psi$  *n*)|  
*map-fm-binders* *f* (*ExQ*  $\varphi$ ) *n* = *ExQ*(*map-fm-binders* *f*  $\varphi$  (*n*+1))|  
*map-fm-binders* *f* (*AllQ*  $\varphi$ ) *n* = *AllQ*(*map-fm-binders* *f*  $\varphi$  (*n*+1))|  
*map-fm-binders* *f* (*AllN* *i*  $\varphi$ ) *n* = *AllN* *i* (*map-fm-binders* *f*  $\varphi$  (*n*+*i*))|  
*map-fm-binders* *f* (*ExN* *i*  $\varphi$ ) *n* = *ExN* *i* (*map-fm-binders* *f*  $\varphi$  (*n*+*i*))|  
*map-fm-binders* *f* (*Neg*  $\varphi$ ) *n* = *Neg*(*map-fm-binders* *f*  $\varphi$  *n*)

**fun** *lowerFm* :: *nat*  $\Rightarrow$  *nat*  $\Rightarrow$  *atom fm*  $\Rightarrow$  *atom fm* **where**  
*lowerFm* *d* *amount* *f* = *map-fm-binders* ( $\lambda a. \lambda x. \text{lowerAtom } (d+x) \text{ amount } a$ ) *f*  
0

**fun** *delete-nth* :: *nat*  $\Rightarrow$  *real list*  $\Rightarrow$  *real list* **where**  
*delete-nth* *n* *xs* = *take* *n* *xs* @ *drop* *n* *xs*

**lemma** *eval-lowerFm-helper* :  
**assumes** *freeIn* *i* *F*  
**assumes** *length* *init* = *i*  
**shows** *eval* (*lowerFm* *i* 1 *F*) (*init* @ *xs*) = *eval* *F* (*init*@[*x*]@*xs*)  
<*proof*>

**lemma** *eval-lowerFm* :  
**assumes** *h* : *freeIn* 0 *F*  
**shows**  $\forall xs. (\text{eval } (\text{lowerFm } 0 \ 1 \ F) \ xs = \text{eval } (ExQ \ F) \ xs)$   
<*proof*>

**fun** *liftAtom* :: *nat*  $\Rightarrow$  *nat*  $\Rightarrow$  *atom*  $\Rightarrow$  *atom* **where**  
*liftAtom* *d* *amount* (*Eq* *p*) = *Eq*(*liftPoly* *d* *amount* *p*)|  
*liftAtom* *d* *amount* (*Less* *p*) = *Less*(*liftPoly* *d* *amount* *p*)|  
*liftAtom* *d* *amount* (*Leq* *p*) = *Leq*(*liftPoly* *d* *amount* *p*)|  
*liftAtom* *d* *amount* (*Neq* *p*) = *Neq*(*liftPoly* *d* *amount* *p*)

**fun** *liftFm* :: *nat*  $\Rightarrow$  *nat*  $\Rightarrow$  *atom fm*  $\Rightarrow$  *atom fm* **where**  
*liftFm* *d* *amount* *f* = *map-fm-binders* ( $\lambda a. \lambda x. \text{liftAtom } (d+x) \text{ amount } a$ ) *f* 0

**fun** *insert-into* :: *real list*  $\Rightarrow$  *nat*  $\Rightarrow$  *real list*  $\Rightarrow$  *real list* **where**  
*insert-into* *xs* *n* *l* = *take* *n* *xs* @ *l* @ *drop* *n* *xs*

**lemma** *higherPoly<sub>0</sub>-add* :  $\text{higherPoly}_0\ x\ y\ (p + q) = \text{higherPoly}_0\ x\ y\ p + \text{higherPoly}_0\ x\ y\ q$   
 ⟨proof⟩

**lemma** *liftPoly-add*:  $\text{liftPoly}\ w\ z\ (a + b) = (\text{liftPoly}\ w\ z\ a) + (\text{liftPoly}\ w\ z\ b)$   
 ⟨proof⟩

**lemma** *vars-lift-add* :  $\text{vars}(\text{liftPoly}\ a\ b\ (p+q)) \subseteq \text{vars}(\text{liftPoly}\ a\ b\ (p)) \cup \text{vars}(\text{liftPoly}\ a\ b\ (q))$   
 ⟨proof⟩

**lemma** *mapping-of-lift-add* :  $\text{mapping-of}\ (\text{liftPoly}\ x\ y\ (a + b)) = \text{mapping-of}\ (\text{liftPoly}\ x\ y\ a) + \text{mapping-of}\ (\text{liftPoly}\ x\ y\ b)$   
 ⟨proof⟩

**lemma** *coeff-lift-add* :  $\text{MPoly-Type.coeff}\ (\text{liftPoly}\ x\ y\ (a + b))\ m = \text{MPoly-Type.coeff}\ (\text{liftPoly}\ x\ y\ a)\ m + \text{MPoly-Type.coeff}\ (\text{liftPoly}\ x\ y\ b)\ m$   
 ⟨proof⟩

**lemma** *lift-add* :  $\text{insertion}\ (f::\text{nat}\Rightarrow\text{real})\ (\text{liftPoly}\ 0\ z\ (a + b)) = \text{insertion}\ f\ (\text{liftPoly}\ 0\ z\ a + \text{liftPoly}\ 0\ z\ b)$   
 ⟨proof⟩

**lemma** *lower-power-zero* :  $\text{lowerPowers}\ a\ b\ 0 = 0$   
 ⟨proof⟩

**lemma** *lift-vars-monom* :  $\text{vars}\ (\text{liftPoly}\ i\ j\ ((\text{MPoly-Type.monom}\ m\ a)::\text{real}\ \text{mpoly})) = (\lambda x. \text{if } x \geq i \text{ then } x+j \text{ else } x) \text{ 'vars}(\text{MPoly-Type.monom}\ m\ a)$   
 ⟨proof⟩

**lemma** *lift-clear-vars* :  $\text{vars}\ (\text{liftPoly}\ i\ j\ (p::\text{real}\ \text{mpoly})) \cap \{i..<i + j\} = \{\}$   
 ⟨proof⟩

**lemma** *lift0*:  $(\text{liftPoly}\ i\ j\ 0) = 0$   
 ⟨proof⟩

**lemma** *lower0*:  $(\text{lowerPoly}\ i\ j\ 0) = 0$   
 ⟨proof⟩

**lemma** *lower-lift-monom* :  $\text{insertion}\ f\ (\text{MPoly-Type.monom}\ m\ a :: \text{real}\ \text{mpoly}) = \text{insertion}\ f\ (\text{lowerPoly}\ i\ j\ (\text{liftPoly}\ i\ j\ (\text{MPoly-Type.monom}\ m\ a)))$   
 ⟨proof⟩

**lemma** *lower-lift* :  $\text{insertion}\ f\ (p::\text{real}\ \text{mpoly}) = \text{insertion}\ f\ (\text{lowerPoly}\ i\ j\ (\text{liftPoly}\ i\ j\ p))$   
 ⟨proof⟩

**lemma** *lift-insertion* :  $\forall$  *init*.  
 $length\ init = (i::nat) \longrightarrow$   
 $(\forall I\ xs.$   
 $(insertion\ (nth\ default\ 0\ (init\ @\ xs))\ (p::real\ mpoly)) = (insertion$   
 $((nth\ default\ 0)\ (init\ @\ I\ @\ xs))\ (liftPoly\ i\ (length\ I)\ p)))$   
 $\langle proof \rangle$

**lemma** *eval-liftFm-helper* :  
**assumes**  $length\ init = i$   
**assumes**  $length\ I = amount$   
**shows**  $eval\ F\ (init\ @\ xs) = eval\ (liftFm\ i\ amount\ F)\ (init@I@xs)$   
 $\langle proof \rangle$

**lemma** *eval-liftFm* :  
**assumes**  $length\ I = amount$   
**assumes**  $length\ L \geq d$   
**shows**  $eval\ F\ L = eval\ (liftFm\ d\ amount\ F)\ (insert-into\ L\ d\ I)$   
 $\langle proof \rangle$

**lemma** *not-in-lift* :  $var \notin vars(p::real\ mpoly) \implies var + z \notin vars(liftPoly\ 0\ z\ p)$   
 $\langle proof \rangle$

**lemma** *lift-const* :  $insertion\ f\ (liftPoly\ 0\ z\ (Const\ (C::real))) = insertion\ f\ (Const$   
 $C\ ::\ real\ mpoly)$   
 $\langle proof \rangle$

**lemma** *liftPoly-sub*:  $liftPoly\ 0\ z\ (a - b) = (liftPoly\ 0\ z\ a) - (liftPoly\ 0\ z\ b)$   
 $\langle proof \rangle$

**lemma** *lift-sub* :  $insertion\ (f::nat \Rightarrow real)\ (liftPoly\ 0\ z\ (a - b)) = insertion\ f$   
 $(liftPoly\ 0\ z\ a - liftPoly\ 0\ z\ b)$   
 $\langle proof \rangle$

**lemma** *lift-minus* :  
**assumes**  $insertion\ (f::nat \Rightarrow real)\ (liftPoly\ 0\ z\ (c - Const\ (C::real))) = 0$   
**shows**  $insertion\ f\ (liftPoly\ 0\ z\ c) = C$   
 $\langle proof \rangle$

**end**

**lemma** *lift00* :  $liftPoly\ 0\ 0\ (a::real\ mpoly) = a$   
 $\langle proof \rangle$

**end**

## 5.2 Swapping Indices

**theory** *Reindex*

**imports** *Debruijn*  
**begin**

**context includes** *poly-mapping.lifting* **begin**

**definition**  $\text{swap } i \ j \ x = (\text{if } x = i \ \text{then } j \ \text{else if } x = j \ \text{then } i \ \text{else } x)$

**lemma**  $\text{swap-swap} : \text{swap } i \ j \ (\text{swap } i \ j \ x) = x$   
*<proof>*

**lemma**  $\text{finite-swap-ne} : \text{finite } \{x. f \ x \neq c\} \implies \text{finite } \{x. f \ (\text{swap } b \ i \ x) \neq c\}$   
*<proof>*

**lift-definition**  $\text{swap0} :: \text{nat} \Rightarrow \text{nat} \Rightarrow (\text{nat} \Rightarrow_0 'a) \Rightarrow (\text{nat} \Rightarrow_0 'a :: \text{zero})$   
**is**  $\lambda b \ i \ p \ x. p \ (\text{swap } b \ i \ x) :: 'a$   
*<proof>*

**lemma**  $\text{swap0-swap0} : \text{swap0 } n \ i \ (\text{swap0 } n \ i \ x) = x$   
*<proof>*

**lemma**  $\text{inj-swap} : \text{inj } (\text{swap } b \ i)$   
*<proof>*

**lemma**  $\text{inj-swap0} : \text{inj } (\text{swap0 } b \ i)$   
*<proof>*

**lemma**  $\text{swap0-eq} : \text{lookup } (\text{swap0 } b \ i \ p) \ x = \text{lookup } p \ (\text{swap } b \ i \ x)$   
*<proof>*

**lemma**  $\text{eq-onp-swap} : \text{eq-onp } (\lambda f. \text{finite } \{x. f \ x \neq 0\}) \ (\lambda x. \text{lookup } m \ (\text{swap } b \ i \ x))$   
 $(\lambda x. \text{lookup } m \ (\text{swap } b \ i \ x))$   
*<proof>*

**lemma**  $\text{keys-swap} : \text{keys } (\text{swap0 } b \ i \ m) = \text{swap } b \ i \ ' \ \text{keys } m$   
*<proof>*

**context includes** *fmap.lifting* **begin**

**lift-definition**  $\text{swap}_f :: \text{nat} \Rightarrow \text{nat} \Rightarrow (\text{nat}, 'a) \text{ fmap} \Rightarrow (\text{nat}, 'a :: \text{zero}) \text{ fmap}$   
**is**  $\lambda b \ i \ p \ x. p \ (\text{swap } b \ i \ x)$   
*<proof>*

**lemma**  $\text{compute-swap}_f[\text{code}] : \text{swap}_f \ b \ i \ (\text{fmap-of-list } xs) =$   
 $\text{fmap-of-list } (\text{map } (\lambda(k, v). (\text{swap } b \ i \ k, v)) \ xs)$   
*<proof>*

**lemma** *compute-swap*[code]:  $\text{swap0 } n \ i \ (Pm\text{-fmap } xs) = Pm\text{-fmap } (\text{swap}_f \ n \ i \ xs)$   
 ⟨proof⟩

**lift-definition** *swapPoly<sub>0</sub>*:: $\text{nat} \Rightarrow \text{nat} \Rightarrow ((\text{nat} \Rightarrow_0 \text{nat}) \Rightarrow_0 'a :: \text{zero}) \Rightarrow ((\text{nat} \Rightarrow_0 \text{nat}) \Rightarrow_0 'a)$  is  
 $\lambda b \ i \ (mp :: (\text{nat} \Rightarrow_0 \text{nat}) \Rightarrow 'a) \ \text{mon}. \ mp \ (\text{swap0 } b \ i \ \text{mon})$   
 ⟨proof⟩

**lemma** *swap-zero*[simp]:  $\text{swap0 } b \ i \ 0 = 0$   
 ⟨proof⟩

**context includes** *fmap.lifting* **begin**

**lift-definition** *swapPoly<sub>f</sub>*:: $\text{nat} \Rightarrow \text{nat} \Rightarrow ((\text{nat} \Rightarrow_0 \text{nat}), 'a :: \text{zero}) \text{fmap} \Rightarrow ((\text{nat} \Rightarrow_0 \text{nat}), 'a) \text{fmap}$  is  
 $\lambda b \ i \ (mp :: ((\text{nat} \Rightarrow_0 \text{nat}) \rightarrow 'a)) \ \text{mon} :: (\text{nat} \Rightarrow_0 \text{nat}). \ mp \ (\text{swap0 } b \ i \ \text{mon})$   
 ⟨proof⟩

**lemma** *keys-swap<sub>0</sub>*:  $\text{keys } (\text{swapPoly}_0 \ b \ i \ mp) = \text{swap0 } b \ i \ ' \ (\text{keys } mp)$   
 ⟨proof⟩

**end**

**lemma** *compute-swapPoly<sub>0</sub>*[code]:  $\text{swapPoly}_0 \ n \ i \ (Pm\text{-fmap } m) = Pm\text{-fmap } (\text{swapPoly}_f \ n \ i \ m)$   
 ⟨proof⟩

**lemma** *compute-swapPoly<sub>f</sub>*[code]:  $\text{swapPoly}_f \ n \ i \ (\text{fmap-of-list } xs) = (\text{fmap-of-list } (\text{map } (\lambda(\text{mon}, c). (\text{swap0 } n \ i \ \text{mon}, c)) \ xs))$   
 ⟨proof⟩

**end**

**end**

**lift-definition** *swap-poly*:: $\text{nat} \Rightarrow \text{nat} \Rightarrow 'a :: \text{zero} \ \text{mpoly} \Rightarrow 'a \ \text{mpoly}$  is *swapPoly<sub>0</sub>*  
 ⟨proof⟩

**value** *swap-poly 0 1* (Var 0 :: real mpoly)

**lemma** *coeff-swap-poly*:  $MPoly\text{-Type}.coeff \ (\text{swap-poly } b \ i \ mp) \ x = MPoly\text{-Type}.coeff \ mp \ (\text{swap0 } b \ i \ x)$   
 ⟨proof⟩

**lemma** *monomials-swap-poly*:  $\text{monomials } (\text{swap-poly } b \ i \ mp) = \text{swap0 } b \ i \ ' \ (\text{monomials } mp)$

*<proof>*

**fun** *swap-atom* :: *nat*  $\Rightarrow$  *nat*  $\Rightarrow$  *atom*  $\Rightarrow$  *atom* **where**

*swap-atom* *a b* (*Eq* *p*) = *Eq* (*swap-poly* *a b p*)|  
*swap-atom* *a b* (*Less* *p*) = *Less* (*swap-poly* *a b p*)|  
*swap-atom* *a b* (*Leq* *p*) = *Leq* (*swap-poly* *a b p*)|  
*swap-atom* *a b* (*Neq* *p*) = *Neq* (*swap-poly* *a b p*)

**fun** *swap-fm* :: *nat*  $\Rightarrow$  *nat*  $\Rightarrow$  *atom fm*  $\Rightarrow$  *atom fm* **where**

*swap-fm* *a b* *TrueF* = *TrueF*|  
*swap-fm* *a b* *FalseF* = *FalseF*|  
*swap-fm* *a b* (*Atom* *At*) = *Atom*(*swap-atom* *a b At*)|  
*swap-fm* *a b* (*And* *A B*) = *And*(*swap-fm* *a b A*)(*swap-fm* *a b B*)|  
*swap-fm* *a b* (*Or* *A B*) = *Or*(*swap-fm* *a b A*)(*swap-fm* *a b B*)|  
*swap-fm* *a b* (*Neg* *A*) = *Neg*(*swap-fm* *a b A*)|  
*swap-fm* *a b* (*ExQ* *A*) = *ExQ*(*swap-fm* (*a+1*) (*b+1*) *A*)|  
*swap-fm* *a b* (*AllQ* *A*) = *AllQ*(*swap-fm* (*a+1*) (*b+1*) *A*)|  
*swap-fm* *a b* (*ExN* *i A*) = *ExN* *i* (*swap-fm* (*a+i*) (*b+i*) *A*)|  
*swap-fm* *a b* (*AllN* *i A*) = *AllN* *i* (*swap-fm* (*a+i*) (*b+i*) *A*)

**fun** *swap-list* :: *nat*  $\Rightarrow$  *nat*  $\Rightarrow$  '*a list*  $\Rightarrow$  '*a list***where**

*swap-list* *i j l* = *l*[*j* := *nth l i*, *i* := *nth l j*]

**lemma** *swap-list-cons*: *swap-list* (*Suc* *a*) (*Suc* *b*) (*x* # *L*) = *x* # *swap-list* *a b L*

*<proof>*

**lemma** *inj-on* : *inj-on* (*swap0* *a b*) (*monomials* *p*)

*<proof>*

**lemma** *inj-on'* : *inj-on* (*swap* *a b*) (*keys* *m*)

*<proof>*

**lemma** *swap-list* :

**assumes** *a* < *length L*

**assumes** *b* < *length L*

**shows** *nth-default* 0 (*L*[*b* := *L* ! *a*, *a* := *L* ! *b*]) (*swap* *a b xa*) = *nth-default* 0 *L xa*

*<proof>*

**lemma** *swap-poly* :

**assumes** *length L* > *a*

**assumes** *length L* > *b*

**shows** *insertion* (*nth-default* 0 *L*) *p* = *insertion* (*nth-default* 0 (*swap-list* *a b L*)) (*swap-poly* *a b p*)

*<proof>*

**lemma** *swap-fm* :

**assumes** *length L* > *a*

**assumes** *length L* > *b*

**shows**  $eval\ F\ L = eval\ (swap-fm\ a\ b\ F)\ (swap-list\ a\ b\ L)$   
 ⟨proof⟩

**lemma**  $eval\ (ExQ\ (ExQ\ F))\ L = eval\ (ExQ\ (ExQ\ (swap-fm\ 0\ 1\ F)))\ L$   
 ⟨proof⟩

**lemma** *swap-atom*:

**assumes**  $length\ L > a$

**assumes**  $length\ L > b$

**shows**  $aEval\ F\ L = aEval\ (swap-atom\ a\ b\ F)\ (swap-list\ a\ b\ L)$

⟨proof⟩

**end**

## 6 Optimizations

**theory** *Optimizations*

**imports** *Debruijn*

**begin**

Does negation normal form conversion

**fun** *nnf* ::  $atom\ fm \Rightarrow atom\ fm$  **where**

$nnf\ TrueF = TrueF$  |

$nnf\ FalseF = FalseF$  |

$nnf\ (Atom\ a) = Atom\ a$  |

$nnf\ (And\ \varphi_1\ \varphi_2) = And\ (nnf\ \varphi_1)\ (nnf\ \varphi_2)$  |

$nnf\ (Or\ \varphi_1\ \varphi_2) = Or\ (nnf\ \varphi_1)\ (nnf\ \varphi_2)$  |

$nnf\ (ExQ\ \varphi) = ExQ\ (nnf\ \varphi)$  |

$nnf\ (AllQ\ \varphi) = AllQ\ (nnf\ \varphi)$  |

$nnf\ (AllN\ i\ \varphi) = AllN\ i\ (nnf\ \varphi)$  |

$nnf\ (ExN\ i\ \varphi) = ExN\ i\ (nnf\ \varphi)$  |

$nnf\ (Neg\ TrueF) = FalseF$  |

$nnf\ (Neg\ FalseF) = TrueF$  |

$nnf\ (Neg\ (Neg\ \varphi)) = (nnf\ \varphi)$  |

$nnf\ (Neg\ (And\ \varphi_1\ \varphi_2)) = (Or\ (nnf\ (Neg\ \varphi_1))\ (nnf\ (Neg\ \varphi_2)))$  |

$nnf\ (Neg\ (Or\ \varphi_1\ \varphi_2)) = (And\ (nnf\ (Neg\ \varphi_1))\ (nnf\ (Neg\ \varphi_2)))$  |

$nnf\ (Neg\ (Atom\ a)) = Atom\ (aNeg\ a)$  |

$nnf\ (Neg\ (ExQ\ \varphi)) = AllQ\ (nnf\ (Neg\ \varphi))$  |

$nnf\ (Neg\ (AllQ\ \varphi)) = ExQ\ (nnf\ (Neg\ \varphi))$  |

$nnf\ (Neg\ (AllN\ i\ \varphi)) = ExN\ i\ (nnf\ (Neg\ \varphi))$  |

$nnf\ (Neg\ (ExN\ i\ \varphi)) = AllN\ i\ (nnf\ (Neg\ \varphi))$

### 6.1 Simplify Constants

**fun** *simp-atom* ::  $atom \Rightarrow atom\ fm$  **where**

$simp-atom\ (Eq\ p) = (case\ get-if-const\ p\ of\ None \Rightarrow Atom\ (Eq\ p) \mid Some\ (r) \Rightarrow$   
 (if  $r=0$  then  $TrueF$  else  $FalseF$ )) |

$simp-atom\ (Less\ p) = (case\ get-if-const\ p\ of\ None \Rightarrow Atom\ (Less\ p) \mid Some\ (r) \Rightarrow$   
 (if  $r<0$  then  $TrueF$  else  $FalseF$ )) |

```

simp-atom (Leq p) = (case get-if-const p of None  $\Rightarrow$  Atom(Leq p) | Some(r)  $\Rightarrow$ 
(if r $\leq$ 0 then TrueF else FalseF))|
simp-atom (Neq p) = (case get-if-const p of None  $\Rightarrow$  Atom(Neq p) | Some(r)
 $\Rightarrow$  (if r $\neq$ 0 then TrueF else FalseF))

```

```

fun simpfm :: atom fm  $\Rightarrow$  atom fm where
  simpfm TrueF = TrueF|
  simpfm FalseF = FalseF|
  simpfm (Atom a) = simp-atom a|
  simpfm (And  $\varphi$   $\psi$ ) = and (simpfm  $\varphi$ ) (simpfm  $\psi$ )|
  simpfm (Or  $\varphi$   $\psi$ ) = or (simpfm  $\varphi$ ) (simpfm  $\psi$ )|
  simpfm (ExQ  $\varphi$ ) = ExQ (simpfm  $\varphi$ )|
  simpfm (Neg  $\varphi$ ) = neg (simpfm  $\varphi$ )|
  simpfm (AllQ  $\varphi$ ) = AllQ (simpfm  $\varphi$ )|
  simpfm (AllN i  $\varphi$ ) = AllN i (simpfm  $\varphi$ )|
  simpfm (ExN i  $\varphi$ ) = ExN i (simpfm  $\varphi$ )

```

## 6.2 Group Quantifiers

```

fun groupQuantifiers :: atom fm  $\Rightarrow$  atom fm where
  groupQuantifiers TrueF = TrueF|
  groupQuantifiers FalseF = FalseF|
  groupQuantifiers (And A B) = And (groupQuantifiers A) (groupQuantifiers B)|
  groupQuantifiers (Or A B) = Or (groupQuantifiers A) (groupQuantifiers B)|
  groupQuantifiers (Neg A) = Neg (groupQuantifiers A)|
  groupQuantifiers (Atom A) = Atom A|
  groupQuantifiers (ExQ (ExQ A)) = groupQuantifiers (ExN 2 A)|
  groupQuantifiers (ExQ (ExN j A)) = groupQuantifiers (ExN (j+1) A)|
  groupQuantifiers (ExN j (ExQ A)) = groupQuantifiers (ExN (j+1) A)|
  groupQuantifiers (ExN i (ExN j A)) = groupQuantifiers (ExN (i+j) A)|
  groupQuantifiers (ExQ A) = ExQ (groupQuantifiers A)|
  groupQuantifiers (AllQ (AllQ A)) = groupQuantifiers (AllN 2 A)|
  groupQuantifiers (AllQ (AllN j A)) = groupQuantifiers (AllN (j+1) A)|
  groupQuantifiers (AllN j (AllQ A)) = groupQuantifiers (AllN (j+1) A)|
  groupQuantifiers (AllN i (AllN j A)) = groupQuantifiers (AllN (i+j) A)|
  groupQuantifiers (AllQ A) = AllQ (groupQuantifiers A)|
  groupQuantifiers (AllN j A) = AllN j A|
  groupQuantifiers (ExN j A) = ExN j A

```

## 6.3 Clear Quantifiers

clearQuantifiers F goes through the formula F and removes all quantifiers whose variables are not present in the formula. For example, clearQuantifiers (ExQ(TrueF)) evaluates to TrueF. This preserves the truth value of the formula as shown in the clearQuantifiers\_eval proof. This is used within the QE overall procedure to eliminate quantifiers in the cases where QE was successful.

```

fun depth' :: 'a fm  $\Rightarrow$  natwhere

```

```

depth' TrueF = 1|
depth' FalseF = 1|
depth' (Atom -) = 1|
depth' (And  $\varphi \psi$ ) = max (depth'  $\varphi$ ) (depth'  $\psi$ ) + 1|
depth' (Or  $\varphi \psi$ ) = max (depth'  $\varphi$ ) (depth'  $\psi$ ) + 1|
depth' (Neg  $\varphi$ ) = depth'  $\varphi$  + 1|
depth' (ExQ  $\varphi$ ) = depth'  $\varphi$  + 1|
depth' (AllQ  $\varphi$ ) = depth'  $\varphi$  + 1|
depth' (AllN  $i \varphi$ ) = depth'  $\varphi$  +  $i * 2 + 1$ |
depth' (ExN  $i \varphi$ ) = depth'  $\varphi$  +  $i * 2 + 1$ 

```

**function** *clearQuantifiers* :: *atom fm*  $\Rightarrow$  *atom fm* **where**

```

clearQuantifiers TrueF = TrueF|
clearQuantifiers FalseF = FalseF|
clearQuantifiers (Atom a) = simp-atom a|
clearQuantifiers (And  $\varphi \psi$ ) = and (clearQuantifiers  $\varphi$ ) (clearQuantifiers  $\psi$ )|
clearQuantifiers (Or  $\varphi \psi$ ) = or (clearQuantifiers  $\varphi$ ) (clearQuantifiers  $\psi$ )|
clearQuantifiers (Neg  $\varphi$ ) = neg (clearQuantifiers  $\varphi$ )|
clearQuantifiers (ExQ  $\varphi$ ) =
  (let  $\varphi'$  = clearQuantifiers  $\varphi$  in
   (if freeIn 0  $\varphi'$  then lowerFm 0 1  $\varphi'$  else ExQ  $\varphi'$ ))|
clearQuantifiers (AllQ  $\varphi$ ) =
  (let  $\varphi'$  = clearQuantifiers  $\varphi$  in
   (if freeIn 0  $\varphi'$  then lowerFm 0 1  $\varphi'$  else AllQ  $\varphi'$ ))|
clearQuantifiers (ExN 0  $\varphi$ ) = clearQuantifiers  $\varphi$ |
clearQuantifiers (ExN (Suc i)  $\varphi$ ) = clearQuantifiers (ExN i (ExQ  $\varphi$ ))|
clearQuantifiers (AllN 0  $\varphi$ ) = clearQuantifiers  $\varphi$ |
clearQuantifiers (AllN (Suc i)  $\varphi$ ) = clearQuantifiers (AllN i (AllQ  $\varphi$ ))
<proof>

```

**termination**

```

<proof>

```

## 6.4 Push Forall

**fun** *push-forall* :: *atom fm*  $\Rightarrow$  *atom fm* **where**

```

push-forall TrueF = TrueF|
push-forall FalseF = FalseF|
push-forall (Atom a) = simp-atom a|
push-forall (And  $\varphi \psi$ ) = and (push-forall  $\varphi$ ) (push-forall  $\psi$ )|
push-forall (Or  $\varphi \psi$ ) = or (push-forall  $\varphi$ ) (push-forall  $\psi$ )|
push-forall (ExQ  $\varphi$ ) = ExQ (push-forall  $\varphi$ )|
push-forall (ExN i  $\varphi$ ) = ExN i (push-forall  $\varphi$ )|
push-forall (Neg  $\varphi$ ) = neg (push-forall  $\varphi$ )|
push-forall (AllQ TrueF) = TrueF|
push-forall (AllQ FalseF) = FalseF|
push-forall (AllQ (Atom a)) = (if freeIn 0 (Atom a) then Atom(lowerAtom 0 1
a) else AllQ (Atom a))|
push-forall (AllQ (And  $\varphi \psi$ )) = and (push-forall (AllQ  $\varphi$ )) (push-forall (AllQ
 $\psi$ ))|

```

```

push-forall (AllQ (Or  $\varphi$   $\psi$ )) = (
  if freeIn 0  $\varphi$ 
  then(
    if freeIn 0  $\psi$ 
    then or (lowerFm 0 1  $\varphi$ ) (lowerFm 0 1  $\psi$ )
    else or (lowerFm 0 1  $\varphi$ ) (AllQ  $\psi$ )
  else (
    if freeIn 0  $\psi$ 
    then or (AllQ  $\varphi$ ) (lowerFm 0 1  $\psi$ )
    else AllQ (or  $\varphi$   $\psi$ )
  )|
push-forall (AllQ  $\varphi$ ) = (if freeIn 0  $\varphi$  then lowerFm 0 1  $\varphi$  else AllQ  $\varphi$ )|
push-forall (AllN  $i$   $\varphi$ ) = AllN  $i$  (push-forall  $\varphi$ )

```

## 6.5 Unpower

**fun** to-list :: nat  $\Rightarrow$  real mpoly  $\Rightarrow$  (real mpoly \* nat) list **where**  
to-list  $v$   $p$  = [(isolate-variable-sparse  $p$   $v$   $x$ ,  $x$ ).  $x$   $\leftarrow$  [0.. $(MPoly-Type.degree$   $p$   $v$ )+1]]

**fun** chop :: (real mpoly \* nat) list  $\Rightarrow$  (real mpoly \* nat) list **where**  
chop [] = []  
chop (( $p,i$ )# $L$ ) = (if  $p=0$  then chop  $L$  else ( $p,i$ )# $L$ )

**fun** decreasePower :: nat  $\Rightarrow$  real mpoly  $\Rightarrow$  real mpoly \* nat **where**  
decreasePower  $v$   $p$  = (case chop (to-list  $v$   $p$ ) of []  $\Rightarrow$  ( $p,0$ ) | (( $p,i$ )# $L$ )  $\Rightarrow$  (sum-list [term \* (Var  $v$ )  $\wedge$  ( $x-i$ ). (term, $x$ ) $\leftarrow$ (( $p,i$ )# $L$ ), $i$ )

**fun** unpower :: nat  $\Rightarrow$  atom fm  $\Rightarrow$  atom fm **where**  
unpower  $v$  (Atom (Eq  $p$ )) = (case decreasePower  $v$   $p$  of ( $-,0$ )  $\Rightarrow$  Atom(Eq  $p$ ) | ( $p,-$ )  $\Rightarrow$  Or(Atom (Eq  $p$ ))(Atom (Eq (Var  $v$ ))) )|  
unpower  $v$  (Atom (Neq  $p$ )) = (case decreasePower  $v$   $p$  of ( $-,0$ )  $\Rightarrow$  Atom(Neq  $p$ ) | ( $p,-$ )  $\Rightarrow$  And(Atom (Neq  $p$ ))(Atom (Neq (Var  $v$ ))) )|  
unpower  $v$  (Atom (Less  $p$ )) = (case decreasePower  $v$   $p$  of ( $-,0$ )  $\Rightarrow$  Atom(Less  $p$ ) | ( $p,n$ )  $\Rightarrow$   
 if  $n \bmod 2 = 0$  then  
 And(Atom (Less  $p$ ))(Atom(Neq (Var  $v$ )))  
 else  
 Or  
 (And (Atom (Less ( $p$ ))) (Atom (Less ( $-$  Var  $v$ ))))  
 (And (Atom (Less ( $-p$ ))) (Atom (Less (Var  $v$ ))))  
 )|  
unpower  $v$  (Atom (Leq  $p$ )) = (case decreasePower  $v$   $p$  of ( $-,0$ )  $\Rightarrow$  Atom(Leq  $p$ ) | ( $p,n$ )  $\Rightarrow$   
 if  $n \bmod 2 = 0$  then  
 Or (Atom (Leq  $p$ )) (Atom (Eq (Var  $v$ )))  
 else  
 Or (Atom (Eq  $p$ ))  
 (Or

```

      (And (Atom (Less ( p))) (Atom (Leq (- Var v))))
      (And (Atom (Less (-p)) (Atom (Leq (Var v))))))
    )|
  unpower v (And a b) = And (unpower v a) (unpower v b)|
  unpower v (Or a b) = Or (unpower v a) (unpower v b)|
  unpower v (Neg a) = Neg (unpower v a)|
  unpower v (TrueF) = TrueF|
  unpower v (FalseF) = FalseF|
  unpower v (AllQ F) = AllQ(unpower (v+1) F)|
  unpower v (ExQ F) = ExQ (unpower (v+1) F)|
  unpower v (AllN x F) = AllN x (unpower (v+x) F)|
  unpower v (ExN x F) = ExN x (unpower (v+x) F)

```

**end**

## 6.6 Optimization Proofs

```

theory OptimizationProofs
  imports Optimizations
begin

```

```

lemma neg-nnf :  $\forall \Gamma. (\neg \text{eval (nnf (Neg } \varphi)) \Gamma) = \text{eval (nnf } \varphi) \Gamma$ 
  <proof>

```

```

theorem eval-nnf :  $\forall \Gamma. \text{eval } \varphi \Gamma = \text{eval (nnf } \varphi) \Gamma$ 
  <proof>

```

```

theorem negation-free-nnf : negation-free (nnf  $\varphi$ )
  <proof>

```

```

lemma groupQuantifiers-eval :  $\text{eval } F L = \text{eval (groupQuantifiers } F) L$ 
  <proof>

```

```

theorem simp-atom-eval :  $a\text{Eval } a \text{ } xs = \text{eval (simp-atom } a) \text{ } xs$ 
  <proof>

```

```

lemma simpfm-eval :  $\forall L. \text{eval } \varphi L = \text{eval (simpfm } \varphi) L$ 
  <proof>

```

```

lemma exQ-clearQuantifiers:
  assumes ExQ :  $\bigwedge xs. \text{eval (clearQuantifiers } \varphi) \text{ } xs = \text{eval } \varphi \text{ } xs$ 
  shows  $\text{eval (clearQuantifiers (ExQ } \varphi)) \text{ } xs = \text{eval (ExQ } \varphi) \text{ } xs$ 
  <proof>

```

**lemma** *allQ-clearQuantifiers* :

**assumes**  $AllQ : \bigwedge xs. eval (clearQuantifiers \varphi) xs = eval \varphi xs$

**shows**  $eval (clearQuantifiers (AllQ \varphi)) xs = eval (AllQ \varphi) xs$

*<proof>*

**lemma** *clearQuantifiers-eval* :  $eval (clearQuantifiers \varphi) xs = eval \varphi xs$

*<proof>*

**lemma** *push-forall-eval-AllQ* :  $\forall xs. eval (AllQ \varphi) xs = eval (push-forall (AllQ \varphi)) xs$

*<proof>*

**lemma** *push-forall-eval* :  $\forall xs. eval \varphi xs = eval (push-forall \varphi) xs$

*<proof>*

**lemma** *map-fm-binders-negation-free* :

**assumes** *negation-free*  $\varphi$

**shows** *negation-free*  $(map-fm-binders f \varphi n)$

*<proof>*

**lemma** *negation-free-and* :

**assumes** *negation-free*  $\varphi$

**assumes** *negation-free*  $\psi$

**shows** *negation-free*  $(and \varphi \psi)$

*<proof>*

**lemma** *negation-free-or* :

**assumes** *negation-free*  $\varphi$

**assumes** *negation-free*  $\psi$

**shows** *negation-free*  $(or \varphi \psi)$

*<proof>*

**lemma** *push-forall-negation-free-all* :

**assumes** *negation-free*  $\varphi$

**shows** *negation-free*  $(push-forall (AllQ \varphi))$

*<proof>*

**lemma** *push-forall-negation-free* :

**assumes** *negation-free*  $\varphi$

**shows** *negation-free*  $(push-forall \varphi)$

*<proof>*

**lemma** *to-list-insertion*:  $insertion f p = sum-list [insertion f term * (f v) \wedge i. (term, i) \leftarrow (to-list v p)]$

*<proof>*

**lemma** *to-list-p*:  $p = sum-list [term * (Var v) \wedge i. (term, i) \leftarrow (to-list v p)]$

*<proof>*

**fun** *chophelper* :: (real mpoly \* nat) list  $\Rightarrow$  (real mpoly \* nat) list  $\Rightarrow$  (real mpoly \* nat) list \* (real mpoly \* nat) list **where**  
*chophelper* [] L = (L,[])|  
*chophelper* ((p,i)#L) R = (if p=0 then *chophelper* L (R @ [(p,i)]) else (R,(p,i)#L))

**lemma** *preserve* :  
**assumes** (a,b)=*chophelper* L L'  
**shows** a@b=L'@L  
 <proof>

**lemma** *compare* :  
**assumes** (a,b)=*chophelper* L L'  
**shows** chop L = b  
 <proof>

**lemma** *allzero*:  
**assumes**  $\forall (p,i) \in \text{set}(L'). p=0$   
**assumes** (a,b)=*chophelper* L L'  
**shows**  $\forall (p,i) \in \text{set}(a). p=0$   
 <proof>

**lemma** *separate*:  
**assumes** (a,b)=*chophelper* (to-list v p) []  
**shows** insertion f p = sum-list [insertion f term \* (f v)  $\wedge$  i. (term,i) $\leftarrow$ a] +  
 sum-list [insertion f term \* (f v)  $\wedge$  i. (term,i) $\leftarrow$ b]  
 <proof>

**lemma** *chopped* :  
**assumes** (a,b)=*chophelper* (to-list v p) []  
**shows** insertion f p = sum-list [insertion f term \* (f v)  $\wedge$  i. (term,i) $\leftarrow$ b]  
 <proof>

**lemma** *insertion-chop* :  
**shows** insertion f p = sum-list [insertion f term \* (f v)  $\wedge$  i. (term,i) $\leftarrow$ (chop  
 (to-list v p))]  
 <proof>

**lemma** *sorted* : sorted-wrt ( $\lambda(-,i).\lambda(-,i'). i < i'$ ) (to-list v p)  
 <proof>

**lemma** *sublist* : sublist (chop L) L  
 <proof>

**lemma** *move-exp* :  
**assumes** (p',i)#L = (chop (to-list v p))  
**shows** insertion f p = sum-list [insertion f term \* (f v)  $\wedge$  (d-i). (term,d) $\leftarrow$ (chop  
 (to-list v p))] \* (f v)  $\wedge$  i

*<proof>*

**lemma** *insert-Var-Zero* : *insertion f (Var v) = f v*  
*<proof>*

**lemma** *decreasePower-insertion* :  
  **assumes** *decreasePower v p = (p',i)*  
  **shows** *insertion f p = insertion f p' \* (f v) ^ i*  
*<proof>*

**lemma** *unpower-eval*: *eval (unpower v  $\varphi$ ) L = eval  $\varphi$  L*  
*<proof>*

**lemma** *to-list-filter*: *p = sum-list [term \* (Var v) ^ i. (term,i) ← ((filter ( $\lambda(x,-)$ .  
 $x \neq 0$ ) (to-list v p)))]*  
*<proof>*

**end**

## 7 Algorithms

### 7.1 Equality VS Helper Functions

**theory** *VSAlgos*  
  **imports** *Debruijn Optimizations*  
**begin**

This is a subprocess which simply separates out the equality atoms from the other kinds of atoms

Note that we search for equality atoms that are of degree one or two

This is used within the equalityVS algorithm

**fun** *find-eq* :: *nat*  $\Rightarrow$  *atom list*  $\Rightarrow$  *real mpoly list* \* *atom list* **where**  
  *find-eq var [] = ([], [])* |  
  *find-eq var ((Less p)#as) = (let (A,B) = find-eq var as in (A,Less p#B))* |  
  *find-eq var ((Eq p)#as) = (let (A,B) = find-eq var as in*  
    *if MPoly-Type.degree p var < 3  $\wedge$  MPoly-Type.degree p var  $\neq$  0*  
    *then (p # A,B)*  
    *else (A,Eq p # B)*  
  )|  
  *find-eq var ((Leq p)#as) = (let (A,B) = find-eq var as in (A,Leq p#B))* |  
  *find-eq var ((Neq p)#as) = (let (A,B) = find-eq var as in (A,Neq p#B))*

```

fun split-p :: nat ⇒ real mpoly ⇒ atom fm where
  split-p var p = And (Atom (Eq (isolate-variable-sparse p var 2)))
                    (And (Atom (Eq (isolate-variable-sparse p var 1)))
                        (Atom (Eq (isolate-variable-sparse p var 0))))

```

The linearsubstitution virtually substitutes in an equation of  $b * x + c = 0$  into an arbitrary atom

linearsubstitution x b c (Eq p) = F corresponds to removing variable x from polynomial p and replacing it with an equivalent function F where F doesn't mention variable x

If there exists a way to assign variables that makes  $p = 0$  true, then that same set of variables will make F true

If there exists a way to assign variables that makes F true and also have  $b*x+c=0$ , then that same set of variables will make  $p=0$  true

Same applies for other kinds of atoms that aren't equality

```

fun linear-substitution :: nat ⇒ real mpoly ⇒ real mpoly ⇒ atom ⇒ atom where
  linear-substitution var a b (Eq p) =
    (let d = MPoly-Type.degree p var in
     (Eq (∑ i∈{0..<(d+1)}. isolate-variable-sparse p var i * (a^i) * (b^(d-i)))))
  ) |
  linear-substitution var a b (Less p) =
    (let d = MPoly-Type.degree p var in
     let P = (∑ i∈{0..<(d+1)}. isolate-variable-sparse p var i * (a^i) * (b^(d-i)))))
    in
    (Less(P * (b^(d mod 2)))))
  ) |
  linear-substitution var a b (Leq p) =
    (let d = MPoly-Type.degree p var in
     let P = (∑ i∈{0..<(d+1)}. isolate-variable-sparse p var i * (a^i) * (b^(d-i)))))
    in
    (Leq(P * (b^(d mod 2)))))
  ) |
  linear-substitution var a b (Neq p) =
    (let d = MPoly-Type.degree p var in
     (Neq (∑ i∈{0..<(d+1)}. isolate-variable-sparse p var i * (a^i) * (b^(d-i)))))
    )

```

```

fun linear-substitution-fm-helper :: nat ⇒ real mpoly ⇒ real mpoly ⇒ atom fm ⇒
nat ⇒ atom fm where
  linear-substitution-fm-helper var b c F z = liftmap (λx.λA. Atom(linear-substitution
(var+x) (liftPoly 0 x b) (liftPoly 0 x c) A)) F z

```

```

fun linear-substitution-fm :: nat ⇒ real mpoly ⇒ real mpoly ⇒ atom fm ⇒ atom
fm where
  linear-substitution-fm var b c F = linear-substitution-fm-helper var b c F 0

```

quadraticpart1 var a b A takes in an expression of the form  $(a+b * \text{sqrt}(c))/d$

for an arbitrary  $c$  and substitutes it in for the variable  $var$  in the atom  $A$

**fun** *quadratic-part-1* :: *nat*  $\Rightarrow$  *real mpoly*  $\Rightarrow$  *real mpoly*  $\Rightarrow$  *real mpoly*  $\Rightarrow$  *atom*  $\Rightarrow$  *real mpoly* **where**

```

  quadratic-part-1 var a b d (Eq p) = (
    let deg = MPoly-Type.degree p var in
     $\sum_{i \in \{0..<(deg+1)\}}$ . (isolate-variable-sparse p var i) * ((a+b*(Var var))i) *
    (d(deg - i))
  ) |
  quadratic-part-1 var a b d (Less p) = (
    let deg = MPoly-Type.degree p var in
    let P =  $\sum_{i \in \{0..<(deg+1)\}}$ . (isolate-variable-sparse p var i) * ((a+b*(Var
var))i) * (d(deg - i)) in
    P * (d(deg mod 2))
  ) |
  quadratic-part-1 var a b d (Leq p) = (
    let deg = MPoly-Type.degree p var in
    let P =  $\sum_{i \in \{0..<(deg+1)\}}$ . (isolate-variable-sparse p var i) * ((a+b*(Var
var))i) * (d(deg - i)) in
    P * (d(deg mod 2))
  ) |
  quadratic-part-1 var a b d (Neq p) = (
    let deg = MPoly-Type.degree p var in
     $\sum_{i \in \{0..<(deg+1)\}}$ . (isolate-variable-sparse p var i) * ((a+b*(Var var))i) *
    (d(deg - i))
  )

```

**fun** *quadratic-part-2* :: *nat*  $\Rightarrow$  *real mpoly*  $\Rightarrow$  *real mpoly*  $\Rightarrow$  *real mpoly* **where**

```

  quadratic-part-2 var sq p = (
    let deg = MPoly-Type.degree p var in
     $\sum_{i \in \{0..<deg+1\}}$ .
    (isolate-variable-sparse p var i)*(sq(i div 2)) * (Const(of-nat(i mod 2))) *
    (Var var)
    +(isolate-variable-sparse p var i)*(sq(i div 2)) * Const(1-of-nat(i mod 2))
  )

```

*quadratic-sub* var a b c d A represents virtually substituting an expression of the form  $(a+b*\sqrt{c})/d$  into variable  $var$  in atom  $A$

**primrec** *quadratic-sub* :: *nat*  $\Rightarrow$  *real mpoly*  $\Rightarrow$  *real mpoly*  $\Rightarrow$  *real mpoly*  $\Rightarrow$  *real mpoly*  $\Rightarrow$  *atom*  $\Rightarrow$  *atom fm* **where**

```

  quadratic-sub var a b c d (Eq p) = (
    let (p1::real mpoly) = quadratic-part-1 var a b d (Eq p) in
    let (p2::real mpoly) = quadratic-part-2 var c p1 in
    let (A::real mpoly) = isolate-variable-sparse p2 var 0 in
    let (B::real mpoly) = isolate-variable-sparse p2 var 1 in
    And
    (Atom(Leq (A*B)))
    (Atom (Eq (A2-B2*c)))
  ) |

```

```

quadratic-sub var a b c d (Less p) = (
  let (p1::real mpoly) = quadratic-part-1 var a b d (Less p) in
  let (p2::real mpoly) = quadratic-part-2 var c p1 in
  let (A::real mpoly) = isolate-variable-sparse p2 var 0 in
  let (B::real mpoly) = isolate-variable-sparse p2 var 1 in
  Or
  (And
    (Atom(Less(A)))
    (Atom (Less (B^2*c-A^2))))
  (And
    (Atom(Leq B))
    (Or
      (Atom(Less A))
      (Atom(Less (A^2-B^2*c))))))
) |
quadratic-sub var a b c d (Leq p) = (
  let (p1::real mpoly) = quadratic-part-1 var a b d (Leq p) in
  let (p2::real mpoly) = quadratic-part-2 var c p1 in
  let (A::real mpoly) = isolate-variable-sparse p2 var 0 in
  let (B::real mpoly) = isolate-variable-sparse p2 var 1 in
  Or
  (And
    (Atom(Leq(A)))
    (Atom (Leq(B^2*c-A^2))))
  (And
    (Atom(Leq B))
    (Atom(Leq (A^2-B^2*c))))
) |
quadratic-sub var a b c d (Neq p) = (
  let (p1::real mpoly) = quadratic-part-1 var a b d (Neq p) in
  let (p2::real mpoly) = quadratic-part-2 var c p1 in
  let (A::real mpoly) = isolate-variable-sparse p2 var 0 in
  let (B::real mpoly) = isolate-variable-sparse p2 var 1 in
  Or
  (Atom(Less(-A*B)))
  (Atom (Neq(A^2-B^2*c)))
)

```

```

fun quadratic-sub-fm-helper :: nat ⇒ real mpoly ⇒ real mpoly ⇒ real mpoly ⇒
real mpoly ⇒ atom fm ⇒ nat ⇒ atom fm where
  quadratic-sub-fm-helper var a b c d F z = liftmap (λx.λA. quadratic-sub (var+x)
(liftPoly 0 x a) (liftPoly 0 x b) (liftPoly 0 x c) (liftPoly 0 x d) A) F z

```

```

fun quadratic-sub-fm :: nat ⇒ real mpoly ⇒ real mpoly ⇒ real mpoly ⇒ real mpoly
⇒ atom fm ⇒ atom fm where
  quadratic-sub-fm var a b c d F = quadratic-sub-fm-helper var a b c d F 0

```

## 7.2 General VS Helper Functions

**fun** *allZero* :: *real mpoly*  $\Rightarrow$  *nat*  $\Rightarrow$  *atom fm* **where**

*allZero* *p* *var* = *list-conj* [*Atom*(*Eq*(*isolate-variable-sparse* *p* *var* *i*)). *i* < - [0..*(MPoly-Type.degree* *p* *var*)+1]]

**fun** *alternateNegInfinity* :: *real mpoly*  $\Rightarrow$  *nat*  $\Rightarrow$  *atom fm* **where**

*alternateNegInfinity* *p* *var* = *foldl* ( $\lambda F.\lambda i.$   
*let* *a-n* = *isolate-variable-sparse* *p* *var* *i* *in*  
*let* *exp* = (*if* *i mod 2 = 0* *then* *Const*(1) *else* *Const*(-1)) *in*  
*or* (*Atom*(*Less* (*exp* \* *a-n*)))  
(*and* (*Atom* (*Eq* *a-n*)) *F*)  
) *FalseF* ([0..*(MPoly-Type.degree* *p* *var*)+1])

**fun** *substNegInfinity* :: *nat*  $\Rightarrow$  *atom*  $\Rightarrow$  *atom fm* **where**

*substNegInfinity* *var* (*Eq* *p*) = *allZero* *p* *var* |  
*substNegInfinity* *var* (*Less* *p*) = *alternateNegInfinity* *p* *var* |  
*substNegInfinity* *var* (*Leq* *p*) = *Or* (*alternateNegInfinity* *p* *var*) (*allZero* *p* *var*) |  
*substNegInfinity* *var* (*Neq* *p*) = *Neg* (*allZero* *p* *var*)

**function** *convertDerivative* :: *nat*  $\Rightarrow$  *real mpoly*  $\Rightarrow$  *atom fm* **where**

*convertDerivative* *var* *p* = (*if* (*MPoly-Type.degree* *p* *var*) = 0 *then* *Atom* (*Less* *p*)  
*else*  
*Or* (*Atom* (*Less* *p*)) (*And* (*Atom*(*Eq* *p*)) (*convertDerivative* *var* (*derivative* *var*  
*p*))))  
<*proof*>

**termination**

<*proof*>

**fun** *substInfinitesimalLinear* :: *nat*  $\Rightarrow$  *real mpoly*  $\Rightarrow$  *real mpoly*  $\Rightarrow$  *atom*  $\Rightarrow$  *atom*  
*fm* **where**

*substInfinitesimalLinear* *var* *b* *c* (*Eq* *p*) = *allZero* *p* *var* |  
*substInfinitesimalLinear* *var* *b* *c* (*Less* *p*) =  
*liftmap*  
( $\lambda x. \lambda A. \text{Atom}(\text{linear-substitution } (\text{var}+x) (\text{liftPoly } 0 \ x \ b) (\text{liftPoly } 0 \ x \ c) \ A)$ )  
(*convertDerivative* *var* *p*)  
0 |  
*substInfinitesimalLinear* *var* *b* *c* (*Leq* *p*) =  
*Or*  
(*allZero* *p* *var*)  
(*liftmap*  
( $\lambda x. \lambda A. \text{Atom}(\text{linear-substitution } (\text{var}+x) (\text{liftPoly } 0 \ x \ b) (\text{liftPoly } 0 \ x \ c) \ A)$ )  
(*convertDerivative* *var* *p*)  
0) |  
*substInfinitesimalLinear* *var* *b* *c* (*Neq* *p*) = *neg* (*allZero* *p* *var*)

```

fun substInfinitesimalQuadratic :: nat ⇒ real mpoly ⇒ real mpoly ⇒ real mpoly
⇒ real mpoly ⇒ atom ⇒ atom fm where
  substInfinitesimalQuadratic var a b c d (Eq p) = allZero p var|
  substInfinitesimalQuadratic var a b c d (Less p) = quadratic-sub-fm var a b c d
(convertDerivative var p)|
  substInfinitesimalQuadratic var a b c d (Leq p) =
Or
  (allZero p var)
  (quadratic-sub-fm var a b c d (convertDerivative var p))|
  substInfinitesimalQuadratic var a b c d (Neq p) = neg (allZero p var)

```

```

fun substInfinitesimalLinear-fm :: nat ⇒ real mpoly ⇒ real mpoly ⇒ atom fm ⇒
atom fm where
  substInfinitesimalLinear-fm var b c F = liftmap (λx.λA. substInfinitesimalLinear
(var+x) (liftPoly 0 x b) (liftPoly 0 x c) A) F 0

```

```

fun substInfinitesimalQuadratic-fm :: nat ⇒ real mpoly ⇒ real mpoly ⇒ real mpoly
⇒ real mpoly ⇒ atom fm ⇒ atom fm where
  substInfinitesimalQuadratic-fm var a b c d F = liftmap (λx.λA. substInfinitesi-
malQuadratic (var+x) (liftPoly 0 x a) (liftPoly 0 x b) (liftPoly 0 x c) (liftPoly 0 x
d) A) F 0

```

### 7.3 VS Algorithms

elimVar var L F attempts to do quadratic elimination on the variable defined by var. L is the list of conjunctive atoms, F is a list of unnecessary garbage

```

fun elimVar :: nat ⇒ atom list ⇒ (atom fm) list ⇒ atom ⇒ atom fm where
  elimVar var L F (Eq p) = (
let (a,b,c) = get-coeffs var p in

```

(Or

```

  (And (And (Atom (Eq a)) (Atom (Neq b)))
(list-conj (
  (map (λa. Atom (linear-substitution var (-c) b a)) L)@
  (map (linear-substitution-fm var (-c) b) F)
)))

```

```

  (And (Atom (Neq a)) (And (Atom (Leq (-(b^2)+4*a*c)))
(Or (list-conj (
  (map (quadratic-sub var (-b) 1 (b^2-4*a*c) (2*a)) L)@
  (map (quadratic-sub-fm var (-b) 1 (b^2-4*a*c) (2*a)) F)
)))
(list-conj (
  (map (quadratic-sub var (-b) (-1) (b^2-4*a*c) (2*a)) L)@

```

```

      (map (quadratic-sub-fm var (-b) (-1) (b^2-4*a*c) (2*a)) F)
    ))
  ))
) |
elimVar var L F (Less p) = (
let (a,b,c) = get-coeffs var p in
(Or

(And (And (Atom (Eq a)) (Atom (Neq b)))
(list-conj (
(map (substInfinitesimalLinear var (-c) b) L)
@ (map (substInfinitesimalLinear-fm var (-c) b) F)
)))

(And (Atom (Neq a)) (And (Atom (Leq (-b^2)+4*a*c)))
(Or (list-conj (
(map (substInfinitesimalQuadratic var (-b) 1 (b^2-4*a*c) (2*a)) L)@
(map (substInfinitesimalQuadratic-fm var (-b) 1 (b^2-4*a*c) (2*a)) F)
))
(list-conj (
(map (substInfinitesimalQuadratic var (-b) (-1) (b^2-4*a*c) (2*a)) L)@
(map (substInfinitesimalQuadratic-fm var (-b) (-1) (b^2-4*a*c) (2*a))
F)
))
))
)|
elimVar var L F (Neq p) = (
let (a,b,c) = get-coeffs var p in
(Or

(And (And (Atom (Eq a)) (Atom (Neq b)))
(list-conj (
(map (substInfinitesimalLinear var (-c) b) L)
@ (map (substInfinitesimalLinear-fm var (-c) b) F)
)))

(And (Atom (Neq a)) (And (Atom (Leq (-b^2)+4*a*c)))
(Or (list-conj (
(map (substInfinitesimalQuadratic var (-b) 1 (b^2-4*a*c) (2*a)) L)@
(map (substInfinitesimalQuadratic-fm var (-b) 1 (b^2-4*a*c) (2*a)) F)
))
(list-conj (
(map (substInfinitesimalQuadratic var (-b) (-1) (b^2-4*a*c) (2*a)) L)@
(map (substInfinitesimalQuadratic-fm var (-b) (-1) (b^2-4*a*c) (2*a))
F)
))
))
)

```

```

F)
  ))
  ))
  )))
|
elimVar var L F (Leq p) = (
let (a,b,c) = get-coeffs var p in

(Or

  (And (And (Atom (Eq a)) (Atom (Neg b)))
    (list-conj (
      (map (λa. Atom (linear-substitution var (-c) b a)) L)@
      (map (linear-substitution-fm var (-c) b) F)
    )))

  (And (Atom (Neg a)) (And (Atom (Leq (-b^2)+4*a*c)))
    (Or (list-conj (
      (map (quadratic-sub var (-b) 1 (b^2-4*a*c) (2*a)) L)@
      (map (quadratic-sub-fm var (-b) 1 (b^2-4*a*c) (2*a)) F)
    ))
      (list-conj (
        (map (quadratic-sub var (-b) (-1) (b^2-4*a*c) (2*a)) L)@
        (map (quadratic-sub-fm var (-b) (-1) (b^2-4*a*c) (2*a)) F)
      ))
    ))
  ))
)

```

```

fun ge-eq-one :: nat ⇒ atom list ⇒ atom fm list ⇒ atom fm where
ge-eq-one var L F =
(case find-eq var L of
  (p#A,L') ⇒ Or (And (Neg (split-p var p))
    ((elimVar var L F) (Eq p))
  )
  (And (split-p var p)
    (list-conj (map Atom ((map Eq A) @ L') @ F))
  )
  | ([],L') ⇒ list-conj ((map Atom L) @ F)
)

```

```

fun check-nonzero-const :: real mpoly ⇒ bool where
check-nonzero-const p = (case get-if-const p of Some x ⇒ x ≠ 0 | None ⇒ False)

```

```

fun find-lucky-eq :: nat ⇒ atom list ⇒ real mpoly option where

```

```

    find-lucky-eq v [] = None |
    find-lucky-eq v (Eq p#L) =
    (let (a,b,c) = get-coeffs v p in
    (if (MPoly-Type.degree p v = 1 ∨ MPoly-Type.degree p v = 2) ∧ (check-nonzero-const
    a ∨ check-nonzero-const b ∨ check-nonzero-const c) then Some p else
    find-lucky-eq v L
    )) |
    find-lucky-eq v (-#L) = find-lucky-eq v L

```

```

fun luckyFind :: nat ⇒ atom list ⇒ atom fm list ⇒ atom fm option where
    luckyFind v L F = (case find-lucky-eq v L of Some p ⇒ Some ((elimVar v L F)
    (Eq p)) | None ⇒ None)

```

```

fun luckyFind' :: nat ⇒ atom list ⇒ atom fm list ⇒ atom fm where
    luckyFind' v L F = (case find-lucky-eq v L of Some p ⇒ (elimVar v L F) (Eq p)
    | None ⇒ And (list-conj (map Atom L)) (list-conj F))

```

```

fun find-luckiest-eq :: nat ⇒ atom list ⇒ real mpoly option where
    find-luckiest-eq v [] = None |
    find-luckiest-eq v (Eq p#L) =
    (if (MPoly-Type.degree p v = 1 ∨ MPoly-Type.degree p v = 2) then
    (let (a,b,c) = get-coeffs v p in
    (case get-if-const a of None ⇒ find-luckiest-eq v L
    | Some a ⇒ (case get-if-const b of None ⇒ find-luckiest-eq v L
    | Some b ⇒ (case get-if-const c of None ⇒ find-luckiest-eq v L
    | Some c ⇒ if a≠0∨b≠0∨c≠0 then Some p else find-luckiest-eq v L))))
    else
    find-luckiest-eq v L
    ) |
    find-luckiest-eq v (-#L) = find-luckiest-eq v L

```

```

fun luckiestFind :: nat ⇒ atom list ⇒ atom fm list ⇒ atom fm where
    luckiestFind v L F = (case find-luckiest-eq v L of Some p ⇒ (elimVar v L F) (Eq
    p) | None ⇒ And (list-conj (map Atom L)) (list-conj F))

```

```

primrec ge-eq-repeat-helper :: nat ⇒ real mpoly list ⇒ atom list ⇒ atom fm list
⇒ atom fm where
    ge-eq-repeat-helper var [] L F = list-conj ((map Atom L) @ F) |
    ge-eq-repeat-helper var (p#A) L F =
    Or (And (Neg (split-p var p))
    ((elimVar var ((map Eq (p#A)) @ L) F) (Eq p))
    )
    (And (split-p var p)
    (ge-eq-repeat-helper var A L F))

```

```

)

fun qe-eq-repeat :: nat ⇒ atom list ⇒ atom fm list ⇒ atom fm where
  qe-eq-repeat var L F =
    (case luckyFind var L F of Some(F) ⇒ F | None ⇒
      (let (A,L') = find-eq var L in
        qe-eq-repeat-helper var A L' F
      )
    )
)

```

```

fun all-degree-2 :: nat ⇒ atom list ⇒ bool where
  all-degree-2 var [] = True |
  all-degree-2 var (Eq p#as) = ((MPoly-Type.degree p var ≤ 2) ∧ (all-degree-2 var as)) |
  all-degree-2 var (Less p#as) = ((MPoly-Type.degree p var ≤ 2) ∧ (all-degree-2 var as)) |
  all-degree-2 var (Leq p#as) = ((MPoly-Type.degree p var ≤ 2) ∧ (all-degree-2 var as)) |
  all-degree-2 var (Neg p#as) = ((MPoly-Type.degree p var ≤ 2) ∧ (all-degree-2 var as))
)

```

```

fun gen-qe :: nat ⇒ atom list ⇒ atom fm list ⇒ atom fm where
  gen-qe var L F = (case F of
    [] ⇒ (case luckyFind var L [] of Some F ⇒ F | None ⇒ (
      (if all-degree-2 var L
        then list-disj (list-conj (map (substNegInfinity var) L) # (map (elimVar var L []) L))
        else (qe-eq-repeat var L [])))
    | - ⇒ qe-eq-repeat var L F
  )
)

```

## 7.4 DNF

```

fun dnf :: atom fm ⇒ (atom list * atom fm list) list where
  dnf TrueF = [([],[])] |
  dnf FalseF = [] |
  dnf (Atom φ) = [(φ,[])] |
  dnf (And φ1 φ2) = [(A@B,A'@B').(A,A')←dnf φ1,(B,B')←dnf φ2] |
  dnf (Or φ1 φ2) = dnf φ1 @ dnf φ2 |
  dnf (ExQ φ) = [([],[ExQ φ])] |
  dnf (Neg φ) = [([],[Neg φ])] |
  dnf (AllQ φ) = [([],[AllQ φ])] |
  dnf (AllN i φ) = [([],[AllN i φ])] |
  dnf (ExN i φ) = [([],[ExN i φ])]
)

```

*dnf F* returns the "disjunctive normal form" of *F*, but since *F* can contain quantifiers, we return (L,R,n) terms in a list. each term in the list represents a conjunction over the outside disjunctive list

L is all the atoms we are able to reach, we are allowed to go underneath exists binders

R is the remaining formulas (negation exists cannot be simplified) which are also under the same number of exist binders.

n is the total number of binders each conjunct has

**fun** *dnf-modified* :: *atom fm*  $\Rightarrow$  (*atom list* \* *atom fm list* \* *nat*) *list* **where**

```

dnf-modified TrueF = [([],[],0)] |
dnf-modified FalseF = [] |
dnf-modified (Atom  $\varphi$ ) = [[( $\varphi$ ),[],0)] |
dnf-modified (And  $\varphi_1$   $\varphi_2$ ) = [
let A = map (liftAtom d1 d2) A in
let B = map (liftAtom 0 d1) B in
let A' = map (liftFm d1 d2) A' in
let B' = map (liftFm 0 d1) B' in
(A @ B, A' @ B', d1+d2).
(A, A', d1)  $\leftarrow$  dnf-modified  $\varphi_1$ , (B, B', d2)  $\leftarrow$  dnf-modified  $\varphi_2$ ] |
dnf-modified (Or  $\varphi_1$   $\varphi_2$ ) = dnf-modified  $\varphi_1$  @ dnf-modified  $\varphi_2$  |
dnf-modified (ExQ  $\varphi$ ) = [(A, A', d+1). (A, A', d)  $\leftarrow$  dnf-modified  $\varphi$ ] |
dnf-modified (Neg  $\varphi$ ) = [([], [Neg  $\varphi$ ], 0)] |
dnf-modified (AllQ  $\varphi$ ) = [([], [AllQ  $\varphi$ ], 0)] |
dnf-modified (AllN i  $\varphi$ ) = [([], [AllN i  $\varphi$ ], 0)] |
dnf-modified (ExN i  $\varphi$ ) = [(A, A', d+i). (A, A', d)  $\leftarrow$  dnf-modified  $\varphi$ ]

```

**fun** *QE-dnf* :: (*atom fm*  $\Rightarrow$  *atom fm*)  $\Rightarrow$  (*nat*  $\Rightarrow$  *nat*  $\Rightarrow$  *atom list*  $\Rightarrow$  *atom fm list*  $\Rightarrow$  *atom fm*)  $\Rightarrow$  *atom fm*  $\Rightarrow$  *atom fm* **where**

```

QE-dnf opt step (And  $\varphi_1$   $\varphi_2$ ) = and (QE-dnf opt step  $\varphi_1$ ) (QE-dnf opt step  $\varphi_2$ ) |
QE-dnf opt step (Or  $\varphi_1$   $\varphi_2$ ) = or (QE-dnf opt step  $\varphi_1$ ) (QE-dnf opt step  $\varphi_2$ ) |
QE-dnf opt step (Neg  $\varphi$ ) = neg(QE-dnf opt step  $\varphi$ ) |
QE-dnf opt step (ExQ  $\varphi$ ) = list-disj [ExN (n+1) (step 1 n al fl). (al, fl, n)  $\leftarrow$  (dnf-modified(opt(QE-dnf
opt step  $\varphi$ )))] |
QE-dnf opt step (TrueF) = TrueF |
QE-dnf opt step (FalseF) = FalseF |
QE-dnf opt step (Atom a) = simp-atom a |
QE-dnf opt step (AllQ  $\varphi$ ) = Neg(list-disj [ExN (n+1) (step 1 n al fl). (al, fl, n)  $\leftarrow$  (dnf-modified(opt(neg(QE-dnf
opt step  $\varphi$ )))] |
QE-dnf opt step (ExN 0  $\varphi$ ) = QE-dnf opt step  $\varphi$  |
QE-dnf opt step (AllN 0  $\varphi$ ) = QE-dnf opt step  $\varphi$  |
QE-dnf opt step (AllN (Suc i)  $\varphi$ ) = Neg(list-disj [ExN (n+i+1) (step (Suc i)
(n+i) al fl). (al, fl, n)  $\leftarrow$  (dnf-modified(opt(neg(QE-dnf opt step  $\varphi$ )))] |
QE-dnf opt step (ExN (Suc i)  $\varphi$ ) = list-disj [ExN (n+i+1) (step (Suc i) (n+i)
al fl). (al, fl, n)  $\leftarrow$  (dnf-modified(opt(QE-dnf opt step  $\varphi$ )))]

```

**fun** *QE-dnf'* :: (*atom fm*  $\Rightarrow$  *atom fm*)  $\Rightarrow$  (*nat*  $\Rightarrow$  (*atom list* \* *atom fm list* \* *nat*) *list*  $\Rightarrow$  *atom fm*)  $\Rightarrow$  *atom fm*  $\Rightarrow$  *atom fm* **where**

```

QE-dnf' opt step (And  $\varphi_1$   $\varphi_2$ ) = and (QE-dnf' opt step  $\varphi_1$ ) (QE-dnf' opt step
 $\varphi_2$ ) |

```

```

QE-dnf' opt step (Or  $\varphi_1 \varphi_2$ ) = or (QE-dnf' opt step  $\varphi_1$ ) (QE-dnf' opt step  $\varphi_2$ )
|
QE-dnf' opt step (Neg  $\varphi$ ) = neg(QE-dnf' opt step  $\varphi$ ) |
QE-dnf' opt step (ExQ  $\varphi$ ) = step 1 (dnf-modified(opt(QE-dnf' opt step  $\varphi$ )))|
QE-dnf' opt step (TrueF) = TrueF|
QE-dnf' opt step (FalseF) = FalseF|
QE-dnf' opt step (Atom  $a$ ) = simp-atom  $a$ |
QE-dnf' opt step (AllQ  $\varphi$ ) = Neg(step 1 (dnf-modified(opt(neg(QE-dnf' opt step
 $\varphi$ ))))))|
QE-dnf' opt step (ExN 0  $\varphi$ ) = QE-dnf' opt step  $\varphi$ |
QE-dnf' opt step (AllN 0  $\varphi$ ) = QE-dnf' opt step  $\varphi$ |
QE-dnf' opt step (AllN (Suc  $i$ )  $\varphi$ ) = Neg(step (Suc  $i$ ) (dnf-modified(opt(neg(QE-dnf'
opt step  $\varphi$ ))))))|
QE-dnf' opt step (ExN (Suc  $i$ )  $\varphi$ ) = step (Suc  $i$ ) (dnf-modified(opt(QE-dnf' opt
step  $\varphi$ )))

```

## 7.5 Repeat QE multiple times

```

fun countQuantifiers :: atom fm  $\Rightarrow$  nat where
  countQuantifiers (Atom -) = 0|
  countQuantifiers (TrueF) = 0|
  countQuantifiers (FalseF) = 0|
  countQuantifiers (And  $a b$ ) = countQuantifiers  $a$  + countQuantifiers  $b$ |
  countQuantifiers (Or  $a b$ ) = countQuantifiers  $a$  + countQuantifiers  $b$ |
  countQuantifiers (Neg  $a$ ) = countQuantifiers  $a$ |
  countQuantifiers (ExQ  $a$ ) = countQuantifiers  $a$  + 1|
  countQuantifiers (AllQ  $a$ ) = countQuantifiers  $a$  + 1|
  countQuantifiers (ExN  $n a$ ) = countQuantifiers  $a$  +  $n$ |
  countQuantifiers (AllN  $n a$ ) = countQuantifiers  $a$  +  $n$ 

fun repeatAmountOfQuantifiers-helper :: (atom fm  $\Rightarrow$  atom fm)  $\Rightarrow$  nat  $\Rightarrow$  atom
fm  $\Rightarrow$  atom fm where
  repeatAmountOfQuantifiers-helper step 0  $F$  =  $F$ |
  repeatAmountOfQuantifiers-helper step (Suc  $i$ )  $F$  = repeatAmountOfQuantifiers-helper
step  $i$  (step  $F$ )

fun repeatAmountOfQuantifiers :: (atom fm  $\Rightarrow$  atom fm)  $\Rightarrow$  atom fm  $\Rightarrow$  atom fm
where
  repeatAmountOfQuantifiers step  $F$  = (
let  $F$  = step  $F$  in
let  $n$  = countQuantifiers  $F$  in
repeatAmountOfQuantifiers-helper step  $n$   $F$ 
)

end

```

## 7.6 Heuristic Algorithms

```

theory Heuristic
imports VSAlgos Reindex Optimizations

```

```

begin
fun IdentityHeuristic :: nat ⇒ atom list ⇒ atom fm list ⇒ nat where
  IdentityHeuristic n - - = n

fun step-augment :: (nat ⇒ atom list ⇒ atom fm list ⇒ atom fm) ⇒ (nat ⇒ atom
list ⇒ atom fm list ⇒ nat) ⇒ nat ⇒ nat ⇒ atom list ⇒ atom fm list ⇒ atom fm
where
  step-augment step heuristic 0 var L F = list-conj (map fm.Atom L @ F) |
  step-augment step heuristic (Suc 0) 0 L F = step 0 L F |
  step-augment step heuristic - 0 L F = list-conj (map fm.Atom L @ F) |
  step-augment step heuristic (Suc amount) (Suc i) L F =(
  let var = heuristic (Suc i) L F in
  let swappedL = map (swap-atom (i+1) var) L in
  let swappedF = map (swap-fm (i+1) var) F in
  list-disj[step-augment step heuristic amount i al fl. (al,fl)<-dnf ((push-forall
○ nnf ○ unpower 0 o groupQuantifiers o clearQuantifiers)(step (i+1) swappedL
swappedF))])

fun the-real-step-augment :: (nat ⇒ atom list ⇒ atom fm list ⇒ atom fm) ⇒ nat
⇒ (atom list * atom fm list * nat) list ⇒ atom fm where
  the-real-step-augment step 0 F = list-disj (map (λ(L,F,n). ExN n (list-conj (map
fm.Atom L @ F))) F) |
  the-real-step-augment step (Suc amount) F =(
  ExQ (the-real-step-augment step amount (dnf-modified ((push-forall ○ nnf ○ un-
power 0 o groupQuantifiers o clearQuantifiers)(list-disj(map (λ(L,F,n). ExN n
(step (n+amount) L F) F))))))

fun aquireData :: nat ⇒ atom list ⇒ (nat fset*nat fset*nat fset)where
  aquireData n L = fold (λA (l,e,g).
  case A of
  Eq p ⇒
  (
  funion l (fset-of-list(filter (λv. let (a,b,c) = get-coeffs v p in
((MPoly-Type.degree p v = 1 ∨ MPoly-Type.degree p v = 2) ∧ (check-nonzero-const
a ∨ check-nonzero-const b ∨ check-nonzero-const c))) [0..<(n+1)])),
  funion e (fset-of-list(filter (λv.(MPoly-Type.degree p v = 1 ∨ MPoly-Type.degree
p v = 2)) [0..<(n+1)]))
  .ffilter (λv. MPoly-Type.degree p v ≤ 2) g)
  | Leq p ⇒ (l,e,ffilter (λv. MPoly-Type.degree p v ≤ 2) g)
  | Neq p ⇒ (l,e,ffilter (λv. MPoly-Type.degree p v ≤ 2) g)
  | Less p ⇒ (l,e,ffilter (λv. MPoly-Type.degree p v ≤ 2) g)
  ) L (fempty,fempty,fset-of-list [0..<(n+1)])

datatype natpair = Pair nat*nat

instantiation natpair :: linorder

```

```

begin
definition [simp]: less-eq (A::natpair) B = (case A of Pair(a,b) => (case B of
Pair(c,d) => if a=c then b<=d else a<c))
definition [simp]: less (A::natpair) B = (case A of Pair(a,b) => (case B of Pair(c,d)
=> if a=c then b<d else a<c))
instance <proof>
end

```

```

fun getBest :: nat fset => atom list => nat option where
  getBest S L = (let X = fset-of-list(map (\x. Pair(count-list (map (\l. case l of
    Eq p => MPoly-Type.degree p x = 0
  | Less p => MPoly-Type.degree p x = 0
  | Neg p => MPoly-Type.degree p x = 0
  | Leq p => MPoly-Type.degree p x = 0
  ) L) False,x)) (sorted-list-of-fset S)) in
(case (sorted-list-of-fset X) of [] => None | Cons (Pair(x,v)) - => Some v))

```

```

fun heuristicPicker :: nat => atom list => atom fm list => (nat*(nat => atom list
=> atom fm list => atom fm)) option where
  heuristicPicker n L F = (case (let (l,e,g) = acquireData n L in
(case getBest l L of
  None => (case F of
    [] =>
      (case getBest g L of
        None => (case getBest e L of None => None | Some v => Some(v,ge-eq-repeat))
        | Some v => Some(v,gen-ge)
      )
    | - => (case getBest e L of None => None | Some v => Some(v,ge-eq-repeat))
  )
  | Some v => Some(v,luckyFind')
)) of None => None | Some(var,step) => (if var > n then None else Some(var,step)))

```

```

fun superPicker :: nat => nat => atom list => atom fm list => atom fm where
  superPicker 0 var L F = list-conj (map fm.Atom L @ F)|
  superPicker amount 0 L F = (case heuristicPicker 0 L F of Some(0,step) => step
0 L F | - => list-conj (map fm.Atom L @ F)) |
  superPicker (Suc amount) (Suc i) L F =(
  case heuristicPicker (Suc i) L F of
    Some(var,step) =>
      let swappedL = map (swap-atom (i+1) var) L in
      let swappedF = map (swap-fm (i+1) var) F in
      list-disj[superPicker amount i al fl. (al,fl)<-dnf ((push-forall o nnf o unpower
0 o groupQuantifiers o clearQuantifiers)(step (i+1) swappedL swappedF))]
    | None => list-conj (map fm.Atom L @ F))

```

```

datatype quadnat = Quad nat × nat × nat × nat

```

```

instantiation quadnat :: linorder begin
definition [simp]: A < B =
  (case A of Quad(a1,b1,c1,d1) => (case B of Quad(a2,b2,c2,d2) =>
    (if a1=a2 then (
      if b1=b2 then (
        if c1=c2 then d1 < d2 else c1 < c2
      ) else b1 < b2
    ) else a1 < a2)))
definition [simp]: A ≤ B =
  (case A of Quad(a1,b1,c1,d1) => (case B of Quad(a2,b2,c2,d2) =>
    (if a1=a2 then (
      if b1=b2 then (
        if c1=c2 then d1 ≤ d2 else c1 < c2
      ) else b1 < b2
    ) else a1 < a2)))
instance ⟨proof⟩
end

fun brownsHeuristic :: nat => atom list => atom fm list => nat where
  brownsHeuristic n L = (case sorted-list-of-fset (fset-of-list (map (λx.
    case (foldl (λ(maxdeg,totaldeg,appearancecount) l.
      let p = case l of Eq p => p | Less p => p | Leq p => p | Neq p => p in
      let deg = MPoly-Type.degree p x in
      (max maxdeg deg,totaldeg+deg,appearancecount+(if deg>0 then 1 else 0))) (0,0,0)
    L) of (a,b,c) => Quad(a,b,c,x)
  ) [0..<n])) of [] => n | Cons (Quad(-,-, -,x)) - => if x>n then n else x)

end
theory PrettyPrinting
imports
  ExecutablePolyProps
  PolyAtoms
  Polynomials.Show-Polynomials
  Polynomials.Power-Products
begin

global-interpretation drlex-pm: linorder drlex-pm drlex-pm-strict
defines Min-drlex-pm = linorder.Min drlex-pm
  and Max-drlex-pm = linorder.Max drlex-pm
  and sorted-drlex-pm = linorder.sorted drlex-pm
  and sorted-list-of-set-drlex-pm = linorder.sorted-list-of-set drlex-pm
  and sort-key-drlex-pm = linorder.sort-key drlex-pm
  and insort-key-drlex-pm = linorder.insort-key drlex-pm
  and part-drlex-pm = drlex-pm.part
  ⟨proof⟩

definition monomials-list mp = drlex-pm.sorted-list-of-set (monomials mp)

```

**definition** *shows-monomial-gen*::( $(nat \times nat) \Rightarrow shows$ )  $\Rightarrow$  ( $'a \Rightarrow shows$ )  $\Rightarrow shows$   
 $\Rightarrow (nat \Rightarrow_0 nat) \Rightarrow 'a \text{ option} \Rightarrow shows$  **where**  
*shows-monomial-gen shows-factor shows-coeff sep mon cff* =  
*shows-sep* ( $\lambda s$ . *case s of*  
  *Inl cff*  $\Rightarrow shows-coeff cff$   
  | *Inr factor*  $\Rightarrow shows-factor factor$   
) *sep* ((*case cff of None*  $\Rightarrow []$  | *Some cff*  $\Rightarrow [Inl cff]$ ) @ *map Inr (Poly-Mapping.items mon)*)

**definition** *shows-factor-compact factor* =  
(*case factor of* ( $k, v$ )  $\Rightarrow shows-string "x" +@+ shows k +@+$   
(*if*  $v = 1$  *then* *shows-string ""* *else* *shows-string "^"* +@+ *shows v*))

**definition** *shows-factor-Var factor* =  
(*case factor of* ( $k, v$ )  $\Rightarrow shows-string "(Var " +@+ shows k +@+ shows-string "$   
 $)" +@+$   
(*if*  $v = 1$  *then* *shows-string ""* *else* *shows-string "^"* +@+ *shows v*))

**definition** *shows-monomial-compact*::( $'a \Rightarrow shows$ )  $\Rightarrow (nat \Rightarrow_0 nat) \Rightarrow 'a \text{ option}$   
 $\Rightarrow shows$  **where**  
*shows-monomial-compact shows-coeff m* =  
*shows-monomial-gen shows-factor-compact shows-coeff (shows-string " ") m*

**definition** *shows-monomial-Var*::( $'a \Rightarrow shows$ )  $\Rightarrow (nat \Rightarrow_0 nat) \Rightarrow 'a \text{ option} \Rightarrow$   
 $shows$  **where**  
*shows-monomial-Var shows-coeff m* =  
*shows-monomial-gen shows-factor-Var shows-coeff (shows-string "\*" ) m*

**fun** *shows-mpoly* ::  $bool \Rightarrow ('a \Rightarrow shows) \Rightarrow 'a::\{zero,one\} \text{ mpoly} \Rightarrow shows$  **where**  
*shows-mpoly input shows-coeff p* = *shows-sep* ( $\lambda mon$ .  
(*if input then shows-monomial-Var* ( $\lambda x$ . *shows-paren (shows-string "Const "*  
+@+ *shows-paren (shows-coeff x)*)) *else shows-monomial-compact shows-coeff*)  
*mon*  
(*let cff = MPoly-Type.coeff p mon in if cff = 1 then None else Some cff*)  
)  
(*shows-string " + "*)  
(*monomials-list p*)

**definition** *rat-of-real* ( $x::real$ ) =  
(*if* ( $\exists r::rat. x = of-rat r$ ) *then* (*THE*  $r. x = of-rat r$ ) *else* *9999999999.9999999999*)

**lemma** *rat-of-real*: *rat-of-real*  $x = r$  **if**  $x = of-rat r$   
*<proof>*

**lemma** *rat-of-real-code*[*code*]: *rat-of-real* (*Ratreal*  $r$ ) =  $r$   
*<proof>*

**definition** *shows-real*  $x = \text{shows (rat-of-real } x)$

**experiment begin**

**abbreviation** *foo*  $\equiv ((\text{Var } 0::\text{real mpoly}) + \text{Const } (0.5) * \text{Var } 1 + \text{Var } 2)^3$

**value** [code] *shows-mpoly True shows-real foo* ""

**lemma** *foo-eq*:  $\text{foo} = (\text{Var } 0)^3 + (\text{Const } (3/2)) * (\text{Var } 0)^2 * (\text{Var } 1) + (\text{Const } (3)) * (\text{Var } 0)^2 * (\text{Var } 2) + (\text{Const } (3/4)) * (\text{Var } 0) * (\text{Var } 1)^2 + (\text{Const } (3)) * (\text{Var } 0) * (\text{Var } 1) * (\text{Var } 2) + (\text{Const } (3)) * (\text{Var } 0) * (\text{Var } 2)^2 + (\text{Const } (1/8)) * (\text{Var } 1)^3 + (\text{Const } (3/4)) * (\text{Var } 1)^2 * (\text{Var } 2) + (\text{Const } (3/2)) * (\text{Var } 1) * (\text{Var } 2)^2 + (\text{Var } 2)^3$

*<proof>*

**value** [code] *shows-mpoly False shows-real foo* ""

**value** [code] *shows-mpoly False (shows-paren o shows-mpoly False shows-real) (extract-var foo 0)* ""

**value** [code] *shows-list-gen (shows-mpoly False shows-real)*

"" "" "" "" "" "" "" "" "" ""

*(Polynomial.coeffs (mpoly-to-nested-poly foo 0))* ""

**end**

**fun** *shows-atom* :: *bool*  $\Rightarrow$  *atom*  $\Rightarrow$  *shows where*

*shows-atom c (Eq p) = (shows-string "(" + @+ shows-mpoly c shows-real p + @+ shows-string "=0")*

*shows-atom c (Less p) = (shows-string "(" + @+ shows-mpoly c shows-real p + @+ shows-string "<0")*

*shows-atom c (Leq p) = (shows-string "(" + @+ shows-mpoly c shows-real p + @+ shows-string "<=0")*

*shows-atom c (Neg p) = (shows-string "(" + @+ shows-mpoly c shows-real p + @+ shows-string "~=0")*

**fun** *depth'* :: 'a *fm*  $\Rightarrow$  *natwhere*

*depth' TrueF = 1*

*depth' FalseF = 1*

*depth' (Atom -) = 1*

*depth' (And  $\varphi$   $\psi$ ) = max (depth'  $\varphi$ ) (depth'  $\psi$ ) + 1*

*depth' (Or  $\varphi$   $\psi$ ) = max (depth'  $\varphi$ ) (depth'  $\psi$ ) + 1*

*depth' (Neg  $\varphi$ ) = depth'  $\varphi$  + 1*

*depth' (ExQ  $\varphi$ ) = depth'  $\varphi$  + 1*

*depth' (AllQ  $\varphi$ ) = depth'  $\varphi$  + 1*

*depth' (AllN i  $\varphi$ ) = depth'  $\varphi$  + i \* 2 + 1*

*depth' (ExN i  $\varphi$ ) = depth'  $\varphi$  + i \* 2 + 1*

**function** *shows-fm* :: *bool*  $\Rightarrow$  *atom fm*  $\Rightarrow$  *shows where*

*shows-fm c (Atom a) = shows-atom c a*

*shows-fm c (TrueF) = shows-string "(T)"*

*shows-fm c (FalseF) = shows-string "(F)"*

*shows-fm c (And  $\varphi$   $\psi$ ) = (shows-string "(" + @+ shows-fm c  $\varphi$  + @+ shows-string " and " + @+ shows-fm c  $\psi$  + @+ shows-string ("))*

```

  shows-fm c (Or  $\varphi$   $\psi$ ) = (shows-string "(" +@+ shows-fm c  $\varphi$  +@+ shows-string
  " or " +@+ shows-fm c  $\psi$  +@+ shows-string ")")|
  shows-fm c (Neg  $\varphi$ ) = (shows-string "(neg " +@+ shows-fm c  $\varphi$  +@+ shows-string
  ")")|
  shows-fm c (ExQ  $\varphi$ ) = (shows-string "(exists" +@+ shows-fm c  $\varphi$  +@+ shows-string
  ")")|
  shows-fm c (AllQ  $\varphi$ ) = (shows-string "(forall" +@+ shows-fm c  $\varphi$  +@+ shows-string
  ")")|
  shows-fm c (ExN 0  $\varphi$ ) = shows-fm c  $\varphi$ |
  shows-fm c (ExN (Suc n)  $\varphi$ ) = shows-fm c (ExQ(ExN n  $\varphi$ ))|
  shows-fm c (AllN 0  $\varphi$ ) = shows-fm c  $\varphi$ |
  shows-fm c (AllN (Suc n)  $\varphi$ ) = shows-fm c (AllQ(AllN n  $\varphi$ ))
  <proof>
termination
  <proof>

```

```

value shows-fm False (ExQ (Or (AllQ(And (Neg TrueF) (Neg FalseF))) (Atom(Eq(Const
4)))))) []
value shows-fm True (ExQ (Or (AllQ(And (Neg TrueF) (Neg FalseF))) (Atom(Eq(Const
4)))))) []

```

**end**

## 7.7 Top-Level Algorithms

**theory** Exports

**imports** Heuristic VSAlgos Optimizations

*HOL.String HOL-Library.Code-Target-Int HOL-Library.Code-Target-Nat PrettyPrinting Show.Show-Real*

**begin**

**definition** opt = (push-forall  $\circ$  nnf  $\circ$  unpower 0  $\circ$  clearQuantifiers)

**definition** opt-group = (push-forall  $\circ$  nnf  $\circ$  unpower 0  $\circ$  groupQuantifiers  $\circ$  clear-Quantifiers)

**definition** VSLuckiest = opt  $\circ$  (QE-dnf opt ( $\lambda$ amount. luckiestFind))  $\circ$  opt

**definition** VSLuckiestBlocks = opt-group  $\circ$  (QE-dnf' opt-group (the-real-step-augment luckiestFind))  $\circ$  opt-group

**definition** VSEquality = opt  $\circ$  (QE-dnf opt( $\lambda$ x. qe-eq-repeat))  $\circ$  VSLuckiest  $\circ$  opt

**definition** VSEqualityBlocks = opt-group  $\circ$  (QE-dnf' opt-group (the-real-step-augment qe-eq-repeat))  $\circ$  VSLuckiestBlocks  $\circ$  opt-group

**definition** VSGeneralBlocks = opt-group  $\circ$  (QE-dnf' opt-group (the-real-step-augment gen-qe))  $\circ$  VSLuckiestBlocks  $\circ$  opt-group

**definition** VSLuckyBlocks = opt-group  $\circ$  (QE-dnf' opt-group (the-real-step-augment luckyFind'))  $\circ$  VSLuckiestBlocks  $\circ$  opt-group

**definition** VSLEGBlocks = VSGeneralBlocks  $\circ$  VSEqualityBlocks  $\circ$  VSLuckyBlocks

**definition**  $VSEqualityBlocksLimited = opt\text{-}group \ o \ (QE\text{-}dnf \ opt\text{-}group \ (step\text{-}augment \ gen\text{-}eq\text{-}repeat \ IdentityHeuristic)) \ o \ VSLuckiestBlocks \ o \ opt\text{-}group$

**definition**  $VSEquality\text{-}3\text{-}times = VSEquality \ o \ VSEquality \ o \ VSEquality$

**definition**  $VSGeneral = opt \ o \ (QE\text{-}dnf \ opt \ (\lambda x. \ gen\text{-}qe)) \ o \ VSLuckiest \ o \ opt$

**definition**  $VSGeneralBlocksLimited = opt\text{-}group \ o \ (QE\text{-}dnf \ opt\text{-}group \ (step\text{-}augment \ gen\text{-}qe \ IdentityHeuristic)) \ o \ VSLuckiestBlocks \ o \ opt\text{-}group$

**definition**  $VSBrowns = opt\text{-}group \ o \ (QE\text{-}dnf \ opt\text{-}group \ (step\text{-}augment \ gen\text{-}qe \ brown\text{-}sHeuristic)) \ o \ VSLuckiestBlocks \ o \ opt\text{-}group$

**definition**  $VSGeneral\text{-}3\text{-}times = VSGeneral \ o \ VSGeneral \ o \ VSGeneral$

**definition**  $VSLucky = opt \ o \ (QE\text{-}dnf \ opt \ (\lambda amount. \ luckyFind')) \ o \ VSLuckiest \ o \ opt$

**definition**  $VSLuckyBlocksLimited = opt\text{-}group \ o \ (QE\text{-}dnf \ opt\text{-}group \ (step\text{-}augment \ luckyFind' \ IdentityHeuristic)) \ o \ VSLuckiestBlocks \ o \ opt\text{-}group$

**definition**  $VSLEG = VSGeneral \ o \ VSEquality \ o \ VSLucky$

**definition**  $VSHuristic = opt\text{-}group \ o \ (QE\text{-}dnf \ opt\text{-}group \ (superPicker)) \ o \ VSLuckiestBlocks \ o \ opt\text{-}group$

**definition**  $VSLuckiestRepeat = repeatAmountOfQuantifiers \ VSLuckiest$

**definition**  $add :: real \ mpoly \Rightarrow real \ mpoly \Rightarrow real \ mpoly \ \mathbf{where}$   
 $add \ p \ q = p + q$

**definition**  $minus :: real \ mpoly \Rightarrow real \ mpoly \Rightarrow real \ mpoly \ \mathbf{where}$   
 $minus \ p \ q = p - q$

**definition**  $mult :: real \ mpoly \Rightarrow real \ mpoly \Rightarrow real \ mpoly \ \mathbf{where}$   
 $mult \ p \ q = p * q$

**definition**  $pow :: real \ mpoly \Rightarrow integer \Rightarrow real \ mpoly \ \mathbf{where}$   
 $pow \ p \ n = p \wedge (nat\text{-}of\text{-}integer \ n)$

**definition**  $C :: real \Rightarrow real \ mpoly \ \mathbf{where}$   
 $C \ r = Const \ r$

**definition**  $V :: integer \Rightarrow real \ mpoly \ \mathbf{where}$   
 $V \ n = Var \ (nat\text{-}of\text{-}integer \ n)$

**definition**  $real\text{-}of\text{-}int :: integer \Rightarrow real$   
**where**  $real\text{-}of\text{-}int \ n = real \ (nat\text{-}of\text{-}integer \ n)$

**definition**  $real\text{-}mult :: real \Rightarrow real \Rightarrow real$   
**where**  $real\text{-}mult \ n \ m = n * m$

**definition**  $real\text{-}div :: real \Rightarrow real \Rightarrow real$   
**where**  $real\text{-}div \ n \ m = n / m$

**definition**  $real\text{-}plus :: real \Rightarrow real \Rightarrow real$   
**where**  $real\text{-}plus \ n \ m = n + m$

**definition** *real-minus* :: *real*  $\Rightarrow$  *real*  $\Rightarrow$  *real*  
**where** *real-minus* *n m* = *n - m*

**fun** *is-quantifier-free* :: *atom fm*  $\Rightarrow$  *bool* **where**  
*is-quantifier-free* (*ExQ x*) = *False* |  
*is-quantifier-free* (*AllQ x*) = *False* |  
*is-quantifier-free* (*And a b*) = (*is-quantifier-free a*  $\wedge$  *is-quantifier-free b*) |  
*is-quantifier-free* (*Or a b*) = (*is-quantifier-free a*  $\wedge$  *is-quantifier-free b*) |  
*is-quantifier-free* (*Neg a*) = *is-quantifier-free a* |  
*is-quantifier-free a* = *True*

**fun** *is-solved* :: *atom fm*  $\Rightarrow$  *bool* **where**  
*is-solved TrueF* = *True* |  
*is-solved FalseF* = *True* |  
*is-solved A* = *False*

**definition** *print-mpoly* :: (*real*  $\Rightarrow$  *String.literal*)  $\Rightarrow$  *real mpoly*  $\Rightarrow$  *String.literal* **where**  
*print-mpoly f p* = *String.implode* ((*shows-mpoly True* ( $\lambda x.\lambda y.$  (*String.explode o*  
*f*) *x @ y*)) *p* ""')

**definition** *Unpower* = *unpower 0*

**export-code**

*print-mpoly*  
*VSGeneral VSEquality VSLucky VSLEG VSLuckiest*  
*VSGeneralBlocksLimited VSEqualityBlocksLimited VSLuckyBlocksLimited*  
*VSGeneralBlocks VSEqualityBlocks VSLuckyBlocks VSLEGBlocks VSLuckiest-*  
*Blocks*  
*QE-dnf*  
*gen-qe qe-eq-repeat*  
*simpfm push-forall nnf Unpower*  
*is-quantifier-free is-solved*  
*add mult C V pow minus*  
*Eq Or is-quantifier-free*

*real-of-int real-mult real-div real-plus real-minus*

*VSGeneral-3-times VSEquality-3-times VSHuristic VSLuckiestRepeat VSBrowns*  
**in SML module-name** *VS*

**end**

## 8 Equality VS Proofs

### 8.1 Linear Case

**theory** *LinearCase*  
**imports** *VSAlgos*  
**begin**

**theorem** *var-not-in-linear* :  
**assumes**  $var \notin vars\ b$   
**assumes**  $var \notin vars\ c$   
**shows**  $freeIn\ var\ (Atom\ (linear-substitution\ var\ b\ c\ A))$   
 $\langle proof \rangle$

**lemma** *linear-eq* :  
**assumes**  $lLength : length\ L > var$   
**assumes**  $nonzero : C \neq 0$   
**assumes**  $var \notin vars\ b$   
**assumes**  $var \notin vars\ c$   
**assumes**  $hb : insertion\ (nth-default\ 0\ (list-update\ L\ var\ (B/C)))\ b = (B::real)$   
**assumes**  $hc : insertion\ (nth-default\ 0\ (list-update\ L\ var\ (B/C)))\ c = (C::real)$   
**shows**  $aEval\ (Eq(p))\ (list-update\ L\ var\ (B/C)) = (aEval\ (linear-substitution\ var\ b\ c\ (Eq(p)))\ (list-update\ L\ var\ v))$   
 $\langle proof \rangle$

**lemma** *linear-less* :  
**assumes**  $lLength : length\ L > var$   
**assumes**  $nonzero : C \neq 0$   
**assumes**  $var \notin vars\ b$   
**assumes**  $var \notin vars\ c$   
**assumes**  $insertion\ (nth-default\ 0\ (list-update\ L\ var\ (B/C)))\ b = (B::real)$   
**assumes**  $insertion\ (nth-default\ 0\ (list-update\ L\ var\ (B/C)))\ c = (C::real)$   
**shows**  $aEval\ (Less(p))\ (list-update\ L\ var\ (B/C)) = (aEval\ (linear-substitution\ var\ b\ c\ (Less(p)))\ (list-update\ L\ var\ v))$   
 $\langle proof \rangle$

**lemma** *linear-leq* :  
**assumes**  $lLength : length\ L > var$   
**assumes**  $nonzero : C \neq 0$   
**assumes**  $var \notin vars\ b$   
**assumes**  $var \notin vars\ c$

**assumes** *insertion* (*nth-default* 0 (*list-update* L var (*B/C*))) *b* = (*B::real*)  
**assumes** *insertion* (*nth-default* 0 (*list-update* L var (*B/C*))) *c* = (*C::real*)  
**shows** *aEval* (*Leq*(*p*)) (*list-update* L var (*B/C*)) = (*aEval* (*linear-substitution*  
var *b c* (*Leq*(*p*))) (*list-update* L var *v*))  
<*proof*>

**lemma** *linear-neq* :

**assumes** *lLength* : *length* L > var  
**assumes** *nonzero* : *C* ≠ 0  
**assumes** var ∉ vars *b*  
**assumes** var ∉ vars *c*  
**assumes** *insertion* (*nth-default* 0 (*list-update* L var (*B/C*))) *b* = (*B::real*)  
**assumes** *insertion* (*nth-default* 0 (*list-update* L var (*B/C*))) *c* = (*C::real*)  
**shows** *aEval* (*Neq*(*p*)) (*list-update* L var (*B/C*)) = (*aEval* (*linear-substitution*  
var *b c* (*Neq*(*p*))) (*list-update* L var *v*))  
<*proof*>

**theorem** *linear* :

**assumes** *lLength* : *length* L > var  
**assumes** *C* ≠ 0  
**assumes** var ∉ vars *b*  
**assumes** var ∉ vars *c*  
**assumes** *insertion* (*nth-default* 0 (*list-update* L var (*B/C*))) *b* = (*B::real*)  
**assumes** *insertion* (*nth-default* 0 (*list-update* L var (*B/C*))) *c* = (*C::real*)  
**shows** *aEval* *A* (*list-update* L var (*B/C*)) = (*aEval* (*linear-substitution* var *b c*  
*A*) (*list-update* L var *v*))  
<*proof*>

**lemma** *var-not-in-linear-fm-helper* :

**assumes** var ∉ vars *b*  
**assumes** var ∉ vars *c*  
**shows** *freeIn* (var+z) (*linear-substitution-fm-helper* var *b c F z*)  
<*proof*>

**theorem** *var-not-in-linear-fm* :

**assumes** var ∉ vars *b*  
**assumes** var ∉ vars *c*  
**shows** *freeIn* var (*linear-substitution-fm* var *b c F*)  
<*proof*>

```

lemma linear-fm-helper :
  assumes  $C \neq 0$ 
  assumes  $var \notin \text{vars } b$ 
  assumes  $var \notin \text{vars } c$ 
  assumes insertion (nth-default 0 (list-update (drop z L) var (B/C)))  $b = (B::\text{real})$ 
  assumes insertion (nth-default 0 (list-update (drop z L) var (B/C)))  $c = (C::\text{real})$ 
  assumes  $\text{lLength} : \text{length } L > \text{var} + z$ 
  shows  $\text{eval } F (\text{list-update } L (\text{var} + z) (B/C)) = (\text{eval } (\text{linear-substitution-fm-helper } \text{var } b \ c \ F \ z) (\text{list-update } L (\text{var} + z) \ v))$ 
  <proof>

```

```

theorem linear-fm :
  assumes  $\text{lLength} : \text{length } L > \text{var}$ 
  assumes  $C \neq 0$ 
  assumes  $var \notin \text{vars } b$ 
  assumes  $var \notin \text{vars } c$ 
  assumes insertion (nth-default 0 (list-update L var (B/C)))  $b = (B::\text{real})$ 
  assumes insertion (nth-default 0 (list-update L var (B/C)))  $c = (C::\text{real})$ 
  shows  $\text{eval } F (\text{list-update } L \ \text{var} \ (B/C)) = (\forall v. \text{eval } (\text{linear-substitution-fm } \text{var } b \ c \ F) (\text{list-update } L \ \text{var} \ v))$ 
  <proof>
end

```

## 8.2 Quadratic Case

```

theory QuadraticCase
  imports VSAlgos
begin

```

```

lemma quad-part-1-eq :
  assumes  $\text{lLength} : \text{length } L > \text{var}$ 
  assumes  $\text{hdeg} : \text{MPoly-Type.degree } (p::\text{real mpoly}) \ \text{var} = (\text{deg}::\text{nat})$ 
  assumes  $\text{nonzero} : D \neq 0$ 
  assumes  $ha : \forall x. \text{insertion } (\text{nth-default } 0 (\text{list-update } L \ \text{var} \ x)) \ a = (A::\text{real})$ 
  assumes  $hb : \forall x. \text{insertion } (\text{nth-default } 0 (\text{list-update } L \ \text{var} \ x)) \ b = (B::\text{real})$ 
  assumes  $hd : \forall x. \text{insertion } (\text{nth-default } 0 (\text{list-update } L \ \text{var} \ x)) \ d = (D::\text{real})$ 
  shows  $\text{aEval } (Eq \ p) (\text{list-update } L \ \text{var} \ ((A+B*C)/D)) = \text{aEval } (Eq (\text{quadratic-part-1 } \text{var } a \ b \ d \ (Eq \ p))) (\text{list-update } L \ \text{var} \ C)$ 
  <proof>

```

```

lemma quad-part-1-less :
  assumes  $\text{lLength} : \text{length } L > \text{var}$ 
  assumes  $\text{hdeg} : \text{MPoly-Type.degree } (p::\text{real mpoly}) \ \text{var} = (\text{deg}::\text{nat})$ 
  assumes  $\text{nonzero} : D \neq 0$ 
  assumes  $ha : \forall x. \text{insertion } (\text{nth-default } 0 (\text{list-update } L \ \text{var} \ x)) \ a = (A::\text{real})$ 
  assumes  $hb : \forall x. \text{insertion } (\text{nth-default } 0 (\text{list-update } L \ \text{var} \ x)) \ b = (B::\text{real})$ 

```

**assumes**  $hd : \forall x. \text{insertion } (nth\text{-default } 0 \text{ (list-update } L \text{ var } x)) \text{ } d = (D::real)$   
**shows**  $aEval \text{ (Less } p \text{) (list-update } L \text{ var } ((A+B*C)/D)) = aEval \text{ (Less(quadratic-part-1}$   
 $\text{var } a \text{ } b \text{ } d \text{ (Less } p \text{)) (list-update } L \text{ var } C)$   
 $\langle proof \rangle$

**lemma** *quad-part-1-leq* :

**assumes**  $lLength : \text{length } L > \text{var}$   
**assumes**  $hdeg : MPoly\text{-Type.degree } (p::real \text{ mpoly}) \text{ var} = (deg::nat)$   
**assumes**  $nonzero : D \neq 0$   
**assumes**  $ha : \forall x. \text{insertion } (nth\text{-default } 0 \text{ (list-update } L \text{ var } x)) \text{ } a = (A::real)$   
**assumes**  $hb : \forall x. \text{insertion } (nth\text{-default } 0 \text{ (list-update } L \text{ var } x)) \text{ } b = (B::real)$   
**assumes**  $hd : \forall x. \text{insertion } (nth\text{-default } 0 \text{ (list-update } L \text{ var } x)) \text{ } d = (D::real)$   
**shows**  $aEval \text{ (Leq } p \text{) (list-update } L \text{ var } ((A+B*C)/D)) = aEval \text{ (Leq(quadratic-part-1}$   
 $\text{var } a \text{ } b \text{ } d \text{ (Leq } p \text{)) (list-update } L \text{ var } C)$   
 $\langle proof \rangle$

**lemma** *quad-part-1-neq* :

**assumes**  $lLength : \text{length } L > \text{var}$   
**assumes**  $hdeg : MPoly\text{-Type.degree } (p::real \text{ mpoly}) \text{ var} = (deg::nat)$   
**assumes**  $nonzero : D \neq 0$   
**assumes**  $ha : \forall x. \text{insertion } (nth\text{-default } 0 \text{ (list-update } L \text{ var } x)) \text{ } a = (A::real)$   
**assumes**  $hb : \forall x. \text{insertion } (nth\text{-default } 0 \text{ (list-update } L \text{ var } x)) \text{ } b = (B::real)$   
**assumes**  $hd : \forall x. \text{insertion } (nth\text{-default } 0 \text{ (list-update } L \text{ var } x)) \text{ } d = (D::real)$   
**shows**  $aEval \text{ (Neq } p \text{) (list-update } L \text{ var } ((A+B*C)/D)) = aEval \text{ (Neq(quadratic-part-1}$   
 $\text{var } a \text{ } b \text{ } d \text{ (Neq } p \text{)) (list-update } L \text{ var } C)$   
 $\langle proof \rangle$

**lemma** *sqrt-case* :

**assumes**  $detGreater0 : SQ \geq 0$   
**shows**  $((SQ \wedge^{i \text{ div } 2}) * \text{real } (i \text{ mod } 2) * \text{sqrt } SQ + SQ \wedge^{(i \text{ div } 2)} * (1 - \text{real}$   
 $(i \text{ mod } 2))) = (\text{sqrt } SQ) \wedge^i$   
 $\langle proof \rangle$

**lemma** *sum-over-sqrt* :

**assumes**  $detGreater0 : SQ \geq 0$   
**shows**  $(\sum_{i \in \{0..<n+1\}}. ((f \text{ } i::real) * (SQ \wedge^{i \text{ div } 2}) * \text{real } (i \text{ mod } 2) * \text{sqrt } SQ$   
 $+ f \text{ } i * SQ \wedge^{(i \text{ div } 2)} * (1 - \text{real } (i \text{ mod } 2))))$   
 $= (\sum_{i \in \{0..<n+1\}}. ((f \text{ } i::real) * ((\text{sqrt } SQ) \wedge^i)))$   
 $\langle proof \rangle$

**lemma** *quad-part-2-eq* :

**assumes**  $lLength : \text{length } L > \text{var}$   
**assumes**  $detGreater0 : SQ \geq 0$   
**assumes**  $hdeg : MPoly\text{-Type.degree } (p::real \text{ mpoly}) \text{ var} = (deg :: nat)$

**assumes**  $hsq : \forall x. \text{insertion } (nth\text{-default } 0 \text{ (list-update } L \text{ var } x)) \text{ sq} = (SQ::real)$   
**shows**  $aEval \text{ (Eq } p \text{ (list-update } L \text{ var (sqrt } SQ)) = aEval \text{ (Eq(quadratic-part-2$   
 $\text{var } sq \text{ } p)) \text{ (list-update } L \text{ var (sqrt } SQ))$   
 $\langle proof \rangle$

**lemma** *quad-part-2-less* :

**assumes**  $lLength : \text{length } L > \text{var}$   
**assumes**  $detGreater0 : SQ \geq 0$   
**assumes**  $hdeg : MPoly\text{-Type.degree } (p::real \text{ mpoly}) \text{ var} = (deg :: nat)$   
**assumes**  $hsq : \forall x. \text{insertion } (nth\text{-default } 0 \text{ (list-update } L \text{ var } x)) \text{ sq} = (SQ::real)$   
**shows**  $aEval \text{ (Less } p \text{ (list-update } L \text{ var (sqrt } SQ)) = aEval \text{ (Less(quadratic-part-2$   
 $\text{var } sq \text{ } p)) \text{ (list-update } L \text{ var (sqrt } SQ))$   
 $\langle proof \rangle$

**lemma** *quad-part-2-neq* :

**assumes**  $lLength : \text{length } L > \text{var}$   
**assumes**  $detGreater0 : SQ \geq 0$   
**assumes**  $hdeg : MPoly\text{-Type.degree } (p::real \text{ mpoly}) \text{ var} = (deg :: nat)$   
**assumes**  $hsq : \forall x. \text{insertion } (nth\text{-default } 0 \text{ (list-update } L \text{ var } x)) \text{ sq} = (SQ::real)$   
**shows**  $aEval \text{ (Neq } p \text{ (list-update } L \text{ var (sqrt } SQ)) = aEval \text{ (Neq(quadratic-part-2$   
 $\text{var } sq \text{ } p)) \text{ (list-update } L \text{ var (sqrt } SQ))$   
 $\langle proof \rangle$

**lemma** *quad-part-2-leq* :

**assumes**  $lLength : \text{length } L > \text{var}$   
**assumes**  $detGreater0 : SQ \geq 0$   
**assumes**  $hdeg : MPoly\text{-Type.degree } (p::real \text{ mpoly}) \text{ var} = (deg :: nat)$   
**assumes**  $hsq : \forall x. \text{insertion } (nth\text{-default } 0 \text{ (list-update } L \text{ var } x)) \text{ sq} = (SQ::real)$   
**shows**  $aEval \text{ (Leq } p \text{ (list-update } L \text{ var (sqrt } SQ)) = aEval \text{ (Leq(quadratic-part-2$   
 $\text{var } sq \text{ } p)) \text{ (list-update } L \text{ var (sqrt } SQ))$   
 $\langle proof \rangle$

**lemma** *quad-part-2-deg* :

**assumes**  $sqfree : (\text{var}::nat) \notin \text{vars}(sq::real \text{ mpoly})$   
**shows**  $MPoly\text{-Type.degree } (\text{quadratic-part-2 var } sq \text{ } p) \text{ var} \leq 1$   
 $\langle proof \rangle$

**lemma** *quad-equality-helper* :

**assumes**  $lLength : \text{length } L > \text{var}$   
**assumes**  $detGreat0 : Cv \geq 0$   
**assumes**  $hC : \forall x. \text{insertion } (nth\text{-default } 0 \text{ (list-update } L \text{ var } x)) \text{ (C::real mpoly)}$   
 $= (Cv::real)$   
**assumes**  $hA : \forall x. \text{insertion } (nth\text{-default } 0 \text{ (list-update } L \text{ var } x)) \text{ (A::real mpoly)}$   
 $= (Av::real)$   
**assumes**  $hB : \forall x. \text{insertion } (nth\text{-default } 0 \text{ (list-update } L \text{ var } x)) \text{ (B::real mpoly)}$

= (*Bv::real*)  
**shows** *aEval* (*Eq* ( $A + B * \text{Var } var$ )) (*list-update* *L* *var* (*sqrt* *Cv*)) = *eval* (*And* (*Atom*(*Leq* ( $A*B$ ))) (*Atom* (*Eq* ( $A^2 - B^2 * C$ )))) (*list-update* *L* *var* (*sqrt* *Cv*))  
<*proof*>

**lemma** *quadratic-sub-eq* :

**assumes** *lLength* : *length* *L* > *var*  
**assumes** *nonzero* : *Dv*  $\neq$  0  
**assumes** *detGreater0* : *Cv*  $\geq$  0  
**assumes** *freeC* : *var*  $\notin$  *vars* *c*  
**assumes** *ha* :  $\forall x.$  *insertion* (*nth-default* 0 (*list-update* *L* *var* *x*)) (*a::real* *mpoly*)  
= (*Av* :: *real*)  
**assumes** *hb* :  $\forall x.$  *insertion* (*nth-default* 0 (*list-update* *L* *var* *x*)) (*b::real* *mpoly*)  
= (*Bv* :: *real*)  
**assumes** *hc* :  $\forall x.$  *insertion* (*nth-default* 0 (*list-update* *L* *var* *x*)) (*c::real* *mpoly*)  
= (*Cv* :: *real*)  
**assumes** *hd* :  $\forall x.$  *insertion* (*nth-default* 0 (*list-update* *L* *var* *x*)) (*d::real* *mpoly*)  
= (*Dv* :: *real*)  
**shows** *aEval* (*Eq* *p*) (*list-update* *L* *var* ((*Av*+*Bv*\**sqrt*(*Cv*))/*Dv*)) = *eval* (*quadratic-sub* *var* *a* *b* *c* *d* (*Eq* *p*)) (*list-update* *L* *var* (*sqrt* *Cv*))  
<*proof*>

**lemma** *quadratic-sub-less-helper* :

**assumes** *lLength* : *length* *L* > *var*  
**assumes** *detGreat0* : *Cv*  $\geq$  0  
**assumes** *hC* :  $\forall x.$  *insertion* (*nth-default* 0 (*list-update* *L* *var* *x*)) (*C::real* *mpoly*)  
= (*Cv*::*real*)  
**assumes** *hA* :  $\forall x.$  *insertion* (*nth-default* 0 (*list-update* *L* *var* *x*)) (*A::real* *mpoly*)  
= (*Av*::*real*)  
**assumes** *hB* :  $\forall x.$  *insertion* (*nth-default* 0 (*list-update* *L* *var* *x*)) (*B::real* *mpoly*)  
= (*Bv*::*real*)  
**shows** *aEval* (*Less* ( $A + B * \text{Var } var$ )) (*list-update* *L* *var* (*sqrt* *Cv*)) = *eval* (*Or* (*And* (*fm.Atom* (*Less* *A*)) (*fm.Atom* (*Less* ( $B^2 * C - A^2$ )))) (*And* (*fm.Atom* (*Leq* *B*)) (*Or* (*fm.Atom* (*Less* *A*)) (*fm.Atom* (*Less* ( $A^2 - B^2 * C$ )))))) (*list-update* *L* *var* (*sqrt* *Cv*))  
<*proof*>

**lemma** *quadratic-sub-less* :

**assumes** *lLength* : *length* *L* > *var*  
**assumes** *nonzero* : *Dv*  $\neq$  0  
**assumes** *detGreater0* : *Cv*  $\geq$  0  
**assumes** *freeC* : *var*  $\notin$  *vars* *c*  
**assumes** *ha* :  $\forall x.$  *insertion* (*nth-default* 0 (*list-update* *L* *var* *x*)) (*a::real* *mpoly*)  
= (*Av* :: *real*)  
**assumes** *hb* :  $\forall x.$  *insertion* (*nth-default* 0 (*list-update* *L* *var* *x*)) (*b::real* *mpoly*)  
= (*Bv* :: *real*)  
**assumes** *hc* :  $\forall x.$  *insertion* (*nth-default* 0 (*list-update* *L* *var* *x*)) (*c::real* *mpoly*)  
= (*Cv* :: *real*)

**assumes**  $hd : \forall x. \text{insertion } (nth\text{-default } 0 \text{ (list-update } L \text{ var } x)) \text{ (d::real mpoly)}$   
 $= (Dv :: real)$   
**shows**  $aEval \text{ (Less } p) \text{ (list-update } L \text{ var } ((Av+Bv*\text{sqrt}(Cv))/Dv)) = eval \text{ (quadratic-sub}$   
 $\text{var } a \text{ b c d (Less } p) \text{ (list-update } L \text{ var } (\text{sqrt } Cv))$   
 $\langle proof \rangle$

**lemma quadratic-sub-leq-helper :**

**assumes**  $lLength : \text{length } L > \text{var}$   
**assumes**  $detGreat0 : Cv \geq 0$   
**assumes**  $hC : \forall x. \text{insertion } (nth\text{-default } 0 \text{ (list-update } L \text{ var } x)) \text{ (C::real mpoly)}$   
 $= (Cv::real)$   
**assumes**  $hA : \forall x. \text{insertion } (nth\text{-default } 0 \text{ (list-update } L \text{ var } x)) \text{ (A::real mpoly)}$   
 $= (Av::real)$   
**assumes**  $hB : \forall x. \text{insertion } (nth\text{-default } 0 \text{ (list-update } L \text{ var } x)) \text{ (B::real mpoly)}$   
 $= (Bv::real)$   
**shows**  $aEval \text{ (Leq } (A + B * \text{Var } var)) \text{ (list-update } L \text{ var } (\text{sqrt } Cv)) =$   
 $eval \text{ (Or(And(Atom(Leq(A)))(Atom \text{ (Leq(B^2*C-A^2)))(And \text{ (Atom(Leq } B))$   
 $\text{(Atom(Leq (A^2-B^2*C)))))) \text{ (list-update } L \text{ var } (\text{sqrt } Cv))$   
 $\langle proof \rangle$

**lemma quadratic-sub-leq :**

**assumes**  $lLength : \text{length } L > \text{var}$   
**assumes**  $nonzero : Dv \neq 0$   
**assumes**  $detGreater0 : Cv \geq 0$   
**assumes**  $freeC : \text{var} \notin \text{vars } c$   
**assumes**  $ha : \forall x. \text{insertion } (nth\text{-default } 0 \text{ (list-update } L \text{ var } x)) \text{ (a::real mpoly)}$   
 $= (Av :: real)$   
**assumes**  $hb : \forall x. \text{insertion } (nth\text{-default } 0 \text{ (list-update } L \text{ var } x)) \text{ (b::real mpoly)}$   
 $= (Bv :: real)$   
**assumes**  $hc : \forall x. \text{insertion } (nth\text{-default } 0 \text{ (list-update } L \text{ var } x)) \text{ (c::real mpoly)}$   
 $= (Cv :: real)$   
**assumes**  $hd : \forall x. \text{insertion } (nth\text{-default } 0 \text{ (list-update } L \text{ var } x)) \text{ (d::real mpoly)}$   
 $= (Dv :: real)$   
**shows**  $aEval \text{ (Leq } p) \text{ (list-update } L \text{ var } ((Av+Bv*\text{sqrt}(Cv))/Dv)) = eval \text{ (quadratic-sub}$   
 $\text{var } a \text{ b c d (Leq } p) \text{ (list-update } L \text{ var } (\text{sqrt } Cv))$   
 $\langle proof \rangle$

**lemma quadratic-sub-neq :**

**assumes**  $lLength : \text{length } L > \text{var}$   
**assumes**  $nonzero : Dv \neq 0$   
**assumes**  $detGreater0 : Cv \geq 0$   
**assumes**  $freeC : \text{var} \notin \text{vars } c$   
**assumes**  $ha : \forall x. \text{insertion } (nth\text{-default } 0 \text{ (list-update } L \text{ var } x)) \text{ (a::real mpoly)}$   
 $= (Av :: real)$   
**assumes**  $hb : \forall x. \text{insertion } (nth\text{-default } 0 \text{ (list-update } L \text{ var } x)) \text{ (b::real mpoly)}$   
 $= (Bv :: real)$   
**assumes**  $hc : \forall x. \text{insertion } (nth\text{-default } 0 \text{ (list-update } L \text{ var } x)) \text{ (c::real mpoly)}$   
 $= (Cv :: real)$

**assumes**  $hd : \forall x. \text{insertion } (nth\text{-default } 0 \text{ (list-update } L \text{ var } x)) \text{ (d::real mpoly)}$   
 $= (Dv :: real)$   
**shows**  $aEval \text{ (Neq } p) \text{ (list-update } L \text{ var } ((Av+Bv*\text{sqrt}(Cv))/Dv)) = eval \text{ (quadratic-sub}$   
 $\text{var } a \text{ b c d (Neq } p)) \text{ (list-update } L \text{ var } (\text{sqrt } Cv))$   
 $\langle proof \rangle$

**theorem** *free-in-quad* :  
**assumes**  $freeA : \text{var} \notin \text{vars } a$   
**assumes**  $freeB : \text{var} \notin \text{vars } b$   
**assumes**  $freeC : \text{var} \notin \text{vars } c$   
**assumes**  $freeD : \text{var} \notin \text{vars } d$   
**shows**  $freeIn \text{ var (quadratic-sub var } a \text{ b c d } A)$   
 $\langle proof \rangle$

**theorem** *quadratic-sub* :  
**assumes**  $lLength : \text{length } L > \text{var}$   
**assumes**  $nonzero : Dv \neq 0$   
**assumes**  $detGreater0 : Cv \geq 0$   
**assumes**  $freeC : \text{var} \notin \text{vars } c$   
**assumes**  $ha : \forall x. \text{insertion } (nth\text{-default } 0 \text{ (list-update } L \text{ var } x)) \text{ (a::real mpoly)}$   
 $= (Av :: real)$   
**assumes**  $hb : \forall x. \text{insertion } (nth\text{-default } 0 \text{ (list-update } L \text{ var } x)) \text{ (b::real mpoly)}$   
 $= (Bv :: real)$   
**assumes**  $hc : \forall x. \text{insertion } (nth\text{-default } 0 \text{ (list-update } L \text{ var } x)) \text{ (c::real mpoly)}$   
 $= (Cv :: real)$   
**assumes**  $hd : \forall x. \text{insertion } (nth\text{-default } 0 \text{ (list-update } L \text{ var } x)) \text{ (d::real mpoly)}$   
 $= (Dv :: real)$   
**shows**  $aEval A \text{ (list-update } L \text{ var } ((Av+Bv*\text{sqrt}(Cv))/Dv)) = eval \text{ (quadratic-sub}$   
 $\text{var } a \text{ b c d } A) \text{ (list-update } L \text{ var } (\text{sqrt } Cv))$   
 $\langle proof \rangle$

**lemma** *free-in-quad-fm-helper* :  
**assumes**  $freeA : \text{var} \notin \text{vars } a$   
**assumes**  $freeB : \text{var} \notin \text{vars } b$   
**assumes**  $freeC : \text{var} \notin \text{vars } c$   
**assumes**  $freeD : \text{var} \notin \text{vars } d$   
**shows**  $freeIn \text{ (var+z) (quadratic-sub-fm-helper var } a \text{ b c d } F \text{ z)}$   
 $\langle proof \rangle$

**theorem** *free-in-quad-fm* :  
**assumes**  $freeA : \text{var} \notin \text{vars } a$   
**assumes**  $freeB : \text{var} \notin \text{vars } b$   
**assumes**  $freeC : \text{var} \notin \text{vars } c$   
**assumes**  $freeD : \text{var} \notin \text{vars } d$   
**shows**  $freeIn \text{ var (quadratic-sub-fm var } a \text{ b c d } A)$   
 $\langle proof \rangle$

```

lemma quadratic-sub-fm-helper :
  assumes nonzero :  $Dv \neq 0$ 
  assumes detGreater0 :  $Cv \geq 0$ 
  assumes freeC :  $var \notin vars\ c$ 
  assumes lLength :  $length\ L > var+z$ 
  assumes ha :  $\forall x. insertion\ (nth-default\ 0\ (list-update\ (drop\ z\ L)\ var\ x))\ (a::real\ mpoly) = (Av :: real)$ 
  assumes hb :  $\forall x. insertion\ (nth-default\ 0\ (list-update\ (drop\ z\ L)\ var\ x))\ (b::real\ mpoly) = (Bv :: real)$ 
  assumes hc :  $\forall x. insertion\ (nth-default\ 0\ (list-update\ (drop\ z\ L)\ var\ x))\ (c::real\ mpoly) = (Cv :: real)$ 
  assumes hd :  $\forall x. insertion\ (nth-default\ 0\ (list-update\ (drop\ z\ L)\ var\ x))\ (d::real\ mpoly) = (Dv :: real)$ 
  shows  $eval\ F\ (list-update\ L\ (var+z)\ ((Av+Bv*sqrt(Cv))/Dv)) = eval\ (quadratic-sub-fm-helper\ var\ a\ b\ c\ d\ F\ z)\ (list-update\ L\ (var+z)\ (sqrt\ Cv))$ 
  <proof>

```

```

theorem quadratic-sub-fm :
  assumes lLength :  $length\ L > var$ 
  assumes nonzero :  $Dv \neq 0$ 
  assumes detGreater0 :  $Cv \geq 0$ 
  assumes freeC :  $var \notin vars\ c$ 
  assumes ha :  $\forall x. insertion\ (nth-default\ 0\ (list-update\ L\ var\ x))\ (a::real\ mpoly) = (Av :: real)$ 
  assumes hb :  $\forall x. insertion\ (nth-default\ 0\ (list-update\ L\ var\ x))\ (b::real\ mpoly) = (Bv :: real)$ 
  assumes hc :  $\forall x. insertion\ (nth-default\ 0\ (list-update\ L\ var\ x))\ (c::real\ mpoly) = (Cv :: real)$ 
  assumes hd :  $\forall x. insertion\ (nth-default\ 0\ (list-update\ L\ var\ x))\ (d::real\ mpoly) = (Dv :: real)$ 
  shows  $eval\ F\ (list-update\ L\ var\ ((Av+Bv*sqrt(Cv))/Dv)) = eval\ (quadratic-sub-fm\ var\ a\ b\ c\ d\ F)\ (list-update\ L\ var\ (sqrt\ Cv))$ 
  <proof>
end

```

### 8.3 Lemmas of the elimVar function

```

theory EliminateVariable
  imports LinearCase QuadraticCase HOL-Library.Quadratic-Discriminant
begin

```

```

lemma elimVar-eq :
  assumes hlength :  $length\ xs = var$ 

```

```

assumes in-list :  $Eq\ p \in set(L)$ 
assumes low-pow :  $MPoly-Type.degree\ p\ var = 1 \vee MPoly-Type.degree\ p\ var =$ 
2
shows  $((\exists x. eval\ (list-conj\ (map\ fm.Atom\ L\ @\ F))\ (xs\ @\ x\ \#\ \Gamma)) =$ 
 $((\exists x. eval\ (elimVar\ var\ L\ F\ (Eq\ p))\ (xs\ @\ x\ \#\ \Gamma))) \vee (\forall x. aEval\ (Eq\ p)\ (xs\ @$ 
 $x\ \#\ \Gamma)))$ 
<proof>

```

simply states that the variable is free in the equality case of the elimVar function

```

lemma freeIn-elimVar-eq :  $freeIn\ var\ (elimVar\ var\ L\ F\ (Eq\ p))$ 
<proof>

```

Theorem 20.2 in the textbook

```

lemma elimVar-eq-2 :
assumes hlength :  $length\ xs = var$ 
assumes in-list :  $Eq\ p \in set(L)$ 
assumes low-pow :  $MPoly-Type.degree\ p\ var = 1 \vee MPoly-Type.degree\ p\ var =$ 
2
assumes nonzero :  $\forall x.$ 
 $insertion\ (nth-default\ 0\ (xs\ @\ x\ \#\ \Gamma))\ (isolate-variable-sparse\ p\ var\ 2)$ 
 $\neq 0$ 
 $\vee insertion\ (nth-default\ 0\ (xs\ @\ x\ \#\ \Gamma))\ (isolate-variable-sparse\ p\ var$ 
1)  $\neq 0$ 
 $\vee insertion\ (nth-default\ 0\ (xs\ @\ x\ \#\ \Gamma))\ (isolate-variable-sparse\ p\ var$ 
0)  $\neq 0$  (is ?non0)
shows  $(\exists x. eval\ (list-conj\ (map\ fm.Atom\ L\ @\ F))\ (xs\ @\ x\ \#\ \Gamma)) =$ 
 $(\exists x. eval\ (elimVar\ var\ L\ F\ (Eq\ p))\ (xs\ @\ x\ \#\ \Gamma))$ 
<proof>

```

end

## 8.4 Overall LuckyFind Proofs

```

theory LuckyFind
imports EliminateVariable
begin

```

```

theorem luckyFind-eval:
assumes luckyFind  $x\ L\ F = Some\ F'$ 
assumes length  $xs = x$ 
shows  $(\exists x. (eval\ (list-conj\ ((map\ Atom\ L)\ @\ F))\ (xs\ @\ (x\ \#\ \Gamma)))) = (\exists x. (eval$ 
 $F'\ (xs\ @\ (x\ \#\ \Gamma))))$ 
<proof>

```

**lemma** *luckyFind'-eval* :  
**assumes** *length xs = var*  
**shows**  $(\exists x. \text{eval } (\text{list-conj } (\text{map } \text{fm.Atom } L \text{ @ } F)) (xs \text{ @ } x \# \Gamma)) = (\exists x. \text{eval } (\text{luckyFind}' \text{ var } L \text{ } F) (xs \text{ @ } x \# \Gamma))$   
 $\langle \text{proof} \rangle$

**lemma** *luckiestFind-eval* :  
**assumes** *length xs = var*  
**shows**  $(\exists x. \text{eval } (\text{list-conj } (\text{map } \text{fm.Atom } L \text{ @ } F)) (xs \text{ @ } x \# \Gamma)) = (\exists x. \text{eval } (\text{luckiestFind} \text{ var } L \text{ } F) (xs \text{ @ } x \# \Gamma))$   
 $\langle \text{proof} \rangle$

**end**

## 8.5 Overall Equality VS Proofs

**theory** *EqualityVS*  
**imports** *EliminateVariable LuckyFind*  
**begin**

**lemma** *degree-find-eq* :  
**assumes** *find-eq var L = (A,L')*  
**shows**  $\forall p \in \text{set}(A). \text{MPoly-Type.degree } p \text{ var} = 1 \vee \text{MPoly-Type.degree } p \text{ var} = 2$   
 $\langle \text{proof} \rangle$

**lemma** *list-in-find-eq* :  
**assumes** *find-eq var L = (A,L')*  
**shows**  $\text{set}(\text{map } \text{Eq } A \text{ @ } L') = \text{set } L$   
 $\langle \text{proof} \rangle$

**lemma** *qe-eq-one-eval* :  
**assumes** *hlength : length xs = var*  
**shows**  $(\exists x. (\text{eval } (\text{list-conj } ((\text{map } \text{Atom } L) \text{ @ } F)) (xs \text{ @ } (x\#\Gamma)))) = (\exists x. (\text{eval } (\text{qe-eq-one} \text{ var } L \text{ } F) (xs \text{ @ } (x\#\Gamma))))$   
 $\langle \text{proof} \rangle$

**lemma** *qe-eq-repeat-helper-eval-case1* :  
**assumes** *hlength : length xs = var*  
**assumes** *degreeGood :  $\forall p \in \text{set}(A). \text{MPoly-Type.degree } p \text{ var} = 1 \vee \text{MPoly-Type.degree } p \text{ var} = 2$*   
**shows**  $((\text{eval } (\text{list-conj } ((\text{map } (\text{Atom } \circ \text{Eq}) \text{ } A) \text{ @ } (\text{map } \text{Atom } L) \text{ @ } F)) (xs \text{ @ } (x\#\Gamma))))$

$\implies (eval (qe-eq-repeat-helper var A L F) (xs @ x \# \Gamma))$   
 $\langle proof \rangle$

**lemma** *qe-eq-repeat-helper-eval-case2* :  
**assumes** *hlength* : *length xs = var*  
**assumes** *degreeGood* :  $\forall p \in set(A). MPoly-Type.degree p var = 1 \vee MPoly-Type.degree p var = 2$   
**shows**  $(eval (qe-eq-repeat-helper var A L F) (xs @ x \# \Gamma))$   
 $\implies \exists x. ((eval (list-conj ((map (Atom o Eq) A) @ (map Atom L) @ F))$   
 $(xs @ (x \# \Gamma))))$   
 $\langle proof \rangle$

**lemma** *qe-eq-repeat-eval* :  
**assumes** *hlength* : *length xs = var*  
**shows**  $(\exists x. (eval (list-conj ((map Atom L) @ F)) (xs @ (x \# \Gamma)))) = (\exists x. (eval (qe-eq-repeat var L F) (xs @ (x \# \Gamma))))$   
 $\langle proof \rangle$

**end**

## 9 General VS Proofs

### 9.1 Univariate Atoms

**theory** *UniAtoms*  
**imports** *Debruijn*  
**begin**

**datatype** *atomUni* = *LessUni real \* real \* real* | *EqUni real \* real \* real* | *LeqUni real \* real \* real* | *NeqUni real \* real \* real*

**datatype** (*atoms: 'a*) *fmUni* =  
*TrueFUni* | *FalseFUni* | *AtomUni 'a* | *AndUni 'a fmUni 'a fmUni* | *OrUni 'a fmUni 'a fmUni*

**fun** *aEvalUni* :: *atomUni*  $\Rightarrow$  *real*  $\Rightarrow$  *bool* **where**  
*aEvalUni* (*EqUni* (*a,b,c*)) *x* = ( $a*x^2+b*x+c = 0$ ) |  
*aEvalUni* (*LessUni* (*a,b,c*)) *x* = ( $a*x^2+b*x+c < 0$ ) |  
*aEvalUni* (*LeqUni* (*a,b,c*)) *x* = ( $a*x^2+b*x+c \leq 0$ ) |  
*aEvalUni* (*NeqUni* (*a,b,c*)) *x* = ( $a*x^2+b*x+c \neq 0$ )

**fun** *aNegUni* :: *atomUni*  $\Rightarrow$  *atomUni* **where**  
*aNegUni* (*LessUni* (*a,b,c*)) = *LeqUni* ( $-a,-b,-c$ ) |  
*aNegUni* (*EqUni* *p*) = *NeqUni* *p* |  
*aNegUni* (*LeqUni* (*a,b,c*)) = *LessUni* ( $-a,-b,-c$ ) |  
*aNegUni* (*NeqUni* *p*) = *EqUni* *p*

```

fun evalUni :: atomUni fmUni ⇒ real ⇒ bool where
  evalUni (AtomUni a) x = aEvalUni a x |
  evalUni (TrueFUni) - = True |
  evalUni (FalseFUni) - = False |
  evalUni (AndUni φ ψ) x = ((evalUni φ x) ∧ (evalUni ψ x)) |
  evalUni (OrUni φ ψ) x = ((evalUni φ x) ∨ (evalUni ψ x))

```

```

fun negUni :: atomUni fmUni ⇒ atomUni fmUni where
  negUni (AtomUni a) = AtomUni(aNegUni a) |
  negUni (TrueFUni) = FalseFUni |
  negUni (FalseFUni) = TrueFUni |
  negUni (AndUni φ ψ) = (OrUni (negUni φ) (negUni ψ)) |
  negUni (OrUni φ ψ) = (AndUni (negUni φ) (negUni ψ))

```

```

fun convert-poly :: nat ⇒ real mpoly ⇒ real list ⇒ (real * real * real) option
where
  convert-poly var p xs = (
    if MPoly-Type.degree p var < 3
    then let (A,B,C) = get-coeffs var p in Some(insertion (nth-default 0 (xs)) A,insertion
      (nth-default 0 (xs)) B,insertion (nth-default 0 (xs)) C)
    else None)

```

```

fun convert-atom :: nat ⇒ atom ⇒ real list ⇒ atomUni option where
  convert-atom var (Less p) xs = map-option LessUni (convert-poly var p xs)|
  convert-atom var (Eq p) xs = map-option EqUni (convert-poly var p xs)|
  convert-atom var (Leq p) xs = map-option LeqUni (convert-poly var p xs)|
  convert-atom var (Neq p) xs = map-option NeqUni (convert-poly var p xs)

```

```

lemma convert-atom-change :
  assumes length xs' = var
  shows convert-atom var At (xs' @ x # Γ) = convert-atom var At (xs' @ x' # Γ)
  ⟨proof⟩

```

```

lemma degree-convert-eq :
  assumes convert-poly var p xs = Some(a)
  shows MPoly-Type.degree p var < 3
  ⟨proof⟩

```

```

lemma poly-to-univar :
  assumes MPoly-Type.degree p var < 3
  assumes get-coeffs var p = (A,B,C)
  assumes a = insertion (nth-default 0 (xs'@y#xs)) A
  assumes b = insertion (nth-default 0 (xs'@y#xs)) B
  assumes c = insertion (nth-default 0 (xs'@y#xs)) C
  assumes length xs' = var
  shows insertion (nth-default 0 (xs'@x#xs)) p = (a*x^2)+(b*x)+c
  ⟨proof⟩

```

**lemma** *aEval-aEvalUni*:

**assumes** *convert-atom var a (xs'@x#xs) = Some a'*  
**assumes** *length xs' = var*  
**shows** *aEval a (xs'@x#xs) = aEvalUni a' x*  
 ⟨*proof*⟩

**fun** *convert-fm* :: *nat* ⇒ *atom fm* ⇒ *real list* ⇒ (*atomUni fmUni*) **option** **where**  
*convert-fm var (Atom a) Γ = map-option (AtomUni) (convert-atom var a Γ) |*  
*convert-fm var (TrueF) - = Some TrueFUni |*  
*convert-fm var (FalseF) - = Some FalseFUni |*  
*convert-fm var (And φ ψ) Γ = (case ((convert-fm var φ Γ),(convert-fm var ψ*  
*Γ)) of (Some a, Some b) ⇒ Some (AndUni a b) | - ⇒ None) |*  
*convert-fm var (Or φ ψ) Γ = (case ((convert-fm var φ Γ),(convert-fm var ψ Γ))*  
*of (Some a, Some b) ⇒ Some (OrUni a b) | - ⇒ None) |*  
*convert-fm var (Neg φ) Γ = None |*  
*convert-fm var (ExQ φ) Γ = None |*  
*convert-fm var (AllQ φ) Γ = None|*  
*convert-fm var (AllN i φ) Γ = None|*  
*convert-fm var (ExN i φ) Γ = None*

**lemma** *eval-evalUni*:

**assumes** *convert-fm var F (xs'@x#xs) = Some F'*  
**assumes** *length xs' = var*  
**shows** *eval F (xs'@x#xs) = evalUni F' x*  
 ⟨*proof*⟩

**fun** *grab-atoms* :: *nat* ⇒ *atom fm* ⇒ *atom list* **option** **where**

*grab-atoms var TrueF = Some([]) |*  
*grab-atoms var FalseF = Some([]) |*  
*grab-atoms var (Atom(Eq p)) = (if MPoly-Type.degree p var < 3 then (if MPoly-Type.degree*  
*p var > 0 then Some([Eq p]) else Some([])) else None)|*  
*grab-atoms var (Atom(Less p)) = (if MPoly-Type.degree p var < 3 then (if*  
*MPoly-Type.degree p var > 0 then Some([Less p]) else Some([])) else None)|*  
*grab-atoms var (Atom(Leq p)) = (if MPoly-Type.degree p var < 3 then (if*  
*MPoly-Type.degree p var > 0 then Some([Leq p]) else Some([])) else None)|*  
*grab-atoms var (Atom(Neq p)) = (if MPoly-Type.degree p var < 3 then (if*  
*MPoly-Type.degree p var > 0 then Some([Neq p]) else Some([])) else None)|*  
*grab-atoms var (And a b) = (*  
*case grab-atoms var a of*  
*Some(al) ⇒ (*  
*case grab-atoms var b of*  
*Some(bl) ⇒ Some(al@bl)*  
*| None ⇒ None*  
*)*  
*| None ⇒ None*  
*)|*

```

    grab-atoms var (Or a b) = (
case grab-atoms var a of
  Some(al) ⇒ (
    case grab-atoms var b of
      Some(bl) ⇒ Some(al@bl)
    | None ⇒ None
  )
| None ⇒ None
)|

```

```

grab-atoms var (Neg -) = None|
grab-atoms var (ExQ -) = None|
grab-atoms var (AllQ -) = None|
grab-atoms var (AllN i -) = None|
grab-atoms var (ExN i -) = None

```

**lemma** *nil-grab* : (grab-atoms var F = Some []) ⇒ (freeIn var F)  
<proof>

**fun** *isSome* :: 'a option ⇒ bool **where**  
*isSome* (Some -) = True |  
*isSome* None = False

**lemma** *grab-atoms-convert* : (isSome (grab-atoms var F)) = (isSome (convert-fm  
var F xs))  
<proof>

**lemma** *convert-aNeg* :  
**assumes** *convert-atom* var A (xs'@x#xs) = Some(A')  
**assumes** *length* xs' = var  
**shows** *aEval* (aNeg A) (xs'@x#xs) = *aEvalUni* (aNegUni A') x  
<proof>

**lemma** *convert-neg* :  
**assumes** *convert-fm* var F (xs'@x#xs) = Some(F')  
**assumes** *length* xs' = var  
**shows** *eval* (Neg F) (xs'@x#xs) = *evalUni* (negUni F') x  
<proof>

**fun** *list-disj-Uni* :: 'a fmUni list ⇒ 'a fmUni **where**  
*list-disj-Uni* [] = FalseFUni|  
*list-disj-Uni* (x#xs) = OrUni x (list-disj-Uni xs)

**fun** *list-conj-Uni* :: 'a fmUni list ⇒ 'a fmUni **where**  
*list-conj-Uni* [] = TrueFUni|  
*list-conj-Uni* (x#xs) = AndUni x (list-conj-Uni xs)

**lemma** *eval-list-disj-Uni* :  $evalUni (list-disj-Uni L) x = (\exists l \in set(L). evalUni l x)$   
 ⟨proof⟩

**lemma** *eval-list-conj-Uni* :  $evalUni (list-conj-Uni A) x = (\forall l \in set A. evalUni l x)$   
 ⟨proof⟩

**lemma** *eval-list-conj-Uni-append* :  $evalUni (list-conj-Uni (A @ B)) x = (evalUni (list-conj-Uni (A)) x \wedge evalUni (list-conj-Uni (B)) x)$   
 ⟨proof⟩

**fun** *map-atomUni* :: ('a ⇒ 'a fmUni) ⇒ 'a fmUni ⇒ 'a fmUni **where**  
*map-atomUni* f (AtomUni a) = f a |  
*map-atomUni* f (TrueFUni) = TrueFUni |  
*map-atomUni* f (FalseFUni) = FalseFUni |  
*map-atomUni* f (AndUni φ ψ) = (AndUni (map-atomUni f φ) (map-atomUni f ψ)) |  
*map-atomUni* f (OrUni φ ψ) = (OrUni (map-atomUni f φ) (map-atomUni f ψ))

**fun** *map-atom* :: (atom ⇒ atom fm) ⇒ atom fm ⇒ atom fm **where**  
*map-atom* f TrueF = TrueF |  
*map-atom* f FalseF = FalseF |  
*map-atom* f (Atom a) = f a |  
*map-atom* f (And φ ψ) = And (map-atom f φ) (map-atom f ψ) |  
*map-atom* f (Or φ ψ) = Or (map-atom f φ) (map-atom f ψ) |  
*map-atom* f (Neg φ) = TrueF |  
*map-atom* f (ExQ φ) = TrueF |  
*map-atom* f (AllQ φ) = TrueF |  
*map-atom* f (ExN i φ) = TrueF |  
*map-atom* f (AllN i φ) = TrueF

**fun** *getPoly* :: atomUni => real \* real \* real **where**  
*getPoly* (EqUni p) = p |  
*getPoly* (LeqUni p) = p |  
*getPoly* (NeqUni p) = p |  
*getPoly* (LessUni p) = p

**lemma** *liftatom-map-atom* :  
**assumes**  $\exists F'. convert-fm var F xs = Some F'$   
**shows**  $liftmap f F 0 = map-atom (f 0) F$   
 ⟨proof⟩

**lemma** *eval-map* :  $(\exists l \in set(map f L). evalUni l x) = (\exists l \in set(L). evalUni (f l) x)$   
 ⟨proof⟩

**lemma** *eval-map-all* :  $(\forall l \in set(map f L). evalUni l x) = (\forall l \in set(L). evalUni (f l) x)$   
 ⟨proof⟩

**lemma** *eval-append* :  $(\exists l \in \text{set } (A \# B). \text{evalUni } l \ x) = (\text{evalUni } A \ x \vee (\exists l \in \text{set } (B). \text{evalUni } l \ x))$   
 ⟨proof⟩

**lemma** *eval-conj-atom* :  $\text{evalUni } (\text{list-conj-Uni } (\text{map } \text{AtomUni } L)) \ x = (\forall l \in \text{set}(L). \text{aEvalUni } l \ x)$   
 ⟨proof⟩  
**end**

## 9.2 Negative Infinity

**theory** *NegInfinity*  
**imports** *HOL-Analysis.Poly-Roots VSAlgos*  
**begin**

**lemma** *freeIn-allzero* :  $\text{freeIn } \text{var } (\text{allZero } p \ \text{var})$   
 ⟨proof⟩

**lemma** *allzero-eval* :  
**assumes**  $l\text{Length} : \text{var} < \text{length } L$   
**shows**  $(\exists x. \forall y < x. \text{aEval } (\text{Eq } p) (\text{list-update } L \ \text{var } y) ) = (\forall x. \text{eval } (\text{allZero } p \ \text{var}) (\text{list-update } L \ \text{var } x))$   
 ⟨proof⟩

**lemma** *freeIn-altNegInf* :  $\text{freeIn } \text{var } (\text{alternateNegInfinity } p \ \text{var})$   
 ⟨proof⟩

**theorem** *freeIn-substNegInfinity* :  $\text{freeIn } \text{var } (\text{substNegInfinity } \text{var } A)$   
 ⟨proof⟩

**end**  
**theory** *NegInfinityUni*  
**imports** *UniAtoms NegInfinity QE*  
**begin**

**fun** *allZero'* ::  $\text{real} * \text{real} * \text{real} \Rightarrow \text{atomUni } \text{fmUni}$  **where**  
*allZero'*  $(a, b, c) = \text{AndUni}(\text{AndUni}(\text{AtomUni}(\text{EqUni}(0, 0, a))) (\text{AtomUni}(\text{EqUni}(0, 0, b))))$   
 $(\text{AtomUni}(\text{EqUni}(0, 0, c)))$

**lemma** *convert-allZero* :

```

assumes convert-poly var p (xs'@x#xs) = Some p'
assumes length xs' = var
shows eval (allZero p var) (xs'@x#xs) = evalUni (allZero' p') x
⟨proof⟩

```

```

fun alternateNegInfinity' :: real * real * real ⇒ atomUni fmUni where
  alternateNegInfinity' (a,b,c) =
  OrUni(AtomUni(LessUni(0,0,a)))(
  AndUni(AtomUni(EqUni(0,0,a)) (
    OrUni(AtomUni(LessUni(0,0,-b)))(
    AndUni(AtomUni(EqUni(0,0,b)))(
      AtomUni(LessUni(0,0,c))
    )
  )
  )
  )

```

```

lemma convert-alternateNegInfinity :
assumes convert-poly var p (xs'@x#xs) = Some X
assumes length xs' = var
shows eval (alternateNegInfinity p var) (xs'@x#xs) = evalUni (alternateNegInfinity'
X) x
⟨proof⟩

```

```

fun substNegInfinityUni :: atomUni ⇒ atomUni fmUni where
  substNegInfinityUni (EqUni p) = allZero' p |
  substNegInfinityUni (LessUni p) = alternateNegInfinity' p|
  substNegInfinityUni (LeqUni p) = OrUni (alternateNegInfinity' p) (allZero' p)|
  substNegInfinityUni (NeqUni p) = negUni (allZero' p)

```

```

lemma convert-substNegInfinity :
assumes convert-atom var A (xs'@x#xs) = Some(A')
assumes length xs' = var
shows eval (substNegInfinity var A) (xs'@x#xs) = evalUni (substNegInfinityUni
A') x
⟨proof⟩

```

```

lemma change-eval-eq:
fixes x y:: real
assumes ((aEvalUni (EqUni(a,b,c)) x ≠ aEvalUni (EqUni(a,b,c)) y) ∧ x < y)
shows (∃ w. x ≤ w ∧ w ≤ y ∧ a*w2 + b*w + c = 0)
⟨proof⟩

```

```

lemma change-eval-lt:
fixes x y:: real
assumes ((aEvalUni (LessUni (a,b,c)) x ≠ aEvalUni (LessUni (a,b,c)) y) ∧ x

```

$< y$ )  
**shows**  $(\exists w. x \leq w \wedge w \leq y \wedge a*w^2 + b*w + c = 0)$   
 $\langle proof \rangle$

**lemma** *no-change-eval-lt*:

**fixes**  $x y:: real$   
**assumes**  $x < y$   
**assumes**  $\neg(\exists w. x \leq w \wedge w \leq y \wedge a*w^2 + b*w + c = 0)$   
**shows**  $((aEvalUni (LessUni (a,b,c)) x = aEvalUni (LessUni (a,b,c)) y))$   
 $\langle proof \rangle$

**lemma** *change-eval-neq*:

**fixes**  $x y:: real$   
**assumes**  $((aEvalUni (NeqUni (a,b,c)) x \neq aEvalUni (NeqUni (a,b,c)) y) \wedge x < y)$   
**shows**  $(\exists w. x \leq w \wedge w \leq y \wedge a*w^2 + b*w + c = 0)$   
 $\langle proof \rangle$

**lemma** *change-eval-leq*:

**fixes**  $x y:: real$   
**assumes**  $((aEvalUni (LeqUni (a,b,c)) x \neq aEvalUni (LeqUni (a,b,c)) y) \wedge x < y)$   
**shows**  $(\exists w. x \leq w \wedge w \leq y \wedge a*w^2 + b*w + c = 0)$   
 $\langle proof \rangle$

**lemma** *change-eval*:

**fixes**  $x y:: real$   
**fixes**  $At:: atomUni$   
**assumes**  $xlt: x < y$   
**assumes** *noteq*:  $((aEvalUni At) x \neq aEvalUni (At) y)$   
**assumes** *getPoly*  $At = (a, b, c)$   
**shows**  $(\exists w. x \leq w \wedge w \leq y \wedge a*w^2 + b*w + c = 0)$   
 $\langle proof \rangle$

**lemma** *no-change-eval*:

**fixes**  $x y:: real$   
**assumes** *getPoly*  $At = (a, b, c)$   
**assumes**  $x < y$   
**assumes**  $\neg(\exists w. x \leq w \wedge w \leq y \wedge a*w^2 + b*w + c = 0)$   
**shows**  $((aEvalUni At) x = aEvalUni (At) y \wedge x < y)$   
 $\langle proof \rangle$

**lemma** *same-eval''* :

**assumes** *getPoly*  $At = (a, b, c)$   
**assumes** *nonz*:  $a \neq 0 \vee b \neq 0 \vee c \neq 0$   
**shows**  $\exists x. \forall y < x. (aEvalUni At y = aEvalUni At x)$   
 $\langle proof \rangle$

**lemma** *inequality-case* :  $(\exists (x::real). \forall (y::real) < x. (a::real) * y^2 + (b::real) * y + (c::real) < 0) =$   
 $(a < 0 \vee a = 0 \wedge (0 < b \vee b = 0 \wedge c < 0))$   
 $\langle proof \rangle$

**lemma** *inequality-case-geq* :  $(\exists (x::real). \forall (y::real) < x. (a::real) * y^2 + (b::real) * y + (c::real) > 0) =$   
 $(a > 0 \vee a = 0 \wedge (0 > b \vee b = 0 \wedge c > 0))$   
 $\langle proof \rangle$

**lemma** *infinity-evalUni-LessUni* :  $(\exists x. \forall y < x. aEvalUni (LessUni p) y) = (evalUni (substNegInfinityUni (LessUni p)) x)$   
 $\langle proof \rangle$

**lemma** *infinity-evalUni-EqUni* :  $(\exists x. \forall y < x. aEvalUni (EqUni p) y) = (evalUni (substNegInfinityUni (EqUni p)) x)$   
 $\langle proof \rangle$

**lemma** *infinity-evalUni-NeqUni* :  $(\exists x. \forall y < x. aEvalUni (NeqUni p) y) = (evalUni (substNegInfinityUni (NeqUni p)) x)$   
 $\langle proof \rangle$

**lemma** *infinity-evalUni-LeqUni* :  $(\exists x. \forall y < x. aEvalUni (LeqUni p) y) = (evalUni (substNegInfinityUni (LeqUni p)) x)$   
 $\langle proof \rangle$

This is the vertical translation for `substNegInfinityUni` where we represent the virtual substitution of negative infinity in the univariate case

**lemma** *infinity-evalUni* :  
**shows**  $(\exists x. \forall y < x. aEvalUni At y) = (evalUni (substNegInfinityUni At) x)$   
 $\langle proof \rangle$

**end**

### 9.3 Infinitesimals

**theory** *Infinitesimals*

**imports** *ExecutablePolyProps LinearCase QuadraticCase NegInfinity Debruijn*  
**begin**

**lemma** *freeIn-substInfinitesimalQuadratic* :  
**assumes**  $var \notin vars\ a$   
 $var \notin vars\ b$   
 $var \notin vars\ c$   
 $var \notin vars\ d$   
**shows**  $freeIn\ var\ (substInfinitesimalQuadratic\ var\ a\ b\ c\ d\ At)$   
 $\langle proof \rangle$

```

lemma freeIn-substInfinitesimalQuadratic-fm :
  assumes var  $\notin$  vars a var  $\notin$  vars b var  $\notin$  vars c var  $\notin$  vars d
  shows freeIn var (substInfinitesimalQuadratic-fm var a b c d F)
  <proof>

lemma freeIn-substInfinitesimalLinear:
  assumes var  $\notin$  vars a var  $\notin$  vars b
  shows freeIn var (substInfinitesimalLinear var a b At)
  <proof>

lemma freeIn-substInfinitesimalLinear-fm:
  assumes var  $\notin$  vars a var  $\notin$  vars b
  shows freeIn var (substInfinitesimalLinear-fm var a b F)
  <proof>

end
theory InfinitesimalsUni
  imports Infinitesimals UniAtoms NegInfinityUni QE

begin

fun convertDerivativeUni :: real * real * real  $\Rightarrow$  atomUni fmUni where
  convertDerivativeUni (a,b,c) =
    OrUni(AtomUni(LessUni(a,b,c)))(AndUni(AtomUni(EqUni(a,b,c)))(
      OrUni(AtomUni(LessUni(0,2*a,b)))(AndUni(AtomUni(EqUni(0,2*a,b)))(
        (AtomUni(LessUni(0,0,2*a)))
      ))
    ))

lemma convert-convertDerivative :
  assumes convert-poly var p (xs'@x#xs) = Some(a,b,c)
  assumes length xs' = var
  shows eval (convertDerivative var p) (xs'@x#xs) = evalUni (convertDerivativeUni
(a,b,c) x
  <proof>

fun linearSubstitutionUni :: real  $\Rightarrow$  real  $\Rightarrow$  atomUni  $\Rightarrow$  atomUni fmUni where
  linearSubstitutionUni b c a = (if aEvalUni a (-c/b) then TrueFUni else False-
FUni)

lemma convert-linearSubstitutionUni:
  assumes convert-atom var a (xs'@x#xs) = Some(a')
  assumes insertion (nth-default 0 (xs'@x#xs)) b = B

```

```

assumes insertion (nth-default 0 (xs'@x#xs)) c = C
assumes B ≠ 0
assumes var∉(vars b)
assumes var∉(vars c)
assumes length xs' = var
shows aEval (linear-substitution var (-c) b a) (xs'@x#xs) = evalUni (linearSubstitutionUni
B C a') x
⟨proof⟩

```

```

fun substInfinitesimalLinearUni :: real ⇒ real ⇒ atomUni ⇒ atomUni fmUni
where

```

```

  substInfinitesimalLinearUni b c (EqUni p) = allZero' p|
  substInfinitesimalLinearUni b c (LessUni p) =
  map-atomUni (linearSubstitutionUni b c) (convertDerivativeUni p)|
  substInfinitesimalLinearUni b c (LeqUni p) =
OrUni
  (allZero' p)
  (map-atomUni (linearSubstitutionUni b c) (convertDerivativeUni p))|
  substInfinitesimalLinearUni b c (NeqUni p) = negUni (allZero' p)

```

```

lemma convert-linear-subst-fm :

```

```

assumes convert-atom var a (xs'@x#xs) = Some a'
assumes insertion (nth-default 0 (xs'@x#xs)) b = B
assumes insertion (nth-default 0 (xs'@x#xs)) c = C
assumes B ≠ 0
assumes var∉(vars b)
assumes var∉(vars c)
assumes length xs' = var
shows aEval (linear-substitution (var + 0) (liftPoly 0 0 (-c)) (liftPoly 0 0 b)
a) (xs'@x#xs) =
  evalUni (linearSubstitutionUni B C a') x
⟨proof⟩

```

```

lemma evalUni-if : evalUni (if cond then TrueFUni else FalseFUni) x = cond
⟨proof⟩

```

```

lemma degree-less-sum' : MPoly-Type.degree (p::real mpoly) var = n ⇒ MPoly-Type.degree
(q::real mpoly) var = m ⇒ n < m ⇒ MPoly-Type.degree (p + q) var = m
⟨proof⟩

```

```

lemma convert-substInfinitesimalLinear-less :

```

```

assumes convert-poly var p (xs'@x#xs) = Some(p')
assumes insertion (nth-default 0 (xs'@x#xs)) b = B
assumes insertion (nth-default 0 (xs'@x#xs)) c = C
assumes B ≠ 0
assumes var∉(vars b)
assumes var∉(vars c)
assumes length xs' = var

```

**shows**  
*eval* (*liftmap*  
 $(\lambda x. \lambda A. \text{Atom}(\text{linear-substitution } (\text{var}+x) (\text{liftPoly } 0 \ x \ (-c)) (\text{liftPoly } 0 \ x \ b) \ A))$   
 $(\text{convertDerivative } \text{var } p)$   
 $0) (xs'@x\#xs) =$   
*evalUni* (*map-atomUni* (*linearSubstitutionUni* *B C*) (*convertDerivativeUni* *p*')) *x*  
 $\langle \text{proof} \rangle$

**lemma** *convert-substInfinitesimalLinear*:  
**assumes** *convert-atom* *var* *a*  $(xs'@x\#xs) = \text{Some}(a')$   
**assumes** *insertion* (*nth-default*  $0 (xs'@x\#xs)$ ) *b* = *B*  
**assumes** *insertion* (*nth-default*  $0 (xs'@x\#xs)$ ) *c* = *C*  
**assumes**  $B \neq 0$   
**assumes**  $\text{var} \notin (\text{vars } b)$   
**assumes**  $\text{var} \notin (\text{vars } c)$   
**assumes**  $\text{length } xs' = \text{var}$   
**shows** *eval* (*substInfinitesimalLinear* *var*  $(-c) \ b \ a$ )  $(xs'@x\#xs) = \text{evalUni} (\text{substInfinitesimalLinearUni} \ B \ C \ a') \ x$   
 $\langle \text{proof} \rangle$

**lemma** *either-or*:  
**fixes**  $r :: \text{real}$   
**assumes**  $a: (\exists y' > r. \forall x \in \{r <..y'\}. (a \text{EvalUni } (\text{EqUni } (a, b, c)) \ x) \vee (a \text{EvalUni } (\text{LessUni } (a, b, c)) \ x))$   
**shows**  $(\exists y' > r. \forall x \in \{r <..y'\}. (a \text{EvalUni } (\text{EqUni } (a, b, c)) \ x)) \vee (\exists y' > r. \forall x \in \{r <..y'\}. (a \text{EvalUni } (\text{LessUni } (a, b, c)) \ x))$   
 $\langle \text{proof} \rangle$

**lemma** *infinitesimal-linear'-helper* :  
**assumes** *at-is*:  $At = \text{LessUni } p \vee At = \text{EqUni } p$   
**assumes**  $B \neq 0$   
**shows**  $((\exists y' :: \text{real} > -C/B. \forall x :: \text{real} \in \{-C/B <..y'\}. a \text{EvalUni } At \ x) = \text{evalUni} (\text{substInfinitesimalLinearUni } B \ C \ At) \ x)$   
 $\langle \text{proof} \rangle$

**lemma** *infinitesimal-linear'* :  
**assumes**  $B \neq 0$   
**shows**  $(\exists y' :: \text{real} > -C/B. \forall x :: \text{real} \in \{-C/B <..y'\}. a \text{EvalUni } At \ x) = \text{evalUni} (\text{substInfinitesimalLinearUni } B \ C \ At) \ x$   
 $\langle \text{proof} \rangle$

**fun** *quadraticSubUni* ::  $\text{real} \Rightarrow \text{real} \Rightarrow \text{real} \Rightarrow \text{real} \Rightarrow \text{atomUni} \Rightarrow \text{atomUni} \text{ fmUni}$   
**where**  
 $\text{quadraticSubUni } a \ b \ c \ d \ A = (\text{if } a \text{EvalUni } A \ ((a+b*\text{sqrt}(c))/d) \ \text{then } \text{TrueFUni} \ \text{else } \text{FalseFUni})$

**fun** *substInfinitesimalQuadraticUni* ::  $\text{real} \Rightarrow \text{real} \Rightarrow \text{real} \Rightarrow \text{real} \Rightarrow \text{atomUni} \Rightarrow$

*atomUni fmUni* **where**

*substInfinitesimalQuadraticUni* *a b c d* (*EqUni* *p*) = *allZero'* *p*|  
*substInfinitesimalQuadraticUni* *a b c d* (*LessUni* *p*) = *map-atomUni* (*quadraticSubUni*  
*a b c d*) (*convertDerivativeUni* *p*)|  
*substInfinitesimalQuadraticUni* *a b c d* (*LeqUni* *p*) = *OrUni*(*map-atomUni* (*quadraticSubUni*  
*a b c d*) (*convertDerivativeUni* *p*)) (*allZero'* *p*)|  
*substInfinitesimalQuadraticUni* *a b c d* (*NeqUni* *p*) = *negUni* (*allZero'* *p*)

**lemma** *weird* :

**fixes** *D::real*

**assumes** *dneq*: *D* ≠ (*0::real*)

**shows**

$((a'::real) * (((A::real) + (B::real) * \text{sqrt}(C::real)) / (D::real))^2 + (b'::real) * (A + B * \text{sqrt}(C)) / D + c' < 0 \vee$   
 $a' * ((A + B * \text{sqrt}(C)) / D)^2 + b' * (A + B * \text{sqrt}(C)) / D + (c'::real) = 0 \wedge$   
 $(b' + a' * (A + B * \text{sqrt}(C)) * 2 / D < 0 \vee$   
 $b' + a' * (A + B * \text{sqrt}(C)) * 2 / D = 0 \wedge 2 * a' < 0)) =$   
 $(a' * ((A + B * \text{sqrt}(C)) / D)^2 + b' * (A + B * \text{sqrt}(C)) / D + c' < 0 \vee$   
 $a' * ((A + B * \text{sqrt}(C)) / D)^2 + b' * (A + B * \text{sqrt}(C)) / D + c' = 0 \wedge$   
 $(2 * a' * (A + B * \text{sqrt}(C)) / D + b' < 0 \vee$   
 $2 * a' * (A + B * \text{sqrt}(C)) / D + b' = 0 \wedge a' < 0))$

*<proof>*

**lemma** *convert-substInfinitesimalQuadratic-less* :

**assumes** *convert-poly* *var p* (*xs'@x#xs*) = *Some* *p'*

**assumes** *insertion* (*nth-default* *0* (*xs'@x#xs*)) *a* = *A*

**assumes** *insertion* (*nth-default* *0* (*xs'@x#xs*)) *b* = *B*

**assumes** *insertion* (*nth-default* *0* (*xs'@x#xs*)) *c* = *C*

**assumes** *insertion* (*nth-default* *0* (*xs'@x#xs*)) *d* = *D*

**assumes** *D* ≠ *0*

**assumes** *0* ≤ *C*

**assumes** *var*∉(*vars* *a*)

**assumes** *var*∉(*vars* *b*)

**assumes** *var*∉(*vars* *c*)

**assumes** *var*∉(*vars* *d*)

**assumes** *length* *xs'* = *var*

**shows** *eval* (*quadratic-sub-fm* *var a b c d* (*convertDerivative* *var p*)) (*xs'@x#xs*)  
= *evalUni* (*map-atomUni* (*quadraticSubUni* *A B C D*) (*convertDerivativeUni* *p'*))  
*x*

*<proof>*

**lemma** *convert-substInfinitesimalQuadratic*:

**assumes** *convert-atom* *var At* (*xs'@x#xs*) = *Some*(*At'*)

**assumes** *insertion* (*nth-default* *0* (*xs'@x#xs*)) *a* = *A*

**assumes** *insertion* (*nth-default* *0* (*xs'@x#xs*)) *b* = *B*

**assumes** *insertion* (*nth-default* *0* (*xs'@x#xs*)) *c* = *C*

**assumes** *insertion* (*nth-default* *0* (*xs'@x#xs*)) *d* = *D*

**assumes** *D* ≠ *0*

```

assumes  $0 \leq C$ 
assumes  $\text{var}\notin(\text{vars } a)$ 
assumes  $\text{var}\notin(\text{vars } b)$ 
assumes  $\text{var}\notin(\text{vars } c)$ 
assumes  $\text{var}\notin(\text{vars } d)$ 
assumes  $\text{length } xs' = \text{var}$ 
shows  $\text{eval } (\text{substInfinesimalQuadratic } \text{var } a \ b \ c \ d \ \text{At}) \ (xs' @ x \# xs) = \text{evalUni}$ 
 $(\text{substInfinesimalQuadraticUni } A \ B \ C \ D \ \text{At}') \ x$ 
  <proof>

```

**lemma** *infinesimal-quad-helper:*

```

fixes  $A \ B \ C \ D :: \text{real}$ 
assumes  $\text{at-is: } \text{At} = \text{LessUni } p \ \vee \ \text{At} = \text{EqUni } p$ 
assumes  $D \neq 0$ 
assumes  $C \geq 0$ 
shows  $(\exists y' :: \text{real} > ((A+B * \text{sqrt}(C))/(D)). \forall x :: \text{real} \in \{((A+B * \text{sqrt}(C))/(D)) < .. y'\}.$ 
 $a.\text{EvalUni } \text{At } x)$ 
  =  $(\text{evalUni } (\text{substInfinesimalQuadraticUni } A \ B \ C \ D \ \text{At}) \ x)$ 
  <proof>

```

**lemma** *infinesimal-quad:*

```

fixes  $A \ B \ C \ D :: \text{real}$ 
assumes  $D \neq 0$ 
assumes  $C \geq 0$ 
shows  $(\exists y' :: \text{real} > ((A+B * \text{sqrt}(C))/(D)). \forall x :: \text{real} \in \{((A+B * \text{sqrt}(C))/(D)) < .. y'\}.$ 
 $a.\text{EvalUni } \text{At } x)$ 
  =  $(\text{evalUni } (\text{substInfinesimalQuadraticUni } A \ B \ C \ D \ \text{At}) \ x)$ 
  <proof>

```

**end**

## 9.4 Overall General VS Proofs

**theory** *DNFUni*

**imports** *QE InfinitesimalsUni*

**begin**

**fun** *DNFUni* ::  $\text{atomUni } \text{fmUni} \Rightarrow \text{atomUni } \text{list } \text{list}$  **where**

$\text{DNFUni } (\text{AtomUni } a) = [[a]]$

$\text{DNFUni } (\text{TrueFUni}) = [[]] \ |$

$\text{DNFUni } (\text{FalseFUni}) = []$

$\text{DNFUni } (\text{AndUni } A \ B) = [A' @ B'. A' \leftarrow \text{DNFUni } A, B' \leftarrow \text{DNFUni } B]$

$\text{DNFUni } (\text{OrUni } A \ B) = \text{DNFUni } A @ \text{DNFUni } B$

**lemma** *eval-DNFUni* :  $\text{evalUni } F \ x = \text{evalUni } (\text{list-disj-Uni}(\text{map } (\text{list-conj-Uni } o$   
 $(\text{map } \text{AtomUni})) \ (\text{DNFUni } F))) \ x$   
 <proof>

```

fun elimVarUni-atom :: atomUni list  $\Rightarrow$  atomUni  $\Rightarrow$  atomUni fmUni where
  elimVarUni-atom F (EqUni (a,b,c)) =
  (OrUni
    (AndUni
      (AndUni (AtomUni (EqUni (0,0,a))) (AtomUni (NegUni (0,0,b))))
      (list-conj-Uni (map (linearSubstitutionUni b c) F)))
      (AndUni (AtomUni (NegUni (0,0,a))) (AndUni (AtomUni (LeqUni (0,0,-(b2)+4*a*c)))
        (OrUni
          (list-conj-Uni (map (quadraticSubUni (-b) 1 (b2-4*a*c) (2*a)) F))
          (list-conj-Uni (map (quadraticSubUni (-b) (-1) (b2-4*a*c) (2*a)) F))
        )
      )
    )
  )
  |
  elimVarUni-atom F (LeqUni (a,b,c)) =
  (OrUni
    (AndUni
      (AndUni (AtomUni (EqUni (0,0,a))) (AtomUni (NegUni (0,0,b))))
      (list-conj-Uni (map (linearSubstitutionUni b c) F)))
      (AndUni (AtomUni (NegUni (0,0,a))) (AndUni (AtomUni (LeqUni (0,0,-(b2)+4*a*c)))
        (OrUni
          (list-conj-Uni (map (quadraticSubUni (-b) 1 (b2-4*a*c) (2*a)) F))
          (list-conj-Uni (map (quadraticSubUni (-b) (-1) (b2-4*a*c) (2*a)) F))
        )
      )
    )
  )
  |
  elimVarUni-atom F (LessUni (a,b,c)) =
  (OrUni
    (AndUni
      (AndUni (AtomUni (EqUni (0,0,a))) (AtomUni (NegUni (0,0,b))))
      (list-conj-Uni (map (substInfinitesimalLinearUni b c) F)))
      (AndUni (AtomUni (NegUni (0,0,a))) (AndUni (AtomUni (LeqUni (0,0,-(b2)+4*a*c)))
        (OrUni
          (list-conj-Uni (map (substInfinitesimalQuadraticUni (-b) 1 (b2-4*a*c)
            (2*a)) F))
          (list-conj-Uni (map (substInfinitesimalQuadraticUni (-b) (-1) (b2-4*a*c)
            (2*a)) F))
        )
      )
    )
  )
  |
  elimVarUni-atom F (NegUni (a,b,c)) =
  (OrUni
    (AndUni
      (AndUni (AtomUni (EqUni (0,0,a))) (AtomUni (NegUni (0,0,b))))
    )
  )

```



$(\forall (a,b,c) \in \text{set } c. (\exists x. \forall y < x. a*y^2 + b*y + c \leq 0)) \wedge$   
 $(\forall (a,b,c) \in \text{set } d. (\exists x. \forall y < x. a*y^2 + b*y + c \neq 0))$   
 $\langle \text{proof} \rangle$

**lemma** *set-split* :

**assumes** *separateAtoms*  $L = (eq, les, leq, neq)$

**shows**  $\text{set } L = \text{set } (\text{map } EqUni \text{ eq } @ \text{map } LessUni \text{ les } @ \text{map } LeqUni \text{ leq } @ \text{map } NeqUni \text{ neq})$

$\langle \text{proof} \rangle$

**lemma** *set-split'* : **assumes** *separateAtoms*  $L = (eq, les, leq, neq)$

**shows**  $\text{set } (\text{map } P \ L) = \text{set } (\text{map } (P \ o \ EqUni) \ \text{eq} \ @ \ \text{map } (P \ o \ LessUni) \ \text{les} \ @ \ \text{map } (P \ o \ LeqUni) \ \text{leq} \ @ \ \text{map } (P \ o \ NeqUni) \ \text{neq})$

$\langle \text{proof} \rangle$

**lemma** *split-elimVar* :

**assumes** *separateAtoms*  $L = (eq, les, leq, neq)$

**shows**  $(\exists l \in \text{set } L. \text{evalUni } (\text{elimVarUni-atom } L' \ l) \ x) =$

$((\exists (a',b',c') \in \text{set } eq. (\text{evalUni } (\text{elimVarUni-atom } L' \ (EqUni(a',b',c'))) \ x))$

$\vee (\exists (a',b',c') \in \text{set } les.$

$(\text{evalUni } (\text{elimVarUni-atom } L' \ (LessUni(a',b',c'))) \ x))$

$\vee (\exists (a',b',c') \in \text{set } leq.$

$(\text{evalUni } (\text{elimVarUni-atom } L' \ (LeqUni(a',b',c'))) \ x))$

$\vee (\exists (a',b',c') \in \text{set } neq.$

$(\text{evalUni } (\text{elimVarUni-atom } L' \ (NeqUni(a',b',c'))) \ x))$

$\langle \text{proof} \rangle$

**lemma** *split-elimvar* :

**assumes** *separateAtoms*  $L = (eq, les, leq, neq)$

**shows**  $\text{evalUni } (\text{elimVarUni-atom } L \ \text{At}) \ x = \text{evalUni } (\text{elimVarUni-atom } ((\text{map } EqUni \ \text{eq}) @ (\text{map } LessUni \ \text{les}) @ \text{map } LeqUni \ \text{leq} @ \text{map } NeqUni \ \text{neq}) \ \text{At}) \ x$

$\langle \text{proof} \rangle$

**lemma** *less* :

$((a' = 0 \wedge b' \neq 0) \wedge$

$(\forall (d, e, f) \in \text{set } a. \text{evalUni } (\text{substInfinesimalLinearUni } b' \ c' \ (EqUni \ (d, e, f))) \ x) \wedge$

$(\forall (d, e, f) \in \text{set } b. \text{evalUni } (\text{substInfinesimalLinearUni } b' \ c' \ (LessUni \ (d, e, f))) \ x) \wedge$

$(\forall (d, e, f) \in \text{set } c. \text{evalUni } (\text{substInfinesimalLinearUni } b' \ c' \ (LeqUni \ (d, e, f))) \ x) \wedge$

$(\forall (d, e, f) \in \text{set } d. \text{evalUni } (\text{substInfinesimalLinearUni } b' \ c' \ (NeqUni \ (d, e, f))) \ x) \vee$

$a' \neq 0 \wedge$

$-b'^2 + 4 * a' * c' \leq 0 \wedge$

$(\forall (d, e, f) \in \text{set } a.$

$$\begin{aligned}
& \text{evalUni} \\
& \quad (\text{substInfinitesimalQuadraticUni } (- b') 1 (b'^2 - 4 * a' * c') (2 * a') \\
& \quad \quad (\text{EqUni } (d, e, f))) \\
& \quad x) \wedge \\
& (\forall (d, e, f) \in \text{set } b. \\
& \quad \text{evalUni} \\
& \quad \quad (\text{substInfinitesimalQuadraticUni } (- b') 1 (b'^2 - 4 * a' * c') (2 * a') \\
& \quad \quad \quad (\text{LessUni } (d, e, f))) \\
& \quad x) \wedge \\
& (\forall (d, e, f) \in \text{set } c. \\
& \quad \text{evalUni} \\
& \quad \quad (\text{substInfinitesimalQuadraticUni } (- b') 1 (b'^2 - 4 * a' * c') (2 * a') \\
& \quad \quad \quad (\text{LeqUni } (d, e, f))) \\
& \quad x) \wedge \\
& (\forall (d, e, f) \in \text{set } d. \\
& \quad \text{evalUni} \\
& \quad \quad (\text{substInfinitesimalQuadraticUni } (- b') 1 (b'^2 - 4 * a' * c') (2 * a') \\
& \quad \quad \quad (\text{NeqUni } (d, e, f))) \\
& \quad x) \vee \\
& (\forall (d, e, f) \in \text{set } a. \\
& \quad \text{evalUni} \\
& \quad \quad (\text{substInfinitesimalQuadraticUni } (- b') (- 1) (b'^2 - 4 * a' * c') (2 * \\
& a') \\
& \quad \quad \quad (\text{EqUni } (d, e, f))) \\
& \quad x) \wedge \\
& (\forall (d, e, f) \in \text{set } b. \\
& \quad \text{evalUni} \\
& \quad \quad (\text{substInfinitesimalQuadraticUni } (- b') (- 1) (b'^2 - 4 * a' * c') (2 * \\
& a') \\
& \quad \quad \quad (\text{LessUni } (d, e, f))) \\
& \quad x) \wedge \\
& (\forall (d, e, f) \in \text{set } c. \\
& \quad \text{evalUni} \\
& \quad \quad (\text{substInfinitesimalQuadraticUni } (- b') (- 1) (b'^2 - 4 * a' * c') (2 * \\
& a') \\
& \quad \quad \quad (\text{LeqUni } (d, e, f))) \\
& \quad x) \wedge \\
& (\forall (d, e, f) \in \text{set } d. \\
& \quad \text{evalUni} \\
& \quad \quad (\text{substInfinitesimalQuadraticUni } (- b') (- 1) (b'^2 - 4 * a' * c') (2 * \\
& a') \\
& \quad \quad \quad (\text{NeqUni } (d, e, f))) \\
& \quad x))) = \\
& ((a' = 0 \wedge b' \neq 0) \wedge \\
& (\forall (d, e, f) \in \text{set } a. \\
& \quad (\exists y'::\text{real} > -c'/b'. \forall x'::\text{real} \in \{-c'/b' <..y'\}. a\text{EvalUni } (\text{EqUni } (d, e, f)) \\
& x)) \wedge \\
& (\forall (d, e, f) \in \text{set } b.
\end{aligned}$$

$$\begin{aligned}
& (\exists y'::\text{real} > -c'/b'. \forall x::\text{real} \in \{-c'/b' < ..y'\}. \text{aEvalUni} (\text{LessUni} (d, e, f))) \\
x)) \wedge & (\forall (d, e, f) \in \text{set } c. \\
& (\exists y'::\text{real} > -c'/b'. \forall x::\text{real} \in \{-c'/b' < ..y'\}. \text{aEvalUni} (\text{LeqUni} (d, e, f))) \\
x)) \wedge & (\forall (d, e, f) \in \text{set } d. \\
& (\exists y'::\text{real} > -c'/b'. \forall x::\text{real} \in \{-c'/b' < ..y'\}. \text{aEvalUni} (\text{NeqUni} (d, e, f))) \\
x)) \vee & a' \neq 0 \wedge \\
& -b'^2 + 4 * a' * c' \leq 0 \wedge \\
& ((\forall (d, e, f) \in \text{set } a. \\
& (\exists y' > (-b' + 1 * \text{sqrt} (b'^2 - 4 * a' * c')) / (2 * a')). \\
\forall x \in \{(-b' + 1 * \text{sqrt} (b'^2 - 4 * a' * c')) / (2 * a') < ..y'\}. \\
& \text{aEvalUni} (\text{EqUni} (d, e, f)) x)) \wedge \\
& (\forall (d, e, f) \in \text{set } b. \\
& (\exists y' > (-b' + 1 * \text{sqrt} (b'^2 - 4 * a' * c')) / (2 * a')). \\
\forall x \in \{(-b' + 1 * \text{sqrt} (b'^2 - 4 * a' * c')) / (2 * a') < ..y'\}. \\
& \text{aEvalUni} (\text{LessUni} (d, e, f)) x)) \wedge \\
& (\forall (d, e, f) \in \text{set } c. \\
& (\exists y' > (-b' + 1 * \text{sqrt} (b'^2 - 4 * a' * c')) / (2 * a')). \\
\forall x \in \{(-b' + 1 * \text{sqrt} (b'^2 - 4 * a' * c')) / (2 * a') < ..y'\}. \\
& \text{aEvalUni} (\text{LeqUni} (d, e, f)) x)) \wedge \\
& (\forall (d, e, f) \in \text{set } d. \\
& (\exists y' > (-b' + 1 * \text{sqrt} (b'^2 - 4 * a' * c')) / (2 * a')). \\
\forall x \in \{(-b' + 1 * \text{sqrt} (b'^2 - 4 * a' * c')) / (2 * a') < ..y'\}. \\
& \text{aEvalUni} (\text{NeqUni} (d, e, f)) x)) \vee \\
& (\forall (d, e, f) \in \text{set } a. \\
& (\exists y' > (-b' + -1 * \text{sqrt} (b'^2 - 4 * a' * c')) / (2 * a')). \\
\forall x \in \{(-b' + -1 * \text{sqrt} (b'^2 - 4 * a' * c')) / (2 * a') < ..y'\}. \\
& \text{aEvalUni} (\text{EqUni} (d, e, f)) x)) \wedge \\
& (\forall (d, e, f) \in \text{set } b. \\
& (\exists y' > (-b' + -1 * \text{sqrt} (b'^2 - 4 * a' * c')) / (2 * a')). \\
\forall x \in \{(-b' + -1 * \text{sqrt} (b'^2 - 4 * a' * c')) / (2 * a') < ..y'\}. \\
& \text{aEvalUni} (\text{LessUni} (d, e, f)) x)) \wedge \\
& (\forall (d, e, f) \in \text{set } c. \\
& (\exists y' > (-b' + -1 * \text{sqrt} (b'^2 - 4 * a' * c')) / (2 * a')). \\
\forall x \in \{(-b' + -1 * \text{sqrt} (b'^2 - 4 * a' * c')) / (2 * a') < ..y'\}. \\
& \text{aEvalUni} (\text{LeqUni} (d, e, f)) x)) \wedge \\
& (\forall (d, e, f) \in \text{set } d. \\
& (\exists y' > (-b' + -1 * \text{sqrt} (b'^2 - 4 * a' * c')) / (2 * a')). \\
\forall x \in \{(-b' + -1 * \text{sqrt} (b'^2 - 4 * a' * c')) / (2 * a') < ..y'\}. \\
& \text{aEvalUni} (\text{NeqUni} (d, e, f)) x)))) \\
\langle \text{proof} \rangle &
\end{aligned}$$

**lemma** *eq-inf* :  $(\forall (a, b, c) \in \text{set } (a::(\text{real} * \text{real} * \text{real}) \text{ list}). \exists x. \forall y < x. a * y^2 + b * y + c = 0) = (\forall (a, b, c) \in \text{set } a. a = 0 \wedge b = 0 \wedge c = 0)$   
 $\langle \text{proof} \rangle$

This is the main quantifier elimination lemma, in the simplified framework.

We are located directly underneath the most internal existential quantifier so  $F$  is completely free in quantifier and consists only of conjunction and disjunction. The variable we are evaluating on,  $x$ , is removed in the `generalVS_DNF` converted formula as expanding out the `evalUni` function determines that  $x$  doesn't play a role in the computation of it. It would be okay to replace the  $x$  in the second half with any variable, but it is simpler this way

This conversion is defined as a "veritcal" translation as we remain within the univariate framework in both sides of the expression

**lemma** `eval-generalVS''` :  $(\exists x. \text{evalUni } (\text{list-conj-Uni } (\text{map AtomUni } L)) x) = \text{evalUni } (\text{generalVS-DNF } L) x$

*<proof>*

**lemma** `forallx-substNegInf` :  $(\neg \text{evalUni } (\text{map-atomUni substNegInfinityUni } F) x) = (\forall x. \neg \text{evalUni } (\text{map-atomUni substNegInfinityUni } F) x)$

*<proof>*

**lemma** `linear-subst-map`:  $\text{evalUni } (\text{map-atomUni } (\text{linearSubstitutionUni } b \ c) \ F) \ x = \text{evalUni } F \ (-c/b)$

*<proof>*

**lemma** `quadratic-subst-map` :  $\text{evalUni } (\text{map-atomUni } (\text{quadraticSubUni } a \ b \ c \ d) \ F) \ x = \text{evalUni } F \ ((a+b*\text{sqrt}(c))/d)$

*<proof>*

**fun** `convert-atom-list` ::  $\text{nat} \Rightarrow \text{atom list} \Rightarrow \text{real list} \Rightarrow (\text{atomUni list}) \text{ option}$  **where**  
`convert-atom-list` var []  $xs = \text{Some } []$   
`convert-atom-list` var (a#as)  $xs =$  (  
`case` `convert-atom` var a  $xs$  of `Some`(a)  $\Rightarrow$   
 $(\text{case } \text{convert-atom-list} \text{ var } as \ xs \text{ of } \text{Some}(as) \Rightarrow \text{Some}(a\#as) \mid \text{None} \Rightarrow \text{None})$   
 $\mid \text{None} \Rightarrow \text{None}$   
 $)$   
 $)$

**lemma** `convert-atom-list-change` :

**assumes**  $\text{length } xs' = \text{var}$

**shows**  $\text{convert-atom-list} \text{ var } L \ (xs' \ @ \ x \ \# \ \Gamma) = \text{convert-atom-list} \text{ var } L \ (xs' \ @ \ x' \ \# \ \Gamma)$

*<proof>*

**lemma** `negInf-convert` :

**assumes** *convert-atom-list* var  $L$  ( $xs' @ x \# xs$ ) = *Some*  $L'$   
**assumes** *length*  $xs' = \text{var}$   
**shows**  $(\forall f \in \text{set } L. \text{eval } (\text{substNegInfinity } \text{var } f) (xs' @ x \# xs))$   
 $= (\forall f \in \text{set } L'. \text{evalUni } (\text{substNegInfinityUni } f) x)$   
 $\langle \text{proof} \rangle$

**lemma** *elimVar-atom-single* :

**assumes** *convert-atom* var  $A$  ( $xs' @ x \# xs$ ) = *Some*  $A'$   
**assumes** *convert-atom-list* var  $L2$  ( $xs' @ x \# xs$ ) = *Some*  $L2'$   
**assumes** *length*  $xs' = \text{var}$   
**shows**  $\text{eval } (\text{elimVar } \text{var } L2 \ [] \ A) (xs' @ x \# xs) = \text{evalUni } (\text{elimVarUni-atom}$   
 $L2' \ A') x$   
 $\langle \text{proof} \rangle$

**lemma** *convert-list* :

**assumes** *convert-atom-list* var  $L$  ( $xs' @ x \# xs$ ) = *Some*  $L'$   
**assumes**  $l \in \text{set}(L)$   
**shows**  $\exists l' \in \text{set } L'. \text{convert-atom } \text{var } l (xs' @ x \# xs) = \text{Some } l'$   
 $\langle \text{proof} \rangle$

**lemma** *convert-list2* :

**assumes** *convert-atom-list* var  $L$  ( $xs' @ x \# xs$ ) = *Some*  $L'$   
**assumes**  $l' \in \text{set}(L')$   
**shows**  $\exists l \in \text{set } L. \text{convert-atom } \text{var } l (xs' @ x \# xs) = \text{Some } l'$   
 $\langle \text{proof} \rangle$

**lemma** *elimVar-atom-convert* :

**assumes** *convert-atom-list* var  $L$  ( $xs' @ x \# xs$ ) = *Some*  $L'$   
**assumes** *convert-atom-list* var  $L2$  ( $xs' @ x \# xs$ ) = *Some*  $L2'$   
**assumes** *length*  $xs' = \text{var}$   
**shows**  $(\exists f \in \text{set } L. \text{eval } (\text{elimVar } \text{var } L2 \ [] \ f) (xs' @ x \# xs))$   
 $= (\exists f \in \text{set } L'. \text{evalUni } (\text{elimVarUni-atom } L2' \ f) x)$   
 $\langle \text{proof} \rangle$

**lemma** *eval-convert* :

**assumes** *convert-atom-list* var  $L$  ( $xs' @ x \# xs$ ) = *Some*  $L'$   
**assumes** *length*  $xs' = \text{var}$   
**shows**  $(\forall f \in \text{set } L. \text{aEval } f (xs' @ x \# xs)) = (\forall f \in \text{set } L'. \text{aEvalUni } f x)$   
 $\langle \text{proof} \rangle$

**lemma** *all-degree-2-convert* :

**assumes** *all-degree-2* var  $L$   
**shows**  $\exists L'. \text{convert-atom-list } \text{var } L \ xs = \text{Some } L'$   
 $\langle \text{proof} \rangle$

**lemma** *gen-ge-eval* :

**assumes** *hlength* : *length*  $xs = \text{var}$   
**shows**  $(\exists x. (\text{eval } (\text{list-conj } ((\text{map } \text{Atom } L) @ F)) (xs @ (x\#\Gamma)))) = (\exists x. (\text{eval}$   
 $(\text{gen-ge } \text{var } L \ F) (xs @ (x\#\Gamma))))$   
 $\langle \text{proof} \rangle$

**lemma** *freeIn-elimVar* : *freeIn var (elimVar var L F A)*  
 ⟨*proof*⟩

**lemma** *freeInDisj*: *freeIn var (list-disj (list-conj (map (substNegInfinity var) L) # map (elimVar var L []) L))*  
 ⟨*proof*⟩

**lemma** *gen-qe-eval'* :  
**assumes** *all-degree-2 var L*  
**assumes** *length xs' = var*  
**shows**  $(\exists x. (eval (list-conj (map Atom L)) (xs'@x#\Gamma))) = (\forall x. (eval (gen-qe var L []) (xs'@x#\Gamma)))$   
*freeIn var (gen-qe var L [])*  
 ⟨*proof*⟩

**lemma** *gen-qe-eval''* :  
**assumes** *all-degree-2 var L*  
**assumes** *length xs' = var*  
**shows**  $(\exists x. (eval (list-conj (map Atom L)) (xs'@x#\Gamma))) = (\forall x. (eval (list-disj (list-conj (map (substNegInfinity var) L) # map (elimVar var L []) L)) (xs'@x#\Gamma)))$   
 ⟨*proof*⟩

**end**

## 10 QE Algorithm Proofs

### 10.1 DNF

**theory** *DNF*  
**imports** *VSAlgos*  
**begin**

**theorem** *dnf-eval* :  
 $(\exists (al,fl)\in set (dnf \varphi).$   
 $(\forall a\in set al. aEval a xs)$   
 $\wedge (\forall f\in set fl. eval f xs))$   
 $= eval \varphi xs$   
 ⟨*proof*⟩

**theorem** *dnf-modified-eval* :  
 $(\exists (al,fl,n)\in set (dnf-modified \varphi).$   
 $(\exists L. (length L = n$

$\wedge (\forall a \in \text{set } al. aEval a (L @ xs))$   
 $\wedge (\forall f \in \text{set } fl. eval f (L @ xs)))) = eval \varphi xs$   
 <proof>  
 end

## 10.2 Recursive QE

**theory** VSQuad

**imports** EqualityVS GeneralVSProofs Reindex OptimizationProofs DNF  
**begin**

**lemma** *existN-eval* :  $\forall xs. eval (ExN n \varphi) xs = (\exists L. (length L = n \wedge eval \varphi (L @ xs)))$   
 <proof>

**lemma** *boundedFlipNegQuantifier* :  $(\neg(\forall x \in A. \neg P x)) = (\exists x \in A. P x)$   
 <proof>

**theorem** *QE-dnf'-eval*:

**assumes** *steph* :  $\bigwedge amount F \Gamma.$   
 $(\exists xs. (length xs = amount \wedge eval (list-disj (map(\lambda(L,F,n). ExN n (list-conj (map fm.Atom L @ F))) F)) (xs @ \Gamma))) = (eval (step amount F) \Gamma)$   
**assumes** *opt* :  $\bigwedge xs F . eval (opt F) xs = eval F xs$   
**shows**  $eval (QE-dnf' opt step \varphi) xs = eval \varphi xs$   
 <proof>

**theorem** *QE-dnf-eval*:

**assumes** *steph* :  $\bigwedge var amount new L F \Gamma.$   
 $amount \leq var + 1 \implies$   
 $(\exists xs. (length xs = var + 1 \wedge eval (list-conj (map fm.Atom L @ F)) (xs @ \Gamma)))$   
 $= (\exists xs. (length xs = var + 1 \wedge eval (step amount var L F) (xs @ \Gamma)))$   
**assumes** *opt* :  $\bigwedge xs F . eval (opt F) xs = eval F xs$   
**shows**  $eval (QE-dnf opt step \varphi) xs = eval \varphi xs$   
 <proof>

**lemma** *opt*:  $eval ((push-forall \circ nnf \circ unpower 0 \circ groupQuantifiers \circ clearQuantifiers) F) L = eval F L$   
 <proof>

**lemma** *opt'*:  $eval ((push-forall ( nnf ( unpower 0 ( groupQuantifiers ( clearQuantifiers F)))))) L = eval F L$   
 <proof>

**lemma** *opt-no-group*:  $eval ((push-forall \circ nnf \circ unpower\ 0 \circ clearQuantifiers)\ F)$   
 $L = eval\ F\ L$   
 ⟨proof⟩

**lemma** *repeatAmountOfQuantifiers-helper-eval* :  
**assumes**  $\bigwedge xs\ F. eval\ F\ xs = eval\ (step\ F)\ xs$   
**shows**  $eval\ F\ xs = eval\ (repeatAmountOfQuantifiers-helper\ step\ n\ F)\ xs$   
 ⟨proof⟩

**lemma** *repeatAmountOfQuantifiers-eval* :  
**assumes**  $\bigwedge xs\ F. eval\ F\ xs = eval\ (step\ F)\ xs$   
**shows**  $eval\ F\ xs = eval\ (repeatAmountOfQuantifiers\ step\ F)\ xs$   
 ⟨proof⟩

**end**

### 10.3 Heuristic Proofs

**theory** *HeuristicProofs*  
**imports** *VSQuad Heuristic OptimizationProofs*  
**begin**

**lemma** *the-real-step-augment*:  
**assumes**  $steph : \bigwedge xs\ var\ L\ F\ \Gamma. length\ xs = var \implies (\exists x. eval\ (list-conj\ (map\ fm.Atom\ L\ @\ F))\ (xs\ @\ x\ \#\ \Gamma)) = (\exists x. eval\ (step\ var\ L\ F)\ (xs\ @\ x\ \#\ \Gamma))$   
**shows**  $(\exists xs. (length\ xs = amount \wedge eval\ (list-disj\ (map\ (\lambda(L,F,n). ExN\ n\ (list-conj\ (map\ fm.Atom\ L\ @\ F)))\ F))\ (xs\ @\ \Gamma))) = (eval\ (the-real-step-augment\ step\ amount\ F)\ \Gamma)$   
 ⟨proof⟩

**lemma** *step-converter* :  
**assumes**  $steph : \bigwedge xs\ var\ L\ F\ \Gamma. length\ xs = var \implies (\exists x. eval\ (list-conj\ (map\ fm.Atom\ L\ @\ F))\ (xs\ @\ x\ \#\ \Gamma)) = (\exists x. eval\ (step\ var\ L\ F)\ (xs\ @\ x\ \#\ \Gamma))$   
**shows**  $\bigwedge var\ L\ F\ \Gamma. (\exists xs. length\ xs = var + 1 \wedge eval\ (list-conj\ (map\ fm.Atom\ L\ @\ F))\ (xs\ @\ \Gamma)) =$   
 $(\exists xs. (length\ xs = (var + 1)) \wedge eval\ (step\ var\ L\ F)\ (xs\ @\ \Gamma))$   
 ⟨proof⟩

**lemma** *step-augmenter-eval* :  
**assumes**  $steph : \bigwedge xs\ var\ L\ F\ \Gamma. length\ xs = var \implies (\exists x. eval\ (list-conj\ (map\ fm.Atom\ L\ @\ F))\ (xs\ @\ x\ \#\ \Gamma)) = (\exists x. eval\ (step\ var\ L\ F)\ (xs\ @\ x\ \#\ \Gamma))$   
**assumes** *heuristic*:  $\bigwedge n\ var\ L\ F. heuristic\ n\ L\ F = var \implies var \leq n$   
**shows**  $\bigwedge var\ amount\ L\ F\ \Gamma.$   
 $amount \leq var + 1 \implies$   
 $(\exists xs. length\ xs = var + 1 \wedge eval\ (list-conj\ (map\ fm.Atom\ L\ @\ F))\ (xs\ @\ \Gamma))$   
 =

$(\exists xs. (\text{length } xs = \text{var} + 1)) \wedge \text{eval } (\text{step-augment step heuristic amount var } L F) (xs @ \Gamma)$   
 <proof>

**lemma** *qe-eq-repeat-eval-augment* :  $\text{amount} \leq \text{var} + 1 \implies$   
 $(\exists xs. (\text{length } xs = \text{var} + 1) \wedge \text{eval } (\text{list-conj } (\text{map fm.Atom } L @ F)) (xs @ \Gamma)) =$   
 $(\exists xs. (\text{length } xs = \text{var} + 1) \wedge \text{eval } (\text{step-augment qe-eq-repeat IdentityHeuristic amount var } L F) (xs @ \Gamma))$   
 <proof>

**lemma** *qe-eq-repeat-eval'* :  
 $(\exists xs. (\text{length } xs = \text{var} + 1) \wedge \text{eval } (\text{list-conj } (\text{map fm.Atom } L @ F)) (xs @ \Gamma)) =$   
 $(\exists xs. (\text{length } xs = \text{var} + 1) \wedge \text{eval } (\text{qe-eq-repeat var } L F) (xs @ \Gamma))$   
 <proof>

**lemma** *gen-qe-eval-augment* :  $\text{amount} \leq \text{var} + 1 \implies$   
 $(\exists xs. (\text{length } xs = \text{var} + 1) \wedge \text{eval } (\text{list-conj } (\text{map fm.Atom } L @ F)) (xs @ \Gamma)) =$   
 $(\exists xs. (\text{length } xs = \text{var} + 1) \wedge \text{eval } (\text{step-augment gen-qe IdentityHeuristic amount var } L F) (xs @ \Gamma))$   
 <proof>

**lemma** *gen-qe-eval'* :  
 $(\exists xs. (\text{length } xs = \text{var} + 1) \wedge \text{eval } (\text{list-conj } (\text{map fm.Atom } L @ F)) (xs @ \Gamma)) =$   
 $(\exists xs. (\text{length } xs = \text{var} + 1) \wedge \text{eval } (\text{gen-qe var } L F) (xs @ \Gamma))$   
 <proof>

**lemma** *luckyFind-eval-augment* :  $\text{amount} \leq \text{var} + 1 \implies$   
 $(\exists xs. (\text{length } xs = \text{var} + 1) \wedge \text{eval } (\text{list-conj } (\text{map fm.Atom } L @ F)) (xs @ \Gamma)) =$   
 $(\exists xs. (\text{length } xs = \text{var} + 1) \wedge \text{eval } (\text{step-augment luckyFind' IdentityHeuristic amount var } L F) (xs @ \Gamma))$   
 <proof>

**lemma** *luckyFind-eval'* :  
 $(\exists xs. (\text{length } xs = \text{var} + 1) \wedge \text{eval } (\text{list-conj } (\text{map fm.Atom } L @ F)) (xs @ \Gamma)) =$   
 $(\exists xs. (\text{length } xs = \text{var} + 1) \wedge \text{eval } (\text{luckyFind' var } L F) (xs @ \Gamma))$   
 <proof>

**lemma** *luckiestFind-eval'* :  
 $(\exists xs. (\text{length } xs = \text{var} + 1) \wedge \text{eval } (\text{list-conj } (\text{map fm.Atom } L @ F)) (xs @ \Gamma)) =$   
 $(\exists xs. (\text{length } xs = \text{var} + 1) \wedge \text{eval } (\text{luckiestFind var } L F) (xs @ \Gamma))$   
 <proof>

**lemma** *sortedListMember* : *sorted-list-of-fset*  $b = \text{var} \# \text{list} \implies \text{fmember } \text{var } b$   
 ⟨*proof*⟩

**lemma** *rangeHeuristic* :  
**assumes** *heuristicPicker*  $n L F = \text{Some } (\text{var}, \text{step})$   
**shows**  $\text{var} \leq n$   
 ⟨*proof*⟩

**lemma** *pickedOneOfThem* :  
**assumes** *heuristicPicker*  $n L F = \text{Some } (\text{var}, \text{step})$   
**shows**  $\text{step} = \text{qe-eq-repeat} \vee \text{step} = \text{gen-qe} \vee \text{step} = \text{luckyFind}'$   
 ⟨*proof*⟩

**lemma** *superPicker-eval* :  
 $\text{amount} \leq \text{var} + 1 \implies (\exists xs. \text{length } xs = \text{var} + 1 \wedge \text{eval } (\text{list-conj } (\text{map } \text{fm.Atom } L @ F)) (xs @ \Gamma)) =$   
 $(\exists xs. (\text{length } xs = (\text{var} + 1)) \wedge \text{eval } (\text{superPicker } \text{amount } \text{var } L F) (xs @ \Gamma))$   
 ⟨*proof*⟩

**lemma** *brownHueristic-less-than*: *brownsHeuristic*  $n L F = \text{var} \implies \text{var} \leq n$   
 ⟨*proof*⟩  
**end**

## 10.4 Top-Level Algorithm Proofs

**theory** *ExportProofs*  
**imports** *HeuristicProofs Exports*

*HOL.String HOL-Library.Code-Target-Int HOL-Library.Code-Target-Nat PrettyPrinting Show.Show-Real*  
**begin**

**theorem** *eval (Unpower f) L = eval f L* ⟨*proof*⟩

**theorem** *VSLuckiest*:  $\forall xs. \text{eval } (\text{VSLuckiest } \varphi) xs = \text{eval } \varphi xs$   
 ⟨*proof*⟩

**theorem** *VSLuckiestBlocks* :  $\forall xs. \text{eval } (\text{VSLuckiestBlocks } \varphi) xs = \text{eval } \varphi xs$   
 ⟨*proof*⟩

**theorem** *VSEquality* :  $\forall xs. \text{eval } (\text{VSEquality } \varphi) xs = \text{eval } \varphi xs$   
 ⟨*proof*⟩

**theorem** *VSEqualityBlocks* :  $\forall xs. \text{eval } (\text{VSEqualityBlocks } \varphi) xs = \text{eval } \varphi xs$

*<proof>*

**theorem** *VSGeneralBlocks* :  $\forall xs. eval (VSGeneralBlocks \varphi) xs = eval \varphi xs$   
*<proof>*

**theorem** *VSLuckyBlocks* :  $\forall xs. eval (VSLuckyBlocks \varphi) xs = eval \varphi xs$   
*<proof>*

**theorem** *VSLEGBlocks* :  $\forall xs. eval (VSLEGBlocks \varphi) xs = eval \varphi xs$   
*<proof>*

**theorem** *VSEqualityBlocksLimited* :  $\forall xs. eval (VSEqualityBlocksLimited \varphi) xs = eval \varphi xs$   
*<proof>*

**theorem** *VSEquality-3-times* :  $\forall xs. eval (VSEquality-3-times \varphi) xs = eval \varphi xs$   
*<proof>*

**theorem** *VSGeneral* :  $\forall xs. eval (VSGeneral \varphi) xs = eval \varphi xs$   
*<proof>*

**theorem** *VSGeneralBlocksLimited* :  $\forall xs. eval (VSGeneralBlocksLimited \varphi) xs = eval \varphi xs$   
*<proof>*

**theorem** *VS Browns* :  $\forall xs. eval (VS Browns \varphi) xs = eval \varphi xs$   
*<proof>*

**theorem** *VSGeneral-3-times* :  $\forall xs. eval (VSGeneral-3-times \varphi) xs = eval \varphi xs$   
*<proof>*

**theorem** *VSLucky* :  $\forall xs. eval (VSLucky \varphi) xs = eval \varphi xs$   
*<proof>*

**theorem** *VSLuckyBlocksLimited* :  $\forall xs. eval (VSLuckyBlocksLimited \varphi) xs = eval \varphi xs$   
*<proof>*

**theorem** *VSLEG* :  $\forall xs. eval (VSLEG \varphi) xs = eval \varphi xs$   
*<proof>*

**theorem** *VSHeuristic* :  $\forall xs. eval (VSHeuristic \varphi) xs = eval \varphi xs$   
*<proof>*

**theorem** *VSLuckiestRepeat* :  $\forall xs. eval (VSLuckiestRepeat \varphi) xs = eval \varphi xs$   
*<proof>*

### export-code

```
print-mpoly
VSGeneral VSEquality VSLucky VSLEG VSLuckiest
VSGeneralBlocksLimited VSEqualityBlocksLimited VSLuckyBlocksLimited
VSGeneralBlocks VSEqualityBlocks VSLuckyBlocks VSLEGBlocks VSLuckiest-
Blocks
QE-dnf
gen-qe qe-eq-repeat
simpfm push-forall nnf Unpower
is-quantifier-free is-solved
add mult C V pow minus
Eq Or is-quantifier-free

real-of-int real-mult real-div real-plus real-minus

VSGeneral-3-times VSEquality-3-times VSHuristic VSLuckiestRepeat VSBrowns
in SML module-name VS
```

end

## References

- [1] A. Chaieb. *Automated methods for formal proofs in simple arithmetics and algebra*. PhD thesis, Technische Universität München, 2008.
- [2] T. Nipkow. Linear quantifier elimination. *J. Autom. Reason.*, 45(2):189–212, 2010.
- [3] A. Platzer. *Logical Foundations of Cyber-Physical Systems*. Springer, Cham, 2018.
- [4] V. Weispfenning. The complexity of linear problems in fields. *J. Symb. Comput.*, 5(1/2):3–27, 1988.
- [5] V. Weispfenning. Quantifier elimination for real algebra – the quadratic case and beyond. *Appl. Algebra Eng. Commun. Comput.*, 8(2):85–101, 1997.