Verified Quadratic Virtual Substitution for Real Arithmetic

Matias Scharager, Katherine Cordwell, Stefan Mitsch and André Platzer

May 26, 2024

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

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 $\mathrm{FOL}_{\mathbb{R}}$ 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 $\mathrm{FOL}_{\mathbb{R}}$ 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* ⇒ *real* ⇒ *int*
  where *sign p x* ≡ (*if poly p x* = *0 then 0 else* (*if poly p x* > *0 then 1 else* −*1*))

definition *sign-num*:: *real* ⇒ *int*
  where *sign-num x* ≡ (*if x* = *0 then 0 else* (*if x* > *0 then 1 else* −*1*))

definition *root-list*:: *real Polynomial.poly* ⇒ *real set*
  where *root-list p* ≡ ({(*x*::*real*). *poly p x* = *0*}::*real set*)

definition *root-set*:: (*real* × *real* × *real*) *set* ⇒ *real set*
  where *root-set les* ≡ ({(*x*::*real*). (∃ (*a*, *b*, *c*) ∈ *les*. *a*∗*x*^*2* + *b*∗*x* + *c* = *0*)}::*real set*)

definition *sorted-root-list-set*:: (*real* × *real* × *real*) *set* ⇒ *real list*
  where *sorted-root-list-set p* ≡ *sorted-list-of-set* (*root-set p*)

definition *nonzero-root-set*:: (*real* × *real* × *real*) *set* ⇒ *real set*
  where *nonzero-root-set les* ≡ ({(*x*::*real*). (∃ (*a*, *b*, *c*) ∈ *les*. (*a* ≠ *0* ∨ *b* ≠ *0*) ∧ *a*∗*x*^*2* + *b*∗*x* + *c* = *0*)}::*real set*)

definition *sorted-nonzero-root-list-set*:: (*real* × *real* × *real*) *set* ⇒ *real list*
  where *sorted-nonzero-root-list-set p* ≡ *sorted-list-of-set* (*nonzero-root-set p*)

lemma *sorted-list-prop*:
  fixes *l*::*real list*

fixes x::*real*
assumes *sorted*: *sorted* l
assumes *lengt*: *length* $l > 0$
assumes *xgt*: $x > l$! 0
assumes *xlt*: $x \leq l$! (*length* $l - 1$)
shows $\exists\, n$. $(n+1) <$ (*length* l) $\wedge$ $x \geq l$!n $\wedge$ $x \leq l$! $(n + 1)$
⟨*proof*⟩

2.2 Quadratic polynomial properties

lemma *quadratic-poly-eval*:
fixes a b c::*real*
fixes x::*real*
shows *poly* [:c, b, a:] $x = a*x$^2 $+ b*x + c$
⟨*proof*⟩

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

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

lemma *nonzero-root-set-finite*:
assumes *fin*: *finite les*
shows *finite* (*nonzero-root-set les*)
⟨*proof*⟩

lemma *discriminant-lemma*:
fixes a b c r::*real*
assumes *aneq*: $a \neq 0$
assumes *beq*: $b = 2 * a * r$
assumes *root*: $a * r$^$2 - 2 * a * r*r + c = 0$
shows $\forall\, x.\ a * x^2 + b * x + c = 0 \longleftrightarrow x = -r$
⟨*proof*⟩

lemma *changes-sign*:
fixes p:: *real Polynomial.poly*
shows $\forall\, x$::*real*. $\forall$ y::*real*. ((*sign* p $x \neq$ *sign* p y $\wedge$ $x < y$) $\longrightarrow$ ($\exists\, c \in$ (*root-list* p). $x \leq c \wedge c \leq y$))
⟨*proof*⟩

lemma *changes-sign-var*:

fixes $a\ b\ c\ x\ y$:: *real*
shows $((\mathit{sign\text{-}num}\ (a*x\hat{~}2 + b*x + c) \neq \mathit{sign\text{-}num}\ (a*y\hat{~}2 + b*y + c)\ \wedge x < y) \Longrightarrow (\exists\, q.\ (a*q\hat{~}2 + b*q + c = 0 \wedge x \leq q \wedge q \leq y)))$
⟨*proof*⟩

2.3 Continuity Properties

lemma *continuity-lem-eq0*:
fixes p::*real*
shows $r < p \Longrightarrow \forall\, x \in \{r <..p\}.\ a * x^2 + b * x + c = 0 \Longrightarrow (a = 0 \wedge b = 0 \wedge c = 0)$
⟨*proof*⟩

lemma *continuity-lem-lt0*:
fixes r:: *real*
fixes $a\ b\ c$:: *real*
shows $\mathit{poly}\ [:c,\ b,\ a:]\ r < 0 \Longrightarrow$
$\exists\, y' > r.\ \forall\, x \in \{r <..y'\}.\ \mathit{poly}\ [:c,\ b,\ a:]\ x < 0$
⟨*proof*⟩

lemma *continuity-lem-gt0*:
fixes r:: *real*
fixes $a\ b\ c$:: *real*
shows $\mathit{poly}\ [:c,\ b,\ a:]\ r > 0 \Longrightarrow$
$\exists\, y' > r.\ \forall\, x \in \{r <..y'\}.\ \mathit{poly}\ [:c,\ b,\ a:]\ x > 0$
⟨*proof*⟩

lemma *continuity-lem-lt0-expanded*:
fixes r:: *real*
fixes $a\ b\ c$:: *real*
shows $a*r\hat{~}2 + b*r + c < 0 \Longrightarrow$
$\exists\, y' > r.\ \forall\, x \in \{r <..y'\}.\ a*x\hat{~}2 + b*x + c < 0$
⟨*proof*⟩

lemma *continuity-lem-gt0-expanded*:
fixes r:: *real*
fixes $a\ b\ c$:: *real*
fixes k::*real*
assumes *kgt*: $k > r$
shows $a*r\hat{~}2 + b*r + c > 0 \Longrightarrow$
$\exists\, x \in \{r <..k\}.\ a*x\hat{~}2 + b*x + c > 0$
⟨*proof*⟩

2.4 Negative Infinity (Limit) Properties

lemma *ysq-dom-y*:
fixes b:: *real*
fixes c:: *real*
shows $\exists\, (w::\mathit{real}).\ \forall\, (y::\ \mathit{real}).\ (y < w \longrightarrow y\hat{~}2 > b*y)$
⟨*proof*⟩

lemma *ysq-dom-y-plus-coeff*:
fixes *b*:: *real*
fixes *c*:: *real*
shows $\exists (w::real).\ \forall (y::real).\ (y < w \longrightarrow y\hat{}2 > b*y + c)$
⟨*proof*⟩

lemma *ysq-dom-y-plus-coeff-2*:
fixes *b*:: *real*
fixes *c*:: *real*
shows $\exists (w::real).\ \forall (y::real).\ (y > w \longrightarrow y\hat{}2 > b*y + c)$
⟨*proof*⟩

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

lemma *pos-lc-dom-quad*:
fixes *a*:: *real*
fixes *b*:: *real*
fixes *c*:: *real*
assumes *alt*: $a > 0$
shows $\exists (w::real).\ \forall (y::real).\ (y > w \longrightarrow a*y\hat{}2 + b*y + c > 0)$
⟨*proof*⟩

2.5 Infinitesimal and Continuity Properties

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

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 $(\exists x.\ (\forall (a, b, c) \in set\ les.\ a * x^2 + b * x + c < 0))$
⟨*proof*⟩

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

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

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

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

2.6 Some Casework

lemma *quadratic-shape1a*:
fixes *a b c x y*::*real*
assumes *agt*: $a > 0$
assumes *xyroots*: $x < y \wedge a*x\hat{}2 + b*x + c = 0 \wedge a*y\hat{}2 + b*y + c = 0$
shows $\bigwedge z.\ (z > x \wedge z < y \Longrightarrow a*z\hat{}2 + b*z + c < 0)$
⟨*proof*⟩

lemma *quadratic-shape1b*:
fixes *a b c x y*::*real*
assumes *agt*: $a > 0$
assumes *xy-roots*: $x < y \wedge a*x\hat{}2 + b*x + c = 0 \wedge a*y\hat{}2 + b*y + c = 0$
shows $\bigwedge z.\ (z > y \Longrightarrow a*z\hat{}2 + b*z + c > 0)$
⟨*proof*⟩

lemma *quadratic-shape2a*:
fixes *a b c x y*::*real*
assumes $a < 0$
assumes $x < y \wedge a*x\hat{}2 + b*x + c = 0 \wedge a*y\hat{}2 + b*y + c = 0$
shows $\bigwedge z.\ (z > x \wedge z < y \Longrightarrow a*z\hat{}2 + b*z + c > 0)$
⟨*proof*⟩

lemma *quadratic-shape2b*:
fixes *a b c x y*::*real*
assumes $a < 0$
assumes $x < y \wedge a*x\hat{}2 + b*x + c = 0 \wedge a*y\hat{}2 + b*y + c = 0$
shows $\bigwedge z.\ (z > y \Longrightarrow a*z\hat{}2 + b*z + c < 0)$
⟨*proof*⟩

lemma *case-d1*:
fixes *a b c r*::*real*
shows $b < 2 * a * r \Longrightarrow$
$a * r\hat{}2 - b*r + c = 0 \Longrightarrow$

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

lemma *case-d4*:
fixes *a b c r::real*
shows $\bigwedge y'.\ b \neq 2 * a * r \Longrightarrow$
$\neg\ b < 2 * a * r \Longrightarrow$
$a * r\hat{\ }2 - b * r + c = 0 \Longrightarrow$
$-r < y' \Longrightarrow \exists\, x\in\{-r<..y'\}.\ \neg\ a * x^2 + b * x + c < 0$
⟨*proof*⟩

lemma *one-root-a-lt0*:
fixes *a b c r y'::real*
assumes *alt*: $a < 0$
assumes *beq*: $b = 2 * a * r$
assumes *root*: $a * r\hat{\ }2 - 2*a*r*r + c = 0$
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 *beq*: $b = 2 * a * r$
assumes *root*: $a * r\hat{\ }2 - 2*a*r*r + c = 0$
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 $a*r\hat{\ }2 + b*r + c > 0 \Longrightarrow$
$\exists\, x\in\{r<..k\}.\ a*x\hat{\ }2 + b*x + c \geq 0$
⟨*proof*⟩

lemma *continuity-lem-lt0-expanded-var*:
fixes *r:: real*
fixes *a b c:: real*
shows $a*r\hat{\ }2 + b*r + c < 0 \Longrightarrow$
$\exists\, y'>\ r.\ \forall\, x\in\{r<..y'\}.\ a*x\hat{\ }2 + b*x + c \leq 0$
⟨*proof*⟩

lemma *nonzcoeffs*:
fixes *a b c r::real*
shows $a\neq 0 \vee b\neq 0 \vee c\neq 0 \Longrightarrow \exists\, y'>r.\ \forall\, x\in\{r<..y'\}.\ a * x^2 + b * x + c \neq 0$

⟨*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 $(\forall ((d::real), (e::real), (f::real))\in set\ leq.\ \exists y'>r.\ \forall x\in\{r<..y'\}.\ d * x^2 + e * x + f \leq 0)$
shows $(\exists x.\ (\forall (a, b, c)\in set\ leq.\ a * x^2 + b * x + c \leq 0))$
⟨*proof*⟩

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

2.8 Setting up and Helper Lemmas

2.8.1 The les_qe lemma

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

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

$\langle proof \rangle$

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

2.8.2 equiv_lemma

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

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

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

$(\forall (d, e, f) \in set\ les.$
$d * ((- b' + - 1 * sqrt\ (b'^2 - 4 * a' * c')) / (2 * a'))^2 +$
$e * ((- b' + - 1 * sqrt\ (b'^2 - 4 * a' * c')) / (2 * a')) +$
f
$< 0))) \lor$
$(\forall (d, e, f) \in set\ eq.\ d = 0 \land e = 0 \land f = 0) \land$
$(\exists x.\ \forall (a, b, c) \in set\ les.\ a * x^2 + b * x + c < 0))$
$\langle proof \rangle$

2.8.3 The eq_qe lemma

lemma *eq-qe-forwards*:
shows $(\exists x.\ (\forall (a, b, c) \in set\ eq.\ a * x^2 + b * x + c = 0) \land$
$(\forall (a, b, c) \in set\ les.\ a * x^2 + b * x + c < 0)) \Longrightarrow$
$((\exists (a', b', c') \in set\ eq.$
$(a' = 0 \land b' \neq 0) \land$
$(\forall (d, e, f) \in set\ eq.\ d * (- c' / b')^2 + e * (- c' / b') + f = 0) \land$
$(\forall (d, e, f) \in set\ les.\ d * (- c' / b')^2 + e * (- c' / b') + f < 0) \lor$
$a' \neq 0 \land$
$- b'^2 + 4 * a' * c' \leq 0 \land$
$((\forall (d, e, f) \in set\ eq.$
$d * ((- b' + 1 * sqrt\ (b'^2 - 4 * a' * c')) / (2 * a'))^2 +$
$e * ((- b' + 1 * sqrt\ (b'^2 - 4 * a' * c')) / (2 * a')) +$
$f =$
$0) \land$
$(\forall (d, e, f) \in set\ les.$
$d * ((- b' + 1 * sqrt\ (b'^2 - 4 * a' * c')) / (2 * a'))^2 +$
$e * ((- b' + 1 * sqrt\ (b'^2 - 4 * a' * c')) / (2 * a')) +$
f
$< 0) \lor$
$(\forall (d, e, f) \in set\ eq.$
$d * ((- b' + - 1 * sqrt\ (b'^2 - 4 * a' * c')) / (2 * a'))^2 +$
$e * ((- b' + - 1 * sqrt\ (b'^2 - 4 * a' * c')) / (2 * a')) +$
$f =$
$0) \land$
$(\forall (d, e, f) \in set\ les.$
$d * ((- b' + - 1 * sqrt\ (b'^2 - 4 * a' * c')) / (2 * a'))^2 +$
$e * ((- b' + - 1 * sqrt\ (b'^2 - 4 * a' * c')) / (2 * a')) +$
f
$< 0))) \lor$
$(\forall (d, e, f) \in set\ eq.\ d = 0 \land e = 0 \land f = 0) \land$
$(\exists x.\ \forall (a, b, c) \in set\ les.\ a * x^2 + b * x + c < 0))$
$\langle proof \rangle$

lemma *eq-qe-backwards*: $((\exists (a', b', c') \in set\ eq.$
$(a' = 0 \land b' \neq 0) \land$
$(\forall (d, e, f) \in set\ eq.\ d * (- c' / b')^2 + e * (- c' / b') + f = 0) \land$
$(\forall (d, e, f) \in set\ les.\ d * (- c' / b')^2 + e * (- c' / b') + f < 0) \lor$
$a' \neq 0 \land$

$- b'^2 + 4 * a' * c' \le 0 \wedge$
$((\forall (d, e, f) \in set\ eq.$
$d * ((- b' + 1 * sqrt\ (b'^2 - 4 * a' * c')) / (2 * a'))^2 +$
$e * ((- b' + 1 * sqrt\ (b'^2 - 4 * a' * c')) / (2 * a')) +$
$f =$
$0) \wedge$
$(\forall (d, e, f) \in set\ les.$
$d * ((- b' + 1 * sqrt\ (b'^2 - 4 * a' * c')) / (2 * a'))^2 +$
$e * ((- b' + 1 * sqrt\ (b'^2 - 4 * a' * c')) / (2 * a')) +$
f
$< 0) \vee$
$(\forall (d, e, f) \in set\ eq.$
$d * ((- b' + - 1 * sqrt\ (b'^2 - 4 * a' * c')) / (2 * a'))^2 +$
$e * ((- b' + - 1 * sqrt\ (b'^2 - 4 * a' * c')) / (2 * a')) +$
$f =$
$0) \wedge$
$(\forall (d, e, f) \in set\ les.$
$d * ((- b' + - 1 * sqrt\ (b'^2 - 4 * a' * c')) / (2 * a'))^2 +$
$e * ((- b' + - 1 * sqrt\ (b'^2 - 4 * a' * c')) / (2 * a')) +$
f
$< 0))) \vee$
$(\forall (d, e, f) \in set\ eq.\ d = 0 \wedge e = 0 \wedge f = 0) \wedge$
$(\exists x.\ \forall (a, b, c) \in set\ les.\ a * x^2 + b * x + c < 0)) \Longrightarrow$
$(\exists x.\ ((\forall (a, b, c) \in set\ eq.\ a * x^2 + b * x + c = 0) \wedge$
$(\forall (a, b, c) \in set\ les.\ a * x^2 + b * x + c < 0)))$

⟨*proof*⟩

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

e ∗ ((− b′ + − 1 ∗ sqrt (b'^2 − 4 ∗ a′ ∗ c′)) / (2 ∗ a′)) +
f =
0) ∧
(∀ (d, e, f)∈set les.
d ∗ ((− b′ + − 1 ∗ sqrt (b'^2 − 4 ∗ a′ ∗ c′)) / (2 ∗ a′))^2 +
e ∗ ((− b′ + − 1 ∗ sqrt (b'^2 − 4 ∗ a′ ∗ c′)) / (2 ∗ a′)) +
f
< 0))) ∨
(∀ (d, e, f)∈set eq. d = 0 ∧ e = 0 ∧ f = 0) ∧
(∃ x. ∀ (a, b, c)∈set les. a ∗ x^2 + b ∗ x + c < 0))
⟨*proof*⟩

2.8.4 The qe_forwards lemma

lemma *qe-forwards-helper-gen*:
fixes *r*:: *real*
assumes *f8*: ¬(∃ ((a′::real), (b′::real), (c′::real))∈set c.
((a′≠ 0 ∨ b′≠ 0) ∧ a′∗r^2 + b′∗r + c′ = 0) ∧
((∀ (d, e, f)∈set a. d ∗ r^2 + e ∗ r + f = 0) ∧
(∀ (d, e, f)∈set b. d ∗ r^2 + e ∗ r + f < 0) ∧
(∀ (d, e, f)∈set c. d ∗ r^2 + e ∗ r + f ≤ 0) ∧
(∀ (d, e, f)∈set d. d ∗ r^2 + e ∗ r + f ≠ 0)))
assumes *alleqset*: ∀ x. (∀ (d, e, f)∈set a. d ∗ x^2 + e ∗ x + f = 0)
assumes *f5*: ¬(∃ (a′, b′, c′)∈set b.
(a′ = 0 ∧ b′ ≠ 0) ∧
(∀ (d, e, f)∈set a.
∃ y′>− c′ / b′. ∀ x∈{− c′ / b′<..y′}. d ∗ x^2 + e ∗ x + f = 0) ∧
(∀ (d, e, f)∈set b.
∃ y′>− c′ / b′. ∀ x∈{− c′ / b′<..y′}. d ∗ x^2 + e ∗ x + f < 0) ∧
(∀ (d, e, f)∈set c.
∃ y′>− c′ / b′. ∀ x∈{− c′ / b′<..y′}. d ∗ x^2 + e ∗ x + f ≤ 0) ∧
(∀ (d, e, f)∈set d.
∃ y′>− c′ / b′. ∀ x∈{− c′ / b′<..y′}. d ∗ x^2 + e ∗ x + f ≠ 0))
assumes *f6*: ¬ (∃ (a′, b′, c′)∈set b. a′ ≠ 0 ∧
− b'^2 + 4 ∗ a′ ∗ c′ ≤ 0 ∧
((∀ (d, e, f)∈set a.
∃ y′>(− b′ + 1 ∗ sqrt (b'^2 − 4 ∗ a′ ∗ c′)) / (2 ∗ a′).
∀ x∈{(− b′ + 1 ∗ sqrt (b'^2 − 4 ∗ a′ ∗ c′)) / (2 ∗ a′)<..y′}.
d ∗ x^2 + e ∗ x + f = 0) ∧
(∀ (d, e, f)∈set b.
∃ y′>(− b′ + 1 ∗ sqrt (b'^2 − 4 ∗ a′ ∗ c′)) / (2 ∗ a′).
∀ x∈{(− b′ + 1 ∗ sqrt (b'^2 − 4 ∗ a′ ∗ c′)) / (2 ∗ a′)<..y′}.
d ∗ x^2 + e ∗ x + f < 0) ∧
(∀ (d, e, f)∈set c.
∃ y′>(− b′ + 1 ∗ sqrt (b'^2 − 4 ∗ a′ ∗ c′)) / (2 ∗ a′).
∀ x∈{(− b′ + 1 ∗ sqrt (b'^2 − 4 ∗ a′ ∗ c′)) / (2 ∗ a′)<..y′}.
d ∗ x^2 + e ∗ x + f ≤ 0) ∧
(∀ (d, e, f)∈set d.
∃ y′>(− b′ + 1 ∗ sqrt (b'^2 − 4 ∗ a′ ∗ c′)) / (2 ∗ a′).

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

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

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

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

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

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

lemma *qe-forwards-helper-lin*:
assumes *alleqset*: $\forall x. (\forall (d, e, f)\in set\ a.\ d * x\hat{\ }2 + e * x + f = 0)$
assumes *f5*: $\neg(\exists (a', b', c')\in set\ b.$
$(a' = 0 \wedge b' \neq 0) \wedge$
$(\forall (d, e, f)\in set\ a.$
$\exists y'>- c' / b'. \forall x\in\{- c' / b'<..y'\}. d * x^2 + e * x + f = 0) \wedge$
$(\forall (d, e, f)\in set\ b.$
$\exists y'>- c' / b'. \forall x\in\{- c' / b'<..y'\}. d * x^2 + e * x + f < 0) \wedge$
$(\forall (d, e, f)\in set\ c.$
$\exists y'>- c' / b'. \forall x\in\{- c' / b'<..y'\}. d * x^2 + e * x + f \leq 0) \wedge$
$(\forall (d, e, f)\in set\ d.$
$\exists y'>- c' / b'. \forall x\in\{- c' / b'<..y'\}. d * x^2 + e * x + f \neq 0))$
assumes *f6*: $\neg (\exists (a', b', c')\in set\ b.\ a' \neq 0 \wedge$
$- b'^2 + 4 * a' * c' \leq 0 \wedge$
$((\forall (d, e, f)\in set\ a.$
$\exists y'>(- b' + 1 * sqrt\ (b'^2 - 4 * a' * c')) / (2 * a').$
$\forall x\in\{(- b' + 1 * sqrt\ (b'^2 - 4 * a' * c')) / (2 * a')<..y'\}.$
$d * x^2 + e * x + f = 0) \wedge$

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

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

assumes $f8$: $\neg(\exists (a',\ b',\ c'){\in}set\ c.\ (a' = 0 \wedge b' \neq 0) \wedge$
$(\forall (d,\ e,\ f){\in}set\ a.\ d * (-\ c'\ /\ b')^2 + e * (-\ c'\ /\ b') + f = 0) \wedge$
$(\forall (d,\ e,\ f){\in}set\ b.\ d * (-\ c'\ /\ b')^2 + e * (-\ c'\ /\ b') + f < 0) \wedge$
$(\forall (d,\ e,\ f){\in}set\ c.\ d * (-\ c'\ /\ b')^2 + e * (-\ c'\ /\ b') + f \leq 0) \wedge$
$(\forall (d,\ e,\ f){\in}set\ d.\ d * (-\ c'\ /\ b')^2 + e * (-\ c'\ /\ b') + f \neq 0))$

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

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

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

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

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

⟨*proof*⟩

lemma *qe-forwards-helper*:
assumes *alleqset*: $\forall x. (\forall (d, e, f) \in set\ a.\ d * x\hat{}2 + e * x + f = 0)$
assumes *f1*: $\neg((\forall (a, b, c) \in set\ a.\ a = 0 \wedge b = 0 \wedge c = 0) \wedge$

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

assumes $f5$: $\neg(\exists (a', b', c') \in set\ b.$
$(a' = 0 \wedge b' \neq 0) \wedge$
$(\forall (d, e, f) \in set\ a.$
$\exists y'> - c' / b'.\ \forall x \in \{- c' / b' <..y'\}.\ d * x^2 + e * x + f = 0) \wedge$
$(\forall (d, e, f) \in set\ b.$
$\exists y'> - c' / b'.\ \forall x \in \{- c' / b' <..y'\}.\ d * x^2 + e * x + f < 0) \wedge$
$(\forall (d, e, f) \in set\ c.$
$\exists y'> - c' / b'.\ \forall x \in \{- c' / b' <..y'\}.\ d * x^2 + e * x + f \leq 0) \wedge$
$(\forall (d, e, f) \in set\ d.$
$\exists y'> - c' / b'.\ \forall x \in \{- c' / b' <..y'\}.\ d * x^2 + e * x + f \neq 0))$

assumes $f6$: $\neg (\exists (a', b', c') \in set\ b.\ a' \neq 0 \wedge$
$- b'^2 + 4 * a' * c' \leq 0 \wedge$
$((\forall (d, e, f) \in set\ a.$
$\exists y'>(- b' + 1 * sqrt\ (b'^2 - 4 * a' * c')) / (2 * a').$
$\forall x \in \{(- b' + 1 * sqrt\ (b'^2 - 4 * a' * c')) / (2 * a') <..y'\}.$
$d * x^2 + e * x + f = 0) \wedge$
$(\forall (d, e, f) \in set\ b.$
$\exists y'>(- b' + 1 * sqrt\ (b'^2 - 4 * a' * c')) / (2 * a').$
$\forall x \in \{(- b' + 1 * sqrt\ (b'^2 - 4 * a' * c')) / (2 * a') <..y'\}.$
$d * x^2 + e * x + f < 0) \wedge$
$(\forall (d, e, f) \in set\ c.$
$\exists y'>(- b' + 1 * sqrt\ (b'^2 - 4 * a' * c')) / (2 * a').$
$\forall x \in \{(- b' + 1 * sqrt\ (b'^2 - 4 * a' * c')) / (2 * a') <..y'\}.$
$d * x^2 + e * x + f \leq 0) \wedge$
$(\forall (d, e, f) \in set\ d.$
$\exists y'>(- b' + 1 * sqrt\ (b'^2 - 4 * a' * c')) / (2 * a').$
$\forall x \in \{(- b' + 1 * sqrt\ (b'^2 - 4 * a' * c')) / (2 * a') <..y'\}.$
$d * x^2 + e * x + f \neq 0)))$

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

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

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

assumes *f13*: $\neg(\exists (a', b', c') \in set\ c.$
$a' \neq 0 \wedge$
$- b'^2 + 4 * a' * c' \leq 0 \wedge$
$((\forall (d, e, f) \in set\ a.$
$d * ((- b' + 1 * sqrt\ (b'^2 - 4 * a' * c')) / (2 * a'))^2 +$
$e * ((- b' + 1 * sqrt\ (b'^2 - 4 * a' * c')) / (2 * a')) +$
$f = 0) \wedge$
$(\forall (d, e, f) \in set\ b.$
$d * ((- b' + 1 * sqrt\ (b'^2 - 4 * a' * c')) / (2 * a'))^2 +$
$e * ((- b' + 1 * sqrt\ (b'^2 - 4 * a' * c')) / (2 * a')) +$
$f < 0) \wedge$
$(\forall (d, e, f) \in set\ c.$
$d * ((- b' + 1 * sqrt\ (b'^2 - 4 * a' * c')) / (2 * a'))^2 +$
$e * ((- b' + 1 * sqrt\ (b'^2 - 4 * a' * c')) / (2 * a')) +$
$f \leq 0) \wedge$
$(\forall (d, e, f) \in set\ d.$
$d * ((- b' + 1 * sqrt\ (b'^2 - 4 * a' * c')) / (2 * a'))^2 +$
$e * ((- b' + 1 * sqrt\ (b'^2 - 4 * a' * c')) / (2 * a')) +$
$f \neq 0)))$

assumes *f9*: $\neg(\exists (a', b', c') \in set\ c.\ \ a' \neq 0 \wedge$
$- b'^2 + 4 * a' * c' \leq 0 \wedge$
$(\forall (d, e, f) \in set\ a.$
$d * ((- b' + - 1 * sqrt\ (b'^2 - 4 * a' * c')) / (2 * a'))^2 +$
$e * ((- b' + - 1 * sqrt\ (b'^2 - 4 * a' * c')) / (2 * a')) +$
$f = 0) \wedge$
$(\forall (d, e, f) \in set\ b.$
$d * ((- b' + - 1 * sqrt\ (b'^2 - 4 * a' * c')) / (2 * a'))^2 +$
$e * ((- b' + - 1 * sqrt\ (b'^2 - 4 * a' * c')) / (2 * a')) +$
$f < 0) \wedge$
$(\forall (d, e, f) \in set\ c.$
$d * ((- b' + - 1 * sqrt\ (b'^2 - 4 * a' * c')) / (2 * a'))^2 +$
$e * ((- b' + - 1 * sqrt\ (b'^2 - 4 * a' * c')) / (2 * a')) +$
$f \leq 0) \wedge$
$(\forall (d, e, f) \in set\ d.$
$d * ((- b' + - 1 * sqrt\ (b'^2 - 4 * a' * c')) / (2 * a'))^2 +$
$e * ((- b' + - 1 * sqrt\ (b'^2 - 4 * a' * c')) / (2 * a')) +$
$f \neq 0))$

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

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

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

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

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

⟨*proof*⟩

lemma *qe-forwards*:
assumes $(\exists x. (\forall (a, b, c)\in set\ a.\ a * x^2 + b * x + c = 0) \wedge$
$\quad (\forall (a, b, c)\in set\ b.\ a * x^2 + b * x + c < 0) \wedge$
$\quad (\forall (a, b, c)\in set\ c.\ a * x^2 + b * x + c \leq 0) \wedge$

$(\forall (a, b, c) \in set\ d.\ a * x^2 + b * x + c \neq 0))$

shows $((\forall (a, b, c) \in set\ a.\ a = 0 \wedge b = 0 \wedge c = 0) \wedge$

$(\forall (a, b, c) \in set\ b.\ \exists x.\ \forall y<x.\ a * y^2 + b * y + c < 0) \wedge$

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

$(\forall (a, b, c) \in set\ d.\ \exists x.\ \forall y<x.\ a * y^2 + b * y + c \neq 0) \vee$

$(\exists (a', b', c') \in set\ a.$

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

$(\forall (d, e, f) \in set\ a.\ d * (- c' / b')^2 + e * (- c' / b') + f = 0) \wedge$

$(\forall (d, e, f) \in set\ b.\ d * (- c' / b')^2 + e * (- c' / b') + f < 0) \wedge$

$(\forall (d, e, f) \in set\ c.\ d * (- c' / b')^2 + e * (- c' / b') + f \leq 0) \wedge$

$(\forall (d, e, f) \in set\ d.\ d * (- c' / b')^2 + e * (- c' / b') + f \neq 0) \vee$

$a' \neq 0 \wedge$

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

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

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

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

$f =$

$0) \wedge$

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

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

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

f

$< 0) \wedge$

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

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

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

f

$\leq 0) \wedge$

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

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

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

$f \neq$

$0) \vee$

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

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

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

$f =$

$0) \wedge$

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

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

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

f

$< 0) \wedge$

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

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

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

f

$\leq 0) \wedge$

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

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

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

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

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

lemma *h1b-helper-les*:
$(\forall ((a{::}real), (b{::}real), (c{::}real)) \in set\ les.\ \exists x.\ \forall y{<}x.\ a * y^2 + b * y + c < 0) \Longrightarrow (\exists y. \forall x{<}y.\ (\forall (a, b, c) \in set\ les.\ a * x^2 + b * x + c < 0))$
⟨*proof*⟩

lemma *h1b-helper-leq*:
$(\forall ((a{::}real), (b{::}real), (c{::}real)) \in set\ leq.\ \exists x.\ \forall y{<}x.\ a * y^2 + b * y + c \leq 0) \Longrightarrow (\exists y. \forall x{<}y.\ (\forall (a, b, c) \in set\ leq.\ a * x^2 + b * x + c \leq 0))$
⟨*proof*⟩

lemma *h1b-helper-neq*:
$(\forall ((a{::}real), (b{::}real), (c{::}real)) \in set\ neq.\ \exists x.\ \forall y{<}x.\ a * y^2 + b * y + c \neq 0) \Longrightarrow (\exists y. \forall x{<}y.\ (\forall (a, b, c) \in set\ neq.\ a * x^2 + b * x + c \neq 0))$
⟨*proof*⟩

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

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

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

2.8.6 The qe_backwards lemma

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

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

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

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

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

2.9 General QE lemmas

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

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

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

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

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

$\langle proof \rangle$

fun *eq-fun* :: *real* ⇒ *real* ⇒ *real* ⇒ (*real*∗*real*∗*real*) *list* ⇒ (*real*∗*real*∗*real*) *list* ⇒ (*real*∗*real*∗*real*) *list* ⇒ (*real*∗*real*∗*real*) *list* ⇒ *bool* **where**

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

```
        case a of
        (d, e, f) ⇒
          d ∗ ((− b′ + − 1 ∗ sqrt (b′² − 4 ∗ a′ ∗ c′)) / (2 ∗ a′))² +
          e ∗ ((− b′ + − 1 ∗ sqrt (b′² − 4 ∗ a′ ∗ c′)) / (2 ∗ a′)) +
          f =
          0) ∧
    (∀ a∈set les.
        case a of
        (d, e, f) ⇒
          d ∗ ((− b′ + − 1 ∗ sqrt (b′² − 4 ∗ a′ ∗ c′)) / (2 ∗ a′))² +
          e ∗ ((− b′ + − 1 ∗ sqrt (b′² − 4 ∗ a′ ∗ c′)) / (2 ∗ a′)) +
          f
          < 0) ∧
    (∀ a∈set leq.
        case a of
        (d, e, f) ⇒
          d ∗ ((− b′ + − 1 ∗ sqrt (b′² − 4 ∗ a′ ∗ c′)) / (2 ∗ a′))² +
          e ∗ ((− b′ + − 1 ∗ sqrt (b′² − 4 ∗ a′ ∗ c′)) / (2 ∗ a′)) +
          f
          ≤ 0) ∧
    (∀ a∈set neq.
        case a of
        (d, e, f) ⇒
          d ∗ ((− b′ + − 1 ∗ sqrt (b′² − 4 ∗ a′ ∗ c′)) / (2 ∗ a′))² +
          e ∗ ((− b′ + − 1 ∗ sqrt (b′² − 4 ∗ a′ ∗ c′)) / (2 ∗ a′)) +
          f ≠
          0)))

fun les-fun :: real ⇒ real ⇒ real ⇒ (real∗real∗real) list ⇒ (real∗real∗real) list ⇒
(real∗real∗real) list ⇒ (real∗real∗real) list ⇒ bool where
  les-fun a′ b′ c′ eq les leq neq = ((a′ = 0 ∧ b′ ≠ 0) ∧
      (∀ (d, e, f)∈set eq.
          ∃ y′>− c′ / b′. ∀ x∈{− c′ / b′<..y′}. d ∗ x² + e ∗ x + f = 0) ∧
      (∀ (d, e, f)∈set les.
          ∃ y′>− c′ / b′. ∀ x∈{− c′ / b′<..y′}. d ∗ x² + e ∗ x + f < 0) ∧
      (∀ (d, e, f)∈set leq.
          ∃ y′>− c′ / b′. ∀ x∈{− c′ / b′<..y′}. d ∗ x² + e ∗ x + f ≤ 0) ∧
      (∀ (d, e, f)∈set neq.
          ∃ y′>− c′ / b′. ∀ x∈{− c′ / b′<..y′}. d ∗ x² + e ∗ x + f ≠ 0) ∨
     a′ ≠ 0 ∧
     − b′² + 4 ∗ a′ ∗ c′ ≤ 0 ∧
     ((∀ (d, e, f)∈set eq.
          ∃ y′>(− b′ + 1 ∗ sqrt (b′² − 4 ∗ a′ ∗ c′)) / (2 ∗ a′).
             ∀ x∈{(− b′ + 1 ∗ sqrt (b′² − 4 ∗ a′ ∗ c′)) / (2 ∗ a′)<..y′}.
                d ∗ x² + e ∗ x + f = 0) ∧
      (∀ (d, e, f)∈set les.
          ∃ y′>(− b′ + 1 ∗ sqrt (b′² − 4 ∗ a′ ∗ c′)) / (2 ∗ a′).
             ∀ x∈{(− b′ + 1 ∗ sqrt (b′² − 4 ∗ a′ ∗ c′)) / (2 ∗ a′)<..y′}.
                d ∗ x² + e ∗ x + f < 0) ∧
```

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

lemma *general-qe'* :

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

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

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

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

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

assumes $A = \bigcup (roots\ `\ (set\ eq))$
assumes $B = \bigcup (roots\ `\ (set\ les))$
assumes $C = \bigcup (roots\ `\ (set\ leq))$
assumes $D = \bigcup (roots\ `\ (set\ neq))$

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

lemma *general-qe''* :

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

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

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

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

assumes $A = \bigcup (roots\ `\ ((X\ (=))))$
assumes $B = \bigcup (roots\ `\ ((X\ (<))))$
assumes $C = \bigcup (roots\ `\ ((X\ (\leq))))$
assumes $D = \bigcup (roots\ `\ ((X\ (\neq))))$

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

theorem *general-qe* :

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

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

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

0)

assumes F_{inf} = (∀ *rel*∈*R*. ∀ (*a*,*b*,*c*)∈(*Atoms rel*). ∃ *x*. ∀ *y*<*x*. *rel* (*a*∗*y*^{2}+*b*∗*y*+*c*) *0*)

assumes *roots* = (λ(*a*,*b*,*c*).
if a=*0* ∧ *b*≠*0 then* {−*c*/*b*} *else*
if a≠*0* ∧ *b*^{2}−*4*∗*a*∗*c*≥*0 then* {(−*b*+*sqrt*(*b*^{2}−*4*∗*a*∗*c*))/(*2*∗*a*)}∪{(−*b*−*sqrt*(*b*^{2}−*4*∗*a*∗*c*))/(*2*∗*a*)} *else* {})

shows (∃ *x*. *F*(*x*)) =
(F_{inf} ∨
(∃ *r*∈⋃ (*roots* ‘ (*Atoms* (=) ∪ *Atoms* (≤))). *F r*) ∨
(∃ *r*∈⋃ (*roots* ‘ (*Atoms* (<) ∪ *Atoms* (≠))). *Fε r*))
⟨*proof*⟩

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* ⇒ (*nat* ⇒$_0$ *nat*) ⇒ ′*a* **is**
lookup ⟨*proof*⟩

lemma *coeff-code*[*code*]: *coeff* = *coeff-code*
⟨*proof*⟩

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

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

lemma *plus-monom-zero*[*simp*]: *p* + *MPoly-Type.monom m 0* = *p*
⟨*proof*⟩

lift-definition *monomials*::′*a*::*zero mpoly* ⇒ (*nat* ⇒$_0$ *nat*) *set* **is**
Poly-Mapping.keys::((*nat*⇒$_0$*nat*) ⇒$_0$ ′*a*) ⇒ *- set* ⟨*proof*⟩

lemma *mpoly-induct* [*case-names monom sum*]:— TODO: overwrites ⟦⋀*m a*. *?P* (*monom m a*); ⋀*p1 p2 m a*. ⟦*?P p1*; *?P p2*; *p2* = *monom m a*; *m* ∉ *keys* (*mapping-of p1*)⟧ ⟹ *?P* (*p1* + *p2*)⟧ ⟹ *?P ?p*
assumes *monom*:⋀*m a*. *P* (*MPoly-Type.monom m a*)
and *sum*:(⋀*p1 p2 m a*. *P p1* ⟹ *P p2* ⟹ *p2* = (*MPoly-Type.monom m a*)

$\Longrightarrow$ *m* $\notin$ *monomials p1*
$\Longrightarrow$ *a* $\neq$ *0* $\Longrightarrow$ *P* (*p1*+*p2*))
  shows *P p* ⟨*proof*⟩

value *monomials* ((*Var 0* + *Const* (*3*::*int*) ∗ *Var 1*)^*2*)

lemma *Sum-any-lookup-times-eq*:
 ($\sum$ *k*. ((*lookup* (*x*::′*a*⇒$_0$(′*b*::*comm-semiring-1*)) (*k*::′*a*)) ∗ ((*f*:: ′*a*⇒(′*b*::*comm-semiring-1*)) *k*))) = ($\sum$ *k*∈*keys* *x*. (*lookup x* (*k*::′*a*)) ∗ (*f k*))
  ⟨*proof*⟩

lemma *Prod-any-power-lookup-eq*:
  ($\prod$ *k*::′*a*. *f k* ^ *lookup* (*x*::′*a*⇒$_0$*nat*) *k*) = ($\prod$ *k*∈*keys x*. *f k* ^ *lookup x k*)
  ⟨*proof*⟩

lemma *insertion-monom*: *insertion i* (*monom m a*) = *a* ∗ ($\prod$ *k*∈*keys m*. *i k* ^ *lookup m k*)
  ⟨*proof*⟩

lemma *monomials-monom*[*simp*]: *monomials* (*monom m a*) = (*if a* = *0 then* {} *else* {*m*})
  ⟨*proof*⟩

lemma *finite-monomials*[*simp*]: *finite* (*monomials m*)
  ⟨*proof*⟩

lemma *monomials-add-disjoint*:
  monomials (*a* + *b*) = *monomials a* ∪ *monomials b* **if** *monomials a* ∩ *monomials b* = {}
  ⟨*proof*⟩

lemma *coeff-monom*[*simp*]: *coeff* (*monom m a*) *m* = *a*
  ⟨*proof*⟩

lemma *coeff-not-in-monomials*[*simp*]: *coeff m x* = *0* **if** *x* ∉ *monomials m*
  ⟨*proof*⟩

code-thms *insertion*

lemma *insertion-code*[*code*]: *insertion i mp* =
   ($\sum$ *m*∈*monomials mp*. (*coeff mp m*) ∗ ($\prod$ *k*∈*keys m*. *i k* ^ *lookup m k*))
⟨*proof*⟩

code-thms *insertion*

value *insertion* (*nth* [*1*, *2.3*]) ((*Var 0* + *Const* (*3*::*int*) ∗ *Var 1*)^*2*)

lift-definition *isolate-variable-sparse*::*'a*::*comm-monoid-add mpoly* ⇒ *nat* ⇒ *nat* ⇒ *'a mpoly* **is**
λ(*mp*::*'a mpoly*) *x d. sum*
(λ*m. monomial* (*coeff mp m*) (*Poly-Mapping.update x 0 m*))
{*m* ∈ *monomials mp. lookup m x* = *d*} ⟨*proof*⟩

lemma *Poly-Mapping-update-code*[*code*]: *Poly-Mapping.update a b* (*Pm-fmap fm*) = *Pm-fmap* (*fmupd a b fm*)
⟨*proof*⟩

lemma *monom-zero* [*simp*] : *monom m 0* = *0*
⟨*proof*⟩

lemma *remove-key-code*[*code*]: *remove-key v* (*Pm-fmap fm*) = *Pm-fmap* (*fmdrop v fm*)
⟨*proof*⟩
lemma *extract-var-code*[*code*]:
extract-var p v =
(∑ *m*∈*monomials p. monom* (*remove-key v m*) (*monom* (*Poly-Mapping.single v* (*lookup m v*)) (*coeff p m*)))
⟨*proof*⟩
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* = ⋃ (*keys* ‘ *monomials p*)
⟨*proof*⟩

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

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

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

fun *get-if-const* :: *real mpoly* ⇒ *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*) **try**
⟨*proof*⟩

definition *derivative* :: *nat* ⇒ *real mpoly* ⇒ *real mpoly* **where**
derivative x p = (∑ *i*∈{*0..degree p x*−*1*}. *isolate-variable-sparse p x* (*i*+*1*) ∗ (*Var x*)^*i* ∗ (*Const* (*i*+*1*)))

get_coeffs *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* ⇒ *real mpoly* ⇒ *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 *ExecutiblePolyProps*
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* ⟷ *lookup c i* ≥ *p* **for** *p*::*nat*
⟨*proof*⟩

lemma *update-minus-monomial*: *Poly-Mapping.update k i (m − monomial i k) = Poly-Mapping.update k i m*
⟨*proof*⟩

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

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

lemma *coeff-eq-zero-iff*: *MPoly-Type.coeff c p = 0 ⟷ p ∉ monomials c*
⟨*proof*⟩

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

lemma *monomials-and-monoms*:
shows *(k ∈ monomials m) = (∃ (a::nat). a ≠ 0 ∧ (monomials (MPoly-Type.monom k a)) ⊆ monomials m)*
⟨*proof*⟩

lemma *mult-monomials-dir-one*:
shows *monomials (p∗q) ⊆ {a+b | a b . a ∈ monomials p ∧ b ∈ monomials q}*
⟨*proof*⟩

lemma *monom-eq-zero-iff*[*simp*]: *MPoly-Type.monom a b = 0 ⟷ b = 0*
⟨*proof*⟩

lemma *update-eq-plus-monomial*:
v ≥ lookup m k ⟹ Poly-Mapping.update k v m = m + monomial (v − lookup m k) k
for *v n::nat*
⟨*proof*⟩

lemma *coeff-monom-mult'*:
MPoly-Type.coeff ((MPoly-Type.monom m' a) ∗ q) (m'm) = a ∗ MPoly-Type.coeff q (m'm − m')
if ∗: *m'm = m' + (m'm − m')*
⟨*proof*⟩

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

lemma *in-monomials-iff*: *x ∈ monomials m ⟷ MPoly-Type.coeff m x ≠ 0*
⟨*proof*⟩

lemma *monomials-monom-mult*: *monomials (MPoly-Type.monom mon a ∗ q) = (if a = 0 then {} else (+) mon ‘ monomials q)*

for *q*::′*a*::*semiring-no-zero-divisors mpoly*
⟨*proof*⟩

3.3 Simplification Lemmas for Const 0 and Const 1

lemma *add-zero* : *P* + *Const 0* = *P*
⟨*proof*⟩

lemma *add-zero-example* : ((*Var 0*)^*2* − (*Const 1*)) + *Const 0* = ((*Var 0*)^*2* − (*Const 1*))
⟨*proof*⟩

lemma *mult-zero-left* : *Const 0* ∗ *P* = *Const 0*
⟨*proof*⟩

lemma *mult-zero-right* : *P* ∗ *Const 0* = *Const 0*
⟨*proof*⟩

lemma *mult-one-left* : *Const 1* ∗ (*P* :: *real mpoly*) = *P*
⟨*proof*⟩

lemma *mult-one-right* : (*P*::*real mpoly*) ∗ *Const 1* = *P*
⟨*proof*⟩

3.4 Coefficient Lemmas

lemma *coeff-zero*[*simp*]: *MPoly-Type.coeff 0 x* = *0*
⟨*proof*⟩

lemma *coeff-sum*: *MPoly-Type.coeff* (*sum f S*) *x* = *sum* (λ*i*. *MPoly-Type.coeff* (*f i*) *x*) *S*
⟨*proof*⟩

lemma *coeff-mult-Var*: *MPoly-Type.coeff* (*x* ∗ *Var i* ^ *p*) *c* = (*MPoly-Type.coeff x* (*c* − *monomial p i*) *when lookup c i* ≥ *p*)
⟨*proof*⟩

lemma *lookup-update-self*[*simp*]: *Poly-Mapping.update i* (*lookup m i*) *m* = *m*
⟨*proof*⟩

lemma *coeff-Const*: *MPoly-Type.coeff* (*Const p*) *m* = (*p when m* = *0*)
⟨*proof*⟩

lemma *coeff-Var*: *MPoly-Type.coeff* (*Var p*) *m* = (*1 when m* = *monomial 1 p*)
⟨*proof*⟩

3.5 Miscellaneous

lemma *update-0-0*[*simp*]: *Poly-Mapping.update x 0 0* = *0*

⟨*proof*⟩

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

lemma *power-both-sides* :
assumes *Ah* : $(A::real) \geq 0$
assumes *Bh* : $(B::real) \geq 0$
shows $(A \leq B) = (A\hat{}2 \leq B\hat{}2)$
⟨*proof*⟩

lemma *in-list-induct-helper*:
assumes $set(xs) \subseteq X$
assumes P []
assumes $(\bigwedge x.\ x \in X \Longrightarrow (\bigwedge xs.\ P\ xs \Longrightarrow P\ (x \# xs)))$
shows $P\ xs$ ⟨*proof*⟩

theorem *in-list-induct* [*case-names Nil Cons*]:
assumes P []
assumes $(\bigwedge x.\ x \in set(xs) \Longrightarrow (\bigwedge xs.\ P\ xs \Longrightarrow P\ (x \# xs)))$
shows $P\ xs$
⟨*proof*⟩

3.5.1 Keys and vars

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

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

lemma *lookupNotIn* : $x \notin keys\ m \Longrightarrow lookup\ m\ x = 0$
⟨*proof*⟩

3.6 Degree Lemmas

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

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

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

lemma *degree-greater-iff*: *MPoly-Type.degree p x* > *k* ⟷ (∃ *m*∈*monomials p*. *lookup m x* > *k*)
⟨*proof*⟩

lemma *degree-eq-iff*:
MPoly-Type.degree p x = *k* ⟷ (*if k* = *0*
then (∀ *m*∈*monomials p*. *lookup m x* = *0*)
else (∃ *m*∈*monomials p*. *lookup m x* = *k*) ∧ (∀ *m*∈*monomials p*. *lookup m x* ≤ *k*))
⟨*proof*⟩

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

lemma *lookup-eq-and-update-lemma*: *lookup m var* = *deg* ∧ *Poly-Mapping.update var 0 m* = *x* ⟷
m = *Poly-Mapping.update var deg x* ∧ *lookup x var* = *0*
⟨*proof*⟩

lemma *degree-const* : *MPoly-Type.degree* (*Const* (*z*::*real*)) (*x*::*nat*) = *0*
⟨*proof*⟩

lemma *degree-one* : *MPoly-Type.degree* (*Var x* :: *real mpoly*) *x* = *1*
⟨*proof*⟩

3.7 Lemmas on treating a multivariate polynomial as univariate

lemma *coeff-isolate-variable-sparse*:
MPoly-Type.coeff (*isolate-variable-sparse p var deg*) *x* =
(*if lookup x var* = *0*
then MPoly-Type.coeff p (*Poly-Mapping.update var deg x*)
else 0)
⟨*proof*⟩

lemma *isovarspar-sum*:
isolate-variable-sparse (*p*+*q*) *var deg* =
isolate-variable-sparse (*p*) *var deg*
+ *isolate-variable-sparse* (*q*) *var deg*
⟨*proof*⟩

lemma *isolate-zero*[*simp*]: *isolate-variable-sparse 0 var n* = *0*
⟨*proof*⟩

lemma *coeff-isolate-variable-sparse-minus-monomial*:
MPoly-Type.coeff (*isolate-variable-sparse mp var i*) (*m* − *monomial i var*) =
(*if lookup m var* ≤ *i then MPoly-Type.coeff mp* (*Poly-Mapping.update var i m*) *else 0*)
⟨*proof*⟩

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

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

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

lemma *isovar-greater-degree* : ∀ *i* > *MPoly-Type.degree p var*. *isolate-variable-sparse p var i* = *0*
⟨*proof*⟩

lemma *isolate-var-0* : *isolate-variable-sparse* (*Var x*::*real mpoly*) *x 0* = *0*
⟨*proof*⟩

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

lemma *isovarspase-monom* :
assumes *notInKeys* : *x* ∉ *keys m*
assumes *notZero* : *a* ≠ *0*
shows *isolate-variable-sparse* (*MPoly-Type.monom m a*) *x 0* = (*MPoly-Type.monom m a* :: *real mpoly*)
⟨*proof*⟩

lemma *isolate-variable-spase-zero* : *lookup m x* = *0* ⟹
insertion (*nth L*) ((*MPoly-Type.monom m a*)::*real mpoly*) = *0* ⟹
a ≠ *0* ⟹ *insertion* (*nth L*) (*isolate-variable-sparse* (*MPoly-Type.monom m a*) *x 0*) = *0*
⟨*proof*⟩

lemma *isovarsparNotIn* : *x* ∉ *vars* (*p*::*real mpoly*) ⟹ *isolate-variable-sparse p x 0* = *p*
⟨*proof*⟩

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

3.8 Summation Lemmas

lemma *summation-normalized* :

assumes *nonzero* : $(B ::real) \neq 0$
shows $(\sum i = 0..<((n::nat)+1). (f\ i :: real) * B \hat{} (n - i)) = (\sum i = 0..<(n+1). (f\ i) / (B \hat{} i)) * (B \hat{} n)$
⟨*proof*⟩

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

lemma *normalize-summation-less* :
assumes *nonzero* : $(B::real)\neq 0$
shows $(\sum i = 0..<(n+1). (f\ i) * B \hat{} (n - i)) * B \hat{} (n\ mod\ 2) < 0$
$\longleftrightarrow$
$(\sum i = 0..<((n::nat)+1). (f\ i::real) / (B \hat{} i)) < 0$
⟨*proof*⟩

3.9 Additional Lemmas with Vars

lemma *not-in-isovarspar* : *isolate-variable-sparse* $(p::real\ mpoly)\ var\ x = (q::real\ mpoly) \Longrightarrow var\notin(vars\ q)$
⟨*proof*⟩

lemma *not-in-add* : $var\notin(vars\ (p::real\ mpoly)) \Longrightarrow var\notin(vars\ (q::real\ mpoly)) \Longrightarrow var\notin(vars\ (p+q))$
⟨*proof*⟩

lemma *not-in-mult* : $var\notin(vars\ (p::real\ mpoly)) \Longrightarrow var\notin(vars\ (q::real\ mpoly)) \Longrightarrow var\notin(vars\ (p*q))$
⟨*proof*⟩

lemma *not-in-neg* : $var\notin(vars(p::real\ mpoly)) \longleftrightarrow var\notin(vars(-p))$
⟨*proof*⟩

lemma *not-in-sub* : $var\notin(vars\ (p::real\ mpoly)) \Longrightarrow var\notin(vars\ (q::real\ mpoly)) \Longrightarrow var\notin(vars\ (p-q))$
⟨*proof*⟩

lemma *not-in-pow* : $var\notin(vars(p::real\ mpoly)) \Longrightarrow var\notin(vars(p \hat{} i))$
⟨*proof*⟩

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

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

⟨*proof*⟩

lemma *not-contains-insertion-helper* :
$\forall x{\in}keys\ (mapping\text{-}of\ p).\ var \notin keys\ x \Longrightarrow$
$(\bigwedge k\ f.\ (k \notin keys\ f) = (lookup\ f\ k = 0)) \Longrightarrow$
$lookup\ (mapping\text{-}of\ p)\ a = 0\ \vee$
$(\prod aa.\ (if\ aa < length\ L\ then\ L[var := y]\ !\ aa\ else\ 0)\ \hat{}\ lookup\ a\ aa) =$
$(\prod aa.\ (if\ aa < length\ L\ then\ L[var := x]\ !\ aa\ else\ 0)\ \hat{}\ lookup\ a\ aa)$
⟨*proof*⟩

lemma *not-contains-insertion* :
assumes *notIn* : $var \notin vars\ (p{::}\ real\ mpoly)$
assumes *exists* : $insertion\ (nth\text{-}default\ 0\ (list\text{-}update\ L\ var\ x))\ p = val$
shows $insertion\ (nth\text{-}default\ 0\ (list\text{-}update\ L\ var\ y))\ p = val$
⟨*proof*⟩

3.10 Insertion Lemmas

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

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

lemma *insertion-sum'* : $\bigwedge(n{::}nat).\ ((insertion\ f\ (\sum i{\le}n.\ g(i))) = (\sum i{\le}n.\ insertion\ f\ (g\ i)))$
⟨*proof*⟩

lemma *insertion-pow* : $insertion\ f\ (p\hat{}i) = (insertion\ f\ p)\hat{}i$
⟨*proof*⟩

lemma *insertion-neg* : $insertion\ f\ (-p) = -insertion\ f\ p$
⟨*proof*⟩

lemma *insertion-var* :
$length\ L > var \Longrightarrow insertion\ (nth\text{-}default\ 0\ (list\text{-}update\ L\ var\ x))\ (Var\ var) = x$
⟨*proof*⟩

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

lemma *notIn-insertion-sub* : $x{\notin}vars(p{::}real\ mpoly) \Longrightarrow x{\notin}vars(q{::}real\ mpoly)$
$\Longrightarrow insertion\ f\ (p{-}q) = insertion\ f\ p - insertion\ f\ q$
⟨*proof*⟩

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

f B
⟨*proof*⟩

lemma *insertion-four* : *insertion* ((*nth-default 0*) *xs*) *4* = *4*
⟨*proof*⟩

lemma *insertion-add-ind-basecase*:
insertion (*nth* (*list-update L var x*)) (($\sum$ *i*::*nat* ≤ *0*. *isolate-variable-sparse p var i* ∗ (*Var var*)^*i*))
= ($\sum$ *i* = *0*..<(*0*+*1*). *insertion* (*nth* (*list-update L var x*)) (*isolate-variable-sparse p var i* ∗ (*Var var*)^*i*))
⟨*proof*⟩

lemma *insertion-add-ind*:
insertion (*nth-default 0* (*list-update L var x*)) (($\sum$ *i*::*nat* ≤ *d*. *isolate-variable-sparse p var i* ∗ (*Var var*)^*i*))
= ($\sum$ *i* = *0*..<(*d*+*1*). *insertion* (*nth-default 0* (*list-update L var x*)) (*isolate-variable-sparse p var i* ∗ (*Var var*)^*i*))
⟨*proof*⟩

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*^*i*))
= *insertion* (*nth-default 0* (*list-update L var x*)) *p*
⟨*proof*⟩

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*))
⟨*proof*⟩
lemma *insertion-zero* : *insertion f* (*Const 0* ::*real mpoly*) = *0*
⟨*proof*⟩

lemma *insertion-one* : *insertion f* (*Const 1* ::*real mpoly*) = *1*
⟨*proof*⟩

lemma *insertion-const* : *insertion f* (*Const c*::*real mpoly*) = (*c*::*real*)
⟨*proof*⟩

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*
⟨*proof*⟩

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*
⟨*proof*⟩

lemma *degree-sum* : (∀ *i*∈{*0..n::nat*}. *MPoly-Type.degree* (*f i* :: *real mpoly*) *var* $\leq$ *x*) ⟹ (*MPoly-Type.degree* (∑ *x*∈{*0..n*}. *f x*) *var*) $\leq$ *x*
⟨*proof*⟩

lemma *degree-sum-less* : (∀ *i*∈{*0..n::nat*}. *MPoly-Type.degree* (*f i* :: *real mpoly*) *var* < *x*) ⟹ (*MPoly-Type.degree* (∑ *x*∈{*0..n*}. *f x*) *var*) < *x*
⟨*proof*⟩

lemma *varNotIn-degree* :
assumes *var* ∉ *vars p*
shows *MPoly-Type.degree p var* = *0*
⟨*proof*⟩

lemma *degree-mult-leq* :
assumes *pnonzero*: (*p*::*real mpoly*)≠*0*
assumes *qnonzero*: (*q*::*real mpoly*)≠*0*
shows *MPoly-Type.degree* ((*p*::*real mpoly*) ∗ (*q*::*real mpoly*)) *var* $\leq$ (*MPoly-Type.degree p var*) + (*MPoly-Type.degree q var*)
⟨*proof*⟩

lemma *degree-exists-monom*:
assumes *p*≠*0*
shows ∃ *m*∈*monomials p*. *lookup m var* = *MPoly-Type.degree p var*
⟨*proof*⟩

lemma *degree-not-exists-monom*:
assumes *p*≠*0*
shows ¬ (∃ *m*∈*monomials p*. *lookup m var* > *MPoly-Type.degree p var*)
⟨*proof*⟩

lemma *degree-monom*: *MPoly-Type.degree* (*MPoly-Type.monom x y*) *v* = (*if y* = *0 then 0 else lookup x v*)
⟨*proof*⟩

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* ∉ *monomials p*
for *p*::*real mpoly*
⟨*proof*⟩

3.11.2 More isolate_variable_sparse lemmas

lemma *isolate-variable-sparse-ne-zeroD*:
isolate-variable-sparse mp v x ≠ *0* ⟹ *x* ≤ *MPoly-Type.degree mp v*
⟨*proof*⟩

context includes *poly.lifting* **begin**
lift-definition *mpoly-to-nested-poly*::′*a*::*comm-monoid-add mpoly* ⇒ *nat* ⇒ ′*a mpoly Polynomial.poly* **is**
λ(*mp*::′*a mpoly*) (*v*::*nat*) (*p*::*nat*). *isolate-variable-sparse mp v p*
— note *extract-var* nests the other way around
⟨*proof*⟩

lemma *degree-eq-0-mpoly-to-nested-polyI*:
mpoly-to-nested-poly mp v = *0* ⟹ *MPoly-Type.degree mp v* = *0*
⟨*proof*⟩

lemma *coeff-eq-zero-mpoly-to-nested-polyD*: *mpoly-to-nested-poly mp v* = *0* ⟹ *MPoly-Type.coeff mp 0* = *0*
⟨*proof*⟩

lemma *mpoly-to-nested-poly-eq-zero-iff*[*simp*]:
mpoly-to-nested-poly mp v = *0* ⟷ *mp* = *0*
⟨*proof*⟩

lemma *isolate-variable-sparse-degree-eq-zero-iff*: *isolate-variable-sparse p v* (*MPoly-Type.degree p v*) = *0* ⟷ *p* = *0*
⟨*proof*⟩

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

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

end

lemma *isolate-variable-sparse-monom*: *isolate-variable-sparse* (*MPoly-Type.monom m a*) *v i* =
(*if a* = *0* ∨ *lookup m v* ≠ *i then 0 else MPoly-Type.monom* (*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*)^*i*:: *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*) ≠ *0*
shows ((*Const k*)::*real mpoly*) ≠ *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* ∨ *MPoly-Type.degree p var* = *2*
shows *p* =
(*isolate-variable-sparse p var 2*)∗(*Var var*)^*2*
+(*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 *ExecutiblePolyProps*
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 φ = (*if* φ=*TrueF then FalseF else if* φ=*FalseF then TrueF else Neg* φ)

definition *and* :: *'a fm* ⇒ *'a fm* ⇒ *'a fm* **where**
and φ_1 φ_2 =
(*if* φ_1=*TrueF then* φ_2 *else if* φ_2=*TrueF then* φ_1 *else*
if φ_1=*FalseF* ∨ φ_2=*FalseF then FalseF else And* φ_1 φ_2)

definition *or* :: *'a fm* ⇒ *'a fm* ⇒ *'a fm* **where**
or φ_1 φ_2 =
(*if* φ_1=*FalseF then* φ_2 *else if* φ_2=*FalseF then* φ_1 *else*
if φ_1=*TrueF* ∨ φ_2=*TrueF then TrueF else Or* φ_1 φ_2)

definition *list-conj* :: *'a fm list* ⇒ *'a fm* **where**
list-conj fs = *foldr and fs TrueF*

definition *list-disj* :: *'a fm list* ⇒ *'a fm* **where**
list-disj fs = *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 ith free variable (indexed at 0) within an atom that is bound in j quantifiers, then we would use 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*)
$\langle$*proof*$\rangle$

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*) Γ = *aEval a* Γ |
eval (*TrueF*) - = *True* |
eval (*FalseF*) - = *False* |
eval (*And* φ ψ) Γ = ((*eval* φ Γ) $\wedge$ (*eval* ψ Γ)) |
eval (*Or* φ ψ) Γ = ((*eval* φ Γ) $\vee$ (*eval* ψ Γ)) |
eval (*Neg* φ) Γ = ($\neg$ (*eval* φ Γ)) |
eval (*ExQ* φ) Γ = ($\exists$ *x*. (*eval* φ (*x*#Γ))) |
eval (*AllQ* φ) Γ = ($\forall$ *x*. (*eval* φ (*x*#Γ)))|
eval (*AllN i* φ) Γ = ($\forall$ *l*. *length l* = *i* $\longrightarrow$ (*eval* φ (*l* @ Γ)))|
eval (*ExN i* φ) Γ = ($\exists$ *l*. *length l* = *i* $\wedge$ (*eval* φ (*l* @ Γ)))

lemma *eval* (*ExQ* (*Or* (*Atom A*) (*Atom B*))) *xs* = *eval* (*Or* (*ExQ*(*Atom A*)) (*ExQ*(*Atom B*))) *xs*
$\langle$*proof*$\rangle$

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

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

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

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

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

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

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

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

lemma *eval-false* : ¬(*eval FalseF L*)
⟨*proof*⟩

lemma *eval-Neg* : *eval* (*Neg* φ) *L* = *eval* (*neg* φ) *L*
⟨*proof*⟩

lemma *eval-Neg-Neg* : *eval* (*Neg* (*Neg* φ)) *L* = *eval* φ *L*
⟨*proof*⟩

lemma *eval-Neg-And* : *eval* (*Neg* (*And* φ ψ)) *L* = *eval* (*Or* (*Neg* φ) (*Neg* ψ)) *L*
⟨*proof*⟩

lemma *aEval-leq* : *aEval* (*Leq p*) *L* = (*aEval* (*Less p*) *L* ∨ *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* ⇒ *atom fm* ⇒ *bool* **where**
freeIn var (*Atom*(*Eq p*)) = (*var* ∉ (*vars p*))|
freeIn var (*Atom*(*Less p*)) = (*var* ∉ (*vars p*))|
freeIn var (*Atom*(*Leq p*)) = (*var* ∉ (*vars p*))|
freeIn var (*Atom*(*Neq p*)) = (*var* ∉ (*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*)ˆ*2* ∗ *Var 3*))))
(*Neg* (*AllQ* (*Atom* (*Leq* (*Const 5* ∗ (*Var 1*)ˆ*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* φ_1 φ_2) = ((*negation-free* φ_1) ∧ (*negation-free* φ_2)) |
negation-free (*Or* φ_1 φ_2) = ((*negation-free* φ_1) ∧ (*negation-free* φ_2)) |
negation-free (*ExQ* φ) = *negation-free* φ |

negation-free (*AllQ* φ) = *negation-free* φ |
negation-free (*AllN* *i* φ) = *negation-free* φ |
negation-free (*ExN* *i* φ) = *negation-free* φ |
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*$\in$*set*(*F*). *freeIn var f*) $\Longrightarrow$ *freeIn var* (*list-conj F*)
⟨*proof*⟩

lemma *freeIn-list-disj* :
assumes $\forall$ *f*$\in$*set* (*L*::*atom fm list*). *freeIn var f*
shows *freeIn var* (*list-disj L*)
⟨*proof*⟩

lemma *var-not-in-aEval* : *freeIn var* (*Atom* φ) $\Longrightarrow$ ($\exists$ *x*. *aEval* φ (*list-update L var x*)) = ($\forall$ *x*. *aEval* φ (*list-update L var x*))
⟨*proof*⟩

lemma *var-not-in-aEval2* : *freeIn 0* (*Atom* φ) $\Longrightarrow$ ($\exists$ *x*. *aEval* φ (*x#L*)) = ($\forall$ *x*. *aEval* φ (*x#L*))
⟨*proof*⟩

lemma *plugInLinear* :
assumes *lLength* : *length L*>*var*
assumes *nonzero* : *B*$\neq$*0*
assumes *hb* : $\forall$ *v*. *insertion* (*nth-default 0* (*list-update L var v*)) *b* = *B*
assumes *hc* : $\forall$ *v*. *insertion* (*nth-default 0* (*list-update L var v*)) *c* = *C*
shows *aEval* (*Eq*(*b*∗*Var var* + *c*)) (*list-update L 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* : ∀ *var*. ∀ *L*. (*freeIn var φ* ⟶ ((∃ *x*. *eval φ* (*list-update L var x*)) = (∀ *x*. *eval φ* (*list-update L var x*))))
⟨*proof*⟩

lemma *var-not-in-eval2* : ∀ *L*. (*freeIn 0 φ* ⟶ ((∃ *x*. *eval φ* (*x*#*L*)) = (∀ *x*. *eval φ* (*x*#*L*))))
⟨*proof*⟩

lemma *var-not-in-eval3* :
assumes *freeIn var φ*
assumes *length xs′* = *var*
shows ((∃ *x*. *eval φ* (*xs′*@*x*#*L*)) = (∀ *x*. *eval φ* (*xs′*@*x*#*L*)))
⟨*proof*⟩

lemma *eval-list-conj* : *eval* (*list-conj F*) *L* = (∀ *f*∈*set*(*F*). *eval f L*)
⟨*proof*⟩

lemma *eval-list-disj* : *eval* (*list-disj F*) *L* = (∃ *f*∈*set*(*F*). *eval f L*)
⟨*proof*⟩
end

5 Debruijn Indicies 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* $b\ i\ x = (if\ x \le b\ then\ x\ else\ x - i{::}nat)$

lemma *inc-above-dec-above*: $x < b \vee b + i \le x \Longrightarrow inc\text{-}above\ b\ i\ (dec\text{-}above\ b\ i\ x) = x$
⟨*proof*⟩
lemma *dec-above-inc-above*: $dec\text{-}above\ b\ i\ (inc\text{-}above\ b\ i\ x) = x$
⟨*proof*⟩

lemma *inc-above-dec-above-iff*: $inc\text{-}above\ b\ i\ (dec\text{-}above\ b\ i\ x) = x \longleftrightarrow x < b \vee b + i \le x$
⟨*proof*⟩

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

lemma *finite-inc-above-ne*: $finite\ \{x.\ f\ x \ne c\} \Longrightarrow finite\ \{x.\ f\ (inc\text{-}above\ b\ i\ x) \ne c\}$
⟨*proof*⟩

lemma *finite-dec-above-ne*: $finite\ \{x.\ f\ x \ne c\} \Longrightarrow finite\ \{x.\ f\ (dec\text{-}above\ b\ i\ x) \ne c\}$
⟨*proof*⟩

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

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

lemma *higherPowers-lowerPowers*: $higherPowers\ n\ i\ (lowerPowers\ n\ i\ x) = x$
⟨*proof*⟩

lemma *inj-lowerPowers*: $inj\ (lowerPowers\ b\ i)$
⟨*proof*⟩

lemma *lowerPowers-higherPowers*:
$(\bigwedge j.\ n \le j \Longrightarrow j < n + i \Longrightarrow lookup\ x\ j = 0) \Longrightarrow lowerPowers\ n\ i\ (higherPowers\ n\ i\ x) = x$
⟨*proof*⟩

lemma *inj-on-higherPowers*: $inj\text{-}on\ (higherPowers\ n\ i)\ \{x.\ \forall j.\ n \le j \wedge j < n + i \longrightarrow lookup\ x\ j = 0\}$
⟨*proof*⟩

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

lemma *lowerPowers-eq*: *lookup* (*lowerPowers* *b* *i* *p*) *x* = (*if* *b* ≤ *x* ∧ *x* < *b* + *i* *then 0 else lookup p* (*dec-above b i x*))
⟨*proof*⟩

lemma *keys-higherPowers*: *keys* (*higherPowers b i m*) = *dec-above b i* ' (*keys m* ∩ {*x*. *x* ∉ {*b*..<*b*+*i*}})
⟨*proof*⟩

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

lift-definition $lowerPowers_f$::*nat* ⇒ *nat* ⇒ (*nat*, '*a*) *fmap* ⇒ (*nat*, '*a*::*zero*) *fmap*
is λ*b i p x*. *if x* ∈ {*b*..<*b*+*i*} *then None else p* (*dec-above b i x*)
⟨*proof*⟩

lift-definition $higherPowers_f$::*nat* ⇒ *nat* ⇒ (*nat*, '*a*) *fmap* ⇒ (*nat*, '*a*) *fmap*
is λ*b i f x*. *f* (*inc-above b i x*)
⟨*proof*⟩

lemma *map-of-map-key-inverse-fun-eq*:
map-of (*map* (λ(*k*, *y*). (*f k*, *y*)) *xs*) *x* = *map-of xs* (*g x*)
if ⋀*x*. *x* ∈ *set xs* ⟹ *g* (*f* (*fst x*)) = *fst x f* (*g x*) = *x*
for *f*::'*a* ⇒ '*b*
⟨*proof*⟩

lemma *map-of-filter-key-in*: *P x* ⟹ *map-of* (*filter* (λ(*k*, *v*). *P k*) *xs*) *x* = *map-of xs x*
⟨*proof*⟩

lemma *map-of-eq-NoneI*: *x*∉*fst*'*set xs* ⟹ *map-of xs x* = *None*
⟨*proof*⟩

lemma *compute-higherPowers*$_f$[*code*]: $higherPowers_f$ *b i* (*fmap-of-list xs*) =
fmap-of-list (*map* (λ(*k*, *v*). (*if k* < *b then k else k* − *i*, *v*)) (*filter* (λ(*k*, *v*). *k* ∉ {*b*..<*b*+*i*}) *xs*))
⟨*proof*⟩

lemma *compute-lowerPowers*$_f$[*code*]: $lowerPowers_f$ *b i* (*fmap-of-list xs*) =
fmap-of-list (*map* (λ(*k*, *v*). (*if k* < *b then k else k* + *i*, *v*)) *xs*)
⟨*proof*⟩

lemma *compute-higherPowers*[*code*]: *higherPowers n i* (*Pm-fmap xs*) = *Pm-fmap* ($higherPowers_f$ *n i xs*)
⟨*proof*⟩

lemma *compute-lowerPowers*[*code*]: *lowerPowers n i* (*Pm-fmap xs*) = *Pm-fmap* ($lowerPowers_f$ *n i xs*)
⟨*proof*⟩

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

lift-definition *lowerPoly*$_0$::*nat* ⇒ *nat* ⇒ ((*nat*⇒$_0$*nat*)⇒$_0$′*a*::*zero*) ⇒ ((*nat*⇒$_0$*nat*)⇒$_0$ ′*a*) **is**
λ*b i* (*mp*::(*nat*⇒$_0$*nat*)⇒′*a*) *mon*. *mp* (*lowerPowers b i mon*)
⟨*proof*⟩

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

lemma *keys-lowerPoly*$_0$: *keys* (*lowerPoly*$_0$ *b i mp*) = *higherPowers b i* ‘ (*keys mp* ∩ {*x*. ∀*j*∈{*b*..<*b*+*i*}. *lookup x j* = *0*})
⟨*proof*⟩

lift-definition *higherPoly*$_0$::*nat* ⇒ *nat* ⇒ ((*nat*⇒$_0$*nat*)⇒$_0$′*a*::*zero*) ⇒ ((*nat*⇒$_0$*nat*)⇒$_0$ ′*a*) **is**
λ*b i* (*mp*::(*nat*⇒$_0$*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*$_f$::*nat* ⇒ *nat* ⇒ ((*nat*⇒$_0$*nat*), ′*a*::*zero*)*fmap* ⇒ ((*nat*⇒$_0$*nat*), ′*a*)*fmap* **is**
λ*b i* (*mp*::((*nat*⇒$_0$*nat*)⇀′*a*)) *mon*::(*nat*⇒$_0$*nat*). *mp* (*lowerPowers b i mon*)
⟨*proof*⟩

lift-definition *higherPoly*$_f$::*nat* ⇒ *nat* ⇒ ((*nat*⇒$_0$*nat*), ′*a*::*zero*)*fmap* ⇒ ((*nat*⇒$_0$*nat*), ′*a*)*fmap* **is**
λ*b i* (*mp*::((*nat*⇒$_0$*nat*)⇀′*a*)) *mon*::(*nat*⇒$_0$*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*$_0$: *keys* (*higherPoly*$_0$ *b i mp*) = *lowerPowers b i* ‘ (*keys mp*)
⟨*proof*⟩

end

lemma *inc-above-id*[*simp*]: $n < m \Longrightarrow$ *inc-above* $m\ i\ n = n$ ⟨*proof*⟩
lemma *inc-above-Suc*[*simp*]: $n \geq m \Longrightarrow$ *inc-above* $m\ i\ n = n + i$ ⟨*proof*⟩

lemma *compute-lowerPoly*$_0$[*code*]: *lowerPoly*$_0$ *n i* (*Pm-fmap m*) = *Pm-fmap* (*lowerPoly*$_f$ *n i m*)
⟨*proof*⟩

lemma *compute-higherPoly*$_0$[*code*]: *higherPoly*$_0$ *n i* (*Pm-fmap m*) = *Pm-fmap* (*higherPoly*$_f$ *n i m*)
⟨*proof*⟩

lemma *compute-lowerPoly*$_f$[*code*]: *lowerPoly*$_f$ *n i* (*fmap-of-list xs*) =
(*fmap-of-list* (*map* (λ(*mon*, *c*). (*higherPowers n i mon*, *c*))
(*filter* (λ(*mon*, *v*). ∀ *j*∈{*n*..<*n*+*i*}. *lookup mon j* = *0*) *xs*)))
⟨*proof*⟩

lemma *compute-higherPoly*$_f$[*code*]: *higherPoly*$_f$ *n i* (*fmap-of-list xs*) =
fmap-of-list (*filter* (λ(*mon*, *v*). ∀ *j*∈{*n*..<*n*+*i*}. *lookup mon j* = *0*)
(*map* (λ(*mon*, *c*). (*lowerPowers n i mon*, *c*)) *xs*))
⟨*proof*⟩

end

lift-definition *lowerPoly*::*nat* ⇒ *nat* ⇒ ′*a*::*zero mpoly* ⇒ ′*a mpoly* **is** *lowerPoly*$_0$
⟨*proof*⟩
lift-definition *liftPoly*::*nat* ⇒ *nat* ⇒ ′*a*::*zero mpoly* ⇒ ′*a mpoly* **is** *higherPoly*$_0$
⟨*proof*⟩

lemma *coeff-lowerPoly*: *MPoly-Type.coeff* (*lowerPoly b i mp*) *x* = *MPoly-Type.coeff mp* (*lowerPowers b i x*)
⟨*proof*⟩

lemma *coeff-liftPoly*: *MPoly-Type.coeff* (*liftPoly b i mp*) *x* = (*if* (∃ *j*∈{*b*..<*b*+*i*}. *lookup x j* > *0*)
then 0
else MPoly-Type.coeff mp (*higherPowers b i x*))
⟨*proof*⟩

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

lemma *monomials-liftPoly*: *monomials* (*liftPoly b i mp*) = *lowerPowers b i* ‘ (*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* ∈ *monomials p* ⟹ *i* ∉ *vars p* ⟹ *lookup x i* = *0*
  ⟨*proof*⟩

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

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

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

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

lemma *aEval-lowerAtom* : (*freeIn 0* (*Atom A*)) ⟹ (*aEval A* (*x*#*L*) = *aEval* (*lowerAtom 0 1 A*) *L*)
  ⟨*proof*⟩

fun *map-fm-binders* :: (*atom* ⇒ *nat* ⇒ *atom*) ⇒ *atom fm* ⇒ *nat* ⇒ *atom fm*
where
map-fm-binders f TrueF n = *TrueF*|
map-fm-binders f FalseF n = *FalseF*|
map-fm-binders f (*Atom* φ) *n* = *Atom* (*f* φ *n*)|
map-fm-binders f (*And* φ ψ) *n* = *And* (*map-fm-binders f* φ *n*) (*map-fm-binders f* ψ *n*)|
map-fm-binders f (*Or* φ ψ) *n* = *Or* (*map-fm-binders f* φ *n*) (*map-fm-binders f* ψ *n*)|
map-fm-binders f (*ExQ* φ) *n* = *ExQ*(*map-fm-binders f* φ (*n*+*1*))|
map-fm-binders f (*AllQ* φ) *n* = *AllQ*(*map-fm-binders f* φ (*n*+*1*))|
map-fm-binders f (*AllN i* φ) *n* = *AllN i* (*map-fm-binders f* φ (*n*+*i*))|
map-fm-binders f (*ExN i* φ) *n* = *ExN i* (*map-fm-binders f* φ (*n*+*i*))|
map-fm-binders f (*Neg* φ) *n* = *Neg*(*map-fm-binders f* φ *n*)

fun *lowerFm* :: *nat* ⇒ *nat* ⇒ *atom fm* ⇒ *atom fm* **where**
lowerFm d amount f = *map-fm-binders* (λ*a*. λ*x*. *lowerAtom* (*d*+*x*) *amount a*) *f 0*

fun *delete-nth* :: *nat* ⇒ *real list* ⇒ *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 ∀*xs*. (*eval* (*lowerFm 0 1 F*) *xs* = *eval* (*ExQ F*) *xs*)
⟨*proof*⟩

fun *liftAtom* :: *nat* ⇒ *nat* ⇒ *atom* ⇒ *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* ⇒ *nat* ⇒ *atom fm* ⇒ *atom fm* **where**
liftFm d amount f = *map-fm-binders* (λ*a*. λ*x*. *liftAtom* (*d*+*x*) *amount a*) *f 0*

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

lemma *higherPoly*$_0$*-add* : *higherPoly*$_0$ *x y* (*p* + *q*) = *higherPoly*$_0$ *x y p* + *higherPoly*$_0$ *x y q*
⟨*proof*⟩

lemma *liftPoly-add*: *liftPoly w z* (*a* + *b*) = (*liftPoly w z a*) + (*liftPoly w z b*)
⟨*proof*⟩

lemma *vars-lift-add* : *vars*(*liftPoly a b* (*p*+*q*)) ⊆ *vars*(*liftPoly a b* (*p*))∪ *vars*(*liftPoly a b* (*q*))
⟨*proof*⟩

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

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

lemma *lift-add* : *insertion* (*f*::*nat*⇒*real*) (*liftPoly 0 z* (*a* + *b*)) = *insertion f* (*liftPoly 0 z a* + *liftPoly 0 z b*)
⟨*proof*⟩

lemma *lower-power-zero* : *lowerPowers a b 0* = *0*
⟨*proof*⟩

lemma *lift-vars-monom* : *vars* (*liftPoly i j* ((*MPoly-Type.monom m a*)::*real mpoly*)) = (λ*x*. *if x*≥*i then x*+*j else x*) ‘ *vars*(*MPoly-Type.monom m a*)
⟨*proof*⟩

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

lemma *lift0*: (*liftPoly i j 0*) = *0*
⟨*proof*⟩

lemma *lower0*: (*lowerPoly i j 0*) = *0*
⟨*proof*⟩

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

lemma *lower-lift* : *insertion f* (*p*::*real mpoly*) = *insertion f* (*lowerPoly i j* (*liftPoly i j p*))
⟨*proof*⟩
lemma *lift-insertion* : ∀ *init*.

length init = (*i*::*nat*) ⟶
(∀ *I xs*.
(*insertion* (*nth-default 0* (*init* @ *xs*)) (*p*::*real mpoly*)) = (*insertion* ((*nth-default 0*) (*init* @ *I* @ *xs*)) (*liftPoly i* (*length I*) *p*)))
⟨*proof*⟩

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*)
⟨*proof*⟩

lemma *eval-liftFm* :
assumes *length I* = *amount*
assumes *length L* ≥ *d*
shows *eval F L* = *eval* (*liftFm d amount F*) (*insert-into L d I*)
⟨*proof*⟩

lemma *not-in-lift* : *var*∉*vars*(*p*::*real mpoly*) ⟹ *var*+*z*∉*vars*(*liftPoly 0 z p*)
⟨*proof*⟩

lemma *lift-const* : *insertion f* (*liftPoly 0 z* (*Const* (*C*::*real*))) = *insertion f* (*Const C* :: *real mpoly*)
⟨*proof*⟩

lemma *liftPoly-sub*: *liftPoly 0 z* (*a* − *b*) = (*liftPoly 0 z a*) − (*liftPoly 0 z b*)
⟨*proof*⟩

lemma *lift-sub* : *insertion* (*f*::*nat*⇒*real*) (*liftPoly 0 z* (*a* − *b*)) = *insertion f* (*liftPoly 0 z a* − *liftPoly 0 z b*)
⟨*proof*⟩

lemma *lift-minus* :
assumes *insertion* (*f*::*nat* ⇒ *real*) (*liftPoly 0 z* (*c* − *Const* (*C*::*real*))) = *0*
shows *insertion f* (*liftPoly 0 z c*) = *C*
⟨*proof*⟩

end

lemma *lift00* : *liftPoly 0 0* (*a*::*real mpoly*) = *a*
⟨*proof*⟩

end

5.2 Swapping Indicies

theory *Reindex*
imports *Debruijn*

begin

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

definition *swap i j x* = (*if x* = *i then j else if x* = *j then i else x*)

lemma *swap-swap* : *swap i j* (*swap i j x*) = *x*
⟨*proof*⟩

lemma *finite-swap-ne*: *finite* {*x*. *f x* ≠ *c*} ⟹ *finite* {*x*. *f* (*swap b i x*) ≠ *c*}
⟨*proof*⟩

lift-definition *swap0*::*nat* ⇒ *nat* ⇒ ($nat \Rightarrow_0$ ′*a*) ⇒ ($nat \Rightarrow_0$ ′*a*::*zero*)
is λ*b i p x*. *p* (*swap b i x*)::′*a*
⟨*proof*⟩

lemma *swap0-swap0*: *swap0 n i* (*swap0 n i x*) = *x*
⟨*proof*⟩

lemma *inj-swap*: *inj* (*swap b i*)
⟨*proof*⟩

lemma *inj-swap0*: *inj* (*swap0 b i*)
⟨*proof*⟩

lemma *swap0-eq*: *lookup* (*swap0 b i p*) *x* = *lookup p* (*swap b i x*)
⟨*proof*⟩

lemma *eq-onp-swap* : *eq-onp* (λ*f*. *finite* {*x*. *f x* ≠ *0*}) (λ*x*. *lookup m* (*swap b i x*))
(λ*x*. *lookup m* (*swap b i x*))
⟨*proof*⟩

lemma *keys-swap*: *keys* (*swap0 b i m*) = *swap b i* ‘ *keys m*
⟨*proof*⟩

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

lift-definition $swap_f$::*nat* ⇒ *nat* ⇒ (*nat*, ′*a*) *fmap* ⇒ (*nat*, ′*a*::*zero*) *fmap*
is λ*b i p x*. *p* (*swap b i x*)
⟨*proof*⟩

lemma *compute-*$swap_f$[*code*]: $swap_f$ *b i* (*fmap-of-list xs*) =
fmap-of-list (*map* (λ(*k*, *v*). (*swap b i k*, *v*)) *xs*)
⟨*proof*⟩

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

lift-definition *swapPoly*$_0$::*nat* ⇒ *nat* ⇒ ((*nat*⇒$_0$*nat*)⇒$_0$'*a*::*zero*) ⇒ ((*nat*⇒$_0$*nat*)⇒$_0$ '*a*) **is**
λ*b i* (*mp*::(*nat*⇒$_0$*nat*)⇒'*a*) *mon*. *mp* (*swap0 b i mon*)
⟨*proof*⟩

lemma *swap-zero*[*simp*]: *swap0 b i 0* = *0*
⟨*proof*⟩

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

lift-definition *swapPoly*$_f$::*nat* ⇒ *nat* ⇒ ((*nat*⇒$_0$*nat*), '*a*::*zero*)*fmap* ⇒ ((*nat*⇒$_0$*nat*), '*a*)*fmap* **is**
λ*b i* (*mp*::((*nat*⇒$_0$*nat*)⇀'*a*)) *mon*::(*nat*⇒$_0$*nat*). *mp* (*swap0 b i mon*)
⟨*proof*⟩

lemma *keys-swap*$_0$: *keys* (*swapPoly*$_0$ *b i mp*) = *swap0 b i* ‘ (*keys mp*)
⟨*proof*⟩

end

lemma *compute-swapPoly*$_0$[*code*]: *swapPoly*$_0$ *n i* (*Pm-fmap m*) = *Pm-fmap* (*swapPoly*$_f$ *n i m*)
⟨*proof*⟩

lemma *compute-swapPoly*$_f$[*code*]: *swapPoly*$_f$ *n i* (*fmap-of-list xs*) =
(*fmap-of-list* (*map* (λ(*mon*, *c*). (*swap0 n i mon*, *c*))
xs))
⟨*proof*⟩

end
end

lift-definition *swap-poly*::*nat* ⇒ *nat* ⇒ '*a*::*zero mpoly* ⇒ '*a mpoly* **is** *swapPoly*$_0$
⟨*proof*⟩

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

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

lemma *monomials-swap-poly*: *monomials* (*swap-poly b i mp*) = *swap0 b i* ‘ (*monomials mp*)
⟨*proof*⟩

fun *swap-atom* :: *nat* ⇒ *nat* ⇒ *atom* ⇒ *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* ⇒ *nat* ⇒ *atom fm* ⇒ *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* ⇒ *nat* ⇒ *'a list* ⇒ *'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* ⇒ *atom fm* **where**
nnf TrueF = *TrueF* |
nnf FalseF = *FalseF* |
nnf (*Atom a*) = *Atom a* |
nnf (*And* φ_1 φ_2) = *And* (*nnf* φ_1) (*nnf* φ_2) |
nnf (*Or* φ_1 φ_2) = *Or* (*nnf* φ_1) (*nnf* φ_2) |
nnf (*ExQ* φ) = *ExQ* (*nnf* φ) |
nnf (*AllQ* φ) = *AllQ* (*nnf* φ)|
nnf (*AllN i* φ) = *AllN i* (*nnf* φ)|
nnf (*ExN i* φ) = *ExN i* (*nnf* φ) |
nnf (*Neg TrueF*) = *FalseF* |
nnf (*Neg FalseF*) = *TrueF* |
nnf (*Neg* (*Neg* φ)) = (*nnf* φ) |
nnf (*Neg* (*And* φ_1 φ_2)) = (*Or* (*nnf* (*Neg* φ_1)) (*nnf* (*Neg* φ_2))) |
nnf (*Neg* (*Or* φ_1 φ_2)) = (*And* (*nnf* (*Neg* φ_1)) (*nnf* (*Neg* φ_2))) |
nnf (*Neg* (*Atom a*)) = *Atom*(*aNeg a*) |
nnf (*Neg* (*ExQ* φ)) = *AllQ* (*nnf* (*Neg* φ)) |
nnf (*Neg* (*AllQ* φ)) = *ExQ* (*nnf* (*Neg* φ))|
nnf (*Neg* (*AllN i* φ)) = *ExN i* (*nnf* (*Neg* φ))|
nnf (*Neg* (*ExN i* φ)) = *AllN i* (*nnf* (*Neg* φ))

6.1 Simplify Constants

fun *simp-atom* :: *atom* ⇒ *atom fm* **where**
simp-atom (*Eq p*) = (*case get-if-const p of None* ⇒ *Atom*(*Eq p*) | *Some*(*r*) ⇒ (*if r=0 then TrueF else FalseF*))|
simp-atom (*Less p*) = (*case get-if-const p of None* ⇒ *Atom*(*Less p*) | *Some*(*r*) ⇒ (*if r<0 then TrueF else FalseF*))|
simp-atom (*Leq p*) = (*case get-if-const p of None* ⇒ *Atom*(*Leq p*) | *Some*(*r*) ⇒ (*if r≤0 then TrueF else FalseF*))|

simp-atom (*Neq p*) = (*case get-if-const p of None* ⇒ *Atom*(*Neq p*) | *Some*(*r*) ⇒ (*if r≠0 then TrueF else FalseF*))

fun *simpfm* :: *atom fm* ⇒ *atom fm* **where**
simpfm TrueF = *TrueF*|
simpfm FalseF = *FalseF*|
simpfm (*Atom a*) = *simp-atom a*|
simpfm (*And φ ψ*) = *and* (*simpfm φ*) (*simpfm ψ*)|
simpfm (*Or φ ψ*) = *or* (*simpfm φ*) (*simpfm ψ*)|
simpfm (*ExQ φ*) = *ExQ* (*simpfm φ*)|
simpfm (*Neg φ*) = *neg* (*simpfm φ*)|
simpfm (*AllQ φ*) = *AllQ*(*simpfm φ*)|
simpfm (*AllN i φ*) = *AllN i* (*simpfm φ*)|
simpfm (*ExN i φ*) = *ExN i* (*simpfm φ*)

6.2 Group Quantifiers

fun *groupQuantifiers* :: *atom fm* ⇒ *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 who's 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* ⇒ *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 + *i* ∗ *2* + *1*|
depth′ (*ExN* *i* φ) = *depth′* φ + *i* ∗ *2* + *1*

function *clearQuantifiers* :: *atom fm* ⇒ *atom fm* **where**
clearQuantifiers TrueF = *TrueF*|
clearQuantifiers FalseF = *FalseF*|
clearQuantifiers (*Atom a*) = *simp-atom a*|
clearQuantifiers (*And* φ ψ) = *and* (*clearQuantifiers* φ) (*clearQuantifiers* ψ)|
clearQuantifiers (*Or* φ ψ) = *or* (*clearQuantifiers* φ) (*clearQuantifiers* ψ)|
clearQuantifiers (*Neg* φ) = *neg* (*clearQuantifiers* φ)|
clearQuantifiers (*ExQ* φ) =
(*let* φ' = *clearQuantifiers* φ *in*
(*if freeIn 0* φ' *then lowerFm 0 1* φ' *else ExQ* φ'))|
clearQuantifiers (*AllQ* φ) =
(*let* φ' = *clearQuantifiers* φ *in*
(*if freeIn 0* φ' *then lowerFm 0 1* φ' *else AllQ* φ'))|
clearQuantifiers (*ExN 0* φ) = *clearQuantifiers* φ|
clearQuantifiers (*ExN* (*Suc i*) φ) = *clearQuantifiers* (*ExN i* (*ExQ* φ))|
clearQuantifiers (*AllN 0* φ) = *clearQuantifiers* φ|
clearQuantifiers (*AllN* (*Suc i*) φ) = *clearQuantifiers* (*AllN i* (*AllQ* φ))
⟨*proof*⟩
termination
⟨*proof*⟩

6.4 Push Forall

fun *push-forall* :: *atom fm* ⇒ *atom fm* **where**
push-forall TrueF = *TrueF*|
push-forall FalseF = *FalseF*|
push-forall (*Atom a*) = *simp-atom a*|
push-forall (*And* φ ψ) = *and* (*push-forall* φ) (*push-forall* ψ)|
push-forall (*Or* φ ψ) = *or* (*push-forall* φ) (*push-forall* ψ)|
push-forall (*ExQ* φ) = *ExQ* (*push-forall* φ)|
push-forall (*ExN i* φ) = *ExN i* (*push-forall* φ)|
push-forall (*Neg* φ) = *neg* (*push-forall* φ)|
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* φ ψ)) = *and* (*push-forall* (*AllQ* φ)) (*push-forall* (*AllQ* ψ))|
push-forall (*AllQ* (*Or* φ ψ)) = (
if freeIn 0 φ

```
  then(
    if freeIn 0 ψ
    then or (lowerFm 0 1 φ) (lowerFm 0 1 ψ)
    else or (lowerFm 0 1 φ) (AllQ ψ))
  else (
    if freeIn 0 ψ
    then or (AllQ φ) (lowerFm 0 1 ψ)
    else AllQ (or φ ψ))
)|
  push-forall (AllQ φ) = (if freeIn 0 φ then lowerFm 0 1 φ else AllQ φ)|
  push-forall (AllN i φ) = AllN i (push-forall  φ)
```

6.5 Unpower

```
fun to-list :: nat ⇒ real mpoly ⇒ (real mpoly ∗ nat) list where
  to-list v p = [(isolate-variable-sparse p v x, x). x ← [0..<(MPoly-Type.degree p
v)+1]]

fun chop :: (real mpoly ∗ nat) list ⇒ (real mpoly ∗ nat) listwhere
  chop [] = []|
  chop ((p,i)#L) = (if p=0 then chop L else (p,i)#L)

fun decreasePower :: nat ⇒ real mpoly ⇒ real mpoly ∗ natwhere
  decreasePower v p = (case chop (to-list v p) of [] ⇒ (p,0) | ((p,i)#L) ⇒ (sum-list
[term ∗ (Var v) ^(x−i). (term,x)←((p,i)#L)],i))

fun unpower :: nat ⇒ atom fm ⇒ atom fm where
   unpower v (Atom (Eq p)) = (case decreasePower v p of (-,0) ⇒ Atom(Eq p)|
(p,-) ⇒ Or(Atom (Eq p))(Atom (Eq (Var v))) )|
  unpower v (Atom (Neq p)) = (case decreasePower v p of (-,0) ⇒ Atom(Neq p)|
(p,-) ⇒ And(Atom (Neq p))(Atom (Neq (Var v))) )|
  unpower v (Atom (Less p)) = (case decreasePower v p of (-,0) ⇒ Atom(Less p)|
(p,n) ⇒
  if n mod 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) ⇒ Atom(Leq p)|
(p,n) ⇒
  if n mod 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* : $\forall\,\Gamma.\ (\neg\ eval\ (nnf\ (Neg\ \varphi))\ \Gamma) = eval\ (nnf\ \varphi)\ \Gamma$
  ⟨*proof*⟩

theorem *eval-nnf* : $\forall\,\Gamma.\ eval\ \varphi\ \Gamma = eval\ (nnf\ \varphi)\ \Gamma$
  ⟨*proof*⟩

theorem *negation-free-nnf* : *negation-free* (*nnf* φ)
⟨*proof*⟩

lemma *groupQuantifiers-eval* : *eval F L* = *eval* (*groupQuantifiers F*) *L*
  ⟨*proof*⟩

theorem *simp-atom-eval* : *aEval a xs* = *eval* (*simp-atom a*) *xs*
⟨*proof*⟩

lemma *simpfm-eval* : $\forall\,L.\ eval\ \varphi\ L = eval\ (simpfm\ \varphi)\ L$
  ⟨*proof*⟩

lemma *exQ-clearQuantifiers*:
  assumes *ExQ* : $\bigwedge xs.\ eval\ (clearQuantifiers\ \varphi)\ xs = eval\ \varphi\ xs$
  shows $eval\ (clearQuantifiers\ (ExQ\ \varphi))\ xs = eval\ (ExQ\ \varphi)\ xs$
⟨*proof*⟩

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

shows *eval* (*clearQuantifiers* (*AllQ* φ)) *xs* = *eval* (*AllQ* φ) *xs*
⟨*proof*⟩

lemma *clearQuantifiers-eval* : *eval* (*clearQuantifiers* φ) *xs* = *eval* φ *xs*
⟨*proof*⟩

lemma *push-forall-eval-AllQ* : ∀ *xs*. *eval* (*AllQ* φ) *xs* = *eval* (*push-forall* (*AllQ* φ)) *xs*
⟨*proof*⟩

lemma *push-forall-eval* : ∀ *xs*. *eval* φ *xs* = *eval* (*push-forall* φ) *xs*
⟨*proof*⟩

lemma *map-fm-binders-negation-free* :
assumes *negation-free* φ
shows *negation-free* (*map-fm-binders f* φ *n*)
⟨*proof*⟩

lemma *negation-free-and* :
assumes *negation-free* φ
assumes *negation-free* ψ
shows *negation-free* (*and* φ ψ)
⟨*proof*⟩

lemma *negation-free-or* :
assumes *negation-free* φ
assumes *negation-free* ψ
shows *negation-free* (*or* φ ψ)
⟨*proof*⟩

lemma *push-forall-negation-free-all* :
assumes *negation-free* φ
shows *negation-free* (*push-forall* (*AllQ* φ))
⟨*proof*⟩

lemma *push-forall-negation-free* :
assumes *negation-free* φ
shows *negation-free*(*push-forall* φ)
⟨*proof*⟩

lemma *to-list-insertion*: *insertion f p* = *sum-list* [*insertion f term* ∗ (*f v*) ^ *i*. (*term*,*i*)←(*to-list v p*)]
⟨*proof*⟩

lemma *to-list-p*: *p* = *sum-list* [*term* ∗ (*Var v*) ^ *i*. (*term*,*i*)←(*to-list v p*)]
⟨*proof*⟩

fun *chophelper* :: (*real mpoly* ∗ *nat*) *list* ⇒ (*real mpoly* ∗ *nat*) *list* ⇒ (*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 ∀ (p,i)∈*set*(L'). p=*0*
assumes (a,b)=*chophelper* L L'
shows ∀ (p,i)∈*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) ^ i. (*term*,i)←a] + *sum-list* [*insertion* f *term* ∗ (f v) ^ i. (*term*,i)←b]
⟨*proof*⟩

lemma *chopped* :
assumes (a,b)=*chophelper* (*to-list* v p) []
shows *insertion* f p = *sum-list* [*insertion* f *term* ∗ (f v) ^ i. (*term*,i)←b]
⟨*proof*⟩

lemma *insertion-chop* :
shows *insertion* f p = *sum-list* [*insertion* f *term* ∗ (f v) ^ i. (*term*,i)←(*chop* (*to-list* v p))]
⟨*proof*⟩

lemma *sorted* : *sorted-wrt* (λ(-,i).λ(-,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) ^(d−i). (*term*,d)←(*chop* (*to-list* v p))] ∗ (f v)^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 φ*) *L* = *eval φ L*
⟨*proof*⟩

lemma *to-list-filter*: *p* = *sum-list* [*term* ∗ (*Var v*) ^ *i*. (*term*,*i*)←((*filter* (λ(*x*,-). *x*≠*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* ⇒ *atom list* ⇒ *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* ∧ *MPoly-Type.degree p var* ≠ *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* (λ*x*.λ*A*. *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 * sqrt(c))/d for an arbitrary c and substitutes it in for the variable var in the atom A

fun *quadratic-part-1* :: *nat* ⇒ *real mpoly* ⇒ *real mpoly* ⇒ *real mpoly* ⇒ *atom* ⇒

real mpoly **where**
quadratic-part-1 var a b d (Eq p) = (
let deg = MPoly-Type.degree p var in
$\sum$ *i∈{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∈{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∈{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∈{0..<(deg+1)}. (isolate-variable-sparse p var i) ∗ ((a+b∗(Var var))^i) ∗ (d^(deg − i))*
)

fun *quadratic-part-2* :: *nat* ⇒ *real mpoly* ⇒ *real mpoly* ⇒ *real mpoly* **where**
quadratic-part-2 var sq p = (
let deg = MPoly-Type.degree p var in
$\sum$ *i∈{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))
)

quadraticsub 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* ⇒ *real mpoly* ⇒ *real mpoly* ⇒ *real mpoly* ⇒ *real mpoly* ⇒ *atom* ⇒ *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 (A^2−B^2∗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* ⇒ *nat* ⇒ *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* ⇒ *nat* ⇒ *atom fm* **where**

```
  alternateNegInfinity p var = foldl (λF.λi.
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* ⇒ *atom* ⇒ *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* ⇒ *real mpoly* ⇒ *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* ⇒ *real mpoly* ⇒ *real mpoly* ⇒ *atom* ⇒ *atom fm* **where**

```
  substInfinitesimalLinear var b c (Eq p) = allZero p var|
  substInfinitesimalLinear var b c (Less p) =
  liftmap
    (λx. λA. Atom(linear-substitution (var+x) (liftPoly 0 x b) (liftPoly 0 x c) A))
    (convertDerivative var p)
    0|
  substInfinitesimalLinear var b c (Leq p) =
Or
  (allZero p var)
  (liftmap
    (λx. λA. Atom(linear-substitution (var+x) (liftPoly 0 x b) (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* (λ*x*.λ*A*. *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* (λ*x*.λ*A*. *substInfinitesimalQuadratic* (*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 conjuctive 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* (λ*a*. *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)
       ))
      ))
     )))
```

```
|
  elimVar var L F (Leq 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)
        ))
       ))
      ))

)


fun qe-eq-one :: nat ⇒ atom list ⇒ atom fm list ⇒ atom fm where
  qe-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 ⇒ boolwhere
  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 optionwhere
  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 *qe-eq-repeat-helper* :: *nat* ⇒ *real mpoly list* ⇒ *atom list* ⇒ *atom fm list* ⇒ *atom fm* **where**
qe-eq-repeat-helper var [] *L F* = *list-conj* ((*map Atom L*) @ *F*)|
qe-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*)
(*qe-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 (*Neq 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* φ_1 φ_2) = [(*A*@*B*,*A*′@*B*′).(*A*,*A*′)←*dnf* φ_1,(*B*,*B*′)←*dnf* φ_2] |
dnf (*Or* φ_1 φ_2) = *dnf* φ_1 @ *dnf* φ_2 |
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* ⇒ (*atom list* ∗ *atom fm list* ∗ *nat*) *list* **where**
dnf-modified TrueF = [([],[],*0*)] |
dnf-modified FalseF = [] |
dnf-modified (*Atom* φ) = [([φ],[],*0*)] |
dnf-modified (*And* φ_1 φ_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*) ← *dnf-modified* φ_1, (*B*,*B′*,*d2*) ← *dnf-modified* φ_2] |
dnf-modified (*Or* φ_1 φ_2) = *dnf-modified* φ_1 @ *dnf-modified* φ_2 |
dnf-modified (*ExQ* φ) = [(*A*,*A′*,*d*+*1*). (*A*,*A′*,*d*) ← *dnf-modified* φ] |
dnf-modified (*Neg* φ) = [([],[*Neg* φ],*0*)]|
dnf-modified (*AllQ* φ) = [([],[*AllQ* φ],*0*)]|
dnf-modified (*AllN i* φ) = [([],[*AllN i* φ],*0*)]|
dnf-modified (*ExN i* φ) = [(*A*,*A′*,*d*+*i*). (*A*,*A′*,*d*) ← *dnf-modified* φ]

fun *QE-dnf* :: (*atom fm* ⇒ *atom fm*) ⇒ (*nat* ⇒ *nat* ⇒ *atom list* ⇒ *atom fm list* ⇒ *atom fm*) ⇒ *atom fm* ⇒ *atom fm* **where**
QE-dnf opt step (*And* φ_1 φ_2) = *and* (*QE-dnf opt step* φ_1) (*QE-dnf opt step* φ_2) |
QE-dnf opt step (*Or* φ_1 φ_2) = *or* (*QE-dnf opt step* φ_1) (*QE-dnf opt step* φ_2) |
QE-dnf opt step (*Neg* φ) = *neg*(*QE-dnf opt step* φ) |
QE-dnf opt step (*ExQ* φ) = *list-disj* [*ExN* (*n*+*1*) (*step 1 n al fl*). (*al*,*fl*,*n*)←(*dnf-modified*(*opt*(*QE-dnf opt step* φ)))]|
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* φ) = *Neg*(*list-disj* [*ExN* (*n*+*1*) (*step 1 n al fl*). (*al*,*fl*,*n*)←(*dnf-modified*(*opt*(*neg*(*QE-dnf opt step* φ))))])|
QE-dnf opt step (*ExN 0* φ) = *QE-dnf opt step* φ|
QE-dnf opt step (*AllN 0* φ) = *QE-dnf opt step* φ|
QE-dnf opt step (*AllN* (*Suc i*) φ) = *Neg*(*list-disj* [*ExN* (*n*+*i*+*1*) (*step* (*Suc i*) (*n*+*i*) *al fl*). (*al*,*fl*,*n*)←(*dnf-modified*(*opt*(*neg*(*QE-dnf opt step* φ))))])|
QE-dnf opt step (*ExN* (*Suc i*) φ) = *list-disj* [*ExN* (*n*+*i*+*1*) (*step* (*Suc i*) (*n*+*i*) *al fl*). (*al*,*fl*,*n*)←(*dnf-modified*(*opt*(*QE-dnf opt step* φ)))]

fun *QE-dnf′* :: (*atom fm* ⇒ *atom fm*) ⇒ (*nat* ⇒ (*atom list* ∗ *atom fm list* ∗ *nat*) *list* ⇒ *atom fm*) ⇒ *atom fm* ⇒ *atom fm* **where**
QE-dnf′ opt step (*And* φ_1 φ_2) = *and* (*QE-dnf′ opt step* φ_1) (*QE-dnf′ opt step* φ_2) |
QE-dnf′ opt step (*Or* φ_1 φ_2) = *or* (*QE-dnf′ opt step* φ_1) (*QE-dnf′ opt step* φ_2) |
QE-dnf′ opt step (*Neg* φ) = *neg*(*QE-dnf′ opt step* φ) |
QE-dnf′ opt step (*ExQ* φ) = *step 1* (*dnf-modified*(*opt*(*QE-dnf′ opt step* φ)))|
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* φ) = *Neg*(*step 1* (*dnf-modified*(*opt*(*neg*(*QE-dnf′ opt step* φ))))))|
QE-dnf′ opt step (*ExN 0* φ) = *QE-dnf′ opt step* φ|
QE-dnf′ opt step (*AllN 0* φ) = *QE-dnf′ opt step* φ|
QE-dnf′ opt step (*AllN* (*Suc i*) φ) = *Neg*(*step* (*Suc i*) (*dnf-modified*(*opt*(*neg*(*QE-dnf′ opt step* φ))))))|
QE-dnf′ opt step (*ExN* (*Suc i*) φ) = *step* (*Suc i*) (*dnf-modified*(*opt*(*QE-dnf′ opt step* φ)))

7.5 Repeat QE multiple times

fun *countQuantifiers* :: *atom fm* ⇒ *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* ⇒ *atom fm*) ⇒ *nat* ⇒ *atom fm* ⇒ *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* ⇒ *atom fm*) ⇒ *atom fm* ⇒ *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 ∘ unpower 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 (λx. Pair(count-list (map (λl. case l of
Eq p ⇒ MPoly-Type.degree p x = 0
| Less p ⇒ MPoly-Type.degree p x = 0
| Neq 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) = aquireData 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,qe-eq-repeat))
| Some v ⇒ Some(v,gen-qe)
)
| - ⇒ (case getBest e L of None ⇒ None | Some v ⇒ Some(v,qe-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 ∘ nnf ∘ 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
    ExecutiblePolyProps
    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 × nat) ⇒ shows) ⇒ ('a ⇒ shows) ⇒ shows
⇒ (nat ⇒0 nat) ⇒ 'a option ⇒ shows where
  shows-monomial-gen shows-factor shows-coeff sep mon cff =
    shows-sep (λs. case s of
        Inl cff ⇒ shows-coeff cff
```

| *Inr factor* ⇒ *shows-factor factor*
) *sep* ((*case cff of None* ⇒ [] | *Some cff* ⇒ [*Inl cff*]) @ *map Inr* (*Poly-Mapping.items mon*))

definition *shows-factor-compact factor* =
(*case factor of* (*k*, *v*) ⇒ *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*) ⇒ *shows-string* ''(*Var* '' +@+ *shows k* +@+ *shows-string* '')'' +@+
(*if v* = *1 then shows-string* '''' *else shows-string* ''^'' +@+ *shows v*))

definition *shows-monomial-compact*::('*a* ⇒ *shows*) ⇒ (*nat* $\Rightarrow_0$ *nat*) ⇒ '*a option* ⇒ *shows* **where**
shows-monomial-compact shows-coeff m =
shows-monomial-gen shows-factor-compact shows-coeff (*shows-string* '' '') *m*

definition *shows-monomial-Var*::('*a* ⇒ *shows*) ⇒ (*nat* $\Rightarrow_0$ *nat*) ⇒ '*a option* ⇒ *shows* **where**
shows-monomial-Var shows-coeff m =
shows-monomial-gen shows-factor-Var shows-coeff (*shows-string* ''∗'') *m*

fun *shows-mpoly* :: *bool* ⇒ ('*a* ⇒ *shows*) ⇒ '*a*::{*zero*,*one*} *mpoly* ⇒ *shows* **where**
shows-mpoly input shows-coeff p = *shows-sep* (λ*mon*.
(*if input then shows-monomial-Var* (λ*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* (∃ *r*::*rat*. *x* = *of-rat r*) *then* (*THE r*. *x* = *of-rat r*) *else 99999999999.99999999999*)

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* = *shows* (*rat-of-real x*)

experiment begin
abbreviation *foo* ≡ ((*Var 0*::*real mpoly*) + *Const* (*0.5*) ∗ *Var 1* + *Var 2*)^*3*
value [*code*] *shows-mpoly True shows-real foo* ''''

lemma *foo-eq*: *foo* = (*Var 0*)^*3* + (*Const* (*3/2*))*(*Var 0*)^*2**(*Var 1*) + (*Const* (*3*))*(*Var 0*)^*2**(*Var 2*) + (*Const* (*3/4*))*(*Var 0*)*(*Var 1*)^*2* + (*Const* (*3*))*(*Var 0*)*(*Var 1*)*(*Var 2*) + (*Const* (*3*))*(*Var 0*)*(*Var 2*)^*2* + (*Const* (*1/8*))*(*Var 1*)^*3* + (*Const* (*3/4*))*(*Var 1*)^*2**(*Var 2*) + (*Const* (*3/2*))*(*Var 1*)*(*Var 2*)^*2* + (*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* ⇒ *atom* ⇒ *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(*Neq p*) = (*shows-string ''*('' +@+ *shows-mpoly c shows-real p* +@+ *shows-string ''~=0*)'')

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* + *i* * *2* + *1*|
depth' (*ExN i φ*) = *depth' φ* + *i* * *2* + *1*

function *shows-fm* :: *bool* ⇒ *atom fm* ⇒ *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 φ ψ*) = (*shows-string ''*('' +@+ *shows-fm c φ* +@+ *shows-string '' and ''* +@+ *shows-fm c ψ* +@+ *shows-string* ('')''))|
shows-fm c (*Or φ ψ*) = (*shows-string ''*('' +@+ *shows-fm c φ* +@+ *shows-string '' or ''* +@+ *shows-fm c ψ* +@+ *shows-string* '')'')|
shows-fm c (*Neg φ*) = (*shows-string ''*(*neg ''* +@+ *shows-fm c φ* +@+ *shows-string* '')'')|
shows-fm c (*ExQ φ*) = (*shows-string ''*(*exists''* +@+ *shows-fm c φ* +@+ *shows-string* '')'')|

shows-fm c (*AllQ* φ) = (*shows-string* ''(*forall*'' +@+ *shows-fm c* φ +@+ *shows-string* '')'')|
shows-fm c (*ExN 0* φ) = *shows-fm c* φ|
shows-fm c (*ExN* (*Suc n*) φ) = *shows-fm c* (*ExQ*(*ExN n* φ))|
shows-fm c (*AllN 0* φ) = *shows-fm c* φ|
shows-fm c (*AllN* (*Suc n*) φ) = *shows-fm c* (*AllQ*(*AllN n* φ))
⟨*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* ∘ *nnf* ∘ *unpower 0 o clearQuantifiers*)
definition *opt-group* = (*push-forall* ∘ *nnf* ∘ *unpower 0 o groupQuantifiers o clearQuantifiers*)

definition *VSLuckiest* = *opt o* (*QE-dnf opt* (λ*amount. luckiestFind*)) *o opt*
definition *VSLuckiestBlocks* =*opt-group o* (*QE-dnf' opt-group* (*the-real-step-augment luckiestFind*)) *o opt-group*
definition *VSEquality* =*opt o* (*QE-dnf opt*(λ*x. qe-eq-repeat*)) *o VSLuckiest o opt*
definition *VSEqualityBlocks* =*opt-group o* (*QE-dnf' opt-group* (*the-real-step-augment qe-eq-repeat*)) *o VSLuckiestBlocks o opt-group*
definition *VSGeneralBlocks* =*opt-group o* (*QE-dnf' opt-group* (*the-real-step-augment gen-qe*))*o VSLuckiestBlocks o opt-group*
definition *VSLuckyBlocks* =*opt-group o* (*QE-dnf' opt-group* (*the-real-step-augment luckyFind'*))*o VSLuckiestBlocks o opt-group*
definition *VSLEGBlocks* = *VSGeneralBlocks o VSEqualityBlocks o VSLuckyBlocks*
definition *VSEqualityBlocksLimited* =*opt-group o* (*QE-dnf opt-group* (*step-augment qe-eq-repeat IdentityHeuristic*)) *o VSLuckiestBlocks o opt-group*
definition *VSEquality-3-times* = *VSEquality o VSEquality o VSEquality*
definition *VSGeneral* = *opt o* (*QE-dnf opt* (λ*x. gen-qe*)) *o VSLuckiest o opt*
definition *VSGeneralBlocksLimited* = *opt-group o* (*QE-dnf opt-group* (*step-augment*

gen-qe IdentityHeuristic)) *o VSLuckiestBlocks o opt-group*
definition *VSBrowns* = *opt-group o* (*QE-dnf opt-group* (*step-augment gen-qe brownsHeuristic*)) *o VSLuckiestBlocks o opt-group*
definition *VSGeneral-3-times* = *VSGeneral o VSGeneral o VSGeneral*
definition *VSLucky* = *opt o* (*QE-dnf opt* (*λamount. luckyFind′*)) *o VSLuckiest o opt*
definition *VSLuckyBlocksLimited* = *opt-group o* (*QE-dnf opt-group* (*step-augment luckyFind′ IdentityHeuristic*)) *o VSLuckiestBlocks o opt-group*
definition *VSLEG* = *VSGeneral o VSEquality o VSLucky*
definition *VSHeuristic* = *opt-group o* (*QE-dnf opt-group* (*superPicker*)) *o VSLuckiestBlocks o opt-group*
definition *VSLuckiestRepeat* = *repeatAmountOfQuantifiers VSLuckiest*

definition *add* :: *real mpoly* ⇒ *real mpoly* ⇒ *real mpoly* **where**
add p q = *p* + *q*

definition *minus* :: *real mpoly* ⇒ *real mpoly* ⇒ *real mpoly* **where**
minus p q = *p* − *q*

definition *mult* :: *real mpoly* ⇒ *real mpoly* ⇒ *real mpoly* **where**
mult p q = *p* ∗ *q*

definition *pow* :: *real mpoly* ⇒ *integer* ⇒ *real mpoly* **where**
pow p n = *p* ^ (*nat-of-integer n*)

definition *C* :: *real* ⇒ *real mpoly* **where**
C r = *Const r*

definition *V* :: *integer* ⇒ *real mpoly* **where**
V n = *Var* (*nat-of-integer n*)

definition *real-of-int* :: *integer* ⇒ *real*
where *real-of-int n* = *real* (*nat-of-integer n*)

definition *real-mult* :: *real* ⇒ *real* ⇒ *real*
where *real-mult n m* = *n* ∗ *m*

definition *real-div* :: *real* ⇒ *real* ⇒ *real*
where *real-div n m* = *n* / *m*

definition *real-plus* :: *real* ⇒ *real* ⇒ *real*
where *real-plus n m* = *n* + *m*

definition *real-minus* :: *real* ⇒ *real* ⇒ *real*
where *real-minus n m* = *n* − *m*

fun *is-quantifier-free* :: *atom fm* ⇒ *bool* **where**
is-quantifier-free (*ExQ x*) =*False*|

is-quantifier-free (*AllQ x*) =*False*|
is-quantifier-free (*And a b*) =(*is-quantifier-free a* ∧ *is-quantifier-free b*)|
is-quantifier-free (*Or a b*) =(*is-quantifier-free a* ∧ *is-quantifier-free b*)|
is-quantifier-free (*Neg a*) =*is-quantifier-free a*|
is-quantifier-free a = *True*

fun *is-solved* :: *atom fm* ⇒ *bool* **where**
is-solved TrueF = *True*|
is-solved FalseF = *True*|
is-solved A = *False*

definition *print-mpoly* :: (*real* ⇒ *String.literal*)⇒ *real mpoly* ⇒ *String.literal* **where**
print-mpoly f p = *String.implode* ((*shows-mpoly True* (λ*x*.λ*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 VSHeuristic 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* ∉ *vars b*
assumes *var* ∉ *vars c*
shows *freeIn var* (*Atom* (*linear-substitution var b c A*))
⟨*proof*⟩

lemma *linear-eq* :
assumes *lLength* : *length L* > *var*
assumes *nonzero* : *C* ≠ *0*
assumes *var* ∉ *vars b*
assumes *var* ∉ *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*))
⟨*proof*⟩

lemma *linear-less* :
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* (*Less*(*p*)) (*list-update L var* (*B*/*C*)) = (*aEval* (*linear-substitution var b c* (*Less*(*p*))) (*list-update L var v*))
⟨*proof*⟩

lemma *linear-leq* :
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* (*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* : $C \neq 0$
  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 $C \neq 0$
  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* ∉ *vars b*
  assumes *var* ∉ *vars c*

assumes *insertion* (*nth-default 0* (*list-update* (*drop z L*) *var* (*B*/*C*))) *b* = (*B*::*real*)
assumes *insertion* (*nth-default 0* (*list-update* (*drop z L*) *var* (*B*/*C*))) *c* = (*C*::*real*)
assumes *lLength* : *length L* > *var*+*z*
shows *eval F* (*list-update L* (*var*+*z*) (*B*/*C*)) = (*eval* (*linear-substitution-fm-helper var b c F z*) (*list-update L* (*var*+*z*) *v*))
⟨*proof*⟩

theorem *linear-fm* :
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 *eval F* (*list-update L var* (*B*/*C*)) = (∀ *v*. *eval* (*linear-substitution-fm var b c F*) (*list-update L var v*))
⟨*proof*⟩
end

8.2 Quadratic Case

theory *QuadraticCase*
imports *VSAlgos*
begin

lemma *quad-part-1-eq* :
assumes *lLength* : *length L* > *var*
assumes *hdeg* : *MPoly-Type.degree* (*p*::*real mpoly*) *var* = (*deg*::*nat*)
assumes *nonzero* : *D* ≠ *0*
assumes *ha* : ∀ *x*. *insertion* (*nth-default 0* (*list-update L var x*)) *a* = (*A*::*real*)
assumes *hb* : ∀ *x*. *insertion* (*nth-default 0* (*list-update L var x*)) *b* = (*B*::*real*)
assumes *hd* : ∀ *x*. *insertion* (*nth-default 0* (*list-update L var x*)) *d* = (*D*::*real*)
shows *aEval* (*Eq p*) (*list-update L var* ((*A*+*B*∗*C*)/*D*)) = *aEval* (*Eq*(*quadratic-part-1 var a b d* (*Eq p*))) (*list-update L var C*)
⟨*proof*⟩

lemma *quad-part-1-less* :
assumes *lLength* : *length L* > *var*
assumes *hdeg* : *MPoly-Type.degree* (*p*::*real mpoly*) *var* = (*deg*::*nat*)
assumes *nonzero* : *D* ≠ *0*
assumes *ha* : ∀ *x*. *insertion* (*nth-default 0* (*list-update L var x*)) *a* = (*A*::*real*)
assumes *hb* : ∀ *x*. *insertion* (*nth-default 0* (*list-update L var x*)) *b* = (*B*::*real*)
assumes *hd* : ∀ *x*. *insertion* (*nth-default 0* (*list-update L var x*)) *d* = (*D*::*real*)
shows *aEval* (*Less p*) (*list-update L var* ((*A*+*B*∗*C*)/*D*)) = *aEval* (*Less*(*quadratic-part-1 var a b d* (*Less p*))) (*list-update L var C*)
⟨*proof*⟩

lemma *quad-part-1-leq* :
assumes *lLength* : *length L > var*
assumes *hdeg* : *MPoly-Type.degree* (*p*::*real mpoly*) *var* = (*deg*::*nat*)
assumes *nonzero* : *D* ≠ *0*
assumes *ha* : ∀ *x*. *insertion* (*nth-default 0* (*list-update L var x*)) *a* = (*A*::*real*)
assumes *hb* : ∀ *x*. *insertion* (*nth-default 0* (*list-update L var x*)) *b* = (*B*::*real*)
assumes *hd* : ∀ *x*. *insertion* (*nth-default 0* (*list-update L var x*)) *d* = (*D*::*real*)
shows *aEval* (*Leq p*) (*list-update L var* ((*A*+*B*∗*C*)/*D*)) = *aEval* (*Leq*(*quadratic-part-1 var a b d* (*Leq p*))) (*list-update L var C*)
⟨*proof*⟩

lemma *quad-part-1-neq* :
assumes *lLength* : *length L > var*
assumes *hdeg* : *MPoly-Type.degree* (*p*::*real mpoly*) *var* = (*deg*::*nat*)
assumes *nonzero* : *D* ≠ *0*
assumes *ha* : ∀ *x*. *insertion* (*nth-default 0* (*list-update L var x*)) *a* = (*A*::*real*)
assumes *hb* : ∀ *x*. *insertion* (*nth-default 0* (*list-update L var x*)) *b* = (*B*::*real*)
assumes *hd* : ∀ *x*. *insertion* (*nth-default 0* (*list-update L var x*)) *d* = (*D*::*real*)
shows *aEval* (*Neq p*) (*list-update L var* ((*A*+*B*∗*C*)/*D*)) = *aEval* (*Neq*(*quadratic-part-1 var a b d* (*Neq p*))) (*list-update L var C*)
⟨*proof*⟩

lemma *sqrt-case* :
assumes *detGreater0* : *SQ* ≥ *0*
shows ((*SQ*^(*i div 2*)) ∗ *real* (*i mod 2*) ∗ *sqrt SQ* + *SQ* ^ (*i div 2*) ∗ (*1* − *real* (*i mod 2*))) = (*sqrt SQ*) ^ *i*
⟨*proof*⟩

lemma *sum-over-sqrt* :
assumes *detGreater0* : *SQ* ≥ *0*
shows (∑ *i*∈{*0*..<*n*+*1*}. ((*f i*::*real*) ∗ (*SQ*^(*i div 2*)) ∗ *real* (*i mod 2*) ∗ *sqrt SQ* +*f i* ∗ *SQ* ^ (*i div 2*) ∗ (*1* − *real* (*i mod 2*))))
=(∑ *i*∈{*0*..<*n*+*1*}. ((*f i*::*real*) ∗ ((*sqrt SQ*)^*i*)))
⟨*proof*⟩

lemma *quad-part-2-eq* :
assumes *lLength* : *length L > var*
assumes *detGreater0* : *SQ*≥*0*
assumes *hdeg* : *MPoly-Type.degree* (*p*::*real mpoly*) *var* = (*deg* ::*nat*)
assumes *hsq* : ∀ *x*. *insertion* (*nth-default 0* (*list-update L var x*)) *sq* = (*SQ*::*real*)
shows *aEval* (*Eq p*) (*list-update L var* (*sqrt SQ*)) = *aEval* (*Eq*(*quadratic-part-2 var sq p*)) (*list-update L var* (*sqrt SQ*))
⟨*proof*⟩

lemma *quad-part-2-less* :
assumes *lLength* : *length L* > *var*
assumes *detGreater0* : *SQ*≥*0*
assumes *hdeg* : *MPoly-Type.degree* (*p*::*real mpoly*) *var* = (*deg* ::*nat*)
assumes *hsq* : ∀ *x*. *insertion* (*nth-default 0* (*list-update L var x*)) *sq* = (*SQ*::*real*)
shows *aEval* (*Less p*) (*list-update L var* (*sqrt SQ*)) = *aEval* (*Less*(*quadratic-part-2 var sq p*)) (*list-update L var* (*sqrt SQ*))
⟨*proof*⟩

lemma *quad-part-2-neq* :
assumes *lLength* : *length L* > *var*
assumes *detGreater0* : *SQ*≥*0*
assumes *hdeg* : *MPoly-Type.degree* (*p*::*real mpoly*) *var* = (*deg* ::*nat*)
assumes *hsq* : ∀ *x*. *insertion* (*nth-default 0* (*list-update L var x*)) *sq* = (*SQ*::*real*)
shows *aEval* (*Neq p*) (*list-update L var* (*sqrt SQ*)) = *aEval* (*Neq*(*quadratic-part-2 var sq p*)) (*list-update L var* (*sqrt SQ*))
⟨*proof*⟩

lemma *quad-part-2-leq* :
assumes *lLength* : *length L* > *var*
assumes *detGreater0* : *SQ*≥*0*
assumes *hdeg* : *MPoly-Type.degree* (*p*::*real mpoly*) *var* = (*deg* ::*nat*)
assumes *hsq* : ∀ *x*. *insertion* (*nth-default 0* (*list-update L var x*)) *sq* = (*SQ*::*real*)
shows *aEval* (*Leq p*) (*list-update L var* (*sqrt SQ*)) = *aEval* (*Leq*(*quadratic-part-2 var sq p*)) (*list-update L var* (*sqrt SQ*))
⟨*proof*⟩

lemma *quad-part-2-deg* :
assumes *sqfree* : (*var*::*nat*)∉*vars*(*sq*::*real mpoly*)
shows *MPoly-Type.degree* (*quadratic-part-2 var sq p*) *var* ≤ *1*
⟨*proof*⟩

lemma *quad-equality-helper* :
assumes *lLength* : *length L* > *var*
assumes *detGreat0* : *Cv*≥*0*
assumes *hC* : ∀ *x*. *insertion* (*nth-default 0* (*list-update L var x*)) (*C*::*real mpoly*) = (*Cv*::*real*)
assumes *hA* : ∀ *x*. *insertion* (*nth-default 0* (*list-update L var x*)) (*A*::*real mpoly*) = (*Av*::*real*)
assumes *hB* : ∀ *x*. *insertion* (*nth-default 0* (*list-update L var x*)) (*B*::*real mpoly*) = (*Bv*::*real*)
shows *aEval* (*Eq* (*A* + *B* ∗ *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* ≠ *0*
assumes *detGreater0* : *Cv* ≥ *0*
assumes *freeC* : *var* ∉ *vars c*
assumes *ha* : ∀ *x*. *insertion* (*nth-default 0* (*list-update L var x*)) (*a*::*real mpoly*) = (*Av* :: *real*)
assumes *hb* : ∀ *x*. *insertion* (*nth-default 0* (*list-update L var x*)) (*b*::*real mpoly*) = (*Bv* :: *real*)
assumes *hc* : ∀ *x*. *insertion* (*nth-default 0* (*list-update L var x*)) (*c*::*real mpoly*) = (*Cv* :: *real*)
assumes *hd* : ∀ *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*≥*0*
assumes *hC* : ∀ *x*. *insertion* (*nth-default 0* (*list-update L var x*)) (*C*::*real mpoly*) = (*Cv*::*real*)
assumes *hA* : ∀ *x*. *insertion* (*nth-default 0* (*list-update L var x*)) (*A*::*real mpoly*) = (*Av*::*real*)
assumes *hB* : ∀ *x*. *insertion* (*nth-default 0* (*list-update L var x*)) (*B*::*real mpoly*) = (*Bv*::*real*)
shows *aEval* (*Less* (*A* + *B* ∗ *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* ≠ *0*
assumes *detGreater0* : *Cv* ≥ *0*
assumes *freeC* : *var* ∉ *vars c*
assumes *ha* : ∀ *x*. *insertion* (*nth-default 0* (*list-update L var x*)) (*a*::*real mpoly*) = (*Av* :: *real*)
assumes *hb* : ∀ *x*. *insertion* (*nth-default 0* (*list-update L var x*)) (*b*::*real mpoly*) = (*Bv* :: *real*)
assumes *hc* : ∀ *x*. *insertion* (*nth-default 0* (*list-update L var x*)) (*c*::*real mpoly*) = (*Cv* :: *real*)
assumes *hd* : ∀ *x*. *insertion* (*nth-default 0* (*list-update L var x*)) (*d*::*real mpoly*) = (*Dv* :: *real*)
shows *aEval* (*Less p*) (*list-update L var* ((*Av*+*Bv*∗*sqrt*(*Cv*))/*Dv*)) = *eval* (*quadratic-sub var a b c d* (*Less p*)) (*list-update L var* (*sqrt Cv*))

⟨*proof*⟩

lemma *quadratic-sub-leq-helper* :
  assumes *lLength* : *length L > var*
  assumes *detGreat0* : *Cv≥0*
  assumes *hC* : ∀ *x. insertion* (*nth-default 0* (*list-update L var x*)) (*C*::*real mpoly*) = (*Cv*::*real*)
  assumes *hA* : ∀ *x. insertion* (*nth-default 0* (*list-update L var x*)) (*A*::*real mpoly*) = (*Av*::*real*)
  assumes *hB* : ∀ *x. insertion* (*nth-default 0* (*list-update L var x*)) (*B*::*real mpoly*) = (*Bv*::*real*)
  shows *aEval* (*Leq* (*A* + *B* ∗ *Var var*)) (*list-update L var* (*sqrt Cv*)) =
  eval (*Or*(*And*(*Atom*(*Leq*(*A*)))(*Atom* (*Leq*(*B*^2∗*C*−*A*^2))))(*And* (*Atom*(*Leq B*)) (*Atom*(*Leq* (*A*^2−*B*^2∗*C*))))) (*list-update L var* (*sqrt Cv*))
⟨*proof*⟩

lemma *quadratic-sub-leq* :
  assumes *lLength* : *length L > var*
  assumes *nonzero* : *Dv ≠ 0*
  assumes *detGreater0* : *Cv ≥ 0*
  assumes *freeC* : *var ∉ vars c*
  assumes *ha* : ∀ *x. insertion* (*nth-default 0* (*list-update L var x*)) (*a*::*real mpoly*) = (*Av* :: *real*)
  assumes *hb* : ∀ *x. insertion* (*nth-default 0* (*list-update L var x*)) (*b*::*real mpoly*) = (*Bv* :: *real*)
  assumes *hc* : ∀ *x. insertion* (*nth-default 0* (*list-update L var x*)) (*c*::*real mpoly*) = (*Cv* :: *real*)
  assumes *hd* : ∀ *x. insertion* (*nth-default 0* (*list-update L var x*)) (*d*::*real mpoly*) = (*Dv* :: *real*)
 shows *aEval* (*Leq p*) (*list-update L var* ((*Av*+*Bv*∗*sqrt*(*Cv*))/*Dv*)) = *eval* (*quadratic-sub var a b c d* (*Leq p*)) (*list-update L var* (*sqrt Cv*))
⟨*proof*⟩

lemma *quadratic-sub-neq* :
  assumes *lLength* : *length L > var*
  assumes *nonzero* : *Dv ≠ 0*
  assumes *detGreater0* : *Cv ≥ 0*
  assumes *freeC* : *var ∉ vars c*
  assumes *ha* : ∀ *x. insertion* (*nth-default 0* (*list-update L var x*)) (*a*::*real mpoly*) = (*Av* :: *real*)
  assumes *hb* : ∀ *x. insertion* (*nth-default 0* (*list-update L var x*)) (*b*::*real mpoly*) = (*Bv* :: *real*)
  assumes *hc* : ∀ *x. insertion* (*nth-default 0* (*list-update L var x*)) (*c*::*real mpoly*) = (*Cv* :: *real*)
  assumes *hd* : ∀ *x. insertion* (*nth-default 0* (*list-update L var x*)) (*d*::*real mpoly*) = (*Dv* :: *real*)
 shows *aEval* (*Neq p*) (*list-update L var* ((*Av*+*Bv*∗*sqrt*(*Cv*))/*Dv*)) = *eval* (*quadratic-sub var a b c d* (*Neq p*)) (*list-update L var* (*sqrt Cv*))

⟨*proof*⟩

theorem *free-in-quad* :
assumes *freeA* : *var*∉ *vars a*
assumes *freeB* : *var*∉ *vars b*
assumes *freeC* : *var*∉ *vars c*
assumes *freeD* : *var*∉ *vars d*
shows *freeIn var* (*quadratic-sub var a b c d A*)
⟨*proof*⟩

theorem *quadratic-sub* :
assumes *lLength* : *length L* > *var*
assumes *nonzero* : *Dv* ≠ *0*
assumes *detGreater0* : *Cv* ≥ *0*
assumes *freeC* : *var* ∉ *vars c*
assumes *ha* : ∀ *x*. *insertion* (*nth-default 0* (*list-update L var x*)) (*a*::*real mpoly*) = (*Av* :: *real*)
assumes *hb* : ∀ *x*. *insertion* (*nth-default 0* (*list-update L var x*)) (*b*::*real mpoly*) = (*Bv* :: *real*)
assumes *hc* : ∀ *x*. *insertion* (*nth-default 0* (*list-update L var x*)) (*c*::*real mpoly*) = (*Cv* :: *real*)
assumes *hd* : ∀ *x*. *insertion* (*nth-default 0* (*list-update L var x*)) (*d*::*real mpoly*) = (*Dv* :: *real*)
shows *aEval A* (*list-update L var* ((*Av*+*Bv***sqrt*(*Cv*))/*Dv*)) = *eval* (*quadratic-sub var a b c d A*) (*list-update L var* (*sqrt Cv*))
⟨*proof*⟩

lemma *free-in-quad-fm-helper* :
assumes *freeA* : *var*∉ *vars a*
assumes *freeB* : *var*∉ *vars b*
assumes *freeC* : *var*∉ *vars c*
assumes *freeD* : *var*∉ *vars d*
shows *freeIn* (*var*+*z*) (*quadratic-sub-fm-helper var a b c d F z*)
⟨*proof*⟩

theorem *free-in-quad-fm* :
assumes *freeA* : *var*∉ *vars a*
assumes *freeB* : *var*∉ *vars b*
assumes *freeC* : *var*∉ *vars c*
assumes *freeD* : *var*∉ *vars d*
shows *freeIn var* (*quadratic-sub-fm var a b c d A*)
⟨*proof*⟩

lemma *quadratic-sub-fm-helper* :

assumes *nonzero* : *Dv* ≠ *0*
assumes *detGreater0* : *Cv* ≥ *0*
assumes *freeC* : *var* ∉ *vars c*
assumes *lLength* : *length L* > *var+z*
assumes *ha* : ∀ *x*. *insertion* (*nth-default 0* (*list-update* (*drop z L*) *var x*)) (*a*::*real mpoly*) = (*Av* :: *real*)
assumes *hb* : ∀ *x*. *insertion* (*nth-default 0* (*list-update* (*drop z L*) *var x*)) (*b*::*real mpoly*) = (*Bv* :: *real*)
assumes *hc* : ∀ *x*. *insertion* (*nth-default 0* (*list-update* (*drop z L*) *var x*)) (*c*::*real mpoly*) = (*Cv* :: *real*)
assumes *hd* : ∀ *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* ≠ *0*
assumes *detGreater0* : *Cv* ≥ *0*
assumes *freeC* : *var* ∉ *vars c*
assumes *ha* : ∀ *x*. *insertion* (*nth-default 0* (*list-update L var x*)) (*a*::*real mpoly*) = (*Av* :: *real*)
assumes *hb* : ∀ *x*. *insertion* (*nth-default 0* (*list-update L var x*)) (*b*::*real mpoly*) = (*Bv* :: *real*)
assumes *hc* : ∀ *x*. *insertion* (*nth-default 0* (*list-update L var x*)) (*c*::*real mpoly*) = (*Cv* :: *real*)
assumes *hd* : ∀ *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* ∈ *set*(*L*)
assumes *low-pow* : *MPoly-Type.degree p var* = *1* ∨ *MPoly-Type.degree p var* = *2*
shows ((∃ *x*. *eval* (*list-conj* (*map fm.Atom L* @ *F*)) (*xs* @ *x* # Γ)) =

$((\exists x.\ eval\ (elimVar\ var\ L\ F\ (Eq\ p))\ (xs\ @\ x\ \#\ \Gamma))) \lor (\forall x.\ aEval\ (Eq\ p)\ (xs\ @\ x\ \#\ \Gamma)))$
⟨*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* ∈ *set*(*L*)
  assumes *low-pow* : *MPoly-Type.degree p var* = *1* ∨ *MPoly-Type.degree p var* = *2*
  assumes *nonzero* : ∀ *x*.
    insertion (*nth-default 0* (*xs* @ *x* # Γ)) (*isolate-variable-sparse p var 2*) ≠ *0*
    ∨ *insertion* (*nth-default 0* (*xs* @ *x* # Γ)) (*isolate-variable-sparse p var 1*) ≠ *0*
    ∨ *insertion* (*nth-default 0* (*xs* @ *x* # Γ)) (*isolate-variable-sparse p var 0*) ≠ *0* (**is** *?non0*)
  shows (∃ *x*. *eval* (*list-conj* (*map fm.Atom L* @ *F*)) (*xs* @ *x* # Γ)) =
    (∃ *x*. *eval* (*elimVar var L F* (*Eq p*)) (*xs* @ *x* # Γ))
⟨*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 (∃ *x*. (*eval* (*list-conj* ((*map Atom L*) @ *F*)) (*xs* @ (*x*#Γ)))) = (∃ *x*.(*eval F′* (*xs* @ (*x*#Γ))))
⟨*proof*⟩

lemma *luckyFind′-eval* :
  assumes *length xs* = *var*

shows $(\exists x.\ eval\ (list\text{-}conj\ (map\ fm.Atom\ L\ @\ F))\ (xs\ @\ x\ \#\ \Gamma)) = (\exists x.\ eval\ (luckyFind'\ var\ L\ F)\ (xs\ @\ x\ \#\ \Gamma))$
⟨*proof*⟩

lemma *luckiestFind-eval* :
assumes *length xs* = *var*
shows $(\exists x.\ eval\ (list\text{-}conj\ (map\ fm.Atom\ L\ @\ F))\ (xs\ @\ x\ \#\ \Gamma)) = (\exists x.\ eval\ (luckiestFind\ var\ L\ F)\ (xs\ @\ x\ \#\ \Gamma))$
⟨*proof*⟩

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 $\forall\, p\in set(A).\ MPoly\text{-}Type.degree\ p\ var = 1 \lor MPoly\text{-}Type.degree\ p\ var = 2$ ⟨*proof*⟩

lemma *list-in-find-eq* :
assumes *find-eq var L* = (A,L')
shows $set(map\ Eq\ A\ @\ L') = set\ L$⟨*proof*⟩

lemma *qe-eq-one-eval* :
assumes *hlength* : *length xs* = *var*
shows $(\exists x.\ (eval\ (list\text{-}conj\ ((map\ Atom\ L)\ @\ F))\ (xs\ @\ (x\#\Gamma)))) = (\exists x.(eval\ (qe\text{-}eq\text{-}one\ var\ L\ F)\ (xs\ @\ (x\#\Gamma))))$
⟨*proof*⟩

lemma *qe-eq-repeat-helper-eval-case1* :
assumes *hlength* : *length xs* = *var*
assumes *degreeGood* : $\forall\, p\in set(A).\ MPoly\text{-}Type.degree\ p\ var = 1 \lor MPoly\text{-}Type.degree\ p\ var = 2$
shows $((eval\ (list\text{-}conj\ ((map\ (Atom\ o\ Eq)\ \ A)\ @\ (map\ Atom\ L)\ @\ F))\ (xs\ @\ (x\#\Gamma))))$
$\Longrightarrow (eval\ (qe\text{-}eq\text{-}repeat\text{-}helper\ var\ A\ L\ F)\ (xs\ @\ x\ \#\ \Gamma))$
⟨*proof*⟩

lemma *qe-eq-repeat-helper-eval-case2* :
assumes *hlength* : *length xs* = *var*
assumes *degreeGood* : ∀ *p*∈*set*(*A*). *MPoly-Type.degree p var* = *1* ∨ *MPoly-Type.degree p var* = *2*
shows (*eval* (*qe-eq-repeat-helper var A L F*) (*xs* @ *x* # Γ))
⟹ ∃ *x*. ((*eval* (*list-conj* ((*map* (*Atom o Eq*) *A*) @ (*map Atom L*) @ *F*)) (*xs* @ (*x*#Γ))))
⟨*proof*⟩

lemma *qe-eq-repeat-eval* :
assumes *hlength* : *length xs* = *var*
shows (∃ *x*. (*eval* (*list-conj* ((*map Atom L*) @ *F*)) (*xs* @ (*x*#Γ)))) = (∃ *x*.(*eval* (*qe-eq-repeat var L F*) (*xs* @ (*x*#Γ))))
⟨*proof*⟩

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* ⇒ *real* ⇒ *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* ≤ *0*) |
aEvalUni (*NeqUni* (*a*,*b*,*c*)) *x* = (*a*∗*x*^*2*+*b*∗*x*+*c* ≠ *0*)

fun *aNegUni* :: *atomUni* ⇒ *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*) $\wedge$ (*evalUni* ψ *x*)) |
evalUni (*OrUni* φ ψ) *x* = ((*evalUni* φ *x*) $\vee$ (*evalUni* ψ *x*))

fun *negUni* :: *atomUni fmUni* $\Rightarrow$ *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* $\Rightarrow$ *real mpoly* $\Rightarrow$ *real list* $\Rightarrow$ (*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* $\Rightarrow$ *atom* $\Rightarrow$ *real list* $\Rightarrow$ *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* = (∃ *l*∈*set*(*L*). *evalUni l x*)
⟨*proof*⟩

lemma *eval-list-conj-Uni* : *evalUni* (*list-conj-Uni A*) *x* = (∀ *l*∈*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* ∧ *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 ∃ *F'*. *convert-fm var F xs* = *Some F'*
shows *liftmap f F 0* = *map-atom* (*f 0*) *F*
⟨*proof*⟩

lemma *eval-map* : (∃ *l*∈*set*(*map f L*). *evalUni l x*) = (∃ *l*∈*set*(*L*). *evalUni* (*f l*) *x*)
⟨*proof*⟩

lemma *eval-map-all* : (∀ *l*∈*set*(*map f L*). *evalUni l x*) = (∀ *l*∈*set*(*L*). *evalUni* (*f l*) *x*)
⟨*proof*⟩

lemma *eval-append* : (∃ *l*∈*set* (*A*#*B*).*evalUni l x*) = (*evalUni A x* ∨ (∃ *l*∈*set* (*B*).*evalUni l x*))

⟨*proof*⟩

lemma *eval-conj-atom* : *evalUni* (*list-conj-Uni* (*map AtomUni L*)) *x* = (∀ *l*∈*set*(*L*). *aEvalUni l x*)
⟨*proof*⟩
end

9.2 Negative Infinity

theory *NegInfinity*
imports *HOL−Analysis.Poly-Roots VSAlgos*
begin

lemma *freeIn-allzero* : *freeIn var* (*allZero p var*)
⟨*proof*⟩

lemma *allzero-eval* :
assumes *lLength* : *var* < *length L*
shows(∃ *x*. ∀ *y*<*x*. *aEval* (*Eq p*) (*list-update L var y*)) = (∀ *x*. *eval* (*allZero p var*) (*list-update L var x*))
⟨*proof*⟩

lemma *freeIn-altNegInf* : *freeIn var* (*alternateNegInfinity p var*)
⟨*proof*⟩

theorem *freeIn-substNegInfinity* : *freeIn var* (*substNegInfinity var A*)
⟨*proof*⟩

end
theory *NegInfinityUni*
imports *UniAtoms NegInfinity QE*
begin

fun *allZero′* :: *real* ∗ *real* ∗ *real* ⇒ *atomUni fmUni* **where**
allZero′ (*a*,*b*,*c*) = *AndUni*(*AndUni*(*AtomUni*(*EqUni*(*0*,*0*,*a*))) (*AtomUni*(*EqUni*(*0*,*0*,*b*)))) (*AtomUni*(*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*∗*w*^*2* + *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 (∃ *w*. *x* ≤ *w* ∧ *w* ≤ *y* ∧ *a*∗*w*^*2* + *b*∗*w* + *c* = *0*)
⟨*proof*⟩

lemma *no-change-eval-lt*:
  fixes *x y*:: *real*
  assumes $x < y$
  assumes $\neg(\exists w.\ x \leq w \wedge w \leq y \wedge a*w\hat{\ }2 + b*w + c = 0)$
  shows ((*aEvalUni* (*LessUni* (*a*,*b*,*c*)) *x* = *aEvalUni* (*LessUni* (*a*,*b*,*c*)) *y*))
  ⟨*proof*⟩

lemma *change-eval-neq*:
  fixes *x y*:: *real*
  assumes ((*aEvalUni* (*NeqUni* (*a*,*b*,*c*)) *x* ≠ *aEvalUni* (*NeqUni* (*a*,*b*,*c*)) *y*) ∧ *x* < *y*)
  shows $(\exists w.\ x \leq w \wedge w \leq y \wedge a*w\hat{\ }2 + b*w + c = 0)$
  ⟨*proof*⟩

lemma *change-eval-leq*:
  fixes *x y*:: *real*
  assumes ((*aEvalUni* (*LeqUni* (*a*,*b*,*c*)) *x* ≠ *aEvalUni* (*LeqUni* (*a*,*b*,*c*)) *y*) ∧ *x* < *y*)
  shows $(\exists w.\ x \leq w \wedge w \leq y \wedge a*w\hat{\ }2 + b*w + c = 0)$
⟨*proof*⟩

lemma *change-eval*:
  fixes *x y*:: *real*
  fixes *At*:: *atomUni*
  assumes *xlt*: $x < y$
  assumes *noteq*: ((*aEvalUni At*) *x* ≠ *aEvalUni* (*At*) *y*)
  assumes *getPoly At* = (*a*, *b*, *c*)
  shows $(\exists w.\ x \leq w \wedge w \leq y \wedge a*w\hat{\ }2 + b*w + c = 0)$
⟨*proof*⟩

lemma *no-change-eval*:
  fixes *x y*:: *real*
  assumes *getPoly At* = (*a*, *b*, *c*)
  assumes $x < y$
  assumes $\neg(\exists w.\ x \leq w \wedge w \leq y \wedge a*w\hat{\ }2 + b*w + c = 0)$
  shows ((*aEvalUni At*) *x* = *aEvalUni* (*At*) *y* ∧ *x* < *y*)
  ⟨*proof*⟩

lemma *same-eval''* :
  assumes *getPoly At* = (*a*, *b*, *c*)
  assumes *nonz*: $a \neq 0 \vee b \neq 0 \vee c \neq 0$
  shows ∃ *x*. ∀ *y*<*x*. (*aEvalUni At y* = *aEvalUni At x*)
⟨*proof*⟩

lemma *inequality-case* : (∃ (*x*::*real*). ∀ (*y*::*real*)<*x*. (*a*::*real*) $*\ y^2$ + (*b*::*real*) ∗ *y* +

$(c::real) < 0) =$
$(a < 0 \vee a = 0 \wedge (0 < b \vee b = 0 \wedge c < 0))$
⟨*proof*⟩

lemma *inequality-case-geq* : $(\exists (x::real). \forall (y::real)<x. (a::real) * y^2 + (b::real) * y + (c::real) > 0) =$
$(a > 0 \vee a = 0 \wedge (0 > b \vee b = 0 \wedge c > 0))$
⟨*proof*⟩

lemma *infinity-evalUni-LessUni* : $(\exists x. \forall y<x.$ *aEvalUni* (*LessUni p*) *y*) = (*evalUni* (*substNegInfinityUni* (*LessUni p*)) *x*)
⟨*proof*⟩

lemma *infinity-evalUni-EqUni* : $(\exists x. \forall y<x.$ *aEvalUni* (*EqUni p*) *y*) = (*evalUni* (*substNegInfinityUni* (*EqUni p*)) *x*)
⟨*proof*⟩

lemma *infinity-evalUni-NeqUni* : $(\exists x. \forall y<x.$ *aEvalUni* (*NeqUni p*) *y*) = (*evalUni* (*substNegInfinityUni* (*NeqUni p*)) *x*)
⟨*proof*⟩

lemma *infinity-evalUni-LeqUni* : $(\exists x. \forall y<x.$ *aEvalUni* (*LeqUni p*) *y*) = (*evalUni* (*substNegInfinityUni* (*LeqUni p*)) *x*)
⟨*proof*⟩

This is the vertical translation for substNegInfinityUni where we represent the virtual substution of negative infinity in the univariate case

lemma *infinity-evalUni* :
shows $(\exists x. \forall y<x.$ *aEvalUni At y*) = (*evalUni* (*substNegInfinityUni At*) *x*)
⟨*proof*⟩

end

9.3 Infinitesimals

theory *Infinitesimals*
imports *ExecutiblePolyProps 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*)
⟨*proof*⟩

lemma *freeIn-substInfinitesimalQuadratic-fm* : **assumes** $var \notin vars\ a$
$var \notin vars\ b$

var ∉ *vars c*
var ∉ *vars d*
shows*freeIn var* (*substInfinitesimalQuadratic-fm var a b c d F*)
⟨*proof*⟩

lemma *freeIn-substInfinitesimalLinear*:
assumes *var* ∉ *vars a var* ∉ *vars b*
shows *freeIn var* (*substInfinitesimalLinear var a b At*)
⟨*proof*⟩

lemma *freeIn-substInfinitesimalLinear-fm*:
assumes *var* ∉ *vars a var* ∉ *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* ⇒ *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* ⇒ *real* ⇒ *atomUni* ⇒ *atomUni fmUni* **where**
linearSubstitutionUni b c a = (*if aEvalUni a* (−*c*/*b*) *then TrueFUni else FalseFUni*)

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*
(λ*x*. λ*A*. *Atom*(*linear-substitution* (*var*+*x*) (*liftPoly 0 x* (−*c*)) (*liftPoly 0 x b*) *A*))
(*convertDerivative var p*)
0) (*xs*′@*x*#*xs*) =
evalUni (*map-atomUni* (*linearSubstitutionUni B C*) (*convertDerivativeUni p*′)) *x*
⟨*proof*⟩
lemma *convert-substInfinitesimalLinear*:
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 *eval* (*substInfinitesimalLinear var* (−*c*) *b a*) (*xs*′@*x*#*xs*) = *evalUni* (*substInfinitesimalLinearUni B C a*′) *x*
⟨*proof*⟩

lemma *either-or*:
fixes *r* :: *real*
assumes *a*: (∃ *y*′>*r*. ∀ *x*∈{*r*<..*y*′}. (*aEvalUni* (*EqUni* (*a*, *b*, *c*)) *x*) ∨ (*aEvalUni* (*LessUni* (*a*, *b*, *c*)) *x*))
shows (∃ *y*′>*r*. ∀ *x*∈{*r*<..*y*′}. (*aEvalUni* (*EqUni* (*a*, *b*, *c*)) *x*)) ∨
(∃ *y*′>*r*. ∀ *x*∈{*r*<..*y*′}. (*aEvalUni* (*LessUni* (*a*, *b*, *c*)) *x*))
⟨*proof*⟩

lemma *infinitesimal-linear*′*-helper* :
assumes *at-is*: *At* = *LessUni p* ∨ *At* = *EqUni p*
assumes *B* ≠ *0*
shows ((∃ *y*′::*real*>−*C*/*B*. ∀ *x*::*real* ∈{−*C*/*B*<..*y*′}. *aEvalUni At x*)
= *evalUni* (*substInfinitesimalLinearUni B C At*) *x*)
⟨*proof*⟩

lemma *infinitesimal-linear*′ :
assumes *B* ≠ *0*
shows (∃ *y*′::*real*>−*C*/*B*. ∀ *x*::*real* ∈{−*C*/*B*<..*y*′}. *aEvalUni At x*)
= *evalUni* (*substInfinitesimalLinearUni B C At*) *x*
⟨*proof*⟩

fun *quadraticSubUni* :: *real* ⇒ *real* ⇒ *real* ⇒ *real* ⇒ *atomUni* ⇒ *atomUni fmUni* **where**
quadraticSubUni a b c d A = (*if aEvalUni A* ((*a*+*b*∗*sqrt*(*c*))/*d*) *then TrueFUni else FalseFUni*)

fun *substInfinitesimalQuadraticUni* :: *real* ⇒ *real* ⇒ *real* ⇒ *real* ⇒ *atomUni* ⇒ *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*: $D \neq (0::real)$
shows
$((a'::real) * (((A::real) + (B::real) * sqrt\ (C::real)) / (D::real))^2 + (b'::real) * (A + B * sqrt\ C) / D + c' < 0 \lor$
$a' * ((A + B * sqrt\ C) / D)^2 + b' * (A + B * sqrt\ C) / D + (c'::real) = 0 \land$
$(b' + a' * (A + B * sqrt\ C) * 2 / D < 0 \lor$
$b' + a' * (A + B * sqrt\ C) * 2 / D = 0 \land 2 * a' < 0)) =$
$(a' * ((A + B * sqrt\ C) / D)^2 + b' * (A + B * sqrt\ C) / D + c' < 0 \lor$
$a' * ((A + B * sqrt\ C) / D)^2 + b' * (A + B * sqrt\ C) / D + c' = 0 \land$
$(2 * a' * (A + B * sqrt\ C) / D + b' < 0 \lor$
$2 * a' * (A + B * sqrt\ C) / D + b' = 0 \land a' < 0))$
⟨*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 $D \neq 0$
assumes $0 \leq C$
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 $D \neq 0$
assumes $0 \leq C$

assumes $var \notin (vars\ a)$
assumes $var \notin (vars\ b)$
assumes $var \notin (vars\ c)$
assumes $var \notin (vars\ d)$
assumes $length\ xs' = var$
shows $eval\ (substInfinitesimalQuadratic\ var\ a\ b\ c\ d\ At)\ (xs'@\ x\#xs) = evalUni\ (substInfinitesimalQuadraticUni\ A\ B\ C\ D\ At')\ x$
$\langle proof \rangle$

lemma *infinitesimal-quad-helper*:
fixes $A\ B\ C\ D::\ real$
assumes *at-is*: $At = LessUni\ p \lor At = EqUni\ p$
assumes $D \neq 0$
assumes $C \geq 0$
shows $(\exists\, y'::real > ((A+B * sqrt(C))/(D)).\ \forall\, x::real \in \{((A+B * sqrt(C))/(D)) <..y'\}.\ aEvalUni\ At\ x)$
$= (evalUni\ (substInfinitesimalQuadraticUni\ A\ B\ C\ D\ At)\ x)$
$\langle proof \rangle$

lemma *infinitesimal-quad*:
fixes $A\ B\ C\ D::\ real$
assumes $D \neq 0$
assumes $C \geq 0$
shows $(\exists\, y'::real > ((A+B * sqrt(C))/(D)).\ \forall\, x::real \in \{((A+B * sqrt(C))/(D)) <..y'\}.\ aEvalUni\ At\ x)$
$= (evalUni\ (substInfinitesimalQuadraticUni\ A\ B\ C\ D\ At)\ x)$
$\langle proof \rangle$

end

9.4 Overall General VS Proofs

theory *DNFUni*
imports *QE InfinitesimalsUni*
begin

fun $DNFUni :: atomUni\ fmUni \Rightarrow atomUni\ list\ list$ **where**
$DNFUni\ (AtomUni\ a) = [[a]]|$
$DNFUni\ (TrueFUni) = [[]]\ |$
$DNFUni\ (FalseFUni) = []|$
$DNFUni\ (AndUni\ A\ B) = [A'\ @\ B'.\ A' \leftarrow DNFUni\ A,\ B' \leftarrow DNFUni\ B]|$
$DNFUni\ (OrUni\ A\ B) = DNFUni\ A\ @\ DNFUni\ B$

lemma *eval-DNFUni* : $evalUni\ F\ x = evalUni\ (list\text{-}disj\text{-}Uni(map\ (list\text{-}conj\text{-}Uni\ o\ (map\ AtomUni))\ (DNFUni\ F)))\ x$
$\langle proof \rangle$

fun $elimVarUni\text{-}atom :: atomUni\ list \Rightarrow atomUni \Rightarrow atomUni\ fmUni$ **where**

```
  elimVarUni-atom F (EqUni (a,b,c)) =
(OrUni
  (AndUni
    (AndUni (AtomUni (EqUni (0,0,a))) (AtomUni (NeqUni (0,0,b))))
      (list-conj-Uni (map (linearSubstitutionUni b c) F)))
   (AndUni (AtomUni (NeqUni (0,0,a))) (AndUni (AtomUni(LeqUni (0,0,−(b^2)+4∗a∗c)))
      (OrUni
        (list-conj-Uni (map (quadraticSubUni (−b) 1 (b^2−4∗a∗c) (2∗a)) F))
        (list-conj-Uni (map (quadraticSubUni (−b) (−1) (b^2−4∗a∗c) (2∗a)) F))
      )
    )
  )
)
 |
  elimVarUni-atom F (LeqUni (a,b,c)) =
(OrUni
  (AndUni
    (AndUni (AtomUni (EqUni (0,0,a))) (AtomUni (NeqUni (0,0,b))))
      (list-conj-Uni (map (linearSubstitutionUni b c) F)))
   (AndUni (AtomUni (NeqUni (0,0,a))) (AndUni (AtomUni(LeqUni (0,0,−(b^2)+4∗a∗c)))
      (OrUni
        (list-conj-Uni (map (quadraticSubUni (−b) 1 (b^2−4∗a∗c) (2∗a)) F))
        (list-conj-Uni (map (quadraticSubUni (−b) (−1) (b^2−4∗a∗c) (2∗a)) F))
      )
    )
  )
)
 |
  elimVarUni-atom F (LessUni (a,b,c)) =
(OrUni
  (AndUni
    (AndUni (AtomUni (EqUni (0,0,a))) (AtomUni (NeqUni (0,0,b))))
      (list-conj-Uni (map (substInfinitesimalLinearUni b c) F)))
   (AndUni (AtomUni (NeqUni (0,0,a))) (AndUni (AtomUni(LeqUni (0,0,−(b^2)+4∗a∗c)))
      (OrUni
          (list-conj-Uni (map(substInfinitesimalQuadraticUni (−b) 1 (b^2−4∗a∗c)
(2∗a)) F))
        (list-conj-Uni (map(substInfinitesimalQuadraticUni (−b) (−1) (b^2−4∗a∗c)
(2∗a)) F))
      )
    )
  )
)
|
  elimVarUni-atom F (NeqUni (a,b,c)) =
(OrUni
  (AndUni
    (AndUni (AtomUni (EqUni (0,0,a))) (AtomUni (NeqUni (0,0,b))))
      (list-conj-Uni (map (substInfinitesimalLinearUni b c) F)))
```

(*AndUni* (*AtomUni* (*NeqUni* (*0*,*0*,*a*))) (*AndUni* (*AtomUni*(*LeqUni* (*0*,*0*,−(*b*^*2*)+*4*∗*a*∗*c*)))
(*OrUni*
(*list-conj-Uni* (*map*(*substInfinitesimalQuadraticUni* (−*b*) *1* (*b*^*2*−*4*∗*a*∗*c*) (*2*∗*a*)) *F*))
(*list-conj-Uni* (*map*(*substInfinitesimalQuadraticUni* (−*b*) (−*1*) (*b*^*2*−*4*∗*a*∗*c*) (*2*∗*a*)) *F*))
)
)
)
)

fun *generalVS-DNF* :: *atomUni list* ⇒ *atomUni fmUni* **where**
generalVS-DNF L = *list-disj-Uni* (*list-conj-Uni*(*map substNegInfinityUni L*) # (*map* (λ*A*. *elimVarUni-atom L A*) *L*))

end
theory *GeneralVSProofs*
imports *DNFUni EqualityVS VSAlgos*
begin

fun *separateAtoms* :: *atomUni list* ⇒ (*real* ∗ *real* ∗ *real*) *list* ∗ (*real* ∗ *real* ∗ *real*) *list* ∗ (*real* ∗ *real* ∗ *real*) *list* ∗ (*real* ∗ *real* ∗ *real*) *list* **where**
separateAtoms [] = ([],[],[],[])|
separateAtoms (*EqUni p* # *L*) = (*let* (*a*,*b*,*c*,*d*) = *separateAtoms*(*L*) *in* (*p*#*a*,*b*,*c*,*d*))|
separateAtoms (*LessUni p* # *L*) = (*let* (*a*,*b*,*c*,*d*) = *separateAtoms*(*L*) *in* (*a*,*p*#*b*,*c*,*d*))|
separateAtoms (*LeqUni p* # *L*) = (*let* (*a*,*b*,*c*,*d*) = *separateAtoms*(*L*) *in* (*a*,*b*,*p*#*c*,*d*))|
separateAtoms (*NeqUni p* # *L*) = (*let* (*a*,*b*,*c*,*d*) = *separateAtoms*(*L*) *in* (*a*,*b*,*c*,*p*#*d*))

lemma *separate-aEval* :
assumes *separateAtoms L* = (*a*,*b*,*c*,*d*)
shows (∀ *l*∈*set L*. *aEvalUni l x*) =
((∀ (*a*,*b*,*c*)∈*set a*. *a*∗*x*^*2*+*b*∗*x*+*c*=*0*) ∧ (∀ (*a*,*b*,*c*)∈*set b*. *a*∗*x*^*2*+*b*∗*x*+*c*<*0*) ∧
(∀ (*a*,*b*,*c*)∈*set c*. *a*∗*x*^*2*+*b*∗*x*+*c*≤*0*) ∧ (∀ (*a*,*b*,*c*)∈*set d*. *a*∗*x*^*2*+*b*∗*x*+*c*≠*0*))
⟨*proof*⟩

lemma *splitAtoms-negInfinity* :
assumes *separateAtoms L* = (*a*,*b*,*c*,*d*)
shows (∀ *l*∈*set L*. *evalUni* (*substNegInfinityUni l*) *x*) = (
(∀ (*a*,*b*,*c*)∈*set a*.(∃ *x*. ∀ *y*<*x*. *a*∗*y*^*2*+*b*∗*y*+*c*=*0*))∧
(∀ (*a*,*b*,*c*)∈*set b*.(∃ *x*. ∀ *y*<*x*. *a*∗*y*^*2*+*b*∗*y*+*c*<*0*))∧
(∀ (*a*,*b*,*c*)∈*set c*.(∃ *x*. ∀ *y*<*x*. *a*∗*y*^*2*+*b*∗*y*+*c*≤*0*))∧

$(\forall (a,b,c)\in set\ d.(\exists x.\ \forall y<x.\ a*y\hat{\ }2+b*y+c\neq 0)))$
⟨*proof*⟩

lemma *set-split* :
assumes *separateAtoms L* = (*eq*,*les*,*leq*,*neq*)
shows *set L* = *set* (*map EqUni eq* @ *map LessUni les* @ *map LeqUni leq* @ *map NeqUni neq*)
⟨*proof*⟩

lemma *set-split′* : **assumes** *separateAtoms L* = (*eq*,*les*,*leq*,*neq*)
shows *set* (*map P L*) = *set* (*map* (*P o EqUni*) *eq* @ *map* (*P o LessUni*) *les* @ *map* (*P o LeqUni*) *leq* @ *map* (*P o NeqUni*) *neq*)
⟨*proof*⟩

lemma *split-elimVar* :
assumes *separateAtoms L* = (*eq*,*les*,*leq*,*neq*)
shows (∃ *l*∈*set L*. *evalUni* (*elimVarUni-atom L′ l*) *x*) =
((∃ (*a′*,*b′*,*c′*)∈*set eq*. (*evalUni* (*elimVarUni-atom L′* (*EqUni*(*a′*,*b′*,*c′*))) *x*))
∨ (∃ (*a′*,*b′*,*c′*)∈*set les*.
(*evalUni* (*elimVarUni-atom L′* (*LessUni*(*a′*,*b′*,*c′*))) *x*))
∨ (∃ (*a′*,*b′*,*c′*)∈*set leq*.
(*evalUni* (*elimVarUni-atom L′* (*LeqUni*(*a′*,*b′*,*c′*))) *x*))
∨ (∃ (*a′*,*b′*,*c′*)∈*set neq*.
(*evalUni* (*elimVarUni-atom L′* (*NeqUni*(*a′*,*b′*,*c′*))) *x*)))
⟨*proof*⟩

lemma *split-elimvar* :
assumes *separateAtoms L* = (*eq*,*les*,*leq*,*neq*)
shows *evalUni* (*elimVarUni-atom L At*) *x* = *evalUni* (*elimVarUni-atom* ((*map EqUni eq*)@(*map LessUni les*) @ *map LeqUni leq* @ *map NeqUni neq*) *At*) *x*
⟨*proof*⟩

lemma *less* :
((*a′* = *0* ∧ *b′* ≠ *0*) ∧
(∀ (*d*, *e*, *f*)∈*set a*. *evalUni* (*substInfinitesimalLinearUni b′ c′* (*EqUni* (*d*, *e*, *f*))) *x*) ∧
(∀ (*d*, *e*, *f*)∈*set b*. *evalUni* (*substInfinitesimalLinearUni b′ c′* (*LessUni* (*d*, *e*, *f*))) *x*) ∧
(∀ (*d*, *e*, *f*)∈*set c*. *evalUni* (*substInfinitesimalLinearUni b′ c′* (*LeqUni* (*d*, *e*, *f*))) *x*) ∧
(∀ (*d*, *e*, *f*)∈*set d*. *evalUni* (*substInfinitesimalLinearUni b′ c′* (*NeqUni* (*d*, *e*, *f*))) *x*) ∨
$a' \neq 0$ ∧
$-\ b'^2 + 4 * a' * c' \leq 0$ ∧
((∀ (*d*, *e*, *f*)∈*set a*.
evalUni

(substInfinitesimalQuadraticUni (− b′) 1 (b′² − 4 ∗ a′ ∗ c′) (2 ∗ a′)
(EqUni (d, e, f)))
x) ∧
(∀ (d, e, f)∈set b.
evalUni
(substInfinitesimalQuadraticUni (− b′) 1 (b′² − 4 ∗ a′ ∗ c′) (2 ∗ a′)
(LessUni (d, e, f)))
x) ∧
(∀ (d, e, f)∈set c.
evalUni
(substInfinitesimalQuadraticUni (− b′) 1 (b′² − 4 ∗ a′ ∗ c′) (2 ∗ a′)
(LeqUni (d, e, f)))
x) ∧
(∀ (d, e, f)∈set d.
evalUni
(substInfinitesimalQuadraticUni (− b′) 1 (b′² − 4 ∗ a′ ∗ c′) (2 ∗ a′)
(NeqUni (d, e, f)))
x) ∨
(∀ (d, e, f)∈set a.
evalUni
(substInfinitesimalQuadraticUni (− b′) (− 1) (b′² − 4 ∗ a′ ∗ c′) (2 ∗
a′)
(EqUni (d, e, f)))
x) ∧
(∀ (d, e, f)∈set b.
evalUni
(substInfinitesimalQuadraticUni (− b′) (− 1) (b′² − 4 ∗ a′ ∗ c′) (2 ∗
a′)
(LessUni (d, e, f)))
x) ∧
(∀ (d, e, f)∈set c.
evalUni
(substInfinitesimalQuadraticUni (− b′) (− 1) (b′² − 4 ∗ a′ ∗ c′) (2 ∗
a′)
(LeqUni (d, e, f)))
x) ∧
(∀ (d, e, f)∈set d.
evalUni
(substInfinitesimalQuadraticUni (− b′) (− 1) (b′² − 4 ∗ a′ ∗ c′) (2 ∗
a′)
(NeqUni (d, e, f)))
x))) =

((a′ = 0 ∧ b′ ≠ 0) ∧
(∀ (d, e, f)∈set a.
(∃ y′::real>−c′/b′. ∀ x::real ∈{−c′/b′<..y′}. aEvalUni (EqUni (d, e, f))
x)) ∧
(∀ (d, e, f)∈set b.
(∃ y′::real>−c′/b′. ∀ x::real ∈{−c′/b′<..y′}. aEvalUni (LessUni (d, e, f))

$x))\wedge$
$(\forall (d, e, f)\in set\ c.$
$(\exists y'::real>-c'/b'.\ \forall x::real \in\{-c'/b'<..y'\}.\ aEvalUni\ (LeqUni\ (d, e, f))$
$x))\ \wedge$
$(\forall (d, e, f)\in set\ d.$
$(\exists y'::real>-c'/b'.\ \forall x::real \in\{-c'/b'<..y'\}.\ aEvalUni\ (NeqUni\ (d, e, f))$
$x))\ \vee$
$a' \neq 0\ \wedge$
$-\ b'^2 + 4 * a' * c' \leq 0\ \wedge$
$((\forall (d, e, f)\in set\ a.$
$(\exists y'>(-\ b' + 1 * sqrt\ (b'^2 - 4 * a' * c'))\ /\ (2 * a').$
$\forall x\in\{(-\ b' + 1 * sqrt\ (b'^2 - 4 * a' * c'))\ /\ (2 * a')<..y'\}.$
$aEvalUni\ (EqUni\ (d,e,f))\ x))\ \wedge$
$(\forall (d, e, f)\in set\ b.$
$(\exists y'>(-\ b' + 1 * sqrt\ (b'^2 - 4 * a' * c'))\ /\ (2 * a').$
$\forall x\in\{(-\ b' + 1 * sqrt\ (b'^2 - 4 * a' * c'))\ /\ (2 * a')<..y'\}.$
$aEvalUni\ (LessUni\ (d,e,f))\ x))\ \wedge$
$(\forall (d, e, f)\in set\ c.$
$(\exists y'>(-\ b' + 1 * sqrt\ (b'^2 - 4 * a' * c'))\ /\ (2 * a').$
$\forall x\in\{(-\ b' + 1 * sqrt\ (b'^2 - 4 * a' * c'))\ /\ (2 * a')<..y'\}.$
$aEvalUni\ (LeqUni\ (d,e,f))\ x))\ \wedge$
$(\forall (d, e, f)\in set\ d.$
$(\exists y'>(-\ b' + 1 * sqrt\ (b'^2 - 4 * a' * c'))\ /\ (2 * a').$
$\forall x\in\{(-\ b' + 1 * sqrt\ (b'^2 - 4 * a' * c'))\ /\ (2 * a')<..y'\}.$
$aEvalUni\ (NeqUni\ (d,e,f))\ x))\ \vee$
$(\forall (d, e, f)\in set\ a.$
$(\exists y'>(-\ b' + -\ 1 * sqrt\ (b'^2 - 4 * a' * c'))\ /\ (2 * a').$
$\forall x\in\{(-\ b' + -\ 1 * sqrt\ (b'^2 - 4 * a' * c'))\ /\ (2 * a')<..y'\}.$
$aEvalUni\ (EqUni\ (d,e,f))\ x))\ \wedge$
$(\forall (d, e, f)\in set\ b.$
$(\exists y'>(-\ b' + -\ 1 * sqrt\ (b'^2 - 4 * a' * c'))\ /\ (2 * a').$
$\forall x\in\{(-\ b' + -\ 1 * sqrt\ (b'^2 - 4 * a' * c'))\ /\ (2 * a')<..y'\}.$
$aEvalUni\ (LessUni\ (d,e,f))\ x))\ \wedge$
$(\forall (d, e, f)\in set\ c.$
$(\exists y'>(-\ b' + -\ 1 * sqrt\ (b'^2 - 4 * a' * c'))\ /\ (2 * a').$
$\forall x\in\{(-\ b' + -\ 1 * sqrt\ (b'^2 - 4 * a' * c'))\ /\ (2 * a')<..y'\}.$
$aEvalUni\ (LeqUni\ (d,e,f))\ x))\ \wedge$
$(\forall (d, e, f)\in set\ d.$
$(\exists y'>(-\ b' + -\ 1 * sqrt\ (b'^2 - 4 * a' * c'))\ /\ (2 * a').$
$\forall x\in\{(-\ b' + -\ 1 * sqrt\ (b'^2 - 4 * a' * c'))\ /\ (2 * a')<..y'\}.$
$aEvalUni\ (NeqUni\ (d,e,f))\ x))))$
⟨*proof*⟩

lemma *eq-inf* : $(\forall (a, b, c)\in set\ (a::(real*real*real)\ list).\ \exists x.\ \forall y<x.\ a * y^2 + b * y + c = 0) = (\forall (a, b, c)\in set\ a.\ a=0\wedge b=0\wedge c=0)$
⟨*proof*⟩

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 simplier 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''* : (∃ *x*. *evalUni* (*list-conj-Uni* (*map AtomUni L*)) *x*) = *evalUni* (*generalVS-DNF L*) *x*
⟨*proof*⟩

lemma *forallx-substNegInf* : (¬*evalUni* (*map-atomUni substNegInfinityUni F*) *x*) = (∀ *x*. ¬ *evalUni* (*map-atomUni substNegInfinityUni F*) *x*)
⟨*proof*⟩

lemma *linear-subst-map*: *evalUni* (*map-atomUni* (*linearSubstitutionUni b c*) *F*) *x* = *evalUni F* (−*c*/*b*)
⟨*proof*⟩

lemma *quadratic-subst-map* : *evalUni* (*map-atomUni* (*quadraticSubUni a b c d*) *F*) *x* = *evalUni F* ((*a*+*b*∗*sqrt*(*c*))/*d*)
⟨*proof*⟩

fun *convert-atom-list* :: *nat* ⇒ *atom list* ⇒ *real list* ⇒ (*atomUni list*) *option* **where**
convert-atom-list var [] *xs* = *Some* []|
convert-atom-list var (*a*#*as*) *xs* = (
case convert-atom var a xs of Some(*a*) ⇒
(*case convert-atom-list var as xs of Some*(*as*) ⇒ *Some*(*a*#*as*) | *None* ⇒ *None*)
| *None* ⇒ *None*
)

lemma *convert-atom-list-change* :
assumes *length xs'* = *var*
shows *convert-atom-list var L* (*xs'* @ *x* # Γ) = *convert-atom-list var L* (*xs'* @ *x'* # Γ)
⟨*proof*⟩

lemma *negInf-convert* :
assumes *convert-atom-list var L* (*xs'* @ *x* # *xs*) = *Some L'*

assumes *length xs′ = var*
shows $(\forall f{\in}set\ L.\ eval\ (substNegInfinity\ var\ f)\ (xs'\ @\ x\ \#\ xs))$
$= (\forall f{\in}set\ L'.\ evalUni\ (substNegInfinityUni\ f)\ x)$
⟨*proof*⟩

lemma *elimVar-atom-single* :
assumes $convert\text{-}atom\ var\ A\ (xs'\ @\ x\ \#\ xs) = Some\ A'$
assumes $convert\text{-}atom\text{-}list\ var\ L2\ (xs'\ @\ x\ \#\ xs) = Some\ L2'$
assumes $length\ xs' = var$
shows $eval\ (elimVar\ var\ L2\ []\ A)\ (xs'\ @\ x\ \#\ xs) = evalUni\ (elimVarUni\text{-}atom\ L2'\ A')\ x$
⟨*proof*⟩

lemma *convert-list* :
assumes $convert\text{-}atom\text{-}list\ var\ L\ (xs'\ @\ x\ \#\ xs) = Some\ L'$
assumes $l{\in}set(L)$
shows $\exists\, l'{\in}\ set\ L'.\ convert\text{-}atom\ var\ l\ (xs'\ @\ x\ \#\ xs) = Some\ l'$
⟨*proof*⟩

lemma *convert-list2* :
assumes $convert\text{-}atom\text{-}list\ var\ L\ (xs'\ @\ x\ \#\ xs) = Some\ L'$
assumes $l'{\in}set(L')$
shows $\exists\, l{\in}\ set\ L.\ convert\text{-}atom\ var\ l\ (xs'\ @\ x\ \#\ xs) = Some\ l'$
⟨*proof*⟩

lemma *elimVar-atom-convert* :
assumes $convert\text{-}atom\text{-}list\ var\ L\ (xs'\ @\ x\ \#\ xs) = Some\ L'$
assumes $convert\text{-}atom\text{-}list\ var\ L2\ (xs'\ @\ x\ \#\ xs) = Some\ L2'$
assumes $length\ xs' = var$
shows $(\exists f{\in}set\ L.\ eval\ (elimVar\ var\ L2\ []\ f)\ (xs'\ @\ x\ \#\ xs))$
$= (\exists f{\in}set\ L'.\ evalUni\ (elimVarUni\text{-}atom\ L2'\ f)\ x)$
⟨*proof*⟩

lemma *eval-convert* :
assumes $convert\text{-}atom\text{-}list\ var\ L\ (xs'\ @\ x\ \#\ xs) = Some\ L'$
assumes $length\ xs' = var$
shows $(\forall f{\in}set\ L.\ aEval\ f\ (xs'\ @\ x\ \#\ xs)) = (\forall f{\in}set\ L'.\ aEvalUni\ f\ x)$
⟨*proof*⟩
lemma *all-degree-2-convert* :
assumes *all-degree-2 var L*
shows $\exists\, L'.\ convert\text{-}atom\text{-}list\ var\ L\ xs = Some\ L'$
⟨*proof*⟩
lemma *gen-qe-eval* :
assumes *hlength* : $length\ xs = var$
shows $(\exists\, x.\ (eval\ (list\text{-}conj\ ((map\ Atom\ L)\ @\ F))\ (xs\ @\ (x\#\Gamma)))) = (\exists\, x.(eval\ (gen\text{-}qe\ var\ L\ F)\ (xs\ @\ (x\#\Gamma))))$
⟨*proof*⟩

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 (∃ *x*. (*eval* (*list-conj* (*map Atom L*)) (*xs′*@*x*#Γ))) = (∀ *x*.(*eval* (*gen-qe var L* []) (*xs′*@*x*#Γ)))
freeIn var (*gen-qe var L* [])
⟨*proof*⟩

lemma *gen-qe-eval′′* :
assumes *all-degree-2 var L*
assumes *length xs′* = *var*
shows (∃ *x*. (*eval* (*list-conj* (*map Atom L*)) (*xs′*@*x*#Γ))) = (∀ *x*.(*eval* (*list-disj* (*list-conj* (*map* (*substNegInfinity var*) *L*) # *map* (*elimVar var L* []) *L*)) (*xs′*@*x*#Γ)))
⟨*proof*⟩

end

10 QE Algorithm Proofs

10.1 DNF

theory *DNF*
imports *VSAlgos*
begin

theorem *dnf-eval* :
(∃ (*al*,*fl*)∈*set* (*dnf* φ).
(∀ *a*∈*set al*. *aEval a xs*)
∧ (∀ *f*∈*set fl*. *eval f xs*))
= *eval* φ *xs*
⟨*proof*⟩

theorem *dnf-modified-eval* :
(∃ (*al*,*fl*,*n*)∈*set* (*dnf-modified* φ).
(∃ *L*. (*length L* = *n*
∧ (∀ *a*∈*set al*. *aEval a* (*L*@*xs*))

$\wedge$ ($\forall f\in set\ fl.\ eval\ f\ (L@xs)))))$ = *eval* φ *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* : $\bigwedge amount\ F\ \Gamma.$
$(\exists xs.\ (length\ xs = amount \wedge eval\ (list\text{-}disj\ (map(\lambda(L,F,n).\ ExN\ n\ (list\text{-}conj\ (map\ fm.Atom\ L\ @\ F)))\ F))\ (xs\ @\ \Gamma))) = (eval\ (step\ amount\ F)\ \Gamma)$
assumes *opt* : $\bigwedge xs\ F\ .\ eval\ (opt\ F)\ xs = eval\ F\ xs$
shows *eval* (*QE-dnf′ opt step* φ) *xs* = *eval* φ *xs*
⟨*proof*⟩

theorem *QE-dnf-eval*:
assumes *steph* : $\bigwedge var\ amount\ new\ L\ F\ \Gamma.$
$amount\leq var+1 \Longrightarrow$
$(\exists xs.\ (length\ xs = var+1 \wedge eval\ (list\text{-}conj\ (map\ fm.Atom\ L\ @\ F))\ (xs\ @\ \Gamma)))$
$= (\exists xs.\ (length\ xs = var+1 \wedge eval\ (step\ amount\ var\ L\ F)\ (xs\ @\ \Gamma)))$
assumes *opt* : $\bigwedge xs\ F\ .\ eval\ (opt\ F)\ xs = eval\ F\ xs$
shows *eval* (*QE-dnf opt step* φ) *xs* = *eval* φ *xs*
⟨*proof*⟩

lemma *opt*: *eval* ((*push-forall* ∘ *nnf* ∘ *unpower 0 o groupQuantifiers o 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* ∘ *nnf* ∘ *unpower 0 o clearQuantifiers*) *F*)

L= *eval F L*
⟨*proof*⟩

lemma *repeatAmountOfQuantifiers-helper-eval* :
assumes ⋀*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 ⋀*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* : ⋀*xs var L F* Γ. *length xs* = *var* ⟹ (∃ *x*. *eval* (*list-conj* (*map fm.Atom L* @ *F*)) (*xs* @ *x* # Γ)) = (∃ *x*. *eval* (*step var L F*) (*xs* @ *x* # Γ))
shows (∃ *xs*. (*length xs* = *amount* ∧ *eval* (*list-disj* (*map*(λ(*L*,*F*,*n*). *ExN n* (*list-conj* (*map fm.Atom L* @ *F*))) *F*)) (*xs* @ Γ))) = (*eval* (*the-real-step-augment step amount F*) Γ)
⟨*proof*⟩

lemma *step-converter* :
assumes *steph* : ⋀*xs var L F* Γ. *length xs* = *var* ⟹ (∃ *x*. *eval* (*list-conj* (*map fm.Atom L* @ *F*)) (*xs* @ *x* # Γ)) = (∃ *x*. *eval* (*step var L F*) (*xs* @ *x* # Γ))
shows ⋀*var L F* Γ. (∃ *xs*. *length xs* = *var* + *1* ∧ *eval* (*list-conj* (*map fm.Atom L* @ *F*)) (*xs* @ Γ)) =
(∃ *xs*. (*length xs* = (*var* + *1*)) ∧ *eval* (*step var L F*) (*xs* @ Γ))
⟨*proof*⟩

lemma *step-augmenter-eval* :
assumes *steph* : ⋀*xs var L F* Γ. *length xs* = *var* ⟹ (∃ *x*. *eval* (*list-conj* (*map fm.Atom L* @ *F*)) (*xs* @ *x* # Γ)) = (∃ *x*. *eval* (*step var L F*) (*xs* @ *x* # Γ))
assumes *heuristic*: ⋀*n var L F*. *heuristic n L F* = *var* ⟹ *var* ≤ *n*
shows ⋀*var amount L F* Γ.
amount ≤ *var* + *1* ⟹
(∃ *xs*. *length xs* = *var* + *1* ∧ *eval* (*list-conj* (*map fm.Atom L* @ *F*)) (*xs* @ Γ))
=
(∃ *xs*. (*length xs* = (*var* + *1*)) ∧ *eval* (*step-augment step heuristic amount var*

L F) (*xs* @ Γ))
⟨*proof*⟩

lemma *qe-eq-repeat-eval-augment* : *amount* ≤ *var+1* ⟹
(∃*xs*. (*length xs* = *var* + *1*) ∧ *eval* (*list-conj* (*map fm.Atom L* @ *F*)) (*xs* @ Γ)) =
(∃*xs*. (*length xs* = *var* + *1*) ∧ *eval* (*step-augment qe-eq-repeat IdentityHeuristic amount var L F*) (*xs* @ Γ))
⟨*proof*⟩

lemma *qe-eq-repeat-eval′* :
(∃*xs*. (*length xs* = *var* + *1*) ∧ *eval* (*list-conj* (*map fm.Atom L* @ *F*)) (*xs* @ Γ)) =
(∃*xs*. (*length xs* = *var* + *1*) ∧ *eval* (*qe-eq-repeat var L F*) (*xs* @ Γ))
⟨*proof*⟩

lemma *gen-qe-eval-augment* : *amount* ≤ *var+1* ⟹
(∃*xs*. (*length xs* = *var* + *1*) ∧ *eval* (*list-conj* (*map fm.Atom L* @ *F*)) (*xs* @ Γ)) =
(∃*xs*. (*length xs* = *var* + *1*) ∧ *eval* (*step-augment gen-qe IdentityHeuristic amount var L F*) (*xs* @ Γ))
⟨*proof*⟩

lemma *gen-qe-eval′* :
(∃*xs*. (*length xs* = *var* + *1*) ∧ *eval* (*list-conj* (*map fm.Atom L* @ *F*)) (*xs* @ Γ)) =
(∃*xs*. (*length xs* = *var* + *1*) ∧ *eval* (*gen-qe var L F*) (*xs* @ Γ))
⟨*proof*⟩

lemma *luckyFind-eval-augment* : *amount* ≤ *var+1* ⟹
(∃*xs*. (*length xs* = *var* + *1*) ∧ *eval* (*list-conj* (*map fm.Atom L* @ *F*)) (*xs* @ Γ)) =
(∃*xs*. (*length xs* = *var* + *1*) ∧ *eval* (*step-augment luckyFind′ IdentityHeuristic amount var L F*) (*xs* @ Γ))
⟨*proof*⟩

lemma *luckyFind-eval′* :
(∃*xs*. (*length xs* = *var* + *1*) ∧ *eval* (*list-conj* (*map fm.Atom L* @ *F*)) (*xs* @ Γ)) =
(∃*xs*. (*length xs* = *var* + *1*) ∧ *eval* (*luckyFind′ var L F*) (*xs* @ Γ))
⟨*proof*⟩

lemma *luckiestFind-eval′* :
(∃*xs*. (*length xs* = *var* + *1*) ∧ *eval* (*list-conj* (*map fm.Atom L* @ *F*)) (*xs* @ Γ)) =
(∃*xs*. (*length xs* = *var* + *1*) ∧ *eval* (*luckiestFind var L F*) (*xs* @ Γ))
⟨*proof*⟩

lemma *sortedListMember* : *sorted-list-of-fset b* = *var* # *list* ⟹ *fmember var b*
⟨*proof*⟩

lemma *rangeHeuristic* :
assumes *heuristicPicker n L F* = *Some* (*var*, *step*)
shows *var*≤*n*
⟨*proof*⟩

lemma *pickedOneOfThem* :
assumes *heuristicPicker n L F* = *Some* (*var*, *step*)
shows *step* = *qe-eq-repeat* ∨ *step* = *gen-qe* ∨ *step* = *luckyFind′*
⟨*proof*⟩

lemma *superPicker-eval* :
amount≤ *var*+*1* ⟹ (∃ *xs*. *length xs* = *var* + *1* ∧ *eval* (*list-conj* (*map fm.Atom L* @ *F*)) (*xs* @ Γ)) =
(∃ *xs*. (*length xs* = (*var* + *1*)) ∧ *eval* (*superPicker amount var L F*) (*xs* @ Γ))
⟨*proof*⟩

lemma *brownHueristic-less-than*: *brownsHeuristic n L F* = *var* ⟹ *var*≤ *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*: ∀ *xs*. *eval* (*VSLuckiest* φ) *xs* = *eval* φ *xs*
⟨*proof*⟩

theorem *VSLuckiestBlocks* : ∀ *xs*. *eval* (*VSLuckiestBlocks* φ) *xs* = *eval* φ *xs*
⟨*proof*⟩

theorem *VSEquality* : ∀ *xs*. *eval* (*VSEquality* φ) *xs* = *eval* φ *xs*
⟨*proof*⟩

theorem *VSEqualityBlocks* : ∀ *xs*. *eval* (*VSEqualityBlocks* φ) *xs* = *eval* φ *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* : $\forall xs.\ eval\ (VSEquality\text{-}3\text{-}times\ \varphi)\ xs = eval\ \varphi\ xs$
⟨*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 *VSBrowns*: $\forall xs.\ eval\ (VSBrowns\ \varphi)\ xs = eval\ \varphi\ xs$
⟨*proof*⟩

theorem *VSGeneral-3-times* : $\forall xs.\ eval\ (VSGeneral\text{-}3\text{-}times\ \varphi)\ xs = eval\ \varphi\ xs$
⟨*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 VSHeuristic 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.