Verified Quadratic Virtual Substitution for Real Arithmetic

Matias Scharager, Katherine Cordwell, Stefan Mitsch and André Platzer

March 17, 2025

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

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

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

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

2 QE lemmas

theory *QE*
  imports *Polynomials.MPoly-Type-Univariate*
    Polynomials.Polynomials Polynomials.MPoly-Type-Class-FMap
    HOL−Library.Quadratic-Discriminant
begin

2.1 Useful Definitions/Setting Up

definition *sign*:: *real Polynomial.poly* ⇒ *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 −
have $\neg(\exists\, n.\ (n{+}1) < (length\ l) \wedge x \geq l\ !n \wedge x \leq l\ !\ (n + 1)) \Longrightarrow False$
proof *clarsimp*
fix n
assume *alln*: $\forall\, n.\ l\ !\ n \leq x \longrightarrow Suc\ n < length\ l \longrightarrow \neg\ x \leq l\ !\ Suc\ n$
have $\forall\, k.\ (k < length\ l \longrightarrow x > l\ !\ k)$
proof *clarsimp*
fix k
show $k < length\ l \Longrightarrow l\ !\ k < x$
proof (*induct k*)
case *0*
then show *?case* **using** *xgt* **by** *auto*
next
case (*Suc k*)
then show *?case* **using** *alln*
using *less-eq-real-def* **by** *auto*
qed
qed
then show *False*
using *xlt diff-Suc-less lengt not-less*
by (*metis One-nat-def*)
qed
then show *?thesis* **by** *auto*
qed

2.2 Quadratic polynomial properties

lemma *quadratic-poly-eval*:
fixes *a b c*::*real*
fixes x::*real*
shows *poly* [:*c*, *b*, *a*:] $x = a{*}x\hat{}2 + b{*}x + c$
proof −
have $x * (b + x * a) = a * x^2 + b * x$ **by** (*metis add.commute distrib-right mult.assoc mult.commute power2-eq-square*)
then show *?thesis* **by** *auto*
qed

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 −
have $\forall\, x.\ a{*}x\hat{}2 + b{*}x + c = poly\ [:c,\ b,\ a:]\ x$
proof *clarsimp*

fix x
show $a * x^2 + b * x = x * (b + x * a)$
using *quadratic-poly-eval*[*of* c b a x] **by** *auto*
qed
then show *?thesis*
by *auto*
qed

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 −
have $\forall (a,\ b,\ c) \in les.\ finite\ \{(x::real).\ a*x\hat{}2 + b*x + c = 0\}$
proof *clarsimp*
fix a b c
assume $(a,\ b,\ c) \in les$
then have $[:c,\ b,\ a:] \neq 0$ **using** *nex* **by** *auto*
then have *finite* $\{x.\ poly\ [:c,\ b,\ a:]\ x = 0\}$
using *poly-roots-finite*[**where** $p = [:c,\ b,\ a:]$] **by** *auto*
then show *finite* $\{x.\ a * x^2 + b * x + c = 0\}$
using *poly-roots-set-same* **by** *auto*
qed
then show *?thesis* **using** *fin*
unfolding *root-set-def* **by** *auto*
qed

lemma *nonzero-root-set-finite*:
assumes *fin*: *finite les*
shows *finite* (*nonzero-root-set les*)
proof −
have $\forall (a,\ b,\ c) \in les.\ (a \neq 0 \vee b \neq 0) \longrightarrow finite\ \{(x::real).\ a*x\hat{}2 + b*x + c = 0\}$
proof *clarsimp*
fix a b c
assume *ins*: $(a,\ b,\ c) \in les$
assume $a = 0 \longrightarrow b \neq 0$
then have $[:c,\ b,\ a:] \neq 0$ **using** *ins* **by** *auto*
then have *finite* $\{x.\ poly\ [:c,\ b,\ a:]\ x = 0\}$
using *poly-roots-finite*[**where** $p = [:c,\ b,\ a:]$] **by** *auto*
then show *finite* $\{x.\ a * x^2 + b * x + c = 0\}$
using *poly-roots-set-same* **by** *auto*
qed
then show *?thesis* **using** *fin*
unfolding *nonzero-root-set-def* **by** *auto*
qed

lemma *discriminant-lemma*:
fixes a b c r::*real*

assumes *aneq*: $a \neq 0$
assumes *beq*: $b = 2 * a * r$
assumes *root*: $a * r\hat{}2 - 2 * a * r*r + c = 0$
shows $\forall x.\ a * x^2 + b * x + c = 0 \longleftrightarrow x = -r$
proof −
have $c = a*r\hat{}2$ **using** *root*
by (*simp add*: *power2-eq-square*)
then have *same*: $b\hat{}2 - 4*a*c = (2 * a * r)\hat{}2 - 4*a*(a*r\hat{}2)$ **using** *beq*
by *blast*
have $(2 * a * r)\hat{}2 = 4*a\hat{}2*r\hat{}2$
by (*simp add*: *mult.commute power2-eq-square*)
then have $(2 * a * r)\hat{}2 - 4*a*(a*(r)\hat{}2) = 0$
using *power2-eq-square* **by** *auto*
then have $b\hat{}2 - 4*a*c = 0$ **using** *same*
by *simp*
then have $\forall x.\ a * x^2 + b * x + c = 0 \longleftrightarrow x = -b\ /\ (2 * a)$
using *discriminant-zero aneq* **unfolding** *discrim-def* **by** *auto*
then show *?thesis* **using** *beq*
by (*simp add*: *aneq*)
qed

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 *clarsimp*
fix *x y*
assume *sign p x* $\neq$ *sign p y*
assume $x < y$
then show $\exists\, c{\in}$*root-list p*. $x \leq c \wedge c \leq y$
using *poly-IVT-pos*[*of x y p*] *poly-IVT-neg*[*of x y p*]
by (*metis* (*mono-tags*) ‹*sign p x* ≠ *sign p y*› *less-eq-real-def linorder-neqE-linordered-idom mem-Collect-eq root-list-def sign-def*)
qed

lemma *changes-sign-var*:
fixes *a b c x y*:: *real*
shows ((*sign-num* $(a*x\hat{}2 + b*x + c) \neq$ *sign-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 *clarsimp*
assume *sn*: *sign-num* $(a * x^2 + b * x + c) \neq$ *sign-num* $(a * y^2 + b * y + c)$
assume *xy*: $x < y$
let *?p* = [:*c*, *b*, *a*:]
have *cs*: ((*sign ?p x* $\neq$ *sign ?p y* $\wedge\ x < y) \longrightarrow (\exists\, c \in$ (*root-list ?p*). $x \leq c \wedge c \leq y))$
using *changes-sign*[*of ?p*] **by** *auto*
have (*sign ?p x* $\neq$ *sign ?p y* $\wedge\ x < y$)

using *sn xy* **unfolding** *sign-def sign-num-def* **using** *quadratic-poly-eval*
by *presburger*
then have $(\exists\, c \in (\textit{root-list}\ ?p).\ x \leq c \wedge c \leq y)$
using *cs*
by *auto*
then obtain q **where** $q \in \textit{root-list}\ ?p \wedge x \leq q \wedge q \leq y$
by *auto*
then have $a{*}q\hat{}2 + b{*}q + c = 0 \wedge x \leq q \wedge q \leq y$
unfolding *root-list-def* **using** *quadratic-poly-eval*[*of c b a q*]
by *auto*
then show $\exists\, q.\ a * q^2 + b * q + c = 0 \wedge x \leq q \wedge q \leq y$
by *auto*
qed

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 −
assume *r-lt*: $r < p$
assume *inf-zer*: $\forall\, x{\in}\{r <..p\}.\ a * x^2 + b * x + c = 0$
have *nf*: ¬*finite* $\{r..{<}p\}$ **using** *Set-Interval.dense-linorder-class.infinite-Ioo r-lt* **by** *auto*
have $\neg(a = 0 \wedge b = 0 \wedge c = 0) \Longrightarrow \textit{False}$
proof −
assume $\neg(a = 0 \wedge b = 0 \wedge c = 0)$
then have $[:c,\ b,\ a:] \neq 0$ **by** *auto*
then have *fin*: *finite* $\{x.\ \textit{poly}\ [:c,\ b,\ a:]\ x = 0\}$ **using** *poly-roots-finite*[**where** $p = [:c,\ b,\ a:]$] **by** *auto*
have $\{x.\ a{*}x\hat{}2 + b{*}x + c = 0\} = \{x.\ \textit{poly}\ [:c,\ b,\ a:]\ x = 0\}$ **using** *quadratic-poly-eval* **by** *auto*
then have *finset*: *finite* $\{x.\ a{*}x\hat{}2 + b{*}x + c = 0\}$ **using** *fin* **by** *auto*
have $\{r <..p\} \subseteq \{x.\ a{*}x\hat{}2 + b{*}x + c = 0\}$ **using** *inf-zer* **by** *blast*
then show *False* **using** *finset nf*
using *finite-subset*
by (*metis* (*no-types*, *lifting*) *infinite-Ioc-iff r-lt*)
qed
then show $(a = 0 \wedge b = 0 \wedge c = 0)$ **by** *auto*
qed

lemma *continuity-lem-lt0*:
fixes *r*:: *real*
fixes *a b c*:: *real*
shows *poly* $[:c,\ b,\ a:]\ r < 0 \Longrightarrow$
$\exists\, y' > r.\ \forall\, x{\in}\{r{<}..y'\}.\ \textit{poly}\ [:c,\ b,\ a:]\ x < 0$
proof −
let *?f* = *poly* [:*c*,*b*,*a*:]

assume *r-ltz*: *poly* [:*c*, *b*, *a*:] *r* < *0*
then have [:*c*, *b*, *a*:] ≠ *0* **by** *auto*
then have *finite* {*x*. *poly* [:*c*, *b*, *a*:] *x* = *0*} **using** *poly-roots-finite*[**where** *p* = [:*c*, *b*, *a*:]]
by *auto*
then have *fin*: *finite* {*x*. *x* > *r* ∧ *poly* [:*c*, *b*, *a*:] *x* = *0*}
using *finite-Collect-conjI* **by** *blast*
let *?l* = *sorted-list-of-set* {*x*. *x* > *r* ∧ *poly* [:*c*, *b*, *a*:] *x* = *0*}
show *?thesis* **proof** (*cases length ?l* = *0*)
case *True*
then have *no-zer*: ¬(∃ *x*>*r*. *poly* [:*c*, *b*, *a*:] *x* = *0*) **using** *sorted-list-of-set-eq-Nil-iff fin* **by** *auto*
then have ⋀*y*. *y* > *r* ∧ *y* < (*r* + *1*) ⟹ *poly* [:*c*, *b*, *a*:] *y* < *0*
proof −
fix *y*
assume *y* > *r* ∧ *y* < *r* + *1*
then show *poly* [:*c*, *b*, *a*:] *y* < *0*
using *r-ltz no-zer poly-IVT-pos*[**where** *a* = *r*, **where** *p* = [:*c*, *b*, *a*:], **where** *b* = *y*]
by (*meson linorder-neqE-linordered-idom*)
qed
then show *?thesis*
by (*metis greaterThanAtMost-iff less-add-one less-eq-real-def linorder-not-le no-zer poly-IVT-pos r-ltz*)
next
case *False*
then have *len-nonz*: *length* (*sorted-list-of-set* {*x*. *r* < *x* ∧ *poly* [:*c*, *b*, *a*:] *x* = *0*}) ≠ *0*
by *blast*
then have ∀ *n* ∈ {*x*. *x* > *r* ∧ *poly* [:*c*, *b*, *a*:] *x* = *0*}. (*nth-default 0 ?l 0*) ≤ *n*
using *fin set-sorted-list-of-set sorted-sorted-list-of-set*
using *in-set-conv-nth leI not-less0 sorted-nth-mono*
by (*smt not-less-iff-gr-or-eq nth-default-def*)
then have *no-zer*: ¬(∃ *x*>*r*. (*x* < (*nth-default 0 ?l 0*) ∧ *poly* [:*c*, *b*, *a*:] *x* = *0*))
using *sorted-sorted-list-of-set* **by** *auto*
then have *fa*: ⋀*y*. *y* > *r* ∧ *y* < (*nth-default 0 ?l 0*) ⟹ *poly* [:*c*, *b*, *a*:] *y* < *0*
proof −
fix *y*
assume *y* > *r* ∧ *y* < (*nth-default 0 ?l 0*)
then show *poly* [:*c*, *b*, *a*:] *y* < *0*
using *r-ltz no-zer poly-IVT-pos*[**where** *a* = *r*, **where** *p* = [:*c*, *b*, *a*:], **where** *b* = *y*]
by (*meson less-imp-le less-le-trans linorder-neqE-linordered-idom*)
qed
have *nth-default 0 ?l 0* > *r* **using** *fin set-sorted-list-of-set*
using *len-nonz length-0-conv length-greater-0-conv mem-Collect-eq nth-mem*
by (*metis* (*no-types*, *lifting*) *nth-default-def*)
then have ∃ (*y'*::*real*). *r* < *y'* ∧ *y'* < (*nth-default 0 ?l 0*)
using *dense* **by** *blast*

```
    then obtain y' where y-prop:r < y' ∧y' < (nth-default 0 ?l 0) by auto
    then have ∀ x∈{r<..y'}. poly [:c, b, a:] x < 0
      using fa by auto
    then show ?thesis using y-prop by blast
  qed
qed

lemma continuity-lem-gt0:
  fixes r:: real
  fixes a b c:: real
  shows poly [:c, b, a:] r > 0 ⟹
    ∃ y'> r. ∀ x∈{r<..y'}. poly [:c, b, a:] x > 0
proof −
  assume r-gtz: poly [:c, b, a:] r > 0
  let ?p = [:−c, −b, −a:]
  have revpoly: ∀ x. −1∗(poly [:c, b, a:] x) = poly [:−c, −b, −a:] x
   by (metis (no-types, opaque-lifting) Polynomial.poly-minus minus-pCons mult-minus1
neg-equal-0-iff-equal)
  then have poly ?p r < 0 using r-gtz
    by (metis mult-minus1 neg-less-0-iff-less)
  then have ∃ y'> r. ∀ x∈{r<..y'}. poly ?p x < 0 using continuity-lem-lt0
    by blast
  then obtain y' where y-prop: y' > r ∧ (∀ x∈{r<..y'}. poly ?p x < 0) by auto
  then have ∀ x∈{r<..y'}. poly [:c, b, a:] x > 0  using revpoly
    using neg-less-0-iff-less by fastforce
  then show ?thesis
    using y-prop  by blast
qed

lemma continuity-lem-lt0-expanded:
  fixes r:: real
  fixes a b c:: real
  shows a∗r^2 + b∗r + c < 0 ⟹
    ∃ y'> r. ∀ x∈{r<..y'}. a∗x^2 + b∗x + c < 0
  using quadratic-poly-eval continuity-lem-lt0
  by (simp add: add.commute)

lemma continuity-lem-gt0-expanded:
  fixes r:: real
  fixes a b c:: real
  fixes k::real
  assumes kgt: k > r
  shows a∗r^2 + b∗r + c > 0 ⟹
    ∃ x∈{r<..k}. a∗x^2 + b∗x + c > 0
proof −
  assume a∗r^2 + b∗r + c > 0
  then have ∃ y'> r. ∀ x∈{r<..y'}. poly [:c, b, a:] x > 0
    using continuity-lem-gt0 quadratic-poly-eval[of c b a r] by auto
  then obtain y' where y-prop: y' > r ∧ (∀ x∈{r<..y'}. poly [:c, b, a:] x > 0)
```

by *auto*
then have $\exists\, q.\ q > r \wedge q < min\ k\ y'$ **using** *kgt dense*
by (*metis min-less-iff-conj*)
then obtain *q* **where** *q-prop*: $q > r \wedge q < min\ k\ y'$ **by** *auto*
then have $a*q\hat{}2 + b*q + c > 0$ **using** *y-prop quadratic-poly-eval*[*of c b a q*]
by (*metis greaterThanAtMost-iff less-eq-real-def min-less-iff-conj*)
then show *?thesis*
using *q-prop* **by** *auto*
qed

2.4 Negative Infinity (Limit) Properties

lemma *ysq-dom-y*:
fixes *b*:: *real*
fixes *c*:: *real*
shows $\exists\,(w{::}real).\ \forall\,(y{::}\ real).\ (y < w \longrightarrow y\hat{}2 > b*y)$
proof −
have *c1*: $b \geq 0 ==> \exists\,(w{::}real).\ \forall\,(y{::}\ real).\ (y < w \longrightarrow y\hat{}2 > b*y)$
proof −
assume $b \geq 0$
then have *p1*: $\forall\,(y{::}\ real).\ (y < -1 \longrightarrow y*b \leq 0)$
by (*simp add*: *mult-nonneg-nonpos2*)
have *p2*: $\forall\,(y{::}\ real).\ (y < -1 \longrightarrow y\hat{}2 > 0)$
by *auto*
then have *h1*: $\forall\,(y{::}\ real).\ (y < -1 \longrightarrow y\hat{}2 > b*y)$
using *p1 p2*
by (*metis less-eq-real-def less-le-trans mult.commute*)
then show *?thesis* **by** *auto*
qed
have *c2*: $b < 0 \wedge b > -1 ==> \exists\,(w{::}real).\ \forall\,(y{::}\ real).\ (y < w \longrightarrow y\hat{}2 > b*y)$
proof −
assume $b < 0 \wedge b > -1$
then have *h1*: $\forall\,(y{::}\ real).\ (y < -1 \longrightarrow y\hat{}2 > b*y)$
by (*simp add*: *power2-eq-square*)
then show *?thesis* **by** *auto*
qed
have *c3*: $b \leq -1 ==> \exists\,(w{::}real).\ \forall\,(y{::}\ real).\ (y < w \longrightarrow y\hat{}2 > b*y)$
proof −
assume $b \leq -1$
then have *h1*: $\forall\,(y{::}\ real).\ (y < b \longrightarrow y\hat{}2 > b*y)$
by (*metis le-minus-one-simps*(*3*) *less-irrefl less-le-trans mult.commute mult-less-cancel-left power2-eq-square*)
then show *?thesis* **by** *auto*
qed
then show *?thesis* **using** *c1 c2 c3*
by (*metis less-trans linorder-not-less*)
qed

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 −
have $\exists(w{::}real).\ \forall(y{::}\ real).\ (y < w \longrightarrow y\hat{}2 > b{*}y)$ **using** *ysq-dom-y* **by** *auto*
then obtain w **where** *w-prop*: $\forall(y{::}\ real).\ (y < w \longrightarrow y\hat{}2 > b{*}y)$ **by** *auto*
have $c \leq 0 \Longrightarrow\ \forall(y{::}real).\ (y < w \longrightarrow y\hat{}2 > b{*}y + c)$
using *w-prop* **by** *auto*
then have *c1*: $c \leq 0 \Longrightarrow \exists(w{::}real).\ \forall(y{::}real).\ (y < w \longrightarrow y\hat{}2 > b{*}y + c)$
by *auto*
have $\exists(w{::}real).\ \forall(y{::}\ real).\ (y < w \longrightarrow y\hat{}2 > (b-c){*}y)$ **using** *ysq-dom-y* **by** *auto*
then obtain k **where** *k-prop*: $\forall(y{::}\ real).\ (y < k \longrightarrow y\hat{}2 > (b-c){*}y)$ **by** *auto*
let *?mn* = *min k* (−*1*)
have $(c{>}\ 0 \Longrightarrow (\forall\ y < -1.\ -c{*}y > c))$
proof −
assume *cgt*: $c{>}\ 0$
show $\forall(y{::}real) < -1.\ -c{*}y > c$
proof *clarsimp*
fix $y{::}real$
assume $y < -1$
then have $-y > 1$
by *auto*
then have $c < c{*}(-y)$ **using** *cgt*
by (*metis* ‹*1* < − *y*› *mult.right-neutral mult-less-cancel-left-pos*)
then show $c < -\ (c * y)$
by *auto*
qed
qed
then have $(c{>}\ 0 \longrightarrow (\forall\ y <$ *?mn*. $(b-c){*}y > b{*}y + c))$
by (*simp add*: *left-diff-distrib*)
then have *c2*: $c > 0 \Longrightarrow\ \forall(y{::}real).\ (y <$ *?mn* $\longrightarrow y\hat{}2 > b{*}y + c)$
using *k-prop*
by *force*
then have *c2*: $c > 0 \Longrightarrow \exists(w{::}real).\ \forall(y{::}real).\ (y < w \longrightarrow y\hat{}2 > b{*}y + c)$
by *blast*
show *?thesis* **using** *c1 c2*
by *fastforce*
qed

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 −
have $\exists(w{::}real).\ \forall(y{::}real).\ (y < w \longrightarrow y\hat{}2 > -b{*}y + c)$
using *ysq-dom-y-plus-coeff*[**where** $b = -b$, **where** $c = c$] **by** *auto*
then obtain w **where** *w-prop*: $\forall(y{::}real).\ (y < w \longrightarrow y\hat{}2 > -b{*}y + c)$ **by** *auto*
let *?mn* = *min w* (−*1*)

have $\forall (y::real).\ (y < ?mn \longrightarrow y\hat{}2 > -b*y + c)$ **using** *w-prop* **by** *auto*
then have $\forall (y::real).\ (y > (-1*?mn) \longrightarrow y\hat{}2 > b*y + c)$
by (*metis* (*no-types*, *opaque-lifting*) *minus-less-iff minus-mult-commute mult-1 power2-eq-iff*)
then show *?thesis* **by** *auto*
qed

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 −
have $\exists (w::real).\ \forall (y::real).\ (y > w \longrightarrow y\hat{}2 > (-b/a)*y + (-c/a))$
using *ysq-dom-y-plus-coeff-2*[**where** $b = -b/a$, **where** $c = -c/a$] **by** *auto*
then have *keyh*: $\exists (w::real).\ \forall (y::real).\ (y > w \longrightarrow a*y\hat{}2 < a*((-b/a)*y + (-c/a)))$
using *alt* **by** *auto*
have *simp1*: $\forall y.\ a*((-b/a)*y + (-c/a)) = a*(-b/a)*y + a*(-c/a)$
using *diff-divide-eq-iff* **by** *fastforce*
have *simp2*: $\forall y.\ a*(-b/a)*y + a*(-c/a) = -b*y + a*(-c/a)$
using *assms* **by** *auto*
have *simp3*: $\forall y.\ -b*y + a*(-c/a) = -b*y - c$
using *assms* **by** *auto*
then have $\forall y.\ a*((-b/a)*y + (-c/a)) = -b*y - c$ **using** *simp1 simp2 simp3* **by** *auto*
then have *keyh2*: $\exists (w::real).\ \forall (y::real).\ (y > w \longrightarrow a*y\hat{}2 < -b*y-c)$
using *keyh* **by** *auto*
have $\forall y.\ a*y\hat{}2 < -b*y-c \longleftrightarrow a*y\hat{}2 + b*y + c < 0$ **by** *auto*
then show *?thesis* **using** *keyh2* **by** *auto*
qed

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 −
have $-a < 0$ **using** *alt*
by *simp*
then have $\exists (w::real).\ \forall (y::real).\ (y > w \longrightarrow -a*y\hat{}2 - b*y - c < 0)$
using *neg-lc-dom-quad*[**where** $a = -a$, **where** $b = -b$, **where** $c = -c$] **by** *auto*
then obtain *w* **where** *w-prop*: $\forall (y::real).\ (y > w \longrightarrow -a*y\hat{}2 - b*y - c < 0)$ **by** *auto*
then have $\forall (y::real).\ (y > w \longrightarrow a*y\hat{}2 + b*y + c > 0)$
by *auto*

then show *?thesis* **by** *auto*
qed

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 (*induct les*)
case *Nil*
then show *?case* **using** *gt-ex* **by** *auto*
next
case (*Cons z les*)
have $\forall\,a{\in}set\ les.\ case\ a\ of\ (d,\ e,\ f) \Rightarrow \exists\,y'{>}q.\ \forall\,x{\in}\{q{<}..y'\}.\ d*x^2+e*x+f<0$
using *Cons.prems* **by** *auto*
then have $\exists\,y'{>}q.\ \forall\,a{\in}set\ les.\ case\ a\ of\ (d,\ e,\ f) \Rightarrow \forall\,x{\in}\{q{<}..y'\}.\ d*x^2+e*x+f<0$
using *Cons.hyps* **by** *auto*
then obtain *y1* **where** *y1-prop* : $y1{>}q \wedge (\forall\,a{\in}set\ les.\ case\ a\ of\ (d,\ e,\ f) \Rightarrow \forall\,x{\in}\{q{<}..y1\}.\ d*x^2+e*x+f<0)$
by *auto*
have $case\ z\ of\ (d,\ e,\ f) \Rightarrow \exists\,y'{>}q.\ \forall\,x{\in}\{q{<}..y'\}.\ d*x^2+e*x+f<0$
using *Cons.prems* **by** *auto*
then obtain *y2* **where** *y2-prop*: $y2{>}q \wedge (case\ z\ of\ (d,\ e,\ f) \Rightarrow (\forall\,x{\in}\{q{<}..y2\}.\ d*x^2+e*x+f<0))$
by *auto*
let $?y = min\ y1\ y2$
have $?y > q \wedge (\forall\,a{\in}set\ (z\#les).\ case\ a\ of\ (d,\ e,\ f) \Rightarrow \forall\,x{\in}\{q{<}..?y\}.\ d*x^2+e*x+f<0)$
using *y1-prop y2-prop*
by *force*
then show *?case*
by *blast*
qed

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 −
have $(\forall\,(d,\ e,\ f){\in}set\ les.\ \exists\,y'{>}r.\ \forall\,x{\in}\{r{<}..y'\}.\ d*x^2+e*x+f<0) \Longrightarrow$
$(\exists\,y'{>}r.\ (\forall\,(d,\ e,\ f){\in}set\ les.\ \forall\,x{\in}\{r{<}..y'\}.\ d*x^2+e*x+f<0))$
using *les-qe-inf-helper assms* **by** *auto*
then have $(\exists\,y'{>}r.\ (\forall\,(d,\ e,\ f){\in}set\ les.\ \forall\,x{\in}\{r{<}..y'\}.\ d*x^2+e*x+f<0))$
using *assms*
by *blast*
then obtain *y* **where** *y-prop*: $y > r \wedge\ (\forall\,(d,\ e,\ f){\in}set\ les.\ \forall\,x{\in}\{r{<}..y\}.\ d*x^2$

$+\ e * x + f < 0$)
  by *auto*
 have $\exists q.\ q > \ r \wedge q < y$ **using** *y-prop dense* **by** *auto*
 then obtain *q* **where** *q-prop*: $q > r \wedge q < y$ **by** *auto*
 then have $(\forall (d,\ e,\ f) \in set\ les.\ d*q\hat{}2 + e*q + f < 0)$
  using *y-prop* **by** *auto*
 then show *?thesis*
  by *auto*
qed

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 −
 fix y'
 assume *beq*: $b = 2 * a * r$
 assume *aprop*: $\neg\ a < 0$
 assume *root*: $a * r^2 - 2 * a * r*r + c = 0$
 assume *rootlt*: $-\ r < y'$
 show $\exists x \in \{-\ r<..y'\}.\ \neg\ a * x^2 + 2 * a* r*x + c < 0$
 proof −
  have *h*: $a = 0 \Longrightarrow (b = 0 \wedge c = 0)$ **using** *beq root*
   by *auto*
  then have *aeq*: $a = 0 \Longrightarrow \exists x \in \{-\ r<..y'\}.\ \neg\ a * x^2 + 2 * a*r*x + c < 0$
   using *rootlt*
  by (*metis add.left-neutral continuity-lem-eq0 less-numeral-extra*(*3*) *mult-zero-left mult-zero-right*)
  then have *alt*: $a > 0 \Longrightarrow \exists x \in \{-\ r<..y'\}.\ \neg\ a * x^2 + 2 * a\ *r*x + c < 0$
  proof −
   assume *agt*: $a > 0$
   then have $\exists (w::real).\ \forall (y::real).\ (y > w \longrightarrow a*y\hat{}2 + b*y + c > 0)$
    using *pos-lc-dom-quad* **by** *auto*
   then obtain *w* **where** *w-prop*: $\forall\ y::real.\ (y > w \longrightarrow a*y\hat{}2 + b*y + c > 0)$
by *auto*
   have *isroot*: $a*(-r)\hat{}2 + b*(-r) + c = 0$ **using** *root beq* **by** *auto*
   then have *wgteq*: $w \geq -(r)$
   proof −
    have $w < -r \Longrightarrow False$
     using *w-prop isroot* **by** *auto*
    then show *?thesis*
     using *not-less* **by** *blast*
   qed
   then have *w1*: $w + 1 > -r$
    by *auto*
   have *w2*: $a*(w + 1)\hat{}2 + b*(w+1) + c > 0$ **using** *w-prop* **by** *auto*

```
      have rootiff: ∀x. a * x^2 + b * x + c = 0 ⟷ x = −r using  discrimi-
nant-lemma[where a = a, where b = b, where c= c, where r = r]
         isroot agt beq by auto
     have allgt: ∀x > −r. a*x^2 + b*x + c > 0
     proof clarsimp
       fix x
       assume x > −r
       have xgtw:  x > w + 1 ⟹ a*x^2 + b*x + c > 0
        using w1 w2 rootiff  poly-IVT-neg[where a = w+1, where b = x, where
p = [:c,b,a:]]
           quadratic-poly-eval
         by (metis less-eq-real-def linorder-not-less)
       have xltw: x < w + 1 ⟹ a*x^2 + b*x + c > 0
        using w1 w2 rootiff poly-IVT-pos[where a= x, where b = w + 1, where
p = [:c,b,a:]]
           quadratic-poly-eval less-eq-real-def linorder-not-less
         by (metis ‹− r < x›)
       then show a*x^2 + b*x + c > 0
         using w2 xgtw xltw by fastforce
     qed
     have ∃z. z > −r ∧ z < y' using rootlt dense[where x = −r, where y = y']

       by auto
     then obtain z where z-prop:  z > −r ∧ z  < y' by auto
     then have a*z^2 + b*z + c > 0 using allgt by auto
     then show ?thesis using z-prop
       using beq greaterThanAtMost-iff by force
   qed
   then show ?thesis using aeq alt aprop
     by linarith
 qed
qed

lemma leq-qe-inf-helper:
 fixes q:: real
 shows(∀(d, e, f)∈set leq. ∃y'> q. ∀x∈{q<..y'}. d * x^2 + e * x + f ≤ 0) ⟹
   (∃y'>q. (∀(d, e, f)∈set leq. ∀x∈{q<..y'}. d * x^2 + e * x + f ≤ 0))
proof (induct leq)
 case Nil
 then show ?case using gt-ex by auto
next
 case (Cons z leq)
 have ∀a∈set leq. case a of (d, e, f) ⇒ ∃y'>q. ∀x∈{q<..y'}. d * x^2 + e * x +
f ≤ 0
   using  Cons.prems by auto
 then have  ∃y'>q. ∀a∈set leq. case a of (d, e, f) ⇒ ∀x∈{q<..y'}. d * x^2 + e
* x + f ≤ 0
   using Cons.hyps by auto
 then obtain y1 where y1-prop : y1>q ∧ (∀a∈set leq. case a of (d, e, f) ⇒
```

$\forall x{\in}\{q{<}..y1\}.\ d * x^2 + e * x + f \leq 0$)
  by *auto*
 have *case z of* $(d,\ e,\ f) \Rightarrow \exists y'{>}q.\ \forall x{\in}\{q{<}..y'\}.\ d * x^2 + e * x + f \leq 0$
  using *Cons.prems* **by** *auto*
 then obtain *y2* **where** *y2-prop*: $y2{>}q \wedge$ (*case z of* $(d,\ e,\ f) \Rightarrow (\forall x{\in}\{q{<}..y2\}.$ $d * x^2 + e * x + f \leq 0)$)
  by *auto*
 let *?y* = *min y1 y2*
 have *?y* $> q \wedge (\forall a{\in}$*set* $(z\#leq).$ *case a of* $(d,\ e,\ f) \Rightarrow \forall x{\in}\{q{<}..?y\}.\ d * x^2 +$ $e * x + f \leq 0)$
  using *y1-prop y2-prop*
  by *force*
 then show *?case*
  by *blast*
qed

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 (*induct neq*)
 case *Nil*
 then show *?case* **using** *gt-ex* **by** *auto*
next
 case (*Cons z neq*)
 have $\forall a{\in}$*set neq. case a of* $(d,\ e,\ f) \Rightarrow \exists y'{>}q.\ \forall x{\in}\{q{<}..y'\}.\ d * x^2 + e * x +$ $f \neq 0$
  using *Cons.prems* **by** *auto*
 then have $\exists y'{>}q.\ \forall a{\in}$*set neq. case a of* $(d,\ e,\ f) \Rightarrow \forall x{\in}\{q{<}..y'\}.\ d * x^2 + e$ $* x + f \neq 0$
  using *Cons.hyps* **by** *auto*
 then obtain *y1* **where** *y1-prop* : $y1{>}q \wedge (\forall a{\in}$*set neq. case a of* $(d,\ e,\ f) \Rightarrow$ $\forall x{\in}\{q{<}..y1\}.\ d * x^2 + e * x + f \neq 0)$
  by *auto*
 have *case z of* $(d,\ e,\ f) \Rightarrow \exists y'{>}q.\ \forall x{\in}\{q{<}..y'\}.\ d * x^2 + e * x + f \neq 0$
  using *Cons.prems* **by** *auto*
 then obtain *y2* **where** *y2-prop*: $y2{>}q \wedge$ (*case z of* $(d,\ e,\ f) \Rightarrow (\forall x{\in}\{q{<}..y2\}.$ $d * x^2 + e * x + f \neq 0)$)
  by *auto*
 let *?y* = *min y1 y2*
 have *?y* $> q \wedge (\forall a{\in}$*set* $(z\#neq).$ *case a of* $(d,\ e,\ f) \Rightarrow \forall x{\in}\{q{<}..?y\}.\ d * x^2 +$ $e * x + f \neq 0)$
  using *y1-prop y2-prop*
  by *force*
 then show *?case*
  by *blast*
qed

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 *clarsimp*
fix *z*
assume *zgt*: $z > x$
assume *zlt*: $z < y$
have *frac-gtz*: $(1/(2*a)) > 0$ **using** *agt*
by *simp*
have *xy-prop*:$(x = (-b + sqrt(b\hat{}2 - 4*a*c))/(2*a) \wedge y = (-b - sqrt(b\hat{}2 - 4*a*c))/(2*a))$
$\vee\ (y = (-b + sqrt(b\hat{}2 - 4*a*c))/(2*a) \wedge x = (-b - sqrt(b\hat{}2 - 4*a*c))/(2*a))$

using *xyroots agt discriminant-iff* **unfolding** *discrim-def* **by** *auto*
have $b\hat{}2 - 4*a*c \geq 0$ **using** *xyroots discriminant-iff*
using *assms*(*1*) *discrim-def* **by** *auto*
then have *pos-discrim*: $b\hat{}2 - 4*a*c > 0$ **using** *xyroots discriminant-zero*
using ‹$0 \leq b^2 - 4 * a * c$› *assms*(*1*) *discrim-def less-eq-real-def linorder-not-less*
by *metis*
then have *sqrt-gt*: $sqrt(b\hat{}2 - 4*a*c) > 0$
using *real-sqrt-gt-0-iff* **by** *blast*
then have $(-\ b - sqrt(b\hat{}2 - 4*a*c)) < (-\ b + sqrt(b\hat{}2 - 4*a*c))$
by *auto*
then have $(-\ b - sqrt(b\hat{}2 - 4*a*c))*(1/(2*a)) < (-\ b + sqrt(b\hat{}2 - 4*a*c))*(1/(2*a))$

using *frac-gtz*
by (*simp add*: *divide-strict-right-mono*)
then have $(-\ b - sqrt(b\hat{}2 - 4*a*c))/(2*a) < (-\ b + sqrt(b\hat{}2 - 4*a*c))/(2*a)$
by *auto*
then have *xandy*: $x = (-\ b - sqrt(b\hat{}2 - 4*a*c))/(2*a) \wedge y = (-\ b + sqrt(b\hat{}2 - 4*a*c))/(2*a)$
using *xy-prop xyroots* **by** *auto*
let *?mdpt* $= -b/(2*a)$
have *xlt*: $x <$ *?mdpt*
using *xandy sqrt-gt* **using** *frac-gtz divide-minus-left divide-strict-right-mono sqrt-gt*
by (*smt* (*verit*) *agt*)
have *ylt*: *?mdpt* $< y$
using *xandy sqrt-gt frac-gtz*
by (*smt* (*verit*, *del-insts*) *divide-strict-right-mono zero-less-divide-1-iff*)
have *mdpt-val*: $a*?mdpt\hat{}2 + b*?mdpt + c < 0$
proof −
have *firsteq*: $a*?mdpt\hat{}2 + b*?mdpt + c = (a*b\hat{}2)/(4*a\hat{}2) - (b\hat{}2)/(2*a) + c$
by (*simp add*: *power2-eq-square*)
have *h1*: $(a*b\hat{}2)/(4*a\hat{}2) = (b\hat{}2)/(4*a)$

```
    by (simp add: power2-eq-square)
   have h2: (b^2)/(2*a) = (2*b^2)/(4*a)
    by linarith
   have h3: c = (4*a*c)/(4*a)
    using agt by auto
  have a*?mdpt^2 + b*?mdpt + c = (b^2)/(4*a) - (2*b^2)/(4*a) + (4*a*c)/(4*a)

    using firsteq h1 h2 h3
    by linarith
   then have a*?mdpt^2 + b*?mdpt + c = (b^2 - 2*b^2 + 4*a*c)/(4*a)
    by (simp add: diff-divide-distrib)
   then have eq2: a*?mdpt^2 + b*?mdpt + c = (4*a*c - b^2)/(4*a)
    by simp
   have h: 4*a*c - b^2 < 0 using pos-discrim by auto
   have 1/(4*a) > 0 using agt by auto
   then have (4*a*c - b^2)*(1/(4*a)) < 0
    using h
    using mult-less-0-iff by blast
   then show ?thesis using eq2 by auto
  qed
  have nex: ¬ (∃k> x. k < y ∧ poly [:c, b, a:] k = 0)
   using discriminant-iff agt
   by (metis discrim-def less-irrefl quadratic-poly-eval xandy)
  have nor2: ¬ (∃x>z. x < - b / (2 * a) ∧ poly [:c, b, a:] x = 0)
   using nex xlt ylt zgt zlt by auto
  have nor: ¬ (∃x>- b / (2 * a). x < z ∧ poly [:c, b, a:] x = 0)
   using nex xlt ylt zgt zlt discriminant-iff agt  by auto
  then have mdpt-lt: ?mdpt < z ⟹ a*z^2 + b*z + c < 0
    using mdpt-val zgt zlt xlt ylt poly-IVT-pos[where p = [:c, b, a:], where a=
?mdpt, where b = z]
     quadratic-poly-eval[of c b a]
   by (metis ‹¬ (∃k>x. k < y ∧ poly [:c, b, a:] k = 0)› linorder-neqE-linordered-idom)

  have mdpt-gt: ?mdpt > z ⟹ a*z^2 + b*z + c < 0
    using zgt zlt mdpt-val xlt ylt nor2 poly-IVT-neg[where p = [:c, b, a:], where
a = z, where b = ?mdpt] quadratic-poly-eval[of c b a]
    by (metis linorder-neqE-linordered-idom nex)
  then show a*z^2 + b*z + c < 0
    using mdpt-lt mdpt-gt mdpt-val by fastforce
qed

lemma quadratic-shape1b:
  fixes a b c x y::real
  assumes agt: a > 0
  assumes xy-roots: x < y ∧ a*x^2 + b*x + c = 0 ∧ a*y^2 + b*y + c = 0
  shows ⋀z. (z > y ⟹ a*z^2 + b*z + c > 0)
proof −
  fix z
  assume z-gt :z > y
```

have *nogt*: $\neg(\exists\, w.\ w > y \wedge a*w\hat{}2 + b*w + c = 0)$ **using** *xy-roots discriminant-iff*
by (*metis agt less-eq-real-def linorder-not-less*)
have $\exists(w{::}real).\ \forall(y{::}real).\ (y > w \longrightarrow a*y\hat{}2 + b*y + c > 0)$
using *agt pos-lc-dom-quad* **by** *auto*
then have $\exists\, k > y.\ \ a*k\hat{}2 + b*k + c > 0$
by (*metis add.commute agt less-add-same-cancel1 linorder-neqE-linordered-idom pos-add-strict*)
then obtain *k* **where** *k-prop*: $k > y \wedge a*k\hat{}2 + b*k + c > 0$ **by** *auto*
have *kgt*: $k > z \Longrightarrow a*z\hat{}2 + b*z + c > 0$
proof −
assume *kgt*: $k > z$
then have *zneq*: $a*z\hat{}2 + b*z + c = 0 \Longrightarrow False$
using *nogt* **using** *z-gt* **by** *blast*
have *znlt*: $a*z\hat{}2 + b*z + c < 0 \Longrightarrow False$
using *kgt k-prop quadratic-poly-eval*[*of c b a*] *z-gt nogt poly-IVT-pos*[**where** $a = z$, **where** $b = k$, **where** $p = [:c,\ b,\ a:]$]
by (*metis less-eq-real-def less-le-trans*)
then show $a*z\hat{}2 + b*z + c > 0$ **using** *zneq znlt*
using *linorder-neqE-linordered-idom* **by** *blast*
qed
have *klt*: $k < z \Longrightarrow a*z\hat{}2 + b*z + c > 0$
proof −
assume *klt*: $k < z$
then have *zneq*: $a*z\hat{}2 + b*z + c = 0 \Longrightarrow False$
using *nogt* **using** *z-gt* **by** *blast*
have *znlt*: $a*z\hat{}2 + b*z + c < 0 \Longrightarrow False$
using *klt k-prop quadratic-poly-eval*[*of c b a*] *z-gt nogt poly-IVT-neg*[**where** $a = k$, **where** $b = z$, **where** $p = [:c,\ b,\ a:]$]
by (*metis add.commute add-less-cancel-left add-mono-thms-linordered-field*(*3*) *less-eq-real-def*)
then show $a*z\hat{}2 + b*z + c > 0$ **using** *zneq znlt*
using *linorder-neqE-linordered-idom* **by** *blast*
qed
then show $a*z\hat{}2 + b*z + c > 0$ **using** *k-prop kgt klt*
by *fastforce*
qed

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 $\bigwedge z.\ (z > x \wedge z < y \Longrightarrow a*z\hat{}2 + b*z + c > 0)$
using *quadratic-shape1a*[**where** $a = -a$, **where** $b = -b$, **where** $c = -c$, **where** $x = x$, **where** $y = y$]
using *assms*(*1*) *assms*(*2*) **by** *fastforce*

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 $\bigwedge z.\ (z > y \Longrightarrow a*z\hat{}2 + b*z + c < 0)$
using *quadratic-shape1b*[**where** $a = -a$, **where** $b = -b$, **where** $c = -c$, **where** $x = x$, **where** $y = y$]
using *assms*(*1*) *assms*(*2*) **by** *force*

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 −
assume *b-lt*: $b < 2*a*r$
assume *root*: $a*r\hat{}2 - b*r + c = 0$
then have $c = b*r - a*r\hat{}2$ **by** *auto*
have *aeq*: $a = 0 \Longrightarrow \exists\, y'{>}{-}\ r.\ \forall\, x{\in}\{-r{<}..y'\}.\ a * x^2 + b * x + c < 0$
proof −
assume *azer*: $a = 0$
then have *bltz*: $b < 0$ **using** *b-lt* **by** *auto*
then have $c = b*r$ **using** *azer root* **by** *auto*
then have *eval*: $\forall\, x.\ a*x\hat{}2 + b*x + c = b*(x + r)$ **using** *azer*
by (*simp add*: *distrib-left*)
have $\forall\, x > -r.\ b*(x + r) < 0$
proof *clarsimp*
fix *x*
assume *xgt*: $-\ r < x$
then have $x + r > 0$
by *linarith*
then show $b * (x + r) < 0$
using *bltz* **using** *mult-less-0-iff* **by** *blast*
qed
then show *?thesis* **using** *eval*
using *less-add-same-cancel1 zero-less-one*
by (*metis greaterThanAtMost-iff*)
qed
have *aneq*: $a \neq 0 \Longrightarrow \exists\, y'{>}{-}\ r.\ \forall\, x{\in}\{-r{<}..y'\}.\ a * x^2 + b * x + c < 0$
proof −
assume *aneq*: (*a*::*real*) $\neq 0$
have $b\hat{}2 - 4*a*c < 0 \Longrightarrow a * r^2 + b * r + c \neq 0$ **using** *root discriminant-negative*[*of a b c r*] **unfolding** *discrim-def*
using *aneq* **by** *auto*
then have $a * r^2 + b * r + c \neq 0 \Longrightarrow$
$a * r^2 - b * r + c = 0 \Longrightarrow$
$b^2 < 4 * a * c \Longrightarrow$ *False*
proof −
assume *a1*: $a * r^2 - b * r + c = 0$
assume *a2*: $b^2 < 4 * a * c$
have *f3*: $(0 \leq -\ 1 * (4 * a * c) + (-\ 1 * b)^2) = (4 * a * c + -\ 1 * (-\ 1 * b)^2 \leq 0)$

by *simp*
have *f4*: $(-\ 1 * b)^2 + -\ 1 * (4 * a * c) = -\ 1 * (4 * a * c) + (-\ 1 * b)^2$
by *auto*
have *f5*: $c + a * r^2 + -\ 1 * b * r = a * r^2 + c + -\ 1 * b * r$
by *auto*
have *f6*: $\forall\, x0\ x1\ x2\ x3.\ (x3::real) * x0^2 + x2 * x0 + x1 = x1 + x3 * x0^2 + x2 * x0$
by *simp*
have *f7*: $\forall\, x1\ x2\ x3.\ (discrim\ x3\ x2\ x1 < 0) = (\neg\ 0 \leq discrim\ x3\ x2\ x1)$
by *auto*
have *f8*: $\forall\, r\ ra\ rb.\ discrim\ r\ ra\ rb = ra^2 + -\ 1 * (4 * r * rb)$
using *discrim-def* **by** *auto*
have $\neg\ 4 * a * c + -\ 1 * (-\ 1 * b)^2 \leq 0$
using *a2* **by** *simp*
then have $a * r^2 + c + -\ 1 * b * r \neq 0$
using *f8 f7 f6 f5 f4 f3* **by** (*metis* (*no-types*) *aneq discriminant-negative*)
then show *False*
using *a1* **by** *linarith*
qed
then have $\neg(b\hat{}2 - 4*a*c < 0)$ **using** *root*
using $\langle b^2 - 4 * a * c < 0 \Longrightarrow a * r^2 + b * r + c \neq 0 \rangle$ **by** *auto*
then have *discrim*: $b^2 \geq 4 * a * c$ **by** *auto*
then have *req*: $r = (b + sqrt(b\hat{}2 - 4*a*c))/(2*a) \vee r = (b - sqrt(b\hat{}2 - 4*a*c))/(2*a)$
using *aneq root discriminant-iff*[**where** a=a, **where** b =−b, **where** c=c, **where** x=r] **unfolding** *discrim-def*
by *auto*
then have $r = (b - sqrt(b\hat{}2 - 4*a*c))/(2*a) \Longrightarrow b > 2*a*r$
proof −
assume *req*: $r = (b - sqrt(b\hat{}2 - 4*a*c))/(2*a)$
then have *h1*: $2*a*r = 2*a*((b - sqrt(b\hat{}2 - 4*a*c))/(2*a))$ **by** *auto*
then have *h2*: $2*a*((b - sqrt(b\hat{}2 - 4*a*c))/(2*a)) = b - sqrt(b\hat{}2 - 4*a*c)$
using *aneq* **by** *auto*
have *h3*: $sqrt(b\hat{}2 - 4*a*c) \geq 0$ **using** *discrim* **by** *auto*
then have $b - sqrt(b\hat{}2 - 4*a*c) < b$
using *b-lt h1 h2* **by** *linarith*
then show *?thesis* **using** *req h2* **by** *auto*
qed
then have *req*: $r = (b + sqrt(b\hat{}2 - 4*a*c))/(2*a)$ **using** *req b-lt* **by** *auto*
then have *discrim2*: $b\hat{}2 - 4 * a*c > 0$ **using** *aneq b-lt* **by** *auto*
then have $\exists\, x\ y.\ x \neq y \wedge a * x^2 + b * x + c = 0 \wedge a * y^2 + b * y + c = 0$
using *aneq discriminant-pos-ex*[*of a b c*] **unfolding** *discrim-def*
by *auto*
then obtain $x\ y$ **where** *xy-prop*: $x < y \wedge a * x^2 + b * x + c = 0 \wedge a * y^2 + b * y + c = 0$
by (*meson linorder-neqE-linordered-idom*)
then have $(x = (-b + sqrt(b\hat{}2 - 4*a*c))/(2*a) \wedge y = (-b - sqrt(b\hat{}2 - 4*a*c))/(2*a))$

$\vee$ $(y = (-b + sqrt(b\hat{}2 - 4*a*c))/(2*a) \wedge x = (-b - sqrt(b\hat{}2 - 4*a*c))/(2*a))$

using *aneq discriminant-iff* **unfolding** *discrim-def* **by** *auto*
then have *xy-prop2*: $(x = (-b + sqrt(b\hat{}2 - 4*a*c))/(2*a) \wedge y = -r)$
$\vee$ $(y = (-b + sqrt(b\hat{}2 - 4*a*c))/(2*a) \wedge x = -r)$ **using** *req*
by (*simp add*: ‹$x = (-\ b + sqrt\ (b^2 - 4 * a * c))\ /\ (2 * a) \wedge y = (-\ b - sqrt\ (b^2 - 4 * a * c))\ /\ (2 * a) \vee y = (-\ b + sqrt\ (b^2 - 4 * a * c))\ /\ (2 * a) \wedge x = (-\ b - sqrt\ (b^2 - 4 * a * c))\ /\ (2 * a)$› *minus-divide-left*)

have *alt*: $a < 0 \Longrightarrow \forall k > -r.\ a * k\hat{}2 + b * k + c < 0$
proof *clarsimp*
fix *k*
assume *alt*: $a < 0$
assume $-\ r < k$
have *alt2*: $(1/(2*a)::real) < 0$ **using** *alt*
by *simp*
have $(-b - sqrt(b\hat{}2 - 4*a*c)) < (-b + sqrt(b\hat{}2 - 4*a*c))$
using *discrim2* **by** *auto*
then have $(-b - sqrt(b\hat{}2 - 4*a*c))* (1/(2*a)::real) > (-b + sqrt(b\hat{}2 - 4*a*c))* (1/(2*a)::real)$
using *alt2*
using *mult-less-cancel-left-neg* **by** *fastforce*
then have *rgtroot*: $-r > (-b + sqrt(b\hat{}2 - 4*a*c))/(2*a)$
using *req* ‹$x = (-\ b + sqrt\ (b^2 - 4 * a * c))\ /\ (2 * a) \wedge y = (-\ b - sqrt\ (b^2 - 4 * a * c))\ /\ (2 * a) \vee y = (-\ b + sqrt\ (b^2 - 4 * a * c))\ /\ (2 * a) \wedge x = (-\ b - sqrt\ (b^2 - 4 * a * c))\ /\ (2 * a)$› *xy-prop2*
by *auto*
then have $(y = -r \wedge x = (-b + sqrt(b\hat{}2 - 4*a*c))/(2*a))$
using *xy-prop xy-prop2* **by** *auto*
then show $a * k\hat{}2 + b * k + c < 0$
using *xy-prop* ‹$-\ r < k$› *alt quadratic-shape2b xy-prop*
by *blast*
qed

have *agt*: $a > 0 \Longrightarrow \exists y'>-\ r.\ \forall x\in\{-r<..y'\}.\ a * x^2 + b * x + c < 0$
proof $-$
assume *agt*: $a> 0$
have *alt2*: $(1/(2*a)::real) > 0$ **using** *agt*
by *simp*
have $(-b - sqrt(b\hat{}2 - 4*a*c)) < (-b + sqrt(b\hat{}2 - 4*a*c))$
using *discrim2* **by** *auto*
then have $(-b - sqrt(b\hat{}2 - 4*a*c))* (1/(2*a)::real) < (-b + sqrt(b\hat{}2 - 4*a*c))* (1/(2*a)::real)$
using *alt2*
proof $-$
have *f1*: $-\ b - sqrt\ (b^2 - c * (4 * a)) < -\ b + sqrt\ (b^2 - c * (4 * a))$
by (*metis* ‹$-\ b - sqrt\ (b^2 - 4 * a * c) < -\ b + sqrt\ (b^2 - 4 * a * c)$› *mult.commute*)
have $0 < a * 2$

```
      using ‹0 < 1 / (2 * a)› by auto
    then show ?thesis
      using f1 by (simp add: divide-strict-right-mono mult.commute)
  qed
  then have rlltroot: −r <  (−b + sqrt(b^2 − 4∗a∗c))/(2∗a)
    using req ‹x = (− b + sqrt (b² − 4 * a * c)) / (2 * a) ∧ y = (− b − sqrt
(b² − 4 * a * c)) / (2 * a) ∨ y = (− b + sqrt (b² − 4 * a * c)) / (2 * a) ∧ x =
(− b − sqrt (b² − 4 * a * c)) / (2 * a)› xy-prop2
    by auto
  then have (x = −r ∧ y = (−b + sqrt(b^2 − 4∗a∗c))/(2∗a))
    using xy-prop xy-prop2 by auto
  have ∃k. x < k ∧ k < y using xy-prop dense by auto
  then obtain k where k-prop: x < k ∧ k < y by auto
  then have ∀x∈{−r<..k}. a * x² + b * x + c < 0
   using agt quadratic-shape1a[where a= a, where b = b, where c= c, where
x = x, where y = y]
      using ‹x = − r ∧ y = (− b + sqrt (b² − 4 * a * c)) / (2 * a)›
greaterThanAtMost-iff xy-prop by auto
  then show ∃y'>− r. ∀x∈{−r<..y'}. a * x² + b * x + c < 0
    using k-prop using ‹x = − r ∧ y = (− b + sqrt (b² − 4 * a * c)) / (2 *
a)› by blast
 qed
 show ?thesis
  using alt agt
 by (metis aneq greaterThanAtMost-iff less-add-same-cancel1 linorder-neqE-linordered-idom
zero-less-one)
qed
show ∃y'>− r. ∀x∈{−r<..y'}. a * x² + b * x + c < 0 using aeq aneq
 by blast
qed

lemma case-d4:
 fixes a b c r::real
 shows ⋀y'. b ≠ 2 * a * r ⟹
       ¬ b < 2 * a * r ⟹
       a ∗r^2 − b * r + c = 0 ⟹
       −r < y' ⟹ ∃x∈{−r<..y'}. ¬ a * x² + b * x + c < 0
proof −
 fix y'
 assume bneq: b ≠ 2 * a * r
 assume bnotless: ¬ b < 2 * a * r
 assume root: a ∗r^2 − b * r + c = 0
 assume y-prop: −r < y'
 have b-gt: b > 2∗a∗r using bneq bnotless by auto
 have aeq: a = 0 ⟹ ∃y'>− r. ∀x∈{−r<..y'}. a * x² + b * x + c > 0
 proof −
  assume azer: a = 0
  then have bgt: b > 0 using b-gt by auto
  then have c = b∗r using azer root by auto
```

then have *eval*: $\forall x.\ a*x\hat{}2 + b*x + c = b*(x + r)$ **using** *azer*
by (*simp add*: *distrib-left*)
have $\forall x > -r.\ b*(x + r) > 0$
proof *clarsimp*
fix *x*
assume *xgt*: $-\ r < x$
then have $x + r > 0$
by *linarith*
then show $b * (x + r) > 0$
using *bgt* **by** *auto*
qed
then show *?thesis* **using** *eval*
using *less-add-same-cancel1 zero-less-one*
by (*metis greaterThanAtMost-iff*)
qed
have *aneq*: $a \neq 0 \Longrightarrow \exists\, y' > -\ r.\ \forall x \in \{-r<..y'\}.\ a * x^2 + b * x + c > 0$
proof −
assume *aneq*: *a≠0*
{
assume *a1*: $a * r^2 - b * r + c = 0$
assume *a2*: $b^2 < 4 * a * c$
have *f3*: $(0 \leq -\ 1 * (4 * a * c) + (-\ 1 * b)^2) = (4 * a * c + -\ 1 * (-\ 1 * b)^2 \leq 0)$
by *simp*
have *f4*: $(-\ 1 * b)^2 + -\ 1 * (4 * a * c) = -\ 1 * (4 * a * c) + (-\ 1 * b)^2$
by *auto*
have *f5*: $c + a * r^2 + -\ 1 * b * r = a * r^2 + c + -\ 1 * b * r$
by *auto*
have *f6*: $\forall x0\ x1\ x2\ x3.\ (x3::real) * x0^2 + x2 * x0 + x1 = x1 + x3 * x0^2 + x2 * x0$
by *simp*
have *f7*: $\forall x1\ x2\ x3.\ (discrim\ x3\ x2\ x1 < 0) = (\neg\ 0 \leq discrim\ x3\ x2\ x1)$
by *auto*
have *f8*: $\forall r\ ra\ rb.\ discrim\ r\ ra\ rb = ra^2 + -\ 1 * (4 * r * rb)$
using *discrim-def* **by** *auto*
have $\neg\ 4 * a * c + -\ 1 * (-\ 1 * b)^2 \leq 0$
using *a2* **by** *simp*
then have $a * r^2 + c + -\ 1 * b * r \neq 0$
using *f8 f7 f6 f5 f4 f3* **by** (*metis* (*no-types*) *aneq discriminant-negative*)
then have *False*
using *a1* **by** *linarith*
} **note** $* = this$
have $b\hat{}2 - 4*a*c < 0 \Longrightarrow a * r^2 + b * r + c \neq 0$ **using** *root discriminant-negative*[*of a b c r*] **unfolding** *discrim-def*
using *aneq* **by** *auto*
then have $\neg(b\hat{}2 - 4*a*c < 0)$ **using** *root* $*$ **by** *auto*
then have *discrim*: $b^2 \geq 4 * a * c$ **by** *auto*
then have *req*: $r = (b + sqrt(b\hat{}2 - 4*a*c))/(2*a) \vee r = (b - sqrt(b\hat{}2 - 4*a*c))/(2*a)$

using *aneq root discriminant-iff*[**where** a=a, **where** b =−b, **where** c=c, **where** x=r] **unfolding** *discrim-def*
by *auto*
then have $r = (b + sqrt(b\hat{}2 - 4*a*c))/(2*a) \Longrightarrow b < 2*a*r$
proof −
assume *req*: $r = (b + sqrt(b\hat{}2 - 4*a*c))/(2*a)$
then have *h1*: $2*a*r = 2*a*((b + sqrt(b\hat{}2 - 4*a*c))/(2*a))$ **by** *auto*
then have *h2*: $2*a*((b + sqrt(b\hat{}2 - 4*a*c))/(2*a)) = b + sqrt(b\hat{}2 - 4*a*c)$
using *aneq* **by** *auto*
have *h3*: $sqrt(b\hat{}2 - 4*a*c) \geq 0$ **using** *discrim* **by** *auto*
then have $b + sqrt(b\hat{}2 - 4*a*c) > b$
using *b-gt h1 h2* **by** *linarith*
then show *?thesis* **using** *req h2* **by** *auto*
qed
then have *req*: $r = (b - sqrt(b\hat{}2 - 4*a*c))/(2*a)$ **using** *req b-gt*
using *aneq discrim* **by** *auto*
then have *discrim2*: $b\hat{}2 - 4 *a*c > 0$ **using** *aneq b-gt* **by** *auto*
then have $\exists x\ y.\ x \neq y \wedge a * x^2 + b * x + c = 0 \wedge a * y^2 + b * y + c = 0$
using *aneq discriminant-pos-ex*[*of a b c*] **unfolding** *discrim-def*
by *auto*
then obtain $x\ y$ **where** *xy-prop*: $x < y \wedge a * x^2 + b * x + c = 0 \wedge a * y^2 + b * y + c = 0$
by (*meson linorder-neqE-linordered-idom*)
then have $(x = (-b + sqrt(b\hat{}2 - 4*a*c))/(2*a) \wedge y = (-b - sqrt(b\hat{}2 - 4*a*c))/(2*a))$
$\vee\ (y = (-b + sqrt(b\hat{}2 - 4*a*c))/(2*a) \wedge x = (-b - sqrt(b\hat{}2 - 4*a*c))/(2*a))$

using *aneq discriminant-iff* **unfolding** *discrim-def* **by** *auto*
then have *xy-prop2*: $(x = (-b - sqrt(b\hat{}2 - 4*a*c))/(2*a) \wedge y = -r)$
$\vee\ (y = (-b - sqrt(b\hat{}2 - 4*a*c))/(2*a) \wedge x = -r)$ **using** *req divide-inverse minus-diff-eq mult.commute mult-minus-right*
by (*smt* (*verit*, *ccfv-SIG*) *uminus-add-conv-diff*)

have *agt*: $a > 0 \Longrightarrow \forall k > -r.\ a * k\hat{}2 + b * k + c > 0$
proof *clarsimp*
fix k
assume *agt*: $a > 0$
assume $-\ r < k$
have *agt2*: $(1/(2*a)::real) > 0$ **using** *agt*
by *simp*
have $(-b - sqrt(b\hat{}2 - 4*a*c)) < (-b + sqrt(b\hat{}2 - 4*a*c))$
using *discrim2* **by** *auto*
then have $(-b - sqrt(b\hat{}2 - 4*a*c))* (1/(2*a)::real) < (-b + sqrt(b\hat{}2 - 4*a*c))* (1/(2*a)::real)$
using *agt2* **by** (*simp add*: *divide-strict-right-mono*)
then have *rgtroot*: $-r > (-b - sqrt(b\hat{}2 - 4*a*c))/(2*a)$
using *req* ‹$x = (-\ b + sqrt\ (b^2 - 4 * a * c))\ /\ (2 * a) \wedge y = (-\ b - sqrt\ (b^2 - 4 * a * c))\ /\ (2 * a) \vee y = (-\ b + sqrt\ (b^2 - 4 * a * c))\ /\ (2 * a) \wedge x =$

$(-\ b - sqrt\ (b^2 - 4 * a * c))\ /\ (2 * a)\rangle$ *xy-prop2*
by *auto*
then have $(x = (-b - sqrt(b\hat{}2 - 4*a*c))/(2*a)) \wedge y = -r$
using *xy-prop xy-prop2*
by *auto*
then show $a * k\hat{}2 + b * k + c > 0$
using $\langle -\ r < k\rangle$ *xy-prop agt quadratic-shape1b*[**where** a= a, **where** b =b, **where** c=c, **where** $x = x$, **where** $y = -r$, **where** $z = k$]
by *blast*
qed

have *agt2*: $a < 0 \Longrightarrow \exists\, y'>-\ r.\ \forall\, x{\in}\{-r<..y'\}.\ a * x^2 + b * x + c > 0$
proof −
assume *alt*: *a<0*
have *alt2*: $(1/(2*a)::real) < 0$ **using** *alt*
by *simp*
have $(-b - sqrt(b\hat{}2 - 4*a*c)) < (-b + sqrt(b\hat{}2 - 4*a*c))$
using *discrim2* **by** *auto*
then have $(-b - sqrt(b\hat{}2 - 4*a*c))* (1/(2*a)::real) > (-b + sqrt(b\hat{}2 - 4*a*c))* (1/(2*a)::real)$
using *alt2* **using** *mult-less-cancel-left-neg* **by** *fastforce*
then have *rlltroot*: $-r < (-b - sqrt(b\hat{}2 - 4*a*c))/(2*a)$
using *req*
using $\langle x = (-\ b + sqrt\ (b^2 - 4 * a * c))\ /\ (2 * a) \wedge y = (-\ b - sqrt\ (b^2 - 4 * a * c))\ /\ (2 * a) \vee y = (-\ b + sqrt\ (b^2 - 4 * a * c))\ /\ (2 * a) \wedge x = (-\ b - sqrt\ (b^2 - 4 * a * c))\ /\ (2 * a)\rangle$
xy-prop2
by *auto*
then have *h*: $(x = -r \wedge y = (-b - sqrt(b\hat{}2 - 4*a*c))/(2*a))$
using *xy-prop xy-prop2*
by *auto*
have $\exists\, k.\ x < k \wedge k < y$ **using** *xy-prop dense* **by** *auto*
then obtain *k* **where** *k-prop*: $x < k \wedge k < y$ **by** *auto*
then have $\forall\, x{\in}\{-r<..k\}.\ a * x^2 + b * x + c > 0$
using *alt quadratic-shape2a*[**where** a= a, **where** b = b, **where** c= c, **where** $x = x$, **where** $y = y$]
xy-prop h greaterThanAtMost-iff **by** *auto*
then show $\exists\, y'>-\ r.\ \forall\, x{\in}\{-r<..y'\}.\ a * x^2 + b * x + c > 0$
using *k-prop* **using** *h* **by** *blast*
qed
show *?thesis*
using *aneq agt agt2*
by (*meson greaterThanAtMost-iff linorder-neqE-linordered-idom y-prop*)
qed
show $\exists\, x{\in}\{-r<..y'\}.\ \neg\ a * x^2 + b * x + c < 0$ **using** *aneq aeq*
by (*metis greaterThanAtMost-iff less-eq-real-def linorder-not-less y-prop*)
qed

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 −
have *root-iff*: $\forall\, x.\ a * x^2 + b * x + c = 0 \longleftrightarrow x = -r$ **using** *alt root discriminant-lemma*[**where** $r = r$] *beq*
by *auto*
have $a < 0 \longrightarrow (\exists\, y.\ \forall\, x > y.\ a{*}x\hat{}2 + b{*}x + c < 0)$ **using** *neg-lc-dom-quad*
by *auto*
then obtain k **where** *k-prop*: $\forall\, x > k.\ a{*}x\hat{}2 + b{*}x + c < 0$ **using** *alt* **by** *auto*
let *?mx* = *max* $(k{+}1)\ (-r + 1)$
have $a{*}$*?mx*$\hat{}2 + b{*}$*?mx* $+ c < 0$ **using** *k-prop* **by** *auto*
then have $\exists\, y > -r.\ a{*}y\hat{}2 + b{*}y + c < 0$
by *force*
then obtain z **where** *z-prop*: $z > -r \wedge a{*}z\hat{}2 + b{*}z + c < 0$ **by** *auto*
have *poly-eval-prop*: $\forall\, (x{::}real).\ poly\ [:c,\ b,\ a\ :]\ x = a{*}x\hat{}2 + b{*}x + c$
using *quadratic-poly-eval* **by** *auto*
then have *nozer*: $\neg(\exists\, x.\ (x > -r \wedge poly\ [:c,\ b,\ a\ :]\ x = 0))$ **using** *root-iff*
by (*simp add*: *add.commute*)
have *poly-z*: *poly* [:*c*, *b*, *a*:] $z < 0$ **using** *z-prop poly-eval-prop* **by** *auto*
have $\forall\, y > -r.\ a{*}y\hat{}2 + b{*}y + c < 0$
proof *clarsimp*
fix y
assume *ygt*: $-\ r < y$
have *h1*: $y = z \Longrightarrow a * y^2 + b * y + c < 0$ **using** *z-prop* **by** *auto*
have *h2*: $y < z \Longrightarrow a * y^2 + b * y + c < 0$ **proof** −
assume *ylt*: $y < z$
have *notz*: $a{*}y\hat{}2 + b{*}y + c \neq 0$ **using** *ygt nozer poly-eval-prop* **by** *auto*
have *h1*: $a\ {*}y\hat{}2 + b{*}y + c > 0 \Longrightarrow$ *poly* [:*c*, *b*, *a*:] $y > 0$ **using** *poly-eval-prop* **by** *auto*
have *ivtprop*: *poly* [:*c*, *b*, *a*:] $y > 0 \Longrightarrow (\exists\, x.\ y < x \wedge x < z \wedge poly\ [:c,\ b,\ a:]\ x = 0)$
using *ylt poly-z poly-IVT-neg*[**where** $a = y$, **where** $b = z$, **where** $p = [:c,\ b,\ a:]$]
by *auto*
then have $a{*}y\hat{}2 + b{*}y + c > 0 \Longrightarrow$ *False* **using** *h1 ivtprop ygt nozer* **by** *auto*
then show $a{*}y\hat{}2 + b{*}y + c < 0$ **using** *notz*
using *linorder-neqE-linordered-idom* **by** *blast*
qed
have *h3*: $y > z \Longrightarrow a * y^2 + b * y + c < 0$
proof −
assume *ygtz*: $y > z$
have *notz*: $a{*}y\hat{}2 + b{*}y + c \neq 0$ **using** *ygt nozer poly-eval-prop* **by** *auto*
have *h1*: $a\ {*}y\hat{}2 + b{*}y + c > 0 \Longrightarrow$ *poly* [:*c*, *b*, *a*:] $y > 0$ **using** *poly-eval-prop* **by** *auto*
have *ivtprop*: *poly* [:*c*, *b*, *a*:] $y > 0 \Longrightarrow (\exists\, x.\ z < x \wedge x < y \wedge poly\ [:c,\ b,\ a:]$

$x = 0$)
  using *ygtz poly-z* **using** *poly-IVT-pos* **by** *blast*
 then have $a*y\hat{}2 + b*y + c > 0 \Longrightarrow$ *False* **using** *h1 ivtprop z-prop nozer* **by** *auto*
 then show $a*y\hat{}2 + b*y + c < 0$ **using** *notz*
  using *linorder-neqE-linordered-idom* **by** *blast*
qed
show $a * y^2 + b * y + c < 0$ **using** *h1 h2 h3*
 using *linorder-neqE-linordered-idom* **by** *blast*
qed
then show *?thesis*
 using ‹$\exists\, y{>}{-}\ r.\ a * y^2 + b * y + c < 0$› *beq* **by** *auto*
qed

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 −
have *h1*: $\exists\, y'{>}{-}\ r.\ \forall\, x{\in}\{-\ r{<}..y'\}.\ a * x^2 + 2 * a * r * x + c < 0 \Longrightarrow$
 $\exists\, y'{>}{-}r.\ \forall\, x{\in}\{-\ r{<}..y'\}.\ a * x^2 + 2 * a *r * x + c \leq 0$
 using *less-eq-real-def* **by** *blast*
then show *?thesis*
 using *one-root-a-lt0*[*of a b r*] *assms* **by** *auto*
qed

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 −
assume *a*: $a*r\hat{}2 + b*r + c > 0$
have *h*: $\exists\, x{\in}\{r{<}..k\}.\ a*x\hat{}2 + b*x + c > 0 \Longrightarrow \exists\, x{\in}\{r{<}..k\}.\ a*x\hat{}2 + b*x + c \geq 0$
 using *less-eq-real-def* **by** *blast*
have $\exists\, x{\in}\{r{<}..k\}.\ a*x\hat{}2 + b*x + c > 0$ **using** *a continuity-lem-gt0-expanded*[*of r k a b c*] *assms* **by** *auto*
then show $\exists\, x{\in}\{r{<}..k\}.\ a*x\hat{}2 + b*x + c \geq 0$
 using *h* **by** *auto*
qed

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 −
assume $a*r\hat{}2 + b*r + c < 0$
then have $\exists\, y'> r.\ \forall\, x\in\{r<..y'\}.\ a*x\hat{}2 + b*x + c < 0$
using *continuity-lem-lt0-expanded* **by** *auto*
then show $\exists\, y'> r.\ \forall\, x\in\{r<..y'\}.\ a*x\hat{}2 + b*x + c \leq 0$
using *less-eq-real-def* **by** *auto*
qed

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 −
assume $a\neq 0 \vee b\neq 0 \vee c\neq 0$
then have *fin*: *finite* $\{x.\ a*x\hat{}2 + b*x + c = 0\}$
by (*metis pCons-eq-0-iff poly-roots-finite poly-roots-set-same*)

let *?s* $= \{x.\ a*x\hat{}2 + b*x + c = 0\}$
have *imp*: $(\exists\, q \in ?s.\ q > r) \Longrightarrow (\exists\, q \in ?s.\ (q > r \wedge (\forall\, x \in ?s.\ x > r \longrightarrow x \geq q)))$
proof −
assume *asm*: $(\exists\, q \in ?s.\ q > r)$
then have *none*: $\{q.\ q \in ?s \wedge q > r\} \neq \{\}$
by *blast*
have *fin2*: *finite* $\{q.\ q \in ?s \wedge q > r\}$ **using** *fin*
by *simp*
have $\exists\, k.\ k = Min\ \{q.\ q \in ?s \wedge q > r\}$ **using** *fin2 none*
by *blast*
then obtain k **where** *k-prop*: $k = Min\ \{q.\ q \in ?s \wedge q > r\}$ **by** *auto*
then have *kp1*: $k \in ?s \wedge k > r$
using *Min-in fin2 none*
by *blast*
then have *kp2*: $\forall\, x \in ?s.\ x > r \longrightarrow x \geq k$
using *Min-le fin2* **using** *k-prop* **by** *blast*
show $(\exists\, q \in ?s.\ (q > r \wedge (\forall\, x \in ?s.\ x > r \longrightarrow x \geq q)))$
using *kp1 kp2* **by** *blast*
qed
have *h2*: $(\exists\, q \in ?s.\ q > r) \Longrightarrow \exists\, y'>r.\ \forall\, x\in\{r<..y'\}.\ a * x^2 + b * x + c \neq 0$
proof −
assume $(\exists\, q \in ?s.\ q > r)$
then obtain q **where** *q-prop*: $q \in ?s \wedge (q > r \wedge (\forall\, x \in ?s.\ x > r \longrightarrow x \geq q))$
using *imp* **by** *blast*
then have $\exists\, w.\ w > r \wedge w < q$ **using** *dense*
by *blast*
then obtain w **where** *w-prop*: $w > r \wedge w < q$ **by** *auto*

then have $\neg(\exists\, x{\in}\{r{<}..w\}.\ x \in ?s)$
using *w-prop q-prop* **by** *auto*
then have $\forall\, x{\in}\{r{<}..w\}.\ a * x^2 + b * x + c \neq 0$
by *blast*
then show $\exists\, y'{>}r.\ \forall\, x{\in}\{r{<}..y'\}.\ a * x^2 + b * x + c \neq 0$
using *w-prop* **by** *blast*
qed
have *h1*: $\neg(\exists\, q \in ?s.\ q > r) \Longrightarrow \exists\, y'{>}r.\ \forall\, x{\in}\{r{<}..y'\}.\ a * x^2 + b * x + c \neq 0$
proof −
assume $\neg(\exists\, q \in ?s.\ q > r)$
then have $\forall\, x{\in}\{r{<}..(r{+}1)\}.\ a * x^2 + b * x + c \neq 0$
using *greaterThanAtMost-iff* **by** *blast*
then show *?thesis*
using *less-add-same-cancel1 less-numeral-extra*(*1*) **by** *blast*
qed
then show $\exists\, y'{>}r.\ \forall\, x{\in}\{r{<}..y'\}.\ a * x^2 + b * x + c \neq 0$
using *h2* **by** *blast*
qed

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 −
let *?A* = $\{(x{::}real).\ x < y\}$
have $\exists\ (n{::}nat)\ f.\ ?A = f\ `\ \{i.\ i < n\} \wedge$ *inj-on f* $\{i.\ i < n\} \Longrightarrow$ *False*
proof *clarsimp*
fix *n*:: *nat*
fix *f*
assume *xlt*: $\{x.\ x < y\} = f\ `\ \{i.\ i < n\}$
assume *injh*: *inj-on f* $\{i.\ i < n\}$
have *?A* $\neq \{\}$
by (*simp add*: *linordered-field-no-lb*)
then have *ngtz*: $n > 0$
using *xlt injh* **using** *gr-implies-not-zero* **by** *auto*
have *cardisn*: *card ?A* = n **using** *xlt injh*
by (*simp add*: *card-image*)
have $\forall\, k{::}nat.\ ((y - (k{::}nat) - 1) \in ?A)$
by *auto*
then have *subs*: $\{k.\ \exists\,(x{::}nat).\ k = y - x - 1 \wedge 0 \leq x \wedge x \leq n\} \subseteq ?A$
by *auto*
have *seteq*: $(\lambda x.\ y - real\ x - 1)\ `\ \{0..n\} = \{k.\ \exists\,(x{::}nat).\ k = y - x - 1 \wedge 0 \leq x \wedge x \leq n\}$
by *auto*
have *injf*: *inj-on* $(\lambda x.\ y - real\ x - 1)\ \{0..n\}$
unfolding *inj-on-def* **by** *auto*
have *card* $\{k.\ \exists\,(x{::}nat).\ k = y - x - 1 \wedge 0 \leq x \wedge x \leq n\} = n + 1$

```
    using  injf seteq card-atMost inj-on-iff-eq-card[where A = {0..n}, where f
= λx. y − x − 1]
    by auto
  then have if-fin: finite ?A ⟹ card ?A ≥ n + 1
    using subs card-mono
    by (metis (lifting) card-mono)
  then have if-inf: infinite ?A ⟹ card ?A = 0
    by (meson card.infinite)
  then show False using if-fin if-inf cardisn ngtz by auto
 qed
 then have nfin: ¬ finite {(x::real). x < y}
  using finite-imp-nat-seg-image-inj-on by blast
 have {(x::real). x < y} ⊆ {x. a∗x^2 + b∗x + c = 0}
  using assms by auto
 then have nfin2: ¬ finite {x. a∗x^2 + b∗x + c = 0}
  using nfin finite-subset by blast
 {
  fix x
  assume a ∗ x^2 + b ∗ x + c = 0
  then have f1: a ∗ (x ∗ x) + x ∗ b + c = 0
   by (simp add: Groups.mult-ac(2) power2-eq-square)
  have f2: ∀ r. c + (r + (c + − c)) = r + c
   by simp
  have f3: ∀ r ra rb. (rb::real) ∗ ra + ra ∗ r = (rb + r) ∗ ra
   by (metis Groups.mult-ac(2) Rings.ring-distribs(2))
  have ∀ r. r + (c + − c) = r
   by simp
  then have c + x ∗ (b + x ∗ a) = 0
   using f3 f2 f1 by (metis Groups.add-ac(3) Groups.mult-ac(1) Groups.mult-ac(2))
 }
 hence {x. a∗x^2 + b∗x + c = 0} ⊆ {x. poly [:c, b, a:] x = 0}
  by auto
 then have  ¬ finite {x. poly [:c, b, a:] x = 0}
  using nfin2 using finite-subset by blast
 then have [:c, b, a:] = 0
  using poly-roots-finite[where p = [:c, b, a:]]  by auto
 then show ?thesis
  by auto
qed



lemma have-inbetween-point-leq:
 fixes r::real
 assumes (∀((d::real), (e::real), (f::real))∈set leq. ∃ y'>r. ∀ x∈{r<..y'}. d ∗ x^2 +
e ∗ x + f ≤ 0)
 shows (∃ x. (∀(a, b, c)∈set leq. a ∗ x^2 + b ∗ x + c ≤ 0))
proof −
 have (∀(d, e, f)∈set leq. ∃ y'>r. ∀ x∈{r<..y'}. d ∗ x^2 + e ∗ x + f ≤ 0) ⟹
```

$(\exists\, y'{>}r.\ (\forall\,(d,\ e,\ f){\in}set\ leq.\ \forall\, x{\in}\{r{<}..y'\}.\ d * x^2 + e * x + f \leq 0))$
using *leq-qe-inf-helper assms* **by** *auto*
then have $(\exists\, y'{>}r.\ (\forall\,(d,\ e,\ f){\in}set\ leq.\ \forall\, x{\in}\{r{<}..y'\}.\ d * x^2 + e * x + f \leq 0))$
using *assms*
by *blast*
then obtain *y* **where** *y-prop*: $y > r \wedge\ (\forall\,(d,\ e,\ f){\in}set\ leq.\ \forall\, x{\in}\{r{<}..y\}.\ d * x^2 + e * x + f \leq 0)$
by *auto*
have $\exists\, q.\ q >\ r \wedge q < y$ **using** *y-prop dense* **by** *auto*
then obtain *q* **where** *q-prop*: $q > r \wedge q < y$ **by** *auto*
then have $(\forall\,(d,\ e,\ f){\in}set\ leq.\ d{*}q\hat{}2 + e{*}q + f \leq 0)$
using *y-prop* **by** *auto*
then show *?thesis*
by *auto*
qed

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 −
have $(\forall\,(d,\ e,\ f){\in}set\ neq.\ \exists\, y'{>}r.\ \forall\, x{\in}\{r{<}..y'\}.\ d * x^2 + e * x + f \neq 0) \Longrightarrow$
$(\exists\, y'{>}r.\ (\forall\,(d,\ e,\ f){\in}set\ neq.\ \forall\, x{\in}\{r{<}..y'\}.\ d * x^2 + e * x + f \neq 0))$
using *neq-qe-inf-helper assms* **by** *auto*
then have $(\exists\, y'{>}r.\ (\forall\,(d,\ e,\ f){\in}set\ neq.\ \forall\, x{\in}\{r{<}..y'\}.\ d * x^2 + e * x + f \neq 0))$
using *assms*
by *blast*
then obtain *y* **where** *y-prop*: $y > r \wedge\ (\forall\,(d,\ e,\ f){\in}set\ neq.\ \forall\, x{\in}\{r{<}..y\}.\ d * x^2 + e * x + f \neq 0)$
by *auto*
have $\exists\, q.\ q >\ r \wedge q < y$ **using** *y-prop dense* **by** *auto*
then obtain *q* **where** *q-prop*: $q > r \wedge q < y$ **by** *auto*
then have $(\forall\,(d,\ e,\ f){\in}set\ neq.\ d{*}q\hat{}2 + e{*}q + f \neq 0)$
using *y-prop* **by** *auto*
then show *?thesis*
by *auto*
qed

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 −
assume *big-asm*: $(((\forall\, (a,\ b,\ c){\in}set\ les.\ \exists\, x.\ \forall\, y{<}x.\ a * y^2 + b * y + c < 0) \vee$
$(\exists\, (a',\ b',\ c'){\in}set\ les.$
$a' = 0 \wedge$
$b' \neq 0 \wedge$
$(\forall\, (d,\ e,\ f){\in}set\ les.$
$\exists\, y'{>}-\ (c'\ /\ b').\ \forall\, x{\in}\{-\ (c'\ /\ b'){<}..y'\}.\ d * x^2 + e * x + f < 0) \vee$
$a' \neq 0 \wedge$
$4 * a' * c' \leq b'^2 \wedge$
$((\forall\, (d,\ e,\ f){\in}set\ les.$
$\exists\, y'{>}(sqrt\ (b'^2 - 4 * a' * c') - b')\ /\ (2 * a').$
$\forall\, x{\in}\{(sqrt\ (b'^2 - 4 * a' * c') - b')\ /\ (2 * a'){<}..y'\}.$
$d * x^2 + e * x + f < 0) \vee$
$(\forall\, (d,\ e,\ f){\in}set\ les.$
$\exists\, y'{>}(-\ b' - sqrt\ (b'^2 - 4 * a' * c'))\ /\ (2 * a').$
$\forall\, x{\in}\{(-\ b' - sqrt\ (b'^2 - 4 * a' * c'))\ /\ (2 * a'){<}..y'\}.$
$d * x^2 + e * x + f < 0)))))$
then have *big-or*: $(\forall\, (a,\ b,\ c){\in}set\ les.\ \exists\, x.\ \forall\, y{<}x.\ a * y^2 + b * y + c < 0) \vee$
$(\exists\, (a',\ b',\ c'){\in}set\ les.$
$a' = 0 \wedge$
$b' \neq 0 \wedge (\forall\, (d,\ e,\ f){\in}set\ les.\ \exists\, y'{>}-\ (c'\ /\ b').\ \forall\, x{\in}\{-\ (c'\ /\ b'){<}..y'\}.$
$d * x^2 + e * x + f < 0))$
$\vee$
$(\exists\, (a',\ b',\ c'){\in}set\ les.$
$a' \neq 0 \wedge$
$4 * a' * c' \leq b'^2 \wedge$
$(\forall\, (d,\ e,\ f){\in}set\ les.$
$\exists\, y'{>}(sqrt\ (b'^2 - 4 * a' * c') - b')\ /\ (2 * a').$
$\forall\, x{\in}\{(sqrt\ (b'^2 - 4 * a' * c') - b')\ /\ (2 * a'){<}..y'\}.\ d * x^2 + e$
$* x + f < 0))$
$\vee$
$(\exists\, (a',\ b',\ c'){\in}set\ les.\ \ a' \neq 0 \wedge$
$4 * a' * c' \leq b'^2 \wedge$
$(\forall\, (d,\ e,\ f){\in}set\ les.$
$\exists\, y'{>}(-\ b' - sqrt\ (b'^2 - 4 * a' * c'))\ /\ (2 * a').$
$\forall\, x{\in}\{(-\ b' - sqrt\ (b'^2 - 4 * a' * c'))\ /\ (2 * a'){<}..y'\}.$
$d * x^2 + e * x + f < 0))$

by *auto*
have *h1-helper*: $(\forall (a,\ b,\ c) \in set\ les.\ \exists x.\ \forall y{<}x.\ a * y^2 + b * y + c < 0) \Longrightarrow (\exists y. \forall x{<}y.\ (\forall (a,\ b,\ c) \in set\ les.\ a * x^2 + b * x + c < 0))$
proof −
show $(\forall (a,\ b,\ c) \in set\ les.\ \exists x.\ \forall y{<}x.\ a * y^2 + b * y + c < 0) \Longrightarrow (\exists y. \forall x{<}y.\ (\forall (a,\ b,\ c) \in set\ les.\ a * x^2 + b * x + c < 0))$
proof (*induct les*)
case *Nil*
then show *?case*
by *auto*
next
case (*Cons q les*)
have *ind*: $\forall a \in set\ (q\ \#\ les).\ case\ a\ of\ (a,\ ba,\ c) \Rightarrow \exists x.\ \forall y{<}x.\ a * y^2 + ba * y + c < 0$
using *Cons.prems*
by *auto*
then have $case\ q\ of\ (a,\ ba,\ c) \Rightarrow \exists x.\ \forall y{<}x.\ a * y^2 + ba * y + c < 0$
by *simp*
then obtain *y2* **where** *y2-prop*: $case\ q\ of\ (a,\ ba,\ c) \Rightarrow\ (\forall y{<}y2.\ a * y^2 + ba * y + c < 0)$
by *auto*
have $\forall a \in set\ les.\ case\ a\ of\ (a,\ ba,\ c) \Rightarrow \exists x.\ \forall y{<}x.\ a * y^2 + ba * y + c < 0$
using *ind* **by** *simp*
then have $\exists y.\ \forall x{<}y.\ \forall a \in set\ les.\ case\ a\ of\ (a,\ ba,\ c) \Rightarrow a * x^2 + ba * x + c < 0$
using *Cons.hyps* **by** *blast*
then obtain *y1* **where** *y1-prop*: $\forall x{<}y1.\ \forall a \in set\ les.\ case\ a\ of\ (a,\ ba,\ c) \Rightarrow a * x\hat{}2 + ba * x + c < 0$
by *blast*
let $?y = min\ y1\ y2$
have $\forall x < ?y.\ (\forall a \in set\ (q\ \#les).\ case\ a\ of\ (a,\ ba,\ c) \Rightarrow a * x\hat{}2 + ba * x + c < 0)$
using *y1-prop y2-prop*
by *fastforce*
then show *?case*
by *blast*
qed
qed
then have *h1*: $(\forall (a,\ b,\ c) \in set\ les.\ \exists x.\ \forall y{<}x.\ a * y^2 + b * y + c < 0) \Longrightarrow (\exists x.\ (\forall (a,\ b,\ c) \in set\ les.\ a * x^2 + b * x + c < 0))$
by (*smt* (*z3*) *infzeros less-eq-real-def not-numeral-le-zero*)

have *h2*: $(\exists (a',\ b',\ c') \in set\ les.\ a' = 0 \wedge b' \neq 0 \wedge (\forall (d,\ e,\ f) \in set\ les.\ \exists y' > -\ (c'\ /\ b').\ \forall x \in \{-\ (c'\ /\ b') < .. y'\}.\ d * x^2 + e * x + f < 0)) \Longrightarrow (\exists x.\ (\forall (a,\ b,\ c) \in set\ les.\ a * x^2 + b * x + c < 0))$
proof −
assume *asm*: $(\exists (a',\ b',\ c') \in set\ les.\ a' = 0 \wedge b' \neq 0 \wedge$

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

then obtain $a'\ b'\ c'$ **where** *abc-prop*: $(a',\ b',\ c') \in set\ les \wedge a' = 0 \wedge b' \neq 0 \wedge$
$(\forall (d,\ e,\ f) \in set\ les.\ \exists\, y' > -\ (c'\ /\ b').\ \forall\, x \in \{-\ (c'\ /\ b') < .. y'\}.\ d * x^2 + e * x + f < 0)$
by *auto*

then show $(\exists\, x.\ (\forall (a,\ b,\ c) \in set\ les.\ a * x^2 + b * x + c < 0))$
using *have-inbetween-point-les* **by** *auto*

qed

have *h3*: $(\exists (a',\ b',\ c') \in set\ les.$
$a' \neq 0 \wedge$
$4 * a' * c' \leq b'^2 \wedge$
$(\forall (d,\ e,\ f) \in set\ les.$
$\exists\, y' > (sqrt\ (b'^2 - 4 * a' * c') - b')\ /\ (2 * a').$
$\forall\, x \in \{(sqrt\ (b'^2 - 4 * a' * c') - b')\ /\ (2 * a') < .. y'\}.\ d * x^2 + e * x + f < 0)) \Longrightarrow ((\exists\, x.\ (\forall (a,\ b,\ c) \in set\ les.\ a * x^2 + b * x + c < 0)))$

proof −

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

then obtain $a'\ b'\ c'$ **where** *abc-prop*: $(a',\ b',\ c') \in set\ les \wedge a' \neq 0 \wedge 4 * a' * c' \leq b'^2 \wedge$
$(\forall (d,\ e,\ f) \in set\ les.$
$\exists\, y' > (sqrt\ (b'^2 - 4 * a' * c') - b')\ /\ (2 * a').$
$\forall\, x \in \{(sqrt\ (b'^2 - 4 * a' * c') - b')\ /\ (2 * a') < .. y'\}.\ d * x^2 + e * x + f < 0)$
by *auto*

then show $(\exists\, x.\ (\forall (a,\ b,\ c) \in set\ les.\ a * x^2 + b * x + c < 0))$
using *have-inbetween-point-les* **by** *auto*

qed

have *h4*: $(\exists (a',\ b',\ c') \in set\ les.\ \ a' \neq 0 \wedge$
$4 * a' * c' \leq b'^2 \wedge$
$(\forall (d,\ e,\ f) \in set\ les.$
$\exists\, y' > (-\ b' - sqrt\ (b'^2 - 4 * a' * c'))\ /\ (2 * a').$
$\forall\, x \in \{(-\ b' - sqrt\ (b'^2 - 4 * a' * c'))\ /\ (2 * a') < .. y'\}.$
$d * x^2 + e * x + f < 0)) \Longrightarrow (\exists\, x.\ (\forall (a,\ b,\ c) \in set\ les.\ a * x^2 + b * x + c < 0))$

proof −

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

then obtain $a'\ b'\ c'$ **where** *abc-prop*: $(a',\ b',\ c') \in set\ les \wedge a' \neq 0 \wedge 4 * a' * c' \leq b'^2 \wedge$

$(\forall (d,\ e,\ f) \in set\ les.$
$\exists y' > (-\ b' - sqrt\ (b'^2 - 4 * a' * c')) / (2 * a').$
$\forall x \in \{(-\ b' - sqrt\ (b'^2 - 4 * a' * c')) / (2 * a') <..y'\}.\ d * x^2 + e * x + f < 0)$
by *auto*
then have $(\exists x.\ (\forall (a,\ b,\ c) \in set\ les.\ a * x^2 + b * x + c < 0))$
using *have-inbetween-point-les* **by** *auto*
then show *?thesis* **using** *asm* **by** *auto*
qed
show *?thesis* **using** *big-or h1 h2 h3 h4*
by *blast*
qed

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

proof −
assume *havex*: $(\exists x.\ (\forall (a,\ b,\ c) \in set\ les.\ a * x^2 + b * x + c < 0))$
then obtain x **where** *x-prop*: $\forall (a,\ b,\ c) \in set\ les.\ a * x^2 + b * x + c < 0$ **by** *auto*
have *h*: $(\neg\ (\forall (a,\ b,\ c) \in set\ les.\ \exists x.\ \forall y < x.\ a * y^2 + b * y + c < 0) \wedge \neg\ (\exists (a',\ b',\ c') \in set\ les.$
$a' = 0 \wedge$
$b' \neq 0 \wedge$
$(\forall (d,\ e,\ f) \in set\ les.\ \exists y' > -\ (c' / b').\ \forall x \in \{-\ (c' / b') <..y'\}.\ d * x^2 + e * x + f < 0)) \wedge$
$\neg\ (\exists (a',\ b',\ c') \in set\ les.$
$a' \neq 0 \wedge$
$4 * a' * c' \leq b'^2 \wedge$
$(\forall (d,\ e,\ f) \in set\ les.$
$\exists y' > (sqrt\ (b'^2 - 4 * a' * c') - b') / (2 * a').$
$\forall x \in \{(sqrt\ (b'^2 - 4 * a' * c') - b') / (2 * a') <..y'\}.\ d * x^2 + e * x$

$+ f < 0)) \wedge$
$\neg (\exists (a', b', c') \in set\ les.$
$a' \neq 0 \wedge$
$4 * a' * c' \leq b'^2 \wedge$
$(\forall (d, e, f) \in set\ les.$
$\exists y' > (- b' - sqrt\ (b'^2 - 4 * a' * c')) / (2 * a').$
$\forall x \in \{(- b' - sqrt\ (b'^2 - 4 * a' * c')) / (2 * a') <..y'\}.\ d * x^2 + e$
$* x + f < 0))) \Longrightarrow False$
proof −
assume *big*: $(\neg (\forall (a, b, c) \in set\ les.\ \exists x.\ \forall y < x.\ a * y^2 + b * y + c < 0) \wedge \neg$
$(\exists (a', b', c') \in set\ les.$
$a' = 0 \wedge$
$b' \neq 0 \wedge$
$(\forall (d, e, f) \in set\ les.\ \exists y' > - (c' / b').\ \forall x \in \{- (c' / b') <..y'\}.\ d * x^2 + e *$
$x + f < 0)) \wedge$
$\neg (\exists (a', b', c') \in set\ les.$
$a' \neq 0 \wedge$
$4 * a' * c' \leq b'^2 \wedge$
$(\forall (d, e, f) \in set\ les.$
$\exists y' > (sqrt\ (b'^2 - 4 * a' * c') - b') / (2 * a').$
$\forall x \in \{(sqrt\ (b'^2 - 4 * a' * c') - b') / (2 * a') <..y'\}.\ d * x^2 + e * x$
$+ f < 0)) \wedge$
$\neg (\exists (a', b', c') \in set\ les.$
$a' \neq 0 \wedge$
$4 * a' * c' \leq b'^2 \wedge$
$(\forall (d, e, f) \in set\ les.$
$\exists y' > (- b' - sqrt\ (b'^2 - 4 * a' * c')) / (2 * a').$
$\forall x \in \{(- b' - sqrt\ (b'^2 - 4 * a' * c')) / (2 * a') <..y'\}.\ d * x^2 + e$
$* x + f < 0)))$
have *notneginf*: $\neg (\forall (a, b, c) \in set\ les.\ \exists x.\ \forall y < x.\ a * y^2 + b * y + c < 0)$
using *big* **by** *auto*
have *notlinroot*: $\neg (\exists (a', b', c') \in set\ les.$
$a' = 0 \wedge$
$b' \neq 0 \wedge$
$(\forall (d, e, f) \in set\ les.\ \exists y' > - (c' / b').\ \forall x \in \{- (c' / b') <..y'\}.\ d * x^2 + e *$
$x + f < 0))$
using *big* **by** *auto*
have *notquadroot1*: $\neg (\exists (a', b', c') \in set\ les.$
$a' \neq 0 \wedge$
$4 * a' * c' \leq b'^2 \wedge$
$(\forall (d, e, f) \in set\ les.$
$\exists y' > (sqrt\ (b'^2 - 4 * a' * c') - b') / (2 * a').$
$\forall x \in \{(sqrt\ (b'^2 - 4 * a' * c') - b') / (2 * a') <..y'\}.\ d * x^2 + e * x$
$+ f < 0))$
using *big* **by** *auto*
have *notquadroot2*: $\neg (\exists (a', b', c') \in set\ les.$
$a' \neq 0 \wedge$
$4 * a' * c' \leq b'^2 \wedge$
$(\forall (d, e, f) \in set\ les.$

```
          ∃ y'>(− b' − sqrt (b'^2 − 4 * a' * c')) / (2 * a').
            ∀ x∈{(− b' − sqrt (b'^2 − 4 * a' * c')) / (2 * a')<..y'}. d * x^2 + e
* x + f < 0))
    using big by auto
  have nok: ¬ (∃ k. ∃ (a, b, c)∈set les. a*k^2 + b*k + c = 0 ∧
          (∀ (d, e, f)∈set les. ∃ y'>k. ∀ x∈{k<..y'}. d * x^2 + e * x + f < 0))
  proof −
    have (∃ k. ∃ (a, b, c)∈set les. a*k^2 + b*k + c = 0 ∧
          (∀ (d, e, f)∈set les. ∃ y'>k. ∀ x∈{k<..y'}. d * x^2 + e * x + f < 0))
⟹ False
    proof −
      assume (∃ k. ∃ (a, b, c)∈set les. a*k^2 + b*k + c = 0 ∧
          (∀ (d, e, f)∈set les. ∃ y'>k. ∀ x∈{k<..y'}. d * x^2 + e * x + f < 0))
      then obtain k a b c where k-prop: (a, b, c) ∈ set les ∧ a*k^2 + b*k + c
= 0 ∧
          (∀ (d, e, f)∈set les. ∃ y'>k. ∀ x∈{k<..y'}. d * x^2 + e * x + f < 0)
        by auto
      have azer: a = 0 ⟹ False
      proof −
        assume azer: a = 0
        then have b = 0 ⟹ c = 0 using k-prop by auto
        then have bnonz: b≠ 0
          using azer x-prop k-prop
          by auto
        then have k = −c/b using k-prop azer
          by (smt (verit, best) mult-eq-0-iff nonzero-mult-div-cancel-left)
        then have (∃ (a', b', c')∈set les.
          a' = 0 ∧ b' ≠ 0 ∧ (∀ (d, e, f)∈set les. ∃ y'>− (c' / b'). ∀ x∈{− (c' /
b')<..y'}. d * x^2 + e * x + f < 0))
          using k-prop azer bnonz by auto
        then show False using notlinroot
          by auto
      qed
      have anonz: a ≠ 0 ⟹ False
      proof −
        assume anonz: a ≠ 0
        let ?r1 = (− b − sqrt (b^2 − 4 * a * c)) / (2 * a)
        let ?r2 = (− b + sqrt (b^2 − 4 * a * c)) / (2 * a)
        have discr: 4 * a * c ≤ b^2 using anonz k-prop discriminant-negative[of
a b c]
          unfolding discrim-def
          by fastforce
        then have k = ?r1 ∨ k = ?r2 using k-prop discriminant-nonneg[of a b
c] unfolding discrim-def
          using anonz
          by auto
        then have (∃ (a', b', c')∈set les.
         a' ≠ 0 ∧
         4 * a' * c' ≤ b'^2 ∧
```

```
        (∀(d, e, f)∈set les.
           ∃y'>(sqrt (b'^2 - 4 * a' * c') - b') / (2 * a').
             ∀x∈{(sqrt (b'^2 - 4 * a' * c') - b') / (2 * a')<..y'}. d * x^2 + e * x
+ f < 0)) ∨
        (∃(a', b', c')∈set les.
        a' ≠ 0 ∧
        4 * a' * c' ≤ b'^2 ∧
        (∀(d, e, f)∈set les.
           ∃y'>(- b' - sqrt (b'^2 - 4 * a' * c')) / (2 * a').
             ∀x∈{(- b' - sqrt (b'^2 - 4 * a' * c')) / (2 * a')<..y'}. d * x^2 + e
* x + f < 0))
         using discr anonz notquadroot1 notquadroot2 k-prop
         by auto
       then show False using  notquadroot1 notquadroot2
         by auto
     qed
     show False
       using azer anonz  by auto
   qed
   then show ?thesis by auto
 qed
 have finset: finite (set les)
   by blast
 have h1: (∃(a, b, c)∈set les. a = 0 ∧ b = 0 ∧ c = 0)  ⟹ False
   using x-prop by fastforce
 then have h2: ¬(∃(a, b, c)∈set les. a = 0 ∧ b = 0 ∧ c = 0) ⟹ False
 proof −
   assume nozer: ¬(∃(a, b, c)∈set les. a = 0 ∧ b = 0 ∧ c = 0)
   then have same-set: root-set (set les) = set (sorted-root-list-set (set les))
     using root-set-finite finset set-sorted-list-of-set
     by (simp add: nozer root-set-finite sorted-root-list-set-def)
   have xnotin: x ∉ root-set (set les)
     unfolding root-set-def using x-prop by auto
   let ?srl = sorted-root-list-set (set les)
   have notinlist: ¬ List.member ?srl x
     using xnotin same-set
     by (simp add: in-set-member)
   then have notmem: ∀n < (length ?srl). x ≠ nth-default 0 ?srl n
     using nth-mem same-set xnotin nth-default-def
     by metis
   show ?thesis
   proof (induct ?srl)
     case Nil
     then have (∀(a, b, c)∈set les. ∃x. ∀y<x. a * y^2 + b * y + c < 0)
     proof clarsimp
       fix a b c
       assume noroots: [] = sorted-root-list-set (set les)
       assume inset: (a, b, c) ∈ set les
       have {} = root-set (set les)
```

```
      using noroots same-set
      by auto
    then have nozero: ¬(∃ x. a*x^2 + b*x + c = 0)
      using inset unfolding root-set-def by auto
    have ∀ y<x. a * y^2 + b * y + c < 0
    proof clarsimp
      fix y
      assume y < x
     then have sign-num (a*x^2 + b*x + c) = sign-num (a*y^2 + b*y + c)
       using nozero  by (metis changes-sign-var)
      then show a * y^2 + b * y + c < 0
       unfolding sign-num-def using x-prop inset
       by (smt split-conv)
    qed
    then show ∃ x. ∀ y<x. a * y^2 + b * y + c < 0
      by auto
  qed
  then show ?case using notneginf by auto
next
  case (Cons w xa)



  then have lengthsrl: length ?srl > 0 by auto
  have neginf: x < nth-default 0 ?srl 0 ⟹ False
  proof −
    assume xlt: x < nth-default 0 ?srl 0
    have all: (∀ (a, b, c)∈set les. ∀ y<x. a * y^2 + b * y + c < 0)
    proof clarsimp
      fix a b c y
      assume inset: (a, b, c) ∈ set les
      assume y < x
      have xl: a*x^2 + b*x + c < 0 using x-prop inset by auto
      have ¬(∃ q. q < nth-default 0 ?srl 0  ∧ a*q^2 + b*q + c = 0)
      proof −
       have (∃ q. q < nth-default 0 ?srl 0  ∧ a*q^2 + b*q + c = 0) ⟹ False
       proof − assume ∃ q. q < nth-default 0 ?srl 0  ∧ a*q^2 + b*q + c = 0
         then obtain q where q-prop: q < nth-default 0 ?srl 0 ∧a*q^2 + b*q
+ c = 0 by auto
           then have  q ∈ root-set (set les) unfolding root-set-def using inset
by auto
           then have List.member ?srl q using same-set
             by (simp add: in-set-member)
           then have q ≥ nth-default 0 ?srl 0
             using sorted-sorted-list-of-set[where A = root-set (set les)]
             unfolding sorted-root-list-set-def
          by (metis ‹q ∈ root-set (set les)› in-set-conv-nth le-less-linear lengthsrl
not-less0 nth-default-nth same-set sorted-nth-mono sorted-root-list-set-def)
           then show False using q-prop by auto
```

```
          qed
          then show ?thesis by auto
        qed
        then have ¬(∃ q. q < x ∧ a*q^2 + b*q + c = 0) using xlt by auto
        then show  a * y^2 + b * y + c < 0
          using xl changes-sign-var[where a = a, where b = b, where c = c,
where x = y, where y = x]
        unfolding sign-num-def using ‹y < x› less-eq-real-def zero-neq-numeral

          by fastforce
      qed
      have (∀(a, b, c)∈set les. ∃ x. ∀ y<x. a * y^2 + b * y + c < 0)
      proof clarsimp
        fix a b c
        assume (a, b, c)∈set les
        then show ∃ x. ∀ y<x. a * y^2 + b * y + c < 0
          using all by blast
      qed
      then show False using notneginf by auto
    qed
    have x > nth-default 0 ?srl (length ?srl − 1) ⟹ (∃ k. ∃(a, b, c)∈set les.
a*k^2 + b*k + c = 0 ∧
          (∀(d, e, f)∈set les. ∃ y'>k. ∀ x∈{k<..y'}. d * x^2 + e * x + f < 0))
    proof −
      assume xgt: x > nth-default 0 ?srl (length ?srl − 1)
      let ?lg = nth-default 0 ?srl (length ?srl − 1)
      have List.member ?srl ?lg
          by (metis diff-less in-set-member lengthsrl nth-default-def nth-mem
zero-less-one)
      then have ?lg ∈ root-set (set les)
        using same-set in-set-member[of ?lg ?srl] by auto
      then have exabc: ∃(a, b, c)∈set les. a*?lg^2 + b*?lg + c = 0
        unfolding root-set-def by auto
      have (∀(d, e, f)∈set les. ∀ q∈{?lg<..x}. d * q^2 + e * q + f < 0)
      proof clarsimp
        fix d e f q
        assume inset: (d, e, f) ∈ set les
           assume qgt: (nth-default 0) (sorted-root-list-set (set les)) (length
(sorted-root-list-set (set les)) − Suc 0) < q
        assume qlt: q ≤ x
        have nor: ¬(∃ r. d * r^2 + e * r + f = 0 ∧ r > ?lg)
        proof −
          have (∃ r. d * r^2 + e * r + f = 0 ∧ r > ?lg) ⟹ False
          proof −
            assume ∃ r. d * r^2 + e * r + f = 0 ∧ r > ?lg
             then obtain r where r-prop: d*r^2 + e*r + f = 0 ∧ r > ?lg by
auto
             then have r ∈ root-set (set les) using inset unfolding root-set-def
by auto
```

then have *List.member ?srl r*
using *same-set in-set-member*
by (*simp add*: *in-set-member*)
then have $r \leq ?lg$ **using** *sorted-sorted-list-of-set nth-default-def*
by (*metis One-nat-def Suc-pred ‹r ∈ root-set (set les)› in-set-conv-nth lengthsrl lessI less-Suc-eq-le same-set sorted-nth-mono sorted-root-list-set-def*)
then show *False* **using** *r-prop* **by** *auto*
qed
then show *?thesis* **by** *auto*
qed
then have *xltz-helper*: $\neg(\exists r.\ r \geq q \wedge d * r\hat{}2 + e * r + f = 0)$
using *qgt* **by** *auto*
then have *xltz*: $d*x\hat{}2 + e*x + f < 0$ **using** *inset x-prop* **by** *auto*
show $d * q^2 + e * q + f < 0$
using *qlt qgt nor changes-sign-var*[*of d - e f -*] *xltz xltz-helper* **unfolding** *sign-num-def*
apply (*auto*)
by *smt*
qed
then have $(\forall (d,\ e,\ f) \in set\ les.\ \exists y' > ?lg.\ \forall x \in \{?lg<..y'\}.\ d * x^2 + e * x + f < 0)$
using *xgt* **by** *auto*
then have $(\exists (a,\ b,\ c) \in set\ les.\ a*?lg\hat{}2 + b*?lg + c = 0 \wedge (\forall (d,\ e,\ f) \in set\ les.\ \exists y' > ?lg.\ \forall x \in \{?lg<..y'\}.\ d * x^2 + e * x + f < 0))$
using *exabc* **by** *auto*
then show $(\exists k.\ \exists (a,\ b,\ c) \in set\ les.\ a*k\hat{}2 + b*k + c = 0 \wedge (\forall (d,\ e,\ f) \in set\ les.\ \exists y' > k.\ \forall x \in \{k<..y'\}.\ d * x^2 + e * x + f < 0))$
by *auto*
qed
then have *posinf*: $x > nth\text{-}default\ 0\ ?srl\ (length\ ?srl - 1) \Longrightarrow False$
using *nok* **by** *auto*
have $(\exists n.\ (n+1) < (length\ ?srl) \wedge x > (nth\text{-}default\ 0\ ?srl)\ n \wedge x < (nth\text{-}default\ 0\ ?srl\ (n+1))) \Longrightarrow (\exists k.\ \exists (a,\ b,\ c) \in set\ les.\ a*k\hat{}2 + b*k + c = 0 \wedge (\forall (d,\ e,\ f) \in set\ les.\ \exists y' > k.\ \forall x \in \{k<..y'\}.\ d * x^2 + e * x + f < 0))$
proof −
assume $\exists n.\ (n+1) < (length\ ?srl) \wedge x > nth\text{-}default\ 0\ ?srl\ n \wedge x < nth\text{-}default\ 0\ ?srl\ (n+1)$
then obtain *n* **where** *n-prop*: $(n+1) < (length\ ?srl) \wedge x > nth\text{-}default\ 0\ ?srl\ n \wedge x < nth\text{-}default\ 0\ ?srl\ (n+1)$ **by** *auto*
let *?elt* = *nth-default 0 ?srl n*
let *?elt2* = *nth-default 0 ?srl* $(n+1)$
have *List.member ?srl ?elt*
using *n-prop nth-default-def*
by (*metis add-lessD1 in-set-member nth-mem*)
then have *?elt* ∈ *root-set* (*set les*)
using *same-set in-set-member*[*of ?elt ?srl*] **by** *auto*
then have *exabc*: $\exists (a,\ b,\ c) \in set\ les.\ a*?elt\hat{}2 + b*?elt + c = 0$
unfolding *root-set-def* **by** *auto*
then obtain *a b c* **where** $(a,\ b,\ c) \in set\ les \wedge a*?elt\hat{}2 + b*?elt + c = 0$

```
    by auto
  have xltel2: x < ?elt2 using n-prop by auto
  have xgtel: x > ?elt  using n-prop by auto
  have (∀(d, e, f)∈set les. ∀q∈{?elt<..x}. d * q^2 + e * q + f < 0)
  proof clarsimp
    fix d e f q
    assume inset: (d, e, f) ∈ set les
    assume qgt: nth-default 0 (sorted-root-list-set (set les)) n < q
    assume qlt: q ≤ x

    have nor: ¬(∃r. d * r^2 + e * r + f = 0 ∧ r > ?elt ∧r < ?elt2)
    proof −
     have (∃r. d * r^2 + e * r + f = 0 ∧ r > ?elt ∧ r < ?elt2) ⟹ False
      proof −
        assume ∃r. d * r^2 + e * r + f = 0 ∧ r > ?elt  ∧ r < ?elt2
       then obtain r where r-prop: d*r^2 + e*r + f = 0 ∧ r > ?elt  ∧ r
< ?elt2 by auto
        then have r ∈ root-set (set les) using inset unfolding root-set-def
by auto
        then have List.member ?srl r
          using same-set in-set-member
          by (simp add: in-set-member)
        then have ∃i < (length ?srl). r = nth-default 0 ?srl i
              by (metis ‹r ∈ root-set (set les)› in-set-conv-nth same-set
nth-default-def)
        then obtain i where i-prop: i < (length ?srl) ∧ r = nth-default 0
?srl i
          by auto
        have r > ?elt using r-prop by auto
        then  have igt:  i > n using i-prop sorted-sorted-list-of-set
            by (smt add-lessD1 leI n-prop nth-default-def sorted-nth-mono
sorted-root-list-set-def)
        have r < ?elt2 using r-prop by auto
        then have ilt:  i < n + 1 using i-prop sorted-sorted-list-of-set
      by (smt leI n-prop nth-default-def sorted-nth-mono sorted-root-list-set-def)

        then show False  using igt ilt
          by auto
      qed
      then show ?thesis by auto
    qed
    then have nor: ¬(∃r. d * r^2 + e * r + f = 0 ∧ r > ?elt ∧r ≤ x)
      using xltel2 xgtel by auto
    then have xltz: d*x^2 + e*x + f < 0 using inset x-prop by auto
    show d * q^2 + e * q + f < 0
   using qlt qgt nor changes-sign-var[of d - e f -] xltz unfolding sign-num-def
      by smt
  qed
  then have  (∀(d, e, f)∈set les. ∃y'>?elt. ∀x∈{?elt<..y'}. d * x^2 + e * x
```

$+ f < 0$)
using *xgtel xltel2* **by** *auto*
then have ($\exists$(a, b, c)$\in$*set les*. a*?elt^2 + b*?elt + c = 0 $\wedge$
($\forall$(d, e, f)$\in$*set les*. $\exists$ y'>?elt. $\forall$ $x\in${?elt<..y'}. $d * x^2 + e * x + f <$
0))
using *exabc* **by** *auto*
then show ($\exists$ k. $\exists$(a, b, c)$\in$*set les*. a*k^2 + b*k + c = 0 $\wedge$
($\forall$(d, e, f)$\in$*set les*. $\exists$ y'>k. $\forall$ $x\in${k<..y'}. $d * x^2 + e * x + f < 0$))
by *auto*
qed
then have *inbetw*: ($\exists$ n. (n+1) < (*length ?srl*) $\wedge$ x > *nth-default 0 ?srl* n
$\wedge$ x < *nth-default 0 ?srl* (n + 1)) $\Longrightarrow$ *False*
using *nok* **by** *auto*
have *lenzer*: *length xa* = 0 $\Longrightarrow$ *False*
proof −
assume *length xa* = 0
have *xis*: x > w $\vee$ x < w
using *notmem Cons.hyps*
by (*smt list.set-intros*(1) *same-set xnotin*)
have *xgt*: x > w $\Longrightarrow$ *False*
proof −
assume *xgt*: x > w
show *False* **using** *posinf Cons.hyps*
by (*metis One-nat-def Suc-eq-plus1* ‹*length xa* = 0› *cancel-comm-monoid-add-class.diff-cancel*
list.size(4) *nth-default-Cons-0 xgt*)
qed
have *xlt*: x < w $\Longrightarrow$ *False*
proof −
assume *xlt*: x < w
show *False* **using** *neginf Cons.hyps*
by (*metis nth-default-Cons-0 xlt*)
qed
show *False* **using** *xis xgt xlt* **by** *auto*
qed
have *lengt*: *length xa* > 0 $\Longrightarrow$ *False*
proof −
assume *length xa* > 0
have x $\geq$ *nth-default 0 ?srl 0* **using** *neginf*
by *fastforce*
then have *xgtf*: x > *nth-default 0 ?srl 0* **using** *notmem*
using *Cons.hyps*(2) **by** *fastforce*
have x $\leq$ *nth-default 0 ?srl* (*length ?srl* − 1) **using** *posinf* **by** *fastforce*
then have ($\exists$ n. (n+1) < (*length ?srl*) $\wedge$ x $\geq$ *nth-default 0 ?srl* n $\wedge$ x $\leq$
nth-default 0 ?srl (n + 1))
using *lengthsrl xgtf notmem sorted-list-prop*[**where** l = *?srl*, **where** x =
x]
by (*metis add-lessD1 diff-less nth-default-nth sorted-root-list-set-def*
sorted-sorted-list-of-set zero-less-one)
then obtain n **where** *n-prop*: (n+1) < (*length ?srl*) $\wedge$ x $\geq$ *nth-default 0*

?srl n ∧ *x* ≤ *nth-default 0 ?srl* (*n* + *1*) **by** *auto*
then have *x* > *nth-default 0 ?srl n* ∧ *x* < *nth-default 0 ?srl* (*n*+*1*)
using *notmem*
by (*metis Suc-eq-plus1 Suc-lessD less-eq-real-def*)
then have (∃ *n*. (*n*+*1*) < (*length ?srl*) ∧ *x* > *nth-default 0 ?srl n* ∧ *x* < *nth-default 0 ?srl* (*n* + *1*))
using *n-prop*
by *blast*
then show *False* **using** *inbetw* **by** *auto*
qed
then show *?case* **using** *lenzer lengt* **by** *auto*
qed
qed
show *False*
using *h1 h2* **by** *auto*
qed
then have *equiv-false*: ¬((∀ (*a*, *b*, *c*)∈*set les*. ∃ *x*. ∀ *y*<*x*. *a* ∗ *y*^{2} + *b* ∗ *y* + *c* < *0*) ∨
(∃ (*a'*, *b'*, *c'*)∈*set les*.
a' = *0* ∧
b' ≠ *0* ∧ (∀ (*d*, *e*, *f*)∈*set les*. ∃ *y'*>− (*c'* / *b'*). ∀ *x*∈{− (*c'* / *b'*)<..*y'*}. *d* ∗ *x*^{2} + *e* ∗ *x* + *f* < *0*))
∨
(∃ (*a'*, *b'*, *c'*)∈*set les*.
a' ≠ *0* ∧
4 ∗ *a'* ∗ *c'* ≤ *b'*^{2} ∧
(∀ (*d*, *e*, *f*)∈*set les*.
∃ *y'*>(*sqrt* (*b'*^{2} − *4* ∗ *a'* ∗ *c'*) − *b'*) / (*2* ∗ *a'*).
∀ *x*∈{(*sqrt* (*b'*^{2} − *4* ∗ *a'* ∗ *c'*) − *b'*) / (*2* ∗ *a'*)<..*y'*}. *d* ∗ *x*^{2} + *e* ∗ *x* + *f* < *0*))
∨
(∃ (*a'*, *b'*, *c'*)∈*set les*. *a'* ≠ *0* ∧
4 ∗ *a'* ∗ *c'* ≤ *b'*^{2} ∧
(∀ (*d*, *e*, *f*)∈*set les*.
∃ *y'*>(− *b'* − *sqrt* (*b'*^{2} − *4* ∗ *a'* ∗ *c'*)) / (*2* ∗ *a'*).
∀ *x*∈{(− *b'* − *sqrt* (*b'*^{2} − *4* ∗ *a'* ∗ *c'*)) / (*2* ∗ *a'*)<..*y'*}.
d ∗ *x*^{2} + *e* ∗ *x* + *f* < *0*))) ⟹ *False*
by *linarith*
have ¬((∀ (*a*, *b*, *c*)∈*set les*. ∃ *x*. ∀ *y*<*x*. *a* ∗ *y*^{2} + *b* ∗ *y* + *c* < *0*) ∨
(∃ (*a'*, *b'*, *c'*)∈*set les*.
a' = *0* ∧
b' ≠ *0* ∧
(∀ (*d*, *e*, *f*)∈*set les*.
∃ *y'*>− (*c'* / *b'*). ∀ *x*∈{− (*c'* / *b'*)<..*y'*}. *d* ∗ *x*^{2} + *e* ∗ *x* + *f* < *0*) ∨
a' ≠ *0* ∧
4 ∗ *a'* ∗ *c'* ≤ *b'*^{2} ∧
((∀ (*d*, *e*, *f*)∈*set les*.
∃ *y'*>(*sqrt* (*b'*^{2} − *4* ∗ *a'* ∗ *c'*) − *b'*) / (*2* ∗ *a'*).
∀ *x*∈{(*sqrt* (*b'*^{2} − *4* ∗ *a'* ∗ *c'*) − *b'*) / (*2* ∗ *a'*)<..*y'*}.

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

proof −
assume $\neg((\forall (a,\ b,\ c) \in set\ les.\ \exists x.\ \forall y < x.\ a * y^2 + b * y + c < 0) \vee$
$(\exists (a',\ b',\ c') \in set\ les.$
$a' = 0 \wedge$
$b' \neq 0 \wedge$
$(\forall (d,\ e,\ f) \in set\ les.$
$\exists y' > -\ (c' / b').\ \forall x \in \{-\ (c' / b') < .. y'\}.\ d * x^2 + e * x + f < 0) \vee$
$a' \neq 0 \wedge$
$4 * a' * c' \leq b'^2 \wedge$
$((\forall (d,\ e,\ f) \in set\ les.$
$\exists y' > (sqrt\ (b'^2 - 4 * a' * c') - b') / (2 * a').$
$\forall x \in \{(sqrt\ (b'^2 - 4 * a' * c') - b') / (2 * a') < .. y'\}.$
$d * x^2 + e * x + f < 0) \vee$
$(\forall (d,\ e,\ f) \in set\ les.$
$\exists y' > (-\ b' - sqrt\ (b'^2 - 4 * a' * c')) / (2 * a').$
$\forall x \in \{(-\ b' - sqrt\ (b'^2 - 4 * a' * c')) / (2 * a') < .. y'\}.$
$d * x^2 + e * x + f < 0))))$
then have $\neg((\forall (a,\ b,\ c) \in set\ les.\ \exists x.\ \forall y < x.\ a * y^2 + b * y + c < 0) \vee$
$(\exists (a',\ b',\ c') \in set\ les.$
$a' = 0 \wedge$
$b' \neq 0 \wedge (\forall (d,\ e,\ f) \in set\ les.\ \exists y' > -\ (c' / b').\ \forall x \in \{-\ (c' / b') < .. y'\}.$
$d * x^2 + e * x + f < 0))$
$\vee$
$(\exists (a',\ b',\ c') \in set\ les.$
$a' \neq 0 \wedge$
$4 * a' * c' \leq b'^2 \wedge$
$(\forall (d,\ e,\ f) \in set\ les.$
$\exists y' > (sqrt\ (b'^2 - 4 * a' * c') - b') / (2 * a').$
$\forall x \in \{(sqrt\ (b'^2 - 4 * a' * c') - b') / (2 * a') < .. y'\}.\ d * x^2 + e$
$* x + f < 0))$
$\vee$
$(\exists (a',\ b',\ c') \in set\ les.\ \ a' \neq 0 \wedge$
$4 * a' * c' \leq b'^2 \wedge$
$(\forall (d,\ e,\ f) \in set\ les.$
$\exists y' > (-\ b' - sqrt\ (b'^2 - 4 * a' * c')) / (2 * a').$
$\forall x \in \{(-\ b' - sqrt\ (b'^2 - 4 * a' * c')) / (2 * a') < .. y'\}.$
$d * x^2 + e * x + f < 0)))$
by *auto*
then show *?thesis*
using *equiv-false* **by** *auto*
qed
then show *?thesis*
by *blast*
qed

lemma *les-qe* :
shows $(\exists x.\ (\forall (a,\ b,\ c){\in}\mathit{set\ les}.\ a * x^2 + b * x + c < 0)) =$
$((\forall (a,\ b,\ c){\in}\mathit{set\ les}.\ \exists x.\ \forall y{<}x.\ a * y^2 + b * y + c < 0) \vee$
$(\exists (a',\ b',\ c'){\in}\mathit{set\ les}.$
$a' = 0 \wedge$
$b' \neq 0 \wedge$
$(\forall (d,\ e,\ f){\in}\mathit{set\ les}.$
$\exists y'{>}-\ (c'\ /\ b').\ \forall x{\in}\{-\ (c'\ /\ b'){<}..y'\}.\ d * x^2 + e * x + f < 0) \vee$
$a' \neq 0 \wedge$
$4 * a' * c' \leq b'^2 \wedge$
$((\forall (d,\ e,\ f){\in}\mathit{set\ les}.$
$\exists y'{>}(\mathit{sqrt}\ (b'^2 - 4 * a' * c') - b')\ /\ (2 * a').$
$\forall x{\in}\{(\mathit{sqrt}\ (b'^2 - 4 * a' * c') - b')\ /\ (2 * a'){<}..y'\}.$
$d * x^2 + e * x + f < 0) \vee$
$(\forall (d,\ e,\ f){\in}\mathit{set\ les}.$
$\exists y'{>}(-\ b' - \mathit{sqrt}\ (b'^2 - 4 * a' * c'))\ /\ (2 * a').$
$\forall x{\in}\{(-\ b' - \mathit{sqrt}\ (b'^2 - 4 * a' * c'))\ /\ (2 * a'){<}..y'\}.$
$d * x^2 + e * x + f < 0))))$
proof −
have *first*: $(\exists x.\ (\forall (a,\ b,\ c){\in}\mathit{set\ les}.\ a * x^2 + b * x + c < 0)) \longrightarrow$
$((\forall (a,\ b,\ c){\in}\mathit{set\ les}.\ \exists x.\ \forall y{<}x.\ a * y^2 + b * y + c < 0) \vee$
$(\exists (a',\ b',\ c'){\in}\mathit{set\ les}.$
$a' = 0 \wedge$
$b' \neq 0 \wedge$
$(\forall (d,\ e,\ f){\in}\mathit{set\ les}.$
$\exists y'{>}-\ (c'\ /\ b').\ \forall x{\in}\{-\ (c'\ /\ b'){<}..y'\}.\ d * x^2 + e * x + f < 0) \vee$
$a' \neq 0 \wedge$
$4 * a' * c' \leq b'^2 \wedge$
$((\forall (d,\ e,\ f){\in}\mathit{set\ les}.$
$\exists y'{>}(\mathit{sqrt}\ (b'^2 - 4 * a' * c') - b')\ /\ (2 * a').$
$\forall x{\in}\{(\mathit{sqrt}\ (b'^2 - 4 * a' * c') - b')\ /\ (2 * a'){<}..y'\}.$
$d * x^2 + e * x + f < 0) \vee$
$(\forall (d,\ e,\ f){\in}\mathit{set\ les}.$
$\exists y'{>}(-\ b' - \mathit{sqrt}\ (b'^2 - 4 * a' * c'))\ /\ (2 * a').$
$\forall x{\in}\{(-\ b' - \mathit{sqrt}\ (b'^2 - 4 * a' * c'))\ /\ (2 * a'){<}..y'\}.$
$d * x^2 + e * x + f < 0))))$
using *les-qe-backward* **by** *auto*
have *second*: $((\forall (a,\ b,\ c){\in}\mathit{set\ les}.\ \exists x.\ \forall y{<}x.\ a * y^2 + b * y + c < 0) \vee$
$(\exists (a',\ b',\ c'){\in}\mathit{set\ les}.$
$a' = 0 \wedge$
$b' \neq 0 \wedge$
$(\forall (d,\ e,\ f){\in}\mathit{set\ les}.$
$\exists y'{>}-\ (c'\ /\ b').\ \forall x{\in}\{-\ (c'\ /\ b'){<}..y'\}.\ d * x^2 + e * x + f < 0) \vee$
$a' \neq 0 \wedge$
$4 * a' * c' \leq b'^2 \wedge$
$((\forall (d,\ e,\ f){\in}\mathit{set\ les}.$
$\exists y'{>}(\mathit{sqrt}\ (b'^2 - 4 * a' * c') - b')\ /\ (2 * a').$
$\forall x{\in}\{(\mathit{sqrt}\ (b'^2 - 4 * a' * c') - b')\ /\ (2 * a'){<}..y'\}.$

$d * x^2 + e * x + f < 0) \vee$
$(\forall (d,\ e,\ f) \in set\ les.$
$\exists y' > (-\ b' - sqrt\ (b'^2 - 4 * a' * c')) / (2 * a').$
$\forall x \in \{(-\ b' - sqrt\ (b'^2 - 4 * a' * c')) / (2 * a') < .. y'\}.$
$d * x^2 + e * x + f < 0)))) \longrightarrow (\exists x.\ (\forall (a,\ b,\ c) \in set\ les.\ a * x^2 +$
$b * x + c < 0))$
using *les-qe-forward* **by** *auto*
have $(\exists x.\ (\forall (a,\ b,\ c) \in set\ les.\ a * x^2 + b * x + c < 0)) \longleftrightarrow$
$((\forall (a,\ b,\ c) \in set\ les.\ \exists x.\ \forall y < x.\ a * y^2 + b * y + c < 0) \vee$
$(\exists (a',\ b',\ c') \in set\ les.$
$a' = 0 \wedge$
$b' \neq 0 \wedge$
$(\forall (d,\ e,\ f) \in set\ les.$
$\exists y' > -\ (c' / b').\ \forall x \in \{-\ (c' / b') < .. y'\}.\ d * x^2 + e * x + f < 0) \vee$
$a' \neq 0 \wedge$
$4 * a' * c' \leq b'^2 \wedge$
$((\forall (d,\ e,\ f) \in set\ les.$
$\exists y' > (sqrt\ (b'^2 - 4 * a' * c') - b') / (2 * a').$
$\forall x \in \{(sqrt\ (b'^2 - 4 * a' * c') - b') / (2 * a') < .. y'\}.$
$d * x^2 + e * x + f < 0) \vee$
$(\forall (d,\ e,\ f) \in set\ les.$
$\exists y' > (-\ b' - sqrt\ (b'^2 - 4 * a' * c')) / (2 * a').$
$\forall x \in \{(-\ b' - sqrt\ (b'^2 - 4 * a' * c')) / (2 * a') < .. y'\}.$
$d * x^2 + e * x + f < 0))))$
using *first second*
by *meson*
then show *?thesis*
by *blast*
qed

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
$< 0))) \vee$
$(\forall (d, e, f) \in set\ eq.\ d = 0 \wedge e = 0 \wedge f = 0) \wedge$
$(\exists x.\ \forall (a, b, c) \in set\ les.\ a * x^2 + b * x + c < 0))$

proof $-$
let $?t =$ $((\exists (a', b', c') \in set\ eq.$
$(a' = 0 \wedge b' \neq 0) \wedge$
$(\forall (d, e, f) \in set\ eq.\ d * (- c' / b')^2 + e * (- c' / b') + f = 0) \wedge$
$(\forall (d, e, f) \in set\ les.\ d * (- c' / b')^2 + e * (- c' / b') + f < 0) \vee$
$a' \neq 0 \wedge$

$- b'^2 + 4 * a' * c' \leq 0 \wedge$
$((\forall (d, e, f) \in set\ eq.$
$d * ((- b' + 1 * sqrt\ (b'^2 - 4 * a' * c')) / (2 * a'))^2 +$
$e * ((- b' + 1 * sqrt\ (b'^2 - 4 * a' * c')) / (2 * a')) +$
$f =$
$0) \wedge$
$(\forall (d, e, f) \in set\ les.$
$d * ((- b' + 1 * sqrt\ (b'^2 - 4 * a' * c')) / (2 * a'))^2 +$
$e * ((- b' + 1 * sqrt\ (b'^2 - 4 * a' * c')) / (2 * a')) +$
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
$< 0))) \vee$
$(\forall (d, e, f) \in set\ eq.\ d = 0 \wedge e = 0 \wedge f = 0) \wedge$
$(\exists x.\ \forall (a, b, c) \in set\ les.\ a * x^2 + b * x + c < 0))$
have *h1*: $(\exists (a', b', c') \in set\ eq.$
$(a' = 0 \wedge b' \neq 0) \wedge$
$(\forall (d, e, f) \in set\ eq.\ d * (- c' / b')^2 + e * (- c' / b') + f = 0) \wedge$
$(\forall (d, e, f) \in set\ les.\ d * (- c' / b')^2 + e * (- c' / b') + f < 0)) \Longrightarrow ?t$
by *auto*
have *h2*: $(\exists (a', b', c') \in set\ eq.$
$a' \neq 0 \wedge$
$- b'^2 + 4 * a' * c' \leq 0 \wedge$
$((\forall (d, e, f) \in set\ eq.$
$d * ((- b' + 1 * sqrt\ (b'^2 - 4 * a' * c')) / (2 * a'))^2 +$
$e * ((- b' + 1 * sqrt\ (b'^2 - 4 * a' * c')) / (2 * a')) +$
$f =$
$0) \wedge$
$(\forall (d, e, f) \in set\ les.$
$d * ((- b' + 1 * sqrt\ (b'^2 - 4 * a' * c')) / (2 * a'))^2 +$
$e * ((- b' + 1 * sqrt\ (b'^2 - 4 * a' * c')) / (2 * a')) +$
f
$< 0))) \Longrightarrow ?t$
by *auto*
have *h3*: $(\exists (a', b', c') \in set\ eq.\ a' \neq 0 \wedge$
$- b'^2 + 4 * a' * c' \leq 0 \wedge (\forall (d, e, f) \in set\ eq.$
$d * ((- b' + - 1 * sqrt\ (b'^2 - 4 * a' * c')) / (2 * a'))^2 +$
$e * ((- b' + - 1 * sqrt\ (b'^2 - 4 * a' * c')) / (2 * a')) +$
$f =$
$0) \wedge$
$(\forall (d, e, f) \in set\ les.$

$d * ((-\ b' + -\ 1 * sqrt\ (b'^2 - 4 * a' * c')) / (2 * a'))^2 +$
$e * ((-\ b' + -\ 1 * sqrt\ (b'^2 - 4 * a' * c')) / (2 * a')) +$
f
$< 0)) \Longrightarrow ?t$

by *auto*

show *?thesis*

using *big-asm h1 h2 h3*

by *presburger*

qed

2.8.3 The eq_qe lemma

lemma *eq-qe-forwards*:

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

proof −

let *?big-or* $= (\exists (a',\ b',\ c') \in set\ eq.$
$(a' = 0 \wedge b' \neq 0) \wedge$
$(\forall (d,\ e,\ f) \in set\ eq.\ d * (-\ c' / b')^2 + e * (-\ c' / b') + f = 0) \wedge$
$(\forall (d,\ e,\ f) \in set\ les.\ d * (-\ c' / b')^2 + e * (-\ c' / b') + f < 0)) \vee$
$(\exists (a',\ b',\ c') \in set\ eq.$

$a' \neq 0 \wedge$
$-\ b'^2 + 4 * a' * c' \leq 0 \wedge$
$((\forall (d,\ e,\ f) \in set\ eq.$
$d * ((-\ b' + 1 * sqrt\ (b'^2 - 4 * a' * c')) / (2 * a'))^2 +$
$e * ((-\ b' + 1 * sqrt\ (b'^2 - 4 * a' * c')) / (2 * a')) +$
$f =$
$0) \wedge$
$(\forall (d,\ e,\ f) \in set\ les.$
$d * ((-\ b' + 1 * sqrt\ (b'^2 - 4 * a' * c')) / (2 * a'))^2 +$
$e * ((-\ b' + 1 * sqrt\ (b'^2 - 4 * a' * c')) / (2 * a')) +$
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
$< 0)) \vee$
$((\forall (d,\ e,\ f) \in set\ eq.\ d = 0 \wedge e = 0 \wedge f = 0) \wedge$
$(\exists x.\ \forall (a,\ b,\ c) \in set\ les.\ a * x^2 + b * x + c < 0))$
assume *asm*: $(\exists x.\ (\forall (a,\ b,\ c) \in set\ eq.\ a * x^2 + b * x + c = 0) \wedge$
$(\forall (a,\ b,\ c) \in set\ les.\ a * x^2 + b * x + c < 0))$
then obtain x **where** *x-prop*: $(\forall (a,\ b,\ c) \in set\ eq.\ a * x^2 + b * x + c = 0) \wedge$
$(\forall (a,\ b,\ c) \in set\ les.\ a * x^2 + b * x + c < 0)$ **by** *auto*
have $\neg\ (\exists (a',\ b',\ c') \in set\ eq.$
$(a' = 0 \wedge b' \neq 0) \wedge$
$(\forall (d,\ e,\ f) \in set\ eq.\ d * (-\ c' / b')^2 + e * (-\ c' / b') + f = 0) \wedge$
$(\forall (d,\ e,\ f) \in set\ les.\ d * (-\ c' / b')^2 + e * (-\ c' / b') + f < 0)) \wedge$
$\neg\ (\exists (a',\ b',\ c') \in set\ eq.$
$a' \neq 0 \wedge$
$-\ b'^2 + 4 * a' * c' \leq 0 \wedge$
$((\forall (d,\ e,\ f) \in set\ eq.$
$d * ((-\ b' + 1 * sqrt\ (b'^2 - 4 * a' * c')) / (2 * a'))^2 +$
$e * ((-\ b' + 1 * sqrt\ (b'^2 - 4 * a' * c')) / (2 * a')) +$
$f =$
$0) \wedge$
$(\forall (d,\ e,\ f) \in set\ les.$
$d * ((-\ b' + 1 * sqrt\ (b'^2 - 4 * a' * c')) / (2 * a'))^2 +$
$e * ((-\ b' + 1 * sqrt\ (b'^2 - 4 * a' * c')) / (2 * a')) +$
f
$< 0))) \wedge$
$\neg\ (\exists (a',\ b',\ c') \in set\ eq.\ a' \neq 0 \wedge$
$-\ b'^2 + 4 * a' * c' \leq 0 \wedge (\forall (d,\ e,\ f) \in set\ eq.$

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

proof −

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

have *not-lin*: $\neg (\exists (a',\ b',\ c') \in set\ eq.$
$(a' = 0 \wedge b' \neq 0) \wedge$
$(\forall (d,\ e,\ f) \in set\ eq.\ d * (-\ c' / b')^2 + e * (-\ c' / b') + f = 0) \wedge$
$(\forall (d,\ e,\ f) \in set\ les.\ d * (-\ c' / b')^2 + e * (-\ c' / b') + f < 0))$
using *big-conj* **by** *auto*

have *not-quad1*: $\neg (\exists (a',\ b',\ c') \in set\ eq.$
$a' \neq 0 \wedge$

$- b'^2 + 4 * a' * c' \leq 0 \wedge$
$((\forall (d, e, f) \in set\ eq.$
$d * ((- b' + 1 * sqrt\ (b'^2 - 4 * a' * c')) / (2 * a'))^2 +$
$e * ((- b' + 1 * sqrt\ (b'^2 - 4 * a' * c')) / (2 * a')) +$
$f =$
$0) \wedge$
$(\forall (d, e, f) \in set\ les.$
$d * ((- b' + 1 * sqrt\ (b'^2 - 4 * a' * c')) / (2 * a'))^2 +$
$e * ((- b' + 1 * sqrt\ (b'^2 - 4 * a' * c')) / (2 * a')) +$
f
$< 0)))$
using *big-conj* **by** *auto*
have *not-quad2*: $\neg(\exists (a', b', c') \in set\ eq.\ a' \neq 0 \wedge$
$- b'^2 + 4 * a' * c' \leq 0 \wedge (\forall (d, e, f) \in set\ eq.$
$d * ((- b' + - 1 * sqrt\ (b'^2 - 4 * a' * c')) / (2 * a'))^2 +$
$e * ((- b' + - 1 * sqrt\ (b'^2 - 4 * a' * c')) / (2 * a')) +$
$f =$
$0) \wedge$
$(\forall (d, e, f) \in set\ les.$
$d * ((- b' + - 1 * sqrt\ (b'^2 - 4 * a' * c')) / (2 * a'))^2 +$
$e * ((- b' + - 1 * sqrt\ (b'^2 - 4 * a' * c')) / (2 * a')) +$
f
$< 0))$
using *big-conj* **by** *auto*
have *not-zer*: $\neg((\forall (d, e, f) \in set\ eq.\ d = 0 \wedge e = 0 \wedge f = 0) \wedge$
$(\exists x.\ \forall (a, b, c) \in set\ les.\ a * x^2 + b * x + c < 0))$
using *big-conj* **by** *auto*
then have *not-zer1*: $\neg(\forall (d, e, f) \in set\ eq.\ d = 0 \wedge e = 0 \wedge f = 0) \vee$
$\neg\ (\exists x.\ \forall (a, b, c) \in set\ les.\ a * x^2 + b * x + c < 0)$ **by** *auto*
have $(\exists x.\ \forall (a, b, c) \in set\ les.\ a * x^2 + b * x + c < 0)$ **using** *asm*
by *auto*
then have $\neg(\forall (d, e, f) \in set\ eq.\ d = 0 \wedge e = 0 \wedge f = 0)$ **using** *not-zer1* **by** *auto*
then have $\exists\ (d, e, f) \in set\ eq.\ d \neq 0 \vee e \neq 0 \vee f \neq 0$
by *auto*
then obtain *d e f* **where** *def-prop*: $(d, e, f) \in set\ eq \wedge (d \neq 0 \vee e \neq 0 \vee f \neq 0)$ **by** *auto*
then have *eval-at-x*: *d*x^2 + e*x + f = 0* **using** *x-prop* **by** *auto*
have *dnonz*: $d \neq 0 \Longrightarrow False$
proof −
assume *dneq*: $d \neq 0$
then have *discr*: $-(e^2) + 4 * d * f \leq 0$ **using** *discriminant-negative*[*of d e f x*] *eval-at-x* **unfolding** *discrim-def*
by *linarith*
let *?r1* = $(- e + - 1 * sqrt\ (e^2 - 4 * d * f)) / (2 * d)$
let *?r2* = $(- e + 1 * sqrt\ (e^2 - 4 * d * f)) / (2 * d)$
have *xis*: $x = ?r1 \vee x = ?r2$
using *dneq discr discriminant-nonneg*[*of d e f x*] *eval-at-x* **unfolding** *discrim-def*

by *auto*
have *xr1*: $x = ?r1 \Longrightarrow False$
using *not-quad2 x-prop discr def-prop dneq* **by** *auto*
have *xr2*: $x = ?r2 \Longrightarrow False$
using *not-quad1 x-prop discr def-prop dneq* **by** *auto*
show *False* **using** *xr1 xr2 xis* **by** *auto*
qed
then have *dz*: $d = 0$ **by** *auto*
have *enonz*: $e \neq 0 \Longrightarrow False$
proof −
assume *enonz*: $e \neq 0$
then have $x = -f/e$ **using** *dz eval-at-x*
by (*metis add.commute minus-add-cancel mult.commute mult-zero-class.mult-zero-left nonzero-eq-divide-eq*)
then show *False*
using *not-lin x-prop enonz dz def-prop* **by** *auto*
qed
then have *ez*: $e = 0$ **by** *auto*
have *fnonz*: $f \neq 0 \Longrightarrow False$ **using** *ez dz eval-at-x* **by** *auto*
show *False*
using *def-prop dnonz enonz fnonz* **by** *auto*
qed
then have *h*: $\neg(?big\text{-}or) \Longrightarrow False$
by *auto*
then show *?thesis* **using** *equiv-lemma*
by *presburger*
qed

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

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

```
        f =
        0) ∧
      (∀(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))) ∨

   (∃(a', b', c')∈set eq. a' ≠ 0 ∧
     − b'^2 + 4 * a' * c' ≤ 0 ∧ (∀(d, e, f)∈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))
 by auto
have h1: (∃(a', b', c')∈set eq.
      (a' = 0 ∧ b' ≠ 0) ∧
      (∀(d, e, f)∈set eq. d * (− c' / b')^2 + e * (− c' / b') + f = 0) ∧
      (∀(d, e, f)∈set les. d * (− c' / b')^2 + e * (− c' / b') + f < 0)) ⟹
 (∃(x::real). (∀(a, b, c)∈set eq. a * x^2 + b * x + c = 0) ∧
      (∀(a, b, c)∈set les. a * x^2 + b * x + c < 0))
proof −
 assume (∃(a', b', c')∈set eq.
      (a' = 0 ∧ b' ≠ 0) ∧
      (∀(d, e, f)∈set eq. d * (− c' / b')^2 + e * (− c' / b') + f = 0) ∧
      (∀(d, e, f)∈set les. d * (− c' / b')^2 + e * (− c' / b') + f < 0))
 then obtain a' b' c' where abc-prop: (a', b', c')∈set eq ∧
      (a' = 0 ∧ b' ≠ 0) ∧
      (∀(d, e, f)∈set eq. d * (− c' / b')^2 + e * (− c' / b') + f = 0) ∧
      (∀(d, e, f)∈set les. d * (− c' / b')^2 + e * (− c' / b') + f < 0) by auto
 let ?x = (−c' /b')::real
 have (∀(d, e, f)∈set eq. d * ?x^2 + e * ?x + f = 0) ∧
      (∀(d, e, f)∈set les. d * ?x^2 + e * ?x + f < 0) using abc-prop by auto
 then  show ?thesis using abc-prop by blast
qed
have h2:  (∃(a', b', c')∈set eq.
      a' ≠ 0 ∧
      − b'^2 + 4 * a' * c' ≤ 0 ∧
      ((∀(d, e, f)∈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
$< 0))) \Longrightarrow (\exists x.\ (\forall (a,\ b,\ c) \in set\ eq.\ a * x^2 + b * x + c = 0) \wedge$
$(\forall (a,\ b,\ c) \in set\ les.\ a * x^2 + b * x + c < 0))$

proof −

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

then obtain $a'\ b'\ c'$ **where** *abc-prop*: $(a',\ b',\ c') \in set\ eq \wedge a' \neq 0 \wedge$
$-\ b'^2 + 4 * a' * c' \leq 0 \wedge$
$((\forall (d,\ e,\ f) \in set\ eq.$
$d * ((-\ b' + 1 * sqrt\ (b'^2 - 4 * a' * c')) / (2 * a'))^2 +$
$e * ((-\ b' + 1 * sqrt\ (b'^2 - 4 * a' * c')) / (2 * a')) +$
$f =$
$0) \wedge$
$(\forall (d,\ e,\ f) \in set\ les.$
$d * ((-\ b' + 1 * sqrt\ (b'^2 - 4 * a' * c')) / (2 * a'))^2 +$
$e * ((-\ b' + 1 * sqrt\ (b'^2 - 4 * a' * c')) / (2 * a')) +$
f
$< 0))$ **by** *auto*

let $?x = ((-\ b' + 1 * sqrt\ (b'^2 - 4 * a' * c')) / (2 * a')::real)$

have *anonz*: $a' \neq 0$ **using** *abc-prop* **by** *auto*

then have $\exists (q::real).\ q = ?x$ **by** *auto*

then obtain q **where** *q-prop*: $q = ?x$ **by** *auto*

have $(\forall (d,\ e,\ f) \in set\ eq.\ d * (?x)^2 + e * (?x) + f = 0) \wedge$
$(\forall (d,\ e,\ f) \in set\ les.\ d * (?x)^2 + e * (?x) + f < 0)$
using *abc-prop* **by** *auto*

then have $(\forall (d,\ e,\ f) \in set\ eq.\ d * q^2 + e * q + f = 0) \wedge$
$(\forall (d,\ e,\ f) \in set\ les.\ d * q^2 + e * q + f < 0)$ **using** *q-prop* **by** *auto*

then show *?thesis* **by** *auto*

qed

have *h3*: $(\exists (a',\ b',\ c') \in set\ eq.\ a' \neq 0 \wedge$
$-\ b'^2 + 4 * a' * c' \leq 0 \wedge (\forall (d,\ e,\ f) \in set\ eq.$
$d * ((-\ b' + -\ 1 * sqrt\ (b'^2 - 4 * a' * c')) / (2 * a'))^2 +$
$e * ((-\ b' + -\ 1 * sqrt\ (b'^2 - 4 * a' * c')) / (2 * a')) +$
$f =$

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

proof −
assume $(\exists (a',\ b',\ c') \in set\ eq.\ a' \neq 0 \wedge$
$-\ b'^2 + 4 * a' * c' \leq 0 \wedge (\forall (d,\ e,\ f) \in set\ eq.$
$d * ((-\ b' + -\ 1 * sqrt\ (b'^2 - 4 * a' * c')) / (2 * a'))^2 +$
$e * ((-\ b' + -\ 1 * sqrt\ (b'^2 - 4 * a' * c')) / (2 * a')) +$
$f =$
$0) \wedge$
$(\forall (d,\ e,\ f) \in set\ les.$
$d * ((-\ b' + -\ 1 * sqrt\ (b'^2 - 4 * a' * c')) / (2 * a'))^2 +$
$e * ((-\ b' + -\ 1 * sqrt\ (b'^2 - 4 * a' * c')) / (2 * a')) +$
f
$< 0))$
then obtain $a'\ b'\ c'$ **where** *abc-prop*: $a' \neq 0 \wedge$
$-\ b'^2 + 4 * a' * c' \leq 0 \wedge (a',\ b',\ c') \in set\ eq \wedge (\forall (d,\ e,\ f) \in set\ eq.$
$d * ((-\ b' + -\ 1 * sqrt\ (b'^2 - 4 * a' * c')) / (2 * a'))^2 +$
$e * ((-\ b' + -\ 1 * sqrt\ (b'^2 - 4 * a' * c')) / (2 * a')) +$
$f =$
$0) \wedge$
$(\forall (d,\ e,\ f) \in set\ les.$
$d * ((-\ b' + -\ 1 * sqrt\ (b'^2 - 4 * a' * c')) / (2 * a'))^2 +$
$e * ((-\ b' + -\ 1 * sqrt\ (b'^2 - 4 * a' * c')) / (2 * a')) +$
f
$< 0)$ **by** *auto*
let $?x = (-\ b' + -\ 1 * sqrt\ (b'^2 - 4 * a' * c')) / (2 * a')$
have *anonz*: $a' \neq 0$ **using** *abc-prop* **by** *auto*
then have $\exists (q::real).\ q = ?x$ **by** *auto*
then obtain q **where** *q-prop*: $q = ?x$ **by** *auto*
have $(\forall (d,\ e,\ f) \in set\ eq.\ d * (?x)^2 + e * (?x) + f = 0) \wedge$
$(\forall (d,\ e,\ f) \in set\ les.\ d * (?x)^2 + e * (?x) + f < 0)$
using *abc-prop* **by** *auto*
then have $(\forall (d,\ e,\ f) \in set\ eq.\ d * q^2 + e * q + f = 0) \wedge$
$(\forall (d,\ e,\ f) \in set\ les.\ d * q^2 + e * q + f < 0)$ **using** *q-prop* **by** *auto*
then show *?thesis* **by** *auto*
qed
have *h4*: $((\forall (d,\ e,\ f) \in set\ eq.\ d = 0 \wedge e = 0 \wedge f = 0) \wedge$
$(\exists x.\ \forall (a,\ b,\ c) \in set\ les.\ a * x^2 + b * x + c < 0)) \Longrightarrow (\exists x.\ (\forall (a,\ b,\ c) \in set\ eq.$
$a * x^2 + b * x + c = 0) \wedge$
$(\forall (a,\ b,\ c) \in set\ les.\ a * x^2 + b * x + c < 0))$
proof −
assume *asm*: $((\forall (d,\ e,\ f) \in set\ eq.\ d = 0 \wedge e = 0 \wedge f = 0) \wedge$
$(\exists x.\ \forall (a,\ b,\ c) \in set\ les.\ a * x^2 + b * x + c < 0))$
then have *allzer*: $(\forall (d,\ e,\ f) \in set\ eq.\ d = 0 \wedge e = 0 \wedge f = 0)$ **by** *auto*

have $(\exists\, x.\ \forall\,(a,\ b,\ c)\in set\ les.\ a * x^2 + b * x + c < 0)$ **using** *asm* **by** *auto*
then obtain *x* **where** *x-prop*: $\forall\,(a,\ b,\ c)\in set\ les.\ a * x^2 + b * x + c < 0$ **by** *auto*
then have $\forall\,(d,\ e,\ f)\in set\ eq.\ d*x\hat{}2 + e*x + f = 0$
using *allzer* **by** *auto*
then show *?thesis* **using** *x-prop* **by** *auto*
qed
show *?thesis*
using *bigor h1 h2 h3 h4*
by *blast*
qed

lemma *eq-qe* : $(\exists\, x.\ ((\forall\,(a,\ b,\ c)\in set\ eq.\ a * x^2 + b * x + c = 0) \wedge$
$(\forall\,(a,\ b,\ c)\in set\ les.\ a * x^2 + b * x + c < 0))) =$
$((\exists\,(a',\ b',\ c')\in set\ eq.$
$(a' = 0 \wedge b' \neq 0) \wedge$
$(\forall\,(d,\ e,\ f)\in set\ eq.\ d * (-\ c'\ /\ b')^2 + e * (-\ c'\ /\ b') + f = 0) \wedge$
$(\forall\,(d,\ e,\ f)\in set\ les.\ d * (-\ c'\ /\ b')^2 + e * (-\ c'\ /\ b') + f < 0) \vee$
$a' \neq 0 \wedge$
$-\ b'^2 + 4 * a' * c' \leq 0 \wedge$
$((\forall\,(d,\ e,\ f)\in set\ eq.$
$d * ((-\ b' + 1 * sqrt\ (b'^2 - 4 * a' * c'))\ /\ (2 * a'))^2 +$
$e * ((-\ b' + 1 * sqrt\ (b'^2 - 4 * a' * c'))\ /\ (2 * a')) +$
$f =$
$0) \wedge$
$(\forall\,(d,\ e,\ f)\in set\ les.$
$d * ((-\ b' + 1 * sqrt\ (b'^2 - 4 * a' * c'))\ /\ (2 * a'))^2 +$
$e * ((-\ b' + 1 * sqrt\ (b'^2 - 4 * a' * c'))\ /\ (2 * a')) +$
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
$< 0))) \vee$
$(\forall\,(d,\ e,\ f)\in set\ eq.\ d = 0 \wedge e = 0 \wedge f = 0) \wedge$
$(\exists\, x.\ \forall\,(a,\ b,\ c)\in set\ les.\ a * x^2 + b * x + c < 0))$
proof −
have *h1*: $(\exists\, x.\ (\forall\,(a,\ b,\ c)\in set\ eq.\ a * x^2 + b * x + c = 0) \wedge$
$(\forall\,(a,\ b,\ c)\in set\ les.\ a * x^2 + b * x + c < 0)) \longrightarrow$
$((\exists\,(a',\ b',\ c')\in set\ eq.$
$(a' = 0 \wedge b' \neq 0) \wedge$
$(\forall\,(d,\ e,\ f)\in set\ eq.\ d * (-\ c'\ /\ b')^2 + e * (-\ c'\ /\ b') + f = 0) \wedge$

$(\forall\,(d,\ e,\ f)\in set\ les.\ d * (-\ c'\ /\ b')^2 + e * (-\ c'\ /\ b') + f < 0) \vee$
$a' \neq 0 \wedge$
$-\ b'^2 + 4 * a' * c' \leq 0 \wedge$
$((\forall\,(d,\ e,\ f)\in set\ eq.$
$d * ((-\ b' + 1 * sqrt\ (b'^2 - 4 * a' * c'))\ /\ (2 * a'))^2 +$
$e * ((-\ b' + 1 * sqrt\ (b'^2 - 4 * a' * c'))\ /\ (2 * a')) +$
$f =$
$0) \wedge$
$(\forall\,(d,\ e,\ f)\in set\ les.$
$d * ((-\ b' + 1 * sqrt\ (b'^2 - 4 * a' * c'))\ /\ (2 * a'))^2 +$
$e * ((-\ b' + 1 * sqrt\ (b'^2 - 4 * a' * c'))\ /\ (2 * a')) +$
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
$< 0))) \vee$
$(\forall\,(d,\ e,\ f)\in set\ eq.\ d = 0 \wedge e = 0 \wedge f = 0) \wedge$
$(\exists\,x.\ \forall\,(a,\ b,\ c)\in set\ les.\ a * x^2 + b * x + c < 0))$
using *eq-qe-forwards* **by** *auto*
have *h2*: $((\exists\,(a',\ b',\ c')\in set\ eq.$
$(a' = 0 \wedge b' \neq 0) \wedge$
$(\forall\,(d,\ e,\ f)\in set\ eq.\ d * (-\ c'\ /\ b')^2 + e * (-\ c'\ /\ b') + f = 0) \wedge$
$(\forall\,(d,\ e,\ f)\in set\ les.\ d * (-\ c'\ /\ b')^2 + e * (-\ c'\ /\ b') + f < 0) \vee$
$a' \neq 0 \wedge$
$-\ b'^2 + 4 * a' * c' \leq 0 \wedge$
$((\forall\,(d,\ e,\ f)\in set\ eq.$
$d * ((-\ b' + 1 * sqrt\ (b'^2 - 4 * a' * c'))\ /\ (2 * a'))^2 +$
$e * ((-\ b' + 1 * sqrt\ (b'^2 - 4 * a' * c'))\ /\ (2 * a')) +$
$f =$
$0) \wedge$
$(\forall\,(d,\ e,\ f)\in set\ les.$
$d * ((-\ b' + 1 * sqrt\ (b'^2 - 4 * a' * c'))\ /\ (2 * a'))^2 +$
$e * ((-\ b' + 1 * sqrt\ (b'^2 - 4 * a' * c'))\ /\ (2 * a')) +$
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
$< 0))) \vee$
$(\forall (d,\ e,\ f) \in set\ eq.\ d = 0 \wedge e = 0 \wedge f = 0) \wedge$
$(\exists x.\ \forall (a,\ b,\ c) \in set\ les.\ a * x^2 + b * x + c < 0)) \longrightarrow (\exists x.\ (\forall (a,\ b,\ c) \in set\ eq.$
$a * x^2 + b * x + c = 0) \wedge$
$(\forall (a,\ b,\ c) \in set\ les.\ a * x^2 + b * x + c < 0))$
using *eq-qe-backwards* **by** *auto*
have *h3*: $(\exists x.\ (\forall (a,\ b,\ c) \in set\ eq.\ a * x^2 + b * x + c = 0) \wedge$
$(\forall (a,\ b,\ c) \in set\ les.\ a * x^2 + b * x + c < 0)) \longleftrightarrow$
$((\exists (a',\ b',\ c') \in set\ eq.$
$(a' = 0 \wedge b' \neq 0) \wedge$
$(\forall (d,\ e,\ f) \in set\ eq.\ d * (-\ c' / b')^2 + e * (-\ c' / b') + f = 0) \wedge$
$(\forall (d,\ e,\ f) \in set\ les.\ d * (-\ c' / b')^2 + e * (-\ c' / b') + f < 0) \vee$
$a' \neq 0 \wedge$
$-\ b'^2 + 4 * a' * c' \leq 0 \wedge$
$((\forall (d,\ e,\ f) \in set\ eq.$
$d * ((-\ b' + 1 * sqrt\ (b'^2 - 4 * a' * c')) / (2 * a'))^2 +$
$e * ((-\ b' + 1 * sqrt\ (b'^2 - 4 * a' * c')) / (2 * a')) +$
$f =$
$0) \wedge$
$(\forall (d,\ e,\ f) \in set\ les.$
$d * ((-\ b' + 1 * sqrt\ (b'^2 - 4 * a' * c')) / (2 * a'))^2 +$
$e * ((-\ b' + 1 * sqrt\ (b'^2 - 4 * a' * c')) / (2 * a')) +$
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
$< 0))) \vee$
$(\forall (d,\ e,\ f) \in set\ eq.\ d = 0 \wedge e = 0 \wedge f = 0) \wedge$
$(\exists x.\ \forall (a,\ b,\ c) \in set\ les.\ a * x^2 + b * x + c < 0))$
using *h1 h2*
by *smt*
then show *?thesis*
by (*auto*)
qed

2.8.4 The qe_forwards lemma

lemma *qe-forwards-helper-gen*:
fixes *r*:: *real*
assumes *f8*: $\neg(\exists ((a'::real),\ (b'::real),\ (c'::real)) \in set\ c.$

$((a' \neq 0 \vee b' \neq 0) \wedge a' * r\hat{}2 + b' * r + c' = 0) \wedge$
$((\forall (d, e, f) \in set\ a.\ d * r^2 + e * r + f = 0) \wedge$
$(\forall (d, e, f) \in set\ b.\ d * r\hat{}2 + e * r + f < 0) \wedge$
$(\forall (d, e, f) \in set\ c.\ d * r\hat{}2 + e * r + f \leq 0) \wedge$
$(\forall (d, e, f) \in set\ d.\ d * r\hat{}2 + e * r + f \neq 0)))$
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*: $\neg\ (\exists (a', b', c') \in set\ b.\ a' \neq 0 \wedge$
$-\ b'^2 + 4 * a' * c' \leq 0 \wedge$
$((\forall (d, e, f) \in set\ a.$
$\exists y' > (-\ b' + 1 * sqrt\ (b'^2 - 4 * a' * c')) / (2 * a').$
$\forall x \in \{(-\ b' + 1 * sqrt\ (b'^2 - 4 * a' * c')) / (2 * a') < .. y'\}.$
$d * x^2 + e * x + f = 0) \wedge$
$(\forall (d, e, f) \in set\ b.$
$\exists y' > (-\ b' + 1 * sqrt\ (b'^2 - 4 * a' * c')) / (2 * a').$
$\forall x \in \{(-\ b' + 1 * sqrt\ (b'^2 - 4 * a' * c')) / (2 * a') < .. y'\}.$
$d * x^2 + e * x + f < 0) \wedge$
$(\forall (d, e, f) \in set\ c.$
$\exists y' > (-\ b' + 1 * sqrt\ (b'^2 - 4 * a' * c')) / (2 * a').$
$\forall x \in \{(-\ b' + 1 * sqrt\ (b'^2 - 4 * a' * c')) / (2 * a') < .. y'\}.$
$d * x^2 + e * x + f \leq 0) \wedge$
$(\forall (d, e, f) \in set\ d.$
$\exists y' > (-\ b' + 1 * sqrt\ (b'^2 - 4 * a' * c')) / (2 * a').$
$\forall x \in \{(-\ b' + 1 * sqrt\ (b'^2 - 4 * a' * c')) / (2 * a') < .. y'\}.$
$d * x^2 + e * x + f \neq 0)))$
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 *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 −
have $(\exists\,(a',\ b',\ c')\in set\ c.$
$((a'\neq 0 \vee b'\neq 0) \wedge a'*r\hat{}2 + b'*r + c' = 0) \wedge$
$(\forall\,(d,\ e,\ f)\in set\ a.$
$\exists\, y'>r.\ \forall\, x\in\{r<..y'\}.\ d * x^2 + e * x + f = 0) \wedge$
$(\forall\,(d,\ e,\ f)\in set\ b.$
$\exists\, y'>r.\ \forall\, x\in\{r<..y'\}.\ d * x^2 + e * x + f < 0) \wedge$
$(\forall\,(d,\ e,\ f)\in set\ c.$
$\exists\, y'>r.\ \forall\, x\in\{r<..y'\}.\ d * x^2 + e * x + f \leq 0) \wedge$
$(\forall\,(d,\ e,\ f)\in set\ d.$
$\exists\, y'>r.\ \forall\, x\in\{r<..y'\}.\ d * x^2 + e * x + f \neq 0)) \Longrightarrow False$
proof −
assume $(\exists\,(a',\ b',\ c')\in set\ c.$
$((a'\neq 0 \vee b'\neq 0) \wedge a'*r\hat{}2 + b'*r + c' = 0) \wedge$
$(\forall\,(d,\ e,\ f)\in set\ a.$
$\exists\, y'>r.\ \forall\, x\in\{r<..y'\}.\ d * x^2 + e * x + f = 0) \wedge$
$(\forall\,(d,\ e,\ f)\in set\ b.$
$\exists\, y'>r.\ \forall\, x\in\{r<..y'\}.\ d * x^2 + e * x + f < 0) \wedge$
$(\forall\,(d,\ e,\ f)\in set\ c.$
$\exists\, y'>r.\ \forall\, x\in\{r<..y'\}.\ d * x^2 + e * x + f \leq 0) \wedge$
$(\forall\,(d,\ e,\ f)\in set\ d.$
$\exists\, y'>r.\ \forall\, x\in\{r<..y'\}.\ d * x^2 + e * x + f \neq 0))$
then obtain $a'\ b'\ c'$ **where** *abc-prop*: $(a',\ b',\ c')\in set\ c \wedge$
$((a'\neq 0 \vee b'\neq 0) \wedge a'*r\hat{}2 + b'*r + c' = 0) \wedge$
$(\forall\,(d,\ e,\ f)\in set\ a.$
$\exists\, y'>r.\ \forall\, x\in\{r<..y'\}.\ d * x^2 + e * x + f = 0) \wedge$
$(\forall\,(d,\ e,\ f)\in set\ b.$
$\exists\, y'>r.\ \forall\, x\in\{r<..y'\}.\ d * x^2 + e * x + f < 0) \wedge$
$(\forall\,(d,\ e,\ f)\in set\ c.$
$\exists\, y'>r.\ \forall\, x\in\{r<..y'\}.\ d * x^2 + e * x + f \leq 0) \wedge$
$(\forall\,(d,\ e,\ f)\in set\ d.$
$\exists\, y'>r.\ \forall\, x\in\{r<..y'\}.\ d * x^2 + e * x + f \neq 0)$
by *auto*
have *h1*: $(\forall\,(d,\ e,\ f)\in set\ a.\ d * r\hat{}2 + e * r + f = 0)$
using *alleqset*
by *blast*
have *c-prop*: $(\forall\,(d,\ e,\ f)\in set\ c.$
$\exists\, y'>r.\ \forall\, x\in\{r<..y'\}.\ d * x^2 + e * x + f \leq 0)$
using *abc-prop* **by** *auto*

have *h2*: $(\forall (d,\ e,\ f) \in set\ c.\ d * r\hat{}2 + e * r + f \leq 0)$
proof −
have *c1*: $\exists\ (d,\ e,\ f) \in set\ c.\ \ d * (r)^2 + e * (r) + f > 0 \Longrightarrow False$
proof −
assume $\exists\ (d,\ e,\ f) \in set\ c.\ \ d * (r)^2 + e * (r) + f > 0$
then obtain *d e f* **where** *def-prop*: $(d,\ e,\ f) \in set\ c \wedge d * (r)^2 + e * r + f > 0$
by *auto*
have $\exists\, y'{>}r.\ \forall\, x{\in}\{r{<}..y'\}.\ d * x^2 + e * x + f \leq 0$
using *def-prop c-prop* **by** *auto*
then obtain y' **where** *y-prop*: $y' > r \wedge (\forall\, x{\in}\{r{<}..y'\}.\ d * x^2 + e * x + f \leq 0)$ **by** *auto*
have $\exists\, x{\in}\{r{<}..y'\}.\ d{*}x\hat{}2 + e{*}x + f > 0$
using *def-prop continuity-lem-gt0-expanded*[*of r y' d e f*]
using *y-prop* **by** *linarith*
then show *False* **using** *y-prop*
by *auto*
qed
then show *?thesis*
by *fastforce*
qed
have *b-prop*: $(\forall (d,\ e,\ f){\in}set\ b.\ \exists\, y'{>}r.\ \forall\, x{\in}\{r{<}..y'\}.\ d * x^2 + e * x + f < 0)$
using *abc-prop* **by** *auto*
have *h3*: $(\forall (d,\ e,\ f){\in}set\ b.\ d * r^2 + e * r + f < 0)$
proof −
have *c1*: $\exists\ (d,\ e,\ f) \in set\ b.\ \ d * r^2 + e * r + f > 0 \Longrightarrow False$
proof −
assume $\exists\ (d,\ e,\ f) \in set\ b.\ \ d * r^2 + e * r + f > 0$
then obtain *d e f* **where** *def-prop*: $(d,\ e,\ f) \in set\ b \wedge d * r^2 + e * r + f > 0$
by *auto*
then have $\exists\, y'{>}r.\ \forall\, x{\in}\{r{<}..y'\}.\ d * x^2 + e * x + f < 0$
using *b-prop* **by** *auto*
then obtain y' **where** *y-prop*: $y' > r \wedge (\forall\, x{\in}\{r{<}..y'\}.\ d * x^2 + e * x + f < 0)$ **by** *auto*
then have $\exists\, k.\ k > r \wedge k < y' \wedge d * k\hat{}2 + e * k + f < 0$ **using** *dense*
by (*meson dense greaterThanAtMost-iff less-eq-real-def*)
then obtain *k* **where** *k-prop*: $k > r \wedge k < y' \wedge d * k\hat{}2 + e * k + f < 0$
by *auto*
then have $\neg(\exists\, x{>}r.\ x < y' \wedge d * x^2 + e * x + f = 0)$
using *y-prop* **by** *force*
then show *False* **using** *k-prop def-prop y-prop poly-IVT-neg*[*of r k* [:*f*, *e*, *d*:]] *poly-IVT-pos*[*of* $-c'/b'$ *k* [:*f*, *e*, *d*:]]
by (*smt quadratic-poly-eval*)
qed
have *c2*: $\exists\ (d,\ e,\ f) \in set\ b.\ \ d * r^2 + e * r + f = 0 \Longrightarrow False$
proof −
assume $\exists\ (d,\ e,\ f) \in set\ b.\ \ d * r^2 + e * r + f = 0$

then obtain d' e f **where** *def-prop*: $(d',\ e,\ f) \in set\ b \wedge d' * r^2 + e * r + f = 0$
by *auto*
then have *same*: $(d' = 0 \wedge e \neq 0) \Longrightarrow (-f/e = r)$
proof −
assume *asm*: $(d' = 0 \wedge e \neq 0)$
then have $e * r + f = 0$ **using** *def-prop*
by *auto*
then show $-f/e = r$ **using** *asm*
by (*metis* (*no-types*) *add.commute diff-0 divide-minus-left minus-add-cancel nonzero-mult-div-cancel-left uminus-add-conv-diff*)
qed
let $?r = -f/e$
have $(d' = 0 \wedge e \neq 0) \Longrightarrow ((d',\ e,\ f) \in set\ b \wedge ((\forall (d,\ e,\ f) \in set\ a.\ \exists y' > ?r.\ \forall x \in \{?r<..y'\}.\ d * x^2 + e * x + f = 0) \wedge$
$(\forall (d,\ e,\ f) \in set\ b.\ \exists y' > ?r.\ \forall x \in \{?r<..y'\}.\ d * x^2 + e * x + f < 0) \wedge$
$(\forall (d,\ e,\ f) \in set\ c.\ \exists y' > ?r.\ \forall x \in \{?r<..y'\}.\ d * x^2 + e * x + f \leq 0) \wedge$
$(\forall (d,\ e,\ f) \in set\ d.\ \exists y' > ?r.\ \forall x \in \{?r<..y'\}.\ d * x^2 + e * x + f \neq 0)))$
using *same def-prop abc-prop* **by** *auto*
then have $(d' = 0 \wedge e \neq 0) \Longrightarrow (\exists (a',\ b',\ c') \in set\ b.$
$(a' = 0 \wedge b' \neq 0) \wedge$
$(\forall (d,\ e,\ f) \in set\ a.$
$\exists y' > -\ c' / b'.\ \forall x \in \{-\ c' / b'<..y'\}.\ d * x^2 + e * x + f = 0) \wedge$
$(\forall (d,\ e,\ f) \in set\ b.$
$\exists y' > -\ c' / b'.\ \forall x \in \{-\ c' / b'<..y'\}.\ d * x^2 + e * x + f < 0) \wedge$
$(\forall (d,\ e,\ f) \in set\ c.$
$\exists y' > -\ c' / b'.\ \forall x \in \{-\ c' / b'<..y'\}.\ d * x^2 + e * x + f \leq 0) \wedge$
$(\forall (d,\ e,\ f) \in set\ d.$
$\exists y' > -\ c' / b'.\ \forall x \in \{-\ c' / b'<..y'\}.\ d * x^2 + e * x + f \neq 0))$
by *auto*
then have *f1*: $(d' = 0 \wedge e \neq 0) \Longrightarrow False$ **using** *f5*
by *auto*
have *f2*: $(d' = 0 \wedge e = 0 \wedge f = 0) \Longrightarrow False$ **proof** −
assume $(d' = 0 \wedge e = 0 \wedge f = 0)$
then have *allzer*: $\forall x.\ d'*x\hat{}2 + e*x + f = 0$ **by** *auto*
have $\exists y' > r.\ \forall x \in \{r<..y'\}.\ d' * x^2 + e * x + f < 0$
using *b-prop def-prop* **by** *auto*
then obtain y' **where** *y-prop*: $y' > r \wedge (\forall x \in \{r<..y'\}.\ d' * x^2 + e * x + f < 0)$ **by** *auto*
then have $\exists k.\ k > r \wedge k < y' \wedge d' * k\hat{}2 + e * k + f < 0$ **using** *dense*
by (*meson dense greaterThanAtMost-iff less-eq-real-def*)
then show *False* **using** *allzer*
by *auto*
qed
have *f3*: $d' \neq 0 \Longrightarrow False$
proof −
assume *dnonz*: $d' \neq 0$
have *discr*: $-\ e^2 + 4 * d' * f \leq 0$
using *def-prop discriminant-negative*[*of* d' e f] **unfolding** *discrim-def*

```
      using def-prop by fastforce
    then have two-cases: r = (− e + − 1 * sqrt (e^2 − 4 * d' * f)) / (2 *
d')
    ∨ r = (− e + 1 * sqrt (e² − 4 * d' * f)) / (2 * d')
    using def-prop dnonz discriminant-nonneg[of d' e f] unfolding discrim-def
      by fastforce
    have some-props: ((d', e, f) ∈ set b ∧ d' ≠ 0 ∧ − e² + 4 * d' * f ≤ 0)
      using dnonz def-prop discr by auto
    let ?r1 = (− e + − 1 * sqrt (e^2 − 4 * d' * f)) / (2 * d')
    let ?r2 = (− e + 1 * sqrt (e^2 − 4 * d' * f)) / (2 * d')
    have cf1: r = (− e + − 1 * sqrt (e^2 − 4 * d' * f)) / (2 * d') ⟹ False
    proof −
      assume r = (− e + − 1 * sqrt (e^2 − 4 * d' * f)) / (2 * d')
      then have (d', e, f) ∈ set b ∧ d' ≠ 0 ∧ − e² + 4 * d' * f ≤ 0 ∧
((∀(d, e, f)∈set a. ∃y'>?r1. ∀x∈{?r1<..y'}. d * x² + e * x + f = 0) ∧
(∀(d, e, f)∈set b. ∃y'>?r1. ∀x∈{?r1<..y'}. d * x² + e * x + f < 0) ∧
(∀(d, e, f)∈set c. ∃y'>?r1. ∀x∈{?r1<..y'}. d * x² + e * x + f ≤ 0) ∧
(∀(d, e, f)∈set d. ∃y'>?r1. ∀x∈{?r1<..y'}. d * x² + e * x + f ≠ 0))
        using abc-prop some-props by auto
      then show False using f7 by auto
    qed
    have cf2: r = (− e + 1 * sqrt (e^2 − 4 * d' * f)) / (2 * d') ⟹ False
    proof −
      assume r = (− e + 1 * sqrt (e^2 − 4 * d' * f)) / (2 * d')
      then have (d', e, f) ∈ set b ∧ d' ≠ 0 ∧ − e² + 4 * d' * f ≤ 0 ∧
((∀(d, e, f)∈set a. ∃y'>?r2. ∀x∈{?r2<..y'}. d * x² + e * x + f = 0) ∧
(∀(d, e, f)∈set b. ∃y'>?r2. ∀x∈{?r2<..y'}. d * x² + e * x + f < 0) ∧
(∀(d, e, f)∈set c. ∃y'>?r2. ∀x∈{?r2<..y'}. d * x² + e * x + f ≤ 0) ∧
(∀(d, e, f)∈set d. ∃y'>?r2. ∀x∈{?r2<..y'}. d * x² + e * x + f ≠ 0))
        using abc-prop some-props by auto
      then show False using f6 by auto
    qed
    then show False using two-cases cf1 cf2 by auto
  qed

  have eo: (d' ≠ 0) ∨ (d' = 0 ∧ e ≠ 0) ∨ (d' = 0 ∧ e = 0 ∧ f = 0)
    using def-prop by auto
  then show False using f1 f2 f3 by auto
 qed
 show ?thesis using c1 c2
  by fastforce
qed
have d-prop: (∀(d, e, f)∈set d.
      ∃y'>r. ∀x∈{r<..y'}. d * x² + e * x + f ≠ 0)
 using abc-prop by auto
have h4: (∀(d, e, f)∈set d. d * r² + e * r + f ≠ 0)
proof −
 have (∃(d, e, f)∈set d. d * r² + e * r + f = 0) ⟹ False
 proof −
```

assume $\exists\ (d,\ e,\ f) \in set\ d.\ \ d * r^2 + e * r + f = 0$
then obtain $d'\ e\ f$ **where** *def-prop*: $(d',\ e,\ f) \in set\ d \wedge d' * r^2 + e * r + f = 0$
by *auto*
then have *same*: $(d' = 0 \wedge e \neq 0) \Longrightarrow (-f/e = r)$
proof −
assume *asm*: $(d' = 0 \wedge e \neq 0)$
then have $e * r + f = 0$ **using** *def-prop*
by *auto*
then show $-f/e = r$ **using** *asm*
by (*metis* (*no-types*) *add.commute diff-0 divide-minus-left minus-add-cancel nonzero-mult-div-cancel-left uminus-add-conv-diff*)
qed
let $?r = -f/e$
have $(d' = 0 \wedge e \neq 0) \Longrightarrow ((d',\ e,\ f) \in set\ d \wedge ((\forall (d,\ e,\ f) \in set\ a.\ \exists y' > ?r.\ \forall x \in \{?r<..y'\}.\ d * x^2 + e * x + f = 0) \wedge$
$(\forall (d,\ e,\ f) \in set\ b.\ \exists y' > ?r.\ \forall x \in \{?r<..y'\}.\ d * x^2 + e * x + f < 0) \wedge$
$(\forall (d,\ e,\ f) \in set\ c.\ \exists y' > ?r.\ \forall x \in \{?r<..y'\}.\ d * x^2 + e * x + f \leq 0) \wedge$
$(\forall (d,\ e,\ f) \in set\ d.\ \exists y' > ?r.\ \forall x \in \{?r<..y'\}.\ d * x^2 + e * x + f \neq 0)))$
using *same def-prop abc-prop* **by** *auto*
then have $(d' = 0 \wedge e \neq 0) \Longrightarrow (\exists (a',\ b',\ c') \in set\ d.$
$(a' = 0 \wedge b' \neq 0) \wedge$
$(\forall (d,\ e,\ f) \in set\ a.$
$\exists y' > -c'/b'.\ \forall x \in \{-c'/b'<..y'\}.\ d * x^2 + e * x + f = 0) \wedge$
$(\forall (d,\ e,\ f) \in set\ b.$
$\exists y' > -c'/b'.\ \forall x \in \{-c'/b'<..y'\}.\ d * x^2 + e * x + f < 0) \wedge$
$(\forall (d,\ e,\ f) \in set\ c.$
$\exists y' > -c'/b'.\ \forall x \in \{-c'/b'<..y'\}.\ d * x^2 + e * x + f \leq 0) \wedge$
$(\forall (d,\ e,\ f) \in set\ d.$
$\exists y' > -c'/b'.\ \forall x \in \{-c'/b'<..y'\}.\ d * x^2 + e * x + f \neq 0))$
by *auto*
then have *f1*: $(d' = 0 \wedge e \neq 0) \Longrightarrow False$ **using** *f10*
by *auto*
have *f2*: $(d' = 0 \wedge e = 0 \wedge f = 0) \Longrightarrow False$ **proof** −
assume $(d' = 0 \wedge e = 0 \wedge f = 0)$
then have *allzer*: $\forall x.\ d' * x\hat{}2 + e * x + f = 0$ **by** *auto*
have $\exists y' > r.\ \forall x \in \{r<..y'\}.\ d' * x^2 + e * x + f \neq 0$
using *d-prop def-prop*
by *auto*
then obtain y' **where** *y-prop*: $y' > r \wedge (\forall x \in \{r<..y'\}.\ d' * x^2 + e * x + f \neq 0)$ **by** *auto*
then have $\exists k.\ k > r \wedge k < y' \wedge d' * k\hat{}2 + e * k + f \neq 0$ **using** *dense*
by (*meson dense greaterThanAtMost-iff less-eq-real-def*)
then show *False* **using** *allzer*
by *auto*
qed
have *f3*: $d' \neq 0 \Longrightarrow False$
proof −
assume *dnonz*: $d' \neq 0$

have *discr*: $-e^2 + 4 * d' * f \leq 0$
using *def-prop discriminant-negative*[*of* d' e f] **unfolding** *discrim-def*
by *fastforce*
then have *two-cases*: $r = (-e + -1 * sqrt\ (e\hat{}2 - 4 * d' * f)) / (2 * d')$
$\vee\ r = (-e + 1 * sqrt\ (e^2 - 4 * d' * f)) / (2 * d')$
using *def-prop dnonz discriminant-nonneg*[*of* d' e f] **unfolding** *discrim-def*
by *fastforce*
have *some-props*: $((d', e, f) \in set\ d \wedge d' \neq 0 \wedge -e^2 + 4 * d' * f \leq 0)$
using *dnonz def-prop discr* **by** *auto*
let *?r1* $= (-e + -1 * sqrt\ (e\hat{}2 - 4 * d' * f)) / (2 * d')$
let *?r2* $= (-e + 1 * sqrt\ (e\hat{}2 - 4 * d' * f)) / (2 * d')$
have *cf1*: $r = (-e + -1 * sqrt\ (e\hat{}2 - 4 * d' * f)) / (2 * d') \Longrightarrow False$
proof −
assume $r = (-e + -1 * sqrt\ (e\hat{}2 - 4 * d' * f)) / (2 * d')$
then have $(d', e, f) \in set\ d \wedge d' \neq 0 \wedge -e^2 + 4 * d' * f \leq 0 \wedge$
$((\forall (d, e, f) \in set\ a.\ \exists y' > ?r1.\ \forall x \in \{?r1 <..< y'\}.\ d * x^2 + e * x + f = 0) \wedge$
$(\forall (d, e, f) \in set\ b.\ \exists y' > ?r1.\ \forall x \in \{?r1 <..< y'\}.\ d * x^2 + e * x + f < 0) \wedge$
$(\forall (d, e, f) \in set\ c.\ \exists y' > ?r1.\ \forall x \in \{?r1 <..< y'\}.\ d * x^2 + e * x + f \leq 0) \wedge$
$(\forall (d, e, f) \in set\ d.\ \exists y' > ?r1.\ \forall x \in \{?r1 <..< y'\}.\ d * x^2 + e * x + f \neq 0))$
using *abc-prop some-props* **by** *auto*
then show *False* **using** *f12* **by** *auto*
qed
have *cf2*: $r = (-e + 1 * sqrt\ (e\hat{}2 - 4 * d' * f)) / (2 * d') \Longrightarrow False$
proof −
assume $r = (-e + 1 * sqrt\ (e\hat{}2 - 4 * d' * f)) / (2 * d')$
then have $(d', e, f) \in set\ d \wedge d' \neq 0 \wedge -e^2 + 4 * d' * f \leq 0 \wedge$
$((\forall (d, e, f) \in set\ a.\ \exists y' > ?r2.\ \forall x \in \{?r2 <..< y'\}.\ d * x^2 + e * x + f = 0) \wedge$
$(\forall (d, e, f) \in set\ b.\ \exists y' > ?r2.\ \forall x \in \{?r2 <..< y'\}.\ d * x^2 + e * x + f < 0) \wedge$
$(\forall (d, e, f) \in set\ c.\ \exists y' > ?r2.\ \forall x \in \{?r2 <..< y'\}.\ d * x^2 + e * x + f \leq 0) \wedge$
$(\forall (d, e, f) \in set\ d.\ \exists y' > ?r2.\ \forall x \in \{?r2 <..< y'\}.\ d * x^2 + e * x + f \neq 0))$
using *abc-prop some-props* **by** *auto*
then show *False* **using** *f11* **by** *auto*
qed
then show *False* **using** *two-cases cf1 cf2* **by** *auto*
qed

have *eo*: $(d' \neq 0) \vee (d' = 0 \wedge e \neq 0) \vee (d' = 0 \wedge e = 0 \wedge f = 0)$
using *def-prop* **by** *auto*
then show *False* **using** *f1 f2 f3* **by** *auto*
qed
then show *?thesis* **by** *auto*
qed
have $(\exists (a', b', c') \in set\ c.\ ((a' \neq 0 \vee b' \neq 0) \wedge a' * r\hat{}2 + b' * r + c' = 0) \wedge$
$(\forall (d, e, f) \in set\ a.\ d * r^2 + e * r + f = 0) \wedge$
$(\forall (d, e, f) \in set\ b.\ d * r^2 + e * r + f < 0) \wedge$
$(\forall (d, e, f) \in set\ c.\ d * r^2 + e * r + f \leq 0) \wedge$
$(\forall (d, e, f) \in set\ d.\ d * r^2 + e * r + f \neq 0))$
using *h1 h2 h3 h4 abc-prop* **by** *auto*

then show *False* **using** *f8* **by** *auto*
qed
then show *?thesis* **by** *auto*
qed

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

proof −

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

proof −

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

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

$\exists\, y'{>}{-}\ c'\ /\ b'.\ \forall\, x{\in}\{-\ c'\ /\ b'{<}..y'\}.\ d * x^2 + e * x + f \neq 0)$
by *auto*
then have *bnonz*: $b'{\neq}0$ **by** *auto*
have *h1*: $(\forall\,(d,\ e,\ f){\in}set\ a.\ d * (-\ c'\ /\ b')^2 + e * (-\ c'\ /\ b') + f = 0)$
using *bnonz alleqset*
by *blast*
have *c-prop*: $(\forall\,(d,\ e,\ f){\in}set\ c.$
$\exists\, y'{>}{-}\ c'\ /\ b'.\ \forall\, x{\in}\{-\ c'\ /\ b'{<}..y'\}.\ d * x^2 + e * x + f \leq 0)$
using *abc-prop* **by** *auto*
have *h2*: $(\forall\,(d,\ e,\ f){\in}set\ c.\ d * (-\ c'\ /\ b')^2 + e * (-\ c'\ /\ b') + f \leq 0)$
proof −
have *c1*: $\exists\ (d,\ e,\ f) \in set\ c.\ \ d * (-\ c'\ /\ b')^2 + e * (-\ c'\ /\ b') + f > 0 \Longrightarrow$ *False*
proof −
assume $\exists\ (d,\ e,\ f) \in set\ c.\ \ d * (-\ c'\ /\ b')^2 + e * (-\ c'\ /\ b') + f > 0$
then obtain *d e f* **where** *def-prop*: $(d,\ e,\ f) \in set\ c \wedge d * (-\ c'\ /\ b')^2 + e * (-\ c'\ /\ b') + f > 0$
by *auto*
have $\exists\, y'{>}{-}\ c'\ /\ b'.\ \forall\, x{\in}\{-\ c'\ /\ b'{<}..y'\}.\ d * x^2 + e * x + f \leq 0$
using *def-prop c-prop* **by** *auto*
then obtain y' **where** *y-prop*: $y' >{-}\ c'\ /\ b' \wedge (\forall\, x{\in}\{-\ c'\ /\ b'{<}..y'\}.\ d * x^2 + e * x + f \leq 0)$ **by** *auto*
have $\exists\, x{\in}\{(-c'/b'){<}..y'\}.\ d{*}x\hat{}2 + e{*}x + f > 0$
using *def-prop continuity-lem-gt0-expanded*[*of* $(-c'/b')\ y'\ d\ e\ f$]
using *y-prop* **by** *linarith*
then show *False* **using** *y-prop*
by *auto*
qed
then show *?thesis*
by *fastforce*
qed
have *b-prop*: $(\forall\,(d,\ e,\ f){\in}set\ b.$
$\exists\, y'{>}{-}\ c'\ /\ b'.\ \forall\, x{\in}\{-\ c'\ /\ b'{<}..y'\}.\ d * x^2 + e * x + f < 0)$
using *abc-prop* **by** *auto*
have *h3*: $(\forall\,(d,\ e,\ f){\in}set\ b.\ d * (-\ c'\ /\ b')^2 + e * (-\ c'\ /\ b') + f < 0)$
proof −
have *c1*: $\exists\ (d,\ e,\ f) \in set\ b.\ \ d * (-\ c'\ /\ b')^2 + e * (-\ c'\ /\ b') + f > 0 \Longrightarrow$ *False*
proof −
assume $\exists\ (d,\ e,\ f) \in set\ b.\ \ d * (-\ c'\ /\ b')^2 + e * (-\ c'\ /\ b') + f > 0$
then obtain *d e f* **where** *def-prop*: $(d,\ e,\ f) \in set\ b \wedge d * (-\ c'\ /\ b')^2 + e * (-\ c'\ /\ b') + f > 0$
by *auto*
then have $\exists\, y'{>}{-}\ c'\ /\ b'.\ \forall\, x{\in}\{-\ c'\ /\ b'{<}..y'\}.\ d * x^2 + e * x + f < 0$
using *b-prop* **by** *auto*
then obtain y' **where** *y-prop*: $y' >{-}\ c'\ /\ b' \wedge (\forall\, x{\in}\{-\ c'\ /\ b'{<}..y'\}.\ d * x^2 + e * x + f < 0)$ **by** *auto*
then have $\exists\, k.\ k > -c'/b' \wedge k < y' \wedge d * k\hat{}2 + e * k + f < 0$ **using** *dense*

by (*meson dense greaterThanAtMost-iff less-eq-real-def*)
then obtain k **where** *k-prop*: $k > -c'/b' \wedge k < y' \wedge d * k\hat{}2 + e * k + f < 0$
by *auto*
then have $\neg(\exists\, x{>}(-c'/b').\ x < y' \wedge d * x^2 + e * x + f = 0)$
using *y-prop* **by** *force*
then show *False* **using** *k-prop def-prop y-prop poly-IVT-neg*[*of* $-c'/b'$ k [:f, e, d:]] *poly-IVT-pos*[*of* $-c'/b'$ k [:f, e, d:]]
by (*smt quadratic-poly-eval*)
qed
have *c2*: $\exists\ (d, e, f) \in set\ b.\ \ d * (-\ c'\ /\ b')^2 + e * (-\ c'\ /\ b') + f = 0 \Longrightarrow$ *False*
proof −
assume $\exists\ (d, e, f) \in set\ b.\ \ d * (-\ c'\ /\ b')^2 + e * (-\ c'\ /\ b') + f = 0$
then obtain $d'\ e\ f$ **where** *def-prop*: $(d', e, f) \in set\ b \wedge d' * (-\ c'\ /\ b')^2 + e * (-\ c'\ /\ b') + f = 0$
by *auto*
then have *same*: $(d' = 0 \wedge e \neq 0) \Longrightarrow (-f/e = -c'/b')$
proof −
assume *asm*: $(d' = 0 \wedge e \neq 0)$
then have $e * (-\ c'\ /\ b') + f = 0$ **using** *def-prop*
by *auto*
then show $-f/e = -c'/b'$ **using** *asm*
by *auto*
qed
let *?r* $= -f/e$
have $(d' = 0 \wedge e \neq 0) \Longrightarrow ((d', e, f) \in set\ b \wedge ((\forall\,(d, e, f){\in}set\ a.\ \exists\, y'{>}?r.\ \forall\, x{\in}\{?r{<}..y'\}.\ d * x^2 + e * x + f = 0) \wedge$
$(\forall\,(d, e, f){\in}set\ b.\ \exists\, y'{>}?r.\ \forall\, x{\in}\{?r{<}..y'\}.\ d * x^2 + e * x + f < 0) \wedge$
$(\forall\,(d, e, f){\in}set\ c.\ \exists\, y'{>}?r.\ \forall\, x{\in}\{?r{<}..y'\}.\ d * x^2 + e * x + f \leq 0) \wedge$
$(\forall\,(d, e, f){\in}set\ d.\ \exists\, y'{>}?r.\ \forall\, x{\in}\{?r{<}..y'\}.\ d * x^2 + e * x + f \neq 0)))$
using *same def-prop abc-prop* **by** *auto*
then have $(d' = 0 \wedge e \neq 0) \Longrightarrow (\exists\,(a', b', c'){\in}set\ b.$
$(a' = 0 \wedge b' \neq 0) \wedge$
$(\forall\,(d, e, f){\in}set\ a.$
$\exists\, y'{>}-\ c'\ /\ b'.\ \forall\, x{\in}\{-\ c'\ /\ b'{<}..y'\}.\ d * x^2 + e * x + f = 0) \wedge$
$(\forall\,(d, e, f){\in}set\ b.$
$\exists\, y'{>}-\ c'\ /\ b'.\ \forall\, x{\in}\{-\ c'\ /\ b'{<}..y'\}.\ d * x^2 + e * x + f < 0) \wedge$
$(\forall\,(d, e, f){\in}set\ c.$
$\exists\, y'{>}-\ c'\ /\ b'.\ \forall\, x{\in}\{-\ c'\ /\ b'{<}..y'\}.\ d * x^2 + e * x + f \leq 0) \wedge$
$(\forall\,(d, e, f){\in}set\ d.$
$\exists\, y'{>}-\ c'\ /\ b'.\ \forall\, x{\in}\{-\ c'\ /\ b'{<}..y'\}.\ d * x^2 + e * x + f \neq 0))$
by *auto*
then have *f1*: $(d' = 0 \wedge e \neq 0) \Longrightarrow$ *False* **using** *f5*
by *auto*
have *f2*: $(d' = 0 \wedge e = 0 \wedge f = 0) \Longrightarrow$ *False* **proof** −
assume $(d' = 0 \wedge e = 0 \wedge f = 0)$
then have *allzer*: $\forall\, x.\ d'{*}x\hat{}2 + e{*}x + f = 0$ **by** *auto*
have $\exists\, y'{>}-\ c'\ /\ b'.\ \forall\, x{\in}\{-\ c'\ /\ b'{<}..y'\}.\ d' * x^2 + e * x + f < 0$

using *b-prop def-prop* **by** *auto*
then obtain y' **where** *y-prop*: $y' > -c' / b' \wedge (\forall x \in \{-c' / b' < .. y'\}.\ d' * x^2 + e * x + f < 0)$ **by** *auto*
then have $\exists k.\ k > -c'/b' \wedge k < y' \wedge d' * k\hat{}2 + e * k + f < 0$ **using** *dense*
by (*meson dense greaterThanAtMost-iff less-eq-real-def*)
then show *False* **using** *allzer*
by *auto*
qed
have *f3*: $d' \neq 0 \Longrightarrow$ *False*
proof −
assume *dnonz*: $d' \neq 0$
have *discr*: $-e^2 + 4 * d' * f \leq 0$
using *def-prop discriminant-negative*[*of* d' e f] **unfolding** *discrim-def*
by *fastforce*
then have *two-cases*: $(-c' / b') = (-e + -1 * sqrt\ (e\hat{}2 - 4 * d' * f)) / (2 * d')$
$\vee\ (-c' / b') = (-e + 1 * sqrt\ (e^2 - 4 * d' * f)) / (2 * d')$
using *def-prop dnonz discriminant-nonneg*[*of* d' e f] **unfolding** *discrim-def*
by *fastforce*
have *some-props*: $((d', e, f) \in set\ b \wedge d' \neq 0 \wedge -e^2 + 4 * d' * f \leq 0)$
using *dnonz def-prop discr* **by** *auto*
let *?r1* $= (-e + -1 * sqrt\ (e\hat{}2 - 4 * d' * f)) / (2 * d')$
let *?r2* $= (-e + 1 * sqrt\ (e\hat{}2 - 4 * d' * f)) / (2 * d')$
have *cf1*: $(-c' / b') = (-e + -1 * sqrt\ (e\hat{}2 - 4 * d' * f)) / (2 * d') \Longrightarrow$ *False*
proof −
assume $(-c' / b') = (-e + -1 * sqrt\ (e\hat{}2 - 4 * d' * f)) / (2 * d')$
then have $(d', e, f) \in set\ b \wedge d' \neq 0 \wedge -e^2 + 4 * d' * f \leq 0 \wedge$
$((\forall (d, e, f) \in set\ a.\ \exists y' > ?r1.\ \forall x \in \{?r1 < .. y'\}.\ d * x^2 + e * x + f = 0) \wedge$
$(\forall (d, e, f) \in set\ b.\ \exists y' > ?r1.\ \forall x \in \{?r1 < .. y'\}.\ d * x^2 + e * x + f < 0) \wedge$
$(\forall (d, e, f) \in set\ c.\ \exists y' > ?r1.\ \forall x \in \{?r1 < .. y'\}.\ d * x^2 + e * x + f \leq 0) \wedge$
$(\forall (d, e, f) \in set\ d.\ \exists y' > ?r1.\ \forall x \in \{?r1 < .. y'\}.\ d * x^2 + e * x + f \neq 0))$
using *abc-prop some-props* **by** *auto*
then show *False* **using** *f7* **by** *auto*
qed
have *cf2*: $(-c' / b') = (-e + 1 * sqrt\ (e\hat{}2 - 4 * d' * f)) / (2 * d') \Longrightarrow$ *False*
proof −
assume $(-c' / b') = (-e + 1 * sqrt\ (e\hat{}2 - 4 * d' * f)) / (2 * d')$
then have $(d', e, f) \in set\ b \wedge d' \neq 0 \wedge -e^2 + 4 * d' * f \leq 0 \wedge$
$((\forall (d, e, f) \in set\ a.\ \exists y' > ?r2.\ \forall x \in \{?r2 < .. y'\}.\ d * x^2 + e * x + f = 0) \wedge$
$(\forall (d, e, f) \in set\ b.\ \exists y' > ?r2.\ \forall x \in \{?r2 < .. y'\}.\ d * x^2 + e * x + f < 0) \wedge$
$(\forall (d, e, f) \in set\ c.\ \exists y' > ?r2.\ \forall x \in \{?r2 < .. y'\}.\ d * x^2 + e * x + f \leq 0) \wedge$
$(\forall (d, e, f) \in set\ d.\ \exists y' > ?r2.\ \forall x \in \{?r2 < .. y'\}.\ d * x^2 + e * x + f \neq 0))$
using *abc-prop some-props* **by** *auto*
then show *False* **using** *f6* **by** *auto*
qed
then show *False* **using** *two-cases cf1 cf2* **by** *auto*

qed

have *eo*: $(d' \neq 0) \vee (d' = 0 \wedge e \neq 0) \vee (d' = 0 \wedge e = 0 \wedge f = 0)$
using *def-prop* **by** *auto*
then show *False* **using** *f1 f2 f3* **by** *auto*
qed
show *?thesis* **using** *c1 c2*
by *fastforce*
qed
have *d-prop*: $(\forall (d,\ e,\ f) \in set\ d.$
$\exists y' > -\ c' \ /\ b'.\ \forall x \in \{-\ c' \ /\ b' < .. y'\}.\ d * x^2 + e * x + f \neq 0)$
using *abc-prop* **by** *auto*
have *h4*: $(\forall (d,\ e,\ f) \in set\ d.\ d * (-\ c' \ /\ b')^2 + e * (-\ c' \ /\ b') + f \neq 0)$
proof −
have $(\exists (d,\ e,\ f) \in set\ d.\ d * (-\ c' \ /\ b')^2 + e * (-\ c' \ /\ b') + f = 0) \Longrightarrow False$

proof −
assume $\exists\ (d,\ e,\ f) \in set\ d.\ \ d * (-\ c' \ /\ b')^2 + e * (-\ c' \ /\ b') + f = 0$
then obtain $d'\ e\ f$ **where** *def-prop*: $(d',\ e,\ f) \in set\ d \wedge d' * (-\ c' \ /\ b')^2$
$+\ e * (-\ c' \ /\ b') + f = 0$
by *auto*
then have *same*: $(d' = 0 \wedge e \neq 0) \Longrightarrow (-f/e = -c'/b')$
proof −
assume *asm*: $(d' = 0 \wedge e \neq 0)$
then have $e * (-\ c' \ /\ b') + f = 0$ **using** *def-prop*
by *auto*
then show $-f/e = -c'/b'$ **using** *asm*
by *auto*
qed
let *?r* $= -f/e$
have $(d' = 0 \wedge e \neq 0) \Longrightarrow ((d',\ e,\ f) \in set\ d \wedge ((\forall (d,\ e,\ f) \in set\ a.\ \exists y' > ?r.$
$\forall x \in \{?r < .. y'\}.\ d * x^2 + e * x + f = 0) \wedge$
$(\forall (d,\ e,\ f) \in set\ b.\ \exists y' > ?r.\ \forall x \in \{?r < .. y'\}.\ d * x^2 + e * x + f < 0) \wedge$
$(\forall (d,\ e,\ f) \in set\ c.\ \exists y' > ?r.\ \forall x \in \{?r < .. y'\}.\ d * x^2 + e * x + f \leq 0) \wedge$
$(\forall (d,\ e,\ f) \in set\ d.\ \exists y' > ?r.\ \forall x \in \{?r < .. y'\}.\ d * x^2 + e * x + f \neq 0)))$
using *same def-prop abc-prop* **by** *auto*
then have $(d' = 0 \wedge e \neq 0) \Longrightarrow (\exists (a',\ b',\ c') \in set\ d.$
$(a' = 0 \wedge b' \neq 0) \wedge$
$(\forall (d,\ e,\ f) \in set\ a.$
$\exists y' > -\ c' \ /\ b'.\ \forall x \in \{-\ c' \ /\ b' < .. y'\}.\ d * x^2 + e * x + f = 0) \wedge$
$(\forall (d,\ e,\ f) \in set\ b.$
$\exists y' > -\ c' \ /\ b'.\ \forall x \in \{-\ c' \ /\ b' < .. y'\}.\ d * x^2 + e * x + f < 0) \wedge$
$(\forall (d,\ e,\ f) \in set\ c.$
$\exists y' > -\ c' \ /\ b'.\ \forall x \in \{-\ c' \ /\ b' < .. y'\}.\ d * x^2 + e * x + f \leq 0) \wedge$
$(\forall (d,\ e,\ f) \in set\ d.$
$\exists y' > -\ c' \ /\ b'.\ \forall x \in \{-\ c' \ /\ b' < .. y'\}.\ d * x^2 + e * x + f \neq 0))$
by *auto*
then have *f1*: $(d' = 0 \wedge e \neq 0) \Longrightarrow False$ **using** *f10*
by *auto*

have *f2*: $(d' = 0 \wedge e = 0 \wedge f = 0) \Longrightarrow$ *False* **proof** −
assume $(d' = 0 \wedge e = 0 \wedge f = 0)$
then have *allzer*: $\forall x.\ d'*x\hat{}2 + e*x + f = 0$ **by** *auto*
have $\exists y'>-\ c'\ /\ b'.\ \forall x\in\{-\ c'\ /\ b'<..y'\}.\ d' * x^2 + e * x + f \neq 0$
using *d-prop def-prop* **by** *auto*
then obtain y' **where** *y-prop*: $y' >- c'\ /\ b' \wedge (\forall x\in\{-\ c'\ /\ b'<..y'\}.\ d' * x^2 + e * x + f \neq 0)$ **by** *auto*
then have $\exists k.\ k > -c'/b' \wedge k < y' \wedge d' * k\hat{}2 + e * k + f \neq 0$ **using** *dense*
by (*meson dense greaterThanAtMost-iff less-eq-real-def*)
then show *False* **using** *allzer*
by *auto*
qed
have *f3*: $d' \neq 0 \Longrightarrow$ *False*
proof −
assume *dnonz*: $d' \neq 0$
have *discr*: $-\ e^2 + 4 * d' * f \leq 0$
using *def-prop discriminant-negative*[*of* d' e f] **unfolding** *discrim-def*
by *fastforce*
then have *two-cases*: $(-\ c'\ /\ b') = (-\ e + -\ 1 * sqrt\ (e\hat{}2 - 4 * d' * f))\ /\ (2 * d')$
$\vee\ (-\ c'\ /\ b') = (-\ e + 1 * sqrt\ (e^2 - 4 * d' * f))\ /\ (2 * d')$
using *def-prop dnonz discriminant-nonneg*[*of* d' e f] **unfolding** *discrim-def*
by *fastforce*
have *some-props*: $((d',\ e,\ f) \in set\ d \wedge d' \neq 0 \wedge -\ e^2 + 4 * d' * f \leq 0)$
using *dnonz def-prop discr* **by** *auto*
let *?r1* $= (-\ e + -\ 1 * sqrt\ (e\hat{}2 - 4 * d' * f))\ /\ (2 * d')$
let *?r2* $= (-\ e + 1 * sqrt\ (e\hat{}2 - 4 * d' * f))\ /\ (2 * d')$
have *cf1*: $(-\ c'\ /\ b') = (-\ e + -\ 1 * sqrt\ (e\hat{}2 - 4 * d' * f))\ /\ (2 * d') \Longrightarrow$ *False*
proof −
assume $(-\ c'\ /\ b') = (-\ e + -\ 1 * sqrt\ (e\hat{}2 - 4 * d' * f))\ /\ (2 * d')$
then have $(d',\ e,\ f) \in set\ d \wedge d' \neq 0 \wedge -\ e^2 + 4 * d' * f \leq 0 \wedge$
$((\forall (d,\ e,\ f)\in set\ a.\ \exists y'>?r1.\ \forall x\in\{?r1<..y'\}.\ d * x^2 + e * x + f = 0) \wedge$
$(\forall (d,\ e,\ f)\in set\ b.\ \exists y'>?r1.\ \forall x\in\{?r1<..y'\}.\ d * x^2 + e * x + f < 0) \wedge$
$(\forall (d,\ e,\ f)\in set\ c.\ \exists y'>?r1.\ \forall x\in\{?r1<..y'\}.\ d * x^2 + e * x + f \leq 0) \wedge$
$(\forall (d,\ e,\ f)\in set\ d.\ \exists y'>?r1.\ \forall x\in\{?r1<..y'\}.\ d * x^2 + e * x + f \neq 0))$
using *abc-prop some-props* **by** *auto*
then show *False* **using** *f12* **by** *auto*
qed
have *cf2*: $(-\ c'\ /\ b') = (-\ e + \ 1 * sqrt\ (e\hat{}2 - 4 * d' * f))\ /\ (2 * d') \Longrightarrow$ *False*
proof −
assume $(-\ c'\ /\ b') = (-\ e + 1 * sqrt\ (e\hat{}2 - 4 * d' * f))\ /\ (2 * d')$
then have $(d',\ e,\ f) \in set\ d \wedge d' \neq 0 \wedge -\ e^2 + 4 * d' * f \leq 0 \wedge$
$((\forall (d,\ e,\ f)\in set\ a.\ \exists y'>?r2.\ \forall x\in\{?r2<..y'\}.\ d * x^2 + e * x + f = 0) \wedge$
$(\forall (d,\ e,\ f)\in set\ b.\ \exists y'>?r2.\ \forall x\in\{?r2<..y'\}.\ d * x^2 + e * x + f < 0) \wedge$
$(\forall (d,\ e,\ f)\in set\ c.\ \exists y'>?r2.\ \forall x\in\{?r2<..y'\}.\ d * x^2 + e * x + f \leq 0) \wedge$
$(\forall (d,\ e,\ f)\in set\ d.\ \exists y'>?r2.\ \forall x\in\{?r2<..y'\}.\ d * x^2 + e * x + f \neq 0))$

using *abc-prop some-props* **by** *auto*
then show *False* **using** *f11* **by** *auto*
qed
then show *False* **using** *two-cases cf1 cf2* **by** *auto*
qed

have *eo*: $(d' \neq 0) \vee (d' = 0 \wedge e \neq 0) \vee (d' = 0 \wedge e = 0 \wedge f = 0)$
using *def-prop* **by** *auto*
then show *False* **using** *f1 f2 f3* **by** *auto*
qed
then show *?thesis* **by** *auto*
qed
have $(\exists (a', b', c') \in set\ c.\ (a' = 0 \wedge b' \neq 0) \wedge$
$(\forall (d, e, f) \in set\ a.\ d * (-c' / b')^2 + e * (-c' / b') + f = 0) \wedge$
$(\forall (d, e, f) \in set\ b.\ d * (-c' / b')^2 + e * (-c' / b') + f < 0) \wedge$
$(\forall (d, e, f) \in set\ c.\ d * (-c' / b')^2 + e * (-c' / b') + f \leq 0) \wedge$
$(\forall (d, e, f) \in set\ d.\ d * (-c' / b')^2 + e * (-c' / b') + f \neq 0))$
using *h1 h2 h3 h4 bnonz abc-prop* **by** *auto*
then show *False* **using** *f8* **by** *auto*
qed
then show *?thesis* **by** *auto*
qed

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*: $\neg(\exists\,(a',\ b',\ c')\in set\ d.$
$(a' = 0 \wedge b' \neq 0) \wedge$
$(\forall\,(d,\ e,\ f)\in set\ a.$
$\exists\,y'>-\ c' / b'.\ \forall\,x\in\{-\ c' / b'<..y'\}.\ d * x^2 + e * x + f = 0) \wedge$
$(\forall\,(d,\ e,\ f)\in set\ b.$
$\exists\,y'>-\ c' / b'.\ \forall\,x\in\{-\ c' / b'<..y'\}.\ d * x^2 + e * x + f < 0) \wedge$
$(\forall\,(d,\ e,\ f)\in set\ c.$
$\exists\,y'>-\ c' / b'.\ \forall\,x\in\{-\ c' / b'<..y'\}.\ d * x^2 + e * x + f \leq 0) \wedge$
$(\forall\,(d,\ e,\ f)\in set\ d.$
$\exists\,y'>-\ c' / b'.\ \forall\,x\in\{-\ c' / b'<..y'\}.\ d * x^2 + e * x + f \neq 0))$
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: ¬(∃(a', b', c')∈set d.
        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').
             ∀x∈{(- b' + - 1 * sqrt (b'^2 - 4 * a' * c')) / (2 * a')<..y'}.
               d * x^2 + e * x + f ≠ 0))
  shows ¬(∃x. (∀(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 -
  have nor: ∀r. ¬(∃(a', b', c')∈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)))
  proof clarsimp
    fix r t u v
    assume inset: (t, u, v) ∈ set c
    assume eo: t = 0 ⟶ u ≠ 0
    assume zero-eq: t*r^2 + u*r + v = 0
    assume ah: ∀x∈set a. case x of (d, e, f) ⇒ d * r^2 + e * r + f = 0
    assume bh: ∀x∈set b. case x of (d, e, f) ⇒ d * r^2 + e * r + f < 0
    assume ch: ∀x∈set c. case x of (d, e, f) ⇒ d * r^2 + e * r + f ≤ 0
    assume dh: ∀x∈set d. case x of (d, e, f) ⇒ d * r^2 + e * r + f ≠ 0
    have two-cases: t ≠ 0 ∨ (t = 0 ∧ u ≠ 0) using eo by auto
    have c1: t ≠ 0 ⟹ False
    proof -
      assume tnonz:  t ≠ 0
      then have discr-prop: - u^2 + 4 * t * v ≤ 0
        using discriminant-negative[of t u v] zero-eq unfolding discrim-def
        by force
      then have ris: r =  ((-u + - 1 * sqrt (u^2 - 4 * t * v)) / (2 * t)) ∨
       r = ((-u + 1 * sqrt (u^2 - 4 * t * v)) / (2 * t))
        using tnonz discriminant-nonneg[of t u v] zero-eq unfolding discrim-def
by auto
      let ?r1 = ((-u + - 1 * sqrt (u^2 - 4 * t * v)) / (2 * t))
```

let *?r2* = ((−*u* + *1* ∗ *sqrt* (*u*^*2* − *4* ∗ *t* ∗ *v*)) / (*2* ∗ *t*))
have *ris1*: *r* = *?r1* ⟹ *False*
proof −
assume *r* = *?r1*
then have (∃(*a'*, *b'*, *c'*)∈*set c*. *a'* ≠ *0* ∧
− *b'*^*2* + *4* ∗ *a'* ∗ *c'* ≤ *0* ∧
(∀(*d*, *e*, *f*)∈*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*) ∧
(∀(*d*, *e*, *f*)∈*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*) ∧
(∀(*d*, *e*, *f*)∈*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 ≤ *0*) ∧
(∀(*d*, *e*, *f*)∈*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 ≠ *0*))
using *inset ah bh ch dh discr-prop tnonz* **by** *auto*
then show *?thesis*
using *f9* **by** *auto*
qed
have *ris2*: *r* = *?r2* ⟹ *False*
proof −
assume *r* = *?r2*
then have (∃(*a'*, *b'*, *c'*)∈*set c*.
a' ≠ *0* ∧
− *b'*^*2* + *4* ∗ *a'* ∗ *c'* ≤ *0* ∧
((∀(*d*, *e*, *f*)∈*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*) ∧
(∀(*d*, *e*, *f*)∈*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*) ∧
(∀(*d*, *e*, *f*)∈*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 ≤ *0*) ∧
(∀(*d*, *e*, *f*)∈*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 ≠ *0*)))
using *inset ah bh ch dh discr-prop tnonz* **by** *auto*

then show *?thesis*
using *f13* **by** *auto*
qed
show *False* **using** *ris ris1 ris2* **by** *auto*
qed
have *c2*: $(t = 0 \wedge u \neq 0) \Longrightarrow False$
proof −
assume *asm*: $t = 0 \wedge u \neq 0$
then have $r = -v/u$ **using** *zero-eq add.right-neutral nonzero-mult-div-cancel-left*

by (*metis add.commute divide-divide-eq-right divide-eq-0-iff neg-eq-iff-add-eq-0*)
then have $(\exists\,(a',\ b',\ c')\in set\ c.\ (a' = 0 \wedge b' \neq 0) \wedge$
$(\forall\,(d,\ e,\ f)\in set\ a.\ d * (-\ c'\ /\ b')^2 + e * (-\ c'\ /\ b') + f = 0) \wedge$
$(\forall\,(d,\ e,\ f)\in set\ b.\ d * (-\ c'\ /\ b')^2 + e * (-\ c'\ /\ b') + f < 0) \wedge$
$(\forall\,(d,\ e,\ f)\in set\ c.\ d * (-\ c'\ /\ b')^2 + e * (-\ c'\ /\ b') + f \leq 0) \wedge$
$(\forall\,(d,\ e,\ f)\in set\ d.\ d * (-\ c'\ /\ b')^2 + e * (-\ c'\ /\ b') + f \neq 0))$
using *asm inset ah bh ch dh* **by** *auto*
then show *False* **using** *f8*
by *auto*
qed
then show *False* **using** *two-cases c1 c2* **by** *auto*
qed
have *keyh*: $\bigwedge r.\ \neg(\exists\,(a',\ b',\ c')\in set\ c.$
$((a'\neq 0 \vee b'\neq 0) \wedge a'*r\hat{}2 + b'*r + c' = 0) \wedge$
$(\forall\,(d,\ e,\ f)\in set\ a.$
$\exists\,y'>r.\ \forall\,x\in\{r<..y'\}.\ d * x^2 + e * x + f = 0) \wedge$
$(\forall\,(d,\ e,\ f)\in set\ b.$
$\exists\,y'>r.\ \forall\,x\in\{r<..y'\}.\ d * x^2 + e * x + f < 0) \wedge$
$(\forall\,(d,\ e,\ f)\in set\ c.$
$\exists\,y'>r.\ \forall\,x\in\{r<..y'\}.\ d * x^2 + e * x + f \leq 0) \wedge$
$(\forall\,(d,\ e,\ f)\in set\ d.$
$\exists\,y'>r.\ \forall\,x\in\{r<..y'\}.\ d * x^2 + e * x + f \neq 0))$
proof −
fix r
have *h8*: $\neg(\exists\,(a',\ b',\ c')\in set\ c.$
$((a'\neq 0 \vee b'\neq 0) \wedge a'*r\hat{}2 + b'*r + c' = 0) \wedge$
$((\forall\,(d,\ e,\ f)\in set\ a.\ d * r^2 + e * r + f = 0) \wedge$
$(\forall\,(d,\ e,\ f)\in set\ b.\ d * r\hat{}2 + e * r + f < 0) \wedge$
$(\forall\,(d,\ e,\ f)\in set\ c.\ d * r\hat{}2 + e * r + f \leq 0) \wedge$
$(\forall\,(d,\ e,\ f)\in set\ d.\ d * r\hat{}2 + e * r + f \neq 0)))$
using *nor* **by** *auto*
show $\neg(\exists\,(a',\ b',\ c')\in set\ c.$
$((a'\neq 0 \vee b'\neq 0) \wedge a'*r\hat{}2 + b'*r + c' = 0) \wedge$
$(\forall\,(d,\ e,\ f)\in set\ a.$
$\exists\,y'>r.\ \forall\,x\in\{r<..y'\}.\ d * x^2 + e * x + f = 0) \wedge$
$(\forall\,(d,\ e,\ f)\in set\ b.$
$\exists\,y'>r.\ \forall\,x\in\{r<..y'\}.\ d * x^2 + e * x + f < 0) \wedge$
$(\forall\,(d,\ e,\ f)\in set\ c.$
$\exists\,y'>r.\ \forall\,x\in\{r<..y'\}.\ d * x^2 + e * x + f \leq 0) \wedge$

$(\forall\,(d,\ e,\ f)\in set\ d.$
$\exists\,y'>r.\ \forall\,x\in\{r<..y'\}.\ d*x^2+e*x+f\neq 0))$
using *qe-forwards-helper-gen*[*of c r a b d*]
alleqset f5 f6 f7 h8 f10 f11 f12
by *auto*
qed
have *f8a*: $\neg(\exists\,(a',\ b',\ c')\in set\ c.$
$(a'=0\wedge b'\neq 0)\wedge$
$(\forall\,(d,\ e,\ f)\in set\ a.$
$\exists\,y'>-\ c'\ /\ b'.\ \forall\,x\in\{-\ c'\ /\ b'<..y'\}.\ d*x^2+e*x+f=0)\wedge$
$(\forall\,(d,\ e,\ f)\in set\ b.$
$\exists\,y'>-\ c'\ /\ b'.\ \forall\,x\in\{-\ c'\ /\ b'<..y'\}.\ d*x^2+e*x+f<0)\wedge$
$(\forall\,(d,\ e,\ f)\in set\ c.$
$\exists\,y'>-\ c'\ /\ b'.\ \forall\,x\in\{-\ c'\ /\ b'<..y'\}.\ d*x^2+e*x+f\leq 0)\wedge$
$(\forall\,(d,\ e,\ f)\in set\ d.$
$\exists\,y'>-\ c'\ /\ b'.\ \forall\,x\in\{-\ c'\ /\ b'<..y'\}.\ d*x^2+e*x+f\neq 0))$
using *qe-forwards-helper-lin*[*of a b c d*] *alleqset f5 f6 f7 f8 f10 f11 f12*
by *blast*
have *f13a*: $\neg\ (\exists\,(a',\ b',\ c')\in set\ c.\ a'\neq 0\wedge$
$-\ b'^2+4*a'*c'\leq 0\wedge$
$((\forall\,(d,\ e,\ f)\in set\ a.$
$\exists\,y'>(-\ b'+1*sqrt\ (b'^2-4*a'*c'))\ /\ (2*a').$
$\forall\,x\in\{(-\ b'+1*sqrt\ (b'^2-4*a'*c'))\ /\ (2*a')<..y'\}.$
$d*x^2+e*x+f=0)\wedge$
$(\forall\,(d,\ e,\ f)\in set\ b.$
$\exists\,y'>(-\ b'+1*sqrt\ (b'^2-4*a'*c'))\ /\ (2*a').$
$\forall\,x\in\{(-\ b'+1*sqrt\ (b'^2-4*a'*c'))\ /\ (2*a')<..y'\}.$
$d*x^2+e*x+f<0)\wedge$
$(\forall\,(d,\ e,\ f)\in set\ c.$
$\exists\,y'>(-\ b'+1*sqrt\ (b'^2-4*a'*c'))\ /\ (2*a').$
$\forall\,x\in\{(-\ b'+1*sqrt\ (b'^2-4*a'*c'))\ /\ (2*a')<..y'\}.$
$d*x^2+e*x+f\leq 0)\wedge$
$(\forall\,(d,\ e,\ f)\in set\ d.$
$\exists\,y'>(-\ b'+1*sqrt\ (b'^2-4*a'*c'))\ /\ (2*a').$
$\forall\,x\in\{(-\ b'+1*sqrt\ (b'^2-4*a'*c'))\ /\ (2*a')<..y'\}.$
$d*x^2+e*x+f\neq 0)))$
proof −
have $(\exists\,(a',\ b',\ c')\in set\ c.\ a'\neq 0\wedge$
$-\ b'^2+4*a'*c'\leq 0\wedge$
$((\forall\,(d,\ e,\ f)\in set\ a.$
$\exists\,y'>(-\ b'+1*sqrt\ (b'^2-4*a'*c'))\ /\ (2*a').$
$\forall\,x\in\{(-\ b'+1*sqrt\ (b'^2-4*a'*c'))\ /\ (2*a')<..y'\}.$
$d*x^2+e*x+f=0)\wedge$
$(\forall\,(d,\ e,\ f)\in set\ b.$
$\exists\,y'>(-\ b'+1*sqrt\ (b'^2-4*a'*c'))\ /\ (2*a').$
$\forall\,x\in\{(-\ b'+1*sqrt\ (b'^2-4*a'*c'))\ /\ (2*a')<..y'\}.$
$d*x^2+e*x+f<0)\wedge$
$(\forall\,(d,\ e,\ f)\in set\ c.$
$\exists\,y'>(-\ b'+1*sqrt\ (b'^2-4*a'*c'))\ /\ (2*a').$

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

proof −
assume $(\exists (a',\ b',\ c'){\in}set\ c.\ a' \neq 0 \wedge$
$-\ b'^2 + 4 * a' * c' \leq 0 \wedge$
$((\forall (d,\ e,\ f){\in}set\ a.$
$\exists y'{>}(-\ b' + 1 * sqrt\ (b'^2 - 4 * a' * c'))\ /\ (2 * a').$
$\forall x{\in}\{(-\ b' + 1 * sqrt\ (b'^2 - 4 * a' * c'))\ /\ (2 * a'){<}..y'\}.$
$d * x^2 + e * x + f = 0) \wedge$
$(\forall (d,\ e,\ f){\in}set\ b.$
$\exists y'{>}(-\ b' + 1 * sqrt\ (b'^2 - 4 * a' * c'))\ /\ (2 * a').$
$\forall x{\in}\{(-\ b' + 1 * sqrt\ (b'^2 - 4 * a' * c'))\ /\ (2 * a'){<}..y'\}.$
$d * x^2 + e * x + f < 0) \wedge$
$(\forall (d,\ e,\ f){\in}set\ c.$
$\exists y'{>}(-\ b' + 1 * sqrt\ (b'^2 - 4 * a' * c'))\ /\ (2 * a').$
$\forall x{\in}\{(-\ b' + 1 * sqrt\ (b'^2 - 4 * a' * c'))\ /\ (2 * a'){<}..y'\}.$
$d * x^2 + e * x + f \leq 0) \wedge$
$(\forall (d,\ e,\ f){\in}set\ d.$
$\exists y'{>}(-\ b' + 1 * sqrt\ (b'^2 - 4 * a' * c'))\ /\ (2 * a').$
$\forall x{\in}\{(-\ b' + 1 * sqrt\ (b'^2 - 4 * a' * c'))\ /\ (2 * a'){<}..y'\}.$
$d * x^2 + e * x + f \neq 0)))$
then obtain $a'\ b'\ c'$ **where** *abc-prop*: $(a',\ b',\ c'){\in}set\ c \wedge a' \neq 0 \wedge$
$-\ b'^2 + 4 * a' * c' \leq 0 \wedge$
$((\forall (d,\ e,\ f){\in}set\ a.$
$\exists y'{>}(-\ b' + 1 * sqrt\ (b'^2 - 4 * a' * c'))\ /\ (2 * a').$
$\forall x{\in}\{(-\ b' + 1 * sqrt\ (b'^2 - 4 * a' * c'))\ /\ (2 * a'){<}..y'\}.$
$d * x^2 + e * x + f = 0) \wedge$
$(\forall (d,\ e,\ f){\in}set\ b.$
$\exists y'{>}(-\ b' + 1 * sqrt\ (b'^2 - 4 * a' * c'))\ /\ (2 * a').$
$\forall x{\in}\{(-\ b' + 1 * sqrt\ (b'^2 - 4 * a' * c'))\ /\ (2 * a'){<}..y'\}.$
$d * x^2 + e * x + f < 0) \wedge$
$(\forall (d,\ e,\ f){\in}set\ c.$
$\exists y'{>}(-\ b' + 1 * sqrt\ (b'^2 - 4 * a' * c'))\ /\ (2 * a').$
$\forall x{\in}\{(-\ b' + 1 * sqrt\ (b'^2 - 4 * a' * c'))\ /\ (2 * a'){<}..y'\}.$
$d * x^2 + e * x + f \leq 0) \wedge$
$(\forall (d,\ e,\ f){\in}set\ d.$
$\exists y'{>}(-\ b' + 1 * sqrt\ (b'^2 - 4 * a' * c'))\ /\ (2 * a').$
$\forall x{\in}\{(-\ b' + 1 * sqrt\ (b'^2 - 4 * a' * c'))\ /\ (2 * a'){<}..y'\}.$
$d * x^2 + e * x + f \neq 0))$ **by** *auto*
let $?r = (-\ b' + 1 * sqrt\ (b'^2 - 4 * a' * c'))\ /\ (2 * a')$
have *somek*: $\exists k.\ k = ?r$ **by** *auto*
then obtain k **where** *k-prop*: $k = ?r$ **by** *auto*
have $(a' \neq 0 \vee b' \neq 0) \wedge (a' * ?r\hat{}2 + b' * ?r + c' = 0)$
using *abc-prop discriminant-nonneg*[*of* $a'\ b'\ c'$]
unfolding *discrim-def* **apply** (*auto*)

by (*metis* (*mono-tags*, *lifting*) *times-divide-eq-right*)
then have ($\exists$ (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*.
$\exists\, y' > ?r$. $\forall\, x \in \{?r <..y'\}$. $d * x^2 + e * x + f \neq 0$))
using *abc-prop* **by** *auto*
then have ($\exists$ (a', b', c')$\in$*set c*.
($a' \neq 0 \vee b' \neq 0$) $\wedge$ $a' * k\hat{}2 + b' * k + c' = 0$) $\wedge$
($\forall$ (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*.
$\exists\, y' > k$. $\forall\, x \in \{k <..y'\}$. $d * x^2 + e * x + f \neq 0$))
using *k-prop* **by** *auto*
then have $\exists\, k$. ($\exists$ (a', b', c')$\in$*set c*.
($a' \neq 0 \vee b' \neq 0$) $\wedge$ $a' * k\hat{}2 + b' * k + c' = 0$) $\wedge$
($\forall$ (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*.
$\exists\, y' > k$. $\forall\, x \in \{k <..y'\}$. $d * x^2 + e * x + f \neq 0$))
by *auto*
then show *False* **using** *keyh* **by** *auto*
qed
then
show *?thesis*
by *auto*
qed
have *f9a*: $\neg$ ($\exists$ (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*.
$\exists\, y' > (- b' + - 1 * sqrt\ (b'^2 - 4 * a' * c')) / (2 * a')$.
$\forall\, x \in \{(- b' + - 1 * sqrt\ (b'^2 - 4 * a' * c')) / (2 * a') <..y'\}$.
$d * x^2 + e * x + f = 0$) $\wedge$
($\forall$ (d, e, f)$\in$*set b*.
$\exists\, y' > (- b' + - 1 * sqrt\ (b'^2 - 4 * a' * c')) / (2 * a')$.
$\forall\, x \in \{(- b' + - 1 * sqrt\ (b'^2 - 4 * a' * c')) / (2 * a') <..y'\}$.
$d * x^2 + e * x + f < 0$) $\wedge$

```
    (∀(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').
         ∀x∈{(− b' + − 1 * sqrt (b'^2 − 4 * a' * c')) / (2 * a')<..y'}.
           d * x^2 + e * x + f ≠ 0))

proof −
  have (∃(a', b', c')∈set c. 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').
         ∀x∈{(− b' + − 1 * sqrt (b'^2 − 4 * a' * c')) / (2 * a')<..y'}.
           d * x^2 + e * x + f ≠ 0)) ⟹ False
  proof −
    assume (∃(a', b', c')∈set c. 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').
           ∀x∈{(− b' + − 1 * sqrt (b'^2 − 4 * a' * c')) / (2 * a')<..y'}.
             d * x^2 + e * x + f ≠ 0))
    then obtain a' b' c' where abc-prop: (a', b', c')∈set c ∧ 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'}.
```

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

let *?r* $= (-b' + -1 * sqrt\ (b'^2 - 4 * a' * c')) / (2 * a')$

have *somek*: $\exists k.\ k = ?r$ **by** *auto*

then obtain k **where** *k-prop*: $k = ?r$ **by** *auto*

have $(a' \neq 0 \vee b' \neq 0) \wedge (a' * ?r\hat{}2 + b' * ?r + c' = 0)$
using *abc-prop discriminant-nonneg*[*of* a' b' c']
unfolding *discrim-def* **apply** (*auto*)
by (*metis* (*mono-tags*, *lifting*) *times-divide-eq-right*)

then have $(\exists (a',\ b',\ c') \in set\ c.$
$((a' \neq 0 \vee b' \neq 0) \wedge a' * ?r\hat{}2 + b' * ?r + c' = 0) \wedge$
$(\forall (d,\ e,\ f) \in set\ a.$
$\exists y' > ?r.\ \forall x \in \{?r <.. y'\}.\ d * x^2 + e * x + f = 0) \wedge$
$(\forall (d,\ e,\ f) \in set\ b.$
$\exists y' > ?r.\ \forall x \in \{?r <.. y'\}.\ d * x^2 + e * x + f < 0) \wedge$
$(\forall (d,\ e,\ f) \in set\ c.$
$\exists y' > ?r.\ \forall x \in \{?r <.. y'\}.\ d * x^2 + e * x + f \leq 0) \wedge$
$(\forall (d,\ e,\ f) \in set\ d.$
$\exists y' > ?r.\ \forall x \in \{?r <.. y'\}.\ d * x^2 + e * x + f \neq 0))$
using *abc-prop* **by** *auto*

then have $(\exists (a',\ b',\ c') \in set\ c.$
$((a' \neq 0 \vee b' \neq 0) \wedge a' * k\hat{}2 + b' * k + c' = 0) \wedge$
$(\forall (d,\ e,\ f) \in set\ a.$
$\exists y' > k.\ \forall x \in \{k <.. y'\}.\ d * x^2 + e * x + f = 0) \wedge$
$(\forall (d,\ e,\ f) \in set\ b.$
$\exists y' > k.\ \forall x \in \{k <.. y'\}.\ d * x^2 + e * x + f < 0) \wedge$
$(\forall (d,\ e,\ f) \in set\ c.$
$\exists y' > k.\ \forall x \in \{k <.. y'\}.\ d * x^2 + e * x + f \leq 0) \wedge$
$(\forall (d,\ e,\ f) \in set\ d.$
$\exists y' > k.\ \forall x \in \{k <.. y'\}.\ d * x^2 + e * x + f \neq 0))$
using *k-prop* **by** *auto*

then have $\exists k.\ (\exists (a',\ b',\ c') \in set\ c.$
$((a' \neq 0 \vee b' \neq 0) \wedge a' * k\hat{}2 + b' * k + c' = 0) \wedge$
$(\forall (d,\ e,\ f) \in set\ a.$
$\exists y' > k.\ \forall x \in \{k <.. y'\}.\ d * x^2 + e * x + f = 0) \wedge$
$(\forall (d,\ e,\ f) \in set\ b.$
$\exists y' > k.\ \forall x \in \{k <.. y'\}.\ d * x^2 + e * x + f < 0) \wedge$
$(\forall (d,\ e,\ f) \in set\ c.$

$\exists\, y'{>}k.\ \forall\, x{\in}\{k{<}..y'\}.\ d * x^2 + e * x + f \leq 0) \wedge$
$(\forall\, (d,\ e,\ f){\in}set\ d.$
$\exists\, y'{>}k.\ \forall\, x{\in}\{k{<}..y'\}.\ d * x^2 + e * x + f \neq 0))$
by *auto*
then show *False* **using** *keyh* **by** *auto*
qed
then
show *?thesis*
by *auto*
qed

have $(\exists\, x.\ (\forall\, (a,\ b,\ c){\in}set\ b.\ a * x^2 + b * x + c < 0) \wedge$
$(\forall\, (a,\ b,\ c){\in}set\ c.\ a * x^2 + b * x + c \leq 0) \wedge$
$(\forall\, (a,\ b,\ c){\in}set\ d.\ a * x^2 + b * x + c \neq 0)) \Longrightarrow False$
proof −
assume $(\exists\, x.\ (\forall\, (a,\ b,\ c){\in}set\ b.\ a * x^2 + b * x + c < 0) \wedge$
$(\forall\, (a,\ b,\ c){\in}set\ c.\ a * x^2 + b * x + c \leq 0) \wedge$
$(\forall\, (a,\ b,\ c){\in}set\ d.\ a * x^2 + b * x + c \neq 0))$
then obtain *x* **where** *x-prop*: $(\forall\, (a,\ b,\ c){\in}set\ b.\ a * x^2 + b * x + c < 0) \wedge$
$(\forall\, (a,\ b,\ c){\in}set\ c.\ a * x^2 + b * x + c \leq 0) \wedge$
$(\forall\, (a,\ b,\ c){\in}set\ d.\ a * x^2 + b * x + c \neq 0)$ **by** *auto*

let *?srl* = *sorted-nonzero-root-list-set* $(((set\ b) \cup (set\ c)) \cup (set\ d))$
have *alleqsetvar*: $\forall\, (t,\ u,\ v) \in set\ a.\ t = 0 \wedge u = 0 \wedge v = 0$
proof *clarsimp*
fix *t u v*
assume $(t,\ u,\ v) \in set\ a$
then have $\forall\, x.\ t{*}x\widehat{\ }2 + u{*}x + v = 0$
using *alleqset* **by** *auto*
then have $\forall\, x{\in}\{0{<}..1\}.\ t * x^2 + u * x + v = 0$
by *auto*
then show $t = 0 \wedge u = 0 \wedge v = 0$
using *continuity-lem-eq0*[*of 0 1 t u v*]
by *auto*
qed

have *lenzero*: *length ?srl* $= 0 \Longrightarrow False$
proof −
assume *lenzero*: *length ?srl* $= 0$
have *ina*: $(\forall\, (a,\ b,\ c){\in}set\ a.\ a = 0 \wedge b = 0 \wedge c = 0)$
using *alleqsetvar* **by** *auto*
have *inb*: $(\forall\, (a,\ b,\ c){\in}set\ b.\ \forall\, y.\ a * y^2 + b * y + c < 0)$
proof *clarsimp*
fix *t u v y*
assume *insetb*: $(t,\ u,\ v) \in set\ b$
then have $t * x^2 + u * x + v < 0$ **using** *x-prop* **by** *auto*
then have *tuv-prop*: $t \neq 0 \vee u \neq 0 \vee v \neq 0$
by *auto*
then have *tuzer*: $(t = 0 \wedge u = 0) \Longrightarrow \neg(\exists\, q.\ t * q^2 + u * q + v = 0)$

by *simp*
then have *tunonz*: $(t \neq 0 \vee u \neq 0) \Longrightarrow \neg(\exists\, q.\ t * q^2 + u * q + v = 0)$
proof −
assume *tuv-asm*: $t \neq 0 \vee u \neq 0$
have $\exists\, q.\ t * q^2 + u * q + v = 0 \Longrightarrow$ *False*
proof −
assume $\exists\ q.\ t * q^2 + u * q + v = 0$
then obtain q **where** $t * q^2 + u * q + v = 0$ **by** *auto*
then have *qin*: $q \in \{x.\ \exists (a,\ b,\ c) \in set\ b \cup set\ c \cup set\ d.\ (a \neq 0 \vee b \neq 0) \wedge a * x^2 + b * x + c = 0\}$
using *insetb tuv-asm tuv-prop* **by** *auto*
have *set ?srl* = *nonzero-root-set* (*set b* ∪ *set c* ∪ *set d*)
unfolding *sorted-nonzero-root-list-set-def*
using *set-sorted-list-of-set*[*of nonzero-root-set* (*set b* ∪ *set c* ∪ *set d*)]
nonzero-root-set-finite[*of* (*set b* ∪ *set c* ∪ *set d*)]
by *auto*
then have $q \in$ *set ?srl* **using** *qin* **unfolding** *nonzero-root-set-def*
by *auto*
then have *List.member ?srl q*
using *in-set-member*[*of q ?srl*]
by *auto*
then show *False*
using *lenzero*
by (*simp add*: *member-rec*(*2*))
qed
then show *?thesis* **by** *auto*
qed
have *nozer*: $\neg(\exists\, q.\ t * q^2 + u * q + v = 0)$
using *tuzer tunonz*
by *blast*
have *samesn*: *sign-num* $(t*x\hat{}2 + u*x + v)$ = *sign-num* $(t*y\hat{}2 + u*y + v)$
proof −
have $x < y \Longrightarrow$ *sign-num* $(t*x\hat{}2 + u*x + v)$ = *sign-num* $(t*y\hat{}2 + u*y + v)$
using *changes-sign-var*[*of t x u v y*] *nozer* **by** *auto*
have $y < x \Longrightarrow$ *sign-num* $(t*x\hat{}2 + u*x + v)$ = *sign-num* $(t*y\hat{}2 + u*y + v)$
using *changes-sign-var*[*of t y u v x*] *nozer*
proof −
assume $y < x$
then show *?thesis*
using ‹$\nexists\, q.\ t * q^2 + u * q + v = 0$› ‹*sign-num* $(t * y^2 + u * y + v) \neq$ *sign-num* $(t * x^2 + u * x + v) \wedge y < x \Longrightarrow \exists\, q.\ t * q^2 + u * q + v = 0 \wedge y \leq q \wedge q \leq x$› **by** *presburger*
qed
show *?thesis*
using *changes-sign-var* **using** ‹$x < y \Longrightarrow$ *sign-num* $(t * x^2 + u * x + v)$ = *sign-num* $(t * y^2 + u * y + v)$› ‹$y < x \Longrightarrow$ *sign-num* $(t * x^2 + u * x + v)$ = *sign-num* $(t * y^2 + u * y + v)$›

```
      by fastforce
    qed

  have sign-num (t*x^2 + u*x + v) = -1 using insetb unfolding sign-num-def
using x-prop
    by auto
  then have sign-num (t*y^2 + u*y + v) = -1 using samesn by auto
  then show t * y^2 + u * y + v < 0 unfolding sign-num-def
    by smt
 qed
 have inc: (∀(a, b, c)∈set c. ∀ y. a * y^2 + b * y + c ≤ 0)
 proof clarsimp
  fix t u v y
  assume insetc: (t, u, v) ∈ set c
  then have t * x^2 + u * x + v ≤ 0 using x-prop by auto
  then have tuzer: t = 0 ∧ u = 0 ⟹ t*y^2 + u*y + v ≤ 0
  proof −
   assume tandu: t = 0 ∧ u = 0
   then have v ≤ 0 using insetc x-prop
    by auto
   then show t*y^2 + u*y + v ≤ 0 using tandu
    by auto
  qed
  have tunonz: t ≠ 0 ∨ u ≠ 0 ⟹ t*y^2 + u*y + v ≤ 0
  proof −
   assume tuv-asm: t ≠ 0 ∨ u ≠ 0
   have insetcvar: t*x^2 + u*x + v < 0
   proof −
    have t*x^2 + u*x + v = 0 ⟹ False
    proof −
     assume zer: t*x^2 + u*x + v = 0
     then have xin: x ∈ {x. ∃(a, b, c)∈set b ∪ set c ∪ set d. (a ≠ 0 ∨ b
≠ 0) ∧ a * x^2 + b * x + c = 0}
      using insetc tuv-asm by auto
     have set ?srl = nonzero-root-set (set b ∪ set c ∪ set d)
      unfolding sorted-nonzero-root-list-set-def
      using set-sorted-list-of-set[of nonzero-root-set (set b ∪ set c ∪ set d)]
        nonzero-root-set-finite[of (set b ∪ set c ∪ set d)]
      by auto
     then have x ∈ set ?srl using xin unfolding nonzero-root-set-def
      by auto
     then have List.member ?srl x
      using in-set-member[of x ?srl]
      by auto
     then show False using lenzero
      by (simp add: member-rec(2))
    qed
    then show ?thesis
     using ‹t * x^2 + u * x + v ≤ 0› by fastforce
```

qed
then have *tunonz*: $\neg(\exists\, q.\ t * q^2 + u * q + v = 0)$
proof −
have $\exists\, q.\ t * q^2 + u * q + v = 0 \Longrightarrow$ *False*
proof −
assume $\exists\ q.\ t * q^2 + u * q + v = 0$
then obtain q **where** $t * q^2 + u * q + v = 0$ **by** *auto*
then have *qin*: $q \in\ \{x.\ \exists (a,\ b,\ c) \in set\ b \cup set\ c \cup set\ d.\ (a \neq 0 \vee b \neq 0) \wedge a * x^2 + b * x + c = 0\}$
using *insetc tuv-asm* **by** *auto*
have *set ?srl* = *nonzero-root-set* (*set b* ∪ *set c* ∪ *set d*)
unfolding *sorted-nonzero-root-list-set-def*
using *set-sorted-list-of-set*[*of nonzero-root-set* (*set b* ∪ *set c* ∪ *set d*)]
nonzero-root-set-finite[*of* (*set b* ∪ *set c* ∪ *set d*)]
by *auto*
then have $q \in$ *set ?srl* **using** *qin* **unfolding** *nonzero-root-set-def*
by *auto*
then have *List.member ?srl q*
using *in-set-member*[*of q ?srl*]
by *auto*
then show *False*
using *lenzero*
by (*simp add*: *member-rec*(*2*))
qed
then show *?thesis* **by** *auto*
qed
have *nozer*: $\neg(\exists\, q.\ t * q^2 + u * q + v = 0)$
using *tuzer tunonz*
by *blast*
have *samesn*: *sign-num* (*t*∗*x*^*2* + *u*∗*x* + *v*) = *sign-num* (*t*∗*y*^*2* + *u*∗*y* + *v*)
proof −
have $x < y \Longrightarrow$ *sign-num* (*t*∗*x*^*2* + *u*∗*x* + *v*) = *sign-num* (*t*∗*y*^*2* + *u*∗*y* + *v*)
using *changes-sign-var*[*of t x u v y*] *nozer* **by** *auto*
have $y < x \Longrightarrow$ *sign-num* (*t*∗*x*^*2* + *u*∗*x* + *v*) = *sign-num* (*t*∗*y*^*2* + *u*∗*y* + *v*)
using *changes-sign-var*[*of t y u v x*] *nozer*
proof −
assume $y < x$
then show *?thesis*
using ‹$\nexists\, q.\ t * q^2 + u * q + v = 0$› ‹*sign-num* $(t * y^2 + u * y + v) \neq$ *sign-num* $(t * x^2 + u * x + v) \wedge y < x \Longrightarrow \exists\, q.\ t * q^2 + u * q + v = 0 \wedge y \leq q \wedge q \leq x$› **by** *presburger*
qed
show *?thesis*
using *changes-sign-var* **using** ‹$x < y \Longrightarrow$ *sign-num* $(t * x^2 + u * x + v) =$ *sign-num* $(t * y^2 + u * y + v)$› ‹$y < x \Longrightarrow$ *sign-num* $(t * x^2 + u * x + v) =$ *sign-num* $(t * y^2 + u * y + v)$›

```
        by fastforce
      qed

        have sign-num (t*x^2 + u*x + v) = -1 using insetcvar unfolding
sign-num-def using x-prop
        by auto
      then have sign-num (t*y^2 + u*y + v) = -1 using samesn by auto
      then show t * y^2 + u * y + v ≤ 0 unfolding sign-num-def
        by smt
    qed
    then show t * y^2 + u * y + v ≤ 0
      using tuzer tunonz
      by blast
  qed
  have ind: (∀(a, b, c)∈set d. ∀y. a * y^2 + b * y + c ≠ 0)
  proof clarsimp
    fix t u v y
    assume insetd: (t, u, v) ∈ set d
    assume falseasm: t * y^2 + u * y + v = 0
    then have snz: sign-num (t*y^2 + u*y + v) = 0
      unfolding sign-num-def by auto
    have t * x^2 + u * x + v ≠ 0 using insetd x-prop by auto
    then have tuv-prop: t ≠ 0 ∨ u ≠ 0 ∨ v ≠ 0
      by auto
    then have tuzer: (t = 0 ∧ u = 0) ⟹ ¬(∃q. t * q^2 + u * q + v = 0)
      by simp
    then have tunonz: (t ≠ 0 ∨ u ≠ 0) ⟹ ¬(∃q. t * q^2 + u * q + v = 0)
    proof −
      assume tuv-asm: t ≠ 0 ∨ u ≠ 0
      have ∃q. t * q^2 + u * q + v = 0 ⟹ False
      proof −
        assume ∃ q. t * q^2 + u * q + v = 0
        then obtain q where t * q^2 + u * q + v = 0 by auto
        then have qin: q ∈ {x. ∃(a, b, c)∈set b ∪ set c ∪ set d. (a ≠ 0 ∨ b ≠
0) ∧ a * x^2 + b * x + c = 0}
          using insetd tuv-asm tuv-prop by auto
        have set ?srl = nonzero-root-set (set b ∪ set c ∪ set d)
          unfolding sorted-nonzero-root-list-set-def
          using set-sorted-list-of-set[of nonzero-root-set (set b ∪ set c ∪ set d)]
            nonzero-root-set-finite[of (set b ∪ set c ∪ set d)]
          by auto
        then have q ∈ set ?srl using qin unfolding nonzero-root-set-def
          by auto
        then have List.member ?srl q
          using in-set-member[of q ?srl]
          by auto
        then show False
          using lenzero
          by (simp add: member-rec(2))
```

```
      qed
      then show ?thesis by auto
    qed
    have nozer: ¬(∃ q. t * q^2 + u * q + v = 0)
      using  tuzer tunonz
      by blast
   have samesn: sign-num (t*x^2 + u*x + v) = sign-num (t*y^2 + u*y + v)
    proof −
      have x < y ⟹ sign-num (t*x^2 + u*x + v) = sign-num (t*y^2 + u*y
+ v)
        using changes-sign-var[of t x u v y] nozer by auto
      have y < x ⟹ sign-num (t*x^2 + u*x + v) = sign-num (t*y^2 + u*y
+ v)
        using changes-sign-var[of t y u v x] nozer
      proof −
        assume y < x
        then show ?thesis
          using ‹∄ q. t * q^2 + u * q + v = 0› ‹sign-num (t * y^2 + u * y + v)
≠ sign-num (t * x^2 + u * x + v) ∧ y < x ⟹ ∃ q. t * q^2 + u * q + v = 0 ∧ y
≤ q ∧ q ≤ x› by presburger
      qed
      show ?thesis
        using changes-sign-var using ‹x < y ⟹ sign-num (t * x^2 + u * x +
v) = sign-num (t * y^2 + u * y + v)› ‹y < x ⟹ sign-num (t * x^2 + u * x + v)
= sign-num (t * y^2 + u * y + v)›
        by fastforce
    qed

    have sign-num (t*x^2 + u*x + v) = −1 ∨ sign-num (t*x^2 + u*x + v)
= 1
      using insetd unfolding sign-num-def using x-prop
      by auto
    then have sign-num (t*y^2 + u*y + v) = −1 ∨ sign-num (t*y^2 + u*y
+ v) = 1 using samesn by auto
    then show False  using snz by auto
  qed

  have ((∀ (a, b, c)∈set a. a = 0 ∧ b = 0 ∧ c = 0) ∧
  (∀ (a, b, c)∈set b. ∀ y. a * y^2 + b * y + c < 0) ∧
  (∀ (a, b, c)∈set c. ∀ y. a * y^2 + b * y + c ≤ 0) ∧
  (∀ (a, b, c)∈set d. ∀ y. a * y^2 + b * y + c ≠ 0))
    using ina inb inc ind by auto
  then  show False
    using f1
    by auto
qed
have cases-mem: (List.member ?srl x) ⟹ False
proof −
  assume (List.member ?srl x)
```

then have $x \in \{x.\ \exists (a,\ b,\ c) \in set\ b \cup set\ c \cup set\ d.\ (a \neq 0 \vee b \neq 0) \wedge a * x^2 + b * x + c = 0\}$
using *set-sorted-list-of-set nonzero-root-set-finite in-set-member*
by (*metis List.finite-set finite-Un nonzero-root-set-def sorted-nonzero-root-list-set-def*)
then have $\exists\ (a,\ b,\ c) \in (((set\ b) \cup (set\ c)) \cup (set\ d))\ .\ (a \neq 0 \vee b \neq 0) \wedge a*x\hat{}2 + b*x + c = 0$
by *blast*
then obtain *t u v* **where** *def-prop*: $(t,\ u,\ v) \in (((set\ b) \cup (set\ c)) \cup (set\ d)) \wedge (t \neq 0 \vee u \neq 0) \wedge t*x\hat{}2 + u*x + v = 0$
by *auto*
have *notinb*: $(t,\ u,\ v) \notin (set\ b)$
proof −
have $(t,\ u,\ v) \in (set\ b\ ) \Longrightarrow False$
proof −
assume $(t,\ u,\ v) \in (set\ b)$
then have $t*x\hat{}2 + u*x + v < 0$ **using** *x-prop*
by *blast*
then show *False* **using** *def-prop*
by *simp*
qed
then show *?thesis* **by** *auto*
qed
have *notind*: $(t,\ u,\ v) \notin (set\ d)$
proof −
have $(t,\ u,\ v) \in (set\ d) \Longrightarrow False$
proof −
assume $(t,\ u,\ v) \in (set\ d)$
then have $t*x\hat{}2 + u*x + v \neq 0$ **using** *x-prop*
by *blast*
then show *False* **using** *def-prop*
by *simp*
qed
then show *?thesis* **by** *auto*
qed
then have *inset*: $(t,\ u,\ v) \in (set\ c)$
using *def-prop notinb notind*
by *blast*
have *case1*: $t \neq 0 \Longrightarrow False$
proof −
assume *tnonz*: $t \neq 0$
then have *r1or2*:$x = (-\ u + -\ 1 * sqrt\ (u^2 - 4 * t * v))\ /\ (2 * t) \vee$
$x = (-\ u + 1 * sqrt\ (u\hat{}2 - 4 * t * v))\ /\ (2 * t)$
using *def-prop discriminant-negative*[*of t u v*] *discriminant-nonneg*[*of t u v*]
apply (*auto*)
using *notinb* **apply** (*force*)
apply (*simp add*: *discrim-def discriminant-iff*)
using *notind* **by** *force*
have *discrh*: $-1*u\hat{}2 + 4 * t * v \leq 0$

using *tnonz discriminant-negative*[*of t u v*] **unfolding** *discrim-def*
using *def-prop* **by** *force*
have *r1*: $x = (-u + -1 * sqrt\ (u^2 - 4 * t * v)) / (2 * t) \Longrightarrow False$
proof −
assume *xis*: $x = (-u + -1 * sqrt\ (u^2 - 4 * t * v)) / (2 * t)$
have $t \neq 0 \wedge$
$-1 * u\hat{}2 + 4 * t * v \leq 0 \wedge$
$(\forall (d, e, f) \in set\ a.$
$d * x^2 + e * x + f = 0) \wedge$
$(\forall (d, e, f) \in set\ b.$
$d * x^2 + e * x + f < 0) \wedge$
$(\forall (d, e, f) \in set\ c.$
$d * x^2 + e * x + f \leq 0) \wedge$
$(\forall (d, e, f) \in set\ d.$
$d * x^2 + e * x + f \neq 0)$
using *tnonz alleqset discrh x-prop*
by *auto*
then have $(\exists (a', b', c') \in set\ c.\ a' \neq 0 \wedge$
$-b'^2 + 4 * a' * c' \leq 0 \wedge$
$(\forall (d, e, f) \in set\ a.$
$d * ((-b' + -1 * sqrt\ (b'^2 - 4 * a' * c')) / (2 * a'))^2 +$
$e * ((-b' + -1 * sqrt\ (b'^2 - 4 * a' * c')) / (2 * a')) +$
$f = 0) \wedge$
$(\forall (d, e, f) \in set\ b.$
$d * ((-b' + -1 * sqrt\ (b'^2 - 4 * a' * c')) / (2 * a'))^2 +$
$e * ((-b' + -1 * sqrt\ (b'^2 - 4 * a' * c')) / (2 * a')) +$
$f < 0) \wedge$
$(\forall (d, e, f) \in set\ c.$
$d * ((-b' + -1 * sqrt\ (b'^2 - 4 * a' * c')) / (2 * a'))^2 +$
$e * ((-b' + -1 * sqrt\ (b'^2 - 4 * a' * c')) / (2 * a')) +$
$f \leq 0) \wedge$
$(\forall (d, e, f) \in set\ d.$
$d * ((-b' + -1 * sqrt\ (b'^2 - 4 * a' * c')) / (2 * a'))^2 +$
$e * ((-b' + -1 * sqrt\ (b'^2 - 4 * a' * c')) / (2 * a')) +$
$f \neq 0))$
using *xis inset*
by *auto*
then show *False*
using *f9* **by** *auto*
qed
have *r2*: $x = (-u + 1 * sqrt\ (u^2 - 4 * t * v)) / (2 * t) \Longrightarrow False$
proof −
assume *xis*: $x = (-u + 1 * sqrt\ (u^2 - 4 * t * v)) / (2 * t)$
have $t \neq 0 \wedge$
$-1 * u\hat{}2 + 4 * t * v \leq 0 \wedge$
$(\forall (d, e, f) \in set\ a.$
$d * x^2 + e * x + f = 0) \wedge$
$(\forall (d, e, f) \in set\ b.$
$d * x^2 + e * x + f < 0) \wedge$

$(\forall (d,\ e,\ f) \in set\ c.$
$\quad d * x^2 + e * x + f \leq 0) \wedge$
$(\forall (d,\ e,\ f) \in set\ d.$
$\quad d * x^2 + e * x + f \neq 0)$
using *tnonz alleqset discrh x-prop*
by *auto*
then have $(\exists (a',\ b',\ c') \in set\ c.\ \ a' \neq 0 \wedge$
$-\ b'^2 + 4 * a' * c' \leq 0 \wedge$
$(\forall (d,\ e,\ f) \in set\ a.$
$\quad d * ((-\ b' + 1 * sqrt\ (b'^2 - 4 * a' * c')) / (2 * a'))^2 +$
$\quad e * ((-\ b' + 1 * sqrt\ (b'^2 - 4 * a' * c')) / (2 * a')) +$
$\quad f = 0) \wedge$
$(\forall (d,\ e,\ f) \in set\ b.$
$\quad d * ((-\ b' + 1 * sqrt\ (b'^2 - 4 * a' * c')) / (2 * a'))^2 +$
$\quad e * ((-\ b' + 1 * sqrt\ (b'^2 - 4 * a' * c')) / (2 * a')) +$
$\quad f < 0) \wedge$
$(\forall (d,\ e,\ f) \in set\ c.$
$\quad d * ((-\ b' + 1 * sqrt\ (b'^2 - 4 * a' * c')) / (2 * a'))^2 +$
$\quad e * ((-\ b' + 1 * sqrt\ (b'^2 - 4 * a' * c')) / (2 * a')) +$
$\quad f \leq 0) \wedge$
$(\forall (d,\ e,\ f) \in set\ d.$
$\quad d * ((-\ b' + 1 * sqrt\ (b'^2 - 4 * a' * c')) / (2 * a'))^2 +$
$\quad e * ((-\ b' + 1 * sqrt\ (b'^2 - 4 * a' * c')) / (2 * a')) +$
$\quad f \neq 0))$
using *xis inset*
by *auto*
then show *False*
using *f13* **by** *auto*
qed
then show *False*
using *r1or2 r1 r2* **by** *auto*
qed
have *case2*: $(t = 0 \wedge u \neq 0) \Longrightarrow False$
proof −
assume *asm*: $t = 0 \wedge u \neq 0$
then have *xis*: $x = -\ v\ /\ u$ **using** *def-prop notinb add.commute diff-0 divide-non-zero minus-add-cancel uminus-add-conv-diff*
by (*metis mult-zero-left*)
have $((t = 0 \wedge u \neq 0) \wedge$
$(\forall (d,\ e,\ f) \in set\ a.\ d * x^2 + e * x + f = 0) \wedge$
$(\forall (d,\ e,\ f) \in set\ b.\ d * x\hat{}2 + e * x + f < 0) \wedge$
$(\forall (d,\ e,\ f) \in set\ c.\ d * x\hat{}2 + e * x + f \leq 0) \wedge$
$(\forall (d,\ e,\ f) \in set\ d.\ d * x\hat{}2 + e * x + f \neq 0))$
using *asm x-prop alleqset* **by** *auto*
then have $(\exists (a',\ b',\ c') \in set\ c.\ (a' = 0 \wedge b' \neq 0) \wedge$
$(\forall (d,\ e,\ f) \in set\ a.\ d * (-\ c'\ /\ b')^2 + e * (-\ c'\ /\ b') + f = 0) \wedge$
$(\forall (d,\ e,\ f) \in set\ b.\ d * (-\ c'\ /\ b')^2 + e * (-\ c'\ /\ b') + f < 0) \wedge$
$(\forall (d,\ e,\ f) \in set\ c.\ d * (-\ c'\ /\ b')^2 + e * (-\ c'\ /\ b') + f \leq 0) \wedge$
$(\forall (d,\ e,\ f) \in set\ d.\ d * (-\ c'\ /\ b')^2 + e * (-\ c'\ /\ b') + f \neq 0))$

```
        using xis inset
        by auto
      then show False
        using f8 by auto
    qed
    show False
      using def-prop case1 case2 by auto
  qed
  have lengt0: length ?srl ≥ 1 ⟹ False
  proof−
    assume asm: length ?srl ≥ 1

    have cases-lt: x < ?srl ! 0 ⟹ False
    proof −
      assume xlt: x < ?srl ! 0
      have samesign: ∀ (a, b, c) ∈ (set b ∪ set c ∪ set d).
          (∀ y < x. sign-num (a * y^2 + b * y + c) =  sign-num (a*x^2 + b*x +
c))
      proof  clarsimp
        fix t u v y
       assume insetunion: (t, u, v) ∈ set b ∨ (t, u, v)  ∈ set c ∨ (t, u, v) ∈ set d
        assume ylt: y < x
       have tuzer: t = 0 ∧ u = 0 ⟹ sign-num (t * y^2 + u * y + v) = sign-num
(t * x^2 + u * x + v)
         unfolding sign-num-def
         by auto
        have tunonzer: t ≠ 0 ∨ u ≠ 0  ⟹ sign-num (t * y^2 + u * y + v) =
sign-num (t * x^2 + u * x + v)
       proof −
         assume tuv-asm: t≠ 0 ∨ u ≠ 0
         have ¬(∃ q. q < ?srl ! 0 ∧ t * q^2 + u * q + v = 0)
         proof clarsimp
           fix q
           assume qlt: q < sorted-nonzero-root-list-set (set b ∪ set c ∪ set d) ! 0
           assume t * q^2 + u * q + v = 0
           then have qin: q ∈  {x. ∃(a, b, c)∈set b ∪ set c ∪ set d. (a ≠ 0 ∨ b
≠ 0) ∧ a * x^2 + b * x + c = 0}
             using insetunion tuv-asm by auto
           have set ?srl = nonzero-root-set (set b ∪ set c ∪ set d)
             unfolding sorted-nonzero-root-list-set-def
             using set-sorted-list-of-set[of nonzero-root-set (set b ∪ set c ∪ set d)]
               nonzero-root-set-finite[of (set b ∪ set c ∪ set d)]
             by auto
           then have q ∈ set ?srl using qin unfolding nonzero-root-set-def
             by auto
           then have lm: List.member ?srl q
             using in-set-member[of q ?srl]
             by auto
           then have  List.member
```

(*sorted-list-of-set* (*nonzero-root-set* (*set b* ∪ *set c* ∪ *set d*)))
q ⟹
q < *sorted-list-of-set* (*nonzero-root-set* (*set b* ∪ *set c* ∪ *set d*)) !
0 ⟹
(⋀*x xs*. (*x* ∈ *set xs*) = (∃ *i*<*length xs*. *xs* ! *i* = *x*)) ⟹
(⋀*x xs*. (*x* ∈ *set xs*) = *List.member xs x*) ⟹
(⋀*y x*. ¬ *y* ≤ *x* ⟹ *x* < *y*) ⟹
(⋀*xs*. *sorted xs* =
(∀ *i j*. *i* ≤ *j* ⟶ *j* < *length xs* ⟶ *xs* ! *i* ≤ *xs* ! *j*)) ⟹
(⋀*p*. *sorted-nonzero-root-list-set p* ≡
sorted-list-of-set (*nonzero-root-set p*)) ⟹
False
proof −
assume *a1*: *List.member* (*sorted-list-of-set* (*nonzero-root-set* (*set b* ∪ *set c* ∪ *set d*))) *q*
assume *a2*: *q* < *sorted-list-of-set* (*nonzero-root-set* (*set b* ∪ *set c* ∪ *set d*)) ! *0*
have *f3*: *List.member* (*sorted-list-of-set* {*r*. ∃ *p*. *p* ∈ *set b* ∪ *set c* ∪ *set d* ∧ (*case p of* (*ra*, *rb*, *rc*) ⇒ (*ra* ≠ *0* ∨ *rb* ≠ *0*) ∧ *ra* ∗ *r*^{2} + *rb* ∗ *r* + *rc* = *0*)}) *q*
using *a1* **by** (*metis nonzero-root-set-def*)
have *f4*: *q* < *sorted-list-of-set* {*r*. ∃ *p*. *p* ∈ *set b* ∪ *set c* ∪ *set d* ∧ (*case p of* (*ra*, *rb*, *rc*) ⇒ (*ra* ≠ *0* ∨ *rb* ≠ *0*) ∧ *ra* ∗ *r*^{2} + *rb* ∗ *r* + *rc* = *0*)} ! *0*
using *a2* **by** (*metis nonzero-root-set-def*)
have *f5*: *q* ∈ *set* (*sorted-list-of-set* {*r*. ∃ *p*. *p* ∈ *set b* ∪ *set c* ∪ *set d* ∧ (*case p of* (*ra*, *rb*, *rc*) ⇒ (*ra* ≠ *0* ∨ *rb* ≠ *0*) ∧ *ra* ∗ *r*^{2} + *rb* ∗ *r* + *rc* = *0*)})
using *f3* **by** (*meson in-set-member*)
have ∀ *rs r*. ∃ *n*. ((*r*::*real*) ∉ *set rs* ∨ *n* < *length rs*) ∧ (*r* ∉ *set rs* ∨ *rs* ! *n* = *r*)
by (*metis in-set-conv-nth*)
then obtain *nn* :: *real list* ⇒ *real* ⇒ *nat* **where**
f6: ⋀*r rs*. (*r* ∉ *set rs* ∨ *nn rs r* < *length rs*) ∧ (*r* ∉ *set rs* ∨ *rs* ! *nn rs r* = *r*)
by *moura*
then have *sorted-list-of-set* {*r*. ∃ *p*. *p* ∈ *set b* ∪ *set c* ∪ *set d* ∧ (*case p of* (*ra*, *rb*, *rc*) ⇒ (*ra* ≠ *0* ∨ *rb* ≠ *0*) ∧ *ra* ∗ *r*^{2} + *rb* ∗ *r* + *rc* = *0*)} ! *nn* (*sorted-list-of-set* {*r*. ∃ *p*. *p* ∈ *set b* ∪ *set c* ∪ *set d* ∧ (*case p of* (*ra*, *rb*, *rc*) ⇒ (*ra* ≠ *0* ∨ *rb* ≠ *0*) ∧ *ra* ∗ *r*^{2} + *rb* ∗ *r* + *rc* = *0*)}) *q* = *q*
using *f5* **by** *blast*
then have ⋀*n*. ¬ *sorted* (*sorted-list-of-set* {*r*. ∃ *p*. *p* ∈ *set b* ∪ *set c* ∪ *set d* ∧ (*case p of* (*ra*, *rb*, *rc*) ⇒ (*ra* ≠ *0* ∨ *rb* ≠ *0*) ∧ *ra* ∗ *r*^{2} + *rb* ∗ *r* + *rc* = *0*)}) ∨ ¬ *n* ≤ *nn* (*sorted-list-of-set* {*r*. ∃ *p*. *p* ∈ *set b* ∪ *set c* ∪ *set d* ∧ (*case p of* (*ra*, *rb*, *rc*) ⇒ (*ra* ≠ *0* ∨ *rb* ≠ *0*) ∧ *ra* ∗ *r*^{2} + *rb* ∗ *r* + *rc* = *0*)}) *q* ∨ ¬ *nn* (*sorted-list-of-set* {*r*. ∃ *p*. *p* ∈ *set b* ∪ *set c* ∪ *set d* ∧ (*case p of* (*ra*, *rb*, *rc*) ⇒ (*ra* ≠ *0* ∨ *rb* ≠ *0*) ∧ *ra* ∗ *r*^{2} + *rb* ∗ *r* + *rc* = *0*)}) *q* < *length* (*sorted-list-of-set* {*r*. ∃ *p*. *p* ∈ *set b* ∪ *set c* ∪ *set d* ∧ (*case p of* (*ra*, *rb*, *rc*) ⇒ (*ra* ≠ *0* ∨ *rb* ≠ *0*) ∧ *ra* ∗ *r*^{2} + *rb* ∗ *r* + *rc* = *0*)}) ∨ *sorted-list-of-set* {*r*. ∃ *p*. *p* ∈ *set b* ∪ *set c* ∪ *set d* ∧ (*case p of* (*ra*, *rb*, *rc*) ⇒ (*ra* ≠ *0* ∨ *rb* ≠ *0*) ∧ *ra* ∗ *r*^{2} + *rb* ∗ *r* + *rc* = *0*)} ! *n* ≤ *q*

using *not-less0 sorted-iff-nth-mono* **by** (*metis* (*lifting*))
then show *?thesis*
using *f6 f5 f4* **by** (*meson le0 not-less sorted-sorted-list-of-set*)
qed
then show *False* **using** *lm qlt in-set-conv-nth in-set-member not-le-imp-less not-less0 sorted-iff-nth-mono sorted-nonzero-root-list-set-def sorted-sorted-list-of-set*
by *auto*
qed
then have $\neg(\exists\, q.\ q \leq x \wedge t * q^2 + u * q + v = 0)$
using *xlt*
by *auto*
then show *sign-num* $(t * y^2 + u * y + v) =$ *sign-num* $(t * x^2 + u * x + v)$
using *ylt changes-sign-var*[*of t y u v x*]
by *blast*
qed
then show *sign-num* $(t * y^2 + u * y + v) =$ *sign-num* $(t * x^2 + u * x + v)$
using *tuzer*
by *blast*
qed
have *bseth*: $(\forall\,(a,\ b,\ c){\in}set\ b.\ \forall\, y{<}x.\ a * y^2 + b * y + c < 0)$
proof *clarsimp*
fix *t u v y*
assume *insetb*: $(t,\ u,\ v) \in set\ b$
assume *yltx*: $y < x$
have $(t,\ u,\ v) \in (set\ b \cup set\ c \cup set\ d)$ **using** *insetb*
by *auto*
then have *samesn*: *sign-num* $(t * y^2 + u * y + v) =$ *sign-num* $(t * x^2 + u * x + v)$
using *samesign insetb yltx*
by *blast*
have *sign-num* $(t{*}x\hat{}2 + u{*}x + v) = -1$
using *x-prop insetb* **unfolding** *sign-num-def*
by *auto*
then show $t * y^2 + u * y + v < 0$
using *samesn* **unfolding** *sign-num-def*
by (*metis add.right-inverse add.right-neutral linorder-neqE-linordered-idom one-add-one zero-neq-numeral*)
qed
have *bset*: $(\forall\,(a,\ b,\ c){\in}set\ b.\ \exists\, x.\ \forall\, y{<}x.\ a * y^2 + b * y + c < 0)$
proof *clarsimp*
fix *t u v*
assume *inset*: $(t,\ u,\ v) \in set\ b$
then have $\forall\, y{<}x.\ t * y^2 + u * y + v < 0$ **using** *bseth* **by** *auto*
then show $\exists\, x.\ \forall\, y{<}x.\ t * y^2 + u * y + v < 0$
by *auto*
qed
have *cseth*: $(\forall\,(a,\ b,\ c){\in}set\ c.\ \forall\, y{<}x.\ a * y^2 + b * y + c \leq 0)$

proof *clarsimp*
fix $t\ u\ v\ y$
assume *insetc*: $(t,\ u,\ v) \in set\ c$
assume *yltx*: $y < x$
have $(t,\ u,\ v) \in (set\ b \cup set\ c \cup set\ d)$ **using** *insetc*
by *auto*
then have *samesn*: $sign\text{-}num\ (t * y^2 + u * y + v) = sign\text{-}num\ (t * x^2 + u * x + v)$
using *samesign insetc yltx*
by *blast*
have $sign\text{-}num\ (t*x\hat{}2 + u*x + v) = -1 \vee sign\text{-}num\ (t*x\hat{}2 + u*x + v) = 0$
using *x-prop insetc* **unfolding** *sign-num-def*
by *auto*
then show $t * y^2 + u * y + v \leq 0$
using *samesn* **unfolding** *sign-num-def*
using *zero-neq-one* **by** *fastforce*
qed
have *cset*: $(\forall\,(a,\ b,\ c) \in set\ c.\ \exists\, x.\ \forall\, y<x.\ a * y^2 + b * y + c \leq 0)$
proof *clarsimp*
fix $t\ u\ v$
assume *inset*: $(t,\ u,\ v) \in set\ c$
then have $\forall\, y<x.\ t * y^2 + u * y + v \leq 0$ **using** *cseth* **by** *auto*
then show $\exists\, x.\ \forall\, y<x.\ t * y^2 + u * y + v \leq 0$
by *auto*
qed
have *dseth*: $(\forall\,(a,\ b,\ c) \in set\ d.\ \forall\, y<x.\ a * y^2 + b * y + c \neq 0)$
proof *clarsimp*
fix $t\ u\ v\ y$
assume *insetd*: $(t,\ u,\ v) \in set\ d$
assume *yltx*: $y < x$
assume *contrad*: $t * y^2 + u * y + v = 0$
have $(t,\ u,\ v) \in (set\ b \cup set\ c \cup set\ d)$ **using** *insetd*
by *auto*
then have *samesn*: $sign\text{-}num\ (t * y^2 + u * y + v) = sign\text{-}num\ (t * x^2 + u * x + v)$
using *samesign insetd yltx*
by *blast*
have $sign\text{-}num\ (t*x\hat{}2 + u*x + v) = -1 \vee sign\text{-}num\ (t*x\hat{}2 + u*x + v) = 1$
using *x-prop insetd* **unfolding** *sign-num-def*
by *auto*
then have $t * y^2 + u * y + v \neq 0$
using *samesn* **unfolding** *sign-num-def*
by *auto*
then show *False* **using** *contrad* **by** *auto*
qed
have *dset*: $(\forall\,(a,\ b,\ c) \in set\ d.\ \exists\, x.\ \forall\, y<x.\ a * y^2 + b * y + c \neq 0)$
proof *clarsimp*

fix $t\ u\ v$
assume *inset*: $(t,\ u,\ v) \in set\ d$
then have $\forall y{<}x.\ t * y^2 + u * y + v \neq 0$ **using** *dseth* **by** *auto*
then show $\exists x.\ \forall y{<}x.\ t * y^2 + u * y + v \neq 0$
by *auto*
qed
have $(\forall (a,\ b,\ c){\in}set\ a.\ a = 0 \wedge b = 0 \wedge c = 0)$
using *alleqsetvar* **by** *auto*
then have $((\forall (a,\ b,\ c){\in}set\ a.\ a = 0 \wedge b = 0 \wedge c = 0) \wedge$
$(\forall (a,\ b,\ c){\in}set\ b.\ \exists x.\ \forall y{<}x.\ a * y^2 + b * y + c < 0) \wedge$
$(\forall (a,\ b,\ c){\in}set\ c.\ \exists x.\ \forall y{<}x.\ a * y^2 + b * y + c \leq 0) \wedge$
$(\forall (a,\ b,\ c){\in}set\ d.\ \exists x.\ \forall y{<}x.\ a * y^2 + b * y + c \neq 0))$
using *bset cset dset* **by** *auto*
then show *False* **using** *f1* **by** *auto*
qed

have *cases-gt*: $x > ?srl\ !\ (length\ ?srl - 1) \Longrightarrow False$
proof −
assume *xgt*: $x > ?srl\ !\ (length\ ?srl - 1)$
let *?bgrt* $= ?srl\ !\ (length\ ?srl - 1)$
have *samesign*: $\forall\ (a,\ b,\ c) \in (set\ b \cup set\ c \cup set\ d).$
$(\forall y > ?bgrt.\ sign\text{-}num\ (a * y^2 + b * y + c) = sign\text{-}num\ (a{*}x\hat{}2 + b{*}x + c))$
proof *clarsimp*
fix $t\ u\ v\ y$
assume *insetunion*: $(t,\ u,\ v) \in set\ b \vee (t,\ u,\ v) \in set\ c \vee (t,\ u,\ v) \in set\ d$
assume *ygt*: $sorted\text{-}nonzero\text{-}root\text{-}list\text{-}set\ (set\ b \cup set\ c \cup set\ d)\ !$
$(length\ (sorted\text{-}nonzero\text{-}root\text{-}list\text{-}set\ (set\ b \cup set\ c \cup set\ d)) - Suc\ 0) < y$
have *tuzer*: $t = 0 \wedge u = 0 \Longrightarrow sign\text{-}num\ (t * y^2 + u * y + v) = sign\text{-}num\ (t * x^2 + u * x + v)$
unfolding *sign-num-def*
by *auto*
have *tunonzer*: $t \neq 0 \vee u \neq 0 \Longrightarrow sign\text{-}num\ (t * y^2 + u * y + v) = sign\text{-}num\ (t * x^2 + u * x + v)$
proof −
assume *tuv-asm*: $t{\neq}\ 0 \vee u \neq 0$
have $\neg(\exists q.\ q > ?srl\ !\ (length\ ?srl - 1) \wedge t * q^2 + u * q + v = 0)$
proof *clarsimp*
fix q
assume *qgt*: $sorted\text{-}nonzero\text{-}root\text{-}list\text{-}set\ (set\ b \cup set\ c \cup set\ d)\ !$
$(length\ (sorted\text{-}nonzero\text{-}root\text{-}list\text{-}set\ (set\ b \cup set\ c \cup set\ d)) - Suc\ 0) < q$
assume $t * q^2 + u * q + v = 0$
then have *qin*: $q \in \{x.\ \exists (a,\ b,\ c){\in}set\ b \cup set\ c \cup set\ d.\ (a \neq 0 \vee b \neq 0) \wedge a * x^2 + b * x + c = 0\}$
using *insetunion tuv-asm* **by** *auto*
have $set\ ?srl = nonzero\text{-}root\text{-}set\ (set\ b \cup set\ c \cup set\ d)$
unfolding *sorted-nonzero-root-list-set-def*

using *set-sorted-list-of-set*[*of nonzero-root-set* (*set b* ∪ *set c* ∪ *set d*)]
nonzero-root-set-finite[*of* (*set b* ∪ *set c* ∪ *set d*)]
by *auto*
then have $q \in$ *set ?srl* **using** *qin* **unfolding** *nonzero-root-set-def*
by *auto*
then have *List.member ?srl q*
using *in-set-member*[*of q ?srl*]
by *auto*
then show *False* **using** *qgt in-set-conv-nth in-set-member not-le-imp-less not-less0 sorted-iff-nth-mono sorted-nonzero-root-list-set-def sorted-sorted-list-of-set*
by (*smt* (*z3*) *Suc-diff-Suc Suc-n-not-le-n* ‹$q \in$ *set* (*sorted-nonzero-root-list-set* (*set b* ∪ *set c* ∪ *set d*))› *in-set-conv-nth length-0-conv length-greater-0-conv length-sorted-list-of-set lenzero less-Suc-eq-le minus-nat.diff-0 not-le sorted-nth-mono sorted-sorted-list-of-set*)

qed
then have *nor*: $\neg(\exists\, q.\ q > ?bgrt \wedge t * q^2 + u * q + v = 0)$
using *xgt*
by *auto*
have *c1*: $x > y \Longrightarrow$ *sign-num* $(t * y^2 + u * y + v) =$ *sign-num* $(t * x^2 + u * x + v)$
using *nor changes-sign-var*[*of t y u v x*] *xgt ygt*
by *fastforce*
then have *c2*: $y > x \Longrightarrow$ *sign-num* $(t * y^2 + u * y + v) =$ *sign-num* $(t * x^2 + u * x + v)$
using *nor changes-sign-var*[*of t x u v y*] *xgt ygt*
by *force*
then have *c3*: $x = y \Longrightarrow$ *sign-num* $(t * y^2 + u * y + v) =$ *sign-num* $(t * x^2 + u * x + v)$
unfolding *sign-num-def*
by *auto*
then show *sign-num* $(t * y^2 + u * y + v) =$ *sign-num* $(t * x^2 + u * x + v)$
using *c1 c2 c3*
by *linarith*
qed
then show *sign-num* $(t * y^2 + u * y + v) =$ *sign-num* $(t * x^2 + u * x + v)$
using *tuzer*
by *blast*
qed

have $(\forall\, (a,\, b,\, c) \in$ *set a*. $a = 0 \wedge b = 0 \wedge c = 0)$
using *alleqsetvar* **by** *auto*
have *?bgrt* $\in$ *set ?srl*
using *set-sorted-list-of-set nonzero-root-set-finite in-set-member*
using *asm* **by** *auto*
then have *?bgrt* $\in$ *nonzero-root-set* (*set b* ∪ *set c* ∪ *set d*)
unfolding *sorted-nonzero-root-list-set-def*
using *set-sorted-list-of-set nonzero-root-set-finite*

by *auto*
then have $\exists\, t\ u\ v.\ (t,\ u,\ v) \in set\ b \cup set\ c \cup set\ d \wedge (t \neq 0 \vee u \neq 0) \wedge (t * ?bgrt^2 + u * ?bgrt + v = 0)$
unfolding *nonzero-root-set-def* **by** *auto*
then obtain *t u v* **where** *tuvprop1*: $(t,\ u,\ v) \in set\ b \cup set\ c \cup set\ d \wedge (t \neq 0 \vee u \neq 0) \wedge (t * ?bgrt^2 + u * ?bgrt + v = 0)$
by *auto*
then have *tuvprop*: $((t,\ u,\ v) \in set\ b \wedge (t \neq 0 \vee u \neq 0) \wedge (t * ?bgrt^2 + u * ?bgrt + v = 0))$
$\vee\ ((t,\ u,\ v) \in set\ c \wedge (t \neq 0 \vee u \neq 0) \wedge (t * ?bgrt^2 + u * ?bgrt + v = 0)) \vee$
$((t,\ u,\ v) \in set\ d \wedge (t \neq 0 \vee u \neq 0) \wedge (t * ?bgrt^2 + u * ?bgrt + v = 0))$
by *auto*
have *tnonz*: $t \neq 0 \Longrightarrow (-1 * u\hat{}2 + 4 * t * v \leq 0\ \wedge\ (?bgrt = (-u + 1 * sqrt\ (u\hat{}2 - 4 * t * v)) / (2 * t) \vee ?bgrt = (-u + -1 * sqrt\ (u\hat{}2 - 4 * t * v)) / (2 * t)))$
proof −
assume $t \neq 0$
have $-1 * u\hat{}2 + 4 * t * v \leq 0$ **using** *tuvprop1 discriminant-negative*[*of t u v*]
unfolding *discrim-def*
using ‹$t \neq 0$› **by** *force*
then show *?thesis*
using *tuvprop discriminant-nonneg*[*of t u v*]
unfolding *discrim-def*
using ‹$t \neq 0$› **by** *auto*
qed
have *unonz*: $(t = 0 \wedge u \neq 0) \Longrightarrow ?bgrt = -v / u$
proof −
assume $(t = 0 \wedge u \neq 0)$
then have $u * ?bgrt + v = 0$ **using** *tuvprop1*
by *simp*
then show $?bgrt = -v / u$
by (*simp add*: ‹$t = 0 \wedge u \neq 0$› *eq-minus-divide-eq mult.commute*)
qed

have *allpropb*: $(\forall (d,\ e,\ f) \in set\ b.\ \forall y' > ?bgrt.\ \forall x \in \{?bgrt <..y'\}.\ d * x^2 + e * x + f < 0)$
proof *clarsimp*
fix *t1 u1 v1 y1 x1*
assume *ins*: $(t1,\ u1,\ v1) \in set\ b$
assume *x1gt*: *sorted-nonzero-root-list-set* $(set\ b \cup set\ c \cup set\ d)$! $(length\ (sorted\text{-}nonzero\text{-}root\text{-}list\text{-}set\ (set\ b \cup set\ c \cup set\ d)) - Suc\ 0) < x1$
assume $x1 \leq y1$
have *xsn*: *sign-num* $(t1 * x\hat{}2 + u1 * x + v1\ ) = -1$ **using** *ins x-prop* **unfolding** *sign-num-def*
by *auto*
have *sign-num* $(t1 * x1^2 + u1 * x1 + v1\ ) = sign\text{-}num\ (t1 * x\hat{}2 + u1$

$*\ x + v1\ )$
using *ins x1gt samesign*
apply (*auto*)
by *blast*
then show $t1 * x1^2 + u1 * x1 + v1 < 0$ **using** *xsn* **unfolding** *sign-num-def*
by (*metis add.right-inverse add.right-neutral linorder-neqE-linordered-idom one-add-one zero-neq-numeral*)
qed
have *allpropbvar*: $(\forall\,(d,\ e,\ f){\in}set\ b.$
$\exists\,y'{>}?bgrt.\ \forall\,x{\in}\{\,?bgrt{<}..y'\}.\ d * x^2 + e * x + f < 0)$
proof *clarsimp*
fix *t1 u1 v1*
assume $(t1,\ u1,\ v1) \in set\ b$
then have $\forall\,x{\in}\{\,?bgrt{<}..(?bgrt + 1)\}.\ t1 * x^2 + u1 * x + v1 < 0$
using *allpropb*
by *force*
then show $\exists\,y'{>}sorted\text{-}nonzero\text{-}root\text{-}list\text{-}set\ (set\ b \cup set\ c \cup set\ d)\ !$
$(length\ (sorted\text{-}nonzero\text{-}root\text{-}list\text{-}set\ (set\ b \cup set\ c \cup set\ d)) - Suc\ 0).$
$\forall\,x{\in}\{sorted\text{-}nonzero\text{-}root\text{-}list\text{-}set\ (set\ b \cup set\ c \cup set\ d)\ !$
$(length\ (sorted\text{-}nonzero\text{-}root\text{-}list\text{-}set\ (set\ b \cup set\ c \cup set\ d)) - Suc\ 0){<}..y'\}.$
$t1 * x^2 + u1 * x + v1 < 0$
using *less-add-one*
by (*metis One-nat-def*)
qed
have *allpropc*: $(\forall\,(d,\ e,\ f){\in}set\ c.$
$\forall\,y'{>}?bgrt.\ \forall\,x{\in}\{\,?bgrt{<}..y'\}.\ d * x^2 + e * x + f \leq 0)$
proof *clarsimp*
fix *t1 u1 v1 y1 x1*
assume *ins*: $(t1,\ u1,\ v1) \in set\ c$
assume *x1gt*: $sorted\text{-}nonzero\text{-}root\text{-}list\text{-}set\ (set\ b \cup set\ c \cup set\ d)\ !$
$(length\ (sorted\text{-}nonzero\text{-}root\text{-}list\text{-}set\ (set\ b \cup set\ c \cup set\ d)) - Suc\ 0) < x1$
assume $x1 \leq y1$
have *xsn*: $sign\text{-}num\ (t1 * x\hat{}2 + u1 * x + v1\ ) = -1 \vee sign\text{-}num\ (t1 * x\hat{}2 + u1 * x + v1\ ) = 0$ **using** *ins x-prop* **unfolding** *sign-num-def*
by *auto*
have $sign\text{-}num\ (t1 * x1^2 + u1 * x1 + v1\ ) = sign\text{-}num\ (t1 * x\hat{}2 + u1 * x + v1\ )$
using *ins x1gt samesign One-nat-def*
proof −
have $case\ (t1,\ u1,\ v1)\ of\ (r,\ ra,\ rb) \Rightarrow \forall\,raa{>}sorted\text{-}nonzero\text{-}root\text{-}list\text{-}set\ (set\ b \cup set\ c \cup set\ d)\ !\ (length\ (sorted\text{-}nonzero\text{-}root\text{-}list\text{-}set\ (set\ b \cup set\ c \cup set\ d)) - 1).\ sign\text{-}num\ (r * raa^2 + ra * raa + rb) = sign\text{-}num\ (r * x^2 + ra * x + rb)$
by (*smt* (*z3*) *Un-iff ins samesign*)
then show *?thesis*
by (*simp add*: *x1gt*)
qed
then show $t1 * x1^2 + u1 * x1 + v1 \leq 0$ **using** *xsn* **unfolding** *sign-num-def*
by (*metis equal-neg-zero less-numeral-extra*(*3*) *linorder-not-less zero-neq-one*)

qed
have *allpropcvar*: $(\forall\,(d,\ e,\ f){\in}set\ c.$
$\exists\, y'{>}?bgrt.\ \forall\, x{\in}\{\,?bgrt{<}..y'\}.\ d * x^2 + e * x + f \leq 0)$
proof *clarsimp*
fix *t1 u1 v1*
assume $(t1,\ u1,\ v1) \in set\ c$
then have $\forall\, x{\in}\{\,?bgrt{<}..(?bgrt + 1)\}.\ t1 * x^2 + u1 * x + v1 \leq 0$
using *allpropc*
by *force*
then show $\exists\, y'{>}$*sorted-nonzero-root-list-set* $(set\ b \cup set\ c \cup set\ d)$!
$(length\ (sorted\text{-}nonzero\text{-}root\text{-}list\text{-}set\ (set\ b \cup set\ c \cup set\ d)) - Suc\ 0).$
$\forall\, x{\in}\{sorted\text{-}nonzero\text{-}root\text{-}list\text{-}set\ (set\ b \cup set\ c \cup set\ d)$!
$(length\ (sorted\text{-}nonzero\text{-}root\text{-}list\text{-}set\ (set\ b \cup set\ c \cup set\ d)) - Suc\ 0){<}..y'\}.$
$t1 * x^2 + u1 * x + v1 \leq 0$
using *less-add-one One-nat-def*
by *metis*
qed
have *allpropd*: $(\forall\,(d,\ e,\ f){\in}set\ d.$
$\forall\, y'{>}?bgrt.\ \forall\, x{\in}\{\,?bgrt{<}..y'\}.\ d * x^2 + e * x + f \neq 0)$
proof *clarsimp*
fix *t1 u1 v1 y1 x1*
assume *ins*: $(t1,\ u1,\ v1) \in set\ d$
assume *contrad*:$t1 * x1^2 + u1 * x1 + v1 = 0$
assume *x1gt*: $sorted\text{-}nonzero\text{-}root\text{-}list\text{-}set\ (set\ b \cup set\ c \cup set\ d)$!
$(length\ (sorted\text{-}nonzero\text{-}root\text{-}list\text{-}set\ (set\ b \cup set\ c \cup set\ d)) - Suc\ 0) < x1$
assume $x1 \leq y1$
have *xsn*: $sign\text{-}num\ (t1 * x\hat{}2 + u1 * x + v1\ ) = -1 \vee sign\text{-}num\ (t1 * x\hat{}2 + u1 * x + v1\ ) = 1$ **using** *ins x-prop* **unfolding** *sign-num-def*
by *auto*
have $sign\text{-}num\ (t1 * x1^2 + u1 * x1 + v1\ ) = sign\text{-}num\ (t1 * x\hat{}2 + u1 * x + v1\ )$
using *ins x1gt samesign* **apply** (*auto*)
by *blast*
then have $t1 * x1^2 + u1 * x1 + v1 \neq 0$ **using** *xsn* **unfolding** *sign-num-def*
by *auto*
then show *False* **using** *contrad* **by** *auto*
qed
have *allpropdvar*: $(\forall\,(d,\ e,\ f){\in}set\ d.$
$\exists\, y'{>}?bgrt.\ \forall\, x{\in}\{\,?bgrt{<}..y'\}.\ d * x^2 + e * x + f \neq 0)$
proof *clarsimp*
fix *t1 u1 v1*
assume $(t1,\ u1,\ v1) \in set\ d$
then have $\forall\, x{\in}\{\,?bgrt{<}..(?bgrt + 1)\}.\ t1 * x^2 + u1 * x + v1 \neq 0$
using *allpropd*
by *force*
then show $\exists\, y'{>}$*sorted-nonzero-root-list-set* $(set\ b \cup set\ c \cup set\ d)$!
$(length\ (sorted\text{-}nonzero\text{-}root\text{-}list\text{-}set\ (set\ b \cup set\ c \cup set\ d)) - Suc\ 0).$

$\forall x{\in}\{sorted\text{-}nonzero\text{-}root\text{-}list\text{-}set\ (set\ b \cup set\ c \cup set\ d)\ !$
$(length\ (sorted\text{-}nonzero\text{-}root\text{-}list\text{-}set\ (set\ b \cup set\ c \cup set\ d)) - Suc\ 0){<}..y'\}.$
$t1 * x^2 + u1 * x + v1 \neq 0$
using *less-add-one* **by** *force*
qed
have $\forall x.\ (\forall (d,\ e,\ f){\in}set\ a.\ d * x^2 + e * x + f = 0)$ **using** *alleqsetvar*
by *auto*
then have *ast*: $(\forall (d,\ e,\ f){\in}set\ a.\ \forall x{\in}\{?bgrt{<}..(?bgrt + 1)\}.\ d * x^2 + e * x + f = 0)$
by *auto*
have *allpropavar*: $(\forall (d,\ e,\ f){\in}set\ a.\ \exists y'{>}?bgrt.\ \forall x{\in}\{?bgrt{<}..y'\}.\ d * x^2 + e * x + f = 0)$
proof *clarsimp*
fix *t1 u1 v1*
assume $(t1,\ u1,\ v1) \in set\ a$
then have $\forall x{\in}\{?bgrt{<}..(?bgrt + 1)\}.\ t1 * x^2 + u1 * x + v1 = 0$
using *ast* **by** *auto*
then show $\exists y'{>}sorted\text{-}nonzero\text{-}root\text{-}list\text{-}set\ (set\ b \cup set\ c \cup set\ d)\ !$
$(length\ (sorted\text{-}nonzero\text{-}root\text{-}list\text{-}set\ (set\ b \cup set\ c \cup set\ d)) - Suc\ 0).$
$\forall x{\in}\{sorted\text{-}nonzero\text{-}root\text{-}list\text{-}set\ (set\ b \cup set\ c \cup set\ d)\ !$
$(length\ (sorted\text{-}nonzero\text{-}root\text{-}list\text{-}set\ (set\ b \cup set\ c \cup set\ d)) - Suc\ 0){<}..y'\}.$
$t1 * x^2 + u1 * x + v1 = 0$
using *less-add-one One-nat-def*
by *metis*
qed
have *quadsetb*: $((t,\ u,\ v) \in set\ b \wedge t{\neq}\ 0) \Longrightarrow False$
proof −
assume *asm*: $(t,\ u,\ v) \in set\ b \wedge t{\neq}\ 0$
have *bgrt1*: $(?bgrt = (-\ u + 1 * sqrt\ (u\hat{}2 - 4 * t * v)) / (2 * t)) \Longrightarrow False$
proof −
assume *bgrtis*: $?bgrt = (-\ u + 1 * sqrt\ (u\hat{}2 - 4 * t * v)) / (2 * t)$
have *discrim-prop*: $-1{*}u\hat{}2 + 4 * t * v \leq 0$ **using** *asm tnonz*
using ‹$sorted\text{-}nonzero\text{-}root\text{-}list\text{-}set\ (set\ b \cup set\ c \cup set\ d)\ !\ (length\ (sorted\text{-}nonzero\text{-}root\text{-}list\text{-}set\ (set\ b \cup set\ c \cup set\ d)) - 1) = (-\ u + 1 * sqrt\ (u^2 - 4 * t * v)) / (2 * t)$›
by *auto*
have $((t,\ u,\ v){\in}set\ b \wedge t \neq 0 \wedge -1{*}u\hat{}2 + 4 * t * v \leq 0 \wedge$
$((\forall (d,\ e,\ f){\in}set\ a.\ \exists y'{>}?bgrt.\ \forall x{\in}\{?bgrt{<}..y'\}.\ d * x^2 + e * x + f = 0) \wedge$
$(\forall (d,\ e,\ f){\in}set\ b.\ \exists y'{>}?bgrt.\ \forall x{\in}\{?bgrt{<}..y'\}.\ d * x^2 + e * x + f < 0) \wedge$
$(\forall (d,\ e,\ f){\in}set\ c.\ \exists y'{>}?bgrt.\ \forall x{\in}\{?bgrt{<}..y'\}.\ d * x^2 + e * x + f \leq 0) \wedge$
$(\forall (d,\ e,\ f){\in}set\ d.\ \exists y'{>}?bgrt.\ \forall x{\in}\{?bgrt{<}..y'\}.\ d * x^2 + e * x + f \neq 0)))$

using *asm discrim-prop allpropavar allpropbvar allpropcvar allpropdvar*
by *linarith*
then show *False* **using** *f6 bgrtis*
by *auto*
qed
have *bgrt2*: $(?bgrt = (-\ u + -1 * sqrt\ (u\hat{}2 - 4 * t * v)) / (2 * t)) \Longrightarrow$ *False*
proof −
assume *bgrtis*: $?bgrt = (-\ u + -1 * sqrt\ (u\hat{}2 - 4 * t * v)) / (2 * t)$
have *discrim-prop*: $-1 * u\hat{}2 + 4 * t * v \leq 0$ **using** *asm tnonz*
using ‹*sorted-nonzero-root-list-set* (*set b* ∪ *set c* ∪ *set d*) ! (*length* (*sorted-nonzero-root-list-set* (*set b* ∪ *set c* ∪ *set d*)) − *1*) = (− *u* + −*1* ∗ *sqrt* (u^2 − *4* ∗ *t* ∗ *v*)) / (*2* ∗ *t*)›
by *auto*
have $((t,\ u,\ v) \in set\ b \wedge t \neq 0 \wedge -1 * u\hat{}2 + 4 * t * v \leq 0 \wedge$
$((\forall (d,\ e,\ f) \in set\ a.$
$\exists y' > ?bgrt.\ \forall x \in \{?bgrt < .. y'\}.\ d * x^2 + e * x + f = 0) \wedge$
$(\forall (d,\ e,\ f) \in set\ b.$
$\exists y' > ?bgrt.\ \forall x \in \{?bgrt < .. y'\}.\ d * x^2 + e * x + f < 0) \wedge$
$(\forall (d,\ e,\ f) \in set\ c.$
$\exists y' > ?bgrt.\ \forall x \in \{?bgrt < .. y'\}.\ d * x^2 + e * x + f \leq 0) \wedge$
$(\forall (d,\ e,\ f) \in set\ d.$
$\exists y' > ?bgrt.\ \forall x \in \{?bgrt < .. y'\}.\ d * x^2 + e * x + f \neq 0)))$
using *asm discrim-prop allpropavar allpropbvar allpropcvar allpropdvar*
by *linarith*
then show *False* **using** *f7 bgrtis*
by *auto*
qed
show *False* **using** *tnonz bgrt1 bgrt2 asm*
by *auto*
qed
have *linsetb*: $((t,\ u,\ v) \in set\ b \wedge (t = 0 \wedge u \neq 0)) \Longrightarrow$ *False*
proof −
assume *asm*: $(t,\ u,\ v) \in set\ b \wedge (t = 0 \wedge u \neq 0)$
then have *bgrtis*: $?bgrt = (-\ v\ /\ u)$
using *unonz*
by *blast*
have $((t,\ u,\ v) \in set\ b \wedge (t = 0 \wedge u \neq 0) \wedge$
$((\forall (d,\ e,\ f) \in set\ a.$
$\exists y' > ?bgrt.\ \forall x \in \{?bgrt < .. y'\}.\ d * x^2 + e * x + f = 0) \wedge$
$(\forall (d,\ e,\ f) \in set\ b.$
$\exists y' > ?bgrt.\ \forall x \in \{?bgrt < .. y'\}.\ d * x^2 + e * x + f < 0) \wedge$
$(\forall (d,\ e,\ f) \in set\ c.$
$\exists y' > ?bgrt.\ \forall x \in \{?bgrt < .. y'\}.\ d * x^2 + e * x + f \leq 0) \wedge$
$(\forall (d,\ e,\ f) \in set\ d.$
$\exists y' > ?bgrt.\ \forall x \in \{?bgrt < .. y'\}.\ d * x^2 + e * x + f \neq 0)))$
using *asm allpropavar allpropbvar allpropcvar allpropdvar*
by *linarith*
then show *False* **using** *bgrtis f5*

by *auto*
qed
have *insetb*: $((t, u, v) \in set\ b \wedge (t \neq 0 \vee u \neq 0) \wedge (t * ?bgrt^2 + u * ?bgrt + v = 0)) \Longrightarrow False$
using *quadsetb linsetb* **by** *auto*
have *quadsetc*: $(t, u, v) \in set\ c \wedge t \neq 0 \Longrightarrow False$
proof −
assume *asm*: $(t, u, v) \in set\ c \wedge t \neq 0$
have *bgrt1*: $(?bgrt = (-u + 1 * sqrt\ (u\hat{}2 - 4 * t * v)) / (2 * t)) \Longrightarrow False$
proof −
assume *bgrtis*: $?bgrt = (-u + 1 * sqrt\ (u\hat{}2 - 4 * t * v)) / (2 * t)$
have *discrim-prop*: $-1 * u\hat{}2 + 4 * t * v \leq 0$ **using** *asm tnonz*
using ‹*sorted-nonzero-root-list-set* $(set\ b \cup set\ c \cup set\ d)$! $(length\ (sorted\text{-}nonzero\text{-}root\text{-}list\text{-}set\ (set\ b \cup set\ c \cup set\ d)) - 1) = (-u + 1 * sqrt\ (u^2 - 4 * t * v)) / (2 * t)$›
by *auto*
have $((t, u, v) \in set\ c \wedge t \neq 0 \wedge -1 * u\hat{}2 + 4 * t * v \leq 0 \wedge$
$((\forall (d, e, f) \in set\ a.$
$\exists y' > ?bgrt.\ \forall x \in \{?bgrt < .. y'\}.\ d * x^2 + e * x + f = 0) \wedge$
$(\forall (d, e, f) \in set\ b.$
$\exists y' > ?bgrt.\ \forall x \in \{?bgrt < .. y'\}.\ d * x^2 + e * x + f < 0) \wedge$
$(\forall (d, e, f) \in set\ c.$
$\exists y' > ?bgrt.\ \forall x \in \{?bgrt < .. y'\}.\ d * x^2 + e * x + f \leq 0) \wedge$
$(\forall (d, e, f) \in set\ d.$
$\exists y' > ?bgrt.\ \forall x \in \{?bgrt < .. y'\}.\ d * x^2 + e * x + f \neq 0)))$
using *asm discrim-prop allpropavar allpropbvar allpropcvar allpropdvar*
by *linarith*
then show *False* **using** *f13a bgrtis*
by *auto*
qed
have *bgrt2*: $(?bgrt = (-u + -1 * sqrt\ (u\hat{}2 - 4 * t * v)) / (2 * t)) \Longrightarrow False$
proof −
assume *bgrtis*: $?bgrt = (-u + -1 * sqrt\ (u\hat{}2 - 4 * t * v)) / (2 * t)$
have *discrim-prop*: $-1 * u\hat{}2 + 4 * t * v \leq 0$ **using** *asm tnonz*
using ‹*sorted-nonzero-root-list-set* $(set\ b \cup set\ c \cup set\ d)$! $(length\ (sorted\text{-}nonzero\text{-}root\text{-}list\text{-}set\ (set\ b \cup set\ c \cup set\ d)) - 1) = (-u + -1 * sqrt\ (u^2 - 4 * t * v)) / (2 * t)$›
by *auto*
have $((t, u, v) \in set\ c \wedge t \neq 0 \wedge -1 * u\hat{}2 + 4 * t * v \leq 0 \wedge$
$((\forall (d, e, f) \in set\ a.$
$\exists y' > ?bgrt.\ \forall x \in \{?bgrt < .. y'\}.\ d * x^2 + e * x + f = 0) \wedge$
$(\forall (d, e, f) \in set\ b.$
$\exists y' > ?bgrt.\ \forall x \in \{?bgrt < .. y'\}.\ d * x^2 + e * x + f < 0) \wedge$
$(\forall (d, e, f) \in set\ c.$
$\exists y' > ?bgrt.\ \forall x \in \{?bgrt < .. y'\}.\ d * x^2 + e * x + f \leq 0) \wedge$
$(\forall (d, e, f) \in set\ d.$
$\exists y' > ?bgrt.\ \forall x \in \{?bgrt < .. y'\}.\ d * x^2 + e * x + f \neq 0)))$

using *asm discrim-prop allpropavar allpropbvar allpropcvar allpropdvar*
by *linarith*
then show *False* **using** *f9a bgrtis*
by *auto*
qed
show *False* **using** *tnonz bgrt1 bgrt2 asm*
by *auto*
qed
have *linsetc*: $(t, u, v) \in set\ c \wedge (t = 0 \wedge u \neq 0) \Longrightarrow False$
proof −
assume *asm*: $(t, u, v) \in set\ c \wedge (t = 0 \wedge u \neq 0)$
then have *bgrtis*: $?bgrt = (-\ v\ /\ u)$
using *unonz*
by *blast*
have $((t, u, v) \in set\ c \wedge (t = 0 \wedge u \neq 0) \wedge$
$((\forall (d, e, f) \in set\ a.$
$\exists y' > ?bgrt.\ \forall x \in \{?bgrt < .. y'\}.\ d * x^2 + e * x + f = 0) \wedge$
$(\forall (d, e, f) \in set\ b.$
$\exists y' > ?bgrt.\ \forall x \in \{?bgrt < .. y'\}.\ d * x^2 + e * x + f < 0) \wedge$
$(\forall (d, e, f) \in set\ c.$
$\exists y' > ?bgrt.\ \forall x \in \{?bgrt < .. y'\}.\ d * x^2 + e * x + f \leq 0) \wedge$
$(\forall (d, e, f) \in set\ d.$
$\exists y' > ?bgrt.\ \forall x \in \{?bgrt < .. y'\}.\ d * x^2 + e * x + f \neq 0)))$
using *asm allpropavar allpropbvar allpropcvar allpropdvar*
by *linarith*
then show *False* **using** *bgrtis f8a*
by *auto*
qed
have *insetc*: $((t, u, v) \in set\ c \wedge (t \neq 0 \vee u \neq 0) \wedge (t * ?bgrt^2 + u * ?bgrt + v = 0)) \Longrightarrow False$
using *quadsetc linsetc* **by** *auto*
have *quadsetd*: $(t, u, v) \in set\ d \wedge t \neq 0 \Longrightarrow False$
proof −
assume *asm*: $(t, u, v) \in set\ d \wedge t \neq 0$
have *bgrt1*: $(?bgrt = (-\ u + 1 * sqrt\ (u\hat{}2 - 4 * t * v))\ /\ (2 * t)) \Longrightarrow False$
proof −
assume *bgrtis*: $?bgrt = (-\ u + 1 * sqrt\ (u\hat{}2 - 4 * t * v))\ /\ (2 * t)$
have *discrim-prop*: $-1 * u\hat{}2 + 4 * t * v \leq 0$ **using** *asm tnonz*
using ‹*sorted-nonzero-root-list-set* $(set\ b \cup set\ c \cup set\ d)\ !\ (length\ (sorted\text{-}nonzero\text{-}root\text{-}list\text{-}set\ (set\ b \cup set\ c \cup set\ d)) - 1) = (-\ u + 1 * sqrt\ (u^2 - 4 * t * v))\ /\ (2 * t)$›
by *auto*
have $((t, u, v) \in set\ d \wedge t \neq 0 \wedge -1 * u\hat{}2 + 4 * t * v \leq 0 \wedge$
$((\forall (d, e, f) \in set\ a.$
$\exists y' > ?bgrt.\ \forall x \in \{?bgrt < .. y'\}.\ d * x^2 + e * x + f = 0) \wedge$
$(\forall (d, e, f) \in set\ b.$
$\exists y' > ?bgrt.\ \forall x \in \{?bgrt < .. y'\}.\ d * x^2 + e * x + f < 0) \wedge$
$(\forall (d, e, f) \in set\ c.$

$\exists\, y'{>}?bgrt.\ \forall\, x{\in}\{\, ?bgrt{<}..y'\}.\ d * x^2 + e * x + f \leq 0) \wedge$
$(\forall\, (d,\ e,\ f){\in}set\ d.$
$\exists\, y'{>}?bgrt.\ \forall\, x{\in}\{\, ?bgrt{<}..y'\}.\ d * x^2 + e * x + f \neq 0)))$
using *asm discrim-prop allpropavar allpropbvar allpropcvar allpropdvar*
by *linarith*
then show *False* **using** *f11 bgrtis*
by *auto*
qed
have *bgrt2*: $(?bgrt = (-\ u + -1 * sqrt\ (u\hat{}2 - 4 * t * v)) / (2 * t)) \Longrightarrow$ *False*
proof −
assume *bgrtis*: $?bgrt = (-\ u + -1 * sqrt\ (u\hat{}2 - 4 * t * v)) / (2 * t)$
have *discrim-prop*: $-1{*}u\hat{}2 + 4 * t * v \leq 0$ **using** *asm tnonz*
using ‹*sorted-nonzero-root-list-set* $(set\ b \cup set\ c \cup set\ d)$! $(length\ (sorted\text{-}nonzero\text{-}root\text{-}list\text{-}set\ (set\ b \cup set\ c \cup set\ d)) - 1) = (-\ u + -1 * sqrt\ (u^2 - 4 * t * v)) / (2 * t)$›
by *auto*
have $((t,\ u,\ v){\in}set\ d \wedge t \neq 0 \wedge -\ 1{*}u\hat{}2 + 4 * t * v \leq 0 \wedge$
$((\forall\, (d,\ e,\ f){\in}set\ a.$
$\exists\, y'{>}?bgrt.\ \forall\, x{\in}\{\, ?bgrt{<}..y'\}.\ d * x^2 + e * x + f = 0) \wedge$
$(\forall\, (d,\ e,\ f){\in}set\ b.$
$\exists\, y'{>}?bgrt.\ \forall\, x{\in}\{\, ?bgrt{<}..y'\}.\ d * x^2 + e * x + f < 0) \wedge$
$(\forall\, (d,\ e,\ f){\in}set\ c.$
$\exists\, y'{>}?bgrt.\ \forall\, x{\in}\{\, ?bgrt{<}..y'\}.\ d * x^2 + e * x + f \leq 0) \wedge$
$(\forall\, (d,\ e,\ f){\in}set\ d.$
$\exists\, y'{>}?bgrt.\ \forall\, x{\in}\{\, ?bgrt{<}..y'\}.\ d * x^2 + e * x + f \neq 0)))$
using *asm discrim-prop allpropavar allpropbvar allpropcvar allpropdvar*
by *linarith*
then show *False* **using** *f12 bgrtis*
by *auto*
qed
show *False* **using** *tnonz bgrt1 bgrt2 asm*
by *auto*
qed
have *linsetd*: $(t,\ u,\ v) \in set\ d \wedge (t = 0 \wedge u \neq 0) \Longrightarrow$ *False*
proof −
assume *asm*: $(t,\ u,\ v) \in set\ d \wedge (t = 0 \wedge u \neq 0)$
then have *bgrtis*: $?bgrt = (-\ v\ /\ u)$
using *unonz*
by *blast*
have $((t,\ u,\ v){\in}set\ d \wedge (t = 0 \wedge u \neq 0) \wedge$
$((\forall\, (d,\ e,\ f){\in}set\ a.$
$\exists\, y'{>}?bgrt.\ \forall\, x{\in}\{\, ?bgrt{<}..y'\}.\ d * x^2 + e * x + f = 0) \wedge$
$(\forall\, (d,\ e,\ f){\in}set\ b.$
$\exists\, y'{>}?bgrt.\ \forall\, x{\in}\{\, ?bgrt{<}..y'\}.\ d * x^2 + e * x + f < 0) \wedge$
$(\forall\, (d,\ e,\ f){\in}set\ c.$
$\exists\, y'{>}?bgrt.\ \forall\, x{\in}\{\, ?bgrt{<}..y'\}.\ d * x^2 + e * x + f \leq 0) \wedge$
$(\forall\, (d,\ e,\ f){\in}set\ d.$
$\exists\, y'{>}?bgrt.\ \forall\, x{\in}\{\, ?bgrt{<}..y'\}.\ d * x^2 + e * x + f \neq 0)))$

```
          using asm allpropavar allpropbvar allpropcvar allpropdvar
          by linarith
        then show False using bgrtis f10
          by auto
      qed
      have insetd: ((t, u, v) ∈ set d ∧ (t ≠ 0 ∨ u ≠ 0) ∧ (t * ?bgrt^2 + u * ?bgrt
+ v = 0)) ⟹ False
        using quadsetd linsetd by auto
      then show False using insetb insetc insetd tuvprop
        by auto
    qed
    have len1: length ?srl = 1 ⟹ False
    proof −
      assume len1: length ?srl = 1
      have cases: (List.member ?srl x) ∨ x < ?srl ! 0 ∨ x > ?srl ! 0
        using in-set-member lenzero nth-mem by fastforce
      then show False
        using len1 cases-mem cases-lt cases-gt by auto
    qed
    have lengtone: length ?srl > 1 ⟹ False
    proof −
      assume lengt1: length ?srl > 1
       have cases: (List.member ?srl x) ∨ x < ?srl ! 0 ∨ x > ?srl ! (length ?srl
−1)
              ∨ (∃ k ≤ (length ?srl − 2). (?srl ! k < x ∧ x <?srl ! (k + 1)))
      proof −
        have eo: x < ?srl ! 0 ∨ x > ?srl ! (length ?srl −1) ∨ (x ≥ ?srl ! 0 ∧ x
≤ ?srl ! (length ?srl −1))
          by auto
        have ifo: (x ≥ ?srl ! 0 ∧ x ≤ ?srl ! (length ?srl −1)) ⟹ ((List.member
?srl x) ∨ (∃ k ≤ (length ?srl − 2). ?srl ! k < x ∧ x <?srl ! (k + 1)))
        proof −
          assume xinbtw: x ≥ ?srl ! 0 ∧ x ≤ ?srl ! (length ?srl −1)
         then have ¬(List.member ?srl x) ⟹ (∃ k ≤ (length ?srl − 2). ?srl ! k
< x ∧ x <?srl ! (k + 1))
          proof −
            assume nonmem: ¬(List.member ?srl x)
           have ¬(∃ k ≤ (length ?srl − 2). ?srl ! k < x ∧ x <?srl ! (k + 1)) ⟹
False
            proof clarsimp
              assume ∀ k. sorted-nonzero-root-list-set (set b ∪ set c ∪ set d) ! k <
x ⟶
    k ≤ length (sorted-nonzero-root-list-set (set b ∪ set c ∪ set d)) − 2 ⟶
    ¬ x < sorted-nonzero-root-list-set (set b ∪ set c ∪ set d) ! Suc k
              then have allk: (∀ k ≤ length ?srl − 2. ?srl ! k < x ⟶
    ¬ x < ?srl ! Suc k) by auto
              have basec: x ≥ ?srl ! 0 using xinbtw by auto
              have ∀ k ≤ length ?srl − 2. ?srl ! k < x
              proof clarsimp
```

fix k
assume *klteq*: $k \leq$ *length* (*sorted-nonzero-root-list-set* (*set b* $\cup$ *set c* $\cup$ *set d*)) $-$ *2*
show *sorted-nonzero-root-list-set* (*set b* $\cup$ *set c* $\cup$ *set d*) ! $k <$ *x*
using *nonmem klteq basec*
proof (*induct k*)
case *0*
then show *?case*
using *in-set-member lenzero nth-mem* **by** *fastforce*
next
case (*Suc k*)
then show *?case*
by (*smt Suc-leD Suc-le-lessD* ‹$\forall$ *k. sorted-nonzero-root-list-set* (*set b* $\cup$ *set c* $\cup$ *set d*) ! $k <$ *x* $\longrightarrow$ *k* $\leq$ *length* (*sorted-nonzero-root-list-set* (*set b* $\cup$ *set c* $\cup$ *set d*)) $-$ *2* $\longrightarrow \neg$ *x* $<$ *sorted-nonzero-root-list-set* (*set b* $\cup$ *set c* $\cup$ *set d*) ! *Suc k*› *diff-less in-set-member length-0-conv length-greater-0-conv lenzero less-trans-Suc nth-mem pos2*)
qed
qed
then have *x* $\geq$ *?srl* ! (*length ?srl* $-$*1*)
using *allk*
by (*metis One-nat-def Suc-diff-Suc lengt1 less-eq-real-def less-or-eq-imp-le one-add-one plus-1-eq-Suc xinbtw*)
then have *x* $>$ *?srl* ! (*length ?srl* $-$ *1*) **using** *nonmem*
by (*metis One-nat-def Suc-le-D asm diff-Suc-Suc diff-zero in-set-member lessI less-eq-real-def nth-mem*)
then show *False* **using** *xinbtw* **by** *auto*
qed
then show ($\exists$ *k* $\leq$ (*length ?srl* $-$ *2*). *?srl* ! *k* $<$ *x* $\wedge$ *x* $<$*?srl* ! (*k* + *1*))
by *blast*
qed
then show ((*List.member ?srl x*) $\vee$ ($\exists$ *k* $\leq$ (*length ?srl* $-$ *2*). *?srl* ! *k* $<$ *x* $\wedge$ *x* $<$*?srl* ! (*k* + *1*)))
using *sorted-nth-mono*
by *auto*
qed
then show *?thesis* **using** *eo ifo* **by** *auto*
qed

have *cases-btw*: ($\exists$ *k* $\leq$ (*length ?srl* $-$ *2*). *?srl* ! *k* $<$ *x* $\wedge$ *x* $<$*?srl* ! (*k* + *1*)) $\Longrightarrow$ *False*
proof $-$
assume ($\exists$ *k* $\leq$ (*length ?srl* $-$ *2*). *?srl* ! *k* $<$ *x* $\wedge$ *x* $<$*?srl* ! (*k* + *1*))
then obtain *k* **where** *k-prop*: *k* $\leq$ (*length ?srl* $-$ *2*) $\wedge$ *?srl* ! *k* $<$ *x* $\wedge$ *x* $<$*?srl* ! (*k* + *1*)
by *auto*
have *samesign*: $\forall$ (*a*, *b*, *c*) $\in$ (*set b* $\cup$ *set c* $\cup$ *set d*).
($\forall$ *y*. (*?srl* ! *k* $<$ *y* $\wedge$ *y* $<$*?srl* ! (*k* + *1*)) $\longrightarrow$ *sign-num* (*a* $*$ y^2 + *b* $*$ *y* + *c*) = *sign-num* (*a*$*$*x*^2 + *b*$*$*x* + *c*))

proof *clarsimp*
fix t u v y
assume *insetunion*: $(t, u, v) \in set\ b \vee (t, u, v)\ \in set\ c \vee (t, u, v) \in set\ d$
assume *ygt*: $sorted\text{-}nonzero\text{-}root\text{-}list\text{-}set\ (set\ b \cup set\ c \cup set\ d)\ !\ k < y$
assume *ylt*: $y < sorted\text{-}nonzero\text{-}root\text{-}list\text{-}set\ (set\ b \cup set\ c \cup set\ d)\ !\ Suc\ k$
have *tuzer*: $t = 0 \wedge u = 0 \Longrightarrow sign\text{-}num\ (t * y^2 + u * y + v) = sign\text{-}num\ (t * x^2 + u * x + v)$
unfolding *sign-num-def*
by *auto*
have *tunonzer*: $t \neq 0 \vee u \neq 0 \Longrightarrow sign\text{-}num\ (t * y^2 + u * y + v) = sign\text{-}num\ (t * x^2 + u * x + v)$
proof −
assume *tuv-asm*: $t \neq 0 \vee u \neq 0$
have *nor*: $\neg(\exists\, q.\ q > ?srl\ !\ k \wedge q < ?srl\ !\ (k + 1) \wedge t * q^2 + u * q + v = 0)$
proof *clarsimp*
fix q
assume *qlt*: $q < sorted\text{-}nonzero\text{-}root\text{-}list\text{-}set\ (set\ b \cup set\ c \cup set\ d)\ !\ Suc\ k$
assume *qgt*: $sorted\text{-}nonzero\text{-}root\text{-}list\text{-}set\ (set\ b \cup set\ c \cup set\ d)\ !\ k < q$
assume $t * q^2 + u * q + v = 0$
then have *qin*: $q \in \{x.\ \exists (a, b, c) \in set\ b \cup set\ c \cup set\ d.\ (a \neq 0 \vee b \neq 0) \wedge a * x^2 + b * x + c = 0\}$
using *insetunion tuv-asm* **by** *auto*
have $set\ ?srl = nonzero\text{-}root\text{-}set\ (set\ b \cup set\ c \cup set\ d)$
unfolding *sorted-nonzero-root-list-set-def*
using *set-sorted-list-of-set*[*of nonzero-root-set* $(set\ b \cup set\ c \cup set\ d)$]
nonzero-root-set-finite[*of* $(set\ b \cup set\ c \cup set\ d)$]
by *auto*
then have $q \in set\ ?srl$ **using** *qin* **unfolding** *nonzero-root-set-def*
by *auto*
then have *List.member ?srl q*
using *in-set-member*[*of q ?srl*]
by *auto*
then have $\exists\, n < length\ ?srl.\ q = ?srl\ !\ n$
by (*metis* ‹$q \in set\ (sorted\text{-}nonzero\text{-}root\text{-}list\text{-}set\ (set\ b \cup set\ c \cup set\ d))$› *in-set-conv-nth*)
then obtain n **where** *nprop*: $n < length\ ?srl \wedge q = ?srl\ !\ n$ **by** *auto*
then have *ngtk*: $n > k$
proof −
have *sortedh*: *sorted ?srl*
by (*simp add*: *sorted-nonzero-root-list-set-def*)
then have *nlteq*: $n \leq k \Longrightarrow ?srl\ !\ n \leq ?srl\ !\ k$ **using** *nprop k-prop sorted-iff-nth-mono*
using *sorted-nth-mono*
by (*metis Suc-1* ‹$q \in set\ (sorted\text{-}nonzero\text{-}root\text{-}list\text{-}set\ (set\ b \cup set\ c \cup set\ d))$› *diff-less length-pos-if-in-set sup.absorb-iff2 sup.strict-boundedE*

zero-less-Suc)
have *?srl* ! *n* > *?srl* ! *k* **using** *nprop qgt* **by** *auto*
then show *?thesis*
using *nlteq*
by *linarith*
qed
then have *nltkp1*: *n* < *k+1*
proof −
have *sortedh*: *sorted ?srl*
by (*simp add*: *sorted-nonzero-root-list-set-def*)
then have *ngteq*: *k+1* ≤ *n* ⟹ *?srl* ! (*k+1*) ≤ *?srl* ! *n* **using** *nprop k-prop sorted-iff-nth-mono*
by *auto*
have *?srl* ! *n* < *?srl* ! (*k* + *1*) **using** *nprop qlt* **by** *auto*
then show *?thesis*
using *ngteq* **by** *linarith*
qed
then show *False* **using** *ngtk nltkp1* **by** *auto*
qed
have *c1*: *x* > *y* ⟹ *sign-num* (*t* ∗ *y*^{2} + *u* ∗ *y* + *v*) = *sign-num* (*t* ∗ *x*^{2} + *u* ∗ *x* + *v*)
using *nor changes-sign-var*[*of t y u v x*] *k-prop ygt ylt*
by *fastforce*
then have *c2*: *y* > *x* ⟹ *sign-num* (*t* ∗ *y*^{2} + *u* ∗ *y* + *v*) = *sign-num* (*t* ∗ *x*^{2} + *u* ∗ *x* + *v*)
using *nor changes-sign-var*[*of t x u v y*] *k-prop ygt ylt*
by *force*
then have *c3*: *x* = *y* ⟹ *sign-num* (*t* ∗ *y*^{2} + *u* ∗ *y* + *v*) = *sign-num* (*t* ∗ *x*^{2} + *u* ∗ *x* + *v*)
unfolding *sign-num-def*
by *auto*
then show *sign-num* (*t* ∗ *y*^{2} + *u* ∗ *y* + *v*) = *sign-num* (*t* ∗ *x*^{2} + *u* ∗ *x* + *v*)
using *c1 c2 c3*
by *linarith*
qed
then show *sign-num* (*t* ∗ *y*^{2} + *u* ∗ *y* + *v*) = *sign-num* (*t* ∗ *x*^{2} + *u* ∗ *x* + *v*)
using *tuzer*
by *blast*
qed

let *?bgrt* = *?srl* ! *k*

have (∀ (*a*, *b*, *c*)∈*set a*. *a* = *0* ∧ *b* = *0* ∧ *c* = *0*)
using *alleqsetvar* **by** *auto*
have *?bgrt* ∈ *set ?srl*
using *set-sorted-list-of-set nonzero-root-set-finite in-set-member k-prop asm*

by (*smt diff-Suc-less le-eq-less-or-eq less-le-trans nth-mem one-add-one plus-1-eq-Suc zero-less-one*)
then have *?bgrt* ∈ *nonzero-root-set* (*set b* ∪ *set c* ∪ *set d*)
unfolding *sorted-nonzero-root-list-set-def*
using *set-sorted-list-of-set nonzero-root-set-finite*
by *auto*
then have ∃ *t u v*. (*t*, *u*, *v*) ∈ *set b* ∪ *set c* ∪ *set d* ∧(*t* ≠ *0* ∨ *u* ≠ *0*) ∧ (*t* ∗ *?bgrt*^{2} + *u* ∗ *?bgrt* + *v* = *0*)
unfolding *nonzero-root-set-def* **by** *auto*
then obtain *t u v* **where** *tuvprop1*: (*t*, *u*, *v*) ∈ *set b* ∪ *set c* ∪ *set d* ∧(*t* ≠ *0* ∨ *u* ≠ *0*) ∧ (*t* ∗ *?bgrt*^{2} + *u* ∗ *?bgrt* + *v* = *0*)
by *auto*
then have *tuvprop*: ((*t*, *u*, *v*) ∈ *set b* ∧ (*t* ≠ *0* ∨ *u* ≠ *0*) ∧ (*t* ∗ *?bgrt*^{2} + *u* ∗ *?bgrt* + *v* = *0*))
∨ ((*t*, *u*, *v*) ∈ *set c* ∧ (*t* ≠ *0* ∨ *u* ≠ *0*) ∧ (*t* ∗ *?bgrt*^{2} + *u* ∗ *?bgrt* + *v* = *0*)) ∨
((*t*, *u*, *v*) ∈ *set d* ∧ (*t* ≠ *0* ∨ *u* ≠ *0*) ∧ (*t* ∗ *?bgrt*^{2} + *u* ∗ *?bgrt* + *v* = *0*))
by *auto*
have *tnonz*: *t*≠ *0* ⟹ (*−1*∗*u^2* + *4* ∗ *t* ∗ *v* ≤ *0* ∧ (*?bgrt* = (− *u* + *1* ∗ *sqrt* (*u^2* − *4* ∗ *t* ∗ *v*)) / (*2* ∗ *t*) ∨ *?bgrt* = (− *u* + *−1* ∗ *sqrt* (*u^2* − *4* ∗ *t* ∗ *v*)) / (*2* ∗ *t*)))
proof −
assume *t*≠ *0*
have *−1*∗*u^2* + *4* ∗ *t* ∗ *v* ≤ *0* **using** *tuvprop1 discriminant-negative*[*of t u v*]
unfolding *discrim-def*
using ‹*t* ≠ *0*› **by** *force*
then show *?thesis*
using *tuvprop discriminant-nonneg*[*of t u v*]
unfolding *discrim-def*
using ‹*t* ≠ *0*› **by** *auto*
qed
have *unonz*: (*t* = *0* ∧ *u* ≠ *0*) ⟹ *?bgrt* = − *v* / *u*
proof −
assume (*t* = *0* ∧ *u* ≠ *0*)
then have *u*∗*?bgrt* + *v* = *0* **using** *tuvprop1*
by *simp*
then show *?bgrt* = − *v* / *u*
by (*simp add*: ‹*t* = *0* ∧ *u* ≠ *0*› *eq-minus-divide-eq mult.commute*)
qed

have ∃ *y′*. *y′* > *x* ∧ *y′* < *?srl* ! (*k+1*) **using** *k-prop dense*
by *blast*
then obtain *y1* **where** *y1-prop*: *y1* > *x* ∧ *y1* < *?srl* ! (*k+1*) **by** *auto*
then have *y1inbtw*: *y1* > *?srl* ! *k* ∧ *y1* < *?srl* ! (*k+1*) **using** *k-prop*
by *auto*

have *allpropb*: (∀ (*d*, *e*, *f*)∈*set b*.

$\forall x{\in}\{?bgrt{<}..y1\}.\ d * x^2 + e * x + f < 0$)
proof *clarsimp*
fix *t1 u1 v1 x1*
assume *ins*: $(t1,\ u1,\ v1) \in set\ b$
assume *x1gt*: *sorted-nonzero-root-list-set* $(set\ b \cup set\ c \cup set\ d)\ !\ k < x1$
assume *x1lt*: $x1 \leq y1$
have *x1inbtw*: $x1 > ?srl\ !\ k \wedge x1 < ?srl\ !\ (k{+}1)$
using *x1gt x1lt y1inbtw*
by (*smt One-nat-def cases-gt k-prop*)
have *xsn*: *sign-num* $(t1 * x\hat{}2 + u1 * x + v1\ ) = -1$ **using** *ins x-prop* **unfolding** *sign-num-def*
by *auto*
have *sign-num* $(t1 * x1^2 + u1 * x1 + v1\ ) =$ *sign-num* $(t1 * x\hat{}2 + u1 * x + v1\ )$
using *ins x1inbtw samesign*
by *blast*
then show $t1 * x1^2 + u1 * x1 + v1 < 0$ **using** *xsn* **unfolding** *sign-num-def*
by (*metis add.right-inverse add.right-neutral linorder-neqE-linordered-idom one-add-one zero-neq-numeral*)
qed
have *allpropbvar*: $(\forall (d,\ e,\ f){\in}set\ b.$
$\exists y'{>}?bgrt.\ \forall x{\in}\{?bgrt{<}..y'\}.\ d * x^2 + e * x + f < 0$)
proof *clarsimp*
fix *t1 u1 v1*
assume $(t1,\ u1,\ v1) \in set\ b$
then have $\forall x{\in}\{?bgrt{<}..y1\}.\ t1 * x^2 + u1 * x + v1 < 0$
using *allpropb*
by *force*
then show $\exists y'{>}$*sorted-nonzero-root-list-set* $(set\ b \cup set\ c \cup set\ d)\ !\ k.$
$\forall x{\in}\{$*sorted-nonzero-root-list-set* $(set\ b \cup set\ c \cup set\ d)\ !\ k{<}..y'\}.$
$t1 * x^2 + u1 * x + v1 < 0$
using *y1inbtw* **by** *blast*
qed
have *allpropc*: $(\forall (d,\ e,\ f){\in}set\ c.$
$\forall x{\in}\{?bgrt{<}..y1\}.\ d * x^2 + e * x + f \leq 0$)
proof *clarsimp*
fix *t1 u1 v1 x1*
assume *ins*: $(t1,\ u1,\ v1) \in set\ c$
assume *x1gt*: *sorted-nonzero-root-list-set* $(set\ b \cup set\ c \cup set\ d)\ !\ k < x1$
assume *x1lt*: $x1 \leq y1$
have *x1inbtw*: $x1 > ?srl\ !\ k \wedge x1 < ?srl\ !\ (k{+}1)$
using *x1gt x1lt y1inbtw*
by (*smt One-nat-def cases-gt k-prop*)
have *xsn*: *sign-num* $(t1 * x\hat{}2 + u1 * x + v1\ ) = -1 \vee$ *sign-num* $(t1 * x\hat{}2 + u1 * x + v1\ ) = 0$ **using** *ins x-prop* **unfolding** *sign-num-def*
by *auto*
have *sign-num* $(t1 * x1^2 + u1 * x1 + v1\ ) =$ *sign-num* $(t1 * x\hat{}2 + u1 * x + v1\ )$

using *ins x1inbtw samesign*
by *blast*
then show $t1 * x1^2 + u1 * x1 + v1 \leq 0$ **using** *xsn* **unfolding** *sign-num-def*
by (*smt* (*verit*, *del-insts*))
qed
have *allpropcvar*: $(\forall (d, e, f) \in set\ c.$
$\exists y' > ?bgrt.\ \forall x \in \{?bgrt < .. y'\}.\ d * x^2 + e * x + f \leq 0)$
proof *clarsimp*
fix *t1 u1 v1*
assume $(t1, u1, v1) \in set\ c$
then have $\forall x \in \{?bgrt < .. y1\}.\ t1 * x^2 + u1 * x + v1 \leq 0$
using *allpropc*
by *force*
then show $\exists y' > sorted\text{-}nonzero\text{-}root\text{-}list\text{-}set\ (set\ b \cup set\ c \cup set\ d)\ !\ k.$
$\forall x \in \{sorted\text{-}nonzero\text{-}root\text{-}list\text{-}set\ (set\ b \cup set\ c \cup set\ d)\ !\ k < .. y'\}.$
$t1 * x^2 + u1 * x + v1 \leq 0$
using *y1inbtw* **by** *blast*
qed
have *allpropd*: $(\forall (d, e, f) \in set\ d.$
$\forall x \in \{?bgrt < .. y1\}.\ d * x^2 + e * x + f \neq 0)$
proof *clarsimp*
fix *t1 u1 v1 x1*
assume *ins*: $(t1, u1, v1) \in set\ d$
assume *contrad*:$t1 * x1^2 + u1 * x1 + v1 = 0$
assume *x1gt*: $sorted\text{-}nonzero\text{-}root\text{-}list\text{-}set\ (set\ b \cup set\ c \cup set\ d)\ !\ k < x1$
assume *x1lt*: $x1 \leq y1$
have *x1inbtw*: $x1 > ?srl\ !\ k \wedge x1 < ?srl\ !\ (k+1)$
using *x1gt x1lt y1inbtw*
by (*smt One-nat-def cases-gt k-prop*)
have *xsn*: $sign\text{-}num\ (t1 * x\hat{}2 + u1 * x + v1\ ) = -1 \vee sign\text{-}num\ (t1 * x\hat{}2 + u1 * x + v1\ ) = 1$ **using** *ins x-prop* **unfolding** *sign-num-def*
by *auto*
have $sign\text{-}num\ (t1 * x1^2 + u1 * x1 + v1\ ) = sign\text{-}num\ (t1 * x\hat{}2 + u1 * x + v1\ )$
using *ins x1inbtw samesign*
by *blast*
then have $t1 * x1^2 + u1 * x1 + v1 \neq 0$ **using** *xsn* **unfolding** *sign-num-def*
by *auto*
then show *False* **using** *contrad* **by** *auto*
qed
have *allpropdvar*: $(\forall (d, e, f) \in set\ d.$
$\exists y' > ?bgrt.\ \forall x \in \{?bgrt < .. y'\}.\ d * x^2 + e * x + f \neq 0)$
proof *clarsimp*
fix *t1 u1 v1*
assume $(t1, u1, v1) \in set\ d$
then have $\forall x \in \{?bgrt < .. y1\}.\ t1 * x^2 + u1 * x + v1 \neq 0$
using *allpropd*

by *force*
then show $\exists y'>$*sorted-nonzero-root-list-set* (*set b* $\cup$ *set c* $\cup$ *set d*) ! *k*.
$\forall x \in \{$*sorted-nonzero-root-list-set* (*set b* $\cup$ *set c* $\cup$ *set d*) ! *k*<..*y'*$\}$.
$t1 * x^2 + u1 * x + v1 \neq 0$
using *y1inbtw* **by** *blast*
qed
have $\forall x.\ (\forall (d,\ e,\ f) \in$*set a*.
$d * x^2 + e * x + f = 0)$ **using** *alleqsetvar*
by *auto*
then have *ast*: $(\forall (d,\ e,\ f) \in$*set a*.
$\forall x \in \{$*?bgrt*<..(*?bgrt* + *1*)$\}$. $d * x^2 + e * x + f = 0)$
by *auto*
have *allpropavar*: $(\forall (d,\ e,\ f) \in$*set a*.
$\exists y'>$*?bgrt*. $\forall x \in \{$*?bgrt*<..*y'*$\}$. $d * x^2 + e * x + f = 0)$
proof *clarsimp*
fix *t1 u1 v1*
assume $(t1,\ u1,\ v1) \in$ *set a*
then have $\forall x \in \{$*?bgrt*<..(*?bgrt* + *1*)$\}$. $t1 * x^2 + u1 * x + v1 = 0$
using *ast* **by** *auto*
then show $\exists y'>$*sorted-nonzero-root-list-set* (*set b* $\cup$ *set c* $\cup$ *set d*) ! *k*.
$\forall x \in \{$*sorted-nonzero-root-list-set* (*set b* $\cup$ *set c* $\cup$ *set d*) ! *k*<..*y'*$\}$.
$t1 * x^2 + u1 * x + v1 = 0$
using *less-add-one* **by** *blast*
qed

have *quadsetb*: $((t,\ u,\ v) \in$ *set b* $\wedge$ $t \neq 0) \Longrightarrow$ *False*
proof −
assume *asm*: $(t,\ u,\ v) \in$ *set b* $\wedge$ $t \neq 0$
have *bgrt1*: (*?bgrt* = $(-\ u + 1 * sqrt\ (u\hat{}2 - 4 * t * v)) / (2 * t)) \Longrightarrow$
False
proof −
assume *bgrtis*: *?bgrt* = $(-\ u + 1 * sqrt\ (u\hat{}2 - 4 * t * v)) / (2 * t)$
have *discrim-prop*: $-1 * u\hat{}2 + 4 * t * v \leq 0$ **using** *asm tnonz*
by *blast*
have $((t,\ u,\ v) \in$*set b* $\wedge$ $t \neq 0 \wedge -1 * u\hat{}2 + 4 * t * v \leq 0 \wedge$
$((\forall (d,\ e,\ f) \in$*set a*.
$\exists y'>$*?bgrt*. $\forall x \in \{$*?bgrt*<..*y'*$\}$. $d * x^2 + e * x + f = 0) \wedge$
$(\forall (d,\ e,\ f) \in$*set b*.
$\exists y'>$*?bgrt*. $\forall x \in \{$*?bgrt*<..*y'*$\}$. $d * x^2 + e * x + f < 0) \wedge$
$(\forall (d,\ e,\ f) \in$*set c*.
$\exists y'>$*?bgrt*. $\forall x \in \{$*?bgrt*<..*y'*$\}$. $d * x^2 + e * x + f \leq 0) \wedge$
$(\forall (d,\ e,\ f) \in$*set d*.
$\exists y'>$*?bgrt*. $\forall x \in \{$*?bgrt*<..*y'*$\}$. $d * x^2 + e * x + f \neq 0)))$
using *asm discrim-prop allpropavar allpropbvar allpropcvar allpropdvar*
by *linarith*
then show *False* **using** *f6 bgrtis*
by *auto*
qed
have *bgrt2*: (*?bgrt* = $(-\ u + -1 * sqrt\ (u\hat{}2 - 4 * t * v)) / (2 * t))$

$\Longrightarrow$ *False*
proof −
assume *bgrtis*: *?bgrt* = (− *u* + −*1* ∗ *sqrt* (*u*^*2* − *4* ∗ *t* ∗ *v*)) / (*2* ∗ *t*)
have *discrim-prop*: −*1*∗*u*^*2* + *4* ∗ *t* ∗ *v* ≤ *0* **using** *asm tnonz*
by *blast*
have ((*t*, *u*, *v*)∈*set b* ∧ *t* ≠ *0* ∧ − *1*∗*u*^*2* + *4* ∗ *t* ∗ *v* ≤ *0* ∧
((∀(*d*, *e*, *f*)∈*set a*.
∃ *y*′>*?bgrt*. ∀ *x*∈{ *?bgrt*<..*y*′}. *d* ∗ x^2 + *e* ∗ *x* + *f* = *0*) ∧
(∀(*d*, *e*, *f*)∈*set b*.
∃ *y*′>*?bgrt*. ∀ *x*∈{ *?bgrt*<..*y*′}. *d* ∗ x^2 + *e* ∗ *x* + *f* < *0*) ∧
(∀(*d*, *e*, *f*)∈*set c*.
∃ *y*′>*?bgrt*. ∀ *x*∈{ *?bgrt*<..*y*′}. *d* ∗ x^2 + *e* ∗ *x* + *f* ≤ *0*) ∧
(∀(*d*, *e*, *f*)∈*set d*.
∃ *y*′>*?bgrt*. ∀ *x*∈{ *?bgrt*<..*y*′}. *d* ∗ x^2 + *e* ∗ *x* + *f* ≠ *0*)))
using *asm discrim-prop allpropavar allpropbvar allpropcvar allpropdvar*
by *linarith*
then show *False* **using** *f7 bgrtis*
by *auto*
qed
show *False* **using** *tnonz bgrt1 bgrt2 asm*
by *auto*
qed
have *linsetb*: ((*t*, *u*, *v*) ∈ *set b* ∧ (*t* = *0* ∧ *u* ≠ *0*)) $\Longrightarrow$ *False*
proof −
assume *asm*: (*t*, *u*, *v*) ∈ *set b* ∧ (*t* = *0* ∧ *u* ≠ *0*)
then have *bgrtis*: *?bgrt* = (− *v* / *u*)
using *unonz*
by *blast*
have ((*t*, *u*, *v*)∈*set b* ∧ (*t* = *0* ∧ *u* ≠ *0*) ∧
((∀(*d*, *e*, *f*)∈*set a*.
∃ *y*′>*?bgrt*. ∀ *x*∈{ *?bgrt*<..*y*′}. *d* ∗ x^2 + *e* ∗ *x* + *f* = *0*) ∧
(∀(*d*, *e*, *f*)∈*set b*.
∃ *y*′>*?bgrt*. ∀ *x*∈{ *?bgrt*<..*y*′}. *d* ∗ x^2 + *e* ∗ *x* + *f* < *0*) ∧
(∀(*d*, *e*, *f*)∈*set c*.
∃ *y*′>*?bgrt*. ∀ *x*∈{ *?bgrt*<..*y*′}. *d* ∗ x^2 + *e* ∗ *x* + *f* ≤ *0*) ∧
(∀(*d*, *e*, *f*)∈*set d*.
∃ *y*′>*?bgrt*. ∀ *x*∈{ *?bgrt*<..*y*′}. *d* ∗ x^2 + *e* ∗ *x* + *f* ≠ *0*)))
using *asm allpropavar allpropbvar allpropcvar allpropdvar*
by *linarith*
then show *False* **using** *bgrtis f5*
by *auto*
qed
have *insetb*: ((*t*, *u*, *v*) ∈ *set b* ∧ (*t* ≠ *0* ∨ *u* ≠ *0*) ∧ (*t* ∗ *?bgrt*^{2} + *u* ∗ *?bgrt* + *v* = *0*)) $\Longrightarrow$ *False*
using *quadsetb linsetb* **by** *auto*
have *quadsetc*: (*t*, *u*, *v*) ∈ *set c* ∧ *t*≠ *0* $\Longrightarrow$ *False*
proof −
assume *asm*: (*t*, *u*, *v*) ∈ *set c* ∧ *t*≠ *0*
have *bgrt1*: (*?bgrt* = (− *u* + *1* ∗ *sqrt* (*u*^*2* − *4* ∗ *t* ∗ *v*)) / (*2* ∗ *t*)) $\Longrightarrow$

False
proof −
assume *bgrtis*: *?bgrt* = (− *u* + *1* ∗ *sqrt* (*u*^*2* − *4* ∗ *t* ∗ *v*)) / (*2* ∗ *t*)
have *discrim-prop*: −*1*∗*u*^*2* + *4* ∗ *t* ∗ *v* ≤ *0* **using** *asm tnonz*
by *blast*
have ((*t*, *u*, *v*)∈*set c* ∧ *t* ≠ *0* ∧ − *1*∗*u*^*2* + *4* ∗ *t* ∗ *v* ≤ *0* ∧
((∀(*d*, *e*, *f*)∈*set a*.
∃*y*′>*?bgrt*. ∀*x*∈{*?bgrt*<..*y*′}. *d* ∗ x^2 + *e* ∗ *x* + *f* = *0*) ∧
(∀(*d*, *e*, *f*)∈*set b*.
∃*y*′>*?bgrt*. ∀*x*∈{*?bgrt*<..*y*′}. *d* ∗ x^2 + *e* ∗ *x* + *f* < *0*) ∧
(∀(*d*, *e*, *f*)∈*set c*.
∃*y*′>*?bgrt*. ∀*x*∈{*?bgrt*<..*y*′}. *d* ∗ x^2 + *e* ∗ *x* + *f* ≤ *0*) ∧
(∀(*d*, *e*, *f*)∈*set d*.
∃*y*′>*?bgrt*. ∀*x*∈{*?bgrt*<..*y*′}. *d* ∗ x^2 + *e* ∗ *x* + *f* ≠ *0*)))
using *asm discrim-prop allpropavar allpropbvar allpropcvar allpropdvar*
by *linarith*
then show *False* **using** *f13a bgrtis*
by *auto*
qed
have *bgrt2*: (*?bgrt* = (− *u* + −*1* ∗ *sqrt* (*u*^*2* − *4* ∗ *t* ∗ *v*)) / (*2* ∗ *t*))
⟹ *False*
proof −
assume *bgrtis*: *?bgrt* = (− *u* + −*1* ∗ *sqrt* (*u*^*2* − *4* ∗ *t* ∗ *v*)) / (*2* ∗ *t*)
have *discrim-prop*: −*1*∗*u*^*2* + *4* ∗ *t* ∗ *v* ≤ *0* **using** *asm tnonz*
by *blast*
have ((*t*, *u*, *v*)∈*set c* ∧ *t* ≠ *0* ∧ − *1*∗*u*^*2* + *4* ∗ *t* ∗ *v* ≤ *0* ∧
((∀(*d*, *e*, *f*)∈*set a*.
∃*y*′>*?bgrt*. ∀*x*∈{*?bgrt*<..*y*′}. *d* ∗ x^2 + *e* ∗ *x* + *f* = *0*) ∧
(∀(*d*, *e*, *f*)∈*set b*.
∃*y*′>*?bgrt*. ∀*x*∈{*?bgrt*<..*y*′}. *d* ∗ x^2 + *e* ∗ *x* + *f* < *0*) ∧
(∀(*d*, *e*, *f*)∈*set c*.
∃*y*′>*?bgrt*. ∀*x*∈{*?bgrt*<..*y*′}. *d* ∗ x^2 + *e* ∗ *x* + *f* ≤ *0*) ∧
(∀(*d*, *e*, *f*)∈*set d*.
∃*y*′>*?bgrt*. ∀*x*∈{*?bgrt*<..*y*′}. *d* ∗ x^2 + *e* ∗ *x* + *f* ≠ *0*)))
using *asm discrim-prop allpropavar allpropbvar allpropcvar allpropdvar*
by *linarith*
then show *False* **using** *f9a bgrtis*
by *auto*
qed
show *False* **using** *tnonz bgrt1 bgrt2 asm*
by *auto*
qed
have *linsetc*: (*t*, *u*, *v*) ∈ *set c* ∧ (*t* = *0* ∧ *u* ≠ *0*) ⟹ *False*
proof −
assume *asm*: (*t*, *u*, *v*) ∈ *set c* ∧ (*t* = *0* ∧ *u* ≠ *0*)
then have *bgrtis*: *?bgrt* = (− *v* / *u*)
using *unonz*
by *blast*
have ((*t*, *u*, *v*)∈*set c* ∧ (*t* = *0* ∧ *u* ≠ *0*) ∧

((∀(d, e, f)∈set a.
∃y'>?bgrt. ∀x∈{?bgrt<..y'}. d * x^2 + e * x + f = 0) ∧
(∀(d, e, f)∈set b.
∃y'>?bgrt. ∀x∈{?bgrt<..y'}. d * x^2 + e * x + f < 0) ∧
(∀(d, e, f)∈set c.
∃y'>?bgrt. ∀x∈{?bgrt<..y'}. d * x^2 + e * x + f ≤ 0) ∧
(∀(d, e, f)∈set d.
∃y'>?bgrt. ∀x∈{?bgrt<..y'}. d * x^2 + e * x + f ≠ 0)))
using *asm allpropavar allpropbvar allpropcvar allpropdvar*
by *linarith*
then show *False* **using** *bgrtis f8a*
by *auto*
qed
have *insetc*: ((t, u, v) ∈ set c ∧ (t ≠ 0 ∨ u ≠ 0) ∧ (t * ?bgrt^2 + u * ?bgrt + v = 0)) ⟹ *False*
using *quadsetc linsetc* **by** *auto*
have *quadsetd*: (t, u, v) ∈ set d ∧ t≠ 0 ⟹ *False*
proof −
assume *asm*: (t, u, v) ∈ set d ∧ t≠ 0
have *bgrt1*: (?bgrt = (− u + 1 * sqrt (u^2 − 4 * t * v)) / (2 * t)) ⟹ *False*
proof −
assume *bgrtis*: ?bgrt = (− u + 1 * sqrt (u^2 − 4 * t * v)) / (2 * t)
have *discrim-prop*: −1*u^2 + 4 * t * v ≤ 0 **using** *asm tnonz*
by *blast*
have ((t, u, v)∈set d ∧ t ≠ 0 ∧ − 1*u^2 + 4 * t * v ≤ 0 ∧
((∀(d, e, f)∈set a.
∃y'>?bgrt. ∀x∈{?bgrt<..y'}. d * x^2 + e * x + f = 0) ∧
(∀(d, e, f)∈set b.
∃y'>?bgrt. ∀x∈{?bgrt<..y'}. d * x^2 + e * x + f < 0) ∧
(∀(d, e, f)∈set c.
∃y'>?bgrt. ∀x∈{?bgrt<..y'}. d * x^2 + e * x + f ≤ 0) ∧
(∀(d, e, f)∈set d.
∃y'>?bgrt. ∀x∈{?bgrt<..y'}. d * x^2 + e * x + f ≠ 0)))
using *asm discrim-prop allpropavar allpropbvar allpropcvar allpropdvar*
by *linarith*
then show *False* **using** *f11 bgrtis*
by *auto*
qed
have *bgrt2*: (?bgrt = (− u + −1 * sqrt (u^2 − 4 * t * v)) / (2 * t)) ⟹ *False*
proof −
assume *bgrtis*: ?bgrt = (− u + −1 * sqrt (u^2 − 4 * t * v)) / (2 * t)
have *discrim-prop*: −1*u^2 + 4 * t * v ≤ 0 **using** *asm tnonz*
by *blast*
have ((t, u, v)∈set d ∧ t ≠ 0 ∧ − 1*u^2 + 4 * t * v ≤ 0 ∧
((∀(d, e, f)∈set a.
∃y'>?bgrt. ∀x∈{?bgrt<..y'}. d * x^2 + e * x + f = 0) ∧
(∀(d, e, f)∈set b.

$\exists\, y' > ?bgrt.\ \forall\, x \in \{\, ?bgrt < .. y' \}.\ d * x^2 + e * x + f < 0) \wedge$
$(\forall\, (d,\ e,\ f) \in set\ c.$
$\exists\, y' > ?bgrt.\ \forall\, x \in \{\, ?bgrt < .. y' \}.\ d * x^2 + e * x + f \leq 0) \wedge$
$(\forall\, (d,\ e,\ f) \in set\ d.$
$\exists\, y' > ?bgrt.\ \forall\, x \in \{\, ?bgrt < .. y' \}.\ d * x^2 + e * x + f \neq 0)))$
using *asm discrim-prop allpropavar allpropbvar allpropcvar allpropdvar*
by *linarith*
then show *False* **using** *f12 bgrtis*
by *auto*
qed
show *False* **using** *tnonz bgrt1 bgrt2 asm*
by *auto*
qed
have *linsetd*: $(t,\ u,\ v) \in set\ d \wedge (t = 0 \wedge u \neq 0) \Longrightarrow False$
proof −
assume *asm*: $(t,\ u,\ v) \in set\ d \wedge (t = 0 \wedge u \neq 0)$
then have *bgrtis*: $?bgrt = (-\ v\ /\ u)$
using *unonz*
by *blast*
have $((t,\ u,\ v) \in set\ d \wedge (t = 0 \wedge u \neq 0) \wedge$
$((\forall\, (d,\ e,\ f) \in set\ a.$
$\exists\, y' > ?bgrt.\ \forall\, x \in \{\, ?bgrt < .. y' \}.\ d * x^2 + e * x + f = 0) \wedge$
$(\forall\, (d,\ e,\ f) \in set\ b.$
$\exists\, y' > ?bgrt.\ \forall\, x \in \{\, ?bgrt < .. y' \}.\ d * x^2 + e * x + f < 0) \wedge$
$(\forall\, (d,\ e,\ f) \in set\ c.$
$\exists\, y' > ?bgrt.\ \forall\, x \in \{\, ?bgrt < .. y' \}.\ d * x^2 + e * x + f \leq 0) \wedge$
$(\forall\, (d,\ e,\ f) \in set\ d.$
$\exists\, y' > ?bgrt.\ \forall\, x \in \{\, ?bgrt < .. y' \}.\ d * x^2 + e * x + f \neq 0)))$
using *asm allpropavar allpropbvar allpropcvar allpropdvar*
by *linarith*
then show *False* **using** *bgrtis f10*
by *auto*
qed
have *insetd*: $((t,\ u,\ v) \in set\ d \wedge (t \neq 0 \vee u \neq 0) \wedge (t * ?bgrt^2 + u *$
$?bgrt + v = 0)) \Longrightarrow False$
using *quadsetd linsetd* **by** *auto*
then show *False* **using** *insetb insetc insetd tuvprop*
by *auto*
qed
show *False* **using** *cases cases-btw cases-mem cases-lt cases-gt*
by *auto*
qed
show *False* **using** *asm len1 lengtone*
by *linarith*
qed
show *False* **using** *lenzero lengt0*
by *linarith*
qed
then show *?thesis*

by *blast*
qed

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

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

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

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

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

proof −
let *?e2* = (((∀(*a*, *b*, *c*)∈*set a*. *a* = *0* ∧ *b* = *0* ∧ *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*) ∧
(∀(*a*, *b*, *c*)∈*set d*. ∃*x*. ∀*y*<*x*. *a* * *y*^2 + *b* * *y* + *c* ≠ *0*)
∨
(∃(*a*', *b*', *c*')∈*set a*.
(*a*' = *0* ∧ *b*' ≠ *0*) ∧
(∀(*d*, *e*, *f*)∈*set a*. *d* * (− *c*' / *b*')^2 + *e* * (− *c*' / *b*') + *f* = *0*) ∧
(∀(*d*, *e*, *f*)∈*set b*. *d* * (− *c*' / *b*')^2 + *e* * (− *c*' / *b*') + *f* < *0*) ∧
(∀(*d*, *e*, *f*)∈*set c*. *d* * (− *c*' / *b*')^2 + *e* * (− *c*' / *b*') + *f* ≤ *0*) ∧
(∀(*d*, *e*, *f*)∈*set d*. *d* * (− *c*' / *b*')^2 + *e* * (− *c*' / *b*') + *f* ≠ *0*)
∨
a' ≠ *0* ∧
− *b*'^2 + *4* * *a*' * *c*' ≤ *0* ∧
((∀(*d*, *e*, *f*)∈*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) ∧
(∀(*d*, *e*, *f*)∈*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*) ∧
(∀(*d*, *e*, *f*)∈*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 ≤ *0*) ∧
(∀(*d*, *e*, *f*)∈*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 ≠ *0*)
∨
(∀(*d*, *e*, *f*)∈*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*) ∧
(∀(*d*, *e*, *f*)∈*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*) ∧
(∀(*d*, *e*, *f*)∈*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 ≤ *0*) ∧
(∀(*d*, *e*, *f*)∈*set d*.
d * ((− *b*' + − *1* * *sqrt* (*b*'^2 − *4* * *a*' * *c*')) / (*2* * *a*'))^2 +

$$
\begin{array}{l}
\quad e * ((-\ b' + -\ 1 * sqrt\ (b'^2 - 4 * a' * c')) \ /\ (2 * a')) + \\
\quad f \neq 0))) \\
\vee \\
(\exists\,(a',\ b',\ c') \in set\ b. \\
\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}
$$

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

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

let *?f10orf11orf12* $= (\exists\,(a',\ b',\ c')\in set\ d.$
$(a' = 0 \wedge b' \neq 0) \wedge$
$(\forall\,(d,\ e,\ f)\in set\ a.$
$\quad \exists\, y'>-\ c'\ /\ b'.\ \forall\, x\in\{-\ c'\ /\ b'<..y'\}.\ d * x^2 + e * x + f = 0) \wedge$
$(\forall\,(d,\ e,\ f)\in set\ b.$
$\quad \exists\, y'>-\ c'\ /\ b'.\ \forall\, x\in\{-\ c'\ /\ b'<..y'\}.\ d * x^2 + e * x + f < 0) \wedge$
$(\forall\,(d,\ e,\ f)\in set\ c.$
$\quad \exists\, y'>-\ c'\ /\ b'.\ \forall\, x\in\{-\ c'\ /\ b'<..y'\}.\ d * x^2 + e * x + f \leq 0) \wedge$
$(\forall\,(d,\ e,\ f)\in set\ d.$

$$
\begin{array}{l}
\quad \exists\, y'{>}-\ c'\ /\ b'.\ \forall\, x{\in}\{-\ c'\ /\ b'{<}..y'\}.\ d * x^2 + e * x + f \neq 0) \\
\vee \\
\quad a' \neq 0\ \wedge \\
\quad -\ b'^2 + 4 * a' * c' \leq 0\ \wedge \\
\quad ((\forall\,(d,\ e,\ f){\in}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))) \\
\textbf{let}\ ?f8orf9 = (\exists\,(a',\ b',\ c'){\in}set\ c. \\
\quad (a' = 0\ \wedge\ b' \neq 0)\ \wedge \\
\quad (\forall\,(d,\ e,\ f){\in}set\ a.\ d * (-\ c'\ /\ b')^2 + e * (-\ c'\ /\ b') + f = 0)\ \wedge \\
\quad (\forall\,(d,\ e,\ f){\in}set\ b.\ d * (-\ c'\ /\ b')^2 + e * (-\ c'\ /\ b') + f < 0)\ \wedge \\
\quad (\forall\,(d,\ e,\ f){\in}set\ c.\ d * (-\ c'\ /\ b')^2 + e * (-\ c'\ /\ b') + f \leq 0)\ \wedge \\
\quad (\forall\,(d,\ e,\ f){\in}set\ d.\ d * (-\ c'\ /\ b')^2 + e * (-\ c'\ /\ b') + f \neq 0)\ \vee \\
\quad a' \neq 0\ \wedge \\
\quad -\ b'^2 + 4 * a' * c' \leq 0\ \wedge \\
\quad ((\forall\,(d,\ e,\ f){\in}set\ a. \\
\qquad d * ((-\ b' + 1 * sqrt\ (b'^2 - 4 * a' * c'))\ /\ (2 * a'))^2 + \\
\qquad e * ((-\ b' + 1 * sqrt\ (b'^2 - 4 * a' * c'))\ /\ (2 * a')) + \\
\qquad f = 0)\ \wedge
\end{array}
$$

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

let *?f5orf6orf7* $= (\exists\,(a',\ b',\ c')\in set\ b.$
$(a' = 0 \wedge b' \neq 0) \wedge$
$(\forall\,(d,\ e,\ f)\in set\ a.$
$\exists\,y'>-\ c' / b'.\ \forall\,x\in\{-\ c' / b'<..y'\}.\ d * x^2 + e * x + f = 0) \wedge$
$(\forall\,(d,\ e,\ f)\in set\ b.$
$\exists\,y'>-\ c' / b'.\ \forall\,x\in\{-\ c' / b'<..y'\}.\ d * x^2 + e * x + f < 0) \wedge$
$(\forall\,(d,\ e,\ f)\in set\ c.$
$\exists\,y'>-\ c' / b'.\ \forall\,x\in\{-\ c' / b'<..y'\}.\ d * x^2 + e * x + f \leq 0) \wedge$
$(\forall\,(d,\ e,\ f)\in set\ d.$
$\exists\,y'>-\ c' / b'.\ \forall\,x\in\{-\ c' / b'<..y'\}.\ d * x^2 + e * x + f \neq 0)$
$\vee$
$a' \neq 0 \wedge$
$-\ b'^2 + 4 * a' * c' \leq 0 \wedge$
$((\forall\,(d,\ e,\ f)\in set\ a.$
$\exists\,y'>(-\ b' + 1 * sqrt\ (b'^2 - 4 * a' * c')) / (2 * a').$
$\forall\,x\in\{(-\ b' + 1 * sqrt\ (b'^2 - 4 * a' * c')) / (2 * a')<..y'\}.$
$d * x^2 + e * x + f = 0) \wedge$
$(\forall\,(d,\ e,\ f)\in set\ b.$
$\exists\,y'>(-\ b' + 1 * sqrt\ (b'^2 - 4 * a' * c')) / (2 * a').$
$\forall\,x\in\{(-\ b' + 1 * sqrt\ (b'^2 - 4 * a' * c')) / (2 * a')<..y'\}.$

$$
\begin{array}{l}
\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) \\
\vee \\
\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))) \\
\textbf{let}\ ?f2orf3orf4 = (\exists (a',\ b',\ c') \in set\ a. \\
\quad (a' = 0 \wedge b' \neq 0) \wedge \\
\quad (\forall (d,\ e,\ f) \in set\ a.\ d * (-\ c' \ /\ b')^2 + e * (-\ c' \ /\ b') + f = 0) \wedge \\
\quad (\forall (d,\ e,\ f) \in set\ b.\ d * (-\ c' \ /\ b')^2 + e * (-\ c' \ /\ b') + f < 0) \wedge \\
\quad (\forall (d,\ e,\ f) \in set\ c.\ d * (-\ c' \ /\ b')^2 + e * (-\ c' \ /\ b') + f \leq 0) \wedge \\
\quad (\forall (d,\ e,\ f) \in set\ d.\ d * (-\ c' \ /\ b')^2 + e * (-\ c' \ /\ b') + 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. \\
\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 = 0) \wedge \\
\quad (\forall (d,\ e,\ f) \in set\ b. \\
\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 < 0) \wedge \\
\quad (\forall (d,\ e,\ f) \in set\ c. \\
\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 \leq 0) \wedge \\
\quad (\forall (d,\ e,\ f) \in set\ d. \\
\quad\quad d * ((-\ b' + 1 * sqrt\ (b'^2 - 4 * a' * c')) \ /\ (2 * a'))^2 +
\end{array}
$$

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

let *?e1* $= (\exists\, x.\ (\forall (a,\ b,\ c) \in set\ a.\ a * x^2 + b * x + c = 0) \wedge$
$(\forall (a,\ b,\ c) \in set\ b.\ a * x^2 + b * x + c < 0) \wedge$
$(\forall (a,\ b,\ c) \in set\ c.\ a * x^2 + b * x + c \leq 0) \wedge$
$(\forall (a,\ b,\ c) \in set\ d.\ a * x^2 + b * x + c \neq 0))$

let *?f1* $= ((\forall (a,\ b,\ c) \in set\ a.\ a = 0 \wedge b = 0 \wedge c = 0) \wedge$
$(\forall (a,\ b,\ c) \in set\ b.\ \exists\, x.\ \forall\, y < x.\ a * y^2 + b * y + c < 0) \wedge$
$(\forall (a,\ b,\ c) \in set\ c.\ \exists\, x.\ \forall\, y < x.\ a * y^2 + b * y + c \leq 0) \wedge$
$(\forall (a,\ b,\ c) \in set\ d.\ \exists\, x.\ \forall\, y < x.\ a * y^2 + b * y + c \neq 0))$

let *?f2* $= (\exists (a',\ b',\ c') \in set\ a.$
$(a' = 0 \wedge b' \neq 0) \wedge$
$(\forall (d,\ e,\ f) \in set\ a.\ d * (-\ c' / b')^2 + e * (-\ c' / b') + f = 0) \wedge$
$(\forall (d,\ e,\ f) \in set\ b.\ d * (-\ c' / b')^2 + e * (-\ c' / b') + f < 0) \wedge$
$(\forall (d,\ e,\ f) \in set\ c.\ d * (-\ c' / b')^2 + e * (-\ c' / b') + f \leq 0) \wedge$
$(\forall (d,\ e,\ f) \in set\ d.\ d * (-\ c' / b')^2 + e * (-\ c' / b') + f \neq 0))$

let *?f3* $= (\exists (a',\ b',\ c') \in set\ a.\ a' \neq 0 \wedge -\ b'^2 + 4 * a' * c' \leq 0 \wedge$
$(\forall (d,\ e,\ f) \in set\ a.$
$d * ((-\ b' + 1 * sqrt\ (b'^2 - 4 * a' * c')) / (2 * a'))^2 +$
$e * ((-\ b' + 1 * sqrt\ (b'^2 - 4 * a' * c')) / (2 * a')) +$
$f = 0) \wedge$
$(\forall (d,\ e,\ f) \in set\ b.$
$d * ((-\ b' + 1 * sqrt\ (b'^2 - 4 * a' * c')) / (2 * a'))^2 +$
$e * ((-\ b' + 1 * sqrt\ (b'^2 - 4 * a' * c')) / (2 * a')) +$
$f < 0) \wedge$
$(\forall (d,\ e,\ f) \in set\ c.$
$d * ((-\ b' + 1 * sqrt\ (b'^2 - 4 * a' * c')) / (2 * a'))^2 +$
$e * ((-\ b' + 1 * sqrt\ (b'^2 - 4 * a' * c')) / (2 * a')) +$
$f \leq 0) \wedge$
$(\forall (d,\ e,\ f) \in set\ d.$
$d * ((-\ b' + 1 * sqrt\ (b'^2 - 4 * a' * c')) / (2 * a'))^2 +$
$e * ((-\ b' + 1 * sqrt\ (b'^2 - 4 * a' * c')) / (2 * a')) +$

$f \neq 0))$

let *?f4* $= (\exists\,(a',\ b',\ c') \in set\ a.\ a' \neq 0 \wedge -\ b'^2 + 4 * a' * c' \leq 0 \wedge$
$(\forall\,(d,\ e,\ f) \in set\ a.$
$d * ((-\ b' + -\ 1 * sqrt\ (b'^2 - 4 * a' * c')) / (2 * a'))^2 +$
$e * ((-\ b' + -\ 1 * sqrt\ (b'^2 - 4 * a' * c')) / (2 * a')) +$
$f = 0) \wedge$
$(\forall\,(d,\ e,\ f) \in set\ b.$
$d * ((-\ b' + -\ 1 * sqrt\ (b'^2 - 4 * a' * c')) / (2 * a'))^2 +$
$e * ((-\ b' + -\ 1 * sqrt\ (b'^2 - 4 * a' * c')) / (2 * a')) +$
$f < 0) \wedge$
$(\forall\,(d,\ e,\ f) \in set\ c.$
$d * ((-\ b' + -\ 1 * sqrt\ (b'^2 - 4 * a' * c')) / (2 * a'))^2 +$
$e * ((-\ b' + -\ 1 * sqrt\ (b'^2 - 4 * a' * c')) / (2 * a')) +$
$f \leq 0) \wedge$
$(\forall\,(d,\ e,\ f) \in set\ d.$
$d * ((-\ b' + -\ 1 * sqrt\ (b'^2 - 4 * a' * c')) / (2 * a'))^2 +$
$e * ((-\ b' + -\ 1 * sqrt\ (b'^2 - 4 * a' * c')) / (2 * a')) +$
$f \neq 0))$

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

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

let *?f7* $= (\exists\,(a',\ b',\ c') \in set\ b.\ a' \neq 0 \wedge$
$-\ b'^2 + 4 * a' * c' \leq 0 \wedge (\forall\,(d,\ e,\ f) \in set\ a.$
$\exists\,y' > (-\ b' + -\ 1 * sqrt\ (b'^2 - 4 * a' * c')) / (2 * a').$

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

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

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

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

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

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

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

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

$d * x^2 + e * x + f \leq 0) \wedge$
$(\forall (d,\ e,\ f) \in set\ d.$
$\exists y' > (-\ b' + -\ 1 * sqrt\ (b'^2 - 4 * a' * c')) / (2 * a').$
$\forall x \in \{(-\ b' + -\ 1 * sqrt\ (b'^2 - 4 * a' * c')) / (2 * a') < .. y'\}.$
$d * x^2 + e * x + f \neq 0))$
have *h1a*: $(?f1 \vee ?f2orf3orf4 \vee ?f5orf6orf7 \vee ?f8orf9 \vee ?f10orf11orf12) \longrightarrow ?e2$
by *auto*
have *h2*: $(?f2 \vee ?f3 \vee ?f4) \longrightarrow ?f2orf3orf4$ **by** *auto*
then have *h1b*: $(?f1 \vee ?f2 \vee ?f3 \vee ?f4 \vee ?f5orf6orf7 \vee ?f8orf9 \vee ?f10orf11orf12) \longrightarrow ?e2$
using *h1a* **by** *auto*
have *h3*: $(?f5 \vee ?f6 \vee ?f7) \longrightarrow ?f5orf6orf7$ **by** *auto*
then have *h1c*: $(?f1 \vee ?f2 \vee ?f3 \vee ?f4 \vee ?f5 \vee ?f6 \vee ?f7 \vee ?f8orf9 \vee ?f10orf11orf12) \longrightarrow ?e2$
using *h1b* **by** *smt*
have *h4*: $(?f8 \vee ?f9 \vee ?f13) \longrightarrow ?f8orf9$ **by** *auto*
then have *h1d*: $(?f1 \vee ?f2 \vee ?f3 \vee ?f4 \vee ?f5 \vee ?f6 \vee ?f7 \vee ?f8 \vee ?f9 \vee ?f13 \vee ?f10orf11orf12) \longrightarrow ?e2$
using *h1c*
by *smt*
have *h5*: $(?f10 \vee ?f11 \vee ?f12) \longrightarrow ?f10orf11orf12$
by *auto*
then have *bigor*: $(?f1 \vee ?f2 \vee ?f3 \vee ?f4 \vee ?f5 \vee ?f6 \vee ?f7 \vee ?f8 \vee ?f13 \vee ?f9 \vee ?f10 \vee ?f11 \vee ?f12)$
$\longrightarrow ?e2$
using *h1d* **by** *smt*
then have *bigor-var*: $\neg ?e2 \longrightarrow \neg(?f1 \vee ?f2 \vee ?f3 \vee ?f4 \vee ?f5 \vee ?f6 \vee ?f7 \vee ?f8 \vee ?f13 \vee ?f9 \vee ?f10 \vee ?f11 \vee ?f12)$
using *contrapos-nn*
by *smt*
have *not-eq*: $\neg(?f1 \vee ?f2 \vee ?f3 \vee ?f4 \vee ?f5 \vee ?f6 \vee ?f7 \vee ?f8 \vee ?f13 \vee ?f9 \vee ?f10 \vee ?f11 \vee ?f12)$
$=(\neg ?f1 \wedge \neg ?f2 \wedge \neg ?f3 \wedge \neg ?f4 \wedge \neg ?f5 \wedge \neg ?f6 \wedge \neg ?f7 \wedge \neg ?f8 \wedge \neg ?f13 \wedge \neg ?f9 \wedge \neg ?f10 \wedge \neg ?f11 \wedge \neg ?f12)$
by *linarith*
obtain *x* **where** *x-prop*: $(\forall (a,\ b,\ c) \in set\ a.\ a * x^2 + b * x + c = 0) \wedge$
$(\forall (a,\ b,\ c) \in set\ b.\ a * x^2 + b * x + c < 0) \wedge$
$(\forall (a,\ b,\ c) \in set\ c.\ a * x^2 + b * x + c \leq 0) \wedge$
$(\forall (a,\ b,\ c) \in set\ d.\ a * x^2 + b * x + c \neq 0)$ **using** *assms* **by** *auto*
have $(\neg ?f1 \wedge \neg ?f2 \wedge \neg ?f3 \wedge \neg ?f4 \wedge \neg ?f5 \wedge \neg ?f6 \wedge \neg ?f7 \wedge \neg ?f8 \wedge \neg ?f13 \wedge \neg ?f9 \wedge \neg ?f10 \wedge \neg ?f11 \wedge \neg ?f12) \Longrightarrow False$
proof −
assume *big-not*: $\neg\ ((\forall (a,\ b,\ c) \in set\ a.\ a = 0 \wedge b = 0 \wedge c = 0) \wedge$
$(\forall (a,\ b,\ c) \in set\ b.\ \exists x.\ \forall y < x.\ a * y^2 + b * y + c < 0) \wedge$
$(\forall (a,\ b,\ c) \in set\ c.\ \exists x.\ \forall y < x.\ a * y^2 + b * y + c \leq 0) \wedge$
$(\forall (a,\ b,\ c) \in set\ d.\ \exists x.\ \forall y < x.\ a * y^2 + b * y + c \neq 0)) \wedge$
$\neg\ (\exists (a',\ b',\ c') \in set\ a.$
$(a' = 0 \wedge b' \neq 0) \wedge$
$(\forall (d,\ e,\ f) \in set\ a.\ d * (-\ c' / b')^2 + e * (-\ c' / b') + f = 0) \wedge$

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

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

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

$a'\neq 0\ \wedge$

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

$a'\neq 0\ \wedge$

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

$a'\neq 0\ \wedge$

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

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

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

have *c1*: $(\exists\ (d,\ e,\ f)\in set\ a.\ d\neq 0\ \wedge\ -\ e^2+4*d*f\leq 0)\Longrightarrow False$

proof −

assume $(\exists\ (d,\ e,\ f)\in set\ a.\ d\neq 0\ \wedge\ -\ e^2+4*d*f\leq 0)$

then obtain $a'\ b'\ c'$ **where** *abc-prop*: $(a',\ b',\ c')\in set\ a\ \wedge\ a'\neq 0\ \wedge\ -\ b'^2+4*a'*c'\leq 0$

by *auto*

then have $a'*x\hat{}2+b'*x+c'=0$ **using** *x-prop* **by** *auto*

then have *xis*: $x = (-\ b' + -\ 1 * sqrt\ (b'^2 - 4 * a' * c')) / (2 * a') \vee x = (-\ b' + 1 * sqrt\ (b'^2 - 4 * a' * c')) / (2 * a')$
using *abc-prop discriminant-nonneg*[*of* $a'\ b'\ c'$] **unfolding** *discrim-def*
by *auto*
then have $((\forall (d,\ e,\ f) \in set\ a.$
$d * ((-\ b' + 1 * sqrt\ (b'^2 - 4 * a' * c')) / (2 * a'))^2 +$
$e * ((-\ b' + 1 * sqrt\ (b'^2 - 4 * a' * c')) / (2 * a')) +$
$f = 0) \wedge$
$(\forall (d,\ e,\ f) \in set\ b.$
$d * ((-\ b' + 1 * sqrt\ (b'^2 - 4 * a' * c')) / (2 * a'))^2 +$
$e * ((-\ b' + 1 * sqrt\ (b'^2 - 4 * a' * c')) / (2 * a')) +$
$f < 0) \wedge$
$(\forall (d,\ e,\ f) \in set\ c.$
$d * ((-\ b' + 1 * sqrt\ (b'^2 - 4 * a' * c')) / (2 * a'))^2 +$
$e * ((-\ b' + 1 * sqrt\ (b'^2 - 4 * a' * c')) / (2 * a')) +$
$f \leq 0) \wedge$
$(\forall (d,\ e,\ f) \in set\ d.$
$d * ((-\ b' + 1 * sqrt\ (b'^2 - 4 * a' * c')) / (2 * a'))^2 +$
$e * ((-\ b' + 1 * sqrt\ (b'^2 - 4 * a' * c')) / (2 * a')) +$
$f \neq 0)) \vee$
$((\forall (d,\ e,\ f) \in set\ a.$
$d * ((-\ b' + -\ 1 * sqrt\ (b'^2 - 4 * a' * c')) / (2 * a'))^2 +$
$e * ((-\ b' + -\ 1 * sqrt\ (b'^2 - 4 * a' * c')) / (2 * a')) +$
$f = 0) \wedge$
$(\forall (d,\ e,\ f) \in set\ b.$
$d * ((-\ b' + -\ 1 * sqrt\ (b'^2 - 4 * a' * c')) / (2 * a'))^2 +$
$e * ((-\ b' + -\ 1 * sqrt\ (b'^2 - 4 * a' * c')) / (2 * a')) +$
$f < 0) \wedge$
$(\forall (d,\ e,\ f) \in set\ c.$
$d * ((-\ b' + -\ 1 * sqrt\ (b'^2 - 4 * a' * c')) / (2 * a'))^2 +$
$e * ((-\ b' + -\ 1 * sqrt\ (b'^2 - 4 * a' * c')) / (2 * a')) +$
$f \leq 0) \wedge$
$(\forall (d,\ e,\ f) \in set\ d.$
$d * ((-\ b' + -\ 1 * sqrt\ (b'^2 - 4 * a' * c')) / (2 * a'))^2 +$
$e * ((-\ b' + -\ 1 * sqrt\ (b'^2 - 4 * a' * c')) / (2 * a')) +$
$f \neq 0))$
using *x-prop* **by** *auto*
then have $(\exists (a',\ b',\ c') \in set\ a.$
$a' \neq 0 \wedge$
$-\ b'^2 + 4 * a' * c' \leq 0 \wedge$
$(\forall (d,\ e,\ f) \in set\ a.$
$d * ((-\ b' + 1 * sqrt\ (b'^2 - 4 * a' * c')) / (2 * a'))^2 +$
$e * ((-\ b' + 1 * sqrt\ (b'^2 - 4 * a' * c')) / (2 * a')) +$
$f = 0) \wedge$
$(\forall (d,\ e,\ f) \in set\ b.$
$d * ((-\ b' + 1 * sqrt\ (b'^2 - 4 * a' * c')) / (2 * a'))^2 +$
$e * ((-\ b' + 1 * sqrt\ (b'^2 - 4 * a' * c')) / (2 * a')) +$
$f < 0) \wedge$
$(\forall (d,\ e,\ f) \in set\ c.$

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

then show *False*

using *big-not* **by** *auto*

qed

have *c2*: $(\exists\ (d,\ e,\ f) \in set\ a.\ d = 0 \wedge e \neq 0) \Longrightarrow False$

proof −

assume $(\exists\ (d,\ e,\ f) \in set\ a.\ d = 0 \wedge e \neq 0)$

then obtain $a'\ b'\ c'$ **where** *abc-prop*: $(a',\ b',\ c') \in set\ a \wedge a' = 0 \wedge b' \neq 0$
by *auto*

then have $a'*x\hat{}2 + b'*x + c' = 0$ **using** *x-prop* **by** *auto*

then have $b'*x + c' = 0$ **using** *abc-prop* **by** *auto*

then have *xis*: $x = -\ c' / b'$ **using** *abc-prop*

by (*smt divide-non-zero*)

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

using *x-prop* **by** *auto*

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

using *abc-prop xis* **by** *auto*
then show *False*
using *big-not* **by** *auto*
qed
have *c3*: $(\forall\ (d,\ e,\ f) \in set\ a.\ d = 0 \wedge e = 0 \wedge f = 0) \Longrightarrow False$
proof −
assume $(\forall\ (d,\ e,\ f) \in set\ a.\ d = 0 \wedge e = 0 \wedge f = 0)$
then have *equalset*: $\forall x.\ (\forall (d,\ e,\ f) \in set\ a.\ d * x\hat{}2 + e * x + f = 0)$
using *case-prodE* **by** *auto*
have $\neg ?f5 \wedge \neg ?f6 \wedge \neg ?f7 \wedge \neg ?f8 \wedge \neg ?f13 \wedge \neg ?f9 \wedge \neg ?f10 \wedge \neg ?f11 \wedge \neg ?f12$
using *big-not* **by** *auto*
then have $\neg(\exists x.\ (\forall (a,\ b,\ c) \in set\ b.\ a * x^2 + b * x + c < 0) \wedge$
$(\forall (a,\ b,\ c) \in set\ c.\ a * x^2 + b * x + c \leq 0) \wedge$
$(\forall (a,\ b,\ c) \in set\ d.\ a * x^2 + b * x + c \neq 0))$
using *equalset big-not qe-forwards-helper*[*of a b c d*] **by** *auto*
then show *False*
using *x-prop* **by** *auto*
qed
have *eo*: $(\exists\ (d,\ e,\ f) \in set\ a.\ d \neq 0 \wedge -\ e^2 + 4 * d * f \leq 0) \vee (\exists\ (d,\ e,\ f) \in set\ a.\ d = 0 \wedge e \neq 0) \vee (\forall\ (d,\ e,\ f) \in set\ a.\ d = 0 \wedge e = 0 \wedge f = 0)$
proof −
have $(\forall\ (d,\ e,\ f) \in set\ a.\ (d \neq 0 \longrightarrow\ -\ e^2 + 4 * d * f \leq 0))$
proof *clarsimp*
fix *d e f*
assume *in-set*: $(d,\ e,\ f) \in set\ a$
assume *dnonz*: $d \neq 0$
have $d*x\hat{}2 + e*x + f = 0$ **using** *in-set x-prop* **by** *auto*
then show $4 * d * f \leq e^2$
using *dnonz discriminant-negative*[*of d e f*] **unfolding** *discrim-def*
by *fastforce*
qed
then have *discrim-prop*: $\neg(\exists\ (d,\ e,\ f) \in set\ a.\ d \neq 0 \wedge -\ e^2 + 4 * d * f \leq 0) \Longrightarrow \neg(\exists\ (d,\ e,\ f) \in set\ a.\ d \neq 0)$
by *auto*
have $\neg(\exists\ (d,\ e,\ f) \in set\ a.\ d \neq 0) \wedge \neg(\exists\ (d,\ e,\ f) \in set\ a.\ d = 0 \wedge e \neq 0) \Longrightarrow (\forall\ (d,\ e,\ f) \in set\ a.\ d = 0 \wedge e = 0 \wedge f = 0)$
proof −
assume *ne*: $\neg(\exists\ (d,\ e,\ f) \in set\ a.\ d \neq 0) \wedge \neg(\exists\ (d,\ e,\ f) \in set\ a.\ d = 0 \wedge e \neq 0)$
show $(\forall\ (d,\ e,\ f) \in set\ a.\ d = 0 \wedge e = 0 \wedge f = 0)$
proof *clarsimp*
fix *d e f*
assume *in-set*: $(d,\ e,\ f) \in set\ a$
then have *xzer*: $d*x\hat{}2 + e*x + f = 0$ **using** *x-prop* **by** *auto*
have *dzer*: $d = 0$ **using** *ne in-set* **by** *auto*
have *ezer*: $e = 0$ **using** *ne in-set* **by** *auto*
show $d = 0 \wedge e = 0 \wedge f = 0$ **using** *xzer dzer ezer* **by** *auto*
qed
qed

then show *?thesis* **using** *discrim-prop* **by** *auto*
qed
show *False* **using** *c1 c2 c3 eo* **by** *auto*
qed
then have $\neg ?e2 \Longrightarrow False$ **using** *bigor-var not-eq*
by *presburger*
then have $\neg ?e2 \longrightarrow False$ **using** *impI*[*of* $\neg ?e2$ *False*]
by *blast*
then show *?thesis*
by *auto*
qed

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−
have h: $-(c/b) = (sqrt\ (aa^2 - 4 * a * ba) - aa)\ /\ (2 * a)$
using *assms*
by (*smt divide-minus-left nonzero-mult-div-cancel-left times-divide-eq-right*)
have $h1 : \forall\, x \in set\ eq.\ case\ x\ of\ (d,\ e,\ f) \Rightarrow d * (c\ /\ b)^2 + e * - (c\ /\ b) + f = 0$
using *assms*(*5*) **unfolding** *h*[*symmetric*] *Fields.division-ring-class.times-divide-eq-right*[*symmetric*]
Power.ring-1-class.power2-minus **.**
show *?thesis*
using *bspec*[*OF h1 assms*(*6*)] **by** *simp*
qed

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−
have *h*: $-(c/b) = (sqrt\ (aa^2 - 4 * a * ba) - aa) / (2 * a)$
using *assms*
by (*smt divide-minus-left nonzero-mult-div-cancel-left times-divide-eq-right*)
have $h1 : \forall x \in set\ les.\ case\ x\ of\ (d, e, f) \Rightarrow d * (c / b)^2 + e * - (c / b) + f < 0$
using *assms*(5) **unfolding** *h*[*symmetric*] *Fields.division-ring-class.times-divide-eq-right*[*symmetric*]
Power.ring-1-class.power2-minus **.**
show *?thesis*
using *bspec*[*OF h1 assms*(6)] **by** *simp*
qed

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−
have *h*: $-((c::real)/(b::real)) = (- (aa::real) - sqrt\ (aa^2 - 4 * (a::real) * (ba::real))) / (2 * a)$
using *assms*
by (*smt divide-minus-left nonzero-mult-div-cancel-left times-divide-eq-right*)
have $h1 : \forall x \in set\ eq.\ case\ x\ of\ (d, e, f) \Rightarrow d * (c / b)^2 + e * - (c / b) + f = 0$
using *assms*(5) **unfolding** *h*[*symmetric*] *Fields.division-ring-class.times-divide-eq-right*[*symmetric*]
Power.ring-1-class.power2-minus **.**
show *?thesis*
using *bspec*[*OF h1 assms*(6)] **by** *simp*
qed

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−
have h: $-((c{::}real)/(b{::}real)) = (-\ (aa{::}real)\ -sqrt\ (aa^2 - 4 * (a{::}real) * (ba{::}real)))\ /\ (2 * a)$
using $assms$
by $(smt\ divide\text{-}minus\text{-}left\ nonzero\text{-}mult\text{-}div\text{-}cancel\text{-}left\ times\text{-}divide\text{-}eq\text{-}right)$
have $h1 : \forall x{\in}set\ les.\ case\ x\ of\ (d,\ e,\ f) \Rightarrow d * (c\ /\ b)^2 + e * -\ (c\ /\ b) + f < 0$
using $assms(5)$ **unfolding** $h[symmetric]\ Fields.division\text{-}ring\text{-}class.times\text{-}divide\text{-}eq\text{-}right[symmetric]$
$Power.ring\text{-}1\text{-}class.power2\text{-}minus$ **.**
show $?thesis$
using $bspec[OF\ h1\ assms(6)]$ **by** $simp$
qed

lemma $h1b\text{-}helper\text{-}les$:
$(\forall ((a{::}real),\ (b{::}real),\ (c{::}real)){\in}set\ les.\ \exists x.\ \forall y{<}x.\ a * y^2 + b * y + c < 0) \Longrightarrow (\exists y.\forall x{<}y.\ (\forall (a,\ b,\ c){\in}set\ les.\ a * x^2 + b * x + c < 0))$
proof −
show $(\forall (a,\ b,\ c){\in}set\ les.\ \exists x.\ \forall y{<}x.\ a * y^2 + b * y + c < 0) \Longrightarrow (\exists y.\forall x{<}y.\ (\forall (a,\ b,\ c){\in}set\ les.\ a * x^2 + b * x + c < 0))$
proof $(induct\ les)$
case Nil
then show $?case$
by $auto$
next
case $(Cons\ q\ les)$
have ind: $\forall a{\in}set\ (q\ \#\ les).\ case\ a\ of\ (a,\ ba,\ c) \Rightarrow \exists x.\ \forall y{<}x.\ a * y^2 + ba * y + c < 0$
using $Cons.prems$
by $auto$
then have $case\ q\ of\ (a,\ ba,\ c) \Rightarrow \exists x.\ \forall y{<}x.\ a * y^2 + ba * y + c < 0$
by $simp$
then obtain $y2$ **where** $y2\text{-}prop$: $case\ q\ of\ (a,\ ba,\ c) \Rightarrow\ (\forall y{<}y2.\ a * y^2 + ba * y + c < 0)$
by $auto$
have $\forall a{\in}set\ les.\ case\ a\ of\ (a,\ ba,\ c) \Rightarrow \exists x.\ \forall y{<}x.\ a * y^2 + ba * y + c < 0$
using ind **by** $simp$
then have $\exists y.\ \forall x{<}y.\ \forall a{\in}set\ les.\ case\ a\ of\ (a,\ ba,\ c) \Rightarrow a * x^2 + ba * x + c < 0$
using $Cons.hyps$ **by** $blast$
then obtain $y1$ **where** $y1\text{-}prop$: $\forall x{<}y1.\ \forall a{\in}set\ les.\ case\ a\ of\ (a,\ ba,\ c) \Rightarrow a$

$* x\hat{}2 + ba * x + c < 0$
by *blast*
let *?y* = *min y1 y2*
have $\forall\, x < ?y.\ (\forall\, a{\in}set\ (q\ \#les).\ case\ a\ of\ (a,\ ba,\ c) \Rightarrow a * x\hat{}2 + ba * x + c < 0)$
using *y1-prop y2-prop*
by *fastforce*
then show *?case*
by *blast*
qed
qed

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 −
show $(\forall\, (a,\ b,\ c){\in}set\ leq.\ \exists\, x.\ \forall\, y{<}x.\ a * y^2 + b * y + c \leq 0) \Longrightarrow (\exists\, y.\forall\, x{<}y.\ (\forall\, (a,\ b,\ c){\in}set\ leq.\ a * x^2 + b * x + c \leq 0))$
proof (*induct leq*)
case *Nil*
then show *?case*
by *auto*
next
case (*Cons q leq*)
have *ind*: $\forall\, a{\in}set\ (q\ \#\ leq).\ case\ a\ of\ (a,\ ba,\ c) \Rightarrow \exists\, x.\ \forall\, y{<}x.\ a * y^2 + ba * y + c \leq 0$
using *Cons.prems*
by *auto*
then have $case\ q\ of\ (a,\ ba,\ c) \Rightarrow \exists\, x.\ \forall\, y{<}x.\ a * y^2 + ba * y + c \leq 0$
by *simp*
then obtain *y2* **where** *y2-prop*: $case\ q\ of\ (a,\ ba,\ c) \Rightarrow\ (\forall\, y{<}y2.\ a * y^2 + ba * y + c \leq 0)$
by *auto*
have $\forall\, a{\in}set\ leq.\ case\ a\ of\ (a,\ ba,\ c) \Rightarrow \exists\, x.\ \forall\, y{<}x.\ a * y^2 + ba * y + c \leq 0$
using *ind* **by** *simp*
then have $\exists\, y.\ \forall\, x{<}y.\ \forall\, a{\in}set\ leq.\ case\ a\ of\ (a,\ ba,\ c) \Rightarrow a * x^2 + ba * x + c \leq 0$
using *Cons.hyps* **by** *blast*
then obtain *y1* **where** *y1-prop*: $\forall\, x{<}y1.\ \forall\, a{\in}set\ leq.\ case\ a\ of\ (a,\ ba,\ c) \Rightarrow a * x\hat{}2 + ba * x + c \leq 0$
by *blast*
let *?y* = *min y1 y2*
have $\forall\, x < ?y.\ (\forall\, a{\in}set\ (q\ \#leq).\ case\ a\ of\ (a,\ ba,\ c) \Rightarrow a * x\hat{}2 + ba * x + c \leq 0)$
using *y1-prop y2-prop*
by *fastforce*
then show *?case*
by *blast*
qed

qed

lemma *h1b-helper-neq*:
(∀((*a*::*real*), (*b*::*real*), (*c*::*real*))∈*set neq*. ∃*x*. ∀*y*<*x*. *a* ∗ *y*^{2} + *b* ∗ *y* + *c* ≠ *0*) ⟹ (∃*y*.∀*x*<*y*. (∀(*a*, *b*, *c*)∈*set neq*. *a* ∗ *x*^{2} + *b* ∗ *x* + *c* ≠ *0*))
proof −
show (∀(*a*, *b*, *c*)∈*set neq*. ∃*x*. ∀*y*<*x*. *a* ∗ *y*^{2} + *b* ∗ *y* + *c* ≠ *0*) ⟹ (∃*y*.∀*x*<*y*. (∀(*a*, *b*, *c*)∈*set neq*. *a* ∗ *x*^{2} + *b* ∗ *x* + *c* ≠ *0*))
proof (*induct neq*)
case *Nil*
then show *?case*
by *auto*
next
case (*Cons q neq*)
have *ind*: ∀*a*∈*set* (*q* # *neq*). *case a of* (*a*, *ba*, *c*) ⇒ ∃*x*. ∀*y*<*x*. *a* ∗ *y*^{2} + *ba* ∗ *y* + *c* ≠ *0*
using *Cons.prems*
by *auto*
then have *case q of* (*a*, *ba*, *c*) ⇒ ∃*x*. ∀*y*<*x*. *a* ∗ *y*^{2} + *ba* ∗ *y* + *c* ≠ *0*
by *simp*
then obtain *y2* **where** *y2-prop*: *case q of* (*a*, *ba*, *c*) ⇒ (∀*y*<*y2*. *a* ∗ *y*^{2} + *ba* ∗ *y* + *c* ≠ *0*)
by *auto*
have ∀*a*∈*set neq*. *case a of* (*a*, *ba*, *c*) ⇒ ∃*x*. ∀*y*<*x*. *a* ∗ *y*^{2} + *ba* ∗ *y* + *c* ≠ *0*
using *ind* **by** *simp*
then have ∃*y*. ∀*x*<*y*. ∀*a*∈*set neq*. *case a of* (*a*, *ba*, *c*) ⇒ *a* ∗ *x*^{2} + *ba* ∗ *x* + *c* ≠ *0*
using *Cons.hyps* **by** *blast*
then obtain *y1* **where** *y1-prop*: ∀*x*<*y1*. ∀*a*∈*set neq*. *case a of* (*a*, *ba*, *c*) ⇒ *a* ∗ *x^2* + *ba* ∗ *x* + *c* ≠ *0*
by *blast*
let *?y* = *min y1 y2*
have ∀*x* < *?y*. (∀*a*∈*set* (*q* #*neq*). *case a of* (*a*, *ba*, *c*) ⇒ *a* ∗ *x^2* + *ba* ∗ *x* + *c* ≠ *0*)
using *y1-prop y2-prop*
by *fastforce*
then show *?case*
by *blast*
qed
qed

lemma *min-lem*:
fixes *r*::*real*
assumes *a1*: (∃*y'*>*r*. (∀((*d*::*real*), (*e*::*real*), (*f*::*real*))∈*set b*. ∀*x*∈{*r*<..*y'*}. *d* ∗ *x*^{2} + *e* ∗ *x* + *f* < *0*))
assumes *a2*: (∃*y'*>*r*. (∀((*d*::*real*), (*e*::*real*), (*f*::*real*))∈*set c*. ∀*x*∈{*r*<..*y'*}. *d* ∗ *x*^{2} + *e* ∗ *x* + *f* ≤ *0*))
assumes *a3*: (∃*y'*>*r*. (∀((*d*::*real*), (*e*::*real*), (*f*::*real*))∈*set d*. ∀*x*∈{*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 −
obtain *y1* **where** *y1-prop*: $y1 > r \wedge$ ($\forall$ (d, e, f)$\in$*set* b. $\forall x \in \{r<..y1\}$. $d * x^2 + e * x + f < 0$)
using *a1* **by** *auto*
obtain *y2* **where** *y2-prop*: $y2 > r \wedge$ ($\forall$ (d, e, f)$\in$*set* c. $\forall x \in \{r<..y2\}$. $d * x^2 + e * x + f \leq 0$)
using *a2* **by** *auto*
obtain *y3* **where** *y3-prop*: $y3 > r \wedge$ ($\forall$ (d, e, f)$\in$*set* d. $\forall x \in \{r<..y3\}$. $d * x^2 + e * x + f \neq 0$)
using *a3* **by** *auto*
let *?y* = (*min* (*min y1 y2*) *y3*)
have *?y* $> r$ **using** *y1-prop y2-prop y3-prop* **by** *auto*
then have $\exists x.\ x > r \wedge x <$ *?y* **using** *dense*[*of r ?y*]
by *auto*
then obtain x **where** *x-prop*: $x > r \wedge x <$ *?y* **by** *auto*
have *bp*: ($\forall$ (a, b, c)$\in$*set* b. $a * x^2 + b * x + c < 0$)
using *x-prop y1-prop* **by** *auto*
have *cp*: ($\forall$ (a, b, c)$\in$*set* c. $a * x\hat{}2 + b * x + c \leq 0$)
using *x-prop y2-prop* **by** *auto*
have *dp*: ($\forall$ (a, b, c)$\in$*set* d. $a * x^2 + b * x + c \neq 0$)
using *x-prop y3-prop* **by** *auto*
then have ($\forall$ (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. $a * x^2 + b * x + c \neq 0$)
using *bp cp dp* **by** *auto*
then show *?thesis* **by** *auto*
qed

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 −
have $\forall$ (d, e, f)$\in$*set* a. $d = 0 \wedge e = 0 \wedge f = 0$
proof *clarsimp*
fix d e f
assume (d, e, f) $\in$ *set* a
then have $\exists y' > k$. $\forall x \in \{k<..y'\}$. $d * x^2 + e * x + f = 0$

using *asm* **by** *auto*
then obtain y' **where** *y-prop*: $y'>k \wedge (\forall x \in \{k<..y'\}.\ d * x^2 + e * x + f = 0)$
by *auto*
then show $d = 0 \wedge e = 0 \wedge f = 0$ **using** *continuity-lem-eq0*[*of k y' d e f*]
by *auto*
qed
then have *eqprop*: $\forall x.\ (\forall (a,\ b,\ c) \in set\ a.\ a * x^2 + b * x + c = 0)$
by *auto*
have *lesprop*: $(\exists y'>k.\ (\forall (d,\ e,\ f) \in set\ b.\ \forall x \in \{k<..y'\}.\ d * x^2 + e * x + f < 0))$

using *les-qe-inf-helper*[*of b k*] *asm*
by *blast*
have *leqprop*: $(\exists y'>k.\ (\forall (d,\ e,\ f) \in set\ c.\ \forall x \in \{(k)<..y'\}.\ d * x^2 + e * x + f \leq 0))$
using *leq-qe-inf-helper*[*of c k*] *asm*
by *blast*
have *neqprop*: $(\exists y'>(k).\ (\forall (d,\ e,\ f) \in set\ d.\ \forall x \in \{(k)<..y'\}.\ d * x^2 + e * x + f \neq 0))$
using *neq-qe-inf-helper*[*of d k*] *asm*
by *blast*
then have $(\exists x.\ (\forall (a,\ b,\ c) \in set\ b.\ a * x^2 + b * x + c < 0) \wedge$
$(\forall (a,\ b,\ c) \in set\ c.\ a * x^2 + b * x + c \leq 0) \wedge$
$(\forall (a,\ b,\ c) \in set\ d.\ a * x^2 + b * x + c \neq 0))$
using *lesprop leqprop neqprop min-lem*[*of k b c d*]
by *auto*
then show *?thesis*
using *eqprop* **by** *auto*
qed

2.8.6 The qe_backwards lemma

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

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

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

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

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

$\vee$

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

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 $-$

let *?e2* $= (((\forall (a,\ b,\ c){\in}set\ a.\ a = 0 \wedge b = 0 \wedge c = 0) \wedge$
$(\forall (a,\ b,\ c){\in}set\ b.\ \exists x.\ \forall y{<}x.\ a * y^2 + b * y + c < 0) \wedge$
$(\forall (a,\ b,\ c){\in}set\ c.\ \exists x.\ \forall y{<}x.\ a * y^2 + b * y + c \leq 0) \wedge$
$(\forall (a,\ b,\ c){\in}set\ d.\ \exists x.\ \forall y{<}x.\ a * y^2 + b * y + c \neq 0)$
$\vee$
$(\exists (a',\ b',\ c'){\in}set\ a.$
$(a' = 0 \wedge b' \neq 0) \wedge$
$(\forall (d,\ e,\ f){\in}set\ a.\ d * (-\ c' / b')^2 + e * (-\ c' / b') + f = 0) \wedge$
$(\forall (d,\ e,\ f){\in}set\ b.\ d * (-\ c' / b')^2 + e * (-\ c' / b') + f < 0) \wedge$
$(\forall (d,\ e,\ f){\in}set\ c.\ d * (-\ c' / b')^2 + e * (-\ c' / b') + f \leq 0) \wedge$
$(\forall (d,\ e,\ f){\in}set\ d.\ d * (-\ c' / b')^2 + e * (-\ c' / b') + f \neq 0)$
$\vee$
$a' \neq 0 \wedge$
$-\ b'^2 + 4 * a' * c' \leq 0 \wedge$
$((\forall (d,\ e,\ f){\in}set\ a.$
$d * ((-\ b' + 1 * sqrt\ (b'^2 - 4 * a' * c')) / (2 * a'))^2 +$
$e * ((-\ b' + 1 * sqrt\ (b'^2 - 4 * a' * c')) / (2 * a')) +$
$f =$
$0) \wedge$
$(\forall (d,\ e,\ f){\in}set\ b.$
$d * ((-\ b' + 1 * sqrt\ (b'^2 - 4 * a' * c')) / (2 * a'))^2 +$
$e * ((-\ b' + 1 * sqrt\ (b'^2 - 4 * a' * c')) / (2 * a')) +$
$f < 0) \wedge$
$(\forall (d,\ e,\ f){\in}set\ c.$
$d * ((-\ b' + 1 * sqrt\ (b'^2 - 4 * a' * c')) / (2 * a'))^2 +$

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

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

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

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

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

let *?f10orf11orf12* $=\ (\exists\,(a',\ b',\ c')\in set\ d.$
$(a'\ =\ 0\ \wedge\ b'\ \neq\ 0)\ \wedge$
$(\forall\,(d,\ e,\ f)\in set\ a.$
$\exists\,y'>-\ c'\ /\ b'.\ \forall\,x\in\{-\ c'\ /\ b'<..y'\}.\ d\ *\ x^2\ +\ e\ *\ x\ +\ f\ =\ 0)\ \wedge$
$(\forall\,(d,\ e,\ f)\in set\ b.$
$\exists\,y'>-\ c'\ /\ b'.\ \forall\,x\in\{-\ c'\ /\ b'<..y'\}.\ d\ *\ x^2\ +\ e\ *\ x\ +\ f\ <\ 0)\ \wedge$
$(\forall\,(d,\ e,\ f)\in set\ c.$
$\exists\,y'>-\ c'\ /\ b'.\ \forall\,x\in\{-\ c'\ /\ b'<..y'\}.\ d\ *\ x^2\ +\ e\ *\ x\ +\ f\ \leq\ 0)\ \wedge$
$(\forall\,(d,\ e,\ f)\in set\ d.$
$\exists\,y'>-\ c'\ /\ b'.\ \forall\,x\in\{-\ c'\ /\ b'<..y'\}.\ d\ *\ x^2\ +\ e\ *\ x\ +\ f\ \neq\ 0)$
$\vee$
$a'\ \neq\ 0\ \wedge$
$-\ b'^2\ +\ 4\ *\ a'\ *\ c'\ \leq\ 0\ \wedge$
$((\forall\,(d,\ e,\ f)\in set\ a.$
$\exists\,y'>(-\ b'\ +\ 1\ *\ sqrt\ (b'^2\ -\ 4\ *\ a'\ *\ c'))\ /\ (2\ *\ a').$
$\forall\,x\in\{(-\ b'\ +\ 1\ *\ sqrt\ (b'^2\ -\ 4\ *\ a'\ *\ c'))\ /\ (2\ *\ a')<..y'\}.$
$d\ *\ x^2\ +\ e\ *\ x\ +\ f\ =\ 0)\ \wedge$
$(\forall\,(d,\ e,\ f)\in set\ b.$
$\exists\,y'>(-\ b'\ +\ 1\ *\ sqrt\ (b'^2\ -\ 4\ *\ a'\ *\ c'))\ /\ (2\ *\ a').$
$\forall\,x\in\{(-\ b'\ +\ 1\ *\ sqrt\ (b'^2\ -\ 4\ *\ a'\ *\ c'))\ /\ (2\ *\ a')<..y'\}.$
$d\ *\ x^2\ +\ e\ *\ x\ +\ f\ <\ 0)\ \wedge$
$(\forall\,(d,\ e,\ f)\in set\ c.$
$\exists\,y'>(-\ b'\ +\ 1\ *\ sqrt\ (b'^2\ -\ 4\ *\ a'\ *\ c'))\ /\ (2\ *\ a').$
$\forall\,x\in\{(-\ b'\ +\ 1\ *\ sqrt\ (b'^2\ -\ 4\ *\ a'\ *\ c'))\ /\ (2\ *\ a')<..y'\}.$
$d\ *\ x^2\ +\ e\ *\ x\ +\ f\ \leq\ 0)\ \wedge$
$(\forall\,(d,\ e,\ f)\in set\ d.$
$\exists\,y'>(-\ b'\ +\ 1\ *\ sqrt\ (b'^2\ -\ 4\ *\ a'\ *\ c'))\ /\ (2\ *\ a').$
$\forall\,x\in\{(-\ b'\ +\ 1\ *\ sqrt\ (b'^2\ -\ 4\ *\ a'\ *\ c'))\ /\ (2\ *\ a')<..y'\}.$
$d\ *\ x^2\ +\ e\ *\ x\ +\ f\ \neq\ 0)$
$\vee$
$(\forall\,(d,\ e,\ f)\in set\ a.$
$\exists\,y'>(-\ b'\ +\ -\ 1\ *\ sqrt\ (b'^2\ -\ 4\ *\ a'\ *\ c'))\ /\ (2\ *\ a').$
$\forall\,x\in\{(-\ b'\ +\ -\ 1\ *\ sqrt\ (b'^2\ -\ 4\ *\ a'\ *\ c'))\ /\ (2\ *\ a')<..y'\}.$
$d\ *\ x^2\ +\ e\ *\ x\ +\ f\ =\ 0)\ \wedge$
$(\forall\,(d,\ e,\ f)\in set\ b.$
$\exists\,y'>(-\ b'\ +\ -\ 1\ *\ sqrt\ (b'^2\ -\ 4\ *\ a'\ *\ c'))\ /\ (2\ *\ a').$
$\forall\,x\in\{(-\ b'\ +\ -\ 1\ *\ sqrt\ (b'^2\ -\ 4\ *\ a'\ *\ c'))\ /\ (2\ *\ a')<..y'\}.$

$$
\begin{array}{l}
\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 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 d * x^2 + e * x + f \neq 0))) \\
\textbf{let}\ ?f8orf9 = (\exists (a',\ b',\ c') \in set\ c. \\
\quad (a' = 0 \wedge b' \neq 0) \wedge \\
\quad (\forall (d,\ e,\ f) \in set\ a.\ d * (- c' \ /\ b')^2 + e * (- c' \ /\ b') + f = 0) \wedge \\
\quad (\forall (d,\ e,\ f) \in set\ b.\ d * (- c' \ /\ b')^2 + e * (- c' \ /\ b') + f < 0) \wedge \\
\quad (\forall (d,\ e,\ f) \in set\ c.\ d * (- c' \ /\ b')^2 + e * (- c' \ /\ b') + f \leq 0) \wedge \\
\quad (\forall (d,\ e,\ f) \in set\ d.\ d * (- c' \ /\ b')^2 + e * (- c' \ /\ b') + f \neq 0) \vee \\
\quad a' \neq 0 \wedge \\
\quad - b'^2 + 4 * a' * c' \leq 0 \wedge \\
\quad ((\forall (d,\ e,\ f) \in set\ a. \\
\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 = 0) \wedge \\
\quad (\forall (d,\ e,\ f) \in set\ b. \\
\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 < 0) \wedge \\
\quad (\forall (d,\ e,\ f) \in set\ c. \\
\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 \leq 0) \wedge \\
\quad (\forall (d,\ e,\ f) \in set\ d. \\
\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 0) \\
\vee \\
\quad (\forall (d,\ e,\ f) \in set\ a. \\
\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 = 0) \wedge \\
\quad (\forall (d,\ e,\ f) \in set\ b. \\
\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 < 0) \wedge \\
\quad (\forall (d,\ e,\ f) \in set\ c. \\
\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 \leq 0) \wedge \\
\quad (\forall (d,\ e,\ f) \in set\ d. \\
\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')) +
\end{array}
$$

$f \neq 0)))$

let *?f5orf6orf7* $= (\exists\,(a',\ b',\ c') \in set\ b.$

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

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

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

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

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

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

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

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

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

$\vee$

$a' \neq 0 \wedge$

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

$\vee$

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

let *?f2orf3orf4* $= (\exists\,(a',\ b',\ c') \in set\ a.$

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

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

let *?e1* $= (\exists\, x.\ (\forall\,(a,\ b,\ c)\in set\ a.\ a * x^2 + b * x + c = 0) \wedge$
$(\forall\,(a,\ b,\ c)\in set\ b.\ a * x^2 + b * x + c < 0) \wedge$
$(\forall\,(a,\ b,\ c)\in set\ c.\ a * x^2 + b * x + c \leq 0) \wedge$
$(\forall\,(a,\ b,\ c)\in set\ d.\ a * x^2 + b * x + c \neq 0))$

let *?f1* $= ((\forall\,(a,\ b,\ c)\in set\ a.\ a = 0 \wedge b = 0 \wedge c = 0) \wedge$
$(\forall\,(a,\ b,\ c)\in set\ b.\ \exists\, x.\ \forall\, y{<}x.\ a * y^2 + b * y + c < 0) \wedge$
$(\forall\,(a,\ b,\ c)\in set\ c.\ \exists\, x.\ \forall\, y{<}x.\ a * y^2 + b * y + c \leq 0) \wedge$
$(\forall\,(a,\ b,\ c)\in set\ d.\ \exists\, x.\ \forall\, y{<}x.\ a * y^2 + b * y + c \neq 0))$

let *?f2* $= (\exists\,(a',\ b',\ c')\in set\ a.$

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

let *?f3* $= (\exists (a',\ b',\ c') \in set\ a.\ a' \neq 0 \wedge -\ b'^2 + 4 * a' * c' \leq 0 \wedge$
$(\forall (d,\ e,\ f) \in set\ a.$
$d * ((-\ b' + 1 * sqrt\ (b'^2 - 4 * a' * c')) \ /\ (2 * a'))^2 +$
$e * ((-\ b' + 1 * sqrt\ (b'^2 - 4 * a' * c')) \ /\ (2 * a')) +$
$f = 0) \wedge$
$(\forall (d,\ e,\ f) \in set\ b.$
$d * ((-\ b' + 1 * sqrt\ (b'^2 - 4 * a' * c')) \ /\ (2 * a'))^2 +$
$e * ((-\ b' + 1 * sqrt\ (b'^2 - 4 * a' * c')) \ /\ (2 * a')) +$
$f < 0) \wedge$
$(\forall (d,\ e,\ f) \in set\ c.$
$d * ((-\ b' + 1 * sqrt\ (b'^2 - 4 * a' * c')) \ /\ (2 * a'))^2 +$
$e * ((-\ b' + 1 * sqrt\ (b'^2 - 4 * a' * c')) \ /\ (2 * a')) +$
$f \leq 0) \wedge$
$(\forall (d,\ e,\ f) \in set\ d.$
$d * ((-\ b' + 1 * sqrt\ (b'^2 - 4 * a' * c')) \ /\ (2 * a'))^2 +$
$e * ((-\ b' + 1 * sqrt\ (b'^2 - 4 * a' * c')) \ /\ (2 * a')) +$
$f \neq 0))$

let *?f4* $= (\exists (a',\ b',\ c') \in set\ a.\ a' \neq 0 \wedge -\ b'^2 + 4 * a' * c' \leq 0 \wedge$
$(\forall (d,\ e,\ f) \in set\ a.$
$d * ((-\ b' + -\ 1 * sqrt\ (b'^2 - 4 * a' * c')) \ /\ (2 * a'))^2 +$
$e * ((-\ b' + -\ 1 * sqrt\ (b'^2 - 4 * a' * c')) \ /\ (2 * a')) +$
$f = 0) \wedge$
$(\forall (d,\ e,\ f) \in set\ b.$
$d * ((-\ b' + -\ 1 * sqrt\ (b'^2 - 4 * a' * c')) \ /\ (2 * a'))^2 +$
$e * ((-\ b' + -\ 1 * sqrt\ (b'^2 - 4 * a' * c')) \ /\ (2 * a')) +$
$f < 0) \wedge$
$(\forall (d,\ e,\ f) \in set\ c.$
$d * ((-\ b' + -\ 1 * sqrt\ (b'^2 - 4 * a' * c')) \ /\ (2 * a'))^2 +$
$e * ((-\ b' + -\ 1 * sqrt\ (b'^2 - 4 * a' * c')) \ /\ (2 * a')) +$
$f \leq 0) \wedge$
$(\forall (d,\ e,\ f) \in set\ d.$
$d * ((-\ b' + -\ 1 * sqrt\ (b'^2 - 4 * a' * c')) \ /\ (2 * a'))^2 +$
$e * ((-\ b' + -\ 1 * sqrt\ (b'^2 - 4 * a' * c')) \ /\ (2 * a')) +$
$f \neq 0))$

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

$$
\begin{array}{l}
\textbf{let } ?f6 = (\exists\,(a',\ b',\ c')\in set\ b.\ 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))) \\
\textbf{let } ?f7 = (\exists\,(a',\ b',\ c')\in set\ b.\ a' \neq 0\ \wedge \\
\quad -\ b'^2 + 4 * a' * c' \leq 0\ \wedge\ (\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)) \\
\textbf{let } ?f8 = (\exists\,(a',\ b',\ c')\in set\ c.\ (a' = 0\ \wedge\ b' \neq 0)\ \wedge \\
\quad (\forall\,(d,\ e,\ f)\in set\ a.\ d * (-\ c'\ /\ b')^2 + e * (-\ c'\ /\ b') + f = 0)\ \wedge \\
\quad (\forall\,(d,\ e,\ f)\in set\ b.\ d * (-\ c'\ /\ b')^2 + e * (-\ c'\ /\ b') + f < 0)\ \wedge \\
\quad (\forall\,(d,\ e,\ f)\in set\ c.\ d * (-\ c'\ /\ b')^2 + e * (-\ c'\ /\ b') + f \leq 0)\ \wedge \\
\quad (\forall\,(d,\ e,\ f)\in set\ d.\ d * (-\ c'\ /\ b')^2 + e * (-\ c'\ /\ b') + f \neq 0)) \\
\textbf{let } ?f13 = (\exists\,(a',\ b',\ c')\in set\ c. \\
\quad a' \neq 0\ \wedge \\
\quad -\ b'^2 + 4 * a' * c' \leq 0\ \wedge \\
\quad ((\forall\,(d,\ e,\ f)\in set\ a. \\
\qquad d * ((-\ b' + 1 * sqrt\ (b'^2 - 4 * a' * c'))\ /\ (2 * a'))^2 + \\
\qquad e * ((-\ b' + 1 * sqrt\ (b'^2 - 4 * a' * c'))\ /\ (2 * a')) + \\
\qquad f = 0)\ \wedge \\
\quad (\forall\,(d,\ e,\ f)\in set\ b. \\
\qquad d * ((-\ b' + 1 * sqrt\ (b'^2 - 4 * a' * c'))\ /\ (2 * a'))^2 +
\end{array}
$$

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

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

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

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

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

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

have *h1a*: *?e2* $\longrightarrow$ (*?f1* $\vee$ *?f2orf3orf4* $\vee$ *?f5orf6orf7* $\vee$ *?f8orf9* $\vee$ *?f10orf11orf12*)
by *auto*

have *h2*: *?f2orf3orf4* $\longrightarrow$ (*?f2* $\vee$ *?f3* $\vee$ *?f4*) **by** *auto*

then have *h1b*: *?e2* $\longrightarrow$ (*?f1* $\vee$ *?f2* $\vee$ *?f3* $\vee$ *?f4* $\vee$ *?f5orf6orf7* $\vee$ *?f8orf9* $\vee$ *?f10orf11orf12*)
using *h1a* **by** *auto*

have *h3*: *?f5orf6orf7* $\longrightarrow$ (*?f5* $\vee$ *?f6* $\vee$ *?f7*) **by** *auto*

then have *h1c*: *?e2* $\longrightarrow$ (*?f1* $\vee$ *?f2* $\vee$ *?f3* $\vee$ *?f4* $\vee$ *?f5* $\vee$ *?f6* $\vee$ *?f7* $\vee$ *?f8orf9* $\vee$ *?f10orf11orf12*)
using *h1b* **by** *smt*

have *h4*: *?f8orf9* $\longrightarrow$ (*?f8* $\vee$ *?f9* $\vee$ *?f13*) **by** *auto*

then have *h1d*: *?e2* $\longrightarrow$ (*?f1* $\vee$ *?f2* $\vee$ *?f3* $\vee$ *?f4* $\vee$ *?f5* $\vee$ *?f6* $\vee$ *?f7* $\vee$ *?f8* $\vee$ *?f9* $\vee$ *?f13* $\vee$ *?f10orf11orf12*)
using *h1c*
by *smt*

have *h5*: *?f10orf11orf12* $\longrightarrow$ (*?f10* $\vee$ *?f11* $\vee$ *?f12*)
by *auto*

then have *bigor*: *?e2* $\longrightarrow$ (*?f1* $\vee$ *?f2* $\vee$ *?f3* $\vee$ *?f4* $\vee$ *?f5* $\vee$ *?f6* $\vee$ *?f7* $\vee$ *?f8* $\vee$ *?f13* $\vee$ *?f9* $\vee$ *?f10* $\vee$ *?f11* $\vee$ *?f12*)
using *h1d* **by** *smt*

have *?f1* $\Longrightarrow$ *?e1*

proof −

assume *asm*: $(\forall (a,\ b,\ c) \in set\ a.\ a = 0 \wedge b = 0 \wedge c = 0) \wedge$

$(\forall\,(a,\ b,\ c)\in set\ b.\ \exists\,x.\ \forall\,y<x.\ a * y^2 + b * y + c < 0) \wedge$
$(\forall\,(a,\ b,\ c)\in set\ c.\ \exists\,x.\ \forall\,y<x.\ a * y^2 + b * y + c \leq 0) \wedge$
$(\forall\,(a,\ b,\ c)\in set\ d.\ \exists\,x.\ \forall\,y<x.\ a * y^2 + b * y + c \neq 0)$
then have *eqprop*: $\forall\,x.\ \forall\,(a,\ b,\ c)\in set\ a.\ a * x^2 + b * x + c = 0$ **by** *auto*
have $\exists\,y.\ \forall\,x<y.\ \forall\,(a,\ b,\ c)\in set\ b.\ a * x^2 + b * x + c < 0$
using *asm h1b-helper-les* **by** *auto*
then obtain *y1* **where** *y1-prop*: $\forall\,x<y1.\ \forall\,(a,\ b,\ c)\in set\ b.\ a * x^2 + b * x + c < 0$ **by** *auto*
have $\exists\,y.\ \forall\,x<y.\ \forall\,(a,\ b,\ c)\in set\ c.\ a * x^2 + b * x + c \leq 0$
using *asm h1b-helper-leq* **by** *auto*
then obtain *y2* **where** *y2-prop*: $\forall\,x<y2.\ \forall\,(a,\ b,\ c)\in set\ c.\ a * x^2 + b * x + c \leq 0$ **by** *auto*
have $\exists\,y.\ \forall\,x<y.\ \forall\,(a,\ b,\ c)\in set\ d.\ a * x^2 + b * x + c \neq 0$
using *asm h1b-helper-neq* **by** *auto*
then obtain *y3* **where** *y3-prop*: $\forall\,x<y3.\ \forall\,(a,\ b,\ c)\in set\ d.\ a * x^2 + b * x + c \neq 0$ **by** *auto*
let *?y* = $(min\ (min\ y1\ y2)\ y3) - 1$
have *y-prop*: $?y < y1 \wedge ?y < y2 \wedge ?y < y3$
by *auto*
have *ap*: $(\forall\,(a,\ b,\ c)\in set\ a.\ a * ?y^2 + b * ?y + c = 0)$
using *eqprop* **by** *auto*
have *bp*: $(\forall\,(a,\ b,\ c)\in set\ b.\ a * ?y^2 + b * ?y + c < 0)$
using *y-prop y1-prop* **by** *auto*
have *cp*: $(\forall\,(a,\ b,\ c)\in set\ c.\ a * ?y^2 + b * ?y + c \leq 0)$
using *y-prop y2-prop* **by** *auto*
have *dp*: $(\forall\,(a,\ b,\ c)\in set\ d.\ a * ?y^2 + b * ?y + c \neq 0)$
using *y-prop y3-prop* **by** *auto*
then have $(\forall\,(a,\ b,\ c)\in set\ a.\ a * ?y^2 + b * ?y + c = 0) \wedge$
$(\forall\,(a,\ b,\ c)\in set\ b.\ a * ?y^2 + b * ?y + c < 0) \wedge$
$(\forall\,(a,\ b,\ c)\in set\ c.\ a * ?y^2 + b * ?y + c \leq 0) \wedge$
$(\forall\,(a,\ b,\ c)\in set\ d.\ a * ?y^2 + b * ?y + c \neq 0)$
using *ap bp cp dp* **by** *auto*
then show *?thesis* **by** *auto*
qed
then have *h1*: $?f1 \longrightarrow ?e1$
by *auto*
have $?f2 \Longrightarrow ?e1$
proof −
assume $\exists\,(a',\ b',\ c')\in set\ a.$
$(a' = 0 \wedge b' \neq 0) \wedge$
$(\forall\,(d,\ e,\ f)\in set\ a.\ d * (-\ c'\ /\ b')^2 + e * (-\ c'\ /\ b') + f = 0) \wedge$
$(\forall\,(d,\ e,\ f)\in set\ b.\ d * (-\ c'\ /\ b')^2 + e * (-\ c'\ /\ b') + f < 0) \wedge$
$(\forall\,(d,\ e,\ f)\in set\ c.\ d * (-\ c'\ /\ b')^2 + e * (-\ c'\ /\ b') + f \leq 0) \wedge$
$(\forall\,(d,\ e,\ f)\in set\ d.\ d * (-\ c'\ /\ b')^2 + e * (-\ c'\ /\ b') + f \neq 0)$
then obtain $a'\ b'\ c'$ **where** *abc-prop*: $(a',\ b',\ c')\in set\ a \wedge$
$(a' = 0 \wedge b' \neq 0) \wedge$
$(\forall\,(d,\ e,\ f)\in set\ a.\ d * (-\ c'\ /\ b')^2 + e * (-\ c'\ /\ b') + f = 0) \wedge$
$(\forall\,(d,\ e,\ f)\in set\ b.\ d * (-\ c'\ /\ b')^2 + e * (-\ c'\ /\ b') + f < 0) \wedge$
$(\forall\,(d,\ e,\ f)\in set\ c.\ d * (-\ c'\ /\ b')^2 + e * (-\ c'\ /\ b') + f \leq 0) \wedge$

$(\forall (d,\ e,\ f) \in set\ d.\ d * (-\ c' / b')^2 + e * (-\ c' / b') + f \neq 0)$
by *auto*
then have $\exists (x::real).\ x = -c'/b'$ **by** *auto*
then obtain x **where** *x-prop*: $x = -\ c' / b'$ **by** *auto*
then have $(\forall xa \in set\ a.\ case\ xa\ of\ (a,\ b,\ c) \Rightarrow a * x^2 + b * x + c = 0) \wedge$
$(\forall xa \in set\ b.\ case\ xa\ of\ (a,\ b,\ c) \Rightarrow a * x^2 + b * x + c < 0) \wedge$
$(\forall xa \in set\ c.\ case\ xa\ of\ (a,\ b,\ c) \Rightarrow a * x^2 + b * x + c \leq 0) \wedge$
$(\forall xa \in set\ d.\ case\ xa\ of\ (a,\ b,\ c) \Rightarrow a * x^2 + b * x + c \neq 0)$
using *abc-prop* **by** *auto*
then show *?thesis* **by** *auto*
qed
then have *h2*: *?f2* $\longrightarrow$ *?e1*
by *auto*
have *?f3* $\Longrightarrow$ *?e1*
proof −
assume $\exists (a',\ b',\ c') \in set\ a.$
$a' \neq 0 \wedge$
$-\ b'^2 + 4 * a' * c' \leq 0 \wedge$
$(\forall (d,\ e,\ f) \in set\ a.$
$d * ((-\ b' + 1 * sqrt\ (b'^2 - 4 * a' * c')) / (2 * a'))^2 +$
$e * ((-\ b' + 1 * sqrt\ (b'^2 - 4 * a' * c')) / (2 * a')) +$
$f = 0) \wedge$
$(\forall (d,\ e,\ f) \in set\ b.$
$d * ((-\ b' + 1 * sqrt\ (b'^2 - 4 * a' * c')) / (2 * a'))^2 +$
$e * ((-\ b' + 1 * sqrt\ (b'^2 - 4 * a' * c')) / (2 * a')) +$
$f < 0) \wedge$
$(\forall (d,\ e,\ f) \in set\ c.$
$d * ((-\ b' + 1 * sqrt\ (b'^2 - 4 * a' * c')) / (2 * a'))^2 +$
$e * ((-\ b' + 1 * sqrt\ (b'^2 - 4 * a' * c')) / (2 * a')) +$
$f \leq 0) \wedge$
$(\forall (d,\ e,\ f) \in set\ d.$
$d * ((-\ b' + 1 * sqrt\ (b'^2 - 4 * a' * c')) / (2 * a'))^2 +$
$e * ((-\ b' + 1 * sqrt\ (b'^2 - 4 * a' * c')) / (2 * a')) +$
$f \neq 0)$
then obtain $a'\ b'\ c'$ **where** *abc-prop*: $(a',\ b',\ c') \in set\ a \wedge$
$a' \neq 0 \wedge$
$-\ b'^2 + 4 * a' * c' \leq 0 \wedge$
$(\forall (d,\ e,\ f) \in set\ a.$
$d * ((-\ b' + 1 * sqrt\ (b'^2 - 4 * a' * c')) / (2 * a'))^2 +$
$e * ((-\ b' + 1 * sqrt\ (b'^2 - 4 * a' * c')) / (2 * a')) +$
$f = 0) \wedge$
$(\forall (d,\ e,\ f) \in set\ b.$
$d * ((-\ b' + 1 * sqrt\ (b'^2 - 4 * a' * c')) / (2 * a'))^2 +$
$e * ((-\ b' + 1 * sqrt\ (b'^2 - 4 * a' * c')) / (2 * a')) +$
$f < 0) \wedge$
$(\forall (d,\ e,\ f) \in set\ c.$
$d * ((-\ b' + 1 * sqrt\ (b'^2 - 4 * a' * c')) / (2 * a'))^2 +$
$e * ((-\ b' + 1 * sqrt\ (b'^2 - 4 * a' * c')) / (2 * a')) +$
$f \leq 0) \wedge$

$(\forall (d,\ e,\ f) \in set\ d.$
$d * ((-\ b' + 1 * sqrt\ (b'^2 - 4 * a' * c')) / (2 * a'))^2 +$
$e * ((-\ b' + 1 * sqrt\ (b'^2 - 4 * a' * c')) / (2 * a')) +$
$f \neq 0)$ **by** *auto*
then have $\exists (x::real).\ x = (-\ b' + 1 * sqrt\ (b'^2 - 4 * a' * c')) / (2 * a')$ **by** *auto*
then obtain x **where** *x-prop*: $x = (-\ b' + 1 * sqrt\ (b'^2 - 4 * a' * c')) / (2 * a')$ **by** *auto*
then have $(\forall (a,\ b,\ c) \in set\ a.\ a * x^2 + b * x + c = 0) \wedge$
$(\forall (a,\ b,\ c) \in set\ b.\ a * x^2 + b * x + c < 0) \wedge$
$(\forall (a,\ b,\ c) \in set\ c.\ a * x^2 + b * x + c \leq 0) \wedge$
$(\forall (a,\ b,\ c) \in set\ d.\ a * x^2 + b * x + c \neq 0)$ **using** *abc-prop* **by** *auto*
then show *?thesis* **by** *auto*
qed
then have *h3*: *?f3* $\longrightarrow$ *?e1*
by *auto*
have *?f4* $\Longrightarrow$ *?e1*
proof −
assume $\exists (a',\ b',\ c') \in set\ a.$
$a' \neq 0 \wedge$
$-\ b'^2 + 4 * a' * c' \leq 0 \wedge$
$(\forall (d,\ e,\ f) \in set\ a.$
$d * ((-\ b' + -\ 1 * sqrt\ (b'^2 - 4 * a' * c')) / (2 * a'))^2 +$
$e * ((-\ b' + -\ 1 * sqrt\ (b'^2 - 4 * a' * c')) / (2 * a')) +$
$f = 0) \wedge$
$(\forall (d,\ e,\ f) \in set\ b.$
$d * ((-\ b' + -\ 1 * sqrt\ (b'^2 - 4 * a' * c')) / (2 * a'))^2 +$
$e * ((-\ b' + -\ 1 * sqrt\ (b'^2 - 4 * a' * c')) / (2 * a')) +$
$f < 0) \wedge$
$(\forall (d,\ e,\ f) \in set\ c.$
$d * ((-\ b' + -\ 1 * sqrt\ (b'^2 - 4 * a' * c')) / (2 * a'))^2 +$
$e * ((-\ b' + -\ 1 * sqrt\ (b'^2 - 4 * a' * c')) / (2 * a')) +$
$f \leq 0) \wedge$
$(\forall (d,\ e,\ f) \in set\ d.$
$d * ((-\ b' + -\ 1 * sqrt\ (b'^2 - 4 * a' * c')) / (2 * a'))^2 +$
$e * ((-\ b' + -\ 1 * sqrt\ (b'^2 - 4 * a' * c')) / (2 * a')) +$
$f \neq 0)$
then obtain $a'\ b'\ c'$ **where** *abc-prop*: $(a',\ b',\ c') \in set\ a \wedge$
$a' \neq 0 \wedge$
$-\ b'^2 + 4 * a' * c' \leq 0 \wedge$
$(\forall (d,\ e,\ f) \in set\ a.$
$d * ((-\ b' + -\ 1 * sqrt\ (b'^2 - 4 * a' * c')) / (2 * a'))^2 +$
$e * ((-\ b' + -\ 1 * sqrt\ (b'^2 - 4 * a' * c')) / (2 * a')) +$
$f = 0) \wedge$
$(\forall (d,\ e,\ f) \in set\ b.$
$d * ((-\ b' + -\ 1 * sqrt\ (b'^2 - 4 * a' * c')) / (2 * a'))^2 +$
$e * ((-\ b' + -\ 1 * sqrt\ (b'^2 - 4 * a' * c')) / (2 * a')) +$
$f < 0) \wedge$
$(\forall (d,\ e,\ f) \in set\ c.$

$d * ((-\ b' + -\ 1 * sqrt\ (b'^2 - 4 * a' * c')) / (2 * a'))^2 +$
$e * ((-\ b' + -\ 1 * sqrt\ (b'^2 - 4 * a' * c')) / (2 * a')) +$
$f \leq 0) \wedge$
$(\forall (d,\ e,\ f) \in set\ d.$
$d * ((-\ b' + -\ 1 * sqrt\ (b'^2 - 4 * a' * c')) / (2 * a'))^2 +$
$e * ((-\ b' + -\ 1 * sqrt\ (b'^2 - 4 * a' * c')) / (2 * a')) +$
$f \neq 0)$ **by** *auto*
then have $\exists (x::real).\ x = (-\ b' + -\ 1 * sqrt\ (b'^2 - 4 * a' * c')) / (2 * a')$ **by** *auto*
then obtain x **where** *x-prop*: $x = (-\ b' + -\ 1 * sqrt\ (b'^2 - 4 * a' * c')) / (2 * a')$ **by** *auto*
then have $(\forall (a,\ b,\ c) \in set\ a.\ a * x^2 + b * x + c = 0) \wedge$
$(\forall (a,\ b,\ c) \in set\ b.\ a * x^2 + b * x + c < 0) \wedge$
$(\forall (a,\ b,\ c) \in set\ c.\ a * x^2 + b * x + c \leq 0) \wedge$
$(\forall (a,\ b,\ c) \in set\ d.\ a * x^2 + b * x + c \neq 0)$ **using** *abc-prop* **by** *auto*
then show *?thesis* **by** *auto*
qed
then have *?f4* $\longrightarrow$ *?e1* **by** *auto*
have *?f5* $\Longrightarrow$ *?e1*
proof −
assume *asm*: $\exists (a',\ b',\ c') \in set\ b.$
$(a' = 0 \wedge b' \neq 0) \wedge$
$(\forall (d,\ e,\ f) \in set\ a.\ \exists y' > -\ c' / b'.\ \forall x \in \{-\ c' / b' < .. y'\}.\ d * x^2 + e * x + f = 0) \wedge$
$(\forall (d,\ e,\ f) \in set\ b.\ \exists y' > -\ c' / b'.\ \forall x \in \{-\ c' / b' < .. y'\}.\ d * x^2 + e * x + f < 0) \wedge$
$(\forall (d,\ e,\ f) \in set\ c.\ \exists y' > -\ c' / b'.\ \forall x \in \{-\ c' / b' < .. y'\}.\ d * x^2 + e * x + f \leq 0) \wedge$
$(\forall (d,\ e,\ f) \in set\ d.\ \exists y' > -\ c' / b'.\ \forall x \in \{-\ c' / b' < .. y'\}.\ d * x^2 + e * x + f \neq 0)$
then obtain $a'\ b'\ c'$ **where** *abc-prop*: $(a',\ b',\ c') \in set\ b \wedge$
$(a' = 0 \wedge b' \neq 0) \wedge$
$(\forall (d,\ e,\ f) \in set\ a.\ \exists y' > -\ c' / b'.\ \forall x \in \{-\ c' / b' < .. y'\}.\ d * x^2 + e * x + f = 0) \wedge$
$(\forall (d,\ e,\ f) \in set\ b.\ \exists y' > -\ c' / b'.\ \forall x \in \{-\ c' / b' < .. y'\}.\ d * x^2 + e * x + f < 0) \wedge$
$(\forall (d,\ e,\ f) \in set\ c.\ \exists y' > -\ c' / b'.\ \forall x \in \{-\ c' / b' < .. y'\}.\ d * x^2 + e * x + f \leq 0) \wedge$
$(\forall (d,\ e,\ f) \in set\ d.\ \exists y' > -\ c' / b'.\ \forall x \in \{-\ c' / b' < .. y'\}.\ d * x^2 + e * x + f \neq 0)$
by *auto*
then show *?thesis* **using** *qe-infinitesimals-helper*[*of* $a\ -\ c' / b'\ b\ c\ d$]
by *auto*
qed
then have *h5*: *?f5* $\longrightarrow$ *?e1*
by *auto*
have *?f6* $\Longrightarrow$ *?e1*
proof −
assume $\exists (a',\ b',\ c') \in set\ b.$

$a' \neq 0 \wedge$
$- b'^2 + 4 * a' * c' \leq 0 \wedge$
$(\forall (d, e, f) \in set\ a.$
$\exists y' > (- b' + 1 * sqrt\ (b'^2 - 4 * a' * c')) / (2 * a').$
$\forall x \in \{(- b' + 1 * sqrt\ (b'^2 - 4 * a' * c')) / (2 * a') <..< y'\}.$
$d * x^2 + e * x + f = 0) \wedge$
$(\forall (d, e, f) \in set\ b.$
$\exists y' > (- b' + 1 * sqrt\ (b'^2 - 4 * a' * c')) / (2 * a').$
$\forall x \in \{(- b' + 1 * sqrt\ (b'^2 - 4 * a' * c')) / (2 * a') <..< y'\}.$
$d * x^2 + e * x + f < 0) \wedge$
$(\forall (d, e, f) \in set\ c.$
$\exists y' > (- b' + 1 * sqrt\ (b'^2 - 4 * a' * c')) / (2 * a').$
$\forall x \in \{(- b' + 1 * sqrt\ (b'^2 - 4 * a' * c')) / (2 * a') <..< y'\}.$
$d * x^2 + e * x + f \leq 0) \wedge$
$(\forall (d, e, f) \in set\ d.$
$\exists y' > (- b' + 1 * sqrt\ (b'^2 - 4 * a' * c')) / (2 * a').$
$\forall x \in \{(- b' + 1 * sqrt\ (b'^2 - 4 * a' * c')) / (2 * a') <..< y'\}.$
$d * x^2 + e * x + f \neq 0)$
then obtain $a'\ b'\ c'$ **where** *abc-prop*: $(a', b', c') \in set\ b \wedge$
$a' \neq 0 \wedge$
$- b'^2 + 4 * a' * c' \leq 0 \wedge$
$(\forall (d, e, f) \in set\ a.$
$\exists y' > (- b' + 1 * sqrt\ (b'^2 - 4 * a' * c')) / (2 * a').$
$\forall x \in \{(- b' + 1 * sqrt\ (b'^2 - 4 * a' * c')) / (2 * a') <..< y'\}.$
$d * x^2 + e * x + f = 0) \wedge$
$(\forall (d, e, f) \in set\ b.$
$\exists y' > (- b' + 1 * sqrt\ (b'^2 - 4 * a' * c')) / (2 * a').$
$\forall x \in \{(- b' + 1 * sqrt\ (b'^2 - 4 * a' * c')) / (2 * a') <..< y'\}.$
$d * x^2 + e * x + f < 0) \wedge$
$(\forall (d, e, f) \in set\ c.$
$\exists y' > (- b' + 1 * sqrt\ (b'^2 - 4 * a' * c')) / (2 * a').$
$\forall x \in \{(- b' + 1 * sqrt\ (b'^2 - 4 * a' * c')) / (2 * a') <..< y'\}.$
$d * x^2 + e * x + f \leq 0) \wedge$
$(\forall (d, e, f) \in set\ d.$
$\exists y' > (- b' + 1 * sqrt\ (b'^2 - 4 * a' * c')) / (2 * a').$
$\forall x \in \{(- b' + 1 * sqrt\ (b'^2 - 4 * a' * c')) / (2 * a') <..< y'\}.$
$d * x^2 + e * x + f \neq 0)$ **by** *auto*
then show *?thesis* **using** *qe-infinitesimals-helper*[*of a* $(- b' + 1 * sqrt\ (b'^2 - 4 * a' * c')) / (2 * a')$ *b c d*]
by *auto*
qed
then have *h6*: *?f6* $\longrightarrow$ *?e1*
by *auto*
have *?f7* $\Longrightarrow$ *?e1*
proof −
assume $\exists (a', b', c') \in set\ b.$
$a' \neq 0 \wedge$
$- b'^2 + 4 * a' * c' \leq 0 \wedge$
$(\forall (d, e, f) \in set\ a.$

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

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

by *auto*

then show *?thesis* **using** *qe-infinitesimals-helper*[*of a* $(-\, b' + -\, 1 * sqrt\ (b'^2 - 4 * a' * c')) \,/\, (2 * a')\ b\ c\ d$]

by *auto*

qed

then have *h7*: *?f7* $\longrightarrow$ *?e1*

by *auto*

have *?f8* $\Longrightarrow$ *?e1*

proof −

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

then obtain a' b' c' **where** *abc-prop*: $(a',\ b',\ c') \in set\ c \wedge$
$(a' = 0 \wedge b' \neq 0) \wedge$
$(\forall (d,\ e,\ f) \in set\ a.\ d * (-\ c' \ /\ b')^2 + e * (-\ c' \ /\ b') + f = 0) \wedge$
$(\forall (d,\ e,\ f) \in set\ b.\ d * (-\ c' \ /\ b')^2 + e * (-\ c' \ /\ b') + f < 0) \wedge$
$(\forall (d,\ e,\ f) \in set\ c.\ d * (-\ c' \ /\ b')^2 + e * (-\ c' \ /\ b') + f \leq 0) \wedge$
$(\forall (d,\ e,\ f) \in set\ d.\ d * (-\ c' \ /\ b')^2 + e * (-\ c' \ /\ b') + f \neq 0)$ **by** *auto*
then have $\exists (x::real).\ x = (-\ c' \ /\ b')$ **by** *auto*
then obtain x **where** *x-prop*: $x = -\ c' \ /\ b'$ **by** *auto*
then have $(\forall (a,\ b,\ c) \in set\ a.\ a * x^2 + b * x + c = 0) \wedge$
$(\forall (a,\ b,\ c) \in set\ b.\ a * x^2 + b * x + c < 0) \wedge$
$(\forall (a,\ b,\ c) \in set\ c.\ a * x^2 + b * x + c \leq 0) \wedge$
$(\forall (a,\ b,\ c) \in set\ d.\ a * x^2 + b * x + c \neq 0)$ **using** *abc-prop* **by** *auto*
then show *?thesis* **by** *auto*
qed
then have *h8*: *?f8* $\longrightarrow$ *?e1* **by** *auto*
have *?f9* $\Longrightarrow$ *?e1*
proof −
assume $\exists (a',\ b',\ c') \in set\ c.$
$a' \neq 0 \wedge$
$-\ b'^2 + 4 * a' * c' \leq 0 \wedge$
$(\forall (d,\ e,\ f) \in set\ a.$
$d * ((-\ b' + -\ 1 * sqrt\ (b'^2 - 4 * a' * c')) \ /\ (2 * a'))^2 +$
$e * ((-\ b' + -\ 1 * sqrt\ (b'^2 - 4 * a' * c')) \ /\ (2 * a')) +$
$f = 0) \wedge$
$(\forall (d,\ e,\ f) \in set\ b.$
$d * ((-\ b' + -\ 1 * sqrt\ (b'^2 - 4 * a' * c')) \ /\ (2 * a'))^2 +$
$e * ((-\ b' + -\ 1 * sqrt\ (b'^2 - 4 * a' * c')) \ /\ (2 * a')) +$
$f < 0) \wedge$
$(\forall (d,\ e,\ f) \in set\ c.$
$d * ((-\ b' + -\ 1 * sqrt\ (b'^2 - 4 * a' * c')) \ /\ (2 * a'))^2 +$
$e * ((-\ b' + -\ 1 * sqrt\ (b'^2 - 4 * a' * c')) \ /\ (2 * a')) +$
$f \leq 0) \wedge$
$(\forall (d,\ e,\ f) \in set\ d.$
$d * ((-\ b' + -\ 1 * sqrt\ (b'^2 - 4 * a' * c')) \ /\ (2 * a'))^2 +$
$e * ((-\ b' + -\ 1 * sqrt\ (b'^2 - 4 * a' * c')) \ /\ (2 * a')) +$
$f \neq 0)$
then obtain a' b' c' **where** *abc-prop*: $(a',\ b',\ c') \in set\ c \wedge$
$a' \neq 0 \wedge$
$-\ b'^2 + 4 * a' * c' \leq 0 \wedge$
$(\forall (d,\ e,\ f) \in set\ a.$
$d * ((-\ b' + -\ 1 * sqrt\ (b'^2 - 4 * a' * c')) \ /\ (2 * a'))^2 +$
$e * ((-\ b' + -\ 1 * sqrt\ (b'^2 - 4 * a' * c')) \ /\ (2 * a')) +$
$f = 0) \wedge$
$(\forall (d,\ e,\ f) \in set\ b.$
$d * ((-\ b' + -\ 1 * sqrt\ (b'^2 - 4 * a' * c')) \ /\ (2 * a'))^2 +$
$e * ((-\ b' + -\ 1 * sqrt\ (b'^2 - 4 * a' * c')) \ /\ (2 * a')) +$
$f < 0) \wedge$
$(\forall (d,\ e,\ f) \in set\ c.$
$d * ((-\ b' + -\ 1 * sqrt\ (b'^2 - 4 * a' * c')) \ /\ (2 * a'))^2 +$

$e * ((-\ b' + -\ 1 * sqrt\ (b'^2 - 4 * a' * c')) / (2 * a')) +$
$f \leq 0) \wedge$
$(\forall (d,\ e,\ f) \in set\ d.$
$d * ((-\ b' + -\ 1 * sqrt\ (b'^2 - 4 * a' * c')) / (2 * a'))^2 +$
$e * ((-\ b' + -\ 1 * sqrt\ (b'^2 - 4 * a' * c')) / (2 * a')) +$
$f \neq 0)$ **by** *auto*
then have $\exists (x::real).\ x = (-\ b' + -\ 1 * sqrt\ (b'^2 - 4 * a' * c')) / (2 * a')$ **by** *auto*
then obtain x **where** *x-prop*: $x = (-\ b' + -\ 1 * sqrt\ (b'^2 - 4 * a' * c')) / (2 * a')$ **by** *auto*
then have $(\forall (a,\ b,\ c) \in set\ a.\ a * x^2 + b * x + c = 0) \wedge$
$(\forall (a,\ b,\ c) \in set\ b.\ a * x^2 + b * x + c < 0) \wedge$
$(\forall (a,\ b,\ c) \in set\ c.\ a * x^2 + b * x + c \leq 0) \wedge$
$(\forall (a,\ b,\ c) \in set\ d.\ a * x^2 + b * x + c \neq 0)$ **using** *abc-prop* **by** *auto*
then show *?thesis* **by** *auto*
qed
then have *h9*: *?f9* $\longrightarrow$ *?e1* **by** *auto*
have *?f10* $\Longrightarrow$ *?e1*
proof −
assume *asm*: $\exists (a',\ b',\ c') \in set\ d.$
$(a' = 0 \wedge b' \neq 0) \wedge$
$(\forall (d,\ e,\ f) \in set\ a.\ \exists y' > -\ c' / b'.\ \forall x \in \{-\ c' / b' < .. y'\}.\ d * x^2 + e * x + f = 0) \wedge$
$(\forall (d,\ e,\ f) \in set\ b.\ \exists y' > -\ c' / b'.\ \forall x \in \{-\ c' / b' < .. y'\}.\ d * x^2 + e * x + f < 0) \wedge$
$(\forall (d,\ e,\ f) \in set\ c.\ \exists y' > -\ c' / b'.\ \forall x \in \{-\ c' / b' < .. y'\}.\ d * x^2 + e * x + f \leq 0) \wedge$
$(\forall (d,\ e,\ f) \in set\ d.\ \exists y' > -\ c' / b'.\ \forall x \in \{-\ c' / b' < .. y'\}.\ d * x^2 + e * x + f \neq 0)$
then obtain $a'\ b'\ c'$ **where** *abc-prop*: $(a',\ b',\ c') \in set\ d \wedge$
$(a' = 0 \wedge b' \neq 0) \wedge$
$(\forall (d,\ e,\ f) \in set\ a.\ \exists y' > -\ c' / b'.\ \forall x \in \{-\ c' / b' < .. y'\}.\ d * x^2 + e * x + f = 0) \wedge$
$(\forall (d,\ e,\ f) \in set\ b.\ \exists y' > -\ c' / b'.\ \forall x \in \{-\ c' / b' < .. y'\}.\ d * x^2 + e * x + f < 0) \wedge$
$(\forall (d,\ e,\ f) \in set\ c.\ \exists y' > -\ c' / b'.\ \forall x \in \{-\ c' / b' < .. y'\}.\ d * x^2 + e * x + f \leq 0) \wedge$
$(\forall (d,\ e,\ f) \in set\ d.\ \exists y' > -\ c' / b'.\ \forall x \in \{-\ c' / b' < .. y'\}.\ d * x^2 + e * x + f \neq 0)$
by *auto*
then show *?thesis* **using** *qe-infinitesimals-helper*[*of* $a\ -\ c' / b'\ b\ c\ d$]
by *auto*
qed
then have *h10*: *?f10* $\longrightarrow$ *?e1* **by** *auto*
have *?f11* $\Longrightarrow$ *?e1*
proof −
assume $\exists (a',\ b',\ c') \in set\ d.$
$a' \neq 0 \wedge$
$-\ b'^2 + 4 * a' * c' \leq 0 \wedge$

$(\forall (d,\ e,\ f) \in set\ a.$
$\exists y' > (-\ b' + 1 * sqrt\ (b'^2 - 4 * a' * c')) / (2 * a').$
$\forall x \in \{(-\ b' + 1 * sqrt\ (b'^2 - 4 * a' * c')) / (2 * a') <..y'\}.$
$d * x^2 + e * x + f = 0) \wedge$
$(\forall (d,\ e,\ f) \in set\ b.$
$\exists y' > (-\ b' + 1 * sqrt\ (b'^2 - 4 * a' * c')) / (2 * a').$
$\forall x \in \{(-\ b' + 1 * sqrt\ (b'^2 - 4 * a' * c')) / (2 * a') <..y'\}.$
$d * x^2 + e * x + f < 0) \wedge$
$(\forall (d,\ e,\ f) \in set\ c.$
$\exists y' > (-\ b' + 1 * sqrt\ (b'^2 - 4 * a' * c')) / (2 * a').$
$\forall x \in \{(-\ b' + 1 * sqrt\ (b'^2 - 4 * a' * c')) / (2 * a') <..y'\}.$
$d * x^2 + e * x + f \leq 0) \wedge$
$(\forall (d,\ e,\ f) \in set\ d.$
$\exists y' > (-\ b' + 1 * sqrt\ (b'^2 - 4 * a' * c')) / (2 * a').$
$\forall x \in \{(-\ b' + 1 * sqrt\ (b'^2 - 4 * a' * c')) / (2 * a') <..y'\}.$
$d * x^2 + e * x + f \neq 0)$
then obtain $a'\ b'\ c'$ **where** *abc-prop*: $(a',\ b',\ c') \in set\ d \wedge$
$a' \neq 0 \wedge$
$-\ b'^2 + 4 * a' * c' \leq 0 \wedge$
$(\forall (d,\ e,\ f) \in set\ a.$
$\exists y' > (-\ b' + 1 * sqrt\ (b'^2 - 4 * a' * c')) / (2 * a').$
$\forall x \in \{(-\ b' + 1 * sqrt\ (b'^2 - 4 * a' * c')) / (2 * a') <..y'\}.$
$d * x^2 + e * x + f = 0) \wedge$
$(\forall (d,\ e,\ f) \in set\ b.$
$\exists y' > (-\ b' + 1 * sqrt\ (b'^2 - 4 * a' * c')) / (2 * a').$
$\forall x \in \{(-\ b' + 1 * sqrt\ (b'^2 - 4 * a' * c')) / (2 * a') <..y'\}.$
$d * x^2 + e * x + f < 0) \wedge$
$(\forall (d,\ e,\ f) \in set\ c.$
$\exists y' > (-\ b' + 1 * sqrt\ (b'^2 - 4 * a' * c')) / (2 * a').$
$\forall x \in \{(-\ b' + 1 * sqrt\ (b'^2 - 4 * a' * c')) / (2 * a') <..y'\}.$
$d * x^2 + e * x + f \leq 0) \wedge$
$(\forall (d,\ e,\ f) \in set\ d.$
$\exists y' > (-\ b' + 1 * sqrt\ (b'^2 - 4 * a' * c')) / (2 * a').$
$\forall x \in \{(-\ b' + 1 * sqrt\ (b'^2 - 4 * a' * c')) / (2 * a') <..y'\}.$
$d * x^2 + e * x + f \neq 0)$ **by** *auto*
then show *?thesis* **using** *qe-infinitesimals-helper*[*of a* $(-\ b' + 1 * sqrt\ (b'^2 - 4 * a' * c')) / (2 * a')\ b\ c\ d$]
by *auto*
qed
then have *h11*: *?f11* $\longrightarrow$ *?e1* **by** *auto*
have *?f12* $\Longrightarrow$ *?e1* **proof** −
assume $\exists (a',\ b',\ c') \in set\ d.$
$a' \neq 0 \wedge$
$-\ b'^2 + 4 * a' * c' \leq 0 \wedge$
$(\forall (d,\ e,\ f) \in set\ a.$
$\exists y' > (-\ b' + -\ 1 * sqrt\ (b'^2 - 4 * a' * c')) / (2 * a').$
$\forall x \in \{(-\ b' + -\ 1 * sqrt\ (b'^2 - 4 * a' * c')) / (2 * a') <..y'\}.$
$d * x^2 + e * x + f = 0) \wedge$
$(\forall (d,\ e,\ f) \in set\ b.$

$\exists\, y'{>}(-\ b' + -\ 1 * sqrt\ (b'^2 - 4 * a' * c')) \ /\ (2 * a').$
$\forall\, x{\in}\{(-\ b' + -\ 1 * sqrt\ (b'^2 - 4 * a' * c')) \ /\ (2 * a'){<}..y'\}.$
$d * x^2 + e * x + f < 0) \wedge$
$(\forall\, (d,\ e,\ f){\in}set\ c.$
$\exists\, y'{>}(-\ b' + -\ 1 * sqrt\ (b'^2 - 4 * a' * c')) \ /\ (2 * a').$
$\forall\, x{\in}\{(-\ b' + -\ 1 * sqrt\ (b'^2 - 4 * a' * c')) \ /\ (2 * a'){<}..y'\}.$
$d * x^2 + e * x + f \leq 0) \wedge$
$(\forall\, (d,\ e,\ f){\in}set\ d.$
$\exists\, y'{>}(-\ b' + -\ 1 * sqrt\ (b'^2 - 4 * a' * c')) \ /\ (2 * a').$
$\forall\, x{\in}\{(-\ b' + -\ 1 * sqrt\ (b'^2 - 4 * a' * c')) \ /\ (2 * a'){<}..y'\}.$
$d * x^2 + e * x + f \neq 0)$
then obtain $a'\ b'\ c'$ **where** *abc-prop*: $(a',\ b',\ c'){\in}set\ d \wedge$
$a' \neq 0 \wedge$
$-\ b'^2 + 4 * a' * c' \leq 0 \wedge$
$(\forall\, (d,\ e,\ f){\in}set\ a.$
$\exists\, y'{>}(-\ b' + -\ 1 * sqrt\ (b'^2 - 4 * a' * c')) \ /\ (2 * a').$
$\forall\, x{\in}\{(-\ b' + -\ 1 * sqrt\ (b'^2 - 4 * a' * c')) \ /\ (2 * a'){<}..y'\}.$
$d * x^2 + e * x + f = 0) \wedge$
$(\forall\, (d,\ e,\ f){\in}set\ b.$
$\exists\, y'{>}(-\ b' + -\ 1 * sqrt\ (b'^2 - 4 * a' * c')) \ /\ (2 * a').$
$\forall\, x{\in}\{(-\ b' + -\ 1 * sqrt\ (b'^2 - 4 * a' * c')) \ /\ (2 * a'){<}..y'\}.$
$d * x^2 + e * x + f < 0) \wedge$
$(\forall\, (d,\ e,\ f){\in}set\ c.$
$\exists\, y'{>}(-\ b' + -\ 1 * sqrt\ (b'^2 - 4 * a' * c')) \ /\ (2 * a').$
$\forall\, x{\in}\{(-\ b' + -\ 1 * sqrt\ (b'^2 - 4 * a' * c')) \ /\ (2 * a'){<}..y'\}.$
$d * x^2 + e * x + f \leq 0) \wedge$
$(\forall\, (d,\ e,\ f){\in}set\ d.$
$\exists\, y'{>}(-\ b' + -\ 1 * sqrt\ (b'^2 - 4 * a' * c')) \ /\ (2 * a').$
$\forall\, x{\in}\{(-\ b' + -\ 1 * sqrt\ (b'^2 - 4 * a' * c')) \ /\ (2 * a'){<}..y'\}.$
$d * x^2 + e * x + f \neq 0)$
by *auto*
then show *?thesis* **using** *qe-infinitesimals-helper*[*of a* $(-\ b' + -\ 1 * sqrt\ (b'^2 - 4 * a' * c')) \ /\ (2 * a')\ b\ c\ d$]
by *auto*
qed
then have *h12*: *?f12* $\longrightarrow$ *?e1* **by** *auto*
have *?f13* $\Longrightarrow$ *?e1* **proof** −
assume $\exists\, (a',\ b',\ c'){\in}set\ c.$
$a' \neq 0 \wedge$
$-\ b'^2 + 4 * a' * c' \leq 0 \wedge$
$(\forall\, (d,\ e,\ f){\in}set\ a.$
$d * ((-\ b' + 1 * sqrt\ (b'^2 - 4 * a' * c')) \ /\ (2 * a'))^2 +$
$e * ((-\ b' + 1 * sqrt\ (b'^2 - 4 * a' * c')) \ /\ (2 * a')) +$
$f = 0) \wedge$
$(\forall\, (d,\ e,\ f){\in}set\ b.$
$d * ((-\ b' + 1 * sqrt\ (b'^2 - 4 * a' * c')) \ /\ (2 * a'))^2 +$
$e * ((-\ b' + 1 * sqrt\ (b'^2 - 4 * a' * c')) \ /\ (2 * a')) +$
$f < 0) \wedge$
$(\forall\, (d,\ e,\ f){\in}set\ c.$

$d * ((-\ b' + 1 * sqrt\ (b'^2 - 4 * a' * c')) / (2 * a'))^2 +$
$e * ((-\ b' + 1 * sqrt\ (b'^2 - 4 * a' * c')) / (2 * a')) +$
$f \leq 0) \wedge$
$(\forall (d, e, f) \in set\ d.$
$d * ((-\ b' + 1 * sqrt\ (b'^2 - 4 * a' * c')) / (2 * a'))^2 +$
$e * ((-\ b' + 1 * sqrt\ (b'^2 - 4 * a' * c')) / (2 * a')) +$
$f \neq 0)$
then obtain $a'\ b'\ c'$ **where** *abc-prop*: $(a', b', c') \in set\ c \wedge$
$a' \neq 0 \wedge$
$-\ b'^2 + 4 * a' * c' \leq 0 \wedge$
$(\forall (d, e, f) \in set\ a.$
$d * ((-\ b' + 1 * sqrt\ (b'^2 - 4 * a' * c')) / (2 * a'))^2 +$
$e * ((-\ b' + 1 * sqrt\ (b'^2 - 4 * a' * c')) / (2 * a')) +$
$f = 0) \wedge$
$(\forall (d, e, f) \in set\ b.$
$d * ((-\ b' + 1 * sqrt\ (b'^2 - 4 * a' * c')) / (2 * a'))^2 +$
$e * ((-\ b' + 1 * sqrt\ (b'^2 - 4 * a' * c')) / (2 * a')) +$
$f < 0) \wedge$
$(\forall (d, e, f) \in set\ c.$
$d * ((-\ b' + 1 * sqrt\ (b'^2 - 4 * a' * c')) / (2 * a'))^2 +$
$e * ((-\ b' + 1 * sqrt\ (b'^2 - 4 * a' * c')) / (2 * a')) +$
$f \leq 0) \wedge$
$(\forall (d, e, f) \in set\ d.$
$d * ((-\ b' + 1 * sqrt\ (b'^2 - 4 * a' * c')) / (2 * a'))^2 +$
$e * ((-\ b' + 1 * sqrt\ (b'^2 - 4 * a' * c')) / (2 * a')) +$
$f \neq 0)$ **by** *auto*
then have $\exists (x::real).\ x = (-\ b' + 1 * sqrt\ (b'^2 - 4 * a' * c')) / (2 * a')$ **by** *auto*
then obtain x **where** *x-prop*: $x = (-\ b' + 1 * sqrt\ (b'^2 - 4 * a' * c')) / (2 * a')$ **by** *auto*
then have $(\forall (a, b, c) \in set\ a.\ a * x^2 + b * x + c = 0) \wedge$
$(\forall (a, b, c) \in set\ b.\ a * x^2 + b * x + c < 0) \wedge$
$(\forall (a, b, c) \in set\ c.\ a * x^2 + b * x + c \leq 0) \wedge$
$(\forall (a, b, c) \in set\ d.\ a * x^2 + b * x + c \neq 0)$ **using** *abc-prop* **by** *auto*
then show *?thesis* **by** *auto*
qed
then have *h13*: *?f13* $\longrightarrow$ *?e1* **by** *auto*
show *?thesis* **using** *bigor h1 h2 h3 h4 h5 h6 h7 h8 h9 h10 h11 h12 h13*
using *assms*
by (*smt* $\langle \exists (a', b', c') \in set\ a.\ a' \neq 0 \wedge -\ b'^2 + 4 * a' * c' \leq 0 \wedge (\forall (d, e, f) \in set\ a.\ d * ((-\ b' + -\ 1 * sqrt\ (b'^2 - 4 * a' * c')) / (2 * a'))^2 + e * ((-\ b' + -\ 1 * sqrt\ (b'^2 - 4 * a' * c')) / (2 * a')) + f = 0) \wedge (\forall (d, e, f) \in set\ b.\ d * ((-\ b' + -\ 1 * sqrt\ (b'^2 - 4 * a' * c')) / (2 * a'))^2 + e * ((-\ b' + -\ 1 * sqrt\ (b'^2 - 4 * a' * c')) / (2 * a')) + f < 0) \wedge (\forall (d, e, f) \in set\ c.\ d * ((-\ b' + -\ 1 * sqrt\ (b'^2 - 4 * a' * c')) / (2 * a'))^2 + e * ((-\ b' + -\ 1 * sqrt\ (b'^2 - 4 * a' * c')) / (2 * a')) + f \leq 0) \wedge (\forall (d, e, f) \in set\ d.\ d * ((-\ b' + -\ 1 * sqrt\ (b'^2 - 4 * a' * c')) / (2 * a'))^2 + e * ((-\ b' + -\ 1 * sqrt\ (b'^2 - 4 * a' * c')) / (2 * a')) + f \neq 0) \Longrightarrow \exists x.\ (\forall (a, b, c) \in set\ a.\ a * x^2 + b * x + c = 0) \wedge (\forall (a, b, c) \in set\ b.\ a * x^2 + b * x + c < 0) \wedge (\forall (a, b, c) \in set\ c.\ a * x^2 + b * x + c \leq 0) \wedge (\forall (a, b, c) \in set$

$d.\ a * x^2 + b * x + c \neq 0)\rangle)$

qed

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

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

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

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

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

proof −
let *?e1* = $((\forall\,(a,\ b,\ c){\in}set\ a.\ a = 0 \wedge b = 0 \wedge c = 0) \wedge$
$(\forall\,(a,\ b,\ c){\in}set\ b.\ \exists\, x.\ \forall\, y{<}x.\ a * y^2 + b * y + c < 0) \wedge$
$(\forall\,(a,\ b,\ c){\in}set\ c.\ \exists\, x.\ \forall\, y{<}x.\ a * y^2 + b * y + c \leq 0) \wedge$
$(\forall\,(a,\ b,\ c){\in}set\ d.\ \exists\, x.\ \forall\, y{<}x.\ a * y^2 + b * y + c \neq 0) \vee$
$(\exists\,(a',\ b',\ c'){\in}set\ a.$
$(a' = 0 \wedge b' \neq 0) \wedge$
$(\forall\,(d,\ e,\ f){\in}set\ a.\ d * (-\ c'\ /\ b')^2 + e * (-\ c'\ /\ b') + f = 0) \wedge$
$(\forall\,(d,\ e,\ f){\in}set\ b.\ d * (-\ c'\ /\ b')^2 + e * (-\ c'\ /\ b') + f < 0) \wedge$
$(\forall\,(d,\ e,\ f){\in}set\ c.\ d * (-\ c'\ /\ b')^2 + e * (-\ c'\ /\ b') + f \leq 0) \wedge$
$(\forall\,(d,\ e,\ f){\in}set\ d.\ d * (-\ c'\ /\ b')^2 + e * (-\ c'\ /\ b') + f \neq 0) \vee$
$a' \neq 0 \wedge$
$-\ b'^2 + 4 * a' * c' \leq 0 \wedge$
$((\forall\,(d,\ e,\ f){\in}set\ a.$
$d * ((-\ b' + 1 * sqrt\ (b'^2 - 4 * a' * c'))\ /\ (2 * a'))^2 +$
$e * ((-\ b' + 1 * sqrt\ (b'^2 - 4 * a' * c'))\ /\ (2 * a')) +$
$f =$
$0) \wedge$
$(\forall\,(d,\ e,\ f){\in}set\ b.$
$d * ((-\ b' + 1 * sqrt\ (b'^2 - 4 * a' * c'))\ /\ (2 * a'))^2 +$
$e * ((-\ b' + 1 * sqrt\ (b'^2 - 4 * a' * c'))\ /\ (2 * a')) +$
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

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

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

$$
\begin{array}{l}
(\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) \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))))\\
\textbf{let}\ ?e2 = (\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\\
\quad (\forall\,(a,\,b,\,c)\in set\ d.\ a*x^2+b*x+c\neq 0))\\
\textbf{have}\ h1\colon\ ?e1 \longrightarrow ?e2\ \textbf{using}\ qe\text{-}backwards
\end{array}
$$

by *auto*
have *h2*: *?e2* $\longrightarrow$ *?e1* **using** *qe-forwards*
by *auto*
have (*?e2* $\longrightarrow$ *?e1*) $\wedge$ (*?e1* $\longrightarrow$ *?e2*) **using** *h1 h2*
by *force*
then have *?e2* $\longleftrightarrow$ *?e1*
using *iff-conv-conj-imp*[*of ?e1 ?e2*]
by *presburger*
then show *?thesis*
by *auto*
qed

fun *eq-fun* :: *real* $\Rightarrow$ *real* $\Rightarrow$ *real* $\Rightarrow$ (*real*$*$*real*$*$*real*) *list* $\Rightarrow$ (*real*$*$*real*$*$*real*) *list* $\Rightarrow$
(*real*$*$*real*$*$*real*) *list* $\Rightarrow$ (*real*$*$*real*$*$*real*) *list* $\Rightarrow$ *bool* **where**
eq-fun a′ b′ c′ eq les leq neq = ((*a′* = *0* $\wedge$ *b′* $\neq$ *0*) $\wedge$
(∀ *a*∈*set eq*.
case a of (*d*, *e*, *f*) $\Rightarrow$ *d* $*$ (− *c′* / *b′*)^2 + *e* $*$ (− *c′* / *b′*) + *f* = *0*) $\wedge$
(∀ *a*∈*set les*.
case a of (*d*, *e*, *f*) $\Rightarrow$ *d* $*$ (− *c′* / *b′*)^2 + *e* $*$ (− *c′* / *b′*) + *f* < *0*) $\wedge$
(∀ *a*∈*set leq*.
case a of (*d*, *e*, *f*) $\Rightarrow$ *d* $*$ (− *c′* / *b′*)^2 + *e* $*$ (− *c′* / *b′*) + *f* $\leq$ *0*) $\wedge$
(∀ *a*∈*set neq*.
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* $\wedge$
((∀ *a*∈*set eq*.
case a of
(*d*, *e*, *f*) $\Rightarrow$
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) $\wedge$
(∀ *a*∈*set les*.
case a of
(*d*, *e*, *f*) $\Rightarrow$
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*) $\wedge$
(∀ *a*∈*set leq*.
case a of
(*d*, *e*, *f*) $\Rightarrow$
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*) $\wedge$
(∀ *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) ∨
    (∀ a∈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 * x^2 + e * x + f = 0) ∧
      (∀ (d, e, f)∈set les.
        ∃ y'>- c' / b'. ∀ x∈{- c' / b'<..y'}. d * x^2 + e * x + f < 0) ∧
      (∀ (d, e, f)∈set leq.
        ∃ y'>- c' / b'. ∀ x∈{- c' / b'<..y'}. d * x^2 + e * x + f ≤ 0) ∧
      (∀ (d, e, f)∈set neq.
        ∃ y'>- c' / b'. ∀ x∈{- c' / b'<..y'}. d * x^2 + 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').
```

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

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

```
    (∀(a,b,c)∈set leq. ∃x. ∀y<x. a*y^2+b*y+c≤0) ∧
    (∀(a,b,c)∈set neq. ∃x. ∀y<x. a*y^2+b*y+c≠0)
   )

  assumes roots = (λ(a,b,c).
     if a=0 ∧ b≠0 then {−c/b}::real set 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 {})

  assumes A = ⋃(roots ‘ (set eq))
  assumes B = ⋃(roots ‘ (set les))
  assumes C = ⋃(roots ‘ (set leq))
  assumes D = ⋃(roots ‘ (set neq))

  shows (∃x. F(x)) = (F_inf∨(∃r∈A. F r)∨(∃r∈B. Fε r)∨(∃r∈C. F r)∨(∃r∈D.
Fε r))
proof−
  { fix X
    have (∃(a, b, c)∈set X. eq-fun a b c eq les leq neq) = (∃x∈F ‘ ⋃(roots ‘ (set
X)). x)
    proof(induction X)
      case Nil
      then show ?case by auto
    next
      case (Cons p X)
       have h1: (∃x∈F ‘ ⋃ (roots ‘ set (p # X)). x) = ((∃x∈F ‘ roots p. x) ∨
(∃x∈F ‘ ⋃ (roots ‘ set X). x))
        by auto
      have h2 :(case p of (a,b,c) ⇒ eq-fun a b c eq les leq neq) = (∃x∈F ‘ roots p.
x)
        apply(cases p) unfolding assms apply simp by linarith
      show ?case  unfolding h1 Cons[symmetric] using h2 by auto
    qed
  }
  then have eq : ⋀X. (∃(a, b, c)∈set X. eq-fun a b c eq les leq neq) = (∃x∈F ‘
⋃ (roots ‘ set X). x) by auto
  { fix X
    have (∃(a, b, c)∈set X. les-fun a b c eq les leq neq) = (∃x∈Fε ‘⋃(roots ‘ (set
X)). x)
    proof(induction X)
      case Nil
      then show ?case by auto
    next
      case (Cons p X)
       have h1: (∃x∈Fε ‘ ⋃ (roots ‘ set (p # X)). x) = ((∃x∈Fε ‘ roots p. x) ∨
(∃x∈Fε ‘ ⋃ (roots ‘ set X). x))
        by auto
      have h2 :(case p of (a,b,c) ⇒ les-fun a b c eq les leq neq) = (∃x∈Fε ‘ roots
p. x)
```

apply(*cases p*) **unfolding** *assms* **apply** *simp* **by** *linarith*
show *?case* **unfolding** *h1 Cons*[*symmetric*] **using** *h2* **by** *auto*
qed
}
then have *les* : $\bigwedge X.\ (\exists\,(a,\ b,\ c){\in}set\ X.\ les\text{-}fun\ a\ b\ c\ eq\ les\ leq\ neq) = (\exists\,x{\in}F\varepsilon\ `\bigcup\ (roots\ `\ set\ X).\ x)$ **by** *auto*
have *inf* : $(\forall\,(a,\ b,\ c){\in}set\ eq.\ a = 0 \wedge b = 0 \wedge c = 0) = (\forall\,x{\in}set\ eq.\ case\ x\ of\ (a,\ b,\ c) \Rightarrow \exists\,x.\ \forall\,y{<}x.\ a * y^2 + b * y + c = 0)$
proof(*induction eq*)
case *Nil*
then show *?case* **by** *auto*
next
case (*Cons p eq*)
then show *?case* **proof**(*cases p*)
case (*fields a b c*)
show *?thesis* **unfolding** *fields* **using** *infzeros*[*of - a b c*] *Cons* **by** *auto*
qed
qed
show *?thesis*
using *qe*[*of eq les leq neq*]
using *eq*[*of eq*] *eq*[*of leq*] *les*[*of les*] *les*[*of neq*] **unfolding** *inf assms*
by *auto*
qed

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−
define *less* **where** *less* = (λ(*a*::*real*,*b*::*real*,*c*::*real*).λ(*a'*,*b'*,*c'*). *a*<*a'*∨ (*a*=*a'*∧
(*b*<*b'*∨(*b*=*b'*∧*c*<*c'*))))
define *leq* **where** *leq* = ($\lambda x.\lambda y.\ x{=}y \vee$ *less x y*)
have *linorder*: *class.linorder leq less*
unfolding *class.linorder-def class.order-def class.preorder-def class.order-axioms-def
class.linorder-axioms-def*
less-def leq-def **by** *auto*
show *?thesis*
using *assms*(*6*−*8*) **unfolding** *assms*(*1*) **apply** *simp*
using *general-qe'*[*OF - - - assms*(*9*), *of F List.linorder.sorted-list-of-set leq*
(*X* (=)) *List.linorder.sorted-list-of-set leq* (*X* (<)) *List.linorder.sorted-list-of-set*
leq (*X* (≤)) *List.linorder.sorted-list-of-set leq* (*X* (≠)) *F*ε F_{inf} *A B C D*]
unfolding *List.linorder.set-sorted-list-of-set*[*OF linorder assms*(*2*)] *List.linorder.set-sorted-list-of-set*[*OF
linorder assms*(*3*)] *List.linorder.set-sorted-list-of-set*[*OF linorder assms*(*4*)] *List.linorder.set-sorted-list-of-set*[*O
linorder assms*(*5*)]
using *assms*(*10*−*13*)**by** *auto*
qed

theorem *general-qe* :

assumes *R* = {(=), (<), (≤), (≠)}
assumes ∀ *rel*∈*R*. *finite* (*Atoms rel*)

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

assumes *F*ε = ($\lambda r.$ ∀ *rel*∈*R*. ∀ (*a*,*b*,*c*)∈(*Atoms rel*). ∃ *y*>*r*.∀ *x*∈{*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*∈$\bigcup$(*roots* ‘ (*Atoms* (=) ∪ *Atoms* (≤))). *F r*) ∨
(∃ *r*∈$\bigcup$(*roots* ‘ (*Atoms* (<) ∪ *Atoms* (≠))). *F*ε *r*))
using *general-qe''*[*OF assms*(*1*) - - - - *assms*(*3*−*6*) *refl refl refl refl*]
using *assms*(*2*) **unfolding** *assms*(*1*)
by *auto*

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 .

lemma *coeff-code*[*code*]: *coeff* = *coeff-code*
unfolding *coeff-def* **apply**(*transfer*) **by** *auto*

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

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

lemma *plus-monom-zero*[*simp*]: *p* + *MPoly-Type.monom m 0* = *p*
by *transfer auto*

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

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*) ⟹ *m* ∉ *monomials p1*
⟹ *a* ≠ *0* ⟹ *P* (*p1*+*p2*))
shows *P p* **using** *assms*
proof (*induction p rule*: *mpoly-induct*)
case (*sum p1 p2 m a*)
then show *?case*
by (*cases a* = *0*) (*auto simp*: *monomials.rep-eq*)
qed *simp*

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

lemma *Sum-any-lookup-times-eq*:
(∑ *k*. ((*lookup* (*x*::*'a*⇒$_0$(*'b*::*comm-semiring-1*)) (*k*::*'a*)) ∗ ((*f*:: *'a*⇒(*'b*::*comm-semiring-1*)) *k*))) = (∑ *k*∈*keys*
x. (*lookup x* (*k*::*'a*)) ∗ (*f k*))
by (*subst Sum-any.conditionalize*) (*auto simp*: *in-keys-iff intro*!: *Sum-any.cong*)

lemma *Prod-any-power-lookup-eq*:
(∏ k::'a. f k ^ lookup (x::'a⇒₀nat) k) = (∏ k∈keys x. f k ^ lookup x k)
by (*subst Prod-any.conditionalize*) (*auto simp*: *in-keys-iff intro*!:
Prod-any.cong)

lemma *insertion-monom*: *insertion i* (*monom m a*) = *a* ∗ (∏ *k*∈*keys m*. *i k* ^
lookup m k)
by *transfer* (*auto simp*: *insertion-aux-def insertion-fun-def*
Sum-any-lookup-times-eq Prod-any-power-lookup-eq)

lemma *monomials-monom*[*simp*]: *monomials* (*monom m a*) = (*if a* = *0 then* {}
else {*m*})
by *transfer auto*

lemma *finite-monomials*[*simp*]: *finite* (*monomials m*)
by *transfer auto*

lemma *monomials-add-disjoint*:
monomials (*a* + *b*) = *monomials a* ∪ *monomials b* **if** *monomials a* ∩
monomials b = {}
using *that*
by *transfer* (*auto simp add*: *keys-plus-eqI*)

lemma *coeff-monom*[*simp*]: *coeff* (*monom m a*) *m* = *a*
by *transfer simp*

lemma *coeff-not-in-monomials*[*simp*]: *coeff m x* = *0* **if** *x* ∉ *monomials m*
using *that*
by *transfer* (*simp add*: *in-keys-iff*)

code-thms *insertion*

lemma *insertion-code*[*code*]: *insertion i mp* =
(∑ *m*∈*monomials mp*. (*coeff mp m*) ∗ (∏ *k*∈*keys m*. *i k* ^ *lookup m k*))
proof (*induction mp rule*: *mpoly-induct*)
case (*monom m a*)
show *?case*
by (*simp add*: *insertion-monom*)
next
case (*sum p1 p2 m a*)
have *monomials-add*: *monomials* (*p1* + *p2*) = *insert m* (*monomials p1*)
using *sum.hyps*
by (*auto simp*: *monomials-add-disjoint*)
have ∗: *coeff* (*monom m a*) *x* = *0* **if** *x* ∈ *monomials p1* **for** *x*
using *sum.hyps that*
by (*subst coeff-not-in-monomials*) *auto*
show *?case*
unfolding *insertion-add monomials-add sum.IH*
using *sum.hyps*

by (*auto simp*: *coeff-add* ∗ *algebra-simps*)
qed

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*} **.**

lemma *Poly-Mapping-update-code*[*code*]: *Poly-Mapping.update a b* (*Pm-fmap*
fm) = *Pm-fmap* (*fmupd a b fm*)
by (*auto intro*!: *poly-mapping-eqI simp*: *update.rep-eq*
fmlookup-default-def)

lemma *monom-zero* [*simp*] : *monom m 0* = *0*
by (*simp add*: *coeff-all-0*)

lemma *remove-key-code*[*code*]: *remove-key v* (*Pm-fmap fm*) = *Pm-fmap*
(*fmdrop v fm*)
by (*auto simp*: *remove-key-lookup fmlookup-default-def intro*!:
poly-mapping-eqI)
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*)))
apply (*rule extract-var-finite-set*[**where** *S*=*monomials p*])
using *coeff-not-in-monomials* **by** *auto*
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*)
unfolding *monomials.rep-eq vars-def* **..**

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**
by (*metis add.right-inverse eq-neg-iff-add-eq-0 insertion-add insertion-zero*)

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*
by (*auto simp*: *poly-mapping-eq-iff lookup-of-nat fun-eq-iff lookup-single*)

lemma *of-nat-times-monomial*: *of-nat p * monomial c i = monomial (p*c) i*
by (*auto simp*: *poly-mapping-eq-iff prod-fun-def fun-eq-iff of-nat-monomial lookup-single mult-single*)

lemma *monomial-adds-nat-iff*: *monomial p i adds c ⟷ lookup c i ≥ p* **for** *p*::*nat*
apply (*auto simp*: *adds-def lookup-add*)
by (*metis add.left-commute nat-le-iff-add remove-key-sum single-add*)

lemma *update-minus-monomial*: *Poly-Mapping.update k i (m − monomial i k) = Poly-Mapping.update k i m*
by (*auto simp*: *poly-mapping-eq-iff lookup-update update.rep-eq fun-eq-iff lookup-minus lookup-single*)

lemma *monomials-Var*: *monomials (Var x*::*'a*::*zero-neq-one mpoly) = {Poly-Mapping.single x 1}*
by *transfer* (*auto simp*: *Var*$_0$*-def*)

lemma *monomials-Const*: *monomials (Const x) = (if x = 0 then {} else {0})*
by *transfer'* (*auto simp*: *Const*$_0$*-def*)

lemma *coeff-eq-zero-iff*: *MPoly-Type.coeff c p = 0 ⟷ p ∉ monomials c*
by *transfer* (*simp add*: *not-in-keys-iff-lookup-eq-zero*)

lemma *monomials-1*[*simp*]: *monomials 1 = {0}*
by *transfer auto*

lemma *monomials-and-monoms*:
shows *(k ∈ monomials m) = (∃ (a*::*nat). a ≠ 0 ∧ (monomials (MPoly-Type.monom k a)) ⊆ monomials m)*
proof −
show *?thesis* **using** *monomials-monom* **by** *auto*
qed

lemma *mult-monomials-dir-one*:
shows *monomials (p*q) ⊆ {a+b | a b . a ∈ monomials p ∧ b ∈ monomials q}*
using *monomials-and-monoms mult-monom*
by (*simp add*: *keys-mult monomials.rep-eq times-mpoly.rep-eq*)

lemma *monom-eq-zero-iff*[*simp*]: *MPoly-Type.monom a b = 0 ⟷ b = 0*
by (*metis MPolyExtension.coeff-monom MPolyExtension.monom-zero*)

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*

by *transfer auto*

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'*)
by (*subst* ∗) (*rule More-MPoly-Type.coeff-monom-mult*)

lemma *monomials-zero*[*simp*]: *monomials 0* = {}
by *transfer auto*

lemma *in-monomials-iff*: *x* ∈ *monomials m* ⟷ *MPoly-Type.coeff m x* ≠ *0*
using *coeff-eq-zero-iff*[*of m x*] **by** *auto*

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*
apply *auto*
subgoal by *transfer'* (*auto elim*!: *in-keys-timesE*)
subgoal by (*simp add*: *in-monomials-iff More-MPoly-Type.coeff-monom-mult*)
done

3.3 Simplification Lemmas for Const 0 and Const 1

lemma *add-zero* : *P* + *Const 0* = *P*
proof −
have *h*:*P* + *0* = *P* **using** *add-0-right* **by** *auto*
show *?thesis* **unfolding** *Const-def* **using** *h* **by** (*simp add*: *Const*$_0$*-zero zero-mpoly.abs-eq*)
qed

lemma *add-zero-example* : ((*Var 0*)^2 − (*Const 1*)) + *Const 0* = ((*Var 0*)^2 − (*Const 1*))
proof −
show *?thesis* **by** (*simp add* : *add-zero*)
qed

lemma *mult-zero-left* : *Const 0* ∗ *P* = *Const 0*
proof −
have *h*:*0*∗*P* = *0* **by** *simp*
show *?thesis* **unfolding** *Const-def* **using** *h* **by** (*simp add*: *Const*$_0$*-zero zero-mpoly-def*)
qed

lemma *mult-zero-right* : *P* ∗ *Const 0* = *Const 0*
by (*metis mult-zero-left mult-zero-right*)

lemma *mult-one-left* : *Const 1* ∗ (*P* :: *real mpoly*) = *P*
by (*simp add*: *Const.abs-eq Const*$_0$*-one one-mpoly-def*)

lemma *mult-one-right* : (*P*::*real mpoly*) ∗ *Const 1* = *P*
by (*simp add*: *Const.abs-eq Const*$_0$*-one one-mpoly-def*)

3.4 Coefficient Lemmas

lemma *coeff-zero*[*simp*]: *MPoly-Type.coeff 0 x* = *0*
by *transfer auto*

lemma *coeff-sum*: *MPoly-Type.coeff* (*sum f S*) *x* = *sum* (λ*i*. *MPoly-Type.coeff* (*f i*) *x*) *S*
apply (*induction S rule*: *infinite-finite-induct*)
apply (*auto*)
by (*metis More-MPoly-Type.coeff-add*)

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*)
by *transfer*′
(*auto simp*: *Var*$_0$*-def pprod.monomial-power lookup-times-monomial-right of-nat-times-monomial monomial-adds-nat-iff*)

lemma *lookup-update-self*[*simp*]: *Poly-Mapping.update i* (*lookup m i*) *m* = *m*
by (*auto simp*: *lookup-update intro*!: *poly-mapping-eqI*)

lemma *coeff-Const*: *MPoly-Type.coeff* (*Const p*) *m* = (*p when m* = *0*)
by *transfer*′ (*auto simp*: *Const*$_0$*-def lookup-single*)

lemma *coeff-Var*: *MPoly-Type.coeff* (*Var p*) *m* = (*1 when m* = *monomial 1 p*)
by *transfer*′ (*auto simp*: *Var*$_0$*-def lookup-single when-def*)

3.5 Miscellaneous

lemma *update-0-0*[*simp*]: *Poly-Mapping.update x 0 0* = *0*
by (*metis lookup-update-self lookup-zero*)

lemma *mpoly-eq-iff*: *p* = *q* ⟷ (∀ *m*. *MPoly-Type.coeff p m* = *MPoly-Type.coeff q m*)
by *transfer* (*auto simp*: *poly-mapping-eqI*)

lemma *power-both-sides* :
assumes *Ah* : (*A*::*real*) ≥*0*
assumes *Bh* : (*B*::*real*) ≥*0*
shows (*A*≤*B*) = (*A*^*2*≤*B*^*2*)
using *Ah Bh* **by** (*meson power2-le-imp-le power-mono*)

lemma *in-list-induct-helper*:
assumes *set*(*xs*)⊆*X*
assumes *P* []
assumes (⋀*x*. *x*∈*X* ⟹ (⋀*xs*. *P xs* ⟹ *P* (*x* # *xs*)))
shows *P xs* **using** *assms*(*1*)
proof(*induction xs*)

case *Nil*
then show *?case* **using** *assms* **by** *auto*
next
case (*Cons a xs*)
then show *?case* **using** *assms*(*3*) **by** *auto*
qed

theorem *in-list-induct* [*case-names Nil Cons*]:
assumes *P* []
assumes (⋀*x*. *x*∈*set*(*xs*) ⟹ (⋀*xs*. *P xs* ⟹ *P* (*x* # *xs*)))
shows *P xs*
using *in-list-induct-helper*[*of xs set*(*xs*) *P*] **using** *assms* **by** *auto*

3.5.1 Keys and vars

lemma *inKeys-inVars* : *a*≠*0* ⟹ *x* ∈ *keys m* ⟹ *x* ∈ *vars*(*MPoly-Type.monom m a*)
by(*simp add*: *vars-def*)

lemma *notInKeys-notInVars* : *x* ∉ *keys m* ⟹ *x* ∉ *vars*(*MPoly-Type.monom m a*)
by(*simp add*: *vars-def*)

lemma *lookupNotIn* : *x* ∉ *keys m* ⟹ *lookup m x* = *0*
by (*simp add*: *in-keys-iff*)

3.6 Degree Lemmas

lemma *degree-le-iff*: *MPoly-Type.degree p x* ≤ *k* ⟷ (∀ *m*∈*monomials p*. *lookup m x* ≤ *k*)
by *transfer simp*

lemma *degree-less-iff*: *MPoly-Type.degree p x* < *k* ⟷ (*k*>*0* ∧ (∀ *m*∈*monomials p*. *lookup m x* < *k*))
by (*transfer fixing*: *k*) (*cases k* = *0*; *simp*)

lemma *degree-ge-iff*: *k* > *0* ⟹ *MPoly-Type.degree p x* ≥ *k* ⟷ (∃ *m*∈*monomials p*. *lookup m x* ≥ *k*)
using *Max-ge-iff* **by** (*meson degree-less-iff not-less*)

lemma *degree-greater-iff*: *MPoly-Type.degree p x* > *k* ⟷ (∃ *m*∈*monomials p*. *lookup m x* > *k*)
by *transfer*′ (*auto simp*: *Max-gr-iff*)

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*))
by (*subst eq-iff*) (*force simp*: *degree-le-iff degree-ge-iff intro*!: *antisym*)

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*
unfolding *poly-mapping-eq-iff*
by (*force simp*: *update.rep-eq fun-eq-iff*)

lemma *degree-const* : *MPoly-Type.degree* (*Const* (*z*::*real*)) (*x*::*nat*) = *0*
by (*simp add*: *degree-eq-iff monomials-Const*)

lemma *degree-one* : *MPoly-Type.degree* (*Var x* :: *real mpoly*) *x = 1*
by(*simp add*: *degree-eq-iff monomials-Var*)

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)
apply (*transfer fixing*: *x var deg p*)
unfolding *lookup-sum*
unfolding *lookup-single*
apply (*auto simp*: *when-def*)
apply (*subst sum.inter-filter*[*symmetric*])
subgoal by *simp*
subgoal by (*simp add*: *lookup-eq-and-update-lemma Collect-conv-if*)
by (*auto intro*!: *sum.neutral simp add*: *lookup-update*)

lemma *isovarspar-sum*:
isolate-variable-sparse (*p*+*q*) *var deg* =
isolate-variable-sparse (*p*) *var deg*
+ *isolate-variable-sparse* (*q*) *var deg*
apply (*auto simp add*: *mpoly-eq-iff coeff-isolate-variable-sparse*)
apply (*metis More-MPoly-Type.coeff-add coeff-isolate-variable-sparse*)
by (*metis More-MPoly-Type.coeff-add add.comm-neutral coeff-isolate-variable-sparse less-numeral-extra*(*3*))

lemma *isolate-zero*[*simp*]: *isolate-variable-sparse 0 var n = 0*
by *transfer′* (*auto simp*: *mpoly-eq-iff*)

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*)
by (*simp add*: *coeff-isolate-variable-sparse lookup-minus update-minus-monomial*)

lemma *sum-over-zero*: (*mp*::*real mpoly*) = ($\sum$ *i*::*nat* ≤*MPoly-Type.degree mp x*. *isolate-variable-sparse mp x i* ∗ *Var x*^*i*)
by (*auto simp add*: *mpoly-eq-iff coeff-sum coeff-mult-Var if-if-eq-conj not-le coeff-isolate-variable-sparse-minus-monomial when-def lookup-update degree-less-iff simp flip*: *eq-iff intro*!: *coeff-not-in-monomials*)

lemma *const-lookup-zero* : *isolate-variable-sparse* (*Const p* ::*real mpoly*) (*x*::*nat*) (*0*::*nat*) = *Const p*
by (*auto simp*: *mpoly-eq-iff coeff-isolate-variable-sparse coeff-Const when-def*) (*metis lookup-update-self*)

lemma *const-lookup-suc* : *isolate-variable-sparse* (*Const p* :: *real mpoly*) *x* (*Suc i*) = *0*
apply(*auto simp add*: *mpoly-eq-iff coeff-isolate-variable-sparse coeff-Const when-def*)
by (*metis lookup-update lookup-zero nat.distinct*(*1*))

lemma *isovar-greater-degree* : ∀ *i* > *MPoly-Type.degree p var*. *isolate-variable-sparse p var i* = *0*
apply(*auto simp add*: *mpoly-eq-iff coeff-isolate-variable-sparse degree-less-iff*)
by (*metis coeff-not-in-monomials less-irrefl-nat lookup-update*)

lemma *isolate-var-0* : *isolate-variable-sparse* (*Var x*::*real mpoly*) *x 0* = *0*
apply(*auto simp add*: *mpoly-eq-iff coeff-isolate-variable-sparse coeff-Var when-def*)
by (*metis gr0I lookup-single-eq lookup-update-self n-not-Suc-n*)

lemma *isolate-var-one* : *isolate-variable-sparse* (*Var x* :: *real mpoly*) *x 1* = *1*
by (*auto simp add*: *mpoly-eq-iff coeff-isolate-variable-sparse coeff-Var coeff-eq-zero-iff*) (*smt More-MPoly-Type.coeff-monom One-nat-def add-diff-cancel-left' lookup-eq-and-update-lemma lookup-single-eq lookup-update-self monom-one plus-1-eq-Suc single-diff single-zero update-minus-monomial*)

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*)
using *assms*
by (*auto simp add*: *mpoly-eq-iff coeff-isolate-variable-sparse coeff-eq-zero-iff in-keys-iff*) (*metis lookup-update-self*)

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*
by (*simp add*: *isovarspase-monom lookup-eq-zero-in-keys-contradict notInKeys-notInVars*)

lemma *isovarsparNotIn* : *x* ∉ *vars* (*p*::*real mpoly*) ⟹ *isolate-variable-sparse p x*

0 = p
proof(*induction p rule*: *mpoly-induct*)
case (*monom m a*)
then show *?case*
apply(*cases a=0*)
by (*simp-all add*: *isovarspase-monom vars-monom-keys*)
next
case (*sum p1 p2 m a*)
then show *?case*
by (*simp add*: *monomials.rep-eq vars-add-monom isovarspar-sum*)
qed

lemma *degree0isovarspar* :
assumes *deg0* : *MPoly-Type.degree* (*p*::*real mpoly*) *x = 0*
shows *isolate-variable-sparse p x 0 = p*
proof −
have *h1* : *p* = (∑ *i*::*nat* ≤*MPoly-Type.degree p x*. *isolate-variable-sparse p x i ∗ Var x* ^ *i*)
using *sum-over-zero* **by** *auto*
have *h2a* : ∀ *f*. (∑ *i*::*nat* ≤*0*. *f i*) = *f 0*
apply(*simp add*: *sum-def*)
by (*metis add.right-neutral comm-monoid-add-class.sum-def finite.emptyI insert-absorb insert-not-empty sum.empty sum.insert*)
have *h2* : *p = isolate-variable-sparse p x 0 ∗ Var x* ^ *0*
using *deg0 h1 h2a* **by** *auto*
show *?thesis* **using** *h2*
by *auto*
qed

3.8 Summation Lemmas

lemma *summation-normalized* :
assumes *nonzero* : (*B* ::*real*) ≠*0*
shows (∑ *i = 0*..<((*n*::*nat*)+*1*). (*f i* :: *real*) ∗ *B* ^(*n* − *i*)) = (∑ *i = 0*..<(*n*+*1*). (*f i*) / (*B* ^ *i*)) ∗ (*B*^*n*)
proof −
have *h1a* : ∀ *x*::*real*. ((∑ *i = 0*..<(*n*+*1*). (*f i*) / (*B* ^ *i*)) ∗ *x* = (∑ *i = 0*..<(*n*+*1*). ((*f i*) / (*B* ^ *i*)) ∗ *x*))
apply(*induction n*)
apply(*auto*)
by (*simp add*: *distrib-right*)
have *h1* : (∑ *i = 0*..<(*n*+*1*). (*f i*) / (*B* ^ *i*)) ∗ (*B*^*n*) = (∑ *i = 0*..<(*n*+*1*). ((*f i*) / (*B* ^ *i*)) ∗ (*B*^*n*))
using *h1a* **by** *auto*
have *h2* : (∑ *i = 0*..<(*n*+*1*). ((*f i*) / (*B* ^ *i*)) ∗ (*B*^*n*)) = (∑ *i = 0*..<(*n*+*1*). (*f i*) ∗ ((*B*^*n*) / (*B* ^ *i*)))
by *auto*
have *h3* : (∑ *i = 0*..<(*n*+*1*). (*f i*) ∗ ((*B*^*n*) / (*B* ^ *i*))) = (∑ *i = 0*..<(*n*+*1*).

$(f\ i) * B\ \hat{}\ (n - i))$
using *add.right-neutral nonzero power-diff* **by** *fastforce*
show *?thesis* **using** *h1 h2 h3* **by** *auto*
qed

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 −
have *pow-zero* : $\forall i.\ B\hat{}(i :: nat)\neq 0$ **using** *nonzero* **by**(*auto*)
have *single-division-zero* : $\forall X.\ B*(X::real)=0 \longleftrightarrow X=0$ **using** *nonzero* **by**(*auto*)
have *h1*: $(\sum i = 0..<n+1.\ (f\ i)\ /\ (B\ \hat{}\ i)) = 0 \longleftrightarrow ((\sum i = 0..<n+1.\ (f\ i)\ /\ (B\ \hat{}\ i)))*B\hat{}n = 0$
using *nonzero single-division-zero* **by**(*auto*)
have *h2*: $((\sum i = 0..<n+1.\ (f\ i)\ /\ (B\ \hat{}\ i))*(B\hat{}n)) = ((\sum i = 0..<n+1.\ (f\ i) * (B\ \hat{}\ (n- i))))$
using *summation-normalized nonzero* **by** *auto*
show *?thesis* **using** *h1 h2* **by** *auto*
qed

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 −
have *h1* : $(\sum i = 0..<(n+1).\ (f\ i) * B\ \hat{}\ (n - i)) * B\ \hat{}\ (n\ mod\ 2) < 0$
$\longleftrightarrow\ (\sum i = 0..<(n+1).\ (f\ i)\ /\ (B\ \hat{}\ i)) * (B\hat{}n) * B\ \hat{}\ (n\ mod\ 2) < 0$
using *summation-normalized nonzero* **by** *auto*
have *h2a* : $n\ mod\ 2 = 0 \vee n\ mod\ 2 = 1$ **by** *auto*
have *h2b* : $n\ mod\ 2 = 1 \Longrightarrow odd\ n$ **by** *auto*
have *h2c* : $(B\hat{}n) * B\ \hat{}\ (n\ mod\ 2) > 0$
using *h2a h2b* **apply** *auto*
using *nonzero* **apply** *presburger*
by (*metis even-Suc mult.commute nonzero power-Suc zero-less-power-eq*)
have *h2* : $\forall x.\ ((x * (B\hat{}n) * B\ \hat{}\ (n\ mod\ 2) < 0) = (x<0))$
using *h2c* **using** *mult.assoc* **by** (*metis mult-less-0-iff not-square-less-zero*)
show *?thesis* **using** *h1 h2* **by** *auto*
qed

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*)
apply(*simp add*: *isolate-variable-sparse-def vars-def*)
proof −

assume *a1*: *MPoly* ($\sum m \mid m \in$ *monomials p* $\wedge$ *lookup m var* $=$ *x*. *monomial* (*MPoly-Type.coeff p m*) (*Poly-Mapping.update var 0 m*)) $=$ *q*
{ fix *pp* :: *nat* $\Rightarrow_0$ *nat*
have *isolate-variable-sparse p var x* $=$ *q*
using *a1 isolate-variable-sparse.abs-eq* **by** *blast*
then have *var* $\notin$ *keys pp* $\vee$ *pp* $\notin$ *keys* (*mapping-of q*)
by (*metis* (*no-types*) *coeff-def coeff-isolate-variable-sparse in-keys-iff*) **}**
then show $\forall$ *p*$\in$*keys* (*mapping-of q*). *var* $\notin$ *keys p*
by *meson*
qed

lemma *not-in-add* : *var*$\notin$(*vars* (*p*::*real mpoly*)) $\Longrightarrow$ *var*$\notin$(*vars* (*q*::*real mpoly*)) $\Longrightarrow$ *var*$\notin$(*vars* (*p*+*q*))
by (*meson UnE in-mono vars-add*)

lemma *not-in-mult* : *var*$\notin$(*vars* (*p*::*real mpoly*)) $\Longrightarrow$ *var*$\notin$(*vars* (*q*::*real mpoly*)) $\Longrightarrow$ *var*$\notin$(*vars* (*p*$*$*q*))
by (*meson UnE in-mono vars-mult*)

lemma *not-in-neg* : *var*$\notin$(*vars*(*p*::*real mpoly*)) $\longleftrightarrow$ *var*$\notin$(*vars*(−*p*))
proof −
have *h*: *var* $\notin$ (*vars* (−*1*::*real mpoly*)) **by** (*metis add-diff-cancel-right' add-neg-numeral-special*(*8*) *isolate-var-one isolate-zero isovarsparNotIn isovarspar-sum not-in-isovarspar*)
show *?thesis* **using** *not-in-mult* **using** *h* **by** *fastforce*
qed

lemma *not-in-sub* : *var*$\notin$(*vars* (*p*::*real mpoly*)) $\Longrightarrow$ *var*$\notin$(*vars* (*q*::*real mpoly*)) $\Longrightarrow$ *var*$\notin$(*vars* (*p*−*q*))
using *not-in-add not-in-neg* **by** *fastforce*

lemma *not-in-pow* : *var*$\notin$(*vars*(*p*::*real mpoly*)) $\Longrightarrow$ *var*$\notin$(*vars*(*p*^*i*))
proof (*induction i*)
case *0*
then show *?case* **using** *isolate-var-one not-in-isovarspar*
by (*metis power-0*)
next
case (*Suc i*)
then show *?case* **using** *not-in-mult* **by** *simp*
qed

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 (*induction n*)
case *0*
then show *?case* **using** *isolate-zero not-in-isovarspar* **by** *fastforce*
next
case (*Suc n*)
have *h1*: (*sum f* {*0*..<*Suc n*}) $=$ (*sum f* {*0*..< *n*}) + (*f n*) **using** *sum.atLeast0-lessThan-Suc*
by *blast*

have *h2*: *var* ∉ *vars* (*f n*) **by** (*simp add*: *Suc.prems*)
then show *?case* **using** *h1 vars-add* **by** (*simp add*: *Suc.IH Suc.prems not-in-add*)
qed

lemma *not-in-sum* : (∀ *i*. *var*∉(*vars*(*f*(*i*)::*real mpoly*))) ⟹ ∀ (*n*::*nat*). *var*∉(*vars*(∑ *i*∈{*0*..<*n*}.*f*(*i*)))
using *not-in-sum-var* **by** *blast*

lemma *not-contains-insertion-helper* :
∀ *x*∈*keys* (*mapping-of p*). *var* ∉ *keys x* ⟹
(⋀*k f*. (*k* ∉ *keys f*) = (*lookup f k* = *0*)) ⟹
lookup (*mapping-of p*) *a* = *0* ∨
(∏ *aa*. (*if aa* < *length L then L*[*var* := *y*] ! *aa else 0*) ^ *lookup a aa*) =
(∏ *aa*. (*if aa* < *length L then L*[*var* := *x*] ! *aa else 0*) ^ *lookup a aa*)
apply(*cases lookup* (*mapping-of p*) *a* = *0*)
apply *auto*
apply(*rule Prod-any.cong*)
apply *auto*
using *lookupNotIn nth-list-update-neq power-0* **by** *smt*

lemma *not-contains-insertion* :
assumes *notIn* : *var* ∉ *vars* (*p*:: *real mpoly*)
assumes *exists* : *insertion* (*nth-default 0* (*list-update L var x*)) *p* = *val*
shows *insertion* (*nth-default 0* (*list-update L var y*)) *p* = *val*
using *notIn exists*
apply(*simp add*: *insertion-def insertion-aux-def insertion-fun-def*)
unfolding *vars-def nth-default-def*
using *not-in-keys-iff-lookup-eq-zero*
apply *auto*
apply(*rule Sum-any.cong*)
apply *simp*
using *not-contains-insertion-helper*[*of p var - L y x*]
proof −
fix *a* :: *nat* ⇒$_0$ *nat*
assume *a1*: ∀ *x*∈*keys* (*mapping-of p*). *var* ∉ *keys x*
assume ⋀*k f*. ((*k*::*'a*) ∉ *keys f*) = (*lookup f k* = *0*)
then show *lookup* (*mapping-of p*) *a* = *0* ∨ (∏ *n*. (*if n* < *length L then L*[*var* := *y*] ! *n else 0*) ^ *lookup a n*) = (∏ *n*. (*if n* < *length L then L*[*var* := *x*] ! *n else 0*) ^ *lookup a n*)
using *a1* ‹⋀*a*. ⟦∀ *x*∈*keys* (*mapping-of p*). *var* ∉ *keys x*; ⋀*k f*. (*k* ∉ *keys f*) = (*lookup f k* = *0*)⟧ ⟹ *lookup* (*mapping-of p*) *a* = *0* ∨ (∏ *aa*. (*if aa* < *length L then L*[*var* := *y*] ! *aa else 0*) ^ *lookup a aa*) = (∏ *aa*. (*if aa* < *length L then L*[*var* := *x*] ! *aa else 0*) ^ *lookup a aa*)› **by** *blast*
qed

3.10 Insertion Lemmas

lemma *insertion-sum-var* : ((*insertion f* (∑ *i*∈{*0*..<(*n*::*nat*)}.*g*(*i*))) = (∑ *i*∈{*0*..<*n*}. *insertion f* (*g i*)))
proof (*induction n*)

case *0*
then show *?case* **by** *auto*
next
case (*Suc n*)
then show *?case* **by** (*simp add*: *insertion-add*)
qed

lemma *insertion-sum* : $\forall$ (*n*::*nat*). ((*insertion f* ($\sum$ *i*$\in${*0*..<*n*}.*g*(*i*))) = ($\sum$ *i*$\in${*0*..<*n*}. *insertion f* (*g i*)))
using *insertion-sum-var* **by** *blast*

lemma *insertion-sum'* : $\bigwedge$(*n*::*nat*). ((*insertion f* ($\sum$ *i*$\leq$*n*. *g*(*i*))) = ($\sum$ *i*$\leq$*n*. *insertion f* (*g i*)))
by (*metis* (*no-types*, *lifting*) *fun-sum-commute insertion-add insertion-zero sum.cong*)

lemma *insertion-pow* : *insertion f* (*p*^*i*) = (*insertion f p*)^*i*
proof (*induction i*)
case *0*
then show *?case* **by** *auto*
next
case (*Suc n*)
then show *?case* **by** (*simp add*: *insertion-mult*)
qed

lemma *insertion-neg* : *insertion f* (−*p*) = −*insertion f p*
by (*metis add.inverse-inverse insertionNegative*)

lemma *insertion-var* :
length L > *var* $\Longrightarrow$ *insertion* (*nth-default 0* (*list-update L var x*)) (*Var var*) = *x*
by (*auto simp*: *monomials-Var coeff-Var insertion-code nth-default-def*)

lemma *insertion-var-zero* : *insertion* (*nth-default 0* (*x*#*xs*)) (*Var 0*) = *x* **using** *insertion-var*
by *fastforce*

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*
by (*metis ab-group-add-class.ab-diff-conv-add-uminus insertion-add insertion-neg*)

lemma *insertion-sub* : *insertion f* (*A*−*B* :: *real mpoly*) = *insertion f A* − *insertion f B*
using *insertion-neg insertion-add*
by (*metis uminus-add-conv-diff*)

lemma *insertion-four* : *insertion* ((*nth-default 0*) *xs*) *4* = *4*
by (*metis* (*no-types*, *lifting*) *insertion-add insertion-one numeral-plus-numeral*

one-add-one semiring-norm(2) semiring-norm(6))

lemma *insertion-add-ind-basecase*:
*insertion (nth (list-update L var x)) ((∑ i::nat ≤ 0. isolate-variable-sparse p var i * (Var var)^i))*
*= (∑ i = 0..<(0+1). insertion (nth (list-update L var x)) (isolate-variable-sparse p var i * (Var var)^i))*
proof −
have *h1*: *((∑ i::nat ≤ 0. isolate-variable-sparse p var i * (Var var)^i))*
*= (isolate-variable-sparse p var 0 * (Var var)^0)*
by *auto*
show *?thesis* **using** *h1*
by *auto*
qed

lemma *insertion-add-ind*:
*insertion (nth-default 0 (list-update L var x)) ((∑ i::nat ≤ d. isolate-variable-sparse p var i * (Var var)^i))*
*= (∑ i = 0..<(d+1). insertion (nth-default 0 (list-update L var x)) (isolate-variable-sparse p var i * (Var var)^i))*
proof (*induction d*)
case *0*
then show *?case* **using** *insertion-add-ind-basecase nth-default-def*
by *auto*
next
case (*Suc n*)
then show *?case* **using** *insertion-add* **apply** *auto*
by (*simp add*: *insertion-add*)
qed

lemma *sum-over-degree-insertion* :
assumes *lLength* : *length L > var*
assumes *deg* : *MPoly-Type.degree (p::real mpoly) var = d*
shows *(∑ 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 −
have *h1*: *(p::real mpoly) = (∑ i::nat ≤(MPoly-Type.degree p var). isolate-variable-sparse p var i * (Var var)^i)* **using** *sum-over-zero* **by** *auto*
have *h2*: *insertion (nth-default 0 (list-update L var x)) p =*
*insertion (nth-default 0 (list-update L var x)) ((∑ i::nat ≤ d. isolate-variable-sparse p var i * (Var var)^i))* **using** *h1 deg* **by** *auto*
have *h3*: *insertion (nth-default 0 (list-update L var x)) p = (∑ i = 0..<(d+1). insertion (nth-default 0 (list-update L var x)) (isolate-variable-sparse p var i * (Var var)^i))*
using *h2 insertion-add-ind nth-default-def*
by *simp*
show *?thesis* **using** *h3 insertion-var insertion-pow*
by (*metis* (*no-types*, *lifting*) *insertion-mult lLength sum.cong*)

qed

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 −
have *h* : *var* ∉ *vars*(*isolate-variable-sparse* (*p*::*real mpoly*) *var* (*i*::*nat*))
by (*simp add*: *not-in-isovarspar*)
then show *?thesis* **using** *not-contains-insertion*
by *blast*
qed
lemma *insertion-zero* : *insertion f* (*Const 0* ::*real mpoly*) = *0*
by (*metis add-cancel-right-right add-zero insertion-zero*)

lemma *insertion-one* : *insertion f* (*Const 1* ::*real mpoly*) = *1*
by (*metis insertion-one mult.right-neutral mult-one-left*)

lemma *insertion-const* : *insertion f* (*Const c*::*real mpoly*) = (*c*::*real*)
by (*auto simp*: *monomials-Const coeff-Const insertion-code*)

3.11 Putting Things Together

3.11.1 More Degree Lemmas

lemma *degree-add-leq* :
assumes *h1* : *MPoly-Type.degree a var* ≤ *x*
assumes *h2* : *MPoly-Type.degree b var* ≤ *x*
shows *MPoly-Type.degree* (*a*+*b*) *var* ≤ *x*
using *degree-eq-iff coeff-add coeff-not-in-monomials*
by (*smt* (*z3*) *More-MPoly-Type.coeff-add add.left-neutral coeff-eq-zero-iff degree-le-iff h1 h2*)

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*
by (*metis degree-add-leq h1 h2 linorder-not-le nat-less-le*)

lemma *degree-sum* : (∀ *i*∈{*0*..*n*::*nat*}. *MPoly-Type.degree* (*f i* :: *real mpoly*) *var* ≤ *x*) ⟹ (*MPoly-Type.degree* (∑ *x*∈{*0*..*n*}. *f x*) *var*) ≤ *x*
proof(*induction n*)
case *0*
then show *?case* **by** *auto*
next
case (*Suc n*)
then show *?case* **by**(*simp add*: *degree-add-leq*)

qed

lemma *degree-sum-less* : $(\forall i{\in}\{0..n{::}nat\}.\ MPoly\text{-}Type.degree\ (f\ i :: real\ mpoly)\ var < x) \Longrightarrow (MPoly\text{-}Type.degree\ (\sum x{\in}\{0..n\}.\ f\ x)\ var) < x$
proof(*induction n*)
case *0*
then show *?case* **by** *auto*
next
case (*Suc n*)
then show *?case* **by**(*simp add*: *degree-add-less*)
qed

lemma *varNotIn-degree* :
assumes $var \notin vars\ p$
shows $MPoly\text{-}Type.degree\ p\ var = 0$
proof−
have $\forall m{\in}monomials\ p.\ lookup\ m\ var = 0$
using *assms* **unfolding** *vars-def keys-def*
using *monomials.rep-eq* **by** *fastforce*
then show *?thesis*
using *degree-less-iff* **by** *blast*
qed

lemma *degree-mult-leq* :
assumes *pnonzero*: $(p{::}real\ mpoly){\neq}0$
assumes *qnonzero*: $(q{::}real\ mpoly){\neq}0$
shows $MPoly\text{-}Type.degree\ ((p{::}real\ mpoly) * (q{::}real\ mpoly))\ var \leq (MPoly\text{-}Type.degree\ p\ var) + (MPoly\text{-}Type.degree\ q\ var)$
proof(*cases* $MPoly\text{-}Type.degree\ (p{*}q)\ var = 0$)
case *True*
then show *?thesis* **by** *simp*
next
case *False*
have *hp*: $\forall m{\in}monomials\ p.\ lookup\ m\ var \leq MPoly\text{-}Type.degree\ p\ var$ **using** *degree-eq-iff*
by (*metis zero-le*)
have *hq*: $\forall m{\in}monomials\ q.\ lookup\ m\ var \leq MPoly\text{-}Type.degree\ q\ var$ **using** *degree-eq-iff*
by (*metis zero-le*)
have *hpq*: $\forall m{\in}\{a+b \mid a\ b\ .\ a \in monomials\ p \wedge b \in monomials\ q\}.$
$lookup\ m\ var \leq (MPoly\text{-}Type.degree\ p\ var) + (MPoly\text{-}Type.degree\ q\ var)$
by (*smt add-le-mono hp hq mem-Collect-eq plus-poly-mapping.rep-eq*)
have *h1*: $(\forall m{\in}monomials\ (p{*}q).\ lookup\ m\ var \leq (MPoly\text{-}Type.degree\ p\ var) + (MPoly\text{-}Type.degree\ q\ var))$
using *mult-monomials-dir-one hpq*
by *blast*
then show *?thesis* **using** *h1 degree-eq-iff False*
by (*simp add*: *degree-le-iff*)

qed

lemma *degree-exists-monom*:
assumes *p≠0*
shows ∃ *m*∈*monomials p*. *lookup m var* = *MPoly-Type.degree p var*
proof(*cases MPoly-Type.degree p var* =*0*)
case *True*
have *h1*: ∃ *m*∈*monomials p*. *lookup m var* = *0* **unfolding** *monomials-def*
by (*metis True assms*(*1*) *aux degree-eq-iff in-keys-iff mapping-of-inject monomials.rep-eq monomials-def zero-mpoly.rep-eq*)
then show *?thesis* **using** *h1*
using *True* **by** *simp*
next
case *False*
then show *?thesis* **using** *degree-eq-iff assms*(*1*) **apply**(*auto*)
by (*metis degree-eq-iff dual-order.strict-iff-order*)
qed

lemma *degree-not-exists-monom*:
assumes *p≠0*
shows ¬ (∃ *m*∈*monomials p*. *lookup m var* > *MPoly-Type.degree p var*)
proof −
show *?thesis* **using** *degree-less-iff* **by** *blast*
qed

lemma *degree-monom*: *MPoly-Type.degree* (*MPoly-Type.monom x y*) *v* = (*if y* = *0 then 0 else lookup x v*)
by (*auto simp*: *degree-eq-iff*)

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*
using *that*
apply (*subst degree-eq-iff*)
apply (*auto simp*: *monomials-add-disjoint*)
apply (*auto simp*: *degree-eq-iff degree-monom*)
apply (*metis MPoly-Type.degree-zero degree-exists-monom less-numeral-extra*(*3*))
using *degree-le-iff* **apply** *blast*
using *degree-eq-iff*
apply (*metis max-def neq0-conv*)
apply (*metis degree-eq-iff max.coboundedI1 neq0-conv*)
apply (*metis MPoly-Type.degree-zero degree-exists-monom max-def zero-le*)
using *degree-le-iff max.cobounded1* **by** *blast*

3.11.2 More isolate_variable_sparse lemmas

lemma *isolate-variable-sparse-ne-zeroD*:

isolate-variable-sparse mp v x $\neq$ *0* $\Longrightarrow$ *x* $\leq$ *MPoly-Type.degree mp v*
using *isovar-greater-degree leI* **by** *blast*

context includes *poly.lifting* **begin**
lift-definition *mpoly-to-nested-poly*::*'a*::*comm-monoid-add mpoly* $\Rightarrow$ *nat* $\Rightarrow$ *'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
unfolding *MOST-iff-cofinite*
proof −
fix *mp*::*'a mpoly* **and** *v*::*nat*
have {*p*. *isolate-variable-sparse mp v p* $\neq$ *0*} $\subseteq$ {*0*..*MPoly-Type.degree mp v*}
(**is** *?s* $\subseteq$ -)
by (*auto dest*!: *isolate-variable-sparse-ne-zeroD*)
also have *finite* ... **by** *simp*
finally (*finite-subset*) **show** *finite ?s* **.**
qed

lemma *degree-eq-0-mpoly-to-nested-polyI*:
mpoly-to-nested-poly mp v = *0* $\Longrightarrow$ *MPoly-Type.degree mp v* = *0*
apply *transfer'*
apply (*simp add*: *degree-eq-iff*)
apply *transfer'*
apply (*auto simp*: *fun-eq-iff*)
proof −
fix *mpa* :: *'a mpoly* **and** *va* :: *nat* **and** *m* :: *nat* $\Rightarrow_0$ *nat*
assume *a1*: $\forall$ *x*. ($\sum$ *m* | *m* $\in$ *monomials mpa* $\wedge$ *lookup m va* = *x*. *monomial* (*MPoly-Type.coeff mpa m*) (*Poly-Mapping.update va 0 m*)) = *0*
assume *a2*: *m* $\in$ *monomials mpa*
have *f3*: $\forall$ *m p*. *lookup* (*mapping-of m*) *p* $\neq$ (*0*::*'a*) $\vee$ *p* $\notin$ *monomials m*
by (*metis* (*full-types*) *coeff-def coeff-eq-zero-iff*)
have *f4*: $\forall$ *n*. *mapping-of* (*isolate-variable-sparse mpa va n*) = *0*
using *a1* **by** (*simp add*: *isolate-variable-sparse.rep-eq*)
have $\forall$ *p n*. *lookup 0* (*p*::*nat* $\Rightarrow_0$ *nat*) = (*0*::*'a*) $\vee$ (*0*::*nat*) = *n*
by *simp*
then show *lookup m va* = *0*
using *f4 f3 a2* **by** (*metis coeff-def coeff-isolate-variable-sparse lookup-eq-and-update-lemma*)
qed

lemma *coeff-eq-zero-mpoly-to-nested-polyD*: *mpoly-to-nested-poly mp v* = *0* $\Longrightarrow$ *MPoly-Type.coeff mp 0* = *0*
apply *transfer'*
apply *transfer'*
by (*metis* (*no-types*) *coeff-def coeff-isolate-variable-sparse isolate-variable-sparse.rep-eq lookup-zero update-0-0*)

lemma *mpoly-to-nested-poly-eq-zero-iff*[*simp*]:
mpoly-to-nested-poly mp v = *0* $\longleftrightarrow$ *mp* = *0*
apply (*auto simp*: *coeff-eq-zero-mpoly-to-nested-polyD degree-eq-0-mpoly-to-nested-polyI*)

proof −
show *mpoly-to-nested-poly mp v = 0 ⟹ mp = 0*
apply (*frule degree-eq-0-mpoly-to-nested-polyI*)
apply (*frule coeff-eq-zero-mpoly-to-nested-polyD*)
apply (*transfer′ fixing*: *mp v*)
apply (*transfer′ fixing*: *mp v*)
apply (*auto simp*: *fun-eq-iff mpoly-eq-iff intro*!: *sum.neutral*)
proof −
fix *m* :: *nat* ⇒$_0$ *nat*
assume *a1*: ∀ *x*. (∑ *m* | *m* ∈ *monomials mp* ∧ *lookup m v* = *x*. *monomial* (*MPoly-Type.coeff mp m*) (*Poly-Mapping.update v 0 m*)) = *0*
assume *a2*: *MPoly-Type.degree mp v* = *0*
have ∀ *n*. *isolate-variable-sparse mp v n* = *0*
using *a1* **by** (*simp add*: *isolate-variable-sparse.abs-eq zero-mpoly.abs-eq*)
then have *f3*: ∀ *p*. *MPoly-Type.coeff mp p* = *MPoly-Type.coeff 0 p* ∨ *lookup p v* ≠ *0*
by (*metis* (*no-types*) *coeff-isolate-variable-sparse lookup-update-self*)
then show *MPoly-Type.coeff mp m* = *0*
using *a2 coeff-zero*
by (*metis coeff-not-in-monomials degree-eq-iff*)
qed
show *mp* = *0* ⟹ *mpoly-to-nested-poly 0 v* = *0*
subgoal
apply *transfer′*
apply *transfer′*
by (*auto simp*: *fun-eq-iff intro*!: *sum.neutral*)
done
qed

lemma *isolate-variable-sparse-degree-eq-zero-iff*: *isolate-variable-sparse p v* (*MPoly-Type.degree p v*) = *0* ⟷ *p* = *0*
apply (*transfer′*)
apply *auto*
proof −
fix *pa* :: *′a mpoly* **and** *va* :: *nat*
assume (∑ *m* | *m* ∈ *monomials pa* ∧ *lookup m va* = *MPoly-Type.degree pa va*. *monomial* (*MPoly-Type.coeff pa m*) (*Poly-Mapping.update va 0 m*)) = *0*
then have *mapping-of* (*isolate-variable-sparse pa va* (*MPoly-Type.degree pa va*)) = *0*
by (*simp add*: *isolate-variable-sparse.rep-eq*)
then show *pa* = *0*
by (*metis* (*no-types*) *coeff-def coeff-eq-zero-iff coeff-isolate-variable-sparse degree-exists-monom lookup-eq-and-update-lemma lookup-zero*)
qed

lemma *degree-eq-univariate-degree*: *MPoly-Type.degree p v* =
(*if p* = *0 then 0 else Polynomial.degree* (*mpoly-to-nested-poly p v*))
apply *auto*
apply (*rule antisym*)

```
  subgoal
    apply (rule Polynomial.le-degree)
    apply auto
    apply transfer'
    by (simp add: isolate-variable-sparse-degree-eq-zero-iff)
  subgoal apply (rule Polynomial.degree-le)
    apply (auto simp: elim!: degree-eq-zeroE)
    apply transfer'
    by (simp add: isovar-greater-degree)
  done

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)])
  unfolding coeffs-def unfolding mpoly-to-nested-poly-eq-zero-iff degree-eq-univariate-degree
apply auto
  subgoal by transfer' (rule refl)
  by transfer' (rule refl)

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 −
  have ∗: {x. x = m ∧ lookup x v = i} = (if lookup m v = i then {m} else {})
    by auto
  show ?thesis
    by (transfer' fixing: m a v i) (simp add:∗)
qed



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
  apply (auto simp: MPoly-Type.mult-monom)
  subgoal
    apply transfer'
    subgoal for mon v i a q
      apply (auto simp add: monomials-monom-mult sum-distrib-left)
      apply (rule sum.reindex-bij-witness-not-neutral[where
          j=λa. a − mon
```

and $i{=}\lambda a.\ mon + a$
and $S'{=}\{\}$
and $T'{=}\{\}$
])
apply (*auto simp*: *lookup-add*)
apply (*auto simp*: *poly-mapping-eq-iff fun-eq-iff single.rep-eq Poly-Mapping.mult-single*)
apply (*auto simp*: *when-def More-MPoly-Type.coeff-monom-mult*)
by (*auto simp*: *lookup-update lookup-add split*: *if-splits*)
done
subgoal
apply *transfer'*
apply (*auto intro*!: *sum.neutral simp*: *monomials-monom-mult*)
apply (*rule poly-mapping-eqI*)
apply (*auto simp*: *lookup-single when-def*)
by (*simp add*: *lookup-add*)
done

lemma *isolate-variable-sparse-mult*:
isolate-variable-sparse $(p * q)\ v\ n = (\sum i{\leq}n.$ *isolate-variable-sparse* $p\ v\ i *$
isolate-variable-sparse $q\ v\ (n - i))$
for $p\ q{::}'a{::}$*semiring-no-zero-divisors mpoly*
proof (*induction p rule*: *mpoly-induct*)
case (*monom m a*)
then show *?case*
by (*cases* $a = 0$)
(*auto simp add*: *mpoly-eq-iff coeff-sum coeff-mult if-conn if-distrib if-distribR*
isolate-variable-sparse-monom isolate-variable-sparse-monom-mult
cong: *if-cong*)
next
case (*sum p1 p2 m a*)
then show *?case*
by (*simp add*: *distrib-right isovarspar-sum sum.distrib*)
qed

3.11.3 More Miscellaneous

lemma *var-not-in-Const* : *var*$\notin$*vars* (*Const x* :: *real mpoly*)
unfolding *vars-def keys-def*
by (*smt UN-iff coeff-def coeff-isolate-variable-sparse const-lookup-zero keys-def*
lookup-eq-zero-in-keys-contradict)

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*
by (*auto simp*: *poly-eq-iff coeff-mult mpoly-to-nested-poly.rep-eq isolate-variable-sparse-mult*)

lemma *get-if-const-insertion* :
assumes *get-if-const* (*p*::*real mpoly*) = *Some r*

shows *insertion f p = r*
proof−
have *Set.is-empty (vars p)*
apply(*cases Set.is-empty (vars p)*)
apply(*simp*) **using** *assms* **by**(*simp*)
then have *(MPoly-Type.coeff p 0) = r*
using *assms* **by** *simp*
then show *?thesis*
by (*metis Set.is-empty-def ‹Set.is-empty (vars p)› empty-iff insertion-irrelevant-vars insertion-trivial*)
qed

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*
using *assms*
by (*auto simp add*: *degree-eq-univariate-degree mpoly-to-nested-poly-mult Polynomial.degree-mult-eq*)

lemma *degree-eq*:
assumes (*p*::*real mpoly*) = (*q*:: *real mpoly*)
shows *MPoly-Type.degree p x = MPoly-Type.degree q x*
by (*simp add*: *assms*)

lemma *degree-var-i* : *MPoly-Type.degree (((Var x)^i:: real mpoly)) x = i*
proof (*induct i*)
case *0*
then show *?case* **using** *degree-const*
by *simp*
next
case (*Suc i*)
have *multh*: (*Var x*)^(*Suc i*) = ((*Var x*)^*i*::*real mpoly*) * (*Var x*:: *real mpoly*)
using *power-Suc2* **by** *blast*
have *deg0h*: *MPoly-Type.degree 0 x = 0*
by *simp*
have *deg1h*: *MPoly-Type.degree* (*Var x*::*real mpoly*) *x = 1*
using *degree-one*
by *blast*
have *nonzeroh1*: (*Var x*:: *real mpoly*) *≠ 0*
using *degree-eq deg0h deg1h* **by** *auto*
have *nonzeroh2*: ((*Var x*)^*i*:: *real mpoly*) *≠ 0*
using *degree-eq deg0h Suc.hyps*
by (*metis one-neq-zero power-0*)
have *sumh*: (*MPoly-Type.degree* (((*Var x*)^*i*:: *real mpoly*) * (*Var x*:: *real mpoly*))

x) = (*MPoly-Type.degree* ((*Var x*)^*i*::*real mpoly*) *x*) + (*MPoly-Type.degree* (*Var x*::*real mpoly*) *x*)
using *degree-mult*[**where** *p* = (*Var x*)^*i*::*real mpoly*, **where** *q* = *Var x*::*real mpoly*] *nonzeroh1 nonzeroh2*
by *blast*
then show *?case* **using** *sumh deg1h*
by (*metis Suc.hyps Suc-eq-plus1 multh*)
qed

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 −
have *h1*: *n* > *0*
using *assms*(*3*) *neq0-conv* **by** *blast*
have *h2*: (∃ *m*∈*monomials p*. *lookup m var* = *n*) ∧ (∀ *m*∈*monomials p*. *lookup m var* ≤ *n*)
using *degree-eq-iff assms*(*1*)
by (*metis degree-ge-iff h1 order-refl*)
have *h3*: (∃ *m*∈*monomials* (*p* + *q*). *lookup m var* = *n*)
using *h2*
by (*metis MPolyExtension.coeff-add add.right-neutral assms*(*2*) *assms*(*3*) *coeff-eq-zero-iff degree-not-exists-monom*)
have *h4*: (∀ *m*∈*monomials* (*p* + *q*). *lookup m var* ≤ *n*)
using *h2 assms*(*3*) *assms*(*2*)
using *degree-add-leq degree-le-iff dual-order.strict-iff-order* **by** *blast*
show *?thesis* **using** *degree-eq-iff h3 h4*
by (*metis assms*(*3*) *gr-implies-not0*)
qed

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* **using** *degree-less-sum*[*OF assms*(*2*) *assms*(*1*) *assms*(*3*)]
by (*simp add*: *add.commute*)

lemma *nonzero-const-is-nonzero*:
assumes (*k*::*real*) ≠ *0*
shows ((*Const k*)::*real mpoly*) ≠ *0*
by (*metis MPoly-Type.insertion-zero assms insertion-const*)

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 −
define *f* **where** *f i = (isolate-variable-sparse p x (i+1) * (Var x)^(i) * (Const (i+1)))* **for** *i*
define *n* **where** *n = MPoly-Type.degree p x−1*
have *d* : ∃ *m*∈*monomials p. lookup m x = n+1*
using *n-def degree-eq-iff assms*
by (*metis add.commute less-not-refl2 less-one linordered-semidom-class.add-diff-inverse*)

have *h1a* : ∀ *i. MPoly-Type.degree (isolate-variable-sparse p x i) x = 0*
by (*simp add*: *not-in-isovarspar varNotIn-degree*)
have *h1b* : ∀ *i*::*nat. MPoly-Type.degree ((Var x)^i:: real mpoly) x = i*
using *degree-var-i* **by** *auto*
have *h1c1* : ∀ *i. (Var(x)^(i)::real mpoly)≠0*
by (*metis MPoly-Type.degree-zero h1b power-0 zero-neq-one*)
fix *i*
have *h1c1var*: *((Var x)^i:: real mpoly) ≠ 0* **using** *h1c1* **by** *blast*
have *h1c2* : *((Const ((i::nat) + 1))::real mpoly)≠0*
using *nonzero-const-is-nonzero*
by *auto*
have *h1c3*: *(isolate-variable-sparse p x (n + 1)) ≠ 0* **using** *d*
by (*metis One-nat-def Suc-pred add.commute assms isolate-variable-sparse-degree-eq-zero-iff n-def not-gr-zero not-in-isovarspar plus-1-eq-Suc varNotIn-degree*)
have *h3*: *(isolate-variable-sparse p x (i+1) = 0) ⟶ (MPoly-Type.degree (f i) x) = 0*
by (*simp add*: *f-def*)
have *degh1*: *(MPoly-Type.degree (((Const ((i::nat)+1))::real mpoly)*(Var x)^i) x) =*
((MPoly-Type.degree ((Const (i+1))::real mpoly) x) + (MPoly-Type.degree ((Var x)^i:: real mpoly) x))
using *h1c2 h1c1var degree-mult*[**where** *p*=*((Const ((i::nat)+1))::real mpoly)*, **where** *q*=*((Var x)^i:: real mpoly)*]
by *blast*
then have *degh2*: *(MPoly-Type.degree (((Const ((i::nat)+1))::real mpoly)*(Var x)^i) x) = i*
using *degree-var-i degree-const*
by *simp*
have *nonzerohyp*: *(((Const ((i::nat)+1))::real mpoly)*(Var x)^i) ≠ 0*
proof (*induct i*)
case *0*
then show *?case*
by (*simp add*: *nonzero-const-is-nonzero*)
next
case (*Suc i*)
then show *?case* **using** *degree-eq degh2*
by (*metis Suc-eq-plus1 h1c1 mult-eq-0-iff nat.simps*(*3*) *nonzero-const-is-nonzero of-nat-eq-0-iff*)
qed

have *h4a1*: (*isolate-variable-sparse p x (i+1)* ≠ *0*) ⟶ (*MPoly-Type.degree* (*isolate-variable-sparse p x (i+1)* ∗ ((*Var x*)^(*i*) ∗ (*Const (i+1)*)))::*real mpoly*) *x* =
(*MPoly-Type.degree* (*isolate-variable-sparse p x (i+1)*:: *real mpoly*) *x*) + (*MPoly-Type.degree* (((*Const (i+1)*) ∗ (*Var x*)^(*i*))::*real mpoly*) *x*))
using *nonzerohyp degree-mult*[**where** *p* = (*isolate-variable-sparse p x (i+1)*)::*real mpoly*, **where** *q* = ((*Const (i+1)*) ∗ (*Var x*)^(*i*)):: *real mpoly*, **where** *v* = *x*]
by (*simp add*: *mult.commute*)
have *h4*: (*isolate-variable-sparse p x (i+1)* ≠ *0*) ⟶ (*MPoly-Type.degree* (*f i*) *x*) = *i*
using *h4a1 f-def degh2 h1a*
by (*metis* (*no-types*, *lifting*) *degh1 degree-const h1b mult.commute mult.left-commute of-nat-1 of-nat-add*)
have *h5*: (*MPoly-Type.degree* (*f i*) *x*) ≤ *i* **using** *h3 h4* **by** *auto*
have *h6*: (*MPoly-Type.degree* (*f n*) *x*) = *n* **using** *h1c3 h4*
by (*smt One-nat-def add.right-neutral add-Suc-right degree-const degree-mult divisors-zero f-def h1a h1b h1c1 mult.commute nonzero-const-is-nonzero of-nat-1 of-nat-add of-nat-neq-0*)
have *h7a*: *derivative x p* = (∑ *i*∈{*0*..*MPoly-Type.degree p x*−*1*}. *isolate-variable-sparse p x (i+1)* ∗ (*Var x*)^*i* ∗ (*Const (i+1)*)) **using** *derivative-def* **by** *auto*
have *h7b*: (∑ *i*∈{*0*..*MPoly-Type.degree p x*−*1*}. *isolate-variable-sparse p x (i+1)* ∗ (*Var x*)^*i* ∗ (*Const (i+1)*)) = (∑ *i*∈{*0*..*MPoly-Type.degree p x*−*1*}. (*f i*)) **using** *f-def*
by (*metis Suc-eq-plus1 add.commute semiring-1-class.of-nat-simps*(*2*))
have *h7c*: *derivative x p* = (∑ *i*∈{*0*..*MPoly-Type.degree p x*−*1*}. (*f i*)) **using** *h7a h7b* **by** *auto*
have *h7*: *derivative x p* = (∑ *i*∈{*0*..*n*}. (*f i*)) **using** *n-def h7c*
by *blast*
have *h8*: *n* > *0* ⟶ ((*MPoly-Type.degree* (∑ *i*∈{*0*..(*n*−*1*)}. (*f i*)) *x*) < *n*)
proof (*induct n*)
case *0*
then show *?case*
by *blast*
next
case (*Suc n*)
then show *?case* **using** *h5 degree-less-sum*
by (*smt add-cancel-right-right atLeastAtMost-iff degree-const degree-mult degree-sum-less degree-var-i diff-Suc-1 f-def h1a le-imp-less-Suc mult.commute mult-eq-0-iff*)
qed
have *h9a*: *n* = *0* ⟶ *MPoly-Type.degree* (∑ *i*∈{*0*..*n*}. (*f i*)) *x* = *n* **using** *h6*
by *auto*
have *h9b*: *n* > *0* ⟶ *MPoly-Type.degree* (∑ *i*∈{*0*..*n*}. (*f i*)) *x* = *n*
proof −
have *h9bhyp*: *n* > *0* ⟶ (*MPoly-Type.degree* (∑ *i*∈{*0*..*n*}. (*f i*)) *x* = *MPoly-Type.degree* ((∑ *i*∈{*0*..(*n*−*1*)}. (*f i*)) + (*f n*)) *x*)
by (*metis Suc-diff-1 sum.atLeast0-atMost-Suc*)
have *h9bhyp2*: *n* > *0* ⟶ ((*MPoly-Type.degree* ((∑ *i*∈{*0*..(*n*−*1*)}. (*f i*)) + (*f n*)) *x*) = *n*)
using *h6 h8 degree-less-sum*

```
    by (simp add: add.commute)
  then show ?thesis using h9bhyp2 h9bhyp
    by linarith
 qed
 have h9: MPoly-Type.degree (∑ i∈{0..n}. (f i)) x = n using h9a h9b
  by blast
 have h10: MPoly-Type.degree (derivative x p) x = n using h9 h7
  by simp
 show ?thesis using h10 n-def
  using assms by linarith
qed


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−
 have h1a: MPoly-Type.degree p var = 1 ⟶ p =
  isolate-variable-sparse p var 0 +
  isolate-variable-sparse p var 1 * Var var
  using sum-over-zero[where mp=p,where x=var]
  by auto
 have h1b: MPoly-Type.degree p var = 1 ⟶ isolate-variable-sparse p var 2 = 0
  using isovar-greater-degree
  by (simp add: isovar-greater-degree)
 have h1: MPoly-Type.degree p var = 1 ⟶ p =
  isolate-variable-sparse p var 0 +
  isolate-variable-sparse p var 1 * Var var
  + isolate-variable-sparse p var 2 * (Var var)^2 using h1a h1b by auto
 have h2a: MPoly-Type.degree p var = 2 ⟶ p = (∑ i::nat ≤ 2. isolate-variable-sparse
p var i * Var var^i)
  using sum-over-zero[where mp=p, where x=var] by auto
 have h2b: (∑ i::nat ≤ 2. isolate-variable-sparse p var i * Var var^i) =
  (∑ i::nat ≤ 1. isolate-variable-sparse p var i * Var var^i) +
  isolate-variable-sparse p var 2 * (Var var)^2 apply auto
  by (simp add: numerals(2))
 have h2: MPoly-Type.degree p var = 2 ⟶ p =
  isolate-variable-sparse p var 0 +
  isolate-variable-sparse p var 1 * Var var +
  isolate-variable-sparse p var 2 * (Var var)^2
  using h2a h2b by auto
 have h3: isolate-variable-sparse p var 0 +
  isolate-variable-sparse p var 1 * Var var +
  isolate-variable-sparse p var 2 * (Var var)^2 =
  isolate-variable-sparse p var 2 * (Var var)^2 +
```

isolate-variable-sparse p var 1 ∗ Var var +
isolate-variable-sparse p var 0 **by** (*simp add*: *add.commute*)

show *?thesis* **using** *h h1 h2 h3* **by** *presburger*
qed

lemma *degree-isovarspar* : *MPoly-Type.degree* (*isolate-variable-sparse* (*p*::*real mpoly*) *x i*) *x* = *0*
using *not-in-isovarspar varNotIn-degree* **by** *blast*

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* ⇒ *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* ⇒ *real list* ⇒ *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* ≤ *0*) |
aEval (*Neq p*) *L* = (*insertion* (*nth-default 0 L*) *p* ≠ *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* ⟷ (¬ *aEval* (*aNeg a*) *L*)
apply(*cases a*)
apply(*auto*)
apply(*smt insertionNegative*)

apply(*smt insertionNegative*)
apply(*smt insertionNegative*)
apply(*smt insertionNegative*)
done

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* ⇒ *real list* ⇒ *bool* **where**
eval (*Atom a*) Γ = *aEval a* Γ |
eval (*TrueF*) - = *True* |
eval (*FalseF*) - = *False* |
eval (*And* φ ψ) Γ = ((*eval* φ Γ) ∧ (*eval* ψ Γ)) |
eval (*Or* φ ψ) Γ = ((*eval* φ Γ) ∨ (*eval* ψ Γ)) |
eval (*Neg* φ) Γ = (¬ (*eval* φ Γ)) |
eval (*ExQ* φ) Γ = (∃ *x*. (*eval* φ (*x*#Γ))) |
eval (*AllQ* φ) Γ = (∀ *x*. (*eval* φ (*x*#Γ)))|
eval (*AllN i* φ) Γ = (∀ *l*. *length l* = *i* ⟶ (*eval* φ (*l* @ Γ)))|
eval (*ExN i* φ) Γ = (∃ *l*. *length l* = *i* ∧ (*eval* φ (*l* @ Γ)))

lemma *eval* (*ExQ* (*Or* (*Atom A*) (*Atom B*))) *xs* = *eval* (*Or* (*ExQ*(*Atom A*)) (*ExQ*(*Atom B*))) *xs*
by(*auto*)

lemma *eval-neg-neg* : *eval* (*neg* (*neg f*)) *L* ⟷ *eval f L*
by (*simp add*: *neg-def*)

lemma *eval-neg* : (¬ *eval* (*neg f*) *L*) ⟷ *eval f L*
by (*simp add*: *neg-def*)

lemma *eval-and* : *eval* (*and a b*) *L* ⟷ (*eval a L* ∧ *eval b L*)
by (*simp add*: *and-def*)

lemma *eval-or* : *eval* (*or a b*) *L* ⟷ (*eval a L* ∨ *eval b L*)
by (*simp add*: *or-def*)

lemma *eval-Or* : *eval* (*Or a b*) *L* ⟷ (*eval a L* ∨ *eval b L*)
by (*simp*)

lemma *eval-And* : *eval* (*And a b*) *L* ⟷ (*eval a L* ∧ *eval b L*)
by (*simp*)

lemma *eval-not* : *eval* (*neg a*) *L* ⟷ ¬(*eval a L*)
by (*simp add*: *neg-def*)

lemma *eval-true* : *eval TrueF L*
by *simp*

lemma *eval-false* : $\neg(eval\ FalseF\ L)$
by *simp*

lemma *eval-Neg* : $eval\ (Neg\ \varphi)\ L = eval\ (neg\ \varphi)\ L$
by (*simp add*: *eval-not*)

lemma *eval-Neg-Neg* : $eval\ (Neg\ (Neg\ \varphi))\ L = eval\ \varphi\ L$
by (*simp add*: *eval-not*)

lemma *eval-Neg-And* : $eval\ (Neg\ (And\ \varphi\ \psi))\ L = eval\ (Or\ (Neg\ \varphi)\ (Neg\ \psi))\ L$
by *simp*

lemma *aEval-leq* : $aEval\ (Leq\ p)\ L = (aEval\ (Less\ p)\ L \vee aEval\ (Eq\ p)\ L)$
by *auto*

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

fun *freeIn* :: $nat \Rightarrow atom\ fm \Rightarrow bool$ **where**
$freeIn\ var\ (Atom(Eq\ p)) = (var \notin (vars\ p))|$
$freeIn\ var\ (Atom(Less\ p)) = (var \notin (vars\ p))|$
$freeIn\ var\ (Atom(Leq\ p)) = (var \notin (vars\ p))|$
$freeIn\ var\ (Atom(Neq\ p)) = (var \notin (vars\ p))|$
$freeIn\ var\ (TrueF) = True|$
$freeIn\ var\ (FalseF) = True|$
$freeIn\ var\ (And\ a\ b) = ((freeIn\ var\ a) \wedge (freeIn\ var\ b))|$
$freeIn\ var\ (Or\ a\ b) = ((freeIn\ var\ a) \wedge (freeIn\ var\ b))|$
$freeIn\ var\ (Neg\ a) = freeIn\ var\ a|$
$freeIn\ var\ (ExQ\ a) = freeIn\ (var{+}1)\ a|$
$freeIn\ var\ (AllQ\ a) = freeIn\ (var{+}1)\ a|$
$freeIn\ var\ (AllN\ i\ a) = freeIn\ (var{+}i)\ a|$
$freeIn\ var\ (ExN\ i\ a) = freeIn\ (var{+}i)\ a$

fun *liftmap* :: $(nat \Rightarrow atom \Rightarrow atom\ fm) \Rightarrow atom\ fm \Rightarrow nat \Rightarrow atom\ fm$ **where**
$liftmap\ f\ TrueF\ var = TrueF|$
$liftmap\ f\ FalseF\ var = FalseF|$
$liftmap\ f\ (Atom\ a)\ var = f\ var\ a|$
$liftmap\ f\ (And\ \varphi\ \psi)\ var = And\ (liftmap\ f\ \varphi\ var)\ (liftmap\ f\ \psi\ var)|$
$liftmap\ f\ (Or\ \varphi\ \psi)\ var = Or\ (liftmap\ f\ \varphi\ var)\ (liftmap\ f\ \psi\ var)|$
$liftmap\ f\ (Neg\ \varphi)\ var = Neg\ (liftmap\ f\ \varphi\ var)|$
$liftmap\ f\ (ExQ\ \varphi)\ var = ExQ\ (liftmap\ f\ \varphi\ (var{+}1))|$
$liftmap\ f\ (AllQ\ \varphi)\ var = AllQ\ (liftmap\ f\ \varphi\ (var{+}1))|$
$liftmap\ f\ (AllN\ i\ \varphi)\ var = AllN\ i\ (liftmap\ f\ \varphi\ (var{+}i))|$
$liftmap\ f\ (ExN\ i\ \varphi)\ var = ExN\ i\ (liftmap\ f\ \varphi\ (var{+}i))$

```
fun depth :: 'a fm ⇒ natwhere
  depth TrueF = 1|
  depth FalseF = 1|
  depth (Atom -) = 1|
  depth (And φ ψ) = max (depth φ) (depth ψ) + 1|
  depth (Or φ ψ) = max (depth φ) (depth ψ) + 1|
  depth (Neg φ) = depth φ + 1|
  depth (ExQ φ) = depth φ + 1|
  depth (AllQ φ) = depth φ + 1|
  depth (AllN i φ) = depth φ + 1|
  depth (ExN i φ) = depth φ + 1

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

fun negation-free :: atom fm ⇒ bool where
  negation-free TrueF = True |
  negation-free FalseF = True  |
  negation-free (Atom a) = True |
  negation-free (And φ1 φ2) = ((negation-free φ1) ∧ (negation-free φ2)) |
  negation-free (Or φ1 φ2) = ((negation-free φ1) ∧ (negation-free φ2)) |
  negation-free (ExQ φ) = negation-free φ |
  negation-free (AllQ 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 ⟶ eval (Atom(Eq q)) L ⟶ eval (Atom(Eq(p
+ q))) L
  by (simp add: insertion-add)

lemma freeIn-list-conj : (∀f∈set(F). freeIn var f) ⟹ freeIn var (list-conj F)
proof(induct F)
  case Nil
  then show ?case by(simp add: list-conj-def)
next
  case (Cons a F)
  then show ?case by (simp add: PolyAtoms.and-def list-conj-def)
qed

lemma freeIn-list-disj :
  assumes ∀f∈set (L::atom fm list). freeIn var f
  shows freeIn var (list-disj L)
  using assms
proof(induction L)
```

case *Nil*
then show *?case* **unfolding** *list-disj-def* **by** *auto*
next
case (*Cons a L*)
then show *?case* **unfolding** *list-disj-def or-def* **by** *simp*
qed

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(*induction* φ)
case (*Less p*)
then show *?case*
apply(*auto*)
using *not-contains-insertion*
by *metis*
next
case (*Eq p*)
then show *?case*
apply(*auto*)
using *not-contains-insertion*
by *blast*
next
case (*Leq p*)
then show *?case*
apply(*auto*)
using *not-contains-insertion*
by *metis*
next
case (*Neq p*)
then show *?case*
apply(*auto*)
using *not-contains-insertion*
by *metis*
qed

lemma *var-not-in-aEval2* : *freeIn 0* (*Atom* φ) $\Longrightarrow$ ($\exists\, x.$ *aEval* φ (x#*L*)) = ($\forall\, x.$ *aEval* φ (x#*L*))
by (*metis list-update-code*(*2*) *var-not-in-aEval*)

lemma *plugInLinear* :
assumes *lLength* : *length L*>*var*
assumes *nonzero* : *B*≠*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*))
by(*simp add*: *lLength insertion-add insertion-mult nonzero hb hc insertion-var*)

4.4 Some eval results

lemma *doubleExist* : *eval* (*ExN 2 A*) *L* = *eval* (*ExQ* (*ExQ A*)) *L*
apply(*simp*)
proof(*safe*)
fix *l*
assume *h* : *length l* = *2 eval A* (*l* @ *L*)
show ∃ *x xa*. *eval A* (*xa* # *x* # *L*)
proof(*cases l*)
case *Nil*
then show *?thesis* **using** *h* **by** *auto*
next
case (*Cons a list*)
then have *Cons'* : *l* = *a* # *list* **by** *auto*
then show *?thesis* **proof**(*cases list*)
case *Nil*
then show *?thesis* **using** *h Cons* **by** *auto*
next
case (*Cons b list*)
show *?thesis*
apply(*rule exI*[**where** *x*=*b*])**apply**(*rule exI*[**where** *x*=*a*])
using *h Cons' Cons* **by** *auto*
qed
qed
next
fix *x xa*
assume *h* : *eval A* (*xa* # *x* # *L*)
show ∃ *l*. *length l* = *2* ∧ *eval A* (*l* @ *L*)
apply(*rule exI*[**where** *x*=[*xa*,*x*]]) **using** *h* **by** *simp*
qed

lemma *doubleForall* : *eval* (*AllN 2 A*) *L* = *eval* (*AllQ* (*AllQ A*)) *L*
apply(*simp*)**using** *doubleExist eval-neg* **by** *fastforce*

lemma *unwrapExist* : *eval* (*ExN* (*j* + *1*) *A*) *L* = *eval* (*ExQ* (*ExN j A*)) *L*
apply *simp*
apply *safe*
subgoal for *l*
apply(*rule exI*[**where** *x*=*nth l j*])
apply(*rule exI*[**where** *x*=*take j l*])
apply *auto*
by (*metis Cons-nth-drop-Suc append.assoc append-Cons append-eq-append-conv-if append-take-drop-id lessI*)
subgoal for *x l*
apply(*rule exI*[**where** *x*=*l* @ [*x*]])
by *auto*
done

lemma *unwrapExist'* : *eval* (*ExN* (*j* + *1*) *A*) *L* = *eval* (*ExN j* (*ExQ A*)) *L*
apply *simp*

```
  apply safe
  subgoal for l
    apply(rule exI[where x=drop 1 l])
    apply auto
    apply(rule exI[where x=nth l 0])
    by (metis Cons-nth-drop-Suc append-Cons drop0 zero-less-Suc)
  subgoal for l x
    apply(rule exI[where x=x#l])
    by auto
  done

lemma unwrapExist'' : eval (ExN (i + j) A) L = eval (ExN i(ExN j A)) L
  apply simp
  apply safe
  subgoal for l
    apply(rule exI[where x=drop j l])
    apply auto
    apply(rule exI[where x=take j l])
    apply auto
    by (metis append.assoc append-take-drop-id)
  subgoal for l la
    apply(rule exI[where x=la@l])
    by auto
  done

lemma unwrapForall : eval (AllN (j + 1) A) L = eval (AllQ (AllN j A)) L
  using unwrapExist[of j neg A L] eval-neg by fastforce

lemma unwrapForall' : eval (AllN (j + 1) A) L =  eval (AllN j (AllQ A)) L
  using unwrapExist'[of j neg A L] eval-neg by fastforce

lemma unwrapForall'' : eval (AllN (i + j) A) L = eval (AllN i(AllN j A)) L
  using unwrapExist''[of i j neg A L] eval-neg by fastforce

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(induction φ)
  case TrueF
  then show ?case by(auto)
next
  case FalseF
  then show ?case by(auto)
next
  case (Atom x)
  then show ?case
    using var-not-in-aEval eval.simps(1) by blast
next
  case (And φ1 φ2)
  then show ?case by (meson eval.simps(4) freeIn.simps(7))
```

next
case (*Or* $\varphi 1$ $\varphi 2$)
then show *?case* **by** *fastforce*
next
case (*Neg* φ)
then show *?case* **by** (*meson eval.simps*(6) *freeIn.simps*(9))
next
case (*ExQ* φ)
fix *xa L var x*
have (*xa*::*real*) # $L[var := x] = (xa\#L)[var+1:=x]$
by *simp*
then show *?case* **using** *ExQ*
by (*metis Suc-eq-plus1 eval.simps*(7) *freeIn.simps*(10) *list-update-code*(3))
next
case (*AllQ* φ)
fix *xa L var x*
have (*xa*::*real*) # $L[var := x] = (xa\#L)[var+1:=x]$
by *simp*
then show *?case* **using** *AllQ*
by (*metis Suc-eq-plus1 eval.simps*(8) *freeIn.simps*(11) *list-update-code*(3))
next
case (*ExN i* φ)
{fix *xa L var x*
assume *length* (*xa*::*real list*) = *i*
have *xa* @ $L[var := x] = (xa@L)[var+i:=x]$
by (*simp add*: ‹*length xa* = *i*› *list-update-append*)
}
then show *?case* **using** *ExN*
by (*metis eval.simps*(10) *freeIn.simps*(13))
next
case (*AllN i* φ)
{fix *xa L var x*
assume *length* (*xa*::*real list*) = *i*
have *xa* @ $L[var := x] = (xa@L)[var+i:=x]$
by (*simp add*: ‹*length xa* = *i*› *list-update-append*)
}
then show *?case* **using** *AllN*
by (*metis eval.simps*(9) *freeIn.simps*(12))
qed

lemma *var-not-in-eval2* : $\forall L.$ (*freeIn 0* $\varphi \longrightarrow ((\exists x.\ eval\ \varphi\ (x\#L)) = (\forall x.\ eval\ \varphi\ (x\#L))))$
by (*metis list-update-code*(2) *var-not-in-eval*)

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

by (*metis list-update-length var-not-in-eval*)

lemma *eval-list-conj* : *eval* (*list-conj F*) *L* = (∀*f*∈*set*(*F*). *eval f L*)
proof −
{ fix *f F*
have *h* : *eval* (*foldr and F f*) *L* = (*eval f L* ∧ (∀*f* ∈ *set F*. *eval f L*))
apply(*induct F*)
apply *simp*
using *eval-and* **by** *auto*
} then show *?thesis*
by(*simp add:list-conj-def*)
qed

lemma *eval-list-disj* : *eval* (*list-disj F*) *L* = (∃*f*∈*set*(*F*). *eval f L*)
proof −
{ fix *f F*
have *h* : *eval* (*foldr or F f*) *L* = (*eval f L* ∨ (∃*f* ∈ *set F*. *eval f L*))
apply(*induct F*)
apply *simp*
using *eval-or* **by** *auto*
} then show *?thesis*
by(*simp add:list-disj-def*)
qed
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* ≤ *b then x else x* − *i*::*nat*)

lemma *inc-above-dec-above*: *x* < *b* ∨ *b* + *i* ≤ *x* ⟹ *inc-above b i* (*dec-above b i x*) = *x*
by (*auto simp*: *inc-above-def dec-above-def*)
lemma *dec-above-inc-above*: *dec-above b i* (*inc-above b i x*) = *x*
by (*auto simp*: *inc-above-def dec-above-def*)

lemma *inc-above-dec-above-iff*: *inc-above b i* (*dec-above b i x*) = *x* ⟷ *x* < *b* ∨ *b* + *i* ≤ *x*
by (*auto simp*: *inc-above-def dec-above-def*)

lemma *inj-on-dec-above*: *inj-on* (*dec-above b i*) {*x*. *x* < *b* ∨ *b* + *i* ≤ *x*}
by (*rule inj-on-inverseI*[**where** *g* = *inc-above b i*]) (*auto simp*: *inc-above-dec-above*)

lemma *finite-inc-above-ne*: *finite* {*x*. *f x* ≠ *c*} ⟹ *finite* {*x*. *f* (*inc-above b i x*) ≠ *c*}
proof −
fix *b* **and** *f*::*nat*⇒′*a*
assume *f*: *finite* {*x*. *f x* ≠ *c*}
moreover
have *finite* {*x*. *f* (*x* + *i*) ≠ *c*}
proof −
have {*x*. *f* (*x* + *i*) ≠ *c*} = (+) *i* −‘ {*x*. *f x* ≠ *c*}
by (*auto simp*: *ac-simps*)
also have *finite* . . .
by (*rule finite-vimageI*) (*use f* **in** *auto*)
finally show *?thesis* **.**
qed
ultimately have *finite* ({*x*. *f x* ≠ *c*} ∪ {*x*. *f* (*x* + *i*) ≠ *c*})
by *auto*
from - *this* **show** *finite* {*x*. *f* (*inc-above b i x*) ≠ *c*}
by (*rule finite-subset*) (*auto simp*: *inc-above-def*)
qed

lemma *finite-dec-above-ne*: *finite* {*x*. *f x* ≠ *c*} ⟹ *finite* {*x*. *f* (*dec-above b i x*) ≠ *c*}
proof −
fix *b* **and** *f*::*nat*⇒′*a*
assume *f*: *finite* {*x*. *f x* ≠ *c*}
moreover
have *finite* {*x*. *f* (*x* − *i*) ≠ *c*}
proof −
have {*x*. *f* (*x* − *i*) ≠ *c*} ⊆ {*0*..*i*} ∪ ((λ*x*. *x* − *i*) −‘ {*x*. *f x* ≠ *c*} ∩ {*i*<..})
by *auto*
also have *finite* . . .
apply (*rule finite-UnI*[*OF finite-atLeastAtMost*])
by (*rule finite-vimage-IntI*) (*use f* **in** ‹*auto simp*: *inj-on-def*›)
finally (*finite-subset*) **show** *?thesis* **.**
qed
ultimately have *finite* ({*x*. *f x* ≠ *c*} ∪ {*x*. *f* (*x* − *i*) ≠ *c*} ∪ {*b*})
by *auto*
from - *this* **show** *finite* {*x*. *f* (*dec-above b i x*) ≠ *c*}
by (*rule finite-subset*) (*auto simp*: *dec-above-def*)
qed

lift-definition *lowerPowers*::*nat* ⇒ *nat* ⇒ (*nat* ⇒$_0$ *'a*) ⇒ (*nat* ⇒$_0$ *'a*::*zero*)
is λ*b i p x*. *if x* ∈ {*b*..<*b*+*i*} *then 0 else p* (*dec-above b i x*)::*'a*
proof −
fix *b i*::*nat* **and** *p*::*nat* ⇒ *'a*
assume *finite* {*x*. *p x* ≠ *0*}
then have *finite* {*x*. *p* (*dec-above b i x*) ≠ *0*}
by (*rule finite-dec-above-ne*)
from - *this* **show** *finite* {*x*. (*if x* ∈ {*b*..<*b*+*i*} *then 0 else p* (*dec-above b i x*)) ≠ *0*}
by (*rule finite-subset*) *auto*
qed

lift-definition *higherPowers*::*nat* ⇒ *nat* ⇒ (*nat* ⇒$_0$ *'a*) ⇒ (*nat* ⇒$_0$ *'a*::*zero*)
is λ*b i p x*. *p* (*inc-above b i x*)::*'a*
by (*simp-all add*: *finite-inc-above-ne*)

lemma *higherPowers-lowerPowers*: *higherPowers n i* (*lowerPowers n i x*) = *x*
by *transfer* (*force simp*: *dec-above-def inc-above-def antisym-conv2*)

lemma *inj-lowerPowers*: *inj* (*lowerPowers b i*)
using *higherPowers-lowerPowers*
by (*rule inj-on-inverseI*)

lemma *lowerPowers-higherPowers*:
(⋀*j*. *n* ≤ *j* ⟹ *j* < *n* + *i* ⟹ *lookup x j* = *0*) ⟹ *lowerPowers n i* (*higherPowers n i x*) = *x*
by (*transfer fixing*: *n i*) (*force simp*: *inc-above-dec-above*)

lemma *inj-on-higherPowers*: *inj-on* (*higherPowers n i*) {*x*. ∀*j*. *n* ≤ *j* ∧ *j* < *n* + *i* ⟶ *lookup x j* = *0*}
using *lowerPowers-higherPowers*
by (*rule inj-on-inverseI*) *auto*

lemma *higherPowers-eq*: *lookup* (*higherPowers b i p*) *x* = *lookup p* (*inc-above b i x*)
by (*simp-all add*: *higherPowers.rep-eq*)

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*))
by (*auto simp add*: *lowerPowers.rep-eq*)

lemma *keys-higherPowers*: *keys* (*higherPowers b i m*) = *dec-above b i* ‘ (*keys m* ∩ {*x*. *x* ∉ {*b*..<*b*+*i*}})
apply *safe*
subgoal for *x*
apply (*rule image-eqI*[**where** *x*=*inc-above b i x*])
apply (*auto simp*: *dec-above-inc-above in-keys-iff higherPowers.rep-eq*)
by (*metis add-less-cancel-right inc-above-def leD*)
subgoal for *x*

by (*auto simp*: *inc-above-dec-above in-keys-iff higherPowers.rep-eq*)
done

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 −
fix *b i*::*nat* **and** *p*::*nat* ⇒ 'a *option*
assume *finite* (*dom p*)
then have *finite* {*x*. *p x* ≠ *None*} **by** (*simp add*: *dom-def*)

have *dom* (λ*x*. *p* (*dec-above b i x*)) = {*x*. *p* (*dec-above b i x*) ≠ *None*}
by *auto*
also have *finite* ...
by (*rule finite-dec-above-ne*) *fact*
finally
have *finite* (*dom* (λ*x*. *p* (*dec-above b i x*))) **.**
from - *this*
show *finite* (*dom* (λ*x*. *if x* ∈ {*b*..<*b*+*i*} *then None else p* (*dec-above b i x*)))
by (*rule finite-subset*) (*auto split*: *if-splits*)
qed

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 −
fix *b i*::*nat* **and** *f*::*nat* ⇒ 'a *option*
assume *finite* (*dom f*)
then have *finite* {*i*. *f i* ≠ *None*} **by** (*simp add*: *dom-def*)

have *dom* (λ*x*. *f* (*inc-above b i x*)) = {*x*. *f* (*inc-above b i x*) ≠ *None*}
by *auto*
also have *finite* ...
by (*rule finite-inc-above-ne*) *fact*
finally show *finite* (*dom* (λ*x*. *f* (*inc-above b i x*)))
.
qed

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*
using *that*
proof (*induction xs*)
case *Nil*
then show *?case* **by** *simp*
next
case (*Cons a xs*)
from *Cons*

```
  have IH: map-of (map (λa. (f (fst a), snd a)) xs) x = map-of xs (g x)
    by (auto simp: split-beta')
  have inv-into: g (f (fst a)) = fst a
    by (rule Cons) simp
  show ?case
    using Cons
    by (auto simp add: split-beta' inv-into IH)
qed

lemma map-of-filter-key-in: P x ⟹ map-of (filter (λ(k, v). P k) xs) x = map-of
xs x
  by (induction xs) auto

lemma map-of-eq-NoneI: x∉fst`set xs ⟹ map-of xs x = None
  by (induction xs) auto

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 −
  have ∗: map-of (map (λ(k, y). (if k < b then k else k − i, y)) (filter (λ(k, v). b
≤ k ⟶ ¬ k < b + i) xs)) x =
    map-of (filter (λ(k, v). b ≤ k ⟶ ¬ k < b + i) xs) (if x < b then x else x + i)
    for x
    by (rule map-of-map-key-inverse-fun-eq[where g=λk. if k < b then k else k +
i
          and f = λk. if k < b then k else k − i]) auto
  show ?thesis
    by (auto
        simp add: ∗ higherPowers_f.rep-eq lowerPowers_f.rep-eq fmlookup-of-list fm-
lookup-default-def
        inc-above-def
        map-of-filter-key-in
        split: option.splits
        intro!: fmap-ext)
qed

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)
  apply (auto
      simp add: lowerPowers_f.rep-eq fmlookup-of-list fmlookup-default-def
      dec-above-def
      map-of-filter-key-in
      split: option.splits
      intro!: fmap-ext)
  subgoal by (rule map-of-eq-NoneI[symmetric]) (auto split: if-splits)
  subgoal by (subst map-of-map-key-inverse-fun-eq[where g=λk. if k < b then k
else k − i]) auto
  subgoal by (subst map-of-map-key-inverse-fun-eq[where g=λk. if k < b then k
```

else k − i]) *auto*
subgoal by (*rule map-of-eq-NoneI*[*symmetric*]) (*auto split*: *if-splits*)
subgoal by (*subst map-of-map-key-inverse-fun-eq*[**where** *g*=λ*k*. *if k < b then k else k − i*]) *auto*
done

lemma *compute-higherPowers*[*code*]: *higherPowers n i* (*Pm-fmap xs*) = *Pm-fmap* (*higherPowers*$_f$ *n i xs*)
by (*rule poly-mapping-eqI*)
(*auto simp*: *higherPowers*$_f$*.rep-eq higherPowers.rep-eq fmlookup-default-def dec-above-def split*: *option.splits*)

lemma *compute-lowerPowers*[*code*]: *lowerPowers n i* (*Pm-fmap xs*) = *Pm-fmap* (*lowerPowers*$_f$ *n i xs*)
by (*rule poly-mapping-eqI*)
(*auto simp*: *lowerPowers*$_f$*.rep-eq lowerPowers.rep-eq fmlookup-default-def dec-above-def split*: *option.splits*)

lemma *finite-nonzero-coeff*: *finite* {*x*. *MPoly-Type.coeff mpoly x* ≠ *0*}
by *transfer auto*

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 −
fix *b i* **and** *mp*::(*nat* ⇒$_0$ *nat*) ⇒ *'a*
assume *finite* {*x*. *mp x* ≠ *0*}
have {*x*. *mp* (*lowerPowers b i x*) ≠ *0*} = (*lowerPowers b i* −‘ {*x*. *mp x* ≠ *0*})
(**is** *?set* = *?vimage*)
by *auto*
also
from *finite-vimageI*[*OF* ‹*finite* -› *inj-lowerPowers*]
have *finite ?vimage* **.**
finally show *finite ?set* **.**
qed

lemma *higherPowers-zero*[*simp*]: *higherPowers b i 0* = *0*
by *transfer auto*

lemma *keys-lowerPoly*$_0$: *keys* (*lowerPoly*$_0$ *b i mp*) = *higherPowers b i* ‘ (*keys mp* ∩ {*x*. ∀*j*∈{*b*..<*b*+*i*}. *lookup x j* = *0*})
apply (*auto*)
subgoal for *x*
apply (*rule image-eqI*[**where** *x*=*lowerPowers b i x*])
apply (*auto simp*: *higherPowers-lowerPowers in-keys-iff lowerPoly*$_0$*.rep-eq lowerPowers.rep-eq*)
done
subgoal for *x*
apply (*auto simp*: *in-keys-iff lowerPoly*$_0$*.rep-eq*)

```
    apply (subst (asm) lowerPowers-higherPowers)
     apply auto
    done
  done


lift-definition higherPoly₀::nat ⇒ nat ⇒ ((nat⇒₀nat)⇒₀'a::zero) ⇒ ((nat⇒₀nat)⇒₀
'a) is
  λb i (mp::(nat⇒₀nat)⇒'a) mon.
    if (∃j∈{b..<b+i}. lookup mon j > 0)
    then 0
    else mp (higherPowers b i mon)
proof −
  fix b i and mp::(nat ⇒₀ nat) ⇒ 'a
  assume finite {x. mp x ≠ 0}
  have {x. (if ∃j∈{b..<b + i}. 0 < lookup x j then 0 else mp (higherPowers b i
x)) ≠ 0} ⊆
    insert 0 (higherPowers b i −` {x. mp x ≠ 0} ∩ {x. ∀j∈{b..<b+i}. lookup x j
= 0})
    (is ?set ⊆ ?vimage)
    by auto
  also
  from finite-vimage-IntI[OF ‹finite -› inj-on-higherPowers, of b i]
  have finite ?vimage by (auto simp: Ball-def)
  finally (finite-subset) show finite ?set .
qed


context includes fmap.lifting begin

lift-definition lowerPoly_f::nat ⇒ nat ⇒ ((nat⇒₀nat), 'a::zero)fmap ⇒ ((nat⇒₀nat),
'a)fmap is
  λb i (mp::((nat⇒₀nat)⇀'a)) mon::(nat⇒₀nat). mp (lowerPowers b i mon)
proof −− TODO: this is exactly the same proof as the one for lowerPoly₀
  fix b i and mp::(nat ⇒₀ nat) ⇒ 'a option
  assume finite (dom mp)
  also have dom mp = {x. mp x ≠ None} by auto
  finally have finite {x. mp x ≠ None} .
  have (dom (λmon. mp (lowerPowers b i mon))) = {mon. mp (lowerPowers b i
mon) ≠ None}
    (is ?set = -)
    by (auto split: if-splits)
  also have ... = lowerPowers b i −` {x. mp x ≠ None} (is - = ?vimage)
    by auto
  also
  from finite-vimageI[OF ‹finite {x. mp x ≠ None}› inj-lowerPowers]
  have finite ?vimage .
  finally show finite ?set .
qed
```

lift-definition $higherPoly_f$::$nat \Rightarrow nat \Rightarrow ((nat \Rightarrow_0 nat), 'a::zero)fmap \Rightarrow ((nat \Rightarrow_0 nat), 'a)fmap$ **is**
$\lambda b\ i\ (mp::((nat \Rightarrow_0 nat) \rightharpoonup 'a))\ mon::(nat \Rightarrow_0 nat).$
if $(\exists j \in \{b..<b+i\}.\ lookup\ mon\ j > 0)$
then None
else mp $(higherPowers\ b\ i\ mon)$
proof −
fix $b\ i$ **and** $mp::(nat \Rightarrow_0 nat) \rightharpoonup 'a$
assume *finite* $(dom\ mp)$
have $dom\ (\lambda x.\ (if\ \exists j \in \{b..<b+i\}.\ 0 < lookup\ x\ j\ then\ None\ else\ mp\ (higherPowers\ b\ i\ x))) \subseteq$
$insert\ 0\ (higherPowers\ b\ i - `(dom\ mp) \cap \{x.\ \forall j \in \{b..<b+i\}.\ lookup\ x\ j = 0\})$
(**is** *?set* $\subseteq$ *?vimage*)
by (*auto split*: *if-splits*)
also
from *finite-vimage-IntI*[*OF* ‹*finite* -› *inj-on-higherPowers*, *of b i*]
have *finite ?vimage* **by** (*auto simp*: *Ball-def*)
finally (*finite-subset*) **show** *finite ?set* **.**
qed

lemma *keys-lowerPowers*: $keys\ (lowerPowers\ b\ i\ m) = inc\text{-}above\ b\ i\ `(keys\ m)$
apply *safe*
subgoal for *x*
apply (*rule image-eqI*[**where** *x*=*dec-above b i x*])
apply (*auto simp*: *inc-above-dec-above in-keys-iff lowerPowers.rep-eq*)
apply (*metis inc-above-dec-above not-less*)
by *meson*
by (*metis higherPowers.rep-eq higherPowers-lowerPowers in-keys-iff*)

lemma *keys-higherPoly*$_0$: $keys\ (higherPoly_0\ b\ i\ mp) = lowerPowers\ b\ i\ `(keys\ mp)$
apply (*auto*)
subgoal for *x*
apply (*rule image-eqI*[**where** *x*=*higherPowers b i x*])
apply (*auto simp*: *lowerPowers-higherPowers in-keys-iff higherPoly*$_0$*.rep-eq higherPowers.rep-eq*)
apply (*metis atLeastLessThan-iff lowerPowers-higherPowers neq0-conv*)
by *meson*
subgoal for *x*
apply (*auto simp*: *in-keys-iff higherPoly*$_0$*.rep-eq*)
apply (*simp add*: *lowerPowers-eq*)
by (*simp add*: *higherPowers-lowerPowers*)
done

end

lemma *inc-above-id*[*simp*]: $n < m \Longrightarrow inc\text{-}above\ m\ i\ n = n$ **by** (*auto simp*:

inc-above-def)
lemma *inc-above-Suc*[*simp*]: $n \geq m \Longrightarrow$ *inc-above m i n* = *n* + *i* **by** (*auto simp*: *inc-above-def*)

lemma *compute-lowerPoly*$_0$[*code*]: *lowerPoly*$_0$ *n i* (*Pm-fmap m*) = *Pm-fmap* (*lowerPoly*$_f$ *n i m*)
by (*auto simp*: *lowerPoly*$_0$*.rep-eq fmlookup-default-def lowerPoly*$_f$*.rep-eq*
split: *option.splits*
intro!: *poly-mapping-eqI*)

lemma *compute-higherPoly*$_0$[*code*]: *higherPoly*$_0$ *n i* (*Pm-fmap m*) = *Pm-fmap* (*higherPoly*$_f$ *n i m*)
by (*auto simp*: *higherPoly*$_0$*.rep-eq fmlookup-default-def higherPoly*$_f$*.rep-eq*
split: *option.splits*
intro!: *poly-mapping-eqI*)

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*)))
apply (*rule sym*)
apply (*rule fmap-ext*)
unfolding *lowerPoly*$_f$*.rep-eq fmlookup-of-list*
apply (*subst map-of-map-key-inverse-fun-eq*[**where** *g*=*lowerPowers n i*])
subgoal
by (*auto simp add*: *lowerPowers-higherPowers*)
subgoal by (*auto simp add*: *higherPowers-lowerPowers*)
apply (*auto simp*: *fmlookup-of-list lowerPoly*$_f$*.rep-eq map-of-eq-None-iff map-of-filter-key-in*
fmdom′*-fmap-of-list higherPowers.rep-eq lowerPowers.rep-eq dec-above-def*
intro!: *fmap-ext*)
done

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*))
apply (*rule sym*)
apply (*rule fmap-ext*)
unfolding *higherPoly*$_f$*.rep-eq fmlookup-of-list*
apply *auto*
subgoal
by (*rule map-of-eq-NoneI*) *auto*
subgoal
apply (*subst map-of-filter-key-in*)
apply *auto*
apply (*subst map-of-map-key-inverse-fun-eq*[**where** *g*=*higherPowers n i*])
subgoal
by (*auto simp add*: *higherPowers-lowerPowers*)
subgoal by (*auto simp add*: *lowerPowers-higherPowers*)
apply (*auto simp*: *fmlookup-of-list lowerPoly*$_f$*.rep-eq map-of-eq-None-iff map-of-filter-key-in*
fmdom′*-fmap-of-list higherPowers.rep-eq lowerPowers.rep-eq dec-above-def*

```
      intro!: fmap-ext)
   done
 done

end

lift-definition lowerPoly::nat ⇒ nat ⇒ 'a::zero mpoly ⇒ 'a mpoly is lowerPoly₀
.
lift-definition liftPoly::nat ⇒ nat ⇒ 'a::zero mpoly ⇒ 'a mpoly is higherPoly₀ .

lemma coeff-lowerPoly: MPoly-Type.coeff (lowerPoly b i mp) x = MPoly-Type.coeff
mp (lowerPowers b i x)
  by (transfer') (simp add: lowerPoly₀.rep-eq lowerPowers.rep-eq)

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))
  by (transfer') (simp add: higherPowers.rep-eq higherPoly₀.rep-eq )

lemma monomials-lowerPoly: monomials (lowerPoly b i mp) = higherPowers b i
` (monomials mp ∩ {x. ∀j∈{b..<b + i}. lookup x j = 0})
  by transfer' (simp add: keys-lowerPoly₀)


lemma monomials-liftPoly: monomials (liftPoly b i mp) = lowerPowers b i ` (monomials
mp)
  using keys-higherPoly₀
  by (simp add: keys-higherPoly₀ liftPoly.rep-eq monomials.rep-eq)


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
  by (meson degree-eq-iff varNotIn-degree)
```

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*
using *assms dec-above-def nth-append add.commute*
by (*smt add-diff-cancel-left add-le-cancel-left add-strict-increasing append-Nil2 atLeastLessThan-iff le-add-diff-inverse length-append length-greater-0-conv less-imp-le-nat not-less nth-default-append*)

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 −
have ∗: *monomials p ∩ {x. ∀ j∈{i..<i + j}. lookup x j = 0} = monomials p*
using *assms(1)* **by** (*auto intro*: *lookup-not-in-vars-eq-zero*)
then have *monomials p ⊆ {x. ∀ k. i ≤ k ∧ k < i + j ⟶ lookup x k = 0}*
by *force*
have *?lhs = (∑ m∈monomials (lowerPoly i j p). MPoly-Type.coeff (lowerPoly i j p) m ∗ (∏ k∈keys m. (nth-default 0 (prfx @ L)) k ^ lookup m k))*
unfolding *insertion-code* **..**
also have *... = (∑ m∈monomials p. MPoly-Type.coeff p m ∗ (∏ k∈keys m. (nth-default 0 (prfx @ xs @ L) k) ^ lookup m k))*
proof (*rule sum.reindex-cong*)
note *inj-on-higherPowers*[*of i j*]
moreover note ‹*monomials p ⊆* -›
ultimately show *inj-on (higherPowers i j) (monomials p)*
by (*rule inj-on-subset*)
next
show *monomials (lowerPoly i j p) = higherPowers i j ‘ monomials p*
unfolding *monomials-lowerPoly* ∗ **..**
next
fix *m*
assume *m*: *m ∈ monomials p*
from *m* ‹*monomials p ⊆* -› **have** *keys m ⊆ {x. x ∉ {i..<i + j}}*
by *auto*
then have *lookup m k = 0* **if** *i ≤ k k < i + j* **for** *k*
using *that* **by** (*auto simp*: *in-keys-iff*)
then have *lowerPowers i j (higherPowers i j m) = m*
by (*rule lowerPowers-higherPowers*)
then have *MPoly-Type.coeff (lowerPoly i j p) (higherPowers i j m) = MPoly-Type.coeff p m*
unfolding *coeff-lowerPoly* **by** *simp*
moreover
have *(∏ k∈keys (higherPowers i j m). (nth-default 0 (prfx @ L)) k ^ lookup (higherPowers i j m) k) =*

```
    (∏ k∈keys m. (nth-default 0 (prfx @ xs @ L)) k ^ lookup m k)
  proof (rule prod.reindex-cong)
    show keys (higherPowers i j m) = dec-above i j ' keys m
      unfolding keys-higherPowers using ‹keys m ⊆ -› by auto
  next
    from inj-on-dec-above[of i j]
    show inj-on (dec-above i j) (keys m)
      by (rule inj-on-subset) (use ‹keys m ⊆ -› in auto)
  next
    fix k assume k: k ∈ keys m
    from k ‹keys m ⊆ -› have k ∉ {i..<i+j} by auto
    from k ‹keys m ⊆ -›
    have inc-above i j (dec-above i j k) = k
      by (subst inc-above-dec-above) auto
    then have lookup (higherPowers i j m) (dec-above i j k) = lookup m k
      unfolding higherPowers.rep-eq by simp
   moreover have nth-default 0 (prfx @ L) (dec-above i j k) = (nth-default 0
(prfx @ xs @ L)) k
      apply (rule nth-dec-above)
      using assms ‹k ∉ -›
      by auto
    ultimately
    show ((nth-default 0 (prfx @ L)) (dec-above i j k)) ^ lookup (higherPowers i
j m) (dec-above i j k) = ((nth-default 0 (prfx @ xs @ L)) k) ^ lookup m k
      by simp
  qed
  ultimately
   show MPoly-Type.coeff (lowerPoly i j p) (higherPowers i j m) * (∏ k∈keys
(higherPowers i j m). (nth-default 0(prfx @ L)) k ^ lookup (higherPowers i j m)
k) =
       MPoly-Type.coeff p m * (∏ k∈keys m. (nth-default 0 (prfx @ xs @ L)) k ^
lookup m k)
    by simp
 qed
 finally show ?thesis
   unfolding insertion-code .
qed

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)
 using assms nth-default-def apply simp
 by (subst insertion-lowerPoly[where xs=[x]]) auto

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)
using *insertion-lowerPoly1*[*OF assms*, *of Nil x L*]
by *simp*

lemma *aEval-lowerAtom* : (*freeIn 0* (*Atom A*)) ⟹ (*aEval A* (*x*#*L*) = *aEval* (*lowerAtom 0 1 A*) *L*)
by (*cases A*) (*simp-all add*:*insertion-lowerPoly01*)

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*)
using *assms*
proof(*induction F arbitrary* : *i init*)
case *TrueF*
then show *?case* **by** *simp*
next
case *FalseF*
then show *?case* **by** *simp*
next
case (*Atom A*)
then show *?case* **apply**(*cases A*) **by** (*simp-all add*: *insertion-lowerPoly1*)
next
case (*And F1 F2*)

```
  then show ?case by auto
next
  case (Or F1 F2)
  then show ?case by auto
next
  case (Neg F)
  then show ?case by simp
next
  case (ExQ F)
  have map: ⋀y. (map-fm-binders (λa x. lowerAtom (i + x) 1 a) F (y + 1)) =
(map-fm-binders (λa x. lowerAtom (i + 1 + x) 1 a) F y)
    apply(induction F) by(simp-all)
  show ?case apply simp apply(rule ex-cong1)
    subgoal for xa
      using map[of 0] ExQ(1)[of Suc i xa#init] ExQ(2) ExQ(3)
      by simp
    done
next
  case (AllQ F)
  have map: ⋀y. (map-fm-binders (λa x. lowerAtom (i + x) 1 a) F (y + 1)) =
(map-fm-binders (λa x. lowerAtom (i + 1 + x) 1 a) F y)
    apply(induction F) by(simp-all)
  show ?case apply simp apply(rule all-cong1)
    subgoal for xa
      using map[of 0] AllQ(1)[of Suc i xa#init] AllQ(2) AllQ(3)
      by simp
    done
next
  case (ExN x1 F)
  have map: ⋀y. (map-fm-binders (λa x. lowerAtom (i + x) 1 a) F (y + x1)) =
(map-fm-binders (λa x. lowerAtom (i + x1 + x) 1 a) F y)
    apply(induction F) apply(simp-all add:add.commute add.left-commute)
    apply (metis add-Suc)
    apply (metis add-Suc)
    apply (metis add.assoc)
    by (metis add.assoc)
  show ?case apply simp apply(rule ex-cong1)
    subgoal for l
      using map[of 0] ExN(1)[of i+x1 l@init] ExN(2) ExN(3)
      by auto
    done
next
  case (AllN x1 F)
  have map: ⋀y. (map-fm-binders (λa x. lowerAtom (i + x) 1 a) F (y + x1)) =
(map-fm-binders (λa x. lowerAtom (i + x1 + x) 1 a) F y)
    apply(induction F) apply(simp-all add:add.commute add.left-commute)
    apply (metis add-Suc)
    apply (metis add-Suc)
    apply (metis add.assoc)
```

```
    by (metis add.assoc)
  show ?case apply simp apply(rule all-cong1)
    subgoal for l
      using map[of 0] AllN(1)[of i+x1 l@init] AllN(2) AllN(3)
      by auto
    done
qed

lemma eval-lowerFm :
  assumes h : freeIn 0 F
  shows  ∀ xs. (eval (lowerFm 0 1 F) xs = eval (ExQ F) xs)
  using eval-lowerFm-helper[OF h] by simp

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 (λa. λx. 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 higherPoly0-add : higherPoly0 x y (p + q) = higherPoly0 x y p + higher-
Poly0 x y q
   using poly-mapping-eq-iff[where a = higherPoly0 x y (p + q), where b =
higherPoly0 x y p + higherPoly0 x y q]
     plus-poly-mapping.rep-eq[where x = higherPoly0 x y (p + q), where xa =
higherPoly0 x y p + higherPoly0 x y q]
  apply (auto)
  by (simp add: higherPoly0.rep-eq lookup-add poly-mapping-eqI)

lemma liftPoly-add: liftPoly w z (a + b) = (liftPoly w z a) + (liftPoly w z b)
  unfolding liftPoly-def apply (auto)
proof −
  have h1: mapping-of (a + b) = mapping-of a + mapping-of b
    by (simp add: plus-mpoly.rep-eq)
  have h2: MPoly (higherPoly0 w z (mapping-of a + mapping-of b)) =
   MPoly (higherPoly0 w z (mapping-of a)) + MPoly (higherPoly0 w z (mapping-of
b))
  proof −
    have h0a: higherPoly0 w z (mapping-of a + mapping-of b) = (higherPoly0 w z
(mapping-of a)) + (higherPoly0 w z (mapping-of b))
      using higherPoly0-add[of w z - - ] by auto
    then show ?thesis
```

by (*simp add*: *plus-mpoly.abs-eq*)
qed
show *MPoly* (*higherPoly*$_0$ *w z* (*mapping-of* (*a* + *b*))) =
MPoly (*higherPoly*$_0$ *w z* (*mapping-of a*)) +
MPoly (*higherPoly*$_0$ *w z* (*mapping-of b*)) **using** *h1 h2* **by** *auto*
qed

lemma *vars-lift-add* : *vars*(*liftPoly a b* (*p*+*q*)) ⊆ *vars*(*liftPoly a b* (*p*))∪ *vars*(*liftPoly a b* (*q*))
using *liftPoly-add*[*of a b p q*]
using *vars-add*[*of liftPoly a b p liftPoly a b q*]
by *auto*

lemma *mapping-of-lift-add* : *mapping-of* (*liftPoly x y* (*a* + *b*)) = *mapping-of* (*liftPoly x y a*) + *mapping-of* (*liftPoly x y b*)
unfolding *liftPoly.rep-eq plus-mpoly.rep-eq*
using *higherPoly*$_0$*-add* **.**

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−
have *MPoly-Type.coeff* (*liftPoly x y* (*a* + *b*)) *m* = *MPoly-Type.coeff* (*liftPoly x y a* + *liftPoly x y b*) *m*
apply(*simp add* : *mapping-of-lift-add coeff-def*) **using** *lookup-add*
by (*simp add*: *plus-mpoly.rep-eq*)
also have ... = *MPoly-Type.coeff* (*liftPoly x y a*) *m* + *MPoly-Type.coeff* (*liftPoly x y b*) *m*
using *MPolyExtension.coeff-add*[*of liftPoly x y a liftPoly x y b m*] **.**
finally show *?thesis*
by *auto*
qed

lemma *lift-add* : *insertion* (*f*::*nat*⇒*real*) (*liftPoly 0 z* (*a* + *b*)) = *insertion f* (*liftPoly 0 z a* + *liftPoly 0 z b*)
using *liftPoly-add*[*of 0 z a b*]
by *simp*

lemma *lower-power-zero* : *lowerPowers a b 0* = *0*
unfolding *lowerPowers-def dec-above-def id-def* **apply** *auto*
unfolding *Poly-Mapping.lookup-zero* **by** *auto*

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(*cases a*=*0*)
case *True*
then show *?thesis*
by (*smt MPolyExtension.monom-zero add.left-neutral add-diff-cancel-right*′ *image-empty liftPoly-add vars-monom-single-cases*)

next
case *False*
have *h1*: *vars* (*liftPoly i j* (*MPoly-Type.monom m a*)) = *keys* (*lowerPowers i j m*)
unfolding *liftPoly-def*
unfolding *keys-lowerPowers keys-higherPoly*$_0$ *vars-def* **apply** *auto*
apply (*smt imageE keys-higherPoly*$_0$ *keys-lowerPowers lookup-eq-zero-in-keys-contradict lookup-single-not-eq mapping-of-inverse monomials.abs-eq*)
by (*metis False higherPowers.rep-eq higherPowers-lowerPowers image-eqI in-keys-iff keys-higherPoly*$_0$ *lookup-single-eq mapping-of-inverse monomials.abs-eq*)
show *?thesis*
unfolding *h1 vars-monom-keys*[*OF False*]
keys-lowerPowers inc-above-def **by** *auto*
qed

lemma *lift-clear-vars* : *vars* (*liftPoly i j* (*p*::*real mpoly*)) ∩ {*i*..<*i* + *j*} = {}
proof(*induction p rule*: *mpoly-induct*)
case (*monom m a*)
then show *?case*
unfolding *lift-vars-monom* **by** *auto*
next
case (*sum p1 p2 m a*)
then show *?case*
using *vars-lift-add*[*of i j p1 p2*]
by *blast*
qed

lemma *lift0*: (*liftPoly i j 0*) = *0*
by (*simp add*: *coeff-liftPoly mpoly-eq-iff*)

lemma *lower0*: (*lowerPoly i j 0*) = *0*
by (*simp add*: *coeff-all-0 coeff-lowerPoly*)

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 (*cases a=0*)
case *True*
show *?thesis* **unfolding** *True lift0 monom-zero lower0* **..**
next
case *False*
have *h1* : *higherPowers i j* ‘ ({*lowerPowers i j m*} ∩ {*x*. ∀ *j*∈{*i*..<*i* + *j*}. *lookup x j* = *0*}) = {*m*}
apply (*simp add*: *lowerPowers.rep-eq higherPowers.rep-eq*)
using *higherPowers-lowerPowers* **.**
have *higher-lower* : *higherPowers i j* (*lowerPowers i j m*) = *m*
using *higherPowers-lowerPowers* **by** *blast*
show *?thesis* **using** *False*
unfolding *insertion-code monomials-monom* **apply** *auto*
unfolding *monomials-lowerPoly monomials-liftPoly* **apply** *auto*

unfolding *More-MPoly-Type.coeff-monom h1* **apply** *auto*
unfolding *coeff-lowerPoly coeff-liftPoly higherPowers-lowerPowers coeff-monom*
apply(*cases* ∃*ja*∈{*i*..<*i* + *j*}. *0* < *lookup* (*lowerPowers i j m*) *ja*)
apply *auto*
by (*simp add*: *lowerPowers-eq*)
qed

lemma *lower-lift* : *insertion f* (*p*::*real mpoly*) = *insertion f* (*lowerPoly i j* (*liftPoly i j p*))
unfolding *insertion-code* **proof**(*induction p rule*: *mpoly-induct*)
case (*monom m a*)
then show *?case* **using** *lower-lift-monom* **unfolding** *insertion-code monomials-lowerPoly monomials-liftPoly coeff-lowerPoly coeff-liftPoly* **by** *auto*
next
case (*sum p1 p2 m a*)
have *h1* : *monomials p1* ∩ *monomials p2* = {} **using** *sum*
by (*metis Int-insert-right-if0 inf-bot-right monomials-monom*)
have *h4* : *monomials* (*lowerPoly i j* (*liftPoly i j* (*p1* + *p2*))) = *monomials* (*lowerPoly i j* (*liftPoly i j* (*p1*))) ∪ *monomials* (*lowerPoly i j* (*liftPoly i j* (*p2*)))
using *monomials-lowerPoly monomials-liftPoly monomials-add-disjoint*[*OF h1*]
by (*simp add*: *monomials-liftPoly monomials-lowerPoly Int-Un-distrib2 image-Un*)
have *h5* : *MPoly-Type.coeff* (*lowerPoly i j* (*liftPoly i j* (*p1* + *p2*))) = *MPoly-Type.coeff* (*lowerPoly i j* (*liftPoly i j* (*p1*))) + *MPoly-Type.coeff* (*lowerPoly i j* (*liftPoly i j* (*p2*)))
unfolding *coeff-lowerPoly coeff-liftPoly MPolyExtension.coeff-add* **by** *auto*
show *?case*
unfolding *MPolyExtension.coeff-add*
unfolding *h4 h5*
unfolding *monomials-add-disjoint*[*OF h1*]
by (*smt IntE coeff-eq-zero-iff disjoint-iff-not-equal finite-monomials h1 higherPowers-lowerPowers imageE monomials-liftPoly monomials-lowerPoly plus-fun-apply sum.IH*(*1*) *sum.IH*(*2*) *sum.cong sum.union-disjoint*
)
qed
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 *safe*
fix *init I xs*
assume *i* = *length* (*init*::*real list*)
then have *i-len* : *length init* = *i* **by** *auto*
have *h*: *higherPoly*$_0$ *i* (*length* (*I*::*real list*)) (*mapping-of p*) ∈ *UNIV*
by *simp*
have *h2* : *vars* (*liftPoly i* (*length I*) *p*) ∩ {*i*..<*i* + *length I*} = {}
using *lift-clear-vars* **by** *auto*

show *insertion* ((*nth-default 0*) (*init* @ *xs*)) *p* = *insertion* ((*nth-default 0*) (*init* @ *I* @ *xs*)) (*liftPoly* (*length init*) (*length I*) *p*)
using *lower-lift insertion-lowerPoly*[*OF h2 i-len refl*, *of xs*] *i-len* **by** *auto*
qed

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*)
using *assms*(*1*)
proof(*induction F arbitrary*: *i init*)
case (*Atom x*)
then show *?case*
apply(*cases x*) **apply** *simp-all* **using** *lift-insertion assms Atom.prems* **by** *force*+
next
case (*ExQ F*)
have *map*: ⋀*y*. (*map-fm-binders* (λ*a x*. *liftAtom* (*i* + *x*) (*amount*) *a*) *F* (*y* + *Suc 0*)) = (*map-fm-binders* (λ*a x*. *liftAtom* (*i* + *1* + *x*) *amount a*) *F y*)
apply(*induction F*) **by**(*simp-all*)
show *?case* **apply** *simp* **apply**(*rule ex-cong1*)
subgoal for *x*
using *map*[*of 0*] **using** *ExQ*(*1*)[*of x*#*init i*+*1*] *ExQ*(*2*)
by *simp*
done
next
case (*AllQ F*)
have *map*: ⋀*y*. (*map-fm-binders* (λ*a x*. *liftAtom* (*i* + *x*) (*amount*) *a*) *F* (*y* + *Suc 0*)) = (*map-fm-binders* (λ*a x*. *liftAtom* (*i* + *1* + *x*) *amount a*) *F y*)
apply(*induction F*) **by**(*simp-all*)
show *?case* **apply** *simp* **apply**(*rule all-cong1*)
subgoal for *x*
using *map*[*of 0*] **using** *AllQ*(*1*)[*of x*#*init i*+*1*] *AllQ*(*2*)
by *simp*
done
next
case (*ExN x1 F*)
have *map*: ⋀*y*. (*map-fm-binders* (λ*a x*. *liftAtom* (*i* + *x*) (*amount*) *a*) *F* (*y* + *x1*)) = (*map-fm-binders* (λ*a x*. *liftAtom* (*i* + *x1* + *x*) *amount a*) *F y*)
apply(*induction F*) **apply**(*simp-all add*: *add.commute add.left-commute*)
apply (*metis add-Suc*)
apply (*metis add-Suc*)
apply (*metis add.assoc*)
by (*metis add.assoc*)
show *?case* **apply** *simp* **apply**(*rule ex-cong1*)
subgoal for *l*
using *map*[*of 0*] *ExN*(*1*)[*of l*@*init i*+*x1*] *ExN*(*2*)
by *auto*
done
next

case (*AllN x1 F*)
have *map*: $\bigwedge y$. (*map-fm-binders* ($\lambda a\ x$. *liftAtom* (i + x) (*amount*) a) F (y + *x1*)) = (*map-fm-binders* ($\lambda a\ x$. *liftAtom* (i + *x1* + x) *amount* a) F y)
apply(*induction F*) **apply**(*simp-all add*: *add.commute add.left-commute*)
apply (*metis add-Suc*)
apply (*metis add-Suc*)
apply (*metis add.assoc*)
by (*metis add.assoc*)
show *?case* **apply** *simp* **apply**(*rule all-cong1*)
subgoal for *l*
using *map*[*of 0*] *AllN*(*1*)[*of l@init i+x1*] *AllN*(*2*)
by *auto*
done
qed *auto*

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 −
define *init* **where** *init* = *take d L*
then have *length init* = *d* **using** *assms* **by** *simp*
then have (*eval F* (*init* @ (*drop d L*)) = *eval* (*liftFm d amount F*) (*init* @ *I* @ (*drop d L*)))
using *eval-liftFm-helper*[*of init d I amount F* (*drop d L*)] *assms* **by** *auto*
then show *?thesis*
unfolding *insert-into.simps assms init-def* **by** *auto*
qed

lemma *not-in-lift* : *var*∉*vars*(*p*::*real mpoly*) ⟹ *var*+*z*∉*vars*(*liftPoly 0 z p*)
proof(*induction p rule*: *mpoly-induct*)
case (*monom m a*)
then show *?case*
using *lift-vars-monom*[*of 0 z m a*] **by** *auto*
next
case (*sum p1 p2 m a*)
show *?case*
using *sum* **using** *vars-lift-add*[*of 0 z p1 p2*]
vars-add[*of p1 p2*]
by (*metis* (*no-types*, *lifting*) *Set.basic-monos*(*7*) *Un-iff monomials.rep-eq vars-add-monom*)
qed

lemma *lift-const* : *insertion f* (*liftPoly 0 z* (*Const* (*C*::*real*))) = *insertion f* (*Const C* :: *real mpoly*)
apply(*cases C*=*0*)
unfolding *insertion-code monomials-Const coeff-Const monomials-liftPoly* **apply** *auto*
unfolding *lower-power-zero*[*of 0 z*] *lookup-zero power.power-0 comm-monoid-mult-class.prod.neutral-const*

coeff-liftPoly coeff-Const
unfolding *higherPowers-def* **by** *auto*

lemma *liftPoly-sub*: *liftPoly 0 z (a − b) = (liftPoly 0 z a) − (liftPoly 0 z b)*
unfolding *liftPoly-def* **apply** (*auto*)
proof −
have *h1*: *mapping-of (a − b) = mapping-of a − mapping-of b*
by (*simp add*: *minus-mpoly.rep-eq*)
have *h2*: *MPoly (higherPoly$_0$ 0 z (mapping-of a − mapping-of b)) =*
MPoly (higherPoly$_0$ 0 z (mapping-of a)) − MPoly (higherPoly$_0$ 0 z (mapping-of b))
proof −
have *h0a*: *higherPoly$_0$ 0 z (mapping-of a − mapping-of b) = (higherPoly$_0$ 0 z (mapping-of a)) − (higherPoly$_0$ 0 z (mapping-of b))*
using *poly-mapping-eq-iff*[**where** *a = higherPoly$_0$ 0 z (mapping-of a − mapping-of b)*, **where** *b = (higherPoly$_0$ 0 z (mapping-of a)) − (higherPoly$_0$ 0 z (mapping-of b))*]
minus-poly-mapping.rep-eq[**where** *x = higherPoly$_0$ 0 z (mapping-of a − mapping-of b)*, **where** *xa = (higherPoly$_0$ 0 z (mapping-of a)) − (higherPoly$_0$ 0 z (mapping-of b))*]
apply (*auto*)
by (*simp add*: *higherPoly$_0$.rep-eq poly-mapping-eqI minus-poly-mapping.rep-eq*)
then show *?thesis*
by (*simp add*: *minus-mpoly.abs-eq*)
qed
show *MPoly (higherPoly$_0$ 0 z (mapping-of (a − b))) =*
MPoly (higherPoly$_0$ 0 z (mapping-of a)) −
MPoly (higherPoly$_0$ 0 z (mapping-of b)) **using** *h1 h2* **by** *auto*
qed

lemma *lift-sub* : *insertion (f*::*nat*⇒*real) (liftPoly 0 z (a − b)) = insertion f (liftPoly 0 z a − liftPoly 0 z b)*
using *liftPoly-sub*[*of z a b*] **by** *auto*

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−
have *insertion f (liftPoly 0 z (c − Const C)) = insertion f ((liftPoly 0 z c) − (liftPoly 0 z (Const C)))*
by (*simp add*: *lift-sub*)
have *... = insertion f (liftPoly 0 z c) − (insertion f (liftPoly 0 z (Const C)))*
using *insertion-sub* **by** *auto*
also have *... = insertion f (liftPoly 0 z c) − C*
using *lift-const*[*of f z C*] *insertion-const* **by** *auto*
then show *?thesis*
using ‹*insertion f (liftPoly 0 z (c − Const C)) = insertion f (liftPoly 0 z c − liftPoly 0 z (Const C))*› *assms calculation* **by** *auto*
qed

end

lemma *lift00* : *liftPoly 0 0* (*a*::*real mpoly*) = *a*
  unfolding *liftPoly-def* **apply** *auto*
  unfolding *higherPoly*$_0$*-def* **apply** *auto*
  unfolding *higherPowers-def* **apply** *auto*
  by (*simp add*: *mapping-of-inverse*)

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*
  unfolding *swap-def* **by** *auto*

lemma *finite-swap-ne*: *finite* {*x*. *f x* ≠ *c*} ⟹ *finite* {*x*. *f* (*swap b i x*) ≠ *c*}
proof −
  assume *finset*: *finite* {*x*. *f x* ≠ *c*}
  let *?A* = {*x*. *f x* ≠ *c*}
  let *?B* = {*x*. *f* (*swap b i x*) ≠ *c*}
  have *finsubset*: *finite* (*?A* − {*i*, *b*}) **using** *finset* **by** *auto*
  have *sames*: (*?A* − {*i*, *b*}) = (*?B* − {*i*, *b*})
    unfolding *swap-def* **by** *auto*
  then have *finite* (*?B* − {*i*, *b*})
    using *finsubset* **by** *auto*
  then have *finBset*: *finite* ((*?B* − {*i*, *b*}) ∪ {*i*, *b*}) **by** *auto*
  then have *?B* ⊆ ((*?B* − {*i*, *b*}) ∪ {*i*, *b*}) **by** *auto*
  then show *?thesis* **using** *finBset* **by** *auto*
qed

lift-definition *swap0*::*nat* ⇒ *nat* ⇒ (*nat* ⇒$_0$ *'a*) ⇒ (*nat* ⇒$_0$ *'a*::*zero*)
  is λ*b i p x*. *p* (*swap b i x*)::*'a*
proof −
  fix *b i*::*nat* **and** *p*::*nat* ⇒ *'a*
  assume *finite* {*x*. *p x* ≠ *0*}
  then have *finite* {*x*. *p* (*swap b i x*) ≠ *0*}
    by (*rule finite-swap-ne*)
  from - *this* **show** *finite* {*x*. *p* (*swap b i x*) ≠ *0*}
    by (*rule finite-subset*) *auto*

qed

lemma *swap0-swap0*: *swap0 n i* (*swap0 n i x*) = *x*
by *transfer* (*force simp*: *swap-def*)

lemma *inj-swap*: *inj* (*swap b i*)
using *swap-swap*
by (*rule inj-on-inverseI*)

lemma *inj-swap0*: *inj* (*swap0 b i*)
using *swap0-swap0*
by (*rule inj-on-inverseI*)

lemma *swap0-eq*: *lookup* (*swap0 b i p*) *x* = *lookup p* (*swap b i x*)
by (*simp-all add*: *swap0.rep-eq*)

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*))
unfolding *eq-onp-def* **apply** *simp*
apply(*rule finite-swap-ne*)
by *auto*

lemma *keys-swap*: *keys* (*swap0 b i m*) = *swap b i* ‘ *keys m*
apply *safe*
subgoal for *x*
unfolding *swap0-def* **apply** *simp*
unfolding *keys.abs-eq*[*OF eq-onp-swap*]
by (*metis* (*mono-tags*, *lifting*) *Reindex.swap-swap image-eqI lookupNotIn mem-Collect-eq*)
subgoal for *x y*
unfolding *swap0-def* **apply** *simp*
unfolding *keys.abs-eq*[*OF eq-onp-swap*]
by (*metis* (*mono-tags*, *lifting*) *Reindex.swap-swap lookup-eq-zero-in-keys-contradict*
mem-Collect-eq)
done

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 −
fix *b i*::*nat* **and** *p*::*nat* ⇒ *'a option*
assume *finite* (*dom p*)
then have *finite* {*x*. *p x* ≠ *None*} **by** (*simp add*: *dom-def*)

have *dom* (λ*x*. *p* (*swap b i x*)) = {*x*. *p* (*swap b i x*) ≠ *None*}
by *auto*
also have *finite* ...

by (*rule finite-swap-ne*) *fact*
finally
have *finite* (*dom* (λ*x*. *p* (*swap b i x*))) **.**
from - *this*
show *finite* (*dom* (λ*x*. *p* (*swap b i x*)))
by (*rule finite-subset*) (*auto split*: *if-splits*)
qed

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 −
have ∗: *map-of* (*map* (λ(*k*, *y*). (*swap b i k*, *y*)) (*xs*)) *x* =
map-of xs (*swap b i x*)
for *x*
apply (*rule map-of-map-key-inverse-fun-eq*)
unfolding *swap-swap* **by** *auto*
show *?thesis*
unfolding *swap*$_f$*-def* **apply** *simp*
unfolding *fmlookup-of-list*
unfolding *Finite-Map.fmap-of-list.abs-eq*
using *map-of-map-key-inverse-fun-eq*[**where** *f*=*swap b i*, **where** *g*=*swap b i*,
where *xs*=*xs*]
unfolding *swap-swap*
apply *simp*
by *presburger*
qed

lemma *compute-swap*[*code*]: *swap0 n i* (*Pm-fmap xs*) = *Pm-fmap* (*swap*$_f$ *n i xs*)
apply(*rule poly-mapping-eqI*)
by (*auto simp*: *swap*$_f$*.rep-eq swap0.rep-eq fmlookup-default-def swap-def*
split: *option.splits*)

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 −
fix *b i* **and** *mp*::(*nat* ⇒$_0$ *nat*) ⇒ *'a*
assume *finite* {*x*. *mp x* ≠ *0*}
have {*x*. *mp* (*swap0 b i x*) ≠ *0*} = (*swap0 b i* −‘ {*x*. *mp x* ≠ *0*})
(**is** *?set* = *?vimage*)
by *auto*
also
from *finite-vimageI*[*OF* ‹*finite* -› *inj-swap0*]
have *finite ?vimage* **.**
finally show *finite ?set* **.**
qed

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

by *transfer auto*

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

lift-definition $swapPoly_f$::$nat \Rightarrow nat \Rightarrow ((nat \Rightarrow_0 nat), 'a::zero)fmap \Rightarrow ((nat \Rightarrow_0 nat), 'a)fmap$ **is**
$\lambda b\ i\ (mp::((nat \Rightarrow_0 nat) \rightharpoonup 'a))\ mon::(nat \Rightarrow_0 nat).\ mp\ (swap0\ b\ i\ mon)$
proof −− TODO: this is exactly the same proof as the one for $lowerPoly_0$
fix *b i* **and** *mp*::$(nat \Rightarrow_0 nat) \Rightarrow 'a\ option$
assume *finite* (*dom mp*)
also have $dom\ mp = \{x.\ mp\ x \neq None\}$ **by** *auto*
finally have $finite\ \{x.\ mp\ x \neq None\}$ **.**
have $(dom\ (\lambda mon.\ mp\ (swap0\ b\ i\ mon))) = \{mon.\ mp\ (swap0\ b\ i\ mon) \neq None\}$
(**is** *?set* = -)
by (*auto split*: *if-splits*)
also have $\ldots = swap0\ b\ i - `\ \{x.\ mp\ x \neq None\}$ (**is** - = *?vimage*)
by *auto*
also
from *finite-vimageI*[*OF* ‹*finite* $\{x.\ mp\ x \neq None\}$› *inj-swap0*]
have *finite ?vimage* **.**
finally show *finite ?set* **.**
qed

lemma $keys\text{-}swap_0$: $keys\ (swapPoly_0\ b\ i\ mp) = swap0\ b\ i\ `\ (keys\ mp)$
apply (*auto*)
subgoal for *x*
apply (*rule image-eqI*[**where** *x*=*swap0 b i x*])
by (*auto simp*: *swap0-swap0 in-keys-iff* $swapPoly_0.rep\text{-}eq$)
subgoal for *x*
apply (*auto simp*: *in-keys-iff* $swapPoly_0.rep\text{-}eq$)
by (*simp add*: *swap0-swap0*)
done

end

lemma $compute\text{-}swapPoly_0[code]$: $swapPoly_0\ n\ i\ (Pm\text{-}fmap\ m) = Pm\text{-}fmap\ (swapPoly_f\ n\ i\ m)$
by (*auto simp*: $swapPoly_0.rep\text{-}eq$ *fmlookup-default-def* $swapPoly_f.rep\text{-}eq$
split: *option.splits*
intro!: *poly-mapping-eqI*)

lemma $compute\text{-}swapPoly_f[code]$: $swapPoly_f\ n\ i\ (fmap\text{-}of\text{-}list\ xs) =$
(*fmap-of-list* (*map* $(\lambda(mon,\ c).\ (swap0\ n\ i\ mon,\ c))$
xs))
apply (*rule sym*)
apply (*rule fmap-ext*)
unfolding $swapPoly_f.rep\text{-}eq$ *fmlookup-of-list*

apply (*subst map-of-map-key-inverse-fun-eq*[**where** *g*=*swap0 n i*])
unfolding *swap0-swap0* **by** *auto*

end
end

lift-definition *swap-poly*::*nat* ⇒ *nat* ⇒ *'a*::*zero mpoly* ⇒ *'a mpoly* **is** $swapPoly_0$.

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*)
by (*transfer'*) (*simp add*: $swapPoly_0$.*rep-eq*)

lemma *monomials-swap-poly*: *monomials* (*swap-poly b i mp*) = *swap0 b i* ' (*monomials mp*)
by *transfer'* (*simp add*: $keys\text{-}swap_0$)

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*
by *auto*

lemma *inj-on* : *inj-on* (*swap0 a b*) (*monomials p*)
unfolding *inj-on-def*
by (*metis swap0-swap0*)

lemma *inj-on'* : *inj-on* (*swap a b*) (*keys m*)
unfolding *inj-on-def*
by (*meson Reindex.inj-swap injD*)

lemma *swap-list* :
assumes $a < length\ L$
assumes $b < length\ L$
shows $nth\text{-}default\ 0\ (L[b := L\ !\ a,\ a := L\ !\ b])\ (swap\ a\ b\ xa) = nth\text{-}default\ 0\ L\ xa$
using *assms* **unfolding** *swap-def* **apply** *auto*
apply (*simp-all add*: *nth-default-nth*)
by (*simp add*: *nth-default-def*)

lemma *swap-poly* :
assumes $length\ L > a$
assumes $length\ L > b$
shows $insertion\ (nth\text{-}default\ 0\ L)\ p = insertion\ (nth\text{-}default\ 0\ (swap\text{-}list\ a\ b\ L))\ (swap\text{-}poly\ a\ b\ p)$
unfolding *insertion-code* **apply** *auto*
unfolding *monomials.abs-eq*
unfolding *coeff-swap-poly monomials-swap-poly* **apply** *auto*
unfolding *Groups-Big.comm-monoid-add-class.sum.reindex*[*OF inj-on*] **apply** *simp*
unfolding *swap0-swap0*
unfolding *keys-swap*
unfolding *Groups-Big.comm-monoid-mult-class.prod.reindex*[*OF inj-on'*]
apply *simp*
unfolding *swap0-eq swap-swap swap-list*[*OF assms*] **by** *auto*

lemma *swap-fm* :
assumes $length\ L > a$
assumes $length\ L > b$
shows $eval\ F\ L = eval\ (swap\text{-}fm\ a\ b\ F)\ (swap\text{-}list\ a\ b\ L)$
using *assms* **proof**(*induction F arbitrary*: *a b L*)
case *TrueF*
then show *?case* **by** *auto*
next
case *FalseF*
then show *?case* **by** *auto*
next
case (*Atom At*)
then show *?case* **apply**(*cases At*) **using** *swap-poly*[*OF Atom*(*1*) *Atom*(*2*)] **by** *auto*
next
case (*And F1 F2*)
show *?case* **using** *And*(*1*)[*OF And*(*3*−*4*)] *And*(*2*)[*OF And*(*3*−*4*)] **by** *auto*
next
case (*Or F1 F2*)
then show *?case* **using** *Or*(*1*)[*OF Or*(*3*−*4*)] *Or*(*2*)[*OF Or*(*3*−*4*)] **by** *auto*
next
case (*Neg F*)
then show *?case* **using** *Neg*(*1*)[*OF Neg*(*2*−*3*)] **by** *auto*

```
next
  case (ExQ F)
  show ?case apply simp
    apply(rule ex-cong1)
    subgoal for x
     using ExQ(1)[of Suc a x#L Suc b] unfolding swap-list-cons using ExQ(2−3)
by simp
    done
next
  case (AllQ F)
  then show ?case apply simp
    apply(rule all-cong1)
    subgoal for x
     using AllQ(1)[of Suc a x#L Suc b] unfolding swap-list-cons using AllQ(2−3)
by simp
    done
next
  case (ExN i F)
  show ?case
    apply simp
    apply(rule ex-cong1)
    subgoal for l
      using ExN(1)[of a+i l@L b+i]
    by (smt (verit, del-insts) ExN.prems(1) ExN.prems(2) add.commute add-diff-cancel-right'
add-less-cancel-left length-append list-update-append not-add-less2 nth-append swap-list.elims)

    done
next
  case (AllN i F)
  then show ?case
    apply simp apply(rule all-cong1)
   by (smt (z3) add.commute add-diff-cancel-right' le-add2 length-append less-diff-conv2
list-update-append not-add-less2 nth-append)
qed

lemma eval (ExQ (ExQ F)) L = eval (ExQ (ExQ (swap-fm 0 1 F))) L
  apply simp
  apply safe
  subgoal for i j
    apply(rule exI[where x=j])
    apply(rule exI[where x=i])
    using swap-fm[of 0 j # i # L Suc 0 F]
    by simp
  subgoal for i j
    apply(rule exI[where x=j])
    apply(rule exI[where x=i])
    using swap-fm[of 0 i # j # L Suc 0 F]
    by simp
  done
```

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*)
using *swap-fm*[*OF assms*, *of Atom F*] **by** *auto*
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))
  by pat-completeness auto
termination
  apply(relation measures [λA. depth' A])
  by auto
```

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 : ∀Γ. (¬ eval (nnf (Neg φ)) Γ) = eval (nnf φ) Γ
  apply(induction φ)
          apply(simp-all)
  using aNeg-aEval apply blast
  using aNeg-aEval by blast

theorem eval-nnf : ∀Γ. eval φ Γ = eval (nnf φ) Γ
  apply(induction φ)apply(simp-all) using neg-nnf by blast


theorem negation-free-nnf : negation-free (nnf φ)
proof(induction depth φ arbitrary : φ rule: nat-less-induct )
  case 1
  then show ?case
  proof(induction φ)
    case (And φ1 φ2)
    then show ?case apply simp
      by (metis less-Suc-eq-le max.cobounded1 max.cobounded2)
  next
    case (Or φ1 φ2)
    then show ?case apply simp
      by (metis less-Suc-eq-le max.cobounded1 max.cobounded2)
  next
    case (Neg φ)
    then show ?case proof (induction φ)
      case (And φ1 φ2)
      then show ?case apply simp
        by (metis less-Suc-eq max-less-iff-conj not-less-eq)
    next
      case (Or φ1 φ2)
      then show ?case apply simp
```

```
      by (metis less-Suc-eq max-less-iff-conj not-less-eq)
    next
      case (Neg φ)
      then show ?case
        by (metis Suc-eq-plus1 add-lessD1 depth.simps(6) lessI nnf.simps(12))
    qed auto
  qed auto
qed


lemma groupQuantifiers-eval : eval F L = eval (groupQuantifiers F) L
  apply(induction F arbitrary: L  rule:groupQuantifiers.induct)
  unfolding doubleExist unwrapExist unwrapExist' unwrapExist'' doubleForall unwrapForall unwrapForall' unwrapForall''
  apply (auto)
  using doubleExist doubleExist unwrapExist unwrapExist' unwrapExist'' doubleForall unwrapForall unwrapForall' unwrapForall'' apply auto
  by metis+


theorem simp-atom-eval : aEval a xs = eval (simp-atom a) xs
proof(cases a)
  case (Less p)
  then show ?thesis by(cases get-if-const p)(simp-all add:get-if-const-insertion)
next
  case (Eq p)
  then show ?thesis by(cases get-if-const p)(simp-all add:get-if-const-insertion)
next
  case (Leq p)
  then show ?thesis by(cases get-if-const p)(simp-all add:get-if-const-insertion)
next
  case (Neq p)
  then show ?thesis by(cases get-if-const p)(simp-all add:get-if-const-insertion)
qed

lemma simpfm-eval : ∀ L. eval φ L = eval (simpfm φ) L
  apply(induction φ)
  apply(simp-all add: simp-atom-eval eval-and eval-or)
  using eval-neg by blast

lemma exQ-clearQuantifiers:
  assumes ExQ : ⋀xs. eval (clearQuantifiers φ) xs = eval φ xs
  shows eval (clearQuantifiers (ExQ φ)) xs = eval (ExQ φ) xs
proof−
  define φ' where φ' = clearQuantifiers φ
  have h : freeIn 0 φ' ⟹ (eval (lowerFm 0 1 φ') xs = eval (ExQ φ') xs)
    using eval-lowerFm by simp
  have eval (clearQuantifiers (ExQ φ)) xs =
      eval (if freeIn 0 φ' then lowerFm 0 1 φ' else ExQ φ') xs
```

```
    using φ'-def by simp
  also have ... = eval (ExQ φ) xs
    apply(cases freeIn 0 φ')
    using h ExQ φ'-def by(simp-all)
  finally show ?thesis
    by simp
qed

lemma allQ-clearQuantifiers :
  assumes AllQ : ⋀xs. eval (clearQuantifiers φ) xs = eval φ xs
  shows eval (clearQuantifiers (AllQ φ)) xs = eval (AllQ φ) xs
proof−
  define φ' where φ' = clearQuantifiers φ
  have freeIn 0 φ' ⟹ (eval (ExQ φ') xs) = eval (AllQ φ') xs
    by (simp add: var-not-in-eval2)
  then have h : freeIn 0 φ' ⟹ (eval (lowerFm 0 1 φ') xs = eval (AllQ φ') xs)
    using eval-lowerFm by simp
  have eval (clearQuantifiers (AllQ φ)) xs =
      eval (if freeIn 0 φ' then lowerFm 0 1 φ' else AllQ φ') xs
    using φ'-def by simp
  also have ... = eval (AllQ φ) xs
    apply(cases freeIn 0 φ')
    using h AllQ φ'-def by(simp-all)
  finally show ?thesis
    by simp
qed

lemma clearQuantifiers-eval : eval (clearQuantifiers φ) xs = eval φ xs
proof(induction φ arbitrary : xs)
  case (Atom x)
  then show ?case using simp-atom-eval by simp
next
  case (And φ1 φ2)
  then show ?case using eval-and by simp
next
  case (Or φ1 φ2)
  then show ?case using eval-or by simp
next
  case (Neg φ)
  then show ?case using eval-neg by auto
next
  case (ExQ φ)
  then show ?case using exQ-clearQuantifiers by simp
next
  case (AllQ φ)
  then show ?case using allQ-clearQuantifiers by simp
next
  case (ExN x1 φ)
  then show ?case proof(induction x1 arbitrary:φ)
```

```
    case 0
    then show ?case by auto
  next
    case (Suc x1)
    show ?case
      using Suc(1)[of ExQ φ, OF exQ-clearQuantifiers[OF Suc(2)]]
      apply simp
     using Suc-eq-plus1 ‹eval (clearQuantifiers (ExN x1 (ExQ φ))) xs = eval (ExN
x1 (ExQ φ)) xs› eval.simps(10) unwrapExist' by presburger
  qed
next
  case (AllN x1 φ)
  then show ?case proof(induction x1 arbitrary:φ)
    case 0
    then show ?case by auto
  next
    case (Suc x1)
    show ?case
      using Suc(1)[of AllQ φ, OF allQ-clearQuantifiers[OF Suc(2)]]
      apply simp
      using unwrapForall' by force
  qed
qed auto

lemma push-forall-eval-AllQ : ∀ xs. eval (AllQ φ) xs = eval (push-forall (AllQ
φ)) xs
proof(induction φ)
  case TrueF
  then show ?case by simp
next
  case FalseF
  then show ?case by simp
next
  case (Atom x)
  then show ?case
    using aEval-lowerAtom eval.simps(1) eval.simps(8) push-forall.simps(11) by
presburger
next
  case (And φ1 φ2)
  {fix xs
    have eval (AllQ (And φ1 φ2)) xs = (∀ x. eval φ1 (x#xs) ∧ eval φ2 (x#xs))
      by simp
    also have ... = ((∀ x. eval φ1 (x#xs)) ∧ (∀ x. eval φ2 (x#xs)))
      by blast
    also have ... = eval (push-forall (AllQ (And φ1 φ2))) xs
      using And eval-and by(simp)
    finally have eval (AllQ (And φ1 φ2)) xs = eval (push-forall (AllQ (And φ1
φ2))) xs
      by simp
```

```
  }
  then show ?case by simp
next
  case (Or φ1 φ2)
  then show ?case proof(cases freeIn 0 φ1)
    case True
    then have h : freeIn 0 φ1
      by simp
    then show ?thesis proof(cases freeIn 0 φ2)
      case True
      {fix xs
        have ∃ x. eval φ1 (x # xs) = eval (lowerFm 0 1 φ1) xs
          using eval-lowerFm h eval.simps(7) by blast
        then have h1 : ∀ x. eval φ1 (x # xs) = eval (lowerFm 0 1 φ1) xs
          using h var-not-in-eval2 by blast
        have ∃ x. eval φ2 (x # xs) = eval (lowerFm 0 1 φ2) xs
          using eval-lowerFm True eval.simps(7) by blast
        then have h2 : ∀ x. eval φ2 (x # xs) = eval (lowerFm 0 1 φ2) xs
          using True var-not-in-eval2 by blast
       have eval (AllQ (Or φ1 φ2)) xs = eval (push-forall (AllQ (Or φ1 φ2))) xs
          by(simp add:h h1 h2 True eval-or)
      }
      then show ?thesis by simp
    next
      case False
      {fix xs
        have ∃ x. eval φ1 (x # xs) = eval (lowerFm 0 1 φ1) xs
          using eval-lowerFm h eval.simps(7) by blast
        then have ∀ x. eval φ1 (x # xs) = eval (lowerFm 0 1 φ1) xs
          using True var-not-in-eval2 by blast
         then have eval (AllQ (Or φ1 φ2)) xs = eval (push-forall (AllQ (Or φ1
φ2))) xs
          by(simp add:h False eval-or)
      }
      then show ?thesis by simp
    qed
  next
    case False
    then have h : ¬freeIn 0 φ1
      by simp
    then show ?thesis proof(cases freeIn 0 φ2)
      case True
      {fix xs
        have ∃ x. eval φ2 (x # xs) = eval (lowerFm 0 1 φ2) xs
          using eval-lowerFm True eval.simps(7) by blast
        then have ∀ x. eval φ2 (x # xs) = eval (lowerFm 0 1 φ2) xs
          using True var-not-in-eval2 by blast
         then have eval (AllQ (Or φ1 φ2)) xs = eval (push-forall (AllQ (Or φ1
φ2))) xs
```

```
        by(simp add:h True eval-or)
      }
      then show ?thesis by simp
    next
      case False
      then show ?thesis by(simp add:h False eval-or)
    qed
  qed
next
  case (Neg φ)
  {fix xs
    have freeIn 0 (Neg φ) ⟹ (eval (ExQ (Neg φ)) xs) = eval (AllQ (Neg φ)) xs
      using var-not-in-eval2 eval.simps(7) eval.simps(8) by blast
    then have h : freeIn 0 (Neg φ) ⟹ (eval (lowerFm 0 1 (Neg φ)) xs = eval
(AllQ (Neg φ)) xs)
      using eval-lowerFm by blast
    have eval (push-forall (AllQ (Neg φ))) xs =
      eval (if freeIn 0 (Neg φ) then lowerFm 0 1 (Neg φ) else AllQ (Neg φ)) xs
      by simp
    also have ... = eval (AllQ (Neg φ)) xs
      apply(cases freeIn 0 (Neg φ))
      using h  by(simp-all)
    finally have eval (push-forall (AllQ (Neg φ))) xs = eval (AllQ (Neg φ)) xs
      by simp
  }
  then show ?case by simp
next
  case (ExQ φ)
  {fix xs
    have freeIn 0 (ExQ φ) ⟹ (eval (ExQ (ExQ φ)) xs) = eval (AllQ (ExQ φ))
xs
      using var-not-in-eval2 eval.simps(7) eval.simps(8) by blast
    then have h : freeIn 0 (ExQ φ) ⟹ (eval (lowerFm 0 1 (ExQ φ)) xs = eval
(AllQ (ExQ φ)) xs)
      using eval-lowerFm by blast
    have eval (push-forall (AllQ (ExQ φ))) xs =
      eval (if freeIn 0 (ExQ φ) then lowerFm 0 1 (ExQ φ) else AllQ (ExQ φ)) xs
      by simp
    also have ... = eval (AllQ (ExQ φ)) xs
      apply(cases freeIn 0 (ExQ φ))
      using h  by(simp-all)
    finally have eval (push-forall (AllQ (ExQ φ))) xs = eval (AllQ (ExQ φ)) xs
      by simp
  }
  then show ?case by simp
next
  case (AllQ φ)
  {fix xs
    have freeIn 0 (AllQ φ) ⟹ (eval (ExQ (AllQ φ)) xs) = eval (AllQ (AllQ φ))
```

```
xs
      using var-not-in-eval2 eval.simps(7) eval.simps(8) by blast
    then have h : freeIn 0 (AllQ φ) ⟹ (eval (lowerFm 0 1 (AllQ φ)) xs = eval
(AllQ (AllQ φ)) xs)
      using eval-lowerFm by blast
    have eval (push-forall (AllQ (AllQ φ))) xs =
      eval (if freeIn 0 (AllQ φ) then lowerFm 0 1 (AllQ φ) else AllQ (AllQ φ)) xs
      by simp
    also have ... = eval (AllQ (AllQ φ)) xs
      apply(cases freeIn 0 (AllQ φ))
      using h AllQ  by(simp-all)
    finally have eval (push-forall (AllQ (AllQ φ))) xs = eval (AllQ (AllQ φ)) xs
      by simp
  }
  then show ?case by simp
next
  case (ExN x1 φ)
  then show ?case
   using eval.simps(7) eval.simps(8) eval-lowerFm push-forall.simps(17) var-not-in-eval2
by presburger
next
  case (AllN x1 φ)
  then show ?case
   using eval.simps(7) eval.simps(8) eval-lowerFm push-forall.simps(18) var-not-in-eval2
by presburger
qed

lemma push-forall-eval : ∀ xs. eval φ xs = eval (push-forall φ) xs
proof(induction φ)
  case (Atom x)
  then show ?case using simp-atom-eval by simp
next
  case (And φ1 φ2)
  then show ?case using eval-and by auto
next
  case (Or φ1 φ2)
  then show ?case using eval-or by auto
next
  case (Neg φ)
  then show ?case using eval-neg by auto
next
  case (AllQ φ)
  then show ?case using push-forall-eval-AllQ by blast
next
  case (ExN x1 φ)
  then show ?case
    using eval.simps(10) push-forall.simps(7) by presburger
qed auto
```

lemma *map-fm-binders-negation-free* :
assumes *negation-free* φ
shows *negation-free* (*map-fm-binders f* φ *n*)
using *assms* **apply**(*induction* φ *arbitrary* : *n*) **by** *auto*

lemma *negation-free-and* :
assumes *negation-free* φ
assumes *negation-free* ψ
shows *negation-free* (*and* φ ψ)
using *assms* **unfolding** *and-def* **by** *simp*

lemma *negation-free-or* :
assumes *negation-free* φ
assumes *negation-free* ψ
shows *negation-free* (*or* φ ψ)
using *assms* **unfolding** *or-def* **by** *simp*

lemma *push-forall-negation-free-all* :
assumes *negation-free* φ
shows *negation-free* (*push-forall* (*AllQ* φ))
using *assms* **proof**(*induction* φ)
case (*And* φ*1* φ*2*)
show *?case* **apply** *auto*
apply(*rule negation-free-and*)
using *And* **by** *auto*
next
case (*Or* φ*1* φ*2*)
show *?case*
apply *auto*
apply(*rule negation-free-or*)
using *Or map-fm-binders-negation-free negation-free-or* **by** *auto*
next
case (*ExQ* φ)
then show *?case* **using** *map-fm-binders-negation-free* **by** *auto*
next
case (*AllQ* φ)
then show *?case* **using** *map-fm-binders-negation-free* **by** *auto*
next
case (*ExN x1* φ)
then show *?case* **using** *map-fm-binders-negation-free* **by** *auto*
next
case (*AllN x1* φ)
then show *?case* **using** *map-fm-binders-negation-free* **by** *auto*
qed *auto*

lemma *push-forall-negation-free* :
assumes *negation-free* φ
shows *negation-free*(*push-forall* φ)
using *assms* **proof**(*induction* φ)

```
  case (Atom A)
  then show ?case apply(cases A) by auto
next
  case (And φ1 φ2)
  then show ?case by (auto simp add: and-def)
next
  case (Or φ1 φ2)
  then show ?case by (auto simp add: or-def)
next
  case (AllQ φ)
  then show ?case using push-forall-negation-free-all by auto
qed auto


lemma to-list-insertion: insertion f p = sum-list [insertion f term * (f v) ^ i.
(term,i)←(to-list v p)]
proof−
 have h1 : insertion f p = insertion f (∑ i≤MPoly-Type.degree p v. isolate-variable-sparse
p v i * Var v ^ i)
    using sum-over-zero by auto
  have h2 : insertion f (Var v) = f v by (auto simp: monomials-Var coeff-Var
insertion-code)
  define d where d = MPoly-Type.degree p v
  define g where g = (λx. insertion f (isolate-variable-sparse p v x) * f v ^ x)
  have h3 : insertion f (isolate-variable-sparse p v d) * f v ^ d = g d using g-def
by auto
  show ?thesis unfolding h1
    insertion-sum′ insertion-mult insertion-pow h2 apply auto unfolding d-def[symmetric]
g-def[symmetric]
     h3  proof(induction d)
    case 0
    then show ?case by auto
  next
    case (Suc d)
    show ?case
      apply (auto simp add: Suc ) unfolding g-def by auto
  qed
qed

lemma to-list-p: p = sum-list [term * (Var v) ^ i. (term,i)←(to-list v p)]
proof−
  define d where d = MPoly-Type.degree p v
  have (∑ i≤MPoly-Type.degree p v. isolate-variable-sparse p v i * Var v ^ i) =
(∑ (term, i)←to-list v p. term * Var v ^ i)
    unfolding to-list.simps d-def[symmetric] apply(induction d) by auto
  then show ?thesis
    using sum-over-zero[of p v]
    by auto
qed
```

```
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
  using assms
proof(induction L arbitrary : a b L')
  case Nil
  then show ?case using assms by auto
next
  case (Cons A L)
  then show ?case proof(cases A)
    case (Pair p i)
    show ?thesis using Cons unfolding Pair apply(cases p=0) by auto
  qed
qed
lemma compare :
  assumes (a,b)=chophelper L L'
  shows chop L = b
  using assms
proof(induction L arbitrary : a b L')
  case Nil
  then show ?case by auto
next
  case (Cons A L)
  then show ?case proof(cases A)
    case (Pair p i)
    show ?thesis using Cons unfolding Pair apply(cases p=0) by auto
  qed
qed
lemma allzero:
  assumes ∀ (p,i)∈set(L'). p=0
  assumes (a,b)=chophelper L L'
  shows ∀ (p,i)∈set(a). p=0
  using assms proof(induction L arbitrary : a b L')
  case Nil
  then show ?case by auto
next
  case (Cons t L)
  then show ?case
  proof(cases t)
    case (Pair p i)
    show ?thesis proof(cases p=0)
      case True
```

```
    have h1: ∀x∈set (L' @ [(0, i)]). case x of (p, i) ⇒ p = 0
      using Cons(2) by auto
    show ?thesis using Cons(1)[OF h1] Cons(3) True unfolding Pair by auto
   next
    case False
    then show ?thesis using Cons unfolding Pair by auto
   qed
  qed
qed

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]
 using to-list-insertion[of f p v] preserve[OF assms, symmetric] unfolding List.append.left-neutral
  by (simp del: to-list.simps)

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−
  have h1 : ∀(p, i)∈set []. p = 0 by auto
  have (∑(term, i)←a. insertion f term * f v ^ i) = 0
    using allzero[OF h1 assms] proof(induction a)
    case Nil
    then show ?case by auto
  next
    case (Cons a1 a2)
    then show ?case
     apply(cases a1) by simp
  qed
  then show ?thesis using separate[OF assms, of f] by auto
qed

lemma insertion-chop :
   shows insertion f p = sum-list [insertion f term * (f v) ^ i. (term,i)←(chop
(to-list v p))]
proof(cases chophelper (to-list v p) [])
  case (Pair a b)
  show ?thesis using chopped[OF Pair[symmetric], of f] unfolding compare[OF
Pair[symmetric], symmetric] .
qed

lemma sorted : sorted-wrt (λ(-,i).λ(-,i'). i<i') (to-list v p)
proof −
  define d where d = MPoly-Type.degree p v
  show ?thesis unfolding to-list.simps d-def[symmetric]
  proof(induction d)
    case 0
```

```
    then show ?case by auto
  next
    case (Suc d)
    have h : (map (λx. (isolate-variable-sparse p v x, x)) [0..<Suc d + 1]) =
      (map (λx. (isolate-variable-sparse p v x, x)) [0..<Suc d]) @ [(isolate-variable-sparse
p v (Suc d), (Suc d))]
      by auto
    show ?case
      unfolding sorted-wrt-append h
      using Suc
      by auto
  qed
qed

lemma sublist : sublist (chop L) L
proof(induction L)
  case Nil
  then show ?case by auto
next
  case (Cons a L)
  then show ?case proof(cases a)
    case (Pair a b)
    show ?thesis using Cons unfolding Pair apply auto
      by (simp add: sublist-Cons-right)
  qed
qed



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−
  have h : sorted-wrt (λ(-, i) (-, y). i < y) (chop (to-list v p))
  proof−
    define L where L = to-list v p
   show ?thesis using sublist[of to-list v p] sorted[of v p] unfolding L-def[symmetric]
      by (metis sorted-wrt-append sublist-def)
  qed
  then have ∀(term,d)∈set(chop (to-list v p)). d≥i
    unfolding assms[symmetric]  by fastforce
  then have simp : ∀(term,d)∈set(chop(to-list v p)). f v ^(d − i) * f v ^i = f v
^d
    unfolding semiring-normalization-rules(26) by(auto simp del: to-list.simps)
  have insertion f p = sum-list [insertion f term * (f v) ^i. (term,i)←(chop (to-list
v p))] using insertion-chop[of f p v] .
  also have ...= (∑(term, d)←chop (to-list v p). insertion f term * f v ^(d−i) *
f v ^i)
```

```
    using simp
    by (smt Pair-inject case-prodE map-eq-conv mult.assoc split-cong)
  also have ... =  (∑(term, d)←chop (to-list v p). insertion f term * f v ^ (d −
i)) * f v ^ i
  proof−
    define d where d = chop(to-list v p)
    define a where a = f v ^ i
    define b where b = (λ(term, d). insertion f term * f v ^ (d − i))
    have h : (∑(term, d)←d. insertion f term * f v ^ (d − i) * a) = (∑(term,
d)←d. b (term,d) * a)
      using b-def by auto
    show ?thesis unfolding d-def[symmetric] a-def[symmetric]  b-def[symmetric]
h  apply(induction d) apply simp apply auto
      by (simp add: ring-class.ring-distribs(2))
  qed
  finally show ?thesis by auto
qed

lemma insert-Var-Zero : insertion f (Var v) = f v
  unfolding insertion-code monomials-Var apply auto
  unfolding coeff-Var by simp


lemma decreasePower-insertion :
  assumes decreasePower v p = (p',i)
  shows insertion f p = insertion f p' * (f v)^i
proof(cases chop (to-list v p))
  case Nil
  then show ?thesis
    using assms by auto
next
  case (Cons a list)
  then show ?thesis
  proof(cases a)
    case (Pair coef i')
    have i'-def : i'=i using Cons assms Pair by auto
    have chop: chop (to-list v p) = (coef, i) # list using Cons assms unfolding
i'-def Pair by auto
    have p'-def :  p' = (∑(term, x)←chop (to-list v p). term * Var v ^ (x − i))
      using assms Cons Pair by auto
    have p'-insertion : insertion f p' = (∑(term, x)←chop (to-list v p). insertion
f term * f v ^ (x − i))
    proof−
      define d where d = chop (to-list v p)
     have insertion f p' = insertion f (∑(term, x)←chop (to-list v p). term * Var
v ^ (x − i)) using p'-def by auto
      also have ... = (∑(term, x)←chop (to-list v p).  insertion f (term * Var v
^ (x − i)))
        unfolding d-def[symmetric] apply(induction d) apply simp apply(simp
```

```
add:insertion-add) by auto
      also have ... = (∑ (term, x)←chop (to-list v p). insertion f term * f v ^ (x
- i)) unfolding insertion-mult insertion-pow insert-Var-Zero by auto
      finally show ?thesis by auto
    qed
    have h : (coef, i') # list = chop (to-list v p)  using Cons unfolding Pair by
auto
    show ?thesis unfolding p'-insertion
      using move-exp[OF h, of f] unfolding i'-def .
  qed
qed


lemma unpower-eval: eval (unpower v φ) L = eval φ L
proof(induction φ arbitrary: v L)
  case TrueF
  then show ?case by auto
next
  case FalseF
  then show ?case by auto
next
  case (Atom At)
  then show ?case proof(cases At)
    case (Less p)
    obtain q i where h: decreasePower v p = (q, i)
      using prod.exhaust-sel by blast
    have p : ⋀f. insertion f p = insertion f q* (f v)^i
      using decreasePower-insertion[OF h] by auto
    show ?thesis
    proof(cases i=0)
      case True
      then show ?thesis unfolding Less unpower.simps h  by auto
    next
      case False
      obtain x where x-def : Suc x = i using False
        using not0-implies-Suc by auto
      have h1 : i mod 2 = 0 ⟹
    (insertion (nth-default 0 L) q < 0 ∧
     insertion (nth-default 0 L) (Var v) ≠ 0) =
    (insertion (nth-default 0 L) q * nth-default 0 L v ^ i < 0)
      proof −
        assume i mod 2 = 0
        then have ∀ r. ¬ (r::real) ^ i < 0
          by presburger
        then show ?thesis
         by (metis ‹⋀thesis. (⋀x. Suc x = i ⟹ thesis) ⟹ thesis› insert-Var-Zero
linorder-neqE-linordered-idom mult-less-0-iff power-0-Suc power-eq-0-iff)
      qed
      show ?thesis
```

```
      unfolding Less unpower.simps h x-def [symmetric] apply simp
      unfolding x-def p apply (cases ‹even i›)
     using h1 apply (auto simp add: insertion-neg insert-Var-Zero mult-less-0-iff
not-less zero-less-mult-iff elim: oddE)
      done
   qed
 next
   case (Eq p)
   obtain q i where h: decreasePower v p = (q, i)
     using prod.exhaust-sel by blast
   have p : ⋀f. insertion f p = insertion f q* (f v)^i
     using decreasePower-insertion[OF h] by auto
  show ?thesis unfolding Eq unpower.simps h apply simp apply(cases i) apply
simp
     apply simp unfolding p apply simp
     by (metis insert-Var-Zero)
 next
   case (Leq p)
   obtain q i where h: decreasePower v p = (q, i)
     using prod.exhaust-sel by blast
   have p : ⋀f. insertion f p = insertion f q* (f v)^i
     using decreasePower-insertion[OF h] by auto
   show ?thesis
   proof(cases i=0)
     case True
     then show ?thesis unfolding Leq unpower.simps h  by auto
   next
     case False
     obtain x where x-def : Suc x = i using False
       using not0-implies-Suc by auto
     define a where a = insertion (nth-default 0 L) q
     define x' where x' = nth-default 0 L v
     show ?thesis unfolding Leq unpower.simps h x-def[symmetric] apply simp
       unfolding x-def p apply(cases i mod 2 = 0) unfolding insert-Var-Zero
insertion-mult insertion-pow insertion-neg apply simp-all
       unfolding a-def[symmetric] x'-def[symmetric]
     proof−
       assume i mod 2 = 0
       then have x' ^ i ≥0
         by (simp add: ‹i mod 2 = 0› even-iff-mod-2-eq-zero zero-le-even-power)
       then show (a ≤ 0 ∨ x' = 0) = (a * x' ^ i ≤ 0)
         using Rings.ordered-semiring-0-class.mult-nonpos-nonneg[of a x'^i]
         apply auto
         unfolding Rings.linordered-ring-strict-class.mult-le-0-iff
         apply auto
         by (simp add: False power-0-left)
     next
       assume h:  i mod 2 = Suc 0
       show (a = 0 ∨ a < 0 ∧ 0 ≤ x' ∨ 0 < a ∧ x' ≤ 0) = (a * x' ^ i ≤ 0)
```

using *h*
by (*smt even-iff-mod-2-eq-zero mult-less-cancel-right mult-neg-neg mult-nonneg-nonpos mult-pos-pos not-mod2-eq-Suc-0-eq-0 power-0-Suc x-def zero-le-power-eq zero-less-mult-pos2 zero-less-power*)
qed
qed
next
case (*Neq p*)
obtain *q i* **where** *h*: *decreasePower v p* = (*q*, *i*)
using *prod.exhaust-sel* **by** *blast*
have *p* : ⋀*f*. *insertion f p* = *insertion f q*∗ (*f v*) ^ *i*
using *decreasePower-insertion*[*OF h*] **by** *auto*
show *?thesis* **unfolding** *Neq unpower.simps h* **apply** *simp* **apply**(*cases i*)
apply *simp*
apply *simp* **unfolding** *p* **apply** *simp*
by (*metis insert-Var-Zero*)
qed
qed *auto*

lemma *to-list-filter*: *p* = *sum-list* [*term* ∗ (*Var v*) ^ *i*. (*term*,*i*)←((*filter* (λ(*x*,-). *x*≠*0*) (*to-list v p*)))]
proof−
define *L* **where** *L* = *to-list v p*
have (∑ (*term*, *i*)←*to-list v p*. *term* ∗ *Var v* ^ *i*) = (∑ (*term*, *i*)←*filter* (λ(*x*, -). *x* ≠ *0*) (*to-list v p*). *term* ∗ *Var v* ^ *i*)
unfolding *L-def*[*symmetric*] **apply**(*induction L*) **by** *auto*
then show *?thesis*
using *to-list-p*[*of p v*] **by** *auto*
qed

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*
∑*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 = ∑*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 = ∑*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*
∑*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*
∑*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* (λ*x*.λ*A*. *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* (λ*F*.λ*i*.
let a-n = *isolate-variable-sparse p var i in*
let exp = (*if i mod 2* = *0 then Const*(*1*) *else Const*(−*1*)) *in*
or (*Atom*(*Less* (*exp* ∗ *a-n*)))
(*and* (*Atom* (*Eq a-n*)) *F*)
) *FalseF* ([*0*..<((*MPoly-Type.degree p var*)+*1*)])

fun *substNegInfinity* :: *nat* ⇒ *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*))))
by *pat-completeness auto*
termination
apply(*relation measures* [λ(*var*,*p*). *MPoly-Type.degree p var*])
apply *auto*
using *degree-derivative*
by (*metis less-add-one*)

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 (λx.λA. substInfinitesimalLinear
(var+x) (liftPoly 0 x b) (liftPoly 0 x c) A) F 0


fun substInfinitesimalQuadratic-fm :: nat ⇒ real mpoly ⇒ real mpoly ⇒ real mpoly
⇒ real mpoly ⇒ atom fm ⇒ atom fm where
  substInfinitesimalQuadratic-fm var a b c d F = liftmap (λx.λA. substInfinitesi-
malQuadratic (var+x) (liftPoly 0 x a) (liftPoly 0 x b) (liftPoly 0 x c) (liftPoly 0 x
d) A) F 0
```

7.3 VS Algorithms

elimVar var L F attempts to do quadratic elimination on the variable defined by var. L is the list of 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 (λa. Atom (linear-substitution var (−c) b a)) L)@
        (map (linear-substitution-fm var (−c) b) F)
        )))


      (And (Atom (Neq a)) (And (Atom(Leq (−(b^2)+4∗a∗c)))
        (Or (list-conj (
        (map (quadratic-sub var (−b) 1 (b^2−4∗a∗c) (2∗a)) L)@
        (map (quadratic-sub-fm var (−b) 1 (b^2−4∗a∗c) (2∗a)) F)
        ))
        (list-conj (
        (map (quadratic-sub var (−b) (−1) (b^2−4∗a∗c) (2∗a)) L)@
        (map (quadratic-sub-fm var (−b) (−1) (b^2−4∗a∗c) (2∗a)) F)
        ))
       ))
      ))

) |
  elimVar var L F (Less p) =  (
  let (a,b,c) = get-coeffs var p in
    (Or

      (And (And (Atom (Eq a)) (Atom (Neq b)))
      (list-conj (
          (map (substInfinitesimalLinear var (−c) b) L)
          @(map (substInfinitesimalLinear-fm var (−c) b) F)
      )))


      (And (Atom (Neq a)) (And (Atom(Leq (−(b^2)+4∗a∗c)))
        (Or (list-conj (
        (map (substInfinitesimalQuadratic var (−b) 1 (b^2−4∗a∗c) (2∗a)) L)@
        (map (substInfinitesimalQuadratic-fm var (−b) 1 (b^2−4∗a∗c) (2∗a)) F)
        ))
        (list-conj (
       (map (substInfinitesimalQuadratic var (−b) (−1) (b^2−4∗a∗c) (2∗a)) L)@
        (map (substInfinitesimalQuadratic-fm var (−b) (−1) (b^2−4∗a∗c) (2∗a))
F)
        ))
       ))
      ))
)|
  elimVar var L F (Neq p) =  (
  let (a,b,c) = get-coeffs var p in
    (Or

      (And (And (Atom (Eq a)) (Atom (Neq b)))
```

```
      (list-conj (
          (map (substInfinitesimalLinear var (−c) b) L)
          @(map (substInfinitesimalLinear-fm var (−c) b) F)
      )))


      (And (Atom (Neq a)) (And (Atom(Leq (−(b^2)+4∗a∗c)))
        (Or (list-conj (
        (map (substInfinitesimalQuadratic var (−b) 1 (b^2−4∗a∗c) (2∗a)) L)@
        (map (substInfinitesimalQuadratic-fm var (−b) 1 (b^2−4∗a∗c) (2∗a)) F)
        ))
        (list-conj (
       (map (substInfinitesimalQuadratic var (−b) (−1) (b^2−4∗a∗c) (2∗a)) L)@
        (map (substInfinitesimalQuadratic-fm var (−b) (−1) (b^2−4∗a∗c) (2∗a))
F)
        ))
       ))
      )))
|
  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 optionwhere
  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 φ1 φ2) = [(A@B,A'@B').(A,A')←dnf φ1,(B,B')←dnf φ2] |
dnf (Or φ1 φ2) = dnf φ1 @ dnf φ2 |
dnf (ExQ φ) = [([],[ExQ φ])] |
dnf (Neg φ) = [([],[Neg φ])]|
dnf (AllQ φ) = [([],[AllQ φ])]|
dnf (AllN i φ) = [([],[AllN i φ])]|
dnf (ExN i φ) = [([],[ExN i φ])]
```

dnf F returns the "disjunctive normal form" of F, but since F can contain quantifiers, we return (L,R,n) terms in a list. each term in the list represents a conjunction over the outside disjunctive list

L is all the atoms we are able to reach, we are allowed to go underneath exists binders

R is the remaining formulas (negation exists cannot be simplified) which are also under the same number of exist binders.

n is the total number of binders each conjunct has

```
fun dnf-modified :: atom fm ⇒ (atom list * atom fm list * nat) list where
  dnf-modified TrueF = [([],[],0)] |
  dnf-modified FalseF = [] |
  dnf-modified (Atom φ) = [([φ],[],0)] |
  dnf-modified (And φ1 φ2) = [
  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 φ1, (B,B',d2) ← dnf-modified φ2] |
  dnf-modified (Or φ1 φ2) = dnf-modified φ1 @ dnf-modified φ2 |
  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 φ1 φ2) = and (QE-dnf opt step φ1) (QE-dnf opt step φ2) |
  QE-dnf opt step (Or φ1 φ2) = or (QE-dnf opt step φ1) (QE-dnf opt step φ2) |
  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 φ1 φ2) = and (QE-dnf' opt step φ1) (QE-dnf' opt step φ2) |
QE-dnf' opt step (Or φ1 φ2) = or (QE-dnf' opt step φ1) (QE-dnf' opt step φ2)
|
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 (λ*A* (*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
  fix x :: natpair
  fix y :: natpair
  fix z :: natpair
  obtain a b where x : x = Pair (a,b) apply(cases x) by auto
  obtain c d where y : y = Pair (c,d) apply(cases y) by auto
  obtain e f where z : z = Pair (e,f) apply(cases z) by auto
  show (x < y) = strict (≤) x y
    unfolding x y by auto
  show x≤x unfolding x by auto
  show x≤ y ⟹ y≤ z ⟹ x≤ z unfolding x y z apply auto
    apply (metis dual-order.trans not-less-iff-gr-or-eq)
    by (metis less-trans)
  show x ≤ y ⟹ y ≤ x ⟹ x = y unfolding x y apply auto
    apply (metis not-less-iff-gr-or-eq)
    by (metis antisym-conv not-less-iff-gr-or-eq)
  show x ≤ y ∨ y ≤ x unfolding x y by auto
qed
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)) optionwhere
  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
```

```
  fix x y z
  obtain a1 b1 c1 d1 where x : x = Quad(a1,b1,c1,d1)
    by (metis prod-cases4 quadnat.exhaust)
  obtain a2 b2 c2 d2 where y : y = Quad(a2,b2,c2,d2)
    by (metis prod-cases4 quadnat.exhaust)
  obtain a3 b3 c3 d3 where z : z = Quad(a3,b3,c3,d3)
    by (metis prod-cases4 quadnat.exhaust)
  show (x < y) = strict (≤) x y unfolding x y by auto
  show x ≤ x unfolding x by auto
  show x ≤ y ⟹ y ≤ z ⟹ x ≤ z unfolding x y z apply auto
    apply (metis dual-order.trans not-less-iff-gr-or-eq)
    apply (metis less-trans)
    apply (metis dual-order.strict-trans not-less-iff-gr-or-eq)
    apply (metis less-trans)
    apply (metis dual-order.strict-trans not-less-iff-gr-or-eq)
    apply (metis less-trans)
    apply (metis less-trans not-less-iff-gr-or-eq)
    by (metis less-trans)
  show x ≤ y ⟹ y ≤ x ⟹ x = y unfolding x y apply auto
    apply (metis less-imp-not-less)
    apply (metis not-less-iff-gr-or-eq)
    apply (metis not-less-iff-gr-or-eq)
    by (metis antisym-conv not-less-iff-gr-or-eq)
  show x ≤ y ∨ y ≤ x unfolding x y by auto
qed
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*
apply *unfold-locales*
subgoal by (*simp add*: *drlex-pm-strict-def*)
subgoal by (*simp add*: *drlex-pm-refl*)
subgoal using *drlex-pm-trans* **by** *auto*
subgoal by (*simp add*: *drlex-pm-antisym*)
subgoal by (*simp add*: *drlex-pm-lin*)
done

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 (λ*s*. *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* ⇒$_0$ *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* ⇒$_0$ *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
  using that
  unfolding rat-of-real-def
  by simp

lemma rat-of-real-code[code]: rat-of-real (Ratreal r) = r
  by (simp add: rat-of-real)

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
  by code-simp
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 ⇒ natwhere
  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))
  by pat-completeness auto
termination
  apply(relation measures [λ(-,A). depth' A])
  by auto


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 Pret-
tyPrinting Show.Show-Real
begin
```

definition *opt* = (*push-forall* ∘ *nnf* ∘ *unpower 0 o clearQuantifiers*)
definition *opt-group* = (*push-forall* ∘ *nnf* ∘ *unpower 0 o groupQuantifiers o clear-Quantifiers*)

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 brown-sHeuristic*)) *o VSLuckiestBlocks o opt-group*
definition *VSGeneral-3-times* = *VSGeneral o VSGeneral o VSGeneral*
definition *VSLucky* = *opt o* (*QE-dnf opt* (λ*amount. 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 VS-LuckiestBlocks 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 (λx.λy. (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* $\notin$ *vars b*
assumes *var* $\notin$ *vars c*
shows *freeIn var* (*Atom* (*linear-substitution var b c A*))
proof(*cases A*)
case (*Less p*) **define** *d* **where** *d* = *MPoly-Type.degree p var*
then show *?thesis* **using** *Less* **apply** *simp* **unfolding** *d-def*[*symmetric*]
apply *simp* **using** *not-in-sum*
using *not-in-isovarspar assms not-in-mult not-in-neg not-in-pow not-in-add*
by (*metis* (*no-types*, *lifting*))
next
case (*Eq p*)
define *d* **where** *d* = *MPoly-Type.degree p var*
then show *?thesis* **using** *Eq* **apply** *simp* **unfolding** *d-def*[*symmetric*]
apply *simp* **using** *not-in-sum*
using *not-in-isovarspar assms not-in-mult not-in-neg not-in-pow not-in-add*
by (*metis* (*no-types*, *lifting*))
next
case (*Leq p*)
define *d* **where** *d* = *MPoly-Type.degree p var*
then show *?thesis* **using** *Leq* **apply** *simp* **unfolding** *d-def*[*symmetric*]
apply *simp* **using** *not-in-sum*
using *not-in-isovarspar assms not-in-mult not-in-neg not-in-pow not-in-add*
by (*metis* (*no-types*, *lifting*))
next
case (*Neq p*)
define *d* **where** *d* = *MPoly-Type.degree p var*
then show *?thesis* **using** *Neq* **apply** *simp* **unfolding** *d-def*[*symmetric*]

apply *simp* **using** *not-in-sum*
using *not-in-isovarspar assms not-in-mult not-in-neg not-in-pow not-in-add*
by (*metis* (*no-types*, *lifting*))
qed

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 −
define *d* **where** *d* = *MPoly-Type.degree p var*
define *f* **where** *f i* = (*insertion* (*nth-default 0* (*list-update L var* (*B*/*C*))) (*isolate-variable-sparse p var i*) ∗ (*B*) ^ *i*::*real*) **for** *i*
have *h* : ((∑ *i* = *0*..<*d*+*1*. *f i* / *C* ^ *i*) = *0*) =((∑ *i* = *0*..<*d*+*1*. (*f i*) ∗ *C* ^ (*d* − *i*)) = *0*)
using *normalize-summation nonzero* **by**(*auto*)
have *aEval* (*linear-substitution var b c* (*Eq*(*p*))) (*list-update L var* (*B*/*C*)) =
aEval (*Eq*(∑ *i* = *0*..<*d*+*1*. *isolate-variable-sparse p var i* ∗ (*b*) ^ *i* ∗ *c* ^ (*d* − *i*))) (*list-update L var* (*B*/*C*))
by (*metis* (*no-types*, *lifting*) *d-def linear-substitution.simps*(*1*) *sum.cong*)
also have ... = ((∑ *i* = *0*..<(*d*+*1*). *insertion* (*nth-default 0* (*list-update L var* (*B*/*C*))) (*isolate-variable-sparse p var i*) ∗ (*B*) ^ *i* ∗ *C* ^ (*d* − *i*)) = *0*)
using *assms* **by**(*simp add*: *insertion-sum insertion-mult insertion-add insertion-pow insertion-neg lLength*)
also have ... = ((∑ *i* = *0*..<(*d*+*1*). *insertion* (*nth-default 0* (*list-update L var* (*B*/*C*))) (*isolate-variable-sparse p var i*) ∗ (*B*) ^ *i*/ (*C* ^ *i*)) = *0*)
using *h* **by**(*simp add*: *f-def*)
also have ... = ((∑ *i* = *0*..<(*d*+*1*). *insertion* (*nth-default 0* (*list-update L var* (*B*/*C*))) (*isolate-variable-sparse p var i*) ∗ ((*B*/*C*) ^ *i*)) = *0*)
by (*metis* (*no-types*, *lifting*) *power-divide sum.cong times-divide-eq-right*)
also have ... = *aEval* (*Eq*(*p* :: *real mpoly*)) (*list-update L var* (*B*/*C*))
using *sum-over-degree-insertion d-def lLength* **by** *auto*
finally show *?thesis* **using** *assms plugInLinear var-not-in-linear var-not-in-eval*
by (*meson var-not-in-aEval*)
qed

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−
define *d* **where** *d* = *MPoly-Type.degree p var*
define *f* **where** *f i* = (*insertion* (*nth-default 0* (*list-update L var* (*B*/*C*))) (*isolate-variable-sparse p var i*) ∗ (*B*) ^ *i*::*real*) **for** *i*
have *h* : (∑ *i* = *0*..<(*d*+*1*). (*f i*) ∗ *C* ^ (*d* − *i*)) ∗ *C* ^ (*d mod 2*) < *0* ⟷ (∑ *i* = *0*..<((*d*::*nat*)+*1*). (*f i*::*real*) / (*C* ^ *i*)) < *0*
using *nonzero normalize-summation-less* **by** *auto*
have *aEval* (*linear-substitution var b c* (*Less*(*p*))) (*list-update L var* (*B*/*C*))=*aEval* (*Less*((∑ *i*∈{*0*..<(*d*+*1*)}. *isolate-variable-sparse p var i* ∗ (*b*^*i*) ∗ (*c*^(*d*−*i*))) ∗ (*c* ^ (*d mod 2*)))) (*list-update L var* (*B*/*C*))
by (*metis* (*no-types*, *lifting*) *d-def linear-substitution.simps*(*2*) *sum.cong*)
also have ... = ((∑ *i* = *0*..<(*d*+*1*). *insertion* (*nth-default 0* (*list-update L var* (*B*/*C*))) (*isolate-variable-sparse p var i*) ∗ (*B*) ^ *i* ∗ *C* ^ (*d* − *i*)) ∗ *C* ^ (*d mod 2*) < *0*)
using *assms* **by**(*simp add*: *insertion-sum insertion-mult insertion-add insertion-pow insertion-neg lLength*)
also have ... = ((∑ *i* = *0*..<(*d*+*1*). *insertion* (*nth-default 0* (*list-update L var* (*B*/*C*))) (*isolate-variable-sparse p var i*) ∗ (((*B*) ^ *i*) / (*C* ^ *i*))) < *0*)
using *f-def h* **by** *auto*
also have ... = ((∑ *i* = *0*..<(*d*+*1*). *insertion* (*nth-default 0* (*list-update L var* (*B*/*C*))) (*isolate-variable-sparse p var i*) ∗ (*B*/*C*)^*i*) < *0*)
by (*metis* (*no-types*, *lifting*) *power-divide sum.cong*)
also have ... = *aEval* (*Less*(*p*)) (*list-update L var* (*B*/*C*))
using *d-def sum-over-degree-insertion lLength* **by** *auto*
finally show *?thesis* **using** *assms plugInLinear var-not-in-linear var-not-in-eval*
by (*meson var-not-in-aEval*)
qed

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 −
define *d* **where** *d = MPoly-Type.degree p var*
define *f* **where** *f i = (insertion (nth-default 0 (list-update L var (B/C))) (isolate-variable-sparse p var i) * (B) ^ i::real)* **for** *i*
have *h1a : ((∑ i = 0..<(d+1). (f i) * C ^ (d − i)) * C ^ (d mod 2) < 0) = ((∑ i = 0..<((d::nat)+1). (f i::real) / (C ^ i)) < 0)*
using *nonzero normalize-summation-less* **by** *auto*
have *((∑ i = 0..<d+1. f i / C ^ i) = 0) =((∑ i = 0..<d+1. (f i) * C ^ (d − i)) = 0)*
using *normalize-summation nonzero* **by**(*auto*)
also have *... =((∑ i = 0..<d+1. (f i) * C ^ (d − i))* C ^ (d mod 2) = 0)*
using *mult-eq-0-iff nonzero power-not-zero* **by** *blast*
finally have *h1 : ((∑ i = 0..<(d+1). (f i) * C ^ (d − i)) * C ^ (d mod 2) ≤ 0) = ((∑ i = 0..<((d::nat)+1). (f i::real) / (C ^ i)) ≤ 0)*
using *h1a* **by** *smt*
have *aEval (linear-substitution var b c (Leq(p))) (list-update L var (B/C))=aEval (Leq((∑ i∈{0..<(d+1)}. isolate-variable-sparse p var i * (b^i) * (c^(d−i))) * (c ^ (d mod 2)))) (list-update L var (B/C))*
by (*metis* (*no-types*, *lifting*) *d-def linear-substitution.simps*(*3*) *sum.cong*)
also have *... = ((∑ i = 0..<(d+1). insertion (nth-default 0 (list-update L var (B/C))) (isolate-variable-sparse p var i) * (B) ^ i * C ^ (d − i)) * C ^ (d mod 2) ≤ 0)*
using *assms* **by**(*simp add*: *insertion-sum insertion-mult insertion-add insertion-pow insertion-neg lLength*)
also have...= *((∑ i = 0..<(d+1). (insertion (nth-default 0 (list-update L var (B/C))) (isolate-variable-sparse p var i) * (B) ^ i) / (C ^ i)) ≤ 0)*
using *h1 f-def* **by** *auto*
also have *... = ((∑ i = 0..<(d+1). insertion (nth-default 0 (list-update L var (B/C))) (isolate-variable-sparse p var i) * (((B) ^ i) / (C ^ i))) ≤ 0)*
by *auto*
also have *... = ((∑ i = 0..<(d+1). insertion (nth-default 0 (list-update L var (B/C))) (isolate-variable-sparse p var i) * (B/C)^i) ≤ 0)*
by (*metis* (*no-types*, *lifting*) *power-divide sum.cong*)
also have *... = aEval (Leq(p)) (list-update L var (B/C))*
using *d-def sum-over-degree-insertion lLength* **by** *auto*
finally show *?thesis* **using** *assms plugInLinear var-not-in-eval var-not-in-linear*
by (*meson var-not-in-aEval*)
qed

lemma *linear-neq* :
assumes *lLength* : *length L > var*
assumes *nonzero* : *C ≠ 0*
assumes *var ∉ vars b*
assumes *var ∉ vars c*
assumes *insertion (nth-default 0 (list-update L var (B/C))) b = (B::real)*
assumes *insertion (nth-default 0 (list-update L var (B/C))) c = (C::real)*

shows *aEval* (*Neq*(*p*)) (*list-update L var* (*B*/*C*)) = (*aEval* (*linear-substitution var b c* (*Neq*(*p*))) (*list-update L var v*))
proof −
define *d* **where** *d* = *MPoly-Type.degree p var*
have *aEval* (*Eq*(*p*)) (*list-update L var* (*B*/*C*)) = (∀ *v*. *aEval* (*linear-substitution var b c* (*Eq*(*p*))) (*list-update L var v*))
using *linear-eq assms* **by** *auto*
also have ... = (∀ *v*. *eval* (*Atom* (*Eq* ((∑ *i* = *0*..<*d*+*1*. *isolate-variable-sparse p var i* ∗ (*b*) ^ *i* ∗ *c* ^ (*d* − *i*))))) (*list-update L var v*))
by (*metis* (*no-types*, *lifting*) *d-def eval.simps*(*1*) *linear-substitution.simps*(*1*) *sum.cong*)
also have ... = (¬(∀ *v*. *eval* (*Atom* (*Neq* ((∑ *i* = *0*..<*d*+*1*. *isolate-variable-sparse p var i* ∗ (*b*) ^ *i* ∗ *c* ^ (*d* − *i*))))) (*list-update L var v*)))
by (*metis* (*no-types*, *lifting*) *aEval.simps*(*1*) *aEval.simps*(*4*) *eval.simps*(*1*) *assms*(*3*) *assms*(*4*) *not-contains-insertion not-in-isovarspar not-in-mult not-in-pow not-in-sum*)
also have ... = (¬(∀ *v*. *aEval* (*linear-substitution var b c* (*Neq*(*p*))) (*list-update L var v*)))
by (*metis* (*full-types*) *d-def eval.simps*(*1*) *linear-substitution.simps*(*4*))
finally have ... = (¬(*aEval* (*Neq*(*p*)) (*list-update L var* (*B*/*C*)))) **by** *simp*
then show *?thesis*
using *assms*(*3*) *assms*(*4*) *var-not-in-aEval var-not-in-linear* **by** *blast*
qed

theorem *linear* :
assumes *lLength* : *length L* > *var*
assumes *C* ≠ *0*
assumes *var* ∉ *vars b*
assumes *var* ∉ *vars c*
assumes *insertion* (*nth-default 0* (*list-update L var* (*B*/*C*))) *b* = (*B*::*real*)
assumes *insertion* (*nth-default 0* (*list-update L var* (*B*/*C*))) *c* = (*C*::*real*)
shows *aEval A* (*list-update L var* (*B*/*C*)) = (*aEval* (*linear-substitution var b c A*) (*list-update L var v*))
apply(*cases A*) **using** *linear-less*[*OF assms*(*1*−*6*)] *linear-eq*[*OF assms*(*1*−*6*)] *linear-leq*[*OF assms*(*1*−*6*)] *linear-neq*[*OF assms*(*1*−*6*)] **by** *auto*

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(*induction F arbitrary*: *z*)
case *TrueF*

then show *?case* **by**(*simp*)
next
case *FalseF*
then show *?case* **by** *simp*
next
case (*Atom x*)
show *?case* **unfolding** *linear-substitution-fm-helper.simps liftmap.simps*
using *var-not-in-linear*[*OF not-in-lift*[*OF assms*(*1*)] *not-in-lift*[*OF assms*(*2*)],
of z] **by** *blast*
next
case (*And F1 F2*)
then show *?case* **by** *simp*
next
case (*Or F1 F2*)
then show *?case* **by** *simp*
next
case (*Neg F*)
then show *?case* **by** *simp*
next
case (*ExQ F*)
show *?case* **using** *ExQ*[*of z+1*] **by** *simp*
next
case (*AllQ F*)
show *?case* **using** *AllQ*[*of z+1*] **by** *simp*
next
case (*ExN x1 φ*)
then show *?case*
by (*metis* (*no-types*, *lifting*) *freeIn.simps*(*13*) *group-cancel.add1 liftmap.simps*(*10*)
linear-substitution-fm-helper.simps)
next
case (*AllN x1 φ*)
then show *?case*
by (*metis* (*no-types*, *lifting*) *freeIn.simps*(*12*) *group-cancel.add1 liftmap.simps*(*9*)
linear-substitution-fm-helper.simps)
qed

theorem *var-not-in-linear-fm* :
assumes *var* ∉ *vars b*
assumes *var* ∉ *vars c*
shows *freeIn var* (*linear-substitution-fm var b c F*)
using *var-not-in-linear-fm-helper*[*OF assms*, *of 0*] **by** *auto*

lemma *linear-fm-helper* :
assumes *C* ≠ *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*))
using *assms* **proof**(*induction F arbitrary*:*z L*)
case *TrueF*
then show *?case* **by** *auto*
next
case *FalseF*
then show *?case* **by** *auto*
next
case (*Atom x*)
define *L1* **where** *L1* = *drop z L*
define *L2* **where** *L2* = *take z L*
have *L-def* : *L* = *L2* @ *L1* **using** *L1-def L2-def* **by** *auto*
have *h1a* : *insertion* (*nth-default 0 L1*) *b* = *B*
using *not-contains-insertion*[*OF Atom*(*2*), *of L1 B*/*C B*] *Atom*(*4*) **unfolding** *L1-def nth-default-def*
by (*metis list-update-id*)
have *lengthl2* : *length L2* = *z* **using** *L2-def*
using *Atom.prems*(*6*) *min.absorb2* **by** *auto*
have (∀ *I amount*.
length I = *amount* ⟶
(∀ *xs*. *eval* (*fm.Atom* (*Eq* (*b* − *Const B*))) ([] @ *xs*) =
eval (*liftFm 0 amount* (*fm.Atom* (*Eq* (*b* − *Const B*)))) ([] @ *I* @ *xs*)))
by (*metis eval-liftFm-helper list.size*(*3*))
then have *eval* (*Atom*(*Eq* (*b*−*Const B*))) ([] @ *L1*) = *eval* (*liftFm 0 z* (*Atom*(*Eq* (*b*− *Const B*)))) ([] @ *L2* @ *L1*)
using *lengthl2* **by** *auto*
then have (*insertion* (*nth-default 0* (*L2* @ *L1*)) (*liftPoly 0 z* (*b* − *Const B*)) = *0*)
apply(*simp add*: *insertion-sub insertion-const*) **using** *h1a* **by** *auto*
then have *insertion* (*nth-default 0* (*L2* @ *L1*)) (*liftPoly 0 z b*) = *B*
using *lift-minus* **by** *blast*
then have *h1* : *insertion* (*nth-default 0* (*L*[*var* + *z* := *B*/*C*])) (*liftPoly 0 z b*) = *B*
using *not-in-lift*[*OF Atom*(*2*), *of z*] *L-def*
by (*metis list-update-id not-contains-insertion*)
have *h2a* : *insertion* (*nth-default 0 L1*) *c* = *C*
using *not-contains-insertion*[*OF Atom*(*3*), *of L1 B*/*C C*] *Atom*(*5*) **unfolding** *L1-def*
by (*metis list-update-id*)
have (∀ *I amount*.
length I = *amount* ⟶
(∀ *xs*. *eval* (*fm.Atom* (*Eq* (*c* − *Const C*))) ([] @ *xs*) =
eval (*liftFm 0 amount* (*fm.Atom* (*Eq* (*c* − *Const C*)))) ([] @ *I* @ *xs*)))
by (*metis eval-liftFm-helper list.size*(*3*))
then have *eval* (*Atom*(*Eq* (*c*−*Const C*))) ([] @ *L1*) = *eval* (*liftFm 0 z* (*Atom*(*Eq* (*c*− *Const C*)))) ([] @ *L2* @ *L1*)
using *lengthl2* **by** *auto*
then have (*insertion* (*nth-default 0* (*L2* @ *L1*)) (*liftPoly 0 z* (*c* − *Const C*)) =

0)
  apply(*simp add*: *insertion-sub insertion-const*) **using** *h2a* **by** *auto*
 then have *insertion* (*nth-default 0* (*L2* @ *L1*)) (*liftPoly 0 z c*) = *C*
  using *lift-minus* **by** *blast*
 then have *h2* : *insertion* (*nth-default 0* (*L*[*var* + *z* := *B*/*C*])) (*liftPoly 0 z c*) = *C*
  using *not-in-lift*[*OF Atom*(*3*), *of z*] *L-def*
  by (*metis list-update-id not-contains-insertion*)
 show *?case* **using** *linear*[*OF Atom*(*6*) *Atom*(*1*) *not-in-lift*[*OF Atom*(*2*)] *not-in-lift*[*OF Atom*(*3*)], *of B*, *of x*, *OF h1 h2*] **unfolding** *linear-substitution-fm-helper.simps liftmap.simps eval.simps*
  .
next
 case (*And F1 F2*)
 then show *?case* **by** *auto*
next
 case (*Or F1 F2*)
 then show *?case* **using** *var-not-in-linear-fm-helper var-not-in-eval* **unfolding** *linear-substitution-fm-helper.simps liftmap.simps eval.simps*
  by *blast*
next
 case (*Neg F*)
 then show *?case* **using** *var-not-in-linear-fm-helper var-not-in-eval* **unfolding** *linear-substitution-fm-helper.simps liftmap.simps eval.simps*
  by *blast*
next
 case (*ExQ F*)
 have *droph* : (*drop* (*z* + *1*) (*x*#*L*)) = (*drop z L*) **for** *x* **by** *auto*
 have *l* : *x* # *L*[*var* + *z* := *v*] = ((*x*#*L*)[*var*+(*z*+*1*):=*v*]) **for** *x v*
  by *auto*
 have *eval* (*ExQ F*) (*L*[*var* + *z* := *B*/*C*]) =
      (∃*x*. *eval F* ((*x* # *L*)[*var* + (*z* + *1*) := *B*/*C*]))
  apply(*simp*) **unfolding** *l* **done**

 also have ... = (∃*x*. *eval*
            (*liftmap* (λ*x*. λ*a*. *Atom*(*linear-substitution* (*var* + *x*) (*liftPoly 0 x b*) (*liftPoly 0 x c*) *a*)) *F* (*z* + *1*))
            ((*x* # *L*)[*var* + (*z* + *1*) := *v*]))
  apply(*rule ex-cong1*)
  using *ExQ*(*1*)[*of z*+*1*, *OF assms*(*1*) *assms*(*2*) *assms*(*3*)] *droph*
  unfolding *linear-substitution-fm-helper.simps liftmap.simps*
   by (*metis* (*mono-tags*, *lifting*) *ExQ.prems*(*4*) *ExQ.prems*(*5*) *ExQ.prems*(*6*) *One-nat-def Suc-eq-plus1 Suc-less-eq add-Suc-right list.size*(*4*))
 also have ... = (*eval* (*linear-substitution-fm-helper var b c* (*ExQ F*) *z*) (*L*[*var* + *z* := *v*]))
   unfolding *linear-substitution-fm-helper.simps liftmap.simps eval.simps l* **by** *simp*
 finally show *?case* **by** *simp*
next

case (*AllQ F*)
have *droph* : (*drop* (*z* + *1*) (*x*#*L*)) = (*drop z L*) **for** *x* **by** *auto*
have *l* : *x* # *L*[*var* + *z* := *v*] = ((*x*#*L*)[*var*+(*z*+*1*):=*v*]) **for** *x v*
by *auto*
have *eval* (*AllQ F*) (*L*[*var* + *z* := *B*/*C*]) =
(∀ *x*. *eval F* ((*x* # *L*)[*var* + (*z* + *1*) := *B*/*C*]))
apply(*simp*) **unfolding** *l* **done**
also have ... = (∀ *x*. *eval*
(*liftmap* (λ*x*.λ*a*. *Atom*(*linear-substitution* (*var* + *x*) (*liftPoly 0 x b*)
(*liftPoly 0 x c*) *a*)) *F* (*z* + *1*))
((*x* # *L*)[*var* + (*z* + *1*) := *v*]))
apply(*rule all-cong1*)
using *AllQ*(*1*)[*of z*+*1*, *OF assms*(*1*) *assms*(*2*) *assms*(*3*)]
var-not-in-linear-fm-helper[*OF assms*(*2*) *assms*(*3*)] *var-not-in-eval droph*
unfolding *linear-substitution-fm-helper.simps liftmap.simps*
by (*metis* (*mono-tags*, *lifting*) *AllQ*(*7*) *AllQ.prems*(*4*) *AllQ.prems*(*5*) *One-nat-def*
Suc-eq-plus1 Suc-less-eq add-Suc-right list.size(*4*))
also have ... = (*eval* (*linear-substitution-fm-helper var b c* (*AllQ F*) *z*) (*L*[*var* +
z := *v*]))
unfolding *linear-substitution-fm-helper.simps liftmap.simps eval.simps l* **by**
auto
finally show *?case* **by** *simp*
next
case (*ExN x1* φ)
have *list* : ⋀*l*. *length l*=*x1* ⟹ ((*drop* (*z* + *x1*) *l* @ *drop* (*z* + *x1* − *length l*)
L)[*var* := *B* / *C*]) = ((*drop z L*)[*var* := *B* / *C*])
by *auto*
have *map* : ⋀ *z L*. *eval* (*liftmap* (λ*x A*. *fm.Atom* (*linear-substitution* (*var* + *x*)
(*liftPoly 0 x b*) (*liftPoly 0 x c*) *A*)) φ (*z* + *x1*))
L = *eval* (*liftmap* (λ*x A*. *fm.Atom* (*linear-substitution* (*var* + *x1* + *x*) (*liftPoly*
0 (*x*+*x1*) *b*) (*liftPoly 0* (*x*+*x1*) *c*) *A*)) φ *z*)
L
apply(*induction* φ) **apply**(*simp-all add*:*add.commute add.left-commute*)
apply *force*
apply *force*
by (*metis* (*mono-tags*, *lifting*) *ab-semigroup-add-class.add-ac*(*1*))+
show *?case*
apply *simp* **apply**(*rule ex-cong1*)
subgoal for *l*
using *map*[*of z*] *ExN*(*1*)[*OF ExN*(*2*−*4*), *of z*+*x1 l*@*L*] *ExN*(*5*−*7*) *list*
apply *simp*
by (*smt* (*z3*) *add.commute add.left-commute add-diff-cancel-left' add-mono-thms-linordered-field*(*4*)
list list-update-append not-add-less1 order-refl)
done
next
case (*AllN x1* φ)
have *list* : ⋀*l*. *length l*=*x1* ⟹ ((*drop* (*z* + *x1*) *l* @ *drop* (*z* + *x1* − *length l*)
L)[*var* := *B* / *C*]) = ((*drop z L*)[*var* := *B* / *C*])
by *auto*

```
  have map : ⋀ z L. eval (liftmap (λx A. fm.Atom (linear-substitution (var + x)
(liftPoly 0 x b) (liftPoly 0 x c) A)) φ (z + x1))
     L = eval (liftmap (λx A. fm.Atom (linear-substitution (var + x1 + x) (liftPoly
0 (x+x1) b) (liftPoly 0 (x+x1) c) A)) φ z)
      L
    apply(induction φ) apply(simp-all add:add.commute add.left-commute)
    apply force
    apply force
    by (metis (mono-tags, lifting) ab-semigroup-add-class.add-ac(1))+
  show ?case
    apply simp apply(rule all-cong1)
    subgoal for l
      using map[of z] AllN(1)[OF AllN(2−4), of z+x1 l@L] AllN(5−7) list
      apply simp
     by (smt (z3) add.commute add.left-commute add-diff-cancel-left' add-mono-thms-linordered-field(4)
list list-update-append not-add-less1 order-refl)
    done
qed

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))
  unfolding linear-substitution-fm.simps using linear-fm-helper[OF assms(2) assms(3)
assms(4), of 0 L B] assms(1) assms(5) assms(6)
  by (simp add: lLength)
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 −
define f **where** $f\,i = insertion\ (nth\text{-}default\ 0\ (list\text{-}update\ L\ var\ C))\ (isolate\text{-}variable\text{-}sparse\ p\ var\ i) * ((A + B * C)\ \hat{}\ i)$ **for** i
have $h1 : \forall i.\ (insertion\ (nth\text{-}default\ 0\ (list\text{-}update\ L\ var\ C))\ (isolate\text{-}variable\text{-}sparse\ p\ var\ i)) = (insertion\ (nth\text{-}default\ 0\ (list\text{-}update\ L\ var\ ((A+B*C)/D)))\ (isolate\text{-}variable\text{-}sparse\ p\ var\ i))$
by(*simp add*: *insertion-isovarspars-free*)
have $h2 : ((\sum i = 0..<deg+1.\ f\ i\ /\ D\ \hat{}\ i) = 0) = ((\sum i = 0..<deg+1.\ (f\ i) * D\ \hat{}\ (deg - i)) = 0)$
using *normalize-summation nonzero* **by**(*auto*)
have $aEval\ (Eq(quadratic\text{-}part\text{-}1\ var\ a\ b\ d\ (Eq\ p)))\ (list\text{-}update\ L\ var\ C) = ((\sum i = 0..<deg+1.\ (insertion\ (nth\text{-}default\ 0\ (list\text{-}update\ L\ var\ C))\ (isolate\text{-}variable\text{-}sparse\ p\ var\ i) * ((A + B * C)\ \hat{}\ i)) * D\ \hat{}\ (deg - i)) = 0)$
by(*simp add*: *hdeg insertion-sum insertion-add insertion-mult insertion-var insertion-pow ha hb hd lLength*)
also have ... $= ((\sum i = 0..<deg+1.\ (insertion\ (nth\text{-}default\ 0\ (list\text{-}update\ L\ var\ C))\ (isolate\text{-}variable\text{-}sparse\ p\ var\ i) * ((A + B * C)\ \hat{}\ i)) / D\ \hat{}\ i) = 0)$
using *f-def h2* **by** *auto*
also have ... $= ((\sum i = 0..<deg+1.\ (insertion\ (nth\text{-}default\ 0\ (list\text{-}update\ L\ var\ C))\ (isolate\text{-}variable\text{-}sparse\ p\ var\ i) * ((A + B * C)\hat{}i\ /\ (D\ \hat{}\ i)))) = 0)$
by *auto*
also have ... $= ((\sum i = 0..<deg+1.\ (insertion\ (nth\text{-}default\ 0\ (list\text{-}update\ L\ var\ C))\ (isolate\text{-}variable\text{-}sparse\ p\ var\ i) * ((A + B * C)/D)\ \hat{}\ i)) = 0)$
by (*metis* (*no-types*, *lifting*) *power-divide sum.cong*)
also have ... $= ((\sum i = 0..<deg+1.\ (insertion\ (nth\text{-}default\ 0\ (list\text{-}update\ L\ var\ ((A+B*C)/D)))\ (isolate\text{-}variable\text{-}sparse\ p\ var\ i) * ((A + B * C)/D)\ \hat{}\ i))=0)$
using *h1* **by** *auto*
also have ... $= (insertion\ (nth\text{-}default\ 0\ (list\text{-}update\ L\ var\ ((A+B*C)/D)))\ p = 0)$
using *sum-over-degree-insertion hdeg lLength* **by** *auto*
also have ... $= aEval\ (Eq\ p)\ (list\text{-}update\ L\ var\ ((A+B*C)/D))$
using *aEval.simps*(*1*) **by** *blast*
finally show *?thesis* **using** *assms* **by** *auto*
qed

lemma *quad-part-1-less* :
assumes *lLength* : $length\ L > var$
assumes *hdeg* : $MPoly\text{-}Type.degree\ (p::real\ mpoly)\ var = (deg::nat)$
assumes *nonzero* : $D \neq 0$
assumes *ha* : $\forall x.\ insertion\ (nth\text{-}default\ 0\ (list\text{-}update\ L\ var\ x))\ a = (A::real)$
assumes *hb* : $\forall x.\ insertion\ (nth\text{-}default\ 0\ (list\text{-}update\ L\ var\ x))\ b = (B::real)$
assumes *hd* : $\forall x.\ insertion\ (nth\text{-}default\ 0\ (list\text{-}update\ L\ var\ x))\ d = (D::real)$
shows $aEval\ (Less\ p)\ (list\text{-}update\ L\ var\ ((A+B*C)/D)) = aEval\ (Less(quadratic\text{-}part\text{-}1\ var\ a\ b\ d\ (Less\ p)))\ (list\text{-}update\ L\ var\ C)$
proof −
define f **where** $f\,i = insertion\ (nth\text{-}default\ 0\ (list\text{-}update\ L\ var\ C))\ (isolate\text{-}variable\text{-}sparse\ p\ var\ i) * ((A + B * C)\ \hat{}\ i)$ **for** i

have *h1a* : $((\sum i = 0..<deg+1.\ f\ i\ /\ D\ \hat{}\ i) < 0) = ((\sum i = 0..<deg+1.\ (f\ i) * D\ \hat{}\ (deg - i))\ * D\ \hat{}\ (deg\ mod\ 2) < 0)$
using *normalize-summation-less nonzero* **by**(*auto*)
have *h4a* : $\forall i.$ (*insertion* (*nth-default 0* (*list-update L var C*)) (*isolate-variable-sparse p var i*)) = (*insertion* (*nth-default 0* (*list-update L var* ((*A*+*B*∗*C*)/*D*))) (*isolate-variable-sparse p var i*))
by(*simp add*: *insertion-isovarspars-free*)
have $((\sum i = 0..<deg+1.$ (*insertion* (*nth-default 0* (*list-update L var C*)) (*isolate-variable-sparse p var i*) ∗ ((*A* + *B* ∗ *C*) ^ *i*)) ∗ *D* ^ (*deg* − *i*)) ∗ *D* ^ (*deg mod 2*) < 0)
$= ((\sum i = 0..<deg+1.$ (*insertion* (*nth-default 0* (*list-update L var C*)) (*isolate-variable-sparse p var i*) ∗ ((*A* + *B* ∗ *C*) ^ *i*)) / *D* ^ *i*) < 0)
using *h1a f-def* **by** *auto*
also have ...$= ((\sum i = 0..<deg+1.$ (*insertion* (*nth-default 0* (*list-update L var C*)) (*isolate-variable-sparse p var i*) ∗ ((*A* + *B* ∗ *C*)^*i* / (*D* ^ *i*)))) < 0)
by *auto*
also have ...$= ((\sum i = 0..<deg+1.$ (*insertion* (*nth-default 0* (*list-update L var C*)) (*isolate-variable-sparse p var i*) ∗ ((*A* + *B* ∗ *C*)/*D*) ^ *i*)) < 0)
by (*metis* (*no-types*, *lifting*) *power-divide sum.cong*)
also have ... $= ((\sum i = 0..<deg+1.$ (*insertion* (*nth-default 0* (*list-update L var* ((*A*+*B*∗*C*)/*D*))) (*isolate-variable-sparse p var i*) ∗ ((*A* + *B* ∗ *C*)/*D*) ^ *i*))<0)
using *h4a*
by *auto*
also have ... = (*insertion* (*nth-default 0* (*list-update L var* ((*A*+*B*∗*C*)/*D*))) *p* <0)
using *sum-over-degree-insertion hdeg lLength* **by** *auto*
finally show *?thesis*
by(*simp add*: *hdeg lLength insertion-add insertion-mult ha hb hd insertion-sum insertion-pow insertion-var*)
qed

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* : $\forall x.$ *insertion* (*nth-default 0* (*list-update L var x*)) *a* = (*A*::*real*)
assumes *hb* : $\forall x.$ *insertion* (*nth-default 0* (*list-update L var x*)) *b* = (*B*::*real*)
assumes *hd* : $\forall x.$ *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 −
define *f* **where** *f i* = *insertion* (*nth-default 0* (*list-update L var C*)) (*isolate-variable-sparse p var i*) ∗ ((*A* + *B* ∗ *C*) ^ *i*) **for** *i*
have *h1a* : $((\sum i = 0..<deg+1.\ f\ i\ /\ D\ \hat{}\ i) < 0) = ((\sum i = 0..<deg+1.\ (f\ i) * D\ \hat{}\ (deg - i))\ * D\ \hat{}\ (deg\ mod\ 2) < 0)$
using *normalize-summation-less nonzero* **by**(*auto*)
have *h1b* : $((\sum i = 0..<deg+1.\ f\ i\ /\ D\ \hat{}\ i) = 0) = ((\sum i = 0..<deg+1.\ (f\ i) * D\ \hat{}\ (deg - i)) = 0)$
using *normalize-summation nonzero* **by**(*auto*)

have *h1c* : $((\sum i = 0..<deg+1.\ f\ i\ /\ D\ \hat{}\ i) \leq 0) = ((\sum i = 0..<deg+1.\ (f\ i) * D\ \hat{}\ (deg - i))\ * D\ \hat{}\ (deg\ mod\ 2) \leq 0)$
using *h1a h1b nonzero* **by** *auto*
have *h4a* : $\forall i.$ (*insertion* (*nth-default 0* (*list-update L var C*)) (*isolate-variable-sparse p var i*)) = (*insertion* (*nth-default 0* (*list-update L var* ((*A*+*B*∗*C*)/*D*))) (*isolate-variable-sparse p var i*))
by(*simp add*: *insertion-isovarspars-free*)
have $((\sum i = 0..<deg+1.$ (*insertion* (*nth-default 0* (*list-update L var C*)) (*isolate-variable-sparse p var i*) ∗ ((*A* + *B* ∗ *C*) $\hat{}$ *i*)) ∗ *D* $\hat{}$ (*deg* − *i*)) ∗ *D* $\hat{}$ (*deg mod 2*) ≤ 0)= $((\sum i = 0..<deg+1.$ (*insertion* (*nth-default 0* (*list-update L var C*)) (*isolate-variable-sparse p var i*) ∗ ((*A* + *B* ∗ *C*) $\hat{}$ *i*)) / *D* $\hat{}$ *i*) ≤ 0)
using *h1c f-def* **by** *auto*
also have ...=$((\sum i = 0..<deg+1.$ (*insertion* (*nth-default 0* (*list-update L var C*)) (*isolate-variable-sparse p var i*) ∗ ((*A* + *B* ∗ *C*)$\hat{}$*i* / (*D* $\hat{}$ *i*)))) ≤ 0)
by *auto*
also have ...=$((\sum i = 0..<deg+1.$ (*insertion* (*nth-default 0* (*list-update L var C*)) (*isolate-variable-sparse p var i*) ∗ ((*A* + *B* ∗ *C*)/*D*) $\hat{}$ *i*)) ≤ 0)
by (*metis* (*no-types*, *lifting*) *power-divide sum.cong*)
also have ...=$((\sum i = 0..<deg+1.$ (*insertion* (*nth-default 0* (*list-update L var* ((*A*+*B*∗*C*)/*D*))) (*isolate-variable-sparse p var i*) ∗ ((*A* + *B* ∗ *C*)/*D*) $\hat{}$ *i*))≤0)
using *h4a* **by** *auto*
also have ... = (*insertion* (*nth-default 0* (*list-update L var* ((*A*+*B*∗*C*)/*D*))) *p*≤0)
using *sum-over-degree-insertion hdeg lLength* **by** *auto*
finally show *?thesis*
by(*simp add*: *hdeg lLength insertion-add insertion-mult ha hb hd insertion-sum insertion-pow insertion-var*)
qed

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* : $\forall x.$ *insertion* (*nth-default 0* (*list-update L var x*)) *a* = (*A*::*real*)
assumes *hb* : $\forall x.$ *insertion* (*nth-default 0* (*list-update L var x*)) *b* = (*B*::*real*)
assumes *hd* : $\forall x.$ *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 −
have *aEval* (*Eq*(*quadratic-part-1 var a b d* (*Eq p*))) (*list-update L var C*) = *aEval* (*Eq p*) (*list-update L var* ((*A*+*B*∗*C*)/*D*))
using *quad-part-1-eq assms* **by** *blast*
then show *?thesis* **by** *auto*
qed

lemma *sqrt-case* :

assumes *detGreater0* : $SQ \geq 0$
shows $((SQ\hat{}(i\ div\ 2)) * real\ (i\ mod\ 2) * sqrt\ SQ + SQ\ \hat{}\ (i\ div\ 2) * (1 - real\ (i\ mod\ 2))) = (sqrt\ SQ)\ \hat{}\ i$
proof −
have *h1* : $i\ mod\ 2 = 0 \vee (odd\ i \wedge (i\ mod\ 2 = 1))$
by *auto*
have *h2* : $i\ mod\ 2 = 0 \Longrightarrow ((SQ\hat{}(i\ div\ 2)) * real\ (i\ mod\ 2) * sqrt\ SQ + SQ\ \hat{}\ (i\ div\ 2) * (1 - real\ (i\ mod\ 2))) = (sqrt\ SQ)\ \hat{}\ i$
using *detGreater0* **apply** *auto*
by (*simp add*: *real-sqrt-power-even*)
have *h3* : $(odd\ i \wedge (i\ mod\ 2 = 1)) \Longrightarrow ((SQ\hat{}(i\ div\ 2)) * real\ (i\ mod\ 2) * sqrt\ SQ + SQ\ \hat{}\ (i\ div\ 2) * (1 - real\ (i\ mod\ 2))) = (sqrt\ SQ)\ \hat{}\ i$
using *detGreater0* **apply** *auto*
by (*smt One-nat-def add-Suc-right mult.commute nat-arith.rule0 odd-two-times-div-two-succ power.simps*(2) *power-mult real-sqrt-pow2*)
show *?thesis*
using *h1 h2 h3*
by *linarith*
qed

lemma *sum-over-sqrt* :
assumes *detGreater0* : $SQ \geq 0$
shows $(\sum i{\in}\{0..<n+1\}.\ ((f\ i::real) * (SQ\hat{}(i\ div\ 2)) * real\ (i\ mod\ 2) * sqrt\ SQ +f\ i * SQ\ \hat{}\ (i\ div\ 2) * (1 - real\ (i\ mod\ 2))))$
$= (\sum i{\in}\{0..<n+1\}.\ ((f\ i::real) * ((sqrt\ SQ)\hat{}i)))$
apply (*intro sum.cong refl*)
using *sqrt-case*[*OF detGreater0*] *distrib-left mult.assoc* **by** *metis*

lemma *quad-part-2-eq* :
assumes *lLength* : *length L* > *var*
assumes *detGreater0* : $SQ{\geq}0$
assumes *hdeg* : *MPoly-Type.degree* (*p*::*real mpoly*) *var* = (*deg* ::*nat*)
assumes *hsq* : $\forall x.$ *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 −
define *f* **where** *f i* = *insertion* (*nth-default 0* (*list-update L var* (*sqrt SQ*))) (*isolate-variable-sparse p var i*) **for** *i*
have *h1a* : $(\sum i{\in}\{0..<deg+1\}.\ (f\ i * (SQ\hat{}(i\ div\ 2)) * real\ (i\ mod\ 2) * sqrt\ SQ +f\ i * SQ\ \hat{}\ (i\ div\ 2) * (1 - real\ (i\ mod\ 2))))$
$= (\sum i{\in}\{0..<deg+1\}.\ (f\ i * ((sqrt\ SQ)\hat{}i)))$
using *sum-over-sqrt detGreater0* **by** *auto*
have $(\sum i{\in}\{0..<deg+1\}.\ (insertion\ (nth\text{-}default\ 0\ (list\text{-}update\ L\ var\ (sqrt\ SQ)))\ (isolate\text{-}variable\text{-}sparse\ p\ var\ i) * (SQ\hat{}(i\ div\ 2)) * real\ (i\ mod\ 2) * sqrt\ SQ + (insertion\ (nth\text{-}default\ 0\ (list\text{-}update\ L\ var\ (sqrt\ SQ)))\ (isolate\text{-}variable\text{-}sparse\ p\ var\ i)) * SQ\ \hat{}\ (i\ div\ 2) * (1 - real\ (i\ mod\ 2))))$
$= (\sum i{\in}\{0..<deg+1\}.\ (insertion\ (nth\text{-}default\ 0\ (list\text{-}update\ L\ var\ (sqrt\ SQ)))\ (isolate\text{-}variable\text{-}sparse\ p\ var\ i) * ((sqrt\ SQ)\hat{}i)))$
using *h1a f-def* **by** *auto*

also have ... = *insertion* (*nth-default 0* (*list-update L var* (*sqrt SQ*))) *p*
using *sum-over-degree-insertion hdeg lLength* **by** *auto*
finally show *?thesis*
by(*simp add*:*hdeg hsq insertion-add insertion-sum insertion-mult insertion-pow insertion-var insertion-const lLength*)
qed

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 −
define *f* **where** *f i* = *insertion* (*nth-default 0* (*list-update L var* (*sqrt SQ*))) (*isolate-variable-sparse p var i*) **for** *i*
have *h1a* : (∑ *i*∈{*0*..<*deg*+*1*}. (*f i* ∗ (*SQ*^(*i div 2*)) ∗ *real* (*i mod 2*) ∗ *sqrt SQ* +*f i* ∗ *SQ* ^ (*i div 2*) ∗ (*1* − *real* (*i mod 2*))))
=(∑ *i*∈{*0*..<*deg*+*1*}. (*f i* ∗ ((*sqrt SQ*)^*i*)))
using *sum-over-sqrt detGreater0* **by** *auto*
have (∑ *i*∈{*0*..<*deg*+*1*}. (*insertion* (*nth-default 0* (*list-update L var* (*sqrt SQ*))) (*isolate-variable-sparse p var i*) ∗ (*SQ*^(*i div 2*)) ∗ *real* (*i mod 2*) ∗ *sqrt SQ* + (*insertion* (*nth-default 0* (*list-update L var* (*sqrt SQ*))) (*isolate-variable-sparse p var i*)) ∗ *SQ* ^ (*i div 2*) ∗ (*1* − *real* (*i mod 2*))))
=(∑ *i*∈{*0*..<*deg*+*1*}. (*insertion* (*nth-default 0* (*list-update L var* (*sqrt SQ*))) (*isolate-variable-sparse p var i*) ∗ ((*sqrt SQ*)^*i*)))
using *h1a f-def* **by** *auto*
also have ... = *insertion* (*nth-default 0* (*list-update L var* (*sqrt SQ*))) *p*
using *sum-over-degree-insertion hdeg lLength* **by** *auto*
finally show *?thesis*
by(*simp add*:*hdeg hsq insertion-add insertion-sum insertion-mult insertion-pow insertion-var insertion-const lLength*)
qed

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 −
define *f* **where** *f i* = *insertion* (*nth-default 0* (*list-update L var* (*sqrt SQ*))) (*isolate-variable-sparse p var i*) **for** *i*
have *h1a* : (∑ *i*∈{*0*..<*deg*+*1*}. (*f i* ∗ (*SQ*^(*i div 2*)) ∗ *real* (*i mod 2*) ∗ *sqrt SQ* +*f i* ∗ *SQ* ^ (*i div 2*) ∗ (*1* − *real* (*i mod 2*))))
=(∑ *i*∈{*0*..<*deg*+*1*}. (*f i* ∗ ((*sqrt SQ*)^*i*)))
using *sum-over-sqrt detGreater0* **by** *auto*

have ($\sum i{\in}\{0..{<}deg{+}1\}$. (*insertion* (*nth-default 0* (*list-update L var* (*sqrt SQ*))) (*isolate-variable-sparse p var i*) $*$ (*SQ*^(*i div 2*)) $*$ *real* (*i mod 2*) $*$ *sqrt SQ* + (*insertion* (*nth-default 0* (*list-update L var* (*sqrt SQ*))) (*isolate-variable-sparse p var i*)) $*$ *SQ* ^ (*i div 2*) $*$ (*1* $-$ *real* (*i mod 2*))))
$=(\sum i{\in}\{0..{<}deg{+}1\}$. (*insertion* (*nth-default 0* (*list-update L var* (*sqrt SQ*))) (*isolate-variable-sparse p var i*) $*$ ((*sqrt SQ*)^*i*)))
using *h1a f-def* **by** *auto*
also have ... = *insertion* (*nth-default 0* (*list-update L var* (*sqrt SQ*))) *p*
using *sum-over-degree-insertion hdeg lLength* **by** *auto*
finally show *?thesis*
by(*simp add*:*hdeg hsq insertion-add insertion-sum insertion-mult insertion-pow insertion-var insertion-const lLength*)
qed

lemma *quad-part-2-leq* :
assumes *lLength* : *length L* $>$ *var*
assumes *detGreater0* : *SQ*$\geq$*0*
assumes *hdeg* : *MPoly-Type.degree* (*p*::*real mpoly*) *var* = (*deg* ::*nat*)
assumes *hsq* : $\forall$ *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 $-$
define *f* **where** *f i* = *insertion* (*nth-default 0* (*list-update L var* (*sqrt SQ*))) (*isolate-variable-sparse p var i*) **for** *i*
have *h1a* : ($\sum i{\in}\{0..{<}deg{+}1\}$. (*f i* $*$ (*SQ*^(*i div 2*)) $*$ *real* (*i mod 2*) $*$ *sqrt SQ* +*f i* $*$ *SQ* ^ (*i div 2*) $*$ (*1* $-$ *real* (*i mod 2*))))
$=(\sum i{\in}\{0..{<}deg{+}1\}$. (*f i* $*$ ((*sqrt SQ*)^*i*)))
using *sum-over-sqrt detGreater0* **by** *auto*
have ($\sum i{\in}\{0..{<}deg{+}1\}$. (*insertion* (*nth-default 0* (*list-update L var* (*sqrt SQ*))) (*isolate-variable-sparse p var i*) $*$ (*SQ*^(*i div 2*)) $*$ *real* (*i mod 2*) $*$ *sqrt SQ* + (*insertion* (*nth-default 0* (*list-update L var* (*sqrt SQ*))) (*isolate-variable-sparse p var i*)) $*$ *SQ* ^ (*i div 2*) $*$ (*1* $-$ *real* (*i mod 2*))))
$=(\sum i{\in}\{0..{<}deg{+}1\}$. (*insertion* (*nth-default 0* (*list-update L var* (*sqrt SQ*))) (*isolate-variable-sparse p var i*) $*$ ((*sqrt SQ*)^*i*)))
using *h1a f-def* **by** *auto*
also have ... = *insertion* (*nth-default 0* (*list-update L var* (*sqrt SQ*))) *p*
using *sum-over-degree-insertion hdeg lLength* **by** *auto*
finally show *?thesis*
by(*simp add*:*hdeg hsq insertion-add insertion-sum insertion-mult insertion-pow insertion-var insertion-const lLength*)
qed

lemma *quad-part-2-deg* :
assumes *sqfree* : (*var*::*nat*)$\notin$*vars*(*sq*::*real mpoly*)
shows *MPoly-Type.degree* (*quadratic-part-2 var sq p*) *var* $\leq$ *1*
proof $-$
define *deg* **where** *deg* = *MPoly-Type.degree* (*p*::*real mpoly*) *var*
define *f* **where** *f i* = *isolate-variable-sparse p var i* $*$ *sq* ^ (*i div 2*) $*$ *Const* (*real* (*i mod 2*)) $*$ *Var var* **for** *i*

define g **where** $g\ i$ = *isolate-variable-sparse p var i* $*$ *sq* $\hat{}$ (*i div 2*) $*$ *Const* (*1* $-$ *real* (*i mod 2*)) **for** i
have *h1a* : $\forall$ *i*. *MPoly-Type.degree* (*isolate-variable-sparse p var i*) *var* = *0*
by (*simp add*: *varNotIn-degree not-in-isovarspar*)
have *h1b* : $\forall$ *i*. *MPoly-Type.degree* (*sq* $\hat{}$ (*i div 2*)) *var* = *0*
by (*simp add*: *sqfree varNotIn-degree not-in-pow*)
have *h1c* : $\forall$ *i*. *MPoly-Type.degree* (*Const* (*real* (*i mod 2*))) *var* = *0*
using *degree-const* **by** *blast*
have *h1d* : *MPoly-Type.degree* (*Var var* :: *real mpoly*) *var* = *1*
using *degree-one* **by** *auto*
have *h1* : $\forall$ *i*<*deg*+*1*. *MPoly-Type.degree* (*f i*) *var* $\leq$ *1*
using *f-def degree-mult h1a h1b h1c h1d*
by (*smt ExecutiblePolyProps.degree-one add.right-neutral mult.commute mult-eq-0-iff nat-le-linear not-one-le-zero*)
have *h2a* : $\forall$ *i*. *MPoly-Type.degree* (*Const* (*1* $-$ *real* (*i mod 2*))) *var* = *0*
using *degree-const* **by** *blast*
have *h2* : $\forall$ *i*<*deg*+*1*. *MPoly-Type.degree* (*g i*) *var* = *0*
using *g-def degree-mult h1a h1b h2a*
by (*metis* (*no-types*, *lifting*) *add.right-neutral mult-eq-0-iff*)
have *h3* : $\forall$ *i*<*deg*+*1*. *MPoly-Type.degree* (*f i* + *g i*) *var* $\leq$ *1*
using *h1 h2* **by** (*simp add*: *degree-add-leq*)
show *?thesis* **using** *atLeastLessThanSuc-atLeastAtMost degree-sum f-def g-def h3 deg-def* **by** *auto*
qed

lemma *quad-equality-helper* :
assumes *lLength* : *length L* > *var*
assumes *detGreat0* : *Cv*$\geq$*0*
assumes *hC* : $\forall$ *x*. *insertion* (*nth-default 0* (*list-update L var x*)) (*C*::*real mpoly*) = (*Cv*::*real*)
assumes *hA* : $\forall$ *x*. *insertion* (*nth-default 0* (*list-update L var x*)) (*A*::*real mpoly*) = (*Av*::*real*)
assumes *hB* : $\forall$ *x*. *insertion* (*nth-default 0* (*list-update L var x*)) (*B*::*real mpoly*) = (*Bv*::*real*)
shows *aEval* (*Eq* (*A* + *B* $*$ *Var var*)) (*list-update L var* (*sqrt Cv*)) = *eval* (*And* (*Atom*(*Leq* (*A*$*$*B*))) (*Atom* (*Eq* (*A*$\hat{}$*2*$-$*B*$\hat{}$*2*$*$*C*)))) (*list-update L var* (*sqrt Cv*))
proof$-$
have *h1* : $\forall$ *x*. *insertion* (*nth-default 0* (*list-update L var x*)) (*A*$\hat{}$*2*$-$(*B*$\hat{}$*2*)$*$*C*) = *Av*$\hat{}$*2*$-$(*Bv*$\hat{}$*2*)$*$*Cv*
by(*simp add*: *hA hB hC insertion-add insertion-mult insertion-sub insertion-pow*)
have *h2a* : (*Av* + *Bv* $*$ *sqrt Cv* = *0*) = (*Av* = $-$ *Bv* $*$ *sqrt Cv*)
by *auto*
have *h2b* : (*Av* = $-$ *Bv* $*$ *sqrt Cv*) $\Longrightarrow$ (*Av*$\hat{}$*2* = ($-$ *Bv* $*$ *sqrt Cv*)$\hat{}$*2*)
by *simp*
have *h2c* : (*Av*$\hat{}$*2* = ($-$ *Bv* $*$ *sqrt Cv*)$\hat{}$*2*) = (*Av*$\hat{}$*2* = *Bv*$\hat{}$*2* $*$ (*sqrt Cv*)$\hat{}$*2*)

by (*simp add*: *power-mult-distrib*)
have *h2d* : (*Av*^2 = *Bv*^2 * (*sqrt Cv*)^2) = (*Av*^2 = *Bv*^2 * *Cv*)
by (*simp add*: *detGreat0*)
have *h2* : (*Av* + *Bv* * *sqrt Cv* = 0) ⟹ (*Av*^2 = *Bv*^2 * *Cv*)
using *h2a h2b h2c h2d* **by** *blast*
have *h3a* : (*Av*∗*Bv* > 0) ⟹ (*Av* + *Bv* * *sqrt Cv* ≠ 0)
by (*smt detGreat0 mult-nonneg-nonneg real-sqrt-ge-zero zero-less-mult-iff*)
have *h3* : (*Av* + *Bv* * *sqrt Cv* = 0) ⟹ (*Av*∗*Bv*≤ 0)
using *h3a* **by** *linarith*
have *h4* : (*Av* * *Bv* ≤ 0 ∧ *Av*^{2} = *Bv*^{2} * *Cv*) ⟹ (*Av* + *Bv* * *sqrt Cv* = 0)
apply(*cases Av*>0)
apply (*metis detGreat0 h2a h2c h2d mult-minus-left not-le power2-eq-iff real-sqrt-lt-0-iff zero-less-mult-iff*)
by (*smt h2a real-sqrt-abs real-sqrt-mult zero-less-mult-iff*)
show *?thesis*
apply(*simp add*: *hA hB h1 insertion-add insertion-mult insertion-var lLength*)
using *h2 h3 h4* **by** *blast*
qed

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 −
define *p1* **where** (*p1*::*real mpoly*) = *quadratic-part-1 var a b d* (*Eq p*)
define *p2* **where** (*p2*::*real mpoly*) = *quadratic-part-2 var c p1*
define *A* **where** *A* = *isolate-variable-sparse p2 var 0*
define *B* **where** *B* = *isolate-variable-sparse p2 var 1*
have *h3c* : *MPoly-Type.degree p2 var* = 0 ∨ *MPoly-Type.degree p2 var* = 1
using *freeC quad-part-2-deg p2-def* **by** (*meson le-neq-implies-less less-one*)
have *h3d* : *MPoly-Type.degree p2 var* = 0 ⟹ *B* = 0
by(*simp add*: *B-def isovar-greater-degree*)
then have *h3f* : *MPoly-Type.degree p2 var* = 0 ⟹ *p2* = *A* + *B* * *Var var*
by(*simp add*: *h3d A-def degree0isovarspar*)
have *h3g1* : *MPoly-Type.degree p2 var* = 1 ⟹ *p2* = (∑ *i*≤1. *isolate-variable-sparse p2 var i* * *Var var* ^ *i*)
using *sum-over-zero* **by** *metis*
have *h3g2a* : ∀ *f*. (∑ *i*::*nat*≤1. *f i*) = *f* 0 + *f* 1 **by** *simp*

have *h3g2* : $(\sum i{::}nat{\leq}1.$ *isolate-variable-sparse p2 var i* $*$ *Var var* $\hat{}$ *i*$) =$
isolate-variable-sparse p2 var 0 $*$ *Var var* $\hat{}$ *0* $+$ *isolate-variable-sparse p2 var 1* $*$ *Var var* $\hat{}$ *1*
using *h3g2a* **by** *blast*
have *h3g* : *MPoly-Type.degree p2 var* $= 1 \Longrightarrow$ *p2* $= A + B *$ *Var var*
apply(*simp add*: *sum-over-zero A-def B-def*)
using *h3g1 h3g2*
by (*metis* (*no-types*, *lifting*) *One-nat-def mult-cancel-left2 power-0 power-one-right*)
have *h3h* : *p2* $= A + B *$ *Var var*
using *h3c h3f h3g* **by** *auto*

have *h4a* : $\exists$ *x*::*real*. $\forall$ *y*::*real*. *insertion* (*nth-default 0* (*list-update L var y*)) *A* $=$ *x*
using *not-contains-insertion not-in-isovarspar A-def* **by** *blast*
have *h4b* : $\exists$ *x*::*real*. $\forall$ *y*::*real*. *insertion* (*nth-default 0* (*list-update L var y*)) *B* $=$ *x*
using *not-contains-insertion not-in-isovarspar B-def* **by** *blast*

have *aEval* (*Eq p*) (*list-update L var* ((*Av*+*Bv*∗*sqrt*(*Cv*))/*Dv*)) $=$ *aEval* (*Eq p1*) (*list-update L var* (*sqrt Cv*))
using *p1-def quad-part-1-eq nonzero ha hb hd lLength* **by** *blast*
also have *h2* : ... $=$ *aEval* (*Eq p2*) (*list-update L var* (*sqrt Cv*))
using *p2-def quad-part-2-eq lLength detGreater0 hc* **by** *metis*
also have ... $=$ *aEval* (*Eq* $(A + B *$ *Var var*)) (*list-update L var* (*sqrt Cv*))
using *h3h* **by** *auto*
also have ... $=$ *eval* (*And* (*Atom*(*Leq* (*A*∗*B*))) (*Atom* (*Eq* (*A*^*2*−*B*^*2*∗*c*)))) (*list-update L var* (*sqrt Cv*))
using *quad-equality-helper hc detGreater0 h4a h4b lLength* **by** *blast*
also have ... $=$ *eval* (*quadratic-sub var a b c d* (*Eq p*)) (*list-update L var* (*sqrt Cv*))
using *p2-def A-def B-def p1-def quadratic-sub.simps*(*1*) **by** *metis*
finally show *?thesis* **by** *blast*
qed

lemma *quadratic-sub-less-helper* :
assumes *lLength* : *length L* $>$ *var*
assumes *detGreat0* : *Cv*$\geq$*0*
assumes *hC* : $\forall$ *x*. *insertion* (*nth-default 0* (*list-update L var x*)) (*C*::*real mpoly*) $=$ (*Cv*::*real*)
assumes *hA* : $\forall$ *x*. *insertion* (*nth-default 0* (*list-update L var x*)) (*A*::*real mpoly*) $=$ (*Av*::*real*)
assumes *hB* : $\forall$ *x*. *insertion* (*nth-default 0* (*list-update L var x*)) (*B*::*real mpoly*) $=$ (*Bv*::*real*)
shows *aEval* (*Less* $(A + B *$ *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−
have *h1* : ∀ *x*. *insertion* (*nth-default 0* (*list-update L var x*)) (*A*^2−(*B*^2)∗*C*) = *Av*^2−(*Bv*^2)∗*Cv*
by(*simp add*: *hA hB hC insertion-add insertion-mult insertion-sub insertion-pow*)
have *h2* : ∀ *x*. *insertion* (*nth-default 0* (*list-update L var x*)) ((*B*^2)∗*C*−*A*^2) = (*Bv*^2)∗*Cv*−*Av*^2
by(*simp add*: *hA hB hC insertion-add insertion-mult insertion-sub insertion-pow*)
have *h3* : *Av*=*0* ⟹ *Bv*=*0* ⟹ (*Av* + *Bv* ∗ *sqrt Cv* < *0*) = (*Av* < *0* ∧ *Bv*^{2} ∗ *Cv* < *Av*^{2} ∨ *Bv* ≤ *0* ∧ (*Av* < *0* ∨ *Av*^{2} < *Bv*^{2} ∗ *Cv*))
by *simp*
have *h4* : *Av*<*0* ⟹ *Bv*≤*0* ⟹ (*Av* + *Bv* ∗ *sqrt Cv* < *0*) = (*Av* < *0* ∧ *Bv*^{2} ∗ *Cv* < *Av*^{2} ∨ *Bv* ≤ *0* ∧ (*Av* < *0* ∨ *Av*^{2} < *Bv*^{2} ∗ *Cv*))
by (*metis add.right-neutral add-mono-thms-linordered-field*(*5*) *detGreat0 less-eq-real-def mult-less-0-iff mult-zero-class.mult-zero-left mult-zero-class.mult-zero-right real-sqrt-eq-zero-cancel-iff real-sqrt-gt-0-iff*)
have *h5a* : *Av*≥*0* ⟹ *Bv*≤*0* ⟹ (*Av* < −*Bv* ∗ *sqrt Cv*) ⟹ (*Av*^{2} < *Bv*^{2} ∗ *Cv*)
proof −
assume *a1*: *0* ≤ *Av*
assume *a2*: *Av* < − *Bv* ∗ *sqrt Cv*
assume *Bv* ≤ *0*
then have *Av* < *sqrt* (*Cv* ∗ (*Bv* ∗ *Bv*))
using *a2* **by** (*simp add*: *mult.commute real-sqrt-mult*)
then show *?thesis*
using *a1* **by** (*metis* (*no-types*) *mult.commute power2-eq-square real-sqrt-less-iff real-sqrt-mult real-sqrt-pow2-iff*)
qed
have *h5b* : *Av*≥*0* ⟹ *Bv*≤*0* ⟹ (*Av*^{2} < *Bv*^{2} ∗ *Cv*) ⟹ (*Av* < −*Bv* ∗ *sqrt Cv*)
using *real-less-rsqrt real-sqrt-mult* **by** *fastforce*
have *h5* : *Av*≥*0* ⟹ *Bv*≤*0* ⟹ (*Av* + *Bv* ∗ *sqrt Cv* < *0*) = (*Av* < *0* ∧ *Bv*^{2} ∗ *Cv* < *Av*^{2} ∨ *Bv* ≤ *0* ∧ (*Av* < *0* ∨ *Av*^{2} < *Bv*^{2} ∗ *Cv*))
using *h5a h5b* **by** *linarith*
have *h6* : *Av*≥*0* ⟹ *Bv*>*0* ⟹ (*Av* + *Bv* ∗ *sqrt Cv* < *0*) = (*Av* < *0* ∧ *Bv*^{2} ∗ *Cv* < *Av*^{2} ∨ *Bv* ≤ *0* ∧ (*Av* < *0* ∨ *Av*^{2} < *Bv*^{2} ∗ *Cv*))
by (*smt detGreat0 mult-nonneg-nonneg real-sqrt-ge-zero*)
have *h7a* : *Av*<*0* ⟹ *Bv*>*0* ⟹ (*Av* < −*Bv* ∗ *sqrt Cv*) ⟹ (*Bv*^{2} ∗ *Cv* < *Av*^{2})
by (*smt mult-minus-left real-sqrt-abs real-sqrt-le-mono real-sqrt-mult*)
have *h7b* : *Av*<*0* ⟹ *Bv*>*0* ⟹ (*Bv*^{2} ∗ *Cv* < *Av*^{2}) ⟹ (*Av* < −*Bv* ∗ *sqrt Cv*)
by (*metis abs-of-nonneg abs-real-def add.commute less-eq-real-def mult.assoc mult-minus-left power2-eq-square real-add-less-0-iff real-sqrt-less-iff real-sqrt-mult real-sqrt-mult-self*)
have *h7* : *Av*<*0* ⟹ *Bv*>*0* ⟹ (*Av* + *Bv* ∗ *sqrt Cv* < *0*) = (*Av* < *0* ∧ *Bv*^{2} ∗ *Cv* < *Av*^{2} ∨ *Bv* ≤ *0* ∧ (*Av* < *0* ∨ *Av*^{2} < *Bv*^{2} ∗ *Cv*))
using *h7a h7b* **by** *linarith*
show *?thesis*
apply(*simp add*: *hA hB h1 h2 insertion-add insertion-mult insertion-var lLength*)
using *h3 h4 h5 h6 h7* **by** *smt*
qed

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 −
define *p1* **where** (*p1*::*real mpoly*) = *quadratic-part-1 var a b d* (*Less p*)
define *p2* **where** (*p2*::*real mpoly*) = *quadratic-part-2 var c p1*
define *A* **where** *A* = *isolate-variable-sparse p2 var 0*
define *B* **where** *B* = *isolate-variable-sparse p2 var 1*

have *h3b* : *MPoly-Type.degree p2 var* ≤ *1*
using *freeC quad-part-2-deg p2-def* **by** *blast*
then have *h3c* : *MPoly-Type.degree p2 var* = *0* ∨ *MPoly-Type.degree p2 var* = *1*
by *auto*
have *h3d* : *MPoly-Type.degree p2 var* = *0* ⟹ *B* = *0*
by(*simp add*: *B-def isovar-greater-degree*)
then have *h3f* : *MPoly-Type.degree p2 var* = *0* ⟹ *p2* = *A* + *B* ∗ *Var var*
by(*simp add*: *h3d A-def degree0isovarspar*)
have *h3g1* : *MPoly-Type.degree p2 var* = *1* ⟹ *p2* = (∑ *i*≤*1*. *isolate-variable-sparse p2 var i* ∗ *Var var* ^ *i*)
using *sum-over-zero* **by** *metis*
have *h3g2a* : ∀ *f*. (∑ *i*::*nat*≤*1*. *f i*) = *f 0* + *f 1* **by** *simp*
have *h3g2* : (∑ *i*::*nat*≤*1*. *isolate-variable-sparse p2 var i* ∗ *Var var* ^ *i*) =
isolate-variable-sparse p2 var 0 ∗ *Var var* ^ *0* + *isolate-variable-sparse p2 var 1* ∗ *Var var* ^ *1*
using *h3g2a* **by** *blast*
have *h3g* : *MPoly-Type.degree p2 var* = *1* ⟹ *p2* = *A* + *B* ∗ *Var var*
apply(*simp add*: *sum-over-zero A-def B-def*)
using *h3g1 h3g2*
by (*metis* (*no-types*, *lifting*) *One-nat-def mult-cancel-left2 power-0 power-one-right*)
have *h3h* : *p2* = *A* + *B* ∗ *Var var*
using *h3c h3f h3g* **by** *auto*

have *h4a* : ∃ *x*::*real*. ∀ *y*::*real*. *insertion* (*nth-default 0*(*list-update L var y*)) *A* = *x*
using *not-contains-insertion not-in-isovarspar A-def* **by** *blast*
have *h4b* : ∃ *x*::*real*. ∀ *y*::*real*. *insertion* (*nth-default 0*(*list-update L var y*)) *B* = *x*

using *not-contains-insertion not-in-isovarspar B-def* **by** *blast*

have *h1* : *aEval* (*Less p*) (*list-update L var* ((*Av*+*Bv*∗*sqrt*(*Cv*))/*Dv*)) = *aEval* (*Less* (*quadratic-part-1 var a b d* (*Less p*))) (*list-update L var* (*sqrt Cv*))
using *quad-part-1-less assms* **by** *blast*
also have ... = *aEval* (*Less p1*) (*list-update L var* (*sqrt Cv*))
using *p1-def* **by** *auto*
also have ... = *aEval* (*Less* (*quadratic-part-2 var c p1*)) (*list-update L var* (*sqrt Cv*))
using *quad-part-2-less assms* **by** *blast*
also have ... = *aEval* (*Less p2*) (*list-update L var* (*sqrt Cv*))
using *p2-def* **by** *auto*
also have ... = *aEval* (*Less* (*A* + *B* ∗ *Var var*)) (*list-update L var* (*sqrt Cv*))
using *h3h* **by** *auto*
also have ... = *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*))
using *quadratic-sub-less-helper hc detGreater0 h4a h4b lLength* **by** *blast*
also have ... = *eval* (*quadratic-sub var a b c d* (*Less p*)) (*list-update L var* (*sqrt Cv*))
using *p2-def A-def B-def p1-def quadratic-sub.simps*(*2*) **by** *metis*
finally show *?thesis* **by** *blast*
qed

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−
have *h1* : ∀ *x*. *insertion* (*nth-default 0* (*list-update L var x*)) (*A*^*2*−(*B*^*2*)∗*C*) = *Av*^*2*−(*Bv*^*2*)∗*Cv*
by(*simp add*: *hA hB hC insertion-add insertion-mult insertion-sub insertion-pow*)
have *h2* : ∀ *x*. *insertion* (*nth-default 0* (*list-update L var x*)) ((*B*^*2*)∗*C*−*A*^*2*) = (*Bv*^*2*)∗*Cv*−*Av*^*2*
by(*simp add*: *hA hB hC insertion-add insertion-mult insertion-sub insertion-pow*)
have *h3* : *Av*=*0* ⟹ *Bv*=*0* ⟹ (*Av* + *Bv* ∗ *sqrt Cv* ≤ *0*) = (*Av* ≤ *0* ∧ Bv^2 ∗ *Cv* ≤ Av^2 ∨ *Bv* ≤ *0* ∧ Av^2 ≤ Bv^2 ∗ *Cv*)
by *simp*

have *h4* : $Av<0 \Longrightarrow Bv\leq 0 \Longrightarrow (Av + Bv * sqrt\ Cv \leq 0) = (Av \leq 0 \wedge Bv^2 * Cv \leq Av^2 \vee Bv \leq 0 \wedge Av^2 \leq Bv^2 * Cv)$
by (*smt detGreat0 real-sqrt-ge-zero zero-less-mult-iff*)
have *h5* : $Av=0 \Longrightarrow Bv\leq 0 \Longrightarrow (Av + Bv * sqrt\ Cv \leq 0) = (Av \leq 0 \wedge Bv^2 * Cv \leq Av^2 \vee Bv \leq 0 \wedge Av^2 \leq Bv^2 * Cv)$
by (*smt detGreat0 real-sqrt-ge-zero zero-less-mult-iff*)
have *h6* : $Av\geq 0 \Longrightarrow Bv>0 \Longrightarrow (Av + Bv * sqrt\ Cv \leq 0) = (Av \leq 0 \wedge Bv^2 * Cv \leq Av^2 \vee Bv \leq 0 \wedge Av^2 \leq Bv^2 * Cv)$
by (*smt detGreat0 mult-nonneg-nonneg mult-pos-pos real-sqrt-gt-0-iff real-sqrt-zero zero-le-power2 zero-less-mult-pos zero-less-power2*)
have *h7a* : $Av<0 \Longrightarrow Bv>0 \Longrightarrow (Av + Bv * sqrt\ Cv \leq 0) \Longrightarrow Bv^2 * Cv \leq Av^2$
by (*smt real-sqrt-abs real-sqrt-less-mono real-sqrt-mult*)
have *h7b* : $Av<0 \Longrightarrow Bv>0 \Longrightarrow\ Bv^2 * Cv \leq Av^2 \Longrightarrow (Av + Bv * sqrt\ Cv \leq 0)$
by (*smt real-sqrt-abs real-sqrt-less-mono real-sqrt-mult*)
have *h7* : $Av<0 \Longrightarrow Bv>0 \Longrightarrow (Av + Bv * sqrt\ Cv \leq 0) = (Av \leq 0 \wedge Bv^2 * Cv \leq Av^2 \vee Bv \leq 0 \wedge Av^2 \leq Bv^2 * Cv)$
using *h7a h7b* **by** *linarith*
have *h8c* : $(-Bv * sqrt\ Cv)\hat{}2 = Bv^2 * Cv$
by (*simp add*: *detGreat0 power-mult-distrib*)
have *h8a* : $Av>0 \Longrightarrow Bv\leq 0\ \Longrightarrow (Av \leq -Bv * sqrt\ Cv) \Longrightarrow\ Av^2 \leq Bv^2 * Cv$
using *detGreat0 h8c power-both-sides* **by** *smt*
have *h8b* : $Av>0 \Longrightarrow Bv\leq 0\ \Longrightarrow\ \ Av^2 \leq Bv^2 * Cv \Longrightarrow (Av + Bv * sqrt\ Cv \leq 0)$
using *detGreat0 h8c power-both-sides*
by (*smt mult-minus-left real-sqrt-ge-zero zero-less-mult-iff*)
have *h8* : $Av>0 \Longrightarrow Bv\leq 0 \Longrightarrow (Av + Bv * sqrt\ Cv \leq 0) = (Av \leq 0 \wedge Bv^2 * Cv \leq Av^2 \vee Bv \leq 0 \wedge Av^2 \leq Bv^2 * Cv)$
using *h8a h8b* **by** *linarith*
show *?thesis*
apply(*simp add*: *hA hB h1 h2 insertion-add insertion-mult insertion-var lLength*)
using *h3 h4 h5 h6 h7 h8* **by** *smt*
qed

lemma *quadratic-sub-leq* :
assumes *lLength* : *length L* > *var*
assumes *nonzero* : $Dv \neq 0$
assumes *detGreater0* : $Cv \geq 0$
assumes *freeC* : $var \notin vars\ c$
assumes *ha* : $\forall x.\ insertion\ (nth\text{-}default\ 0\ (list\text{-}update\ L\ var\ x))\ (a{::}real\ mpoly) = (Av :: real)$
assumes *hb* : $\forall x.\ insertion\ (nth\text{-}default\ 0\ (list\text{-}update\ L\ var\ x))\ (b{::}real\ mpoly) = (Bv :: real)$
assumes *hc* : $\forall x.\ insertion\ (nth\text{-}default\ 0\ (list\text{-}update\ L\ var\ x))\ (c{::}real\ mpoly) = (Cv :: real)$
assumes *hd* : $\forall x.\ insertion\ (nth\text{-}default\ 0\ (list\text{-}update\ L\ var\ x))\ (d{::}real\ mpoly) = (Dv :: real)$
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 −
define *p1* **where** (*p1*::*real mpoly*) = *quadratic-part-1 var a b d* (*Leq p*)
define *p2* **where** (*p2*::*real mpoly*) = *quadratic-part-2 var c p1*
define *A* **where** *A* = *isolate-variable-sparse p2 var 0*
define *B* **where** *B* = *isolate-variable-sparse p2 var 1*

have *h3b* : *MPoly-Type.degree p2 var* ≤ *1*
using *freeC quad-part-2-deg p2-def lLength* **by** *metis*
then have *h3c* : *MPoly-Type.degree p2 var* = *0* ∨ *MPoly-Type.degree p2 var* = *1*
by *auto*
have *h3d* : *MPoly-Type.degree p2 var* = *0* ⟹ *B* = *0*
by(*simp add*: *B-def isovar-greater-degree*)
then have *h3f* : *MPoly-Type.degree p2 var* = *0* ⟹ *p2* = *A* + *B* ∗ *Var var*
by(*simp add*: *h3d A-def degree0isovarspar*)
have *h3g1* : *MPoly-Type.degree p2 var* = *1* ⟹ *p2* = (∑ *i*≤*1*. *isolate-variable-sparse p2 var i* ∗ *Var var* ^ *i*)
using *sum-over-zero* **by** *metis*
have *h3g2a* : ∀ *f*. (∑ *i*::*nat*≤*1*. *f i*) = *f 0* + *f 1* **by** *simp*
have *h3g2* : (∑ *i*::*nat*≤*1*. *isolate-variable-sparse p2 var i* ∗ *Var var* ^ *i*) =
isolate-variable-sparse p2 var 0 ∗ *Var var* ^ *0* + *isolate-variable-sparse p2 var 1* ∗ *Var var* ^ *1*
using *h3g2a* **by** *blast*
have *h3g* : *MPoly-Type.degree p2 var* = *1* ⟹ *p2* = *A* + *B* ∗ *Var var*
apply(*simp add*: *sum-over-zero A-def B-def*)
using *h3g1 h3g2*
by (*metis* (*no-types*, *lifting*) *One-nat-def mult-cancel-left2 power-0 power-one-right*)
have *h3h* : *p2* = *A* + *B* ∗ *Var var*
using *h3c h3f h3g* **by** *auto*

have *h4a* : ∃ *x*::*real*. ∀ *y*::*real*. *insertion* (*nth-default 0* (*list-update L var y*)) *A* = *x*
using *not-contains-insertion not-in-isovarspar A-def* **by** *blast*
have *h4b* : ∃ *x*::*real*. ∀ *y*::*real*. *insertion* (*nth-default 0* (*list-update L var y*)) *B* = *x*
using *not-contains-insertion not-in-isovarspar B-def* **by** *blast*

have *aEval* (*Leq p*) (*list-update L var* ((*Av*+*Bv*∗*sqrt*(*Cv*))/*Dv*)) = *aEval* (*Leq p1*) (*list-update L var* (*sqrt Cv*))
using *quad-part-1-leq nonzero ha hb hd p1-def lLength* **by** *metis*
also have ... = *aEval* (*Leq p2*) (*list-update L var* (*sqrt Cv*))
using *p2-def quad-part-2-leq hc detGreater0 lLength* **by** *metis*
also have ... = *aEval* (*Leq* (*A* + *B* ∗ *Var var*)) (*list-update L var* (*sqrt Cv*))
using *h3h* **by** *auto*
also have *h4* : ... = *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))
   using quadratic-sub-leq-helper hc detGreater0 h4a h4b lLength by blast
 also have ... = eval (quadratic-sub var a b c d (Leq p)) (list-update L var (sqrt
Cv))
   using p1-def quadratic-sub.simps(3) p2-def A-def B-def by metis
 finally show ?thesis by blast
qed

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 −
 define p1 where (p1::real mpoly) = quadratic-part-1 var a b d (Neq p)
 define p2 where (p2::real mpoly) = quadratic-part-2 var c p1
 define A where A = isolate-variable-sparse p2 var 0
 define B where B = isolate-variable-sparse p2 var 1

 have h3b : MPoly-Type.degree p2 var ≤ 1
   using freeC quad-part-2-deg p2-def lLength by metis
 then have h3c : MPoly-Type.degree p2 var = 0 ∨ MPoly-Type.degree p2 var =
1
   by auto
 have h3d : MPoly-Type.degree p2 var = 0 ⟹ B = 0
   by(simp add: B-def isovar-greater-degree)
 then have h3f : MPoly-Type.degree p2 var = 0 ⟹ p2 = A + B * Var var
   by(simp add: h3d A-def degree0isovarspar)
 have h3g1 : MPoly-Type.degree p2 var = 1 ⟹ p2 = (∑ i≤1. isolate-variable-sparse
p2 var i * Var var ^ i)
   using sum-over-zero by metis
 have h3g2a : ∀ f. (∑ i::nat≤1. f i) = f 0 + f 1 by simp
 have h3g2 : (∑ i::nat≤1. isolate-variable-sparse p2 var i * Var var ^ i) =
          isolate-variable-sparse p2 var 0 * Var var ^ 0 + isolate-variable-sparse
p2 var 1 * Var var ^ 1
   using h3g2a by blast
```

have *h3g* : *MPoly-Type.degree p2 var = 1 ⟹ p2 = A + B * Var var*
apply(*simp add*: *sum-over-zero A-def B-def*)
using *h3g1 h3g2*
by (*metis* (*no-types*, *lifting*) *One-nat-def mult-cancel-left2 power-0 power-one-right*)
have *h3h* : *p2 = A + B * Var var*
using *h3c h3f h3g* **by** *auto*

have *h4a* : ∃ *x*::*real*. ∀ *y*::*real*. *insertion* (*nth-default 0* (*list-update L var y*)) *A = x*
using *not-contains-insertion not-in-isovarspar A-def* **by** *blast*
have *h4b* : ∃ *x*::*real*. ∀ *y*::*real*. *insertion* (*nth-default 0* (*list-update L var y*)) *B = x*
using *not-contains-insertion not-in-isovarspar B-def* **by** *blast*
have *h4c* : *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*))
using *quad-equality-helper hc detGreater0 h4a h4b lLength* **by** *blast*
have *h4d* : *aEval* (*Neq* (*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*))))
using *aEval.simps*(*1*) *aEval.simps*(*4*) *h4c* **by** *blast*
have *h4e* : (*¬* (*eval* (*And* (*Atom*(*Leq* (*A*B*))) (*Atom* (*Eq* (*A^2−B^2*c*)))) (*list-update L var* (*sqrt Cv*))))
= eval (*Or* (*Atom*(*Less*(*−A*B*))) (*Atom* (*Neq*(*A^2−B^2*c*)))) (*list-update L var* (*sqrt Cv*))
by (*metis aNeg.simps*(*2*) *aNeg.simps*(*3*) *aNeg-aEval eval.simps*(*1*) *eval.simps*(*4*) *eval.simps*(*5*) *mult-minus-left*)

have *aEval* (*Neq p*) (*list-update L var* ((*Av+Bv*sqrt*(*Cv*))/*Dv*)) = *aEval* (*Neq p1*) (*list-update L var* (*sqrt Cv*))
using *quad-part-1-neq nonzero ha hb hd p1-def lLength* **by** *blast*
also have ... = *aEval* (*Neq p2*) (*list-update L var* (*sqrt Cv*))
using *p2-def quad-part-2-neq hc detGreater0 lLength* **by** *metis*
also have ... = *aEval* (*Neq* (*A + B * Var var*)) (*list-update L var* (*sqrt Cv*))
using *h3h* **by** *auto*
also have ... = *eval* (*Or*
(*Atom*(*Less*(*−A*B*)))
(*Atom* (*Neq*(*A^2−B^2*c*)))) (*list-update L var* (*sqrt Cv*))
using *h4c h4d h4e* **by** *auto*
also have ... = *eval* (*quadratic-sub var a b c d* (*Neq p*)) (*list-update L var* (*sqrt Cv*))
using *p2-def A-def B-def p1-def quadratic-sub.simps*(*4*) *quadratic-part-1.simps*(*1*) *quadratic-part-1.simps*(*4*)
by (*metis* (*no-types*, *lifting*))
finally show *?thesis* **by** *blast*
qed

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(*cases A*)
case (*Less p*)
define *p1* **where** (*p1*::*real mpoly*) = *quadratic-part-1 var a b d* (*Less p*)
define *p2* **where** (*p2*::*real mpoly*) = *quadratic-part-2 var c p1*
define *A* **where** *A* = *isolate-variable-sparse p2 var 0*
define *B* **where** *B* = *isolate-variable-sparse p2 var 1*
have *h1* : *freeIn var* (*quadratic-sub var a b c d* (*Less p*)) = *freeIn var* (*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*))))))
using *p2-def A-def B-def p1-def quadratic-sub.simps*(*2*) **by** *metis*
have *h2d* : *var*∉*vars*(*4*::*real mpoly*)
by (*metis freeB not-in-add not-in-pow numeral-Bit0 one-add-one power-0*)
have *h2* : *freeIn var* ((*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*)))))))
using *vars-mult not-in-isovarspar A-def B-def not-in-sub not-in-mult not-in-neg not-in-pow not-in-isovarspar h2d freeC*
by (*simp*)
show *?thesis* **using** *h1 h2 Less* **by** *blast*
next
case (*Eq p*)
define *p1* **where** (*p1*::*real mpoly*) = *quadratic-part-1 var a b d* (*Eq p*)
define *p2* **where** (*p2*::*real mpoly*) = *quadratic-part-2 var c p1*
define *A* **where** *A* = *isolate-variable-sparse p2 var 0*
define *B* **where** *B* = *isolate-variable-sparse p2 var 1*
have *h1* : *freeIn var* (*quadratic-sub var a b c d* (*Eq p*)) = *freeIn var* (*And* (*Atom*(*Leq* (*A*∗*B*))) (*Atom* (*Eq* (A^2 − B^2 ∗ *c*))))
using *p2-def A-def B-def p1-def quadratic-sub.simps*(*1*) **by** *metis*
have *h2d* : *var*∉*vars*(*4*::*real mpoly*)
by (*metis freeB not-in-add not-in-pow numeral-Bit0 one-add-one power-0*)
have *h2* : *freeIn var* (*And* (*Atom*(*Leq* (*A*∗*B*))) (*Atom* (*Eq* (A^2 − B^2 ∗ *c*))))
using *vars-mult not-in-isovarspar A-def B-def not-in-sub not-in-mult not-in-neg not-in-pow not-in-isovarspar h2d freeC*
by (*simp*)
show *?thesis* **using** *h1 h2 Eq* **by** *blast*
next
case (*Leq p*)
define *p1* **where** (*p1*::*real mpoly*) = *quadratic-part-1 var a b d* (*Leq p*)
define *p2* **where** (*p2*::*real mpoly*) = *quadratic-part-2 var c p1*
define *A* **where** *A* = *isolate-variable-sparse p2 var 0*
define *B* **where** *B* = *isolate-variable-sparse p2 var 1*
have *h1* : *freeIn var* (*quadratic-sub var a b c d* (*Leq p*)) = *freeIn var* (*Or*(*And*(*Atom*(*Leq*(*A*)))(*Atom* (*Leq*(*B*^*2*∗*c*−*A*^*2*))))(*And*(*Atom*(*Leq B*))(*Atom*(*Leq* (*A*^*2*−*B*^*2*∗*c*)))))

using *p2-def A-def B-def p1-def quadratic-sub.simps*(*3*) **by** *metis*
have *h2d* : *var*∉*vars*(*4*::*real mpoly*)
by (*metis freeB not-in-add not-in-pow numeral-Bit0 one-add-one power-0*)
have *h2* : *freeIn var* (*Or*(*And*(*Atom*(*Leq*(*A*)))(*Atom* (*Leq*(*B*^*2*∗*c*−*A*^*2*))))(*And*(*Atom*(*Leq B*))(*Atom*(*Leq* (*A*^*2*−*B*^*2*∗*c*)))))
using *vars-mult not-in-isovarspar A-def B-def not-in-sub not-in-mult not-in-neg not-in-pow not-in-isovarspar h2d freeC*
by (*simp*)
show *?thesis* **using** *h1 h2 Leq* **by** *blast*
next
case (*Neq p*)
define *p1* **where** (*p1*::*real mpoly*) = *quadratic-part-1 var a b d* (*Neq p*)
define *p2* **where** (*p2*::*real mpoly*) = *quadratic-part-2 var c p1*
define *A* **where** *A* = *isolate-variable-sparse p2 var 0*
define *B* **where** *B* = *isolate-variable-sparse p2 var 1*
have *h1* : *freeIn var* (*quadratic-sub var a b c d* (*Neq p*)) = *freeIn var* (*Or* (*Atom*(*Less*(−*A*∗*B*))) (*Atom* (*Neq*(*A*^*2*−*B*^*2*∗*c*))))
using *p2-def A-def B-def p1-def quadratic-sub.simps*(*4*) **by** *metis*
have *h2d* : *var*∉*vars*(*4*::*real mpoly*)
by (*metis freeB not-in-add not-in-pow numeral-Bit0 one-add-one power-0*)
have *h2* : *freeIn var* (*Or* (*Atom*(*Less*(−*A*∗*B*))) (*Atom* (*Neq*(*A*^*2*−*B*^*2*∗*c*))))
using *vars-mult not-in-isovarspar A-def B-def not-in-sub not-in-mult not-in-neg not-in-pow not-in-isovarspar h2d freeC*
by (*simp*)
show *?thesis* **using** *h1 h2 Neq* **by** *blast*
qed

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(*cases A*)
case (*Less x1*)
then show *?thesis* **using** *quadratic-sub-less assms* **by** *blast*
next
case (*Eq x2*)
then show *?thesis* **using** *quadratic-sub-eq assms* **by** *blast*
next

case (*Leq x3*)
then show *?thesis* **using** *quadratic-sub-leq assms* **by** *blast*
next
case (*Neq x4*)
then show *?thesis* **using** *quadratic-sub-neq assms* **by** *blast*
qed

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(*induction F arbitrary*: *z*)
case *TrueF*
then show *?case* **by** *auto*
next
case *FalseF*
then show *?case* **by** *auto*
next
case (*Atom x*)
then show *?case* **using** *free-in-quad*[*OF not-in-lift*[*OF assms*(*1*)] *not-in-lift*[*OF assms*(*2*)] *not-in-lift*[*OF assms*(*3*)] *not-in-lift*[*OF assms*(*4*)], *of z*] **by** *auto*
next
case (*And F1 F2*)
then show *?case* **by** *auto*
next
case (*Or F1 F2*)
then show *?case* **by** *auto*
next
case (*Neg F*)
then show *?case* **by** *auto*
next
case (*ExQ F*)
show *?case* **using** *ExQ*[*of z*+*1*] **by** *simp*
next
case (*AllQ F*)
show *?case* **using** *AllQ*[*of z*+*1*] **by** *simp*
next
case (*ExN x1 F*)
then show *?case*
by (*metis* (*no-types*, *lifting*) *add.assoc freeIn.simps*(*13*) *liftmap.simps*(*10*) *quadratic-sub-fm-helper.simps*)
next
case (*AllN x1 F*)
then show *?case*
by (*metis* (*no-types*, *lifting*) *freeIn.simps*(*12*) *group-cancel.add1 liftmap.simps*(*9*)

quadratic-sub-fm-helper.simps)
qed

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*)
using *free-in-quad-fm-helper*[*OF assms*, *of 0*] **by** *auto*

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*))
using *assms* **proof**(*induction F arbitrary*: *z L*)
case *TrueF*
then show *?case* **by** *auto*
next
case *FalseF*
then show *?case* **by** *auto*
next
case (*Atom x*)
define *L1* **where** *L1* = *drop z L*
define *L2* **where** *L2* = *take z L*
have *L-def* : *L* = *L2* @ *L1* **using** *L1-def L2-def* **by** *auto*
have *lengthl2* : *length L2* = *z* **using** *L2-def*
using *Atom.prems*(*4*) **by** *auto*
have *eval* (*Atom*(*Eq* (*a*−*Const Av*))) ([] @ *L1*) = *eval* (*liftFm 0 z* (*Atom*(*Eq* (*a*−*Const Av*)))) ([] @ *L2* @ *L1*)
by (*metis eval-liftFm-helper lengthl2 list.size*(*3*))
then have (*insertion* (*nth-default 0* (*L2* @ *L1*)) (*liftPoly 0 z* (*a* − *Const Av*)) = *0*)
apply(*simp add*: *insertion-sub insertion-const*)
using *Atom*(*5*) **unfolding** *L1-def*
by (*metis list-update-id*)

then have *insertion* (*nth-default 0* (*L2* @ *L1*)) (*liftPoly 0 z a*) = *Av*
using *lift-minus* **by** *blast*
then have *a1* : ∀ *x*. *insertion* (*nth-default 0* (*L*[*var* + *z* := *x*])) (*liftPoly 0 z a*) = *Av*
unfolding *L-def*
by (*metis* (*no-types*, *lifting*) *Atom.prems*(*5*) *L1-def add.right-neutral add-diff-cancel-right′ append-eq-append-conv append-eq-append-conv2 length-append lengthl2 lift-insertion list.size*(*3*) *list-update-append not-add-less2*)
have *eval* (*Atom*(*Eq* (*b*−*Const Bv*))) ([] @ *L1*) = *eval* (*liftFm 0 z* (*Atom*(*Eq* (*b*−*Const Bv*)))) ([] @ *L2* @ *L1*)
by (*metis eval-liftFm-helper lengthl2 list.size*(*3*))
then have (*insertion* (*nth-default 0* (*L2* @ *L1*)) (*liftPoly 0 z* (*b* − *Const Bv*)) = *0*)
apply(*simp add*: *insertion-sub insertion-const*)
using *Atom*(*6*) **unfolding** *L1-def*
by (*metis list-update-id*)
then have *insertion* (*nth-default 0* (*L2* @ *L1*)) (*liftPoly 0 z b*) = *Bv*
using *lift-minus* **by** *blast*
then have *a2* : ∀ *x*. *insertion* (*nth-default 0* (*L*[*var* + *z* := *x*])) (*liftPoly 0 z b*) = *Bv*
unfolding *L-def* **using** *Atom*(*6*) *L1-def*
by (*metis L-def add-diff-cancel-right′ append.simps*(*1*) *lengthl2 lift-insertion list.size*(*3*) *list-update-append not-add-less2*)
have *eval* (*Atom*(*Eq* (*c*−*Const Cv*))) ([] @ *L1*) = *eval* (*liftFm 0 z* (*Atom*(*Eq* (*c*−*Const Cv*)))) ([] @ *L2* @ *L1*)
by (*metis eval-liftFm-helper lengthl2 list.size*(*3*))
then have (*insertion* (*nth-default 0* (*L2* @ *L1*)) (*liftPoly 0 z* (*c* − *Const Cv*)) = *0*)
apply(*simp add*: *insertion-sub insertion-const*)
using *Atom*(*7*) **unfolding** *L1-def*
by (*metis list-update-id*)
then have *insertion* (*nth-default 0* (*L2* @ *L1*)) (*liftPoly 0 z c*) = *Cv*
using *lift-minus* **by** *blast*
then have *a3* : ∀ *x*. *insertion* (*nth-default 0* (*L*[*var* + *z* := *x*])) (*liftPoly 0 z c*) = *Cv*
unfolding *L-def* **by** (*metis freeC list-update-id not-contains-insertion not-in-lift*)
have *eval* (*Atom*(*Eq* (*d*−*Const Dv*))) ([] @ *L1*) = *eval* (*liftFm 0 z* (*Atom*(*Eq* (*d*− *Const Dv*)))) ([] @ *L2* @ *L1*)
by (*metis eval-liftFm-helper lengthl2 list.size*(*3*))
then have (*insertion* (*nth-default 0* (*L2* @ *L1*)) (*liftPoly 0 z* (*d* − *Const Dv*)) = *0*)
apply(*simp add*: *insertion-sub insertion-const*)
using *Atom*(*8*) **unfolding** *L1-def*
by (*metis list-update-id*)
then have *insertion* (*nth-default 0* (*L2* @ *L1*)) (*liftPoly 0 z d*) = *Dv*
using *lift-minus* **by** *blast*
then have *a4* : ∀ *x*. *insertion* (*nth-default 0* (*L*[*var* + *z* := *x*])) (*liftPoly 0 z d*) = *Dv*
unfolding *L-def*

by (*metis Atom(8) L1-def L-def add-diff-cancel-right' append.simps(1) lengthl2 lift-insertion list.size(3) list-update-append not-add-less2*)
then show *?case* **apply**(*simp*)
using *quadratic-sub*[*OF Atom(4) Atom(1) Atom(2) not-in-lift*[*OF Atom(3)*], *of* (*liftPoly 0 z a*) *Av* (*liftPoly 0 z b*) *Bv* (*liftPoly 0 z d*) *x*
, *OF a1 a2 a3 a4*] **.**
next
case (*And F1 F2*)
then show *?case* **by** *auto*
next
case (*Or F1 F2*)
then show *?case* **by** *auto*
next
case (*Neg F*)
then show *?case* **by** *auto*
next
case (*ExQ F*)
have *lengthG* : *var* + (*z* + *1*) < *length* (*x*#*L*) **for** *x* **using** *ExQ*(*5*) **by** *auto*
have *forall* : ∀ *x*. *insertion* (*nth-default 0* ((*drop z L*)[*var* := *x*])) *a* = *Av* ⟹
∀ *x*. *insertion* (*nth-default 0* ((*drop* (*z* + *1*) (*x1* # *L*))[*var* := *x*])) *a* = *Av* **for** *x1 a Av*
by *auto*
have *l* : *x* # *L*[*var* + *z* := *v*] = ((*x*#*L*)[*var*+(*z*+*1*):=*v*]) **for** *x v*
by *auto*
have *eval* (*ExQ F*) (*L*[*var* + *z* := (*Av* + *Bv* ∗ *sqrt Cv*) / *Dv*]) =
(∃ *x*. *eval F* (*x* # *L*[*var* + *z* := (*Av* + *Bv* ∗ *sqrt Cv*) / *Dv*]))
by(*simp*)
also have ... = (∃ *x*. *eval*
(*liftmap*
(λ*x*. *quadratic-sub* (*var* + *x*) (*liftPoly 0 x a*) (*liftPoly 0 x b*) (*liftPoly 0 x c*) (*liftPoly 0 x d*))
F (*z* + *1*))
(*x* # *L*[*var* + *z* := *sqrt Cv*]))
apply(*rule ex-cong1*)
unfolding *l*
using *ExQ*(*1*)[*OF ExQ*(*2*) *ExQ*(*3*) *ExQ*(*4*) *lengthG forall*[*OF ExQ*(*6*)] *forall*[*OF ExQ*(*7*)] *forall*[*OF ExQ*(*8*)] *forall*[*OF ExQ*(*9*)]]
unfolding *quadratic-sub-fm-helper.simps liftmap.simps*
by *simp*
also have ... = *eval* (*quadratic-sub-fm-helper var a b c d* (*ExQ F*) *z*) (*L*[*var* + *z* := *sqrt Cv*])
unfolding *quadratic-sub-fm-helper.simps liftmap.simps eval.simps* **by** *auto*
finally show *?case* **by** *simp*
next
case (*AllQ F*)
have *lengthG* : *var* + (*z* + *1*) < *length* (*x*#*L*) **for** *x* **using** *AllQ*(*5*) **by** *auto*
have *forall* : ∀ *x*. *insertion* (*nth-default 0* ((*drop z L*)[*var* := *x*])) *a* = *Av* ⟹
∀ *x*. *insertion* (*nth-default 0* ((*drop* (*z* + *1*) (*x1* # *L*))[*var* := *x*])) *a* = *Av* **for** *x1 a Av*

```
    by auto
  have l : x # L[var + z := v] = ((x#L)[var+(z+1):=v]) for x v
    by auto
  have eval (AllQ F) (L[var + z := (Av + Bv * sqrt Cv) / Dv]) =
    (∀x. eval F (x # L[var + z := (Av + Bv * sqrt Cv) / Dv]))
    by(simp)
  also have ... = (∀x. eval
          (liftmap
            (λx. quadratic-sub (var + x) (liftPoly 0 x a) (liftPoly 0 x b) (liftPoly 0 x
c) (liftPoly 0 x d))
            F (z + 1))
          (x # L[var + z := sqrt Cv]))
    apply(rule all-cong1)
    unfolding l
   using AllQ(1)[OF AllQ(2) AllQ(3) AllQ(4) lengthG forall[OF AllQ(6)] forall[OF
AllQ(7)] forall[OF AllQ(8)] forall[OF AllQ(9)]]
    unfolding quadratic-sub-fm-helper.simps liftmap.simps
    by simp
  also have ... = eval (quadratic-sub-fm-helper var a b c d (AllQ F) z) (L[var +
z := sqrt Cv])
    unfolding quadratic-sub-fm-helper.simps liftmap.simps eval.simps by auto
  finally show ?case by simp
next
  case (ExN x1 F)
  have list : ⋀l. length l=x1 ⟹ ((drop (z + x1) l @ drop (z + x1 − length l)
L)) = ((drop z L))
    by auto
  have map : ⋀ z L. eval (liftmap (λx A. (quadratic-sub (var + x) (liftPoly 0 x a)
(liftPoly 0 x b) (liftPoly 0 x c) (liftPoly 0 x d) A)) F (z + x1))
       L = eval (liftmap (λx A. (quadratic-sub (var + x1 + x) (liftPoly 0 (x+x1)
a) (liftPoly 0 (x+x1) b) (liftPoly 0 (x+x1) c) (liftPoly 0 (x+x1) d) A)) F z)
      L
    apply(induction F) apply(simp-all add:add.commute add.left-commute)
    apply force
    apply force
    by (metis (mono-tags, lifting) ab-semigroup-add-class.add-ac(1))+
  show ?case apply simp apply(rule ex-cong1)
    subgoal for l
      using map[of z] list[of l] ExN(1)[OF ExN(2−4), of z+x1 l@L] ExN(5−9)
list-update-append
      apply auto
      by (simp add: list-update-append) +
    done
next
  case (AllN x1 F)
  have list : ⋀l. length l=x1 ⟹ ((drop (z + x1) l @ drop (z + x1 − length l)
L)) = ((drop z L))
    by auto
  have map : ⋀ z L. eval (liftmap (λx A. (quadratic-sub (var + x) (liftPoly 0 x a)
```

(*liftPoly 0 x b*) (*liftPoly 0 x c*) (*liftPoly 0 x d*) *A*)) *F* (*z* + *x1*))
L = *eval* (*liftmap* (λ*x A*. (*quadratic-sub* (*var* + *x1* + *x*) (*liftPoly 0* (*x*+*x1*) *a*) (*liftPoly 0* (*x*+*x1*) *b*) (*liftPoly 0* (*x*+*x1*) *c*) (*liftPoly 0* (*x*+*x1*) *d*) *A*)) *F z*)
L
apply(*induction F*) **apply**(*simp-all add*:*add.commute add.left-commute*)
apply *force*
apply *force*
by (*metis* (*mono-tags*, *lifting*) *ab-semigroup-add-class.add-ac*(*1*))+
show *?case* **apply** *simp* **apply**(*rule all-cong1*)
subgoal for *l*
using *map*[*of z*] *list*[*of l*] *AllN*(*1*)[*OF AllN*(*2*−*4*), *of z*+*x1 l*@*L*] *AllN*(*5*−*9*)
apply *auto*
by (*simp add*: *list-update-append*) +
done
qed

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*))
unfolding *quadratic-sub-fm.simps* **using** *quadratic-sub-fm-helper*[*OF assms*(*2*) *assms*(*3*) *assms*(*4*), *of 0 L a Av b Bv d F*] *assms*(*1*) *assms*(*5*) *assms*(*6*) *assms*(*7*) *assms*(*8*)
by (*simp add*: *lLength*)
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*)) =
((∃ *x*. *eval* (*elimVar var L F* (*Eq p*)) (*xs @ x # Γ*)))∨ (∀ *x*. *aEval* (*Eq p*) (*xs @ x # Γ*)))
proof−

{ fix *x*
define *A* **where** *A* = (*isolate-variable-sparse p var 2*)
define *B* **where** *B* = (*isolate-variable-sparse p var 1*)
define *C* **where** *C* = (*isolate-variable-sparse p var 0*)
have *freeA* : *var* ∉ *vars A*
unfolding *A-def*
by (*simp add*: *not-in-isovarspar*)
have *freeB* : *var* ∉ *vars B*
unfolding *B-def*
by (*simp add*: *not-in-isovarspar*)
have *freeC* : *var* ∉ *vars C*
unfolding *C-def*
by (*simp add*: *not-in-isovarspar*)
assume *eval* (*list-conj* (*map fm.Atom L @ F*)) (*xs @ x # Γ*)
then have *h* : (∀ *a*∈*set L*. *aEval a* (*xs @ x # Γ*)) ∧ (∀ *f*∈*set F*. *eval f* (*xs @ x # Γ*))
apply(*simp add*:*eval-list-conj*)
by (*meson Un-iff eval.simps*(*1*) *image-eqI*)
define *X* **where** *X*=*xs*@*x*#Γ
have *Xlength* : *length X* > *var*
using *X-def hlength* **by** *auto*
define *Aval* **where** *Aval* = *insertion* (*nth-default 0* (*list-update X var x*)) *A*
define *Bval* **where** *Bval* = *insertion* (*nth-default 0* (*list-update X var x*)) *B*
define *Cval* **where** *Cval* = *insertion* (*nth-default 0* (*list-update X var x*)) *C*
have *hinsert* : (*xs @ x # Γ*)[*var* := *x*] = (*xs @ x #Γ*)
using *hlength* **by** *auto*
have *allAval* : ∀ *x*. *insertion* (*nth-default 0* (*xs @ x # Γ*)) *A* = *Aval*
using *Aval-def*
using *not-contains-insertion*[**where** *var*=*var*, **where** *p* = *A*, *OF freeA*, **where** *L* = *xs @ x #Γ*, **where** *x*=*x*, **where** *val*=*Aval*]
unfolding *X-def hinsert* **using** *hlength* **by** *auto*
have *allBval* : ∀ *x*. *insertion* (*nth-default 0* (*xs @ x # Γ*)) *B* = *Bval*
using *Bval-def*
using *not-contains-insertion*[**where** *var*=*var*, **where** *p* = *B*, *OF freeB*, **where** *L* = *xs @ x #Γ*, **where** *x*=*x*, **where** *val*=*Bval*]
unfolding *X-def hinsert* **using** *hlength* **by** *auto*
have *allCval* : ∀ *x*. *insertion* (*nth-default 0* (*xs @ x # Γ*)) *C* = *Cval*
using *Cval-def*
using *not-contains-insertion*[**where** *var*=*var*, **where** *p* = *C*, *OF freeC*, **where** *L* = *xs @ x #Γ*, **where** *x*=*x*, **where** *val*=*Cval*]
unfolding *X-def hinsert* **using** *hlength* **by** *auto*
have *insertion-p* : *insertion* (*nth-default 0 X*) *p* = *0*

```
    using in-list h aEval.simps(1) X-def by fastforce
  have express-p : p = A * Var var ^ 2 + B * Var var + C
    using express-poly[OF low-pow] unfolding A-def B-def C-def
    by fastforce
  have insertion-p' : Aval *x^2+Bval *x+Cval = 0
   using express-p insertion-p unfolding Aval-def Bval-def Cval-def X-def hinsert
    apply(simp add: insertion-add insertion-mult insertion-pow)
    using  insertion-var by (metis X-def Xlength hinsert)
  have biglemma :
    ((Aval = 0 ∧
     Bval ≠ 0 ∧
     (∀f∈set L. aEval (linear-substitution var (−C) B f) (xs @ x # Γ)) ∧
     (∀f∈set F. eval (linear-substitution-fm var (−C) B f) (xs @ x # Γ)) ∨
     Aval ≠ 0 ∧
     insertion (nth-default 0 (xs @ x # Γ)) 4 *
     Aval *
     Cval
     ≤ (Bval)^2 ∧
     ((∀f∈set L. eval (quadratic-sub var (− B) 1 (B^2 − 4 * A * C) (2 * A) f)
(xs @ x # Γ))∧
      (∀f∈set F. eval (quadratic-sub-fm var (− B) 1 (B^2 − 4 * A * C) (2 * A)
f) (xs @ x # Γ)) ∨
      (∀f∈set L. eval (quadratic-sub var (− B) (−1) (B^2 − 4 * A * C) (2 * A)
f) (xs @ x # Γ)) ∧
      (∀f∈set F. eval (quadratic-sub-fm var (− B) (−1) (B^2 − 4 * A * C) (2 *
A) f) (xs @ x # Γ))) ∨
     Aval = 0 ∧
     Bval = 0 ∧
     Cval = 0))
  proof(cases Aval=0)
    case True
    then have aval0 : Aval=0 by simp
    show ?thesis proof(cases Bval=0)
      case True
      then have bval0 :  Bval=0 by simp
      have h : eval (list-conj (map fm.Atom L @ F)) (xs @ x # Γ)
        using hlength h unfolding X-def
        using ‹eval (list-conj (map fm.Atom L @ F)) (xs @ x # Γ)› by blast
      show ?thesis proof(cases Cval=0)
        case True
        show ?thesis
          by(simp add:aval0 True bval0)
      next
        case False
        show ?thesis
          using insertion-p' aval0 bval0 False by(simp)
      qed
    next
      case False
```

```
      have bh : insertion (nth-default 0 (X[var := − Cval / Bval])) B = Bval
        using allBval unfolding X-def
        using Bval-def X-def freeB not-contains-insertion by blast
      have ch : insertion (nth-default 0 (X[var := − Cval / Bval])) C = Cval
        using allCval unfolding X-def
        using Cval-def X-def freeC not-contains-insertion by blast
      have xh : x=−Cval/Bval
      proof−
        have Bval∗x+Cval = 0
          using insertion-p′ aval0
          by simp
        then show ?thesis using False
          by (smt nonzero-mult-div-cancel-left)
      qed
      have freecneg : var ∉ vars (−C) using freeC not-in-neg by auto
      have h1: (∀ a∈set L. aEval (linear-substitution var (−C) (B) a) (X[var :=
x]))
          using h xh Bval-def Cval-def False LinearCase.linear[OF Xlength False
freecneg freeB, of −Cval] freeB freeC freecneg
        by (metis X-def hinsert insertion-neg)
      have h2 : ∀ f∈set F. eval (linear-substitution-fm var (−C) B f) (X[var :=
x])
        using h xh Bval-def Cval-def False LinearCase.linear-fm[OF Xlength False
freecneg freeB, of −Cval] freeB freeC
        by (metis X-def hinsert insertion-neg)
      show ?thesis using h1 h2 apply(simp add:aval0 False)
        using X-def hlength
        using hinsert by auto
    qed
  next
    case False
    then have aval0 : Aval ≠0 by simp
    have h4 : insertion (nth-default 0 (X[var := x])) 4 = 4
     using insertion-const[where f = (nth-default 0 (X[var := x])), where c=4]
    by (metis MPoly-Type.insertion-one insertion-add numeral-Bit0 one-add-one)
    show ?thesis proof(cases 4 ∗ Aval ∗ Cval ≤ Bval^2)
      case True
      have h1a : var∉vars(−B)
        by(simp add: freeB not-in-neg)
      have h1b : var∉vars(1::real mpoly)
        using isolate-var-one not-in-isovarspar by blast
      have h1c : var∉vars(−1::real mpoly)
        by(simp add: h1b not-in-neg)
      have h1d : var∉vars(4::real mpoly)
        by (metis h1b not-in-add numeral-Bit0 one-add-one)
      have h1e : var∉vars(B^2−4∗A∗C)
        by(simp add: freeB h1d freeA freeC not-in-mult not-in-pow not-in-sub)
      have h1f : var∉vars(2::real mpoly)
        using h1b not-in-add by fastforce
```

have *h1g* : *var∉vars*(*2∗A*)
by(*simp add*: *freeA h1f not-in-mult*)
have *h1h* : *freeIn var* (*quadratic-sub var* (*−B*) (*1*) (*B^2−4∗A∗C*) (*2∗A*) *a*)
using *free-in-quad h1a h1b h1e h1g* **by** *blast*
have *h1i* : *freeIn var* (*quadratic-sub var* (*−B*) (*−1*) (*B^2−4∗A∗C*) (*2∗A*) *a*)
using *free-in-quad h1a h1c h1e h1g* **by** *blast*
have *h2* : *2∗Aval ≠ 0* **using** *aval0* **by** *auto*
have *h3* : *0 ≤* (*Bval^2−4∗Aval∗Cval*) **using** *True* **by** *auto*
have *h4a* : *var ∉ vars 4*
by (*metis monom-numeral notInKeys-notInVars not-in-add not-in-isovarspar not-in-pow one-add-one power.simps*(*1*) *rel-simps*(*76*) *vars-monom-keys*)
have *h4* : *var ∉ vars* (*B^2−4∗A∗C*) **by**(*simp add*: *h4a freeA freeB freeC not-in-pow not-in-mult not-in-sub*)
have *h5* : *∀ x. insertion* (*nth-default 0* (*list-update X var x*)) (*−B*) = *−Bval*
using *allBval* **apply**(*simp add*: *insertion-neg*)
by (*simp add*: *B-def Bval-def insertion-isovarspars-free*)
have *h6* : *∀ x. insertion* (*nth-default 0* (*list-update X var x*)) *1 = 1* **by** *simp*
have *h6a* : *∀ x. insertion* (*nth-default 0* (*list-update X var x*)) (*−1*) = (*−1*) **using** *h6* **by** (*simp add*: *insertion-neg*)
have *h7a* : *∀ x. insertion* (*nth-default 0* (*list-update X var x*)) *4 = 4* **by** (*metis h6 insertion-add numeral-Bit0 one-add-one*)
have *h7b* : *var ∉ vars*(*4∗A∗C*) **using** *freeA freeC* **by** (*simp add*: *h4a not-in-mult*)
have *h7c* : *var ∉ vars*(*B^2*) **using** *freeB not-in-pow* **by** *auto*
have *h7* : *∀ x. insertion* (*nth-default 0* (*list-update X var x*)) (*B^2−4∗A∗C*) = (*Bval^2−4∗Aval∗Cval*)
using *h7a allAval allBval allCval* **unfolding** *X-def* **using** *hlength*
apply (*simp add*: *insertion-mult insertion-sub power2-eq-square*)
by (*metis A-def Aval-def Bval-def C-def Cval-def X-def freeB insertion-isovarspars-free not-contains-insertion*)
have *h8a* : *∀ x. insertion* (*nth-default 0* (*list-update X var x*)) *2 = 2* **by** (*metis h6 insertion-add one-add-one*)
have *h8* : *∀ x. insertion* (*nth-default 0* (*list-update X var x*)) (*2∗A*) = (*2∗Aval*)
apply(*simp add*: *allAval h8a insertion-mult*)
by (*simp add*: *A-def Aval-def insertion-isovarspars-free*)

have *h1* : $- Bval^2 + 4 * Aval * Cval \leq 0$
using *True* **by** *simp*
have *xh* : $x = (- Bval + sqrt\ (Bval^2 - 4 * Aval * Cval)) / (2 * Aval) \vee x = (- Bval - sqrt\ (Bval^2 - 4 * Aval * Cval)) / (2 * Aval)$
using *insertion-p′ aval0 h1*
discriminant-iff **unfolding** *discrim-def* **by** *blast*
have *p1* : $x = (- Bval + sqrt\ (Bval^2 - 4 * Aval * Cval)) / (2 * Aval) \Longrightarrow$
((*∀ a∈ set L. eval* (*quadratic-sub var* (*−B*) *1* (*B^2−4∗A∗C*) (*2∗A*) *a*) (*X*[*var* := *x*]))

$\wedge$(∀ a∈ *set F*. *eval* (*quadratic-sub-fm var* (−*B*) *1* (*B*^*2*−*4*∗*A*∗*C*) (*2*∗*A*) *a*) (*X*[*var* := *x*])))

proof−

assume *x-def* : $x = (-\ Bval + sqrt\ (Bval^2 - 4 * Aval * Cval))\ /\ (2 * Aval)$

then have *h* : (∀ *a*∈*set L*. *aEval a* (*X*[*var* := (− *Bval* + *sqrt* (*Bval*^{2} − *4* ∗ *Aval* ∗ *Cval*)) / (*2* ∗ *Aval*)])) ∧ (∀ *f*∈*set F*. *eval f* (*X*[*var* := (− *Bval* + *sqrt* (*Bval*^{2} − *4* ∗ *Aval* ∗ *Cval*)) / (*2* ∗ *Aval*)]))

using *h*

using *X-def hinsert* **by** *auto*

{ fix *a*

assume *in-list* : *a*∈ *set L*

have *eval* (*quadratic-sub var* (− *B*) *1* (B^2 − *4* ∗ *A* ∗ *C*) (*2* ∗ *A*) *a*) (*X*[*var* := *x*])

using *free-in-quad*[**where** *a*=−*B*,**where** *b*=*1*, **where** *c*=(*B*^*2*−*4*∗*A*∗*C*), **where** *d*=*2*∗*A*,**where** *var*=*var*,*OF h1a h1b h1e h1g*]

using *quadratic-sub*[**where** *a*=−*B*,**where** *b*=*1*, **where** *c*=(*B*^*2*−*4*∗*A*∗*C*), **where** *d*=*2*∗*A*,**where** *var*=*var*, **where** *L*=*X*, *OF Xlength*,

where *Dv*=*2*∗*Aval*, *OF h2*, **where** *Cv*=(*Bval*^*2*−*4*∗*Aval*∗*Cval*), *OF h3*, **where** *Av*=−*Bval*, *OF h4 h5*, **where** *Bv*=*1*, *OF h6 h7 h8*]

h in-list

using *var-not-in-eval* **by** *fastforce*

}

then have *left* : (∀ *a*∈*set L*. *eval* (*quadratic-sub var* (− *B*) *1* (B^2 − *4* ∗ *A* ∗ *C*) (*2* ∗ *A*) *a*) (*X*[*var* := *x*]))

by *simp*

{ fix *a*

assume *in-list* : *a*∈ *set F*

have *eval* (*quadratic-sub-fm var* (− *B*) *1* (B^2 − *4* ∗ *A* ∗ *C*) (*2* ∗ *A*) *a*) (*X*[*var* := *x*])

using *free-in-quad-fm*[**where** *a*=−*B*,**where** *b*=*1*, **where** *c*=(*B*^*2*−*4*∗*A*∗*C*), **where** *d*=*2*∗*A*,**where** *var*=*var*,*OF h1a h1b h1e h1g*]

using *quadratic-sub-fm*[**where** *a*=−*B*,**where** *b*=*1*, **where** *c*=(*B*^*2*−*4*∗*A*∗*C*), **where** *d*=*2*∗*A*,**where** *var*=*var*, **where** *L*=*X*, *OF Xlength*,

where *Dv*=*2*∗*Aval*, *OF h2*, **where** *Cv*=(*Bval*^*2*−*4*∗*Aval*∗*Cval*), *OF h3*, **where** *Av*=−*Bval*, *OF h4 h5*, **where** *Bv*=*1*, *OF h6 h7 h8*]

h in-list

using *var-not-in-eval* **by** *fastforce*

}

then have *right* : (∀ *a*∈*set F*. *eval* (*quadratic-sub-fm var* (− *B*) *1* (B^2 − *4* ∗ *A* ∗ *C*) (*2* ∗ *A*) *a*) (*X*[*var* := *x*]))

by *simp*

show *?thesis*

using *right left* **by** *simp*

qed

have *p2* : $x = (-\ Bval - sqrt\ (Bval^2 - 4 * Aval * Cval))\ /\ (2 * Aval) \Longrightarrow$

$((\forall\, a{\in}\ set\ L.\ eval\ (quadratic\text{-}sub\ var\ (-B)\ (-1)\ (B\hat{}2{-}4{*}A{*}C)\ (2{*}A)\ a)\ (X[var := x])$
$\wedge(\forall\, a{\in}\ set\ F.\ eval\ (quadratic\text{-}sub\text{-}fm\ var\ (-B)\ (-1)\ (B\hat{}2{-}4{*}A{*}C)\ (2{*}A)\ a)\ (X[var := x])))$

proof −

assume *x-def* : $x = (-\ Bval - sqrt\ (Bval^2 - 4 * Aval * Cval))\ /\ (2 * Aval)$

then have h : $(\forall\, a{\in}set\ L.\ aEval\ a\ (X[var := (-\ Bval - sqrt\ (Bval^2 - 4 * Aval * Cval))\ /\ (2 * Aval)])) \wedge (\forall f{\in}set\ F.\ eval\ f\ (X[var := (-\ Bval - sqrt\ (Bval^2 - 4 * Aval * Cval))\ /\ (2 * Aval)]))$

using h

using *X-def hinsert* **by** *auto*

then have $(\forall\, a{\in}set\ L.\ aEval\ a\ (X[var := (-\ Bval - sqrt\ (Bval^2 - 4 * Aval * Cval))\ /\ (2 * Aval)])) \wedge (\forall f{\in}set\ F.\ eval\ f\ (X[var := (-\ Bval - sqrt\ (Bval^2 - 4 * Aval * Cval))\ /\ (2 * Aval)]))$

using h **by** *simp*

{ fix a

assume *in-list* : $a{\in}\ set\ L$

have *eval* $(quadratic\text{-}sub\ var\ (-\ B)\ (-1)\ (B^2 - 4 * A * C)\ (2 * A)\ a)\ (X[var := x])$

using *free-in-quad*[**where** $a{=}{-}B$,**where** $b{=}{-}1$, **where** $c{=}(B\hat{}2{-}4{*}A{*}C)$, **where** $d{=}2{*}A$,**where** $var{=}var$,*OF h1a h1c h1e h1g*]

using *quadratic-sub*[**where** $a{=}{-}B$,**where** $b{=}{-}1$, **where** $c{=}(B\hat{}2{-}4{*}A{*}C)$, **where** $d{=}2{*}A$,**where** $var{=}var$, **where** $L{=}X$, *OF Xlength*,

where $Dv{=}2{*}Aval$, *OF h2*, **where** $Cv{=}(Bval\hat{}2{-}4{*}Aval{*}Cval)$, *OF h3*, **where** $Av{=}{-}Bval$, *OF h4 h5*, **where** $Bv{=}{-}1$, *OF h6a h7 h8*]

h in-list

using *var-not-in-eval* **by** *fastforce*

}

then have *left* : $(\forall\, a{\in}set\ L.\ eval\ (quadratic\text{-}sub\ var\ (-\ B)\ (-1)\ (B^2 - 4 * A * C)\ (2 * A)\ a)\ (X[var := x]))$

by *simp*

{ fix a

assume *in-list* : $a{\in}\ set\ F$

have *eval* $(quadratic\text{-}sub\text{-}fm\ var\ (-\ B)\ (-1)\ (B^2 - 4 * A * C)\ (2 * A)\ a)\ (X[var := x])$

using *free-in-quad-fm*[**where** $a{=}{-}B$,**where** $b{=}{-}1$, **where** $c{=}(B\hat{}2{-}4{*}A{*}C)$, **where** $d{=}2{*}A$,**where** $var{=}var$,*OF h1a h1c h1e h1g*]

using *quadratic-sub-fm*[**where** $a{=}{-}B$,**where** $b{=}{-}1$, **where** $c{=}(B\hat{}2{-}4{*}A{*}C)$, **where** $d{=}2{*}A$,**where** $var{=}var$, **where** $L{=}X$, *OF Xlength*,

where $Dv{=}2{*}Aval$, *OF h2*, **where** $Cv{=}(Bval\hat{}2{-}4{*}Aval{*}Cval)$, *OF h3*, **where** $Av{=}{-}Bval$, *OF h4 h5*, **where** $Bv{=}{-}1$, *OF h6a h7 h8*]

h in-list

using *var-not-in-eval* **by** *fastforce*

}
then have *right* : ($\forall$ *a*$\in$*set F*. *eval* (*quadratic-sub-fm var* (− *B*) (−*1*) (*B*^{2} − *4* ∗ *A* ∗ *C*) (*2* ∗ *A*) *a*) (*X*[*var* := *x*]))
by *simp*
show *?thesis*
using *right left* **by** *simp*
qed
have *subst4* : *insertion* (*nth-default 0* (*xs* @ *x* # Γ)) *4* = *4* **using** *h7a hlength X-def* **by** *auto*
have *disj*: ($\forall$ *a*$\in$*set L*. *eval* (*quadratic-sub var* (− *B*) *1* (*B*^{2} − *4* ∗ *A* ∗ *C*) (*2* ∗ *A*) *a*) (*xs* @ *x* # Γ)) ∧
($\forall$ *a*$\in$*set F*. *eval* (*quadratic-sub-fm var* (− *B*) *1* (*B*^{2} − *4* ∗ *A* ∗ *C*) (*2* ∗ *A*) *a*) (*xs* @ *x* # Γ)) ∨
($\forall$ *a*$\in$*set L*. *eval* (*quadratic-sub var* (− *B*) (−*1*) (*B*^{2} − *4* ∗ *A* ∗ *C*) (*2* ∗ *A*) *a*) (*xs* @ *x* # Γ)) ∧
($\forall$ *a*$\in$*set F*. *eval* (*quadratic-sub-fm var* (− *B*) (−*1*) (*B*^{2} − *4* ∗ *A* ∗ *C*) (*2* ∗ *A*) *a*) (*xs* @ *x* # Γ))
using *xh p1 p2*
unfolding *X-def hinsert* **by** *blast*
show *?thesis* **apply**(*simp add*: *aval0 True h7a subst4*) **using** *disj*
unfolding *X-def hinsert* **by** *auto*
next
case *False*
then have *det* : *0* < − *Bval*^{2} + *4* ∗ *Aval* ∗ *Cval*
by *simp*
show *?thesis* **apply**(*simp add*: *aval0 False h4*) **using** *discriminant-negative* **unfolding** *discrim-def*
using *insertion-p'*
using *aval0 det* **by** *auto*
qed
qed
have ($\exists$ *x*.
(*insertion* (*nth-default 0* (*xs* @ *x* # Γ)) *A* = *0* ∧
insertion (*nth-default 0* (*xs* @ *x* # Γ)) *B* ≠ *0* ∧
($\forall$ *f*$\in$*set L*. *aEval* (*linear-substitution var* (−*C*) (*B*) *f*) (*xs* @ *x* # Γ)) ∧
($\forall$ *f*$\in$*set F*. *eval* (*linear-substitution-fm var* (−*C*) *B f*) (*xs* @ *x* # Γ)) ∨
insertion (*nth-default 0* (*xs* @ *x* # Γ)) *A* ≠ *0* ∧
insertion (*nth-default 0* (*xs* @ *x* # Γ)) *4* ∗
insertion (*nth-default 0* (*xs* @ *x* # Γ)) *A* ∗
insertion (*nth-default 0* (*xs* @ *x* # Γ)) *C*
≤ (*insertion* (*nth-default 0* (*xs* @ *x* # Γ)) *B*)^2 ∧
(($\forall$ *f*$\in$*set L*. *eval* (*quadratic-sub var* (− *B*) *1* (*B*^{2} − *4* ∗ *A* ∗ *C*) (*2* ∗ *A*) *f*) (*xs* @ *x* # Γ))∧
($\forall$ *f*$\in$*set F*. *eval* (*quadratic-sub-fm var* (− *B*) *1* (*B*^{2} − *4* ∗ *A* ∗ *C*) (*2* ∗ *A*) *f*) (*xs* @ *x* # Γ)) ∨
($\forall$ *f*$\in$*set L*. *eval* (*quadratic-sub var* (− *B*) (−*1*) (*B*^{2} − *4* ∗ *A* ∗ *C*) (*2* ∗ *A*) *f*) (*xs* @ *x* # Γ)) ∧

```
      (∀f∈set F. eval (quadratic-sub-fm var (− B) (−1) (B^2 − 4 ∗ A ∗ C) (2 ∗
A) f) (xs @ x # Γ)))) ∨
      (Aval = 0 ∧
      Bval = 0 ∧
      Cval = 0))
    apply(rule exI[where x=x])
    using biglemma
    using allAval allBval allCval unfolding A-def B-def C-def Aval-def Bval-def
Cval-def X-def hinsert
    by auto
  then obtain x where x : (insertion (nth-default 0 (xs @ x # Γ)) A = 0 ∧
      insertion (nth-default 0 (xs @ x # Γ)) B ≠ 0 ∧
      (∀f∈set L. aEval (linear-substitution var (−C) (B) f) (xs @ x # Γ)) ∧
      (∀f∈set F. eval (linear-substitution-fm var (−C) B f) (xs @ x # Γ)) ∨
      insertion (nth-default 0 (xs @ x # Γ)) A ≠ 0 ∧
      insertion (nth-default 0 (xs @ x # Γ)) 4 ∗
      insertion (nth-default 0 (xs @ x # Γ)) A ∗
      insertion (nth-default 0 (xs @ x # Γ)) C
      ≤ (insertion (nth-default 0 (xs @ x # Γ)) B)^2 ∧
      ((∀f∈set L. eval (quadratic-sub var (− B) 1 (B^2 − 4 ∗ A ∗ C) (2 ∗ A) f)
(xs @ x # Γ))∧
      (∀f∈set F. eval (quadratic-sub-fm var (− B) 1 (B^2 − 4 ∗ A ∗ C) (2 ∗ A)
f) (xs @ x # Γ)) ∨
      (∀f∈set L. eval (quadratic-sub var (− B) (−1) (B^2 − 4 ∗ A ∗ C) (2 ∗ A)
f) (xs @ x # Γ)) ∧
      (∀f∈set F. eval (quadratic-sub-fm var (− B) (−1) (B^2 − 4 ∗ A ∗ C) (2 ∗
A) f) (xs @ x # Γ)))) ∨
      (Aval = 0 ∧
      Bval = 0 ∧
      Cval = 0) by auto
  have h : (∃x. eval (elimVar var L F (Eq p)) (xs @ x # Γ))∨(Aval = 0 ∧ Bval
= 0 ∧ Cval = 0)
  proof(cases (Aval = 0 ∧ Bval = 0 ∧ Cval = 0))
    case True
    then show ?thesis by simp
  next
    case False
    have (∃x. eval (elimVar var L F (Eq p)) (xs @ x # Γ))
      apply(rule exI[where x=x])
       apply(simp add: eval-list-conj insertion-mult insertion-sub insertion-pow
insertion-add
         del: quadratic-sub.simps linear-substitution.simps quadratic-sub-fm.simps
linear-substitution-fm.simps)
    unfolding A-def[symmetric] B-def[symmetric] C-def[symmetric] One-nat-def[symmetric]
X-def[symmetric]
      using hlength x
      by (auto simp add:False)
    then show ?thesis by auto
  qed
```

have ($\exists x.\ eval\ (elimVar\ var\ L\ F\ (Eq\ p))\ (xs\ @\ x\ \#\ \Gamma)) \vee (\forall x.\ aEval\ (Eq\ p)\ (xs @\ x \#\ \Gamma))$
proof(*cases* ($\exists x.\ eval\ (elimVar\ var\ L\ F\ (Eq\ p))\ (xs\ @\ x\ \#\ \Gamma)$))
case *True*
then show *?thesis* **by** *auto*
next
case *False*
then have ($Aval = 0 \wedge Bval = 0 \wedge Cval = 0$)
using *h* **by** *auto*
then have ($\forall x.\ aEval\ (Eq\ p)\ (xs\ @\ x\ \#\ \Gamma)$)
unfolding *express-p*
apply(*simp add*:*insertion-add insertion-mult insertion-pow*)
using *allAval allBval allCval* **by** *auto*
then show *?thesis* **by** *auto*
qed
}
then have *left* : ($\exists x.\ eval\ (list\text{-}conj\ (map\ fm.Atom\ L\ @\ F))\ (xs\ @\ x\ \#\ \Gamma)) \Longrightarrow$
$((\exists x.\ eval\ (elimVar\ var\ L\ F\ (Eq\ p))\ (xs\ @\ x\ \#\ \Gamma)) \vee (\forall x.\ aEval\ (Eq\ p)\ (xs @\ x \#\ \Gamma)))$
by *blast*

{
assume *hlength* : *length* (*xs*::*real list*) = *var*
define *A* **where** *A* = (*isolate-variable-sparse p var 2*)
define *B* **where** *B* = (*isolate-variable-sparse p var 1*)
define *C* **where** *C* = (*isolate-variable-sparse p var 0*)
have *freeA* : *var* $\notin$ *vars A*
unfolding *A-def*
by (*simp add*: *not-in-isovarspar*)
have *freeB* : *var* $\notin$ *vars B*
unfolding *B-def*
by (*simp add*: *not-in-isovarspar*)
have *freeC* : *var* $\notin$ *vars C*
unfolding *C-def*
by (*simp add*: *not-in-isovarspar*)
have *express-p* : $p = A*(Var\ var)\hat{}2+B*(Var\ var)+C$
using *express-poly*[*OF low-pow*] **unfolding** *A-def B-def C-def*
by *fastforce*
assume *h* : ($\exists x.\ (eval\ (elimVar\ var\ L\ F\ (Eq\ p))\ (list\text{-}update\ (xs@x\#\Gamma)\ var\ x))$)
fix *x*
define *X* **where** $X=xs@x\#\Gamma$
have *Xlength* : *length X* > *var*
using *X-def hlength* **by** *auto*
define *Aval* **where** *Aval* = *insertion* (*nth-default 0* (*list-update X var x*)) *A*
define *Bval* **where** *Bval* = *insertion* (*nth-default 0* (*list-update X var x*)) *B*
define *Cval* **where** *Cval* = *insertion* (*nth-default 0* (*list-update X var x*)) *C*
have *allAval* : $\forall x.$ *insertion* (*nth-default 0* (*list-update X var x*)) *A* = *Aval*
using *freeA Aval-def*

using *not-contains-insertion* **by** *blast*
have *allBval* : $\forall x.$ *insertion* (*nth-default 0* (*list-update X var x*)) *B* = *Bval*
using *freeB Bval-def*
using *not-contains-insertion* **by** *blast*
have *allCval* : $\forall x.$ *insertion* (*nth-default 0* (*list-update X var x*)) *C* = *Cval*
using *freeC Cval-def*
using *not-contains-insertion* **by** *blast*
assume (*eval* (*elimVar var L F* (*Eq p*)) (*list-update* (*xs*@*x*#Γ) *var x*))
then have *h* : (*eval* (*elimVar var L F* (*Eq p*)) (*list-update X var x*))
unfolding *X-def* **.**

have (*Aval* = *0* ∧ *Bval* ≠ *0* ∧
(∀ *f*∈(λ*a*. *Atom*(*linear-substitution var* (−*C*) *B a*)) ‘ *set L* ∪
linear-substitution-fm var (−*C*) *B* ‘
set F.
eval f (*X*[*var* := *x*])) ∨
Aval ≠ *0* ∧
insertion (*nth-default 0* (*X*[*var* := *x*])) *4* ∗ *Aval* ∗ *Cval* ≤ *Bval*^{2} ∧
((∀ *f*∈(*quadratic-sub var* (−*B*) *1* (*B*^*2*−*4*∗*A*∗*C*) (*2*∗*A*)) ‘
set L ∪
(*quadratic-sub-fm var* (−*B*) *1* (*B*^*2*−*4*∗*A*∗*C*) (*2*∗*A*)) ‘
set F.
eval f (*X*[*var* := *x*]))
∨(∀ *f*∈(*quadratic-sub var* (−*B*) (−*1*) (*B*^*2*−*4*∗*A*∗*C*) (*2*∗*A*)) ‘
set L ∪
(*quadratic-sub-fm var* (−*B*) (−*1*) (*B*^*2*−*4*∗*A*∗*C*) (*2*∗*A*)) ‘
set F.
eval f (*X*[*var* := *x*]))
))
unfolding *Aval-def Bval-def Cval-def A-def B-def C-def*
using *h* **by**(*simp add*: *eval-list-conj insertion-mult insertion-sub insertion-pow insertion-add insertion-var Xlength*)
then have *h* : (*Aval* = *0* ∧ *Bval* ≠ *0* ∧
((∀ *a*∈ *set L*. *aEval* (*linear-substitution var* (−*C*) *B a*) (*X*[*var* := *x*])) ∧
(∀ *a*∈ *set F*. *eval* (*linear-substitution-fm var* (−*C*) *B a*) (*X*[*var* := *x*]))) ∨
Aval ≠ *0* ∧ *insertion* (*nth-default 0* (*X*[*var* := *x*])) *4* ∗ *Aval* ∗ *Cval* ≤ *Bval*^{2} ∧
(((∀ *a*∈ *set L*. *eval* (*quadratic-sub var* (−*B*) *1* (*B*^*2*−*4*∗*A*∗*C*) (*2*∗*A*) *a*) (*X*[*var* := *x*]))
∧(∀ *a*∈ *set F*. *eval* (*quadratic-sub-fm var* (−*B*) *1* (*B*^*2*−*4*∗*A*∗*C*) (*2*∗*A*) *a*) (*X*[*var* := *x*])))
∨((∀ *a*∈ *set L*. *eval* (*quadratic-sub var* (−*B*) (−*1*) (*B*^*2*−*4*∗*A*∗*C*) (*2*∗*A*) *a*) (*X*[*var* := *x*]))
∧(∀ *a*∈ *set F*. *eval* (*quadratic-sub-fm var* (−*B*) (−*1*) (*B*^*2*−*4*∗*A*∗*C*) (*2*∗*A*) *a*) (*X*[*var* := *x*])))))

apply(*cases Aval* = *0*)

```
    apply auto
    by (meson Un-iff eval.simps(1) imageI)
  have h : (∃ x. ((∀ a∈set L . aEval a ((xs@x#Γ)[var := x])) ∧ (∀ f∈set F. eval
f ((xs@x#Γ)[var := x]))))∨(Aval=0∧Bval=0∧Cval=0)
  proof(cases Aval=0)
    case True
    then have aval0 : Aval=0
     by simp
    show ?thesis proof(cases Bval = 0)
     case True
     then have bval0 : Bval = 0 by simp
     show ?thesis proof(cases Cval=0)
      case True
      then show ?thesis using aval0 bval0 True by auto
     next
      case False
      then show ?thesis using h by(simp add:aval0 bval0 False)
     qed
    next
     case False
     have hb : insertion (nth-default 0 (X[var := − Cval / Bval])) B = Bval
      using allBval by simp
      have hc : insertion (nth-default 0 (X[var := − Cval / Bval])) (−C) =
−Cval
       using allCval
       by (simp add: insertion-neg)
     have freecneg : var∉vars(−C) using freeC not-in-neg by auto
     have p1 : (∀ a∈set L. aEval a ((xs @ x # Γ)[var := − Cval / Bval]))
      using h apply(simp add: False aval0)
      using linear[OF Xlength False freecneg freeB hc hb]
        list-update-length var-not-in-linear[OF freecneg freeB]
      unfolding X-def using hlength
      by (metis divide-minus-left)

     have p2 : (∀ a∈set F. eval a ((xs @ x # Γ)[var := − Cval / Bval]))
      using h apply(simp add: False aval0)
      using linear-fm[OF Xlength False freecneg freeB hc hb]
        list-update-length var-not-in-linear-fm[OF freecneg freeB]
      unfolding X-def using hlength var-not-in-eval
    by (metis divide-minus-left linear-substitution-fm.elims linear-substitution-fm-helper.elims)
     show ?thesis
      using p1 p2 hlength by fastforce
    qed
  next
    case False
    then have aval0 : Aval ≠ 0
     by simp
    have h4 : insertion (nth-default 0 (X[var := x])) 4 = 4
     using insertion-const[where f = (nth-default 0 (X[var := x])), where c=4]
```

by (*metis MPoly-Type.insertion-one insertion-add numeral-Bit0 one-add-one*)
show *?thesis* **proof**(*cases 4 * Aval * Cval ≤ Bval²*)
case *True*
then have *h1 : − Bval² + 4 * Aval * Cval ≤ 0*
by *simp*
have *h* : (((∀ *a*∈ *set L. eval* (*quadratic-sub var* (−*B*) *1* (*B^2−4∗A∗C*) (*2∗A*) *a*) (*X*[*var* := *x*]))
∧(∀ *a*∈ *set F. eval* (*quadratic-sub-fm var* (−*B*) *1* (*B^2−4∗A∗C*) (*2∗A*) *a*) (*X*[*var* := *x*])))
∨((∀ *a*∈ *set L. eval* (*quadratic-sub var* (−*B*) (−*1*) (*B^2−4∗A∗C*) (*2∗A*) *a*) (*X*[*var* := *x*]))
∧(∀ *a*∈ *set F. eval* (*quadratic-sub-fm var* (−*B*) (−*1*) (*B^2−4∗A∗C*) (*2∗A*) *a*) (*X*[*var* := *x*]))))
using *h* **by**(*simp add*: *h1 aval0*)
have *h1a* : *var*∉*vars*(−*B*)
by(*simp add*: *freeB not-in-neg*)
have *h1b* : *var*∉*vars*(*1*::*real mpoly*)
using *isolate-var-one not-in-isovarspar* **by** *blast*
have *h1c* : *var*∉*vars*(−*1*::*real mpoly*)
by(*simp add*: *h1b not-in-neg*)
have *h1d* : *var*∉*vars*(*4*::*real mpoly*)
by (*metis h1b not-in-add numeral-Bit0 one-add-one*)
have *h1e* : *var*∉*vars*(*B^2−4∗A∗C*)
by(*simp add*: *freeB h1d freeA freeC not-in-mult not-in-pow not-in-sub*)
have *h1f* : *var*∉*vars*(*2*::*real mpoly*)
using *h1b not-in-add* **by** *fastforce*
have *h1g* : *var*∉*vars*(*2∗A*)
by(*simp add*: *freeA h1f not-in-mult*)
have *h1h* : *freeIn var* (*quadratic-sub var* (−*B*) (*1*) (*B^2−4∗A∗C*) (*2∗A*) *a*)
using *free-in-quad h1a h1b h1e h1g* **by** *blast*
have *h1i* : *freeIn var* (*quadratic-sub var* (−*B*) (−*1*) (*B^2−4∗A∗C*) (*2∗A*) *a*)
using *free-in-quad h1a h1c h1e h1g* **by** *blast*
have *h2* : *2∗Aval ≠ 0* **using** *aval0* **by** *auto*
have *h3* : *0 ≤* (*Bval^2−4∗Aval∗Cval*) **using** *True* **by** *auto*
have *h4a* : *var ∉ vars 4*
by (*metis monom-numeral notInKeys-notInVars not-in-add not-in-isovarspar not-in-pow one-add-one power.simps*(*1*) *rel-simps*(*76*) *vars-monom-keys*)
have *h4* : *var ∉ vars* (*B^2−4∗A∗C*) **by**(*simp add*: *h4a freeA freeB freeC not-in-pow not-in-mult not-in-sub*)
have *h5* : ∀ *x. insertion* (*nth-default 0* (*list-update X var x*)) (−*B*) = −*Bval* **using** *allBval* **by**(*simp add*: *insertion-neg*)
have *h6* : ∀ *x. insertion* (*nth-default 0* (*list-update X var x*)) *1 = 1* **by** *simp*
have *h6a* : ∀ *x. insertion* (*nth-default 0* (*list-update X var x*)) (−*1*) = (−*1*) **using** *h6* **by** (*simp add*: *insertion-neg*)
have *h7a* : ∀ *x. insertion* (*nth-default 0* (*list-update X var x*)) *4 = 4* **by** (*metis h6 insertion-add numeral-Bit0 one-add-one*)
have *h7b* : *var ∉ vars*(*4∗A∗C*) **using** *freeA freeC* **by** (*simp add*: *h4a not-in-mult*)

have *h7c* : *var* $\notin$ *vars*(*B*^*2*) **using** *freeB not-in-pow* **by** *auto*
have *h7* : $\forall$ *x*. *insertion* (*nth-default 0* (*list-update X var x*)) (*B*^*2*−*4*∗*A*∗*C*) = (*Bval*^*2*−*4*∗*Aval*∗*Cval*)
by (*simp add*: *h7a allAval allBval allCval insertion-mult insertion-sub power2-eq-square*)
have *h8a* : $\forall$ *x*. *insertion* (*nth-default 0* (*list-update X var x*)) *2* = *2* **by** (*metis h6 insertion-add one-add-one*)
have *h8* : $\forall$ *x*. *insertion* (*nth-default 0* (*list-update X var x*)) (*2*∗*A*) = (*2*∗*Aval*) **by**(*simp add*: *allAval h8a insertion-mult*)

have *p1* : ($\forall$ *a*∈ *set L*. *eval* (*quadratic-sub var* (−*B*) *1* (*B*^*2*−*4*∗*A*∗*C*) (*2*∗*A*) *a*) (*X*[*var* := *x*]))
$\Longrightarrow$($\forall$ *a*∈ *set F*. *eval* (*quadratic-sub-fm var* (−*B*) *1* (*B*^*2*−*4*∗*A*∗*C*) (*2*∗*A*) *a*) (*X*[*var* := *x*]))
$\Longrightarrow$ $\exists$ *x*. *length xs* = *var* ∧ (($\forall$ *a*∈*set L* . *aEval a* ((*xs*@*x*#Γ)[*var* := *x*])) ∧ ($\forall$ *f*∈*set F*. *eval f* ((*xs*@*x*#Γ)[*var* := *x*])))
proof−
assume *p1* : ($\forall$ *a*∈ *set L*. *eval* (*quadratic-sub var* (−*B*) *1* (*B*^*2*−*4*∗*A*∗*C*) (*2*∗*A*) *a*) (*X*[*var* := *x*]))
assume *p2* : ($\forall$ *a*∈ *set F*. *eval* (*quadratic-sub-fm var* (−*B*) *1* (*B*^*2*−*4*∗*A*∗*C*) (*2*∗*A*) *a*) (*X*[*var* := *x*]))
show *?thesis*
using *free-in-quad*[**where** *a*=−*B*,**where** *b*=*1*, **where** *c*=(*B*^*2*−*4*∗*A*∗*C*), **where** *d*=*2*∗*A*,**where** *var*=*var*,*OF h1a h1b h1e h1g*]
using *quadratic-sub*[**where** *a*=−*B*,**where** *b*=*1*, **where** *c*=(*B*^*2*−*4*∗*A*∗*C*), **where** *d*=*2*∗*A*,**where** *var*=*var*, **where** *L*=*X*, *OF Xlength*,
where *Dv*=*2*∗*Aval*, *OF h2*, **where** *Cv*=(*Bval*^*2*−*4*∗*Aval*∗*Cval*), *OF h3*, **where** *Av*=−*Bval*, *OF h4 h5*, **where** *Bv*=*1*, *OF h6 h7 h8*]
using *free-in-quad-fm*[**where** *a*=−*B*,**where** *b*=*1*, **where** *c*=(*B*^*2*−*4*∗*A*∗*C*), **where** *d*=*2*∗*A*,**where** *var*=*var*,*OF h1a h1b h1e h1g*]
using *quadratic-sub-fm*[**where** *a*=−*B*,**where** *b*=*1*, **where** *c*=(*B*^*2*−*4*∗*A*∗*C*), **where** *d*=*2*∗*A*,**where** *var*=*var*, **where** *L*=*X*, *OF Xlength*,
where *Dv*=*2*∗*Aval*, *OF h2*, **where** *Cv*=(*Bval*^*2*−*4*∗*Aval*∗*Cval*), *OF h3*, **where** *Av*=−*Bval*, *OF h4 h5*, **where** *Bv*=*1*, *OF h6 h7 h8*]
p1 p2
using *var-not-in-eval*
by (*metis X-def hlength list-update-length*)
qed
have *p2* : ($\forall$ *a*∈ *set L*. *eval* (*quadratic-sub var* (−*B*) (−*1*) (*B*^*2*−*4*∗*A*∗*C*) (*2*∗*A*) *a*) (*X*[*var* := *x*]))
$\Longrightarrow$($\forall$ *a*∈ *set F*. *eval* (*quadratic-sub-fm var* (−*B*) (−*1*) (*B*^*2*−*4*∗*A*∗*C*) (*2*∗*A*) *a*) (*X*[*var* := *x*]))
$\Longrightarrow$$\exists$ *x*. *length xs* = *var* ∧ (($\forall$ *a*∈*set L* . *aEval a* ((*xs*@*x*#Γ)[*var* := *x*])) ∧ ($\forall$ *f*∈*set F*. *eval f* ((*xs*@*x*#Γ)[*var* := *x*])))
using *free-in-quad*[**where** *a*=−*B*,**where** *b*=−*1*, **where** *c*=(*B*^*2*−*4*∗*A*∗*C*), **where** *d*=*2*∗*A*,**where** *var*=*var*,*OF h1a h1c h1e h1g*]
using *quadratic-sub*[**where** *a*=−*B*,**where** *b*=−*1*, **where** *c*=(*B*^*2*−*4*∗*A*∗*C*), **where** *d*=*2*∗*A*,**where** *var*=*var*, **where** *L*=*X*, *OF Xlength*,
where *Dv*=*2*∗*Aval*, *OF h2*, **where** *Cv*=(*Bval*^*2*−*4*∗*Aval*∗*Cval*), *OF*

h3, **where** *Av=−Bval*, *OF h4 h5*, **where** *Bv=−1*, *OF h6a h7 h8*]

using *free-in-quad-fm*[**where** *a=−B*,**where** *b=−1*, **where** *c=(B^2−4∗A∗C)*, **where** *d=2∗A*,**where** *var=var*,*OF h1a h1c h1e h1g*]
using *quadratic-sub-fm*[**where** *a=−B*,**where** *b=−1*, **where** *c=(B^2−4∗A∗C)*, **where** *d=2∗A*,**where** *var=var*, **where** *L=X*, *OF Xlength*,
where *Dv=2∗Aval*, *OF h2*, **where** *Cv=(Bval^2−4∗Aval∗Cval)*, *OF h3*, **where** *Av=−Bval*, *OF h4 h5*, **where** *Bv=−1*, *OF h6a h7 h8*]

using *var-not-in-eval* **by** (*metis X-def hlength list-update-length*)
then show *?thesis*
using *h p1 p2* **by** *blast*
next
case *False*
then show *?thesis* **using** *h* **by**(*simp add*: *aval0 False h4*)
qed
qed
have (∃ *x*.((∀ *a*∈*set L* . *aEval a* ((*xs*@*x*#Γ)[*var* := *x*])) ∧ (∀ *f*∈*set F*. *eval f* ((*xs*@*x*#Γ)[*var* := *x*])))∨(∀ *x*. *aEval* (*Eq p*) (*xs* @ *x*#Γ))
proof(*cases* (∃ *x*.((∀ *a*∈*set L* . *aEval a* ((*xs*@*x*#Γ)[*var* := *x*])) ∧ (∀ *f*∈*set F*. *eval f* ((*xs*@*x*#Γ)[*var* := *x*])))))
case *True*
then show *?thesis* **by** *auto*
next
case *False*
then have *Aval=0*∧*Bval=0*∧*Cval=0* **using** *h* **by** *auto*
then have (∀ *x*. *aEval* (*Eq p*) (*xs* @ *x* # Γ))
unfolding *express-p* **apply**(*simp add*:*insertion-add insertion-mult insertion-pow*)
using *allAval allBval allCval hlength* **unfolding** *X-def* **by** *auto*
then show *?thesis* **by** *auto*
qed
}

then have *right* : (∃ *x*. *eval* (*elimVar var L F* (*Eq p*)) (*xs* @ *x* # Γ)) ⟹
((∃ *x*. *eval* (*list-conj* (*map fm.Atom L* @ *F*)) (*xs* @ *x* # Γ))∨(∀ *x*. *aEval* (*Eq p*) (*xs* @ *x* # Γ)))
by (*smt UnE eval.simps*(*1*) *eval-list-conj hlength imageE list-update-length set-append set-map*)

show *?thesis* **using** *right left* **by** *blast*
qed

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−

have *h4* : *var* ∉ *vars*(*4*:: *real mpoly*) **using** *var-not-in-Const*
by (*metis* (*full-types*) *isolate-var-one monom-numeral not-in-isovarspar numeral-One vars-monom-keys zero-neq-numeral*)
have *hlinear*: ∀ *f*∈*set*(*map* (λ*a*. *Atom*(*linear-substitution var* (−*isolate-variable-sparse p var 0*) (*isolate-variable-sparse p var* (*Suc 0*)) *a*)) *L* @
map (*linear-substitution-fm var* (−*isolate-variable-sparse p var 0*) (*isolate-variable-sparse p var* (*Suc 0*)))
F). *freeIn var f*
using
var-not-in-linear[**where** *c*=(*isolate-variable-sparse p var* (*Suc 0*)), **where** *b*=(− *isolate-variable-sparse p var 0*), **where** *var*=*var*]
var-not-in-linear-fm[**where** *c*=(*isolate-variable-sparse p var* (*Suc 0*)), **where** *b*=(−*isolate-variable-sparse p var 0*), **where** *var*=*var*]
not-in-isovarspar not-in-neg **by** *auto*
have *freeA* : *var* ∉ *vars* (− *isolate-variable-sparse p var* (*Suc 0*))
using *not-in-isovarspar not-in-neg* **by** *auto*
have *freeB1* : *var* ∉ *vars* (*1*::*real mpoly*)
by (*metis h4 monom-numeral monom-one notInKeys-notInVars vars-monom-keys zero-neq-numeral*)
have *freeC* : *var* ∉ *vars* (((*isolate-variable-sparse p var* (*Suc 0*))^2 −
4 ∗ *isolate-variable-sparse p var 2* ∗ *isolate-variable-sparse p var 0*))
using *not-in-isovarspar not-in-pow not-in-sub not-in-mult h4* **by** *auto*
have *freeD* : *var* ∉ *vars* ((*2* ∗ *isolate-variable-sparse p var 2*))
using *not-in-isovarspar not-in-mult*
by (*metis mult-2 not-in-add*)
have *freeB2* : *var*∉*vars* (−*1*::*real mpoly*)
using *freeB1 not-in-neg* **by** *auto*
have *quadratic1* : ∀ *f*∈*set*(*map* (*quadratic-sub var* (− *isolate-variable-sparse p var* (*Suc 0*)) *1*
((*isolate-variable-sparse p var* (*Suc 0*))^2 −
4 ∗ *isolate-variable-sparse p var 2* ∗ *isolate-variable-sparse p var 0*)
(*2* ∗ *isolate-variable-sparse p var 2*))
L @
map (*quadratic-sub-fm var* (− *isolate-variable-sparse p var* (*Suc 0*)) *1*
((*isolate-variable-sparse p var* (*Suc 0*))^2 −
4 ∗ *isolate-variable-sparse p var 2* ∗ *isolate-variable-sparse p var 0*)
(*2* ∗ *isolate-variable-sparse p var 2*))
F). *freeIn var f*
using *free-in-quad*[*OF freeA freeB1 freeC freeD*]
free-in-quad-fm[*OF freeA freeB1 freeC freeD*] **by** *auto*
have *quadratic2* : ∀ *f*∈*set*(*map* (*quadratic-sub var* (− *isolate-variable-sparse p var* (*Suc 0*)) (*−1*)
((*isolate-variable-sparse p var* (*Suc 0*))^2 −
4 ∗ *isolate-variable-sparse p var 2* ∗ *isolate-variable-sparse p var 0*)
(*2* ∗ *isolate-variable-sparse p var 2*))
L @
map (*quadratic-sub-fm var* (− *isolate-variable-sparse p var* (*Suc 0*)) (*−1*)
((*isolate-variable-sparse p var* (*Suc 0*))^2 −

```
            4 * isolate-variable-sparse p var 2 * isolate-variable-sparse p var 0)
           (2 * isolate-variable-sparse p var 2))
       F). freeIn var f
   using free-in-quad[OF freeA freeB2 freeC freeD]
     free-in-quad-fm[OF freeA freeB2 freeC freeD] by auto
 show ?thesis
  using not-in-mult not-in-add h4 not-in-pow not-in-sub freeIn-list-conj not-in-isovarspar
hlinear quadratic1 quadratic2
   by simp
qed
```

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−
 define A where A = (isolate-variable-sparse p var 2)
 define B where B = (isolate-variable-sparse p var 1)
 define C where C = (isolate-variable-sparse p var 0)
 have freeA : var ∉ vars A
   unfolding A-def
   by (simp add: not-in-isovarspar)
 have freeB : var ∉ vars B
   unfolding B-def
   by (simp add: not-in-isovarspar)
 have freeC : var ∉ vars C
   unfolding C-def
   by (simp add: not-in-isovarspar)
 have express-p : p = A∗(Var var)^2+B∗(Var var)+C
   using express-poly[OF low-pow] unfolding A-def B-def C-def
   by fastforce
 have af : isolate-variable-sparse p var 2 = A
   using A-def by auto
 have bf : isolate-variable-sparse p var (Suc 0) = B
   using B-def by auto
 have cf : isolate-variable-sparse p var 0 = C
   using C-def by auto
 have xlength : ∀ x. (insertion (nth-default 0 (xs @ x # Γ)) (Var var))= x
```

using *hlength insertion-var*
by (*metis add.commute add-strict-increasing length-append length-greater-0-conv list.distinct*(*1*) *list-update-id nth-append-length order-refl*)
fix *x*
define *c* **where** *c i* = (*insertion* (*nth-default 0* (*xs* @ *x* # Γ)) (*isolate-variable-sparse p var i*)) **for** *i*
have *c2* : ∀ *x*. *insertion* (*nth-default 0* (*xs* @ *x* # Γ)) *A* = *c 2*
using *freeA* **apply**(*simp add*: *A-def c-def*)
by (*simp add*: *hlength insertion-lowerPoly1*)
have *c1* : ∀ *x*. *insertion* (*nth-default 0* (*xs* @ *x* # Γ)) *B* = *c 1*
using *freeB* **apply**(*simp add*: *B-def c-def*)
by (*simp add*: *hlength insertion-lowerPoly1*)
have *c0* : ∀ *x*. *insertion* (*nth-default 0* (*xs* @ *x* # Γ)) *C* = *c 0*
using *freeC* **apply**(*simp add*: *C-def c-def*)
by (*simp add*: *hlength insertion-lowerPoly1*)
have *sum* : ∀ *x*. *c 2* ∗ x^2 + *c* (*Suc 0*) ∗ *x* + *c 0* = (∑ *i*≤*2*. *c i* ∗ *x* ^ *i*)
by (*simp add*: *numerals*(*2*))
have (∀ *x*. *aEval* (*Eq p*) (*xs* @ *x* # Γ)) = (¬ *?non0*)
apply(*simp add* : *af bf cf*)
unfolding *express-p* **apply**(*simp add*:*insertion-add insertion-mult insertion-pow xlength*)
apply(*simp add*:*c2 c1 c0*)
apply(*simp add*: *sum*)
using *polyfun-eq-0*[**where** *c*=*c*, **where** *n*=*2*]
using *sum* **by** *auto*
then have ¬(∀ *x*. *aEval* (*Eq p*) (*xs* @ *x* Γ))
using *nonzero* **by** *auto*
then show *?thesis*
using *disjE*[*OF elimVar-eq*[*OF hlength in-list*, **where** *F*=*F*, **where** Γ=Γ], **where** *R*=*?thesis*]
using ‹(∀ *x*. *aEval* (*Eq p*) (*xs* @ *x* # Γ)) = (¬ (∀ *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*))› *low-pow nonzero* **by** *blast*
qed

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(cases find-lucky-eq x L)
  case None
  then show ?thesis using assms by auto
next
  case (Some p)
  have inset : Eq p ∈ set L
    using Some proof(induction L)
    case Nil
    then show ?case by auto
  next
    case (Cons a L)
    then show ?case proof(cases a)
      case (Less x1)
      then show ?thesis using Cons by auto
    next
      case (Eq p')
      show ?thesis using Cons
        unfolding Eq apply simp apply(cases (MPoly-Type.degree p' x = Suc 0
∨ MPoly-Type.degree p' x = 2)) apply simp-all
      apply(cases check-nonzero-const (isolate-variable-sparse p' x 2)) apply(simp-all)
      apply(cases check-nonzero-const (isolate-variable-sparse p' x 1)) apply(simp-all)
      apply(cases check-nonzero-const (isolate-variable-sparse p' x 0)) by(simp-all)
    next
      case (Leq x3)
      then show ?thesis using Cons by auto
    next
      case (Neq x4)
      then show ?thesis using Cons by auto
    qed
  qed
  have degree : MPoly-Type.degree p x = 1 ∨ MPoly-Type.degree p x = 2
    using Some proof(induction L)
    case Nil
    then show ?case by auto
  next
    case (Cons a L)
    then show ?case proof(cases a)
      case (Less x1)
      then show ?thesis using Cons by auto
    next
      case (Eq p')
      show ?thesis using Cons
        unfolding Eq apply simp apply(cases (MPoly-Type.degree p' x = Suc 0
∨ MPoly-Type.degree p' x = 2)) apply simp-all
      apply(cases check-nonzero-const (isolate-variable-sparse p' x 2)) apply(simp-all)
```

```
    apply(cases check-nonzero-const (isolate-variable-sparse p' x 1)) apply(simp-all)
    apply(cases check-nonzero-const (isolate-variable-sparse p' x 0)) by(simp-all)
   next
    case (Leq x3)
    then show ?thesis using Cons by auto
   next
    case (Neq x4)
    then show ?thesis using Cons by auto
   qed
  qed
  have nonzero : ∀ xa. insertion (nth-default 0 (xs @ xa # Γ)) (isolate-variable-sparse
p x 2) ≠ 0 ∨
      insertion (nth-default 0 (xs @ xa # Γ)) (isolate-variable-sparse p x 1) ≠ 0 ∨
      insertion (nth-default 0 (xs @ xa # Γ)) (isolate-variable-sparse p x 0) ≠ 0
   using Some proof(induction L)
   case Nil
   then show ?case by auto
  next
   case (Cons a L)
   then show ?case proof(cases a)
    case (Less x1)
    then show ?thesis using Cons by auto
   next
    case (Eq p')
    have h : ⋀p xa. check-nonzero-const p ⟹ insertion (nth-default 0 (xs @ xa
# Γ)) p ≠ 0
    proof−
     fix p xa
     assume h : check-nonzero-const p
     show insertion (nth-default 0 (xs @ xa # Γ)) p ≠ 0
      apply(cases get-if-const p)
      using h get-if-const-insertion by simp-all
    qed
    show ?thesis using Cons(2)
    unfolding Eq apply (simp del:get-if-const.simps) apply(cases (MPoly-Type.degree
p' x = Suc 0 ∨ MPoly-Type.degree p' x = 2)) defer using Cons apply simp
     apply (simp del:get-if-const.simps)
     apply(cases check-nonzero-const (isolate-variable-sparse p' x 2))
     apply(simp del:get-if-const.simps) using h
     apply simp
     apply(cases check-nonzero-const (isolate-variable-sparse p' x 1))
     apply(simp del:get-if-const.simps) using h
     apply simp
     apply(cases check-nonzero-const (isolate-variable-sparse p' x 0))
     apply(simp del:get-if-const.simps) using h
     apply simp
     using Cons by auto
   next
    case (Leq x3)
```

```
    then show ?thesis using Cons by auto
   next
    case (Neq x4)
    then show ?thesis using Cons by auto
   qed
  qed
  show ?thesis
   using elimVar-eq-2[OF assms(2) inset degree nonzero] Some assms by auto
qed


lemma luckyFind'-eval :
  assumes length xs = var
  shows (∃x. eval (list-conj (map fm.Atom L @ F)) (xs @ x # Γ)) = (∃x. eval
(luckyFind' var L F) (xs @ x # Γ))
proof(cases find-lucky-eq var L)
  case None
  show ?thesis apply(simp add:eval-list-conj None)
   apply(rule ex-cong1)
   apply auto
   by (meson UnCI eval.simps(1) image-eqI)
next
  case (Some p)
  have ∃F'. luckyFind var L F = Some F' by (simp add:Some)
  then obtain F' where F'-def: luckyFind var L F = Some F' by metis
  show ?thesis
   unfolding luckyFind-eval[OF F'-def assms]
   using F'-def Some by auto
qed



lemma luckiestFind-eval :
  assumes length xs = var
  shows (∃x. eval (list-conj (map fm.Atom L @ F)) (xs @ x # Γ)) = (∃x. eval
(luckiestFind var L F) (xs @ x # Γ))
proof(cases find-luckiest-eq var L)
  case None
  show ?thesis apply(simp add:eval-list-conj None)
   apply(rule ex-cong1)
   apply auto
   by (meson UnCI eval.simps(1) image-eqI)
next
  case (Some p)
  have h1: Eq p ∈ set L
   using Some apply(induction L arbitrary:p)
   apply simp
   subgoal for a L p
    apply(rule find-luckiest-eq.elims[of var a#L Some p])
```

```
    apply simp-all
    subgoal for v p'
      apply(cases MPoly-Type.degree p' v = Suc 0 ∨ MPoly-Type.degree p' v =
2) apply simp-all
       apply(cases Set.is-empty (vars (isolate-variable-sparse p' v 2))) apply
simp-all
     apply(cases Set.is-empty (vars (isolate-variable-sparse p' v (Suc 0)))) apply
simp-all
       apply(cases Set.is-empty (vars (isolate-variable-sparse p' v 0))) apply
simp-all
      apply(cases MPoly-Type.coeff (isolate-variable-sparse p' v (Suc 0)) 0 = 0
⟶
     MPoly-Type.coeff (isolate-variable-sparse p' v 2) 0 = 0 ⟶ MPoly-Type.coeff
(isolate-variable-sparse p' v 0) 0 ≠ 0) by simp-all
    done
  done
 have h2 : MPoly-Type.degree p var = 1 ∨ MPoly-Type.degree p var = 2
  using Some apply(induction L arbitrary:p)
  apply simp
  subgoal for a L p
    apply(rule find-luckiest-eq.elims[of var a#L Some p])
    apply simp-all
    subgoal for v p'
      apply(cases MPoly-Type.degree p' v = Suc 0 ∨ MPoly-Type.degree p' v =
2) apply simp-all
       apply(cases Set.is-empty (vars (isolate-variable-sparse p' v 2))) apply
simp-all
     apply(cases Set.is-empty (vars (isolate-variable-sparse p' v (Suc 0)))) apply
simp-all
       apply(cases Set.is-empty (vars (isolate-variable-sparse p' v 0))) apply
simp-all
      apply(cases MPoly-Type.coeff (isolate-variable-sparse p' v (Suc 0)) 0 = 0
⟶
     MPoly-Type.coeff (isolate-variable-sparse p' v 2) 0 = 0 ⟶ MPoly-Type.coeff
(isolate-variable-sparse p' v 0) 0 ≠ 0) by simp-all
    done
  done
 have h : ⋀p xa. check-nonzero-const p ⟹ insertion (nth-default 0 (xs @ xa #
Γ)) p ≠ 0
 proof−
  fix p xa
  assume h : check-nonzero-const p
  show insertion (nth-default 0 (xs @ xa # Γ)) p ≠ 0
    apply(cases get-if-const p)
    using h get-if-const-insertion by simp-all
 qed

 have h3 : ∀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*
using *Some* **apply**(*induction L arbitrary:p*)
apply *simp*
subgoal for *a L p*
apply(*rule find-luckiest-eq.elims*[*of var a#L Some p*])
apply *simp-all*
subgoal for *v p′*
apply(*cases MPoly-Type.degree p′ v = Suc 0 ∨ MPoly-Type.degree p′ v = 2*) **apply** *simp-all*
apply(*cases Set.is-empty* (*vars* (*isolate-variable-sparse p′ v 2*))) **apply** *simp-all*
apply(*cases Set.is-empty* (*vars* (*isolate-variable-sparse p′ v* (*Suc 0*)))) **apply** *simp-all*
apply(*cases Set.is-empty* (*vars* (*isolate-variable-sparse p′ v 0*))) **apply** *simp-all*
apply(*cases MPoly-Type.coeff* (*isolate-variable-sparse p′ v* (*Suc 0*)) *0 = 0* ⟶
MPoly-Type.coeff (*isolate-variable-sparse p′ v 2*) *0 = 0* ⟶ *MPoly-Type.coeff* (*isolate-variable-sparse p′ v 0*) *0* ≠ *0*) **apply** *simp-all*
using *h*[*of isolate-variable-sparse p′ v 0*] *h*[*of isolate-variable-sparse p′ v* (*Suc 0*)] *h*[*of isolate-variable-sparse p′ v 2*] **apply** *simp*
by *blast*
done
done
show *?thesis* **apply**(*simp-all add*:*Some del*:*elimVar.simps*)
apply(*rule elimVar-eq-2*) **using** *assms* **apply** *simp* **using** *h1 h2 h3* **by** *auto*

qed

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 ∀ *p*∈*set*(*A*). *MPoly-Type.degree p var = 1 ∨ MPoly-Type.degree p var = 2* **using** *assms*(*1*)
proof(*induction L arbitrary*: *A L′*)
case *Nil*
then show *?case* **by** *auto*
next
case (*Cons a L*)

```
  then show ?case proof(cases a)
    case (Less p)
    {fix A' L'
      assume h : find-eq var L = (A', L')
      have A=A'
        using Less Cons h by(simp)
      then have ∀ p∈set A. MPoly-Type.degree p var = 1 ∨ MPoly-Type.degree p
var = 2
        using Cons h by auto
    }
    then show ?thesis by (meson surj-pair)
  next
    case (Eq p)
   then show ?thesis proof(cases MPoly-Type.degree p var = 1 ∨ MPoly-Type.degree
p var = 2)
      case True
      {fix A' L'
        assume h : find-eq var L = (A', L')
        have A= (p#A')
          using Eq Cons h True by auto
        then have ∀ p∈set A. MPoly-Type.degree p var = 1 ∨ MPoly-Type.degree
p var = 2
          using Cons h True by auto
      }
      then show ?thesis by (meson surj-pair)
    next
      case False
      {fix A' L'
        assume h : find-eq var L = (A', L')
        have A=A'
          using Eq Cons h False
         by (smt One-nat-def case-prod-conv find-eq.simps(3) less-2-cases less-SucE
numeral-2-eq-2 numeral-3-eq-3 prod.sel(1))
        then have ∀ p∈set A. MPoly-Type.degree p var = 1 ∨ MPoly-Type.degree
p var = 2
          using Cons h by auto
      }
      then show ?thesis by (meson surj-pair)
    qed
  next
    case (Leq p)
    {fix A' L'
      assume h : find-eq var L = (A', L')
      have A=A'
        using Leq Cons h by(simp)
      then have ∀ p∈set A. MPoly-Type.degree p var = 1 ∨ MPoly-Type.degree p
var = 2
        using Cons h by auto
    }
```

```
    then show ?thesis by (meson surj-pair)
  next
    case (Neq p)
    {fix A' L'
      assume h : find-eq var L = (A', L')
      have A=A'
        using Neq Cons h by(simp)
      then have ∀ p∈set A. MPoly-Type.degree p var = 1 ∨ MPoly-Type.degree p
var = 2
        using Cons h by auto
    }
    then show ?thesis by (meson surj-pair)
  qed
qed

lemma list-in-find-eq :
  assumes find-eq var L = (A,L')
  shows set(map Eq A @ L') = set Lusing assms(1)
proof(induction L arbitrary: A L')
  case Nil
  then show ?case by auto
next
  case (Cons a L)
  then show ?case proof(cases a)
    case (Less p)
    {fix A' L''
      assume h : find-eq var L = (A', L'')
      have A : A=A'
        using Less Cons h by(simp)
      have L : L'=Less p # L''
        using Less Cons h by simp
      have set (map Eq A @ L') = set (a # L)
        apply(simp add: A L Less) using Cons(1)[OF h] by auto
    }
    then show ?thesis by (meson surj-pair)
  next
    case (Eq p)
   then show ?thesis proof(cases MPoly-Type.degree p var = 1 ∨ MPoly-Type.degree
p var = 2)
      case True
      {fix A' L''
        assume h : find-eq var L = (A', L'')
        have A : A=(p#A')
          using Eq Cons h True by auto
        have L : L'= L''
          using Eq Cons h True by auto
        have set (map Eq A @ L') = set (a # L)
          apply(simp add: A L Eq) using Cons(1)[OF h] by auto
      }
```

```
    then show ?thesis by (meson surj-pair)
  next
    case False
    {fix A' L''
     assume h : find-eq var L = (A', L'')
     have A : A=A'
      using Eq Cons h False
       by (smt case-prod-conv degree-find-eq find-eq.simps(3) list.set-intros(1)
prod.sel(1))
     have L : L'=Eq p # L''
      using Eq Cons h
        by (smt A case-prod-conv find-eq.simps(3) not-Cons-self2 prod.sel(1)
prod.sel(2))
     have set (map Eq A @ L') = set (a # L)
      apply(simp add: A L Eq) using Cons(1)[OF h] by auto
    }
    then show ?thesis by (meson surj-pair)
  qed
 next
  case (Leq p)
  {fix A' L''
   assume h : find-eq var L = (A', L'')
   have A : A=A'
    using Leq Cons h by(simp)
   have L : L'=Leq p # L''
    using Leq Cons h by simp
   have set (map Eq A @ L') = set (a # L)
    apply(simp add: A L Leq) using Cons(1)[OF h] by auto
  }
  then show ?thesis by (meson surj-pair)
 next
  case (Neq p)
  {fix A' L''
   assume h : find-eq var L = (A', L'')
   have A : A=A'
    using Neq Cons h by(simp)
   have L : L'=Neq p # L''
    using Neq Cons h by simp
   have set (map Eq A @ L') = set (a # L)
    apply(simp add: A L Neq) using Cons(1)[OF h] by auto
  }
  then show ?thesis by (meson surj-pair)
 qed
qed


lemma qe-eq-one-eval :
 assumes hlength : length xs = var
 shows (∃ x. (eval (list-conj ((map Atom L) @ F)) (xs @ (x#Γ)))) = (∃ x.(eval
```

(*qe-eq-one var L F*) (*xs* @ (*x*#Γ))))
proof(*cases find-eq var L*)
case (*Pair A L'*)
then show *?thesis* **proof**(*cases A*)
case *Nil*
show *?thesis* **proof** *safe*
fix *x*
assume *h* : *eval* (*list-conj* (*map fm.Atom L* @ *F*)) (*xs* @ *x* # Γ)
show ∃ *x*. *eval* (*qe-eq-one var L F*) (*xs* @ *x* # Γ) **apply**(*simp*) **using** *Nil Pair h* **by** *auto*
next
fix *x*
assume *h* : *eval* (*qe-eq-one var L F*) (*xs* @ *x* # Γ)
show ∃ *x*. *eval* (*list-conj* (*map fm.Atom L* @ *F*)) (*xs* @ *x* # Γ)
apply(*rule exI*[**where** *x*=*x*]) **using** *Nil Pair h* **by** *auto*
qed
next
case (*Cons p A'*)
have *set*(*map Eq* (*p* # *A'*) @ *L'*) = *set L*
using *list-in-find-eq*[*OF Pair*] *Cons* **by** *auto*
then have *in-p*: *Eq p* ∈ *set* (*L*)
by *auto*
have *p*∈(*set A*) **using** *Cons* **by** *auto*
then have *low-pow* : *MPoly-Type.degree p var* = *1* ∨ *MPoly-Type.degree p var* = *2*
using *degree-find-eq*[*OF Pair*] **by** *auto*
have (∃ *x*.(*eval* (*qe-eq-one var L F*) (*xs* @ (*x*#Γ)))) =
(∃ *x*.(*eval* (*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*))
)) (*xs* @ (*x*#Γ))))
apply(*rule ex-cong1*) **apply**(*simp only*: *qe-eq-one.simps*) **using** *Pair Cons* **by** *auto*
also have ... = (∃ *x*. ((¬*eval* (*split-p var p*) (*xs* @ *x* # Γ)) ∧ *eval* (*elimVar var L F* (*Eq p*)) (*xs* @ *x* # Γ)) ∨
eval (*split-p var p*) (*xs* @ *x* # Γ) ∧
(∀ *f*∈*set* (*map fm.Atom* (*map Eq A'* @ *L'*) @ *F*). *eval f* (*xs* @ *x* # Γ)))
by(*simp add*: *eval-list-conj*)
also have ... = (∃ *x*. *eval* (*list-conj* (*map fm.Atom L* @ *F*)) (*xs* @ *x* # Γ))
proof(*cases* ∀ *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*)
case *True*
have (∃ *x*. ((¬*eval* (*split-p var p*) (*xs* @ *x* # Γ)) ∧ *eval* (*elimVar var L F* (*Eq p*)) (*xs* @ *x* # Γ)) ∨
eval (*split-p var p*) (*xs* @ *x* # Γ) ∧

```
      (∀f∈set (map fm.Atom (map Eq A' @ L') @ F). eval f (xs @ x # Γ))) =
      (∃x. eval (elimVar var L F (Eq p)) (xs @ x # Γ))
    proof safe
      fix x
      assume eval (elimVar var L F (Eq p)) (xs @ x # Γ)
      then show ∃x. eval (elimVar var L F (Eq p)) (xs @ x # Γ) by auto
    next
      fix x
      assume h : eval (split-p var p) (xs @ x # Γ)
      have ¬ eval (split-p var p) (xs @ x # Γ)
        using True by simp
      then show ∃x. eval (elimVar var L F (Eq p)) (xs @ x # Γ) using h by
simp
    next
      fix x
      assume eval (elimVar var L F (Eq p)) (xs @ x # Γ)
      then show ∃x. ¬ eval (split-p var p) (xs @ x # Γ) ∧ eval (elimVar var L
F (Eq p)) (xs @ x # Γ) ∨
          eval (split-p var p) (xs @ x # Γ) ∧
          (∀f∈set (map fm.Atom (map Eq A' @ L') @ F). eval f (xs @ x # Γ))
        by auto
    qed
    then show ?thesis using elimVar-eq-2[OF hlength in-p low-pow True] by
simp
  next
    case False
    have h1: ∀x. eval (split-p var p) (xs @ x # Γ)
      using False apply(simp) using not-in-isovarspar
      by (metis hlength insertion-lowerPoly1)
    have set(map Eq (p # A') @ L') = set L
      using list-in-find-eq[OF Pair] Cons by auto
    then have h5 : set(map fm.Atom (map Eq (p # A') @ L') @ F) = set(map
fm.Atom L @ F)
      by auto
    have h4 : (∃x. (aEval (Eq p) (xs @ x # Γ)) ∧
      (∀f∈set (map fm.Atom (map Eq A' @ L') @ F). eval f (xs @ x # Γ))) =
       (∃x.(∀f∈set (map fm.Atom (map Eq (p#A') @ L') @ F). eval f (xs @ x
# Γ)))
      by(simp)
    have h2 : (∃x. eval (list-conj (map fm.Atom L @ F)) (xs @ x # Γ)) = (∃x.
(aEval (Eq p) (xs @ x # Γ)) ∧
      (∀f∈set (map fm.Atom (map Eq A' @ L') @ F). eval f (xs @ x # Γ)))
      by(simp only: h4 h5 eval-list-conj)
    have h3 : ∀x. (aEval (Eq p) (xs @ x # Γ))
    proof−
      define A where A = (isolate-variable-sparse p var 2)
      define B where B = (isolate-variable-sparse p var 1)
      define C where C = (isolate-variable-sparse p var 0)
      have freeA : var ∉ vars A
```

unfolding *A-def*
by (*simp add*: *not-in-isovarspar*)
have *freeB* : *var* ∉ *vars B*
unfolding *B-def*
by (*simp add*: *not-in-isovarspar*)
have *freeC* : *var* ∉ *vars C*
unfolding *C-def*
by (*simp add*: *not-in-isovarspar*)
have *express-p* : *p* = *A*∗(*Var var*)^2+*B*∗(*Var var*)+*C*
using *express-poly*[*OF low-pow*] **unfolding** *A-def B-def C-def*
by *fastforce*
have *xlength* : ∀ *x*. (*insertion* (*nth-default 0* (*xs* @ *x* # Γ)) (*Var var*))= *x*
using *hlength insertion-var*
by (*metis add.commute add-strict-increasing length-append length-greater-0-conv list.distinct*(*1*) *list-update-id nth-append-length order-refl*)
fix *x*
define *c* **where** *c i* = (*insertion* (*nth-default 0* (*xs* @ *x* # Γ)) (*isolate-variable-sparse p var i*)) **for** *i*
have *c2* : ∀ *x*. *insertion* (*nth-default 0* (*xs* @ *x* # Γ)) *A* = *c 2*
using *freeA* **apply**(*simp add*: *A-def c-def*)
by (*simp add*: *hlength insertion-lowerPoly1*)
have *c1* : ∀ *x*. *insertion* (*nth-default 0* (*xs* @ *x* # Γ)) *B* = *c 1*
using *freeB* **apply**(*simp add*: *B-def c-def*)
by (*simp add*: *hlength insertion-lowerPoly1*)
have *c0* : ∀ *x*. *insertion* (*nth-default 0* (*xs* @ *x* # Γ)) *C* = *c 0*
using *freeC* **apply**(*simp add*: *C-def c-def*)
by (*simp add*: *hlength insertion-lowerPoly1*)
have *sum* : ∀ *x*. *c 2* ∗ *x*^2 + *c* (*Suc 0*) ∗ *x* + *c 0* = (∑ *i*≤*2*. *c i* ∗ *x* ^ *i*)
by (*simp add*: *numerals*(*2*))
show *?thesis* **unfolding** *express-p* **apply**(*simp add*:*insertion-add insertion-mult insertion-pow xlength*)
apply(*simp add*:*c2 c1 c0 sum polyfun-eq-0*[**where** *c*=*c*, **where** *n*=*2*])
using *False* **apply**(*simp*)
by (*metis A-def B-def C-def One-nat-def c0 c1 c2 le-SucE le-zero-eq numeral-2-eq-2*)
qed
show *?thesis* **apply**(*simp only*: *h1 h2*) **using** *h3* **by**(*simp*)
qed
finally show *?thesis* **by** *auto*
qed
qed

lemma *qe-eq-repeat-helper-eval-case1* :
assumes *hlength* : *length xs* = *var*
assumes *degreeGood* : ∀ *p*∈*set*(*A*). *MPoly-Type.degree p var* = *1* ∨ *MPoly-Type.degree p var* = *2*

shows ((*eval* (*list-conj* ((*map* (*Atom o Eq*) *A*) @ (*map Atom L*) @ *F*)) (*xs* @ (*x*#Γ))))
⟹ (*eval* (*qe-eq-repeat-helper var A L F*) (*xs* @ *x* # Γ))
proof(*induction A rule* : *in-list-induct*)
case *Nil*
then show *?case* **by** *auto*
next
case (*Cons p A′*)
assume *assm* : ((*eval* (*list-conj* ((*map* (*Atom o Eq*) (*p*#*A′*)) @ (*map Atom L*) @ *F*)) (*xs* @ (*x*#Γ))))
then have *h* : *insertion* (*nth-default 0* (*xs* @ *x* # Γ)) *p* = *0* ∧ (*eval* (*qe-eq-repeat-helper var A′ L F*) (*xs* @ *x* # Γ))
using *Cons* **by**(*simp add*: *eval-list-conj*)
have ¬ *eval* (*split-p var p*) (*xs* @ *x* # Γ) ∧ *eval* (*elimVar var* ((*map Eq* (*p*# *A′*)) @ *L*) *F* (*Eq p*)) (*xs* @ *x* # Γ) ∨
eval (*split-p var p*) (*xs* @ *x* # Γ) ∧ *eval* (*qe-eq-repeat-helper var A′ L F*) (*xs* @ *x* # Γ)
proof(*cases eval* (*split-p var p*) (*xs* @ *x* # Γ))
case *True*
then show *?thesis* **using** *h* **by** *blast*
next
case *False*
have *all0* : ∀ *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*
using *False* **apply**(*simp*) **using** *not-in-isovarspar*
by (*metis hlength insertion-lowerPoly1*)
have *in-p* : *Eq p*∈*set*((*map Eq* (*p* # *A′*) @ *L*))
by *auto*
have *p*∈(*set A*) **using** *Cons* **by** *auto*
then have *low-pow* : *MPoly-Type.degree p var* = *1* ∨ *MPoly-Type.degree p var* = *2*
using *degreeGood* **by** *auto*
have *list-manipulate* : *map fm.Atom* (*map Eq* (*p* # *A′*) @ *L*) = *map* (*fm.Atom* ∘ *Eq*) (*p* # *A′*) @ *map fm.Atom L*
by(*simp*)
have *eval* (*elimVar var* ((*map Eq* (*p*# *A′*)) @ *L*) *F* (*Eq p*)) (*xs* @ *x* # Γ)
using *elimVar-eq-2*[*OF hlength in-p low-pow all0*, **where** *F*=*F*] **apply**(*simp only*: *list-manipulate*)
using *assm freeIn-elimVar-eq*[**where** *var*=*var*, **where** *L*=(*map Eq* (*p* # *A′*) @ *L*), **where** *F*=*F*, **where** *p*=*p*]
by (*metis append.assoc hlength list-update-length var-not-in-eval*)
then show *?thesis* **apply**(*simp only*: *False*) **by** *blast*
qed
then show *?case* **by**(*simp only*: *qe-eq-repeat-helper.simps eval.simps*)
qed

lemma *qe-eq-repeat-helper-eval-case2* :

assumes *hlength* : *length xs = var*
assumes *degreeGood* : $\forall$ *p*$\in$*set*(*A*). *MPoly-Type.degree p var = 1* $\vee$ *MPoly-Type.degree p var = 2*
shows (*eval* (*qe-eq-repeat-helper var A L F*) (*xs* @ *x* # Γ))
$\Longrightarrow$ $\exists$ *x*. ((*eval* (*list-conj* ((*map* (*Atom o Eq*) *A*) @ (*map Atom L*) @ *F*)) (*xs* @ (*x*#Γ))))
proof(*induction A rule* : *in-list-induct*)
case *Nil*
then show *?case* **apply**(*simp*) **apply**(*rule exI*[**where** *x*=*x*]) **by** *simp*
next
case (*Cons p A'*)
have *h* : $\neg$ *eval* (*split-p var p*) (*xs* @ *x* # Γ) $\wedge$ *eval* (*elimVar var* ((*map Eq* (*p*# *A'*)) @ *L*) *F* (*Eq p*)) (*xs* @ *x* # Γ) $\vee$
eval (*split-p var p*) (*xs* @ *x* # Γ) $\wedge$ *eval* (*qe-eq-repeat-helper var A' L F*) (*xs* @ *x* # Γ)
using *Cons* **by**(*simp only*:*qe-eq-repeat-helper.simps eval.simps*)
have *p*$\in$*set*(*A*) **using** *Cons*(*1*) **.**
then have *degp* : *MPoly-Type.degree p var = 1* $\vee$ *MPoly-Type.degree p var = 2*
using *degreeGood* **by** *auto*
show *?case* **proof**(*cases eval* (*split-p var p*) (*xs* @ *x* # Γ))
case *True*
have $\exists$ *x*. *eval* (*list-conj* (*map* (*fm.Atom* $\circ$ *Eq*) *A'* @ *map fm.Atom L* @ *F*)) (*xs* @ *x* # Γ)
using *h True Cons* **by** *blast*
then obtain *x* **where** *x-def* : *eval* (*list-conj* (*map* (*fm.Atom* $\circ$ *Eq*) *A'* @ *map fm.Atom L* @ *F*)) (*xs* @ *x* # Γ) **by** *metis*
define *A* **where** *A* = (*isolate-variable-sparse p var 2*)
define *B* **where** *B* = (*isolate-variable-sparse p var 1*)
define *C* **where** *C* = (*isolate-variable-sparse p var 0*)
have *express-p* : *p* = *A* $*$ *Var var* ^2+*B* $*$ *Var var*+*C*
proof(*cases MPoly-Type.degree p var = 1*)
case *True*
have *a0* : *A* = *0* **apply**(*simp add*: *A-def*) **using** *True*
by (*simp add*: *isovar-greater-degree*)
show *?thesis* **using** *sum-over-zero*[**where** *mp*=*p*, **where** *x*=*var*] **apply**(*subst* (*asm*) *True*) **by**(*simp add*:*a0 B-def C-def add.commute*)
next
case *False*
then have *deg* : *MPoly-Type.degree p var = 2* **using** *degp* **by** *blast*
have *flip* : *A* $*$ (*Var var*)^2 + *B* $*$ *Var var* + *C* = *C* + *B* $*$ *Var var* + *A* $*$ (*Var var*)^2 **using** *add.commute* **by** *auto*
show *?thesis* **using** *sum-over-zero*[**where** *mp*=*p*, **where** *x*=*var*] **apply**(*subst* (*asm*) *deg*) **apply**(*simp add*: *flip*) **apply**(*simp add*: *A-def B-def C-def*)
by (*simp add*: *numeral-2-eq-2*)
qed
have *insert-x* : *insertion* (*nth-default 0* (*xs* @ *x* # Γ)) (*Var var*) = *x* **using** *hlength*
by (*metis add.commute add-strict-increasing insertion-var length-append length-greater-0-conv list.distinct*(*1*) *list-update-id nth-append-length order-refl*)

have h : ($aEval$ (Eq p) (xs @ x # Γ))
proof −
have *freeA* : $var \notin vars\ A$
unfolding *A-def*
by (*simp add*: *not-in-isovarspar*)
have *freeB* : $var \notin vars\ B$
unfolding *B-def*
by (*simp add*: *not-in-isovarspar*)
have *freeC* : $var \notin vars\ C$
unfolding *C-def*
by (*simp add*: *not-in-isovarspar*)
have *xlength* : (*insertion* (*nth-default 0* (*xs* @ *x* # Γ)) (*Var var*))= *x*
using *hlength insertion-var*
using *insert-x* **by** *blast*
define *c* **where** *c i* = (*insertion* (*nth-default 0* (*xs* @ *x* # Γ)) (*isolate-variable-sparse p var i*)) **for** *i*
have *c2* : ∀ *x*. *insertion* (*nth-default 0* (*xs* @ *x* # Γ)) *A* = *c 2*
using *freeA* **apply**(*simp add*: *A-def c-def*)
by (*simp add*: *hlength insertion-lowerPoly1*)
have *c1* : ∀ *x*. *insertion* (*nth-default 0* (*xs* @ *x* # Γ)) *B* = *c 1*
using *freeB* **apply**(*simp add*: *B-def c-def*)
by (*simp add*: *hlength insertion-lowerPoly1*)
have *c0* : ∀ *x*. *insertion* (*nth-default 0* (*xs* @ *x* # Γ)) *C* = *c 0*
using *freeC* **apply**(*simp add*: *C-def c-def*)
by (*simp add*: *hlength insertion-lowerPoly1*)
have *sum* : $c\ 2 * x^2 + c\ (Suc\ 0) * x + c\ 0 = (\sum i \leq 2.\ c\ i * x \hat{}\ i)$
by (*simp add*: *numerals*(*2*))
show *?thesis* **apply**(*subst express-p*) **apply**(*simp add*:*insertion-add insertion-mult insertion-pow xlength*)
apply(*simp add*:*c2 c1 c0 sum polyfun-eq-0*[**where** *c*=*c*, **where** *n*=*2*])
using *True* **apply**(*simp*) **using** *le-SucE numeral-2-eq-2*
by (*metis* (*no-types*) *A-def B-def C-def One-nat-def add.left-neutral c0 c1 c2 mult-zero-class.mult-zero-left sum*)
qed
show *?thesis* **apply**(*rule exI*[**where** *x*=*x*]) **using** *x-def h* **apply**(*simp only*:*eval-list-conj*) **by**(*simp*)
next
case *False*
have *all0* : ∀ *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*
using *False* **apply**(*simp*) **using** *not-in-isovarspar*
by (*metis hlength insertion-lowerPoly1*)
have *h* : *eval* (*elimVar var* ((*map Eq* (*p*# *A'*)) @ *L*) *F* (*Eq p*)) (*xs* @ *x* # Γ)
using *False h* **by** *blast*
have *in-list* : *Eq p* ∈ *set* (((*map Eq* (*p*# *A'*)) @ *L*))
by(*simp*)

show *?thesis* **using** *elimVar-eq-2*[*OF hlength in-list*, **where** *F*=*F*, *OF degp all0*] *h*
by (*metis append-assoc map-append map-map*)
qed
qed

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(*cases luckyFind var L F*)
case *None*
then show *?thesis* **proof**(*cases find-eq var L*)
case (*Pair A L'*)
have *degGood* : ∀ *p*∈*set A*. *MPoly-Type.degree p var* = *1* ∨ *MPoly-Type.degree p var* = *2*
using *degree-find-eq*[*OF Pair*] **.**
have (∃ *x*. *eval* (*qe-eq-repeat var L F*) (*xs* @ *x* # Γ)) =(∃ *x*. *eval*
(*qe-eq-repeat-helper var A L' F*)
(*xs* @ *x* # Γ))
using *Pair None* **by** *auto*
also have ...
= (∃ *x*. ((*eval* (*list-conj* ((*map* (*Atom o Eq*) *A*) @ (*map Atom L'*) @ *F*)) (*xs* @ (*x*#Γ)))))
using *qe-eq-repeat-helper-eval-case1*[*OF hlength degGood*, **where** *L*=*L'*, **where** *F*=*F*, **where** Γ=Γ]
qe-eq-repeat-helper-eval-case2[*OF hlength degGood*, **where** *L*=*L'*, **where** *F*=*F*, **where** Γ=Γ]
by *blast*
also have ... = (∃ *x*. (*eval* (*list-conj* ((*map Atom L*) @ *F*)) (*xs* @ (*x*#Γ))))
proof−
have *list-manipulate* : *map* (*fm.Atom* ∘ *Eq*) *A* @ *map fm.Atom L'* = *map fm.Atom* (*map Eq A* @ *L'*)
by *simp*
have *changeA* : *map* (*fm.Atom* ∘ *Eq*) *A* = *map Atom* (*map Eq A*) **by** *auto*
have *split* : (∃ *x*. ∀ *f*∈*set* ((*map* (*fm.Atom* ∘ *Eq*) *A*) @
(*map fm.Atom L'*) @ *F*).
eval f (*xs* @ *x* # Γ)) = (∃ *x*. ∀ *f*∈ (*Atom* ‘ *set* ((*map* (*Eq*) *A*) @ *L'*)) ∪ *set*(*F*).
eval f (*xs* @ *x* # Γ))
apply (*rule ex-cong1*)
apply(*subst changeA*)
by *auto*
show *?thesis* **apply**(*simp only*: *eval-list-conj split list-in-find-eq*[*OF Pair*]) **by** *auto*
qed
finally have *?thesis* **by** *simp*

then show *?thesis* **by** *auto*
qed
next
case (*Some a*)
then show *?thesis* **using** *luckyFind-eval*[*OF Some assms*(*1*)] **by** *auto*
qed

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*) ∧ (*evalUni* ψ *x*)) |
evalUni (*OrUni* φ ψ) *x* = ((*evalUni* φ *x*) ∨ (*evalUni* ψ *x*))

fun *negUni* :: *atomUni fmUni* ⇒ *atomUni fmUni* **where**
negUni (*AtomUni a*) = *AtomUni*(*aNegUni a*) |
negUni (*TrueFUni*) = *FalseFUni* |
negUni (*FalseFUni*) = *TrueFUni* |

```
  negUni (AndUni φ ψ) = (OrUni (negUni φ) (negUni ψ)) |
  negUni (OrUni φ ψ) = (AndUni (negUni φ) (negUni ψ))

fun convert-poly :: nat ⇒ real mpoly ⇒ real list ⇒ (real * real * real) option
where
  convert-poly var p xs = (
  if MPoly-Type.degree p var < 3
 then let (A,B,C) = get-coeffs var p in Some(insertion (nth-default 0 (xs)) A,insertion
(nth-default 0 (xs)) B,insertion (nth-default 0 (xs)) C)
 else None)

fun convert-atom :: nat ⇒ atom ⇒ real list ⇒ atomUni option where
  convert-atom var (Less p) xs = map-option LessUni (convert-poly var p xs)|
  convert-atom var (Eq p) xs = map-option EqUni (convert-poly var p xs)|
  convert-atom var (Leq p) xs = map-option LeqUni (convert-poly var p xs)|
  convert-atom var (Neq p) xs = map-option NeqUni (convert-poly var p xs)

lemma convert-atom-change :
  assumes length xs' = var
 shows convert-atom var At (xs' @ x # Γ) = convert-atom var At (xs' @ x' # Γ)
  apply(cases At)using assms apply simp-all
  by (metis insertion-lowerPoly1 not-in-isovarspar)+

lemma degree-convert-eq :
  assumes convert-poly var p xs = Some(a)
  shows MPoly-Type.degree p var < 3
  using assms apply(cases MPoly-Type.degree p var < 3) by auto

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−
   have ha: ⋀x. a = insertion (nth-default 0 (xs'@x # xs)) A using assms(2)
apply auto
    by (metis assms(3) assms(6) insertion-lowerPoly1 not-in-isovarspar)
   have hb: ⋀x. b = insertion (nth-default 0 (xs'@x # xs)) B using assms(2)
apply auto
    by (metis assms(4) assms(6) insertion-lowerPoly1 not-in-isovarspar)
   have hc: ⋀x. c = insertion (nth-default 0 (xs'@x # xs)) C using assms(2)
apply auto
    by (metis assms(5) assms(6) insertion-lowerPoly1 not-in-isovarspar)
  show ?thesis
  proof(cases MPoly-Type.degree p var = 0)
    case True
```

have *h1* : *var* < *length* (*xs'*@*x*#*xs*) **using** *assms* **by** *auto*
show *?thesis* **using** *assms ha hb hc sum-over-degree-insertion*[*OF h1 True*, *of y*] **apply**(*simp add*: *isovar-greater-degree*[*of p*] *True*)
using *True degree0isovarspar* **by** *force*
next
case *False*
then have *notzero* : *MPoly-Type.degree p var* ≠ *0* **by** *auto*
show *?thesis* **proof**(*cases MPoly-Type.degree p var* = *1*)
case *True*
have *h1* : *var* < *length* (*xs'*@*x*#*xs*) **using** *assms* **by** *auto*
show *?thesis* **using** *sum-over-degree-insertion*[*OF h1 True*, *of x*, *symmetric*] **unfolding** *assms*(*6*)[*symmetric*] *list-update-length* **unfolding** *assms*(*6*) **apply** *simp* **using** *ha hb hc assms* **apply** *auto*
by (*smt* (*verit*, *ccfv-threshold*) *One-nat-def True express-poly h1 insertion-add insertion-mult insertion-pow insertion-var list-update-length*)
next
case *False*
then have *deg2* : *MPoly-Type.degree p var* = *2* **using** *notzero assms* **by** *auto*
have *h1* : *var* < *length* (*xs'*@*x*#*xs*) **using** *assms* **by** *auto*
have *two* : *2* = *Suc*(*Suc 0*) **by** *auto*
show *?thesis*
using *sum-over-degree-insertion*[*OF h1 deg2*, *of x*, *symmetric*] **unfolding** *assms*(*6*)[*symmetric*] *list-update-length* **unfolding** *assms*(*6*) *two* **apply** *simp* **using** *ha hb hc assms* **apply** *auto*
using *deg2 express-poly h1 insertion-add insertion-mult insertion-pow insertion-var list-update-length*
by (*smt* (*verit*, *best*) *numeral-2-eq-2*)
qed
qed
qed

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(*cases a*)
case (*Less x*)
then show *?thesis*
proof(*cases MPoly-Type.degree x var* < *3*)
case *True*
then show *?thesis*
using *assms* **apply**(*simp add*:*Less*)
using *poly-to-univar*[*OF True*]
by (*metis One-nat-def aEvalUni.simps*(*2*) *get-coeffs.elims*)
next
case *False*
then show *?thesis* **using** *assms Less* **by** *auto*
qed
next

```
  case (Eq x)
  then show ?thesis
  proof(cases MPoly-Type.degree x var < 3)
    case True
    then show ?thesis
      using assms apply(simp add:Eq)
      using poly-to-univar[OF True]
      by (metis One-nat-def aEvalUni.simps(1) get-coeffs.elims)
  next
    case False
    then show ?thesis using assms Eq by auto
  qed
next
  case (Leq x)
  then show ?thesis
  proof(cases MPoly-Type.degree x var < 3)
    case True
    then show ?thesis
      using assms apply(simp add:Leq)
      using poly-to-univar[OF True]
      by (metis One-nat-def aEvalUni.simps(3) get-coeffs.elims)
  next
    case False
    then show ?thesis using assms Leq by auto
  qed
next
  case (Neq x)
  then show ?thesis
  proof(cases MPoly-Type.degree x var < 3)
    case True
    then show ?thesis
      using assms apply(simp add:Neq)
      using poly-to-univar[OF True]
      by (metis One-nat-def aEvalUni.simps(4) get-coeffs.elims)
  next
    case False
    then show ?thesis using assms Neq by auto
  qed
qed


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*
using *assms*
proof(*induction F arbitrary*: *F'*)
case *TrueF*
then show *?case* **by** *auto*
next
case *FalseF*
then show *?case* **by** *auto*
next
case (*Atom x*)
then show *?case* **using** *aEval-aEvalUni* **by** *auto*
next
case (*And F1 F2*)
then show *?case* **apply**(*cases convert-fm var F1* (*xs'*@*x*#*xs*)) **apply** *simp* **apply**(*cases convert-fm var F2* (*xs'*@*x*#*xs*)) **by** *auto*
next
case (*Or F1 F2*)
then show *?case* **apply**(*cases convert-fm var F1* (*xs'*@*x*#*xs*)) **apply** *simp* **apply**(*cases convert-fm var F2* (*xs'*@*x*#*xs*)) **by** *auto*
next
case (*Neg F*)
then show *?case* **by** *auto*
next
case (*ExQ F*)
then show *?case* **by** *auto*
next
case (*AllQ F*)
then show *?case* **by** *auto*
next
case (*ExN x1* φ)
then show *?case* **by** *auto*
next
case (*AllN x1* φ)
then show *?case* **by** *auto*
qed

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(induction F)
 case TrueF
 then show ?case by auto
next
 case FalseF
 then show ?case by auto
next
 case (Atom x)
 then show ?case proof(cases x)
   case (Less p)
    then show ?thesis using Atom apply(cases MPoly-Type.degree p var < 3)
apply auto apply(cases MPoly-Type.degree p var > 0) apply auto
     using degree0isovarspar not-in-isovarspar by blast
```

```
  next
    case (Eq p)
     then show ?thesis using Atom apply(cases MPoly-Type.degree p var < 3)
apply auto apply(cases MPoly-Type.degree p var > 0) apply auto
      using degree0isovarspar not-in-isovarspar by blast
  next
    case (Leq p)
     then show ?thesis using Atom apply(cases MPoly-Type.degree p var < 3)
apply auto apply(cases MPoly-Type.degree p var > 0) apply auto
      using degree0isovarspar not-in-isovarspar by blast
  next
    case (Neq p)
     then show ?thesis using Atom apply(cases MPoly-Type.degree p var < 3)
apply auto apply(cases MPoly-Type.degree p var > 0) apply auto
      using degree0isovarspar not-in-isovarspar by blast
  qed
next
  case (And F1 F2)
  then show ?case apply(cases grab-atoms var F1)
    apply(cases grab-atoms var F2) apply(auto)
    apply(cases grab-atoms var F2) apply(auto)
    apply(cases grab-atoms var F2) by(auto)
next
  case (Or F1 F2)
  then show ?case apply(cases grab-atoms var F1)
    apply(cases grab-atoms var F2) apply(auto)
    apply(cases grab-atoms var F2) apply(auto)
    apply(cases grab-atoms var F2) by(auto)
next
  case (Neg F)
  then show ?case by auto
next
  case (ExQ F)
  then show ?case by auto
next
  case (AllQ F)
  then show ?case by auto
next
  case (ExN x1 F)
  then show ?case by auto
next
  case (AllN x1 F)
  then show ?case by auto
qed

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(*induction F*)
  case *TrueF*
  then show *?case* **by** *auto*
next
  case *FalseF*
  then show *?case* **by** *auto*
next
  case (*Atom a*)
  then show *?case* **apply**(*cases a*) **by** *auto*
next
  case (*And F1 F2*)
  then show *?case*
   by (*smt convert-fm.simps*(*4*) *grab-atoms.simps*(*7*) *isSome.elims*(*2*) *isSome.elims*(*3*) *option.distinct*(*1*) *option.simps*(*5*) *option.split-sel-asm prod.simps*(*2*))
next
  case (*Or F1 F2*)
  then show *?case*
   by (*smt convert-fm.simps*(*5*) *grab-atoms.simps*(*8*) *isSome.elims*(*2*) *isSome.elims*(*3*) *option.distinct*(*1*) *option.simps*(*5*) *option.split-sel-asm prod.simps*(*2*))
next
  case (*Neg F*)
  then show *?case* **by** *auto*
next
  case (*ExQ F*)
  then show *?case* **by** *auto*
next
  case (*AllQ F*)
  then show *?case* **by** *auto*
next
  case (*ExN x1 F*)
  then show *?case* **by** *auto*
next
  case (*AllN x1 F*)
  then show *?case* **by** *auto*
qed

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−
  have *aEval* (*aNeg A*) (*xs'*@*x*#*xs*) = (¬ *aEval A* (*xs'*@*x*#*xs*))
    using *aNeg-aEval*[*of A* (*xs'*@*x*#*xs*)] **by** *auto*
  also have ... = (¬ *aEvalUni A' x*)
    using *assms aEval-aEvalUni* **by** *auto*
  also have ... = *aEvalUni* (*aNegUni A'*) *x*
    by(*cases A'*)(*auto*)

finally show *?thesis* **.**
qed

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*
using *assms*
proof(*induction F arbitrary*:*F'*)
case *TrueF*
then show *?case* **by** *auto*
next
case *FalseF*
then show *?case* **by** *auto*
next
case (*Atom p*)
then show *?case*
using *convert-aNeg*[*of - p*]
by (*smt aNeg-aEval convert-fm.simps*(*1*) *evalUni.simps*(*1*) *eval.simps*(*1*) *eval.simps*(*6*) *map-option-eq-Some negUni.simps*(*1*))
next
case (*And F1 F2*)
then show *?case* **apply** *auto*
apply (*metis* (*no-types*, *lifting*) *evalUni.simps*(*5*) *negUni.simps*(*4*) *option.case-eq-if option.collapse option.distinct*(*1*) *option.sel*)
apply (*smt* (*verit*, *del-insts*) *evalUni.simps*(*5*) *isSome.elims*(*1*) *negUni.simps*(*4*) *option.inject option.simps*(*4*) *option.simps*(*5*))
by (*smt* (*verit*, *del-insts*) *evalUni.simps*(*5*) *isSome.elims*(*1*) *negUni.simps*(*4*) *option.inject option.simps*(*4*) *option.simps*(*5*))
next
case (*Or F1 F2*)
then show *?case* **apply** *auto*
apply (*smt* (*verit*, *del-insts*) *evalUni.simps*(*4*) *isSome.elims*(*1*) *negUni.simps*(*5*) *option.inject option.simps*(*4*) *option.simps*(*5*))
apply (*smt* (*verit*, *del-insts*) *evalUni.simps*(*4*) *isSome.elims*(*1*) *negUni.simps*(*5*) *option.inject option.simps*(*4*) *option.simps*(*5*))
by (*smt* (*verit*, *del-insts*) *evalUni.simps*(*4*) *isSome.elims*(*1*) *negUni.simps*(*5*) *option.inject option.simps*(*4*) *option.simps*(*5*))
next
case (*Neg F*)
then show *?case* **by** *auto*
next
case (*ExQ F*)
then show *?case* **by** *auto*
next
case (*AllQ F*)
then show *?case* **by** *auto*
next
case (*ExN x1 F*)

```
  then show ?case by auto
next
  case (AllN x1 F)
  then show ?case by auto
qed


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)
  by(induction L)(auto)

lemma eval-list-conj-Uni : evalUni (list-conj-Uni A) x = (∀ l∈set A. evalUni l x)
  apply(induction A)by auto

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)
  apply(induction A)by auto

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*
using *assms*
apply(*induction F*)
apply(*auto*)
apply *fastforce*
apply (*metis* (*no-types*, *lifting*) *isSome.elims*(*2*) *isSome.elims*(*3*) *option.case-eq-if*)
apply *fastforce*
by (*metis* (*no-types*, *lifting*) *isSome.elims*(*2*) *isSome.elims*(*3*) *option.case-eq-if*)

lemma *eval-map* : (∃ *l*∈*set*(*map f L*). *evalUni l x*) = (∃ *l*∈*set*(*L*). *evalUni* (*f l*) *x*)
by *auto*

lemma *eval-map-all* : (∀ *l*∈*set*(*map f L*). *evalUni l x*) = (∀ *l*∈*set*(*L*). *evalUni* (*f l*) *x*)
by *auto*

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

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

9.2 Negative Infinity

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

lemma *freeIn-allzero* : *freeIn var* (*allZero p var*)
by (*simp add*: *not-in-isovarspar freeIn-list-conj*)

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−
define *n* **where** *n* = *MPoly-Type.degree p var*
define *k* **where** *k i x* =((*insertion* (*nth-default 0*(*list-update L var x*)) (*isolate-variable-sparse*

p var i))) **for** *i x*
{fix *x*
have (*eval* (*allZero p var*) (*list-update L var x*)) =
(∀ *i*∈{*0..<*(*MPoly-Type.degree p var*)+*1*}. *aEval* (*Eq*(*isolate-variable-sparse p var i*)) (*list-update L var x*))
by (*simp add*: *eval-list-conj atLeast0-lessThan-Suc*)
also have ... = (∀ *i*∈{*0..<*(*MPoly-Type.degree p var*)+*1*}. (*insertion* (*nth-default 0*(*list-update L var x*)) (*isolate-variable-sparse p var i*))=*0*)
by *simp*
also have ... = (∀ *i*≤(*MPoly-Type.degree p var*). (*insertion* (*nth-default 0*(*list-update L var x*)) (*isolate-variable-sparse p var i*))=*0*)
by *fastforce*
also have ... = (∀ *y*. (∑ *i*≤(*MPoly-Type.degree p var*). ((*insertion* (*nth-default 0*(*list-update L var x*)) (*isolate-variable-sparse p var i*)) ∗ *y* ^ *i*))=*0*)
using *polyfun-eq-const*[**where** *n*=*MPoly-Type.degree p var*, **where** *k*=*0*, **where** *c*=λ*i*. (*insertion* (*nth-default 0*(*list-update L var x*)) (*isolate-variable-sparse p var i*))]
by (*metis* (*no-types*, *lifting*) *le-add2 le-add-same-cancel2*)
also have ... = (∀ *y*. (∑ *i*≤*n*. (*k i x*) ∗ *y* ^ *i*)=*0*)
using *k-def n-def* **by** *simp*
finally have (*eval* (*allZero p var*) (*list-update L var x*)) = (∀ *y*. (∑ *i*≤*n*. (*k i x*) ∗ *y* ^ *i*)=*0*)
by *simp*
}
then have *h1* : (∀ *x*. (*eval* (*allZero p var*) (*list-update L var x*))) = (∀ *x*.(∀ *y*. (∑ *i*≤*n*. (*k i x*) ∗ *y* ^ *i*)=*0*))
by *simp*

have (∃ *y*. ∀ *x*<*y*. (∑ *i*≤*n*. (*k i x*)∗ *x*^*i*)= *0*) = (∃ *y*. ∀ *x*<*y*. (∑ *i*≤(*MPoly-Type.degree p var*). (*insertion* (*nth-default 0* (*list-update L var x*))(*isolate-variable-sparse p var i*))∗ *x*^*i*)= *0*)
using *k-def n-def* **by** *simp*
also have ... = (∃ *y*. ∀ *x*<*y*. *insertion* (*nth-default 0* (*list-update L var x*)) (∑ *i*≤(*MPoly-Type.degree p var*). (*isolate-variable-sparse p var i*)∗ *Var var*^*i*)= *0*)
by(*simp add*: *insertion-sum*′ *insertion-mult insertion-pow insertion-var lLength*)
also have ... = (∃ *y*. ∀ *x*<*y*. *insertion* (*nth-default 0* (*list-update L var x*)) *p* = *0*)
using *sum-over-zero* **by** *simp*
also have ... = (∃ *y*. ∀ *x*<*y*. *aEval* (*Eq p*) (*list-update L var x*))
by *simp*
finally have *h2* : (∃ *y*. ∀ *x*<*y*. *aEval* (*Eq p*) (*list-update L var x*)) = (∃ *y*. ∀ *x*<*y*. (∑ *i*≤*n*. (*k i x*)∗ *x*^*i*)= *0*)
by *simp*

have *k-all* : ∀ *x y i*. *k i x* = *k i y*
unfolding *k-def*
by (*simp add*: *insertion-isovarspars-free*)
have *h3a* : (∃ *y*. ∀ *x*<*y*. (∑ *i*≤*n*. (*k i x*)∗ *x*^*i*)= *0*) ⟹ (∀ *x*.(∀ *y*. (∑ *i*≤*n*. (*k i x*)

$* \ y \ \widehat{} \ i)=0))$
proof−
assume $h : (\exists\, y.\ \forall\, x{<}y.\ (\sum i{\leq}n.\ (k\ i\ x){*}\ x\widehat{}i){=}\ 0)$
{fix $z\ y$
assume $h : (\forall\, x{<}y.\ (\sum i{\leq}n.\ (k\ i\ x){*}\ x\widehat{}i){=}\ 0)$
have $\forall\, x{<}y.\forall\, i{\leq}n.\ k\ i\ x = k\ i\ z$
unfolding *k-def*
using *insertion-isovarspars-free* **by** *blast*
then have $* : \forall\, x{<}y.\forall\, i{\leq}n.\ k\ i\ x * x \ \widehat{} \ i = k\ i\ z * x\widehat{}i$
by *auto*
then have $\forall\, x{<}y.\ (\sum i{\leq}n.\ k\ i\ x * x \ \widehat{} \ i) = (\sum i{\leq}n.\ k\ i\ z * x \ \widehat{} \ i)$
by (*metis* (*no-types*, *lifting*) *k-all sum.cong*)
then have $\forall\, x{<}y.\ (\sum i{\leq}n.\ (k\ i\ z){*}\ x\widehat{}i){=}\ 0$
using h **by** *simp*
then have $\neg(\mathit{finite}\ \{x.\ (\sum i{\leq}n.\ k\ i\ z * x \ \widehat{} \ i) = 0\})$
using *infinite-Iio*[**where** $a{=}y$] *Inf-many-def*[**where** $P{=}\lambda x.\ (\sum i{\leq}n.\ k\ i\ z * x \ \widehat{} \ i) = 0$]
by (*smt INFM-iff-infinite frequently-mono lessThan-def*)
then have $\forall\, i{\leq}n.\ k\ i\ z = 0$
using *polyfun-rootbound*[**where** $n{=}n$, **where** $c = \lambda i.\ k\ i\ z$]
by *blast*
}
then have $\forall\, x.\forall\, i{\leq}n.\ k\ i\ x = 0$
using h
by (*meson gt-ex*)
then show *?thesis* **by** *simp*
qed
have *h3b* : $(\forall\, x.(\forall\, y.\ (\sum i{\leq}n.\ (k\ i\ x)\ *\ y \ \widehat{} \ i){=}0)) \Longrightarrow (\exists\, y.\ \forall\, x{<}y.\ (\sum i{\leq}n.\ (k\ i\ x){*}\ x\widehat{}i){=}\ 0)$
proof−
assume $h : (\forall\, x.(\forall\, y.\ (\sum i{\leq}n.\ (k\ i\ x)\ *\ y \ \widehat{} \ i){=}0))$
{fix $z\ y\ x$
have $(\sum i{\leq}n.\ (k\ i\ z){*}\ x\widehat{}i){=}\ 0$
using h *k-all* **by** *blast*
then have $\forall\, i{\leq}n.\ k\ i\ z = 0$
using *polyfun-eq-const*[**where** $k{=}0$, **where** $c = \lambda i.\ k\ i\ z$, **where** $n{=}n$]
by (*metis h*)
}
then have $\forall\, x.\forall\, i{\leq}n.\ k\ i\ x = 0$
by (*meson gt-ex*)
then show *?thesis* **by** *simp*
qed
have *h3* : $(\exists\, y.\ \forall\, x{<}y.\ (\sum i{\leq}n.\ (k\ i\ x){*}\ x\widehat{}i){=}\ 0) = (\forall\, x.(\forall\, y.\ (\sum i{\leq}n.\ (k\ i\ x)\ *\ y \ \widehat{} \ i){=}0))$
using *h3a h3b* **by** *auto*
show *?thesis* **using** *h1 h2 h3* **by** *simp*
qed

lemma *freeIn-altNegInf* : *freeIn var* (*alternateNegInfinity p var*)
proof−
have *h1* : ∀ *i*. *var* ∉ (*vars* (*if* (*i*::*nat*) *mod* 2 = 0 *then* (*Const*(1)::*real mpoly*) *else* (*Const*(−1)::*real mpoly*)))
using *var-not-in-Const*[**where** *var* = *var*, **where** *x*=1] *var-not-in-Const*[**where** *var* = *var*, **where** *x*=−1]
by *simp*
define *g* **where** *g* = (λ*F*.λ*i*.
let a-n = *isolate-variable-sparse p var i in*
let exp = (*if i mod* 2 = 0 *then Const*(1) *else Const*(−1)) *in*
or (*Atom*(*Less* (*exp* ∗ *a-n*)))
(*and* (*Atom* (*Eq a-n*)) *F*)
)
have *h3* : ∀ *i*. ∀ *F*. (*freeIn var F* ⟶ *freeIn var* (*g F i*))
using *g-def h1*
by (*smt PolyAtoms.and-def not-in-isovarspar PolyAtoms.or-def freeIn.simps*(1) *freeIn.simps*(2) *freeIn.simps*(7) *freeIn.simps*(8) *not-in-mult*)
define *L* **where** *L* = ([0..<((*MPoly-Type.degree p var*)+1)])
have ∀ *F*. *freeIn var F* ⟶ *freeIn var* (*foldl* (*g*::*atom fm* ⇒ *nat* ⇒ *atom fm*) *F* (*L*::*nat list*))
proof(*induction L*)
case *Nil*
then show *?case* **by** *simp*
next
case (*Cons a L*)
then show *?case* **using** *h3* **by** *simp*
qed
then have *freeIn var* (*foldl g FalseF L*)
using *freeIn.simps*(6) **by** *blast*
then show *?thesis* **using** *g-def L-def* **by** *simp*
qed

theorem *freeIn-substNegInfinity* : *freeIn var* (*substNegInfinity var A*)
apply(*cases A*) **using** *freeIn-altNegInf freeIn-allzero* **by** *simp-all*

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(cases p')
  case (fields a b c)
  then show ?thesis proof(cases MPoly-Type.degree p var = 0)
    case True
    then show ?thesis
      using assms fields
      by(simp add:eval-list-conj isovar-greater-degree)
  next
    case False
    then have nonzero : MPoly-Type.degree p var ≠ 0 by auto
    then show ?thesis
    proof(cases MPoly-Type.degree p var = 1)
      case True
      then show ?thesis
        using assms fields
        apply(simp add:eval-list-conj isovar-greater-degree)
        by (metis)
    next
      case False
     then have degree2 : MPoly-Type.degree p var = 2 using degree-convert-eq[OF
assms(1)] nonzero by auto
      then show ?thesis
        using assms
        apply(simp add:eval-list-conj isovar-greater-degree)
        using insertion-isovarspars-free list-update-code(2)
        apply auto
        by (metis One-nat-def Suc-1 less-2-cases less-Suc-eq numeral-3-eq-3)
    qed
  qed
qed



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(cases X)
  case (fields a b c)
  then show ?thesis proof(cases MPoly-Type.degree p var = 0)
    case True
    then show ?thesis
      using assms
      apply (simp add: isovar-greater-degree)
      apply auto
    apply (metis aEval.simps(2) eval.simps(1) eval-and eval-false eval-or  mult-one-left)
      by (metis aEval.simps(2) eval.simps(1) eval-or  mult-one-left)
  next
    case False
    then have nonzero : MPoly-Type.degree p var ≠ 0 by auto
    then show ?thesis
    proof(cases MPoly-Type.degree p var = 1)
      case True
      have letbind: eval
     (let a-n = isolate-variable-sparse p var (Suc 0)
      in or (fm.Atom (Less (Const (− 1) ∗ a-n)))
          (and (fm.Atom (Eq a-n))
            (let a-n = isolate-variable-sparse p var 0
                in or (fm.Atom (Less (Const 1 ∗ a-n))) (and (fm.Atom (Eq a-n))
FalseF))))
     (xs'@x#xs) =
    eval
     (or (fm.Atom (Less (Const (− 1) ∗ (isolate-variable-sparse p var (Suc 0)))))
          (and (fm.Atom (Eq (isolate-variable-sparse p var (Suc 0))))
            (or (fm.Atom (Less (Const 1 ∗ (isolate-variable-sparse p var 0)))) (and
(fm.Atom (Eq (isolate-variable-sparse p var 0))) FalseF))))
     (xs'@x#xs)
        by meson
      show ?thesis
        using assms True unfolding fields
         by (simp add: isovar-greater-degree letbind eval-or eval-and insertion-mult
insertion-const)
    next
      case False
     then have degree2 : MPoly-Type.degree p var = 2 using degree-convert-eq[OF
assms(1)] nonzero by auto
      have [0..<3] = [0,1,2]
        by (simp add: upt-rec)
      then have unfold :  (foldl
       (λF i. let a-n = isolate-variable-sparse p var i
              in or (fm.Atom (Less ((if i mod 2 = 0 then Const 1 else Const (− 1))
∗ a-n)))
```

```
                (and (fm.Atom (Eq a-n)) F))
      FalseF [0..<3]) =
    (let a-n = isolate-variable-sparse p var 2
          in or (fm.Atom (Less ((Const 1) * a-n)))
              (and (fm.Atom (Eq a-n))
     (let a-n = isolate-variable-sparse p var (Suc 0)
     in or (fm.Atom (Less (Const (- 1) * a-n)))
        (and (fm.Atom (Eq a-n))
          (let a-n = isolate-variable-sparse p var 0
             in or (fm.Atom (Less (Const 1 * a-n))) (and (fm.Atom (Eq a-n))
FalseF))))))
       by auto
     have letbind : eval
    (foldl
      (λF i. let a-n = isolate-variable-sparse p var i
           in or (fm.Atom (Less ((if i mod 2 = 0 then Const 1 else Const (- 1))
* a-n)))
                (and (fm.Atom (Eq a-n)) F))
      FalseF [0..<3]) (xs'@x#xs) =
     eval

(or (fm.Atom (Less ( Const 1 * (isolate-variable-sparse p var 2))))
                (and (fm.Atom (Eq (isolate-variable-sparse p var 2)))
(or (fm.Atom (Less (Const (- 1) * (isolate-variable-sparse p var (Suc 0)))))
        (and (fm.Atom (Eq (isolate-variable-sparse p var (Suc 0))))
          (or (fm.Atom (Less (Const 1 * (isolate-variable-sparse p var 0)))) (and
(fm.Atom (Eq (isolate-variable-sparse p var 0))) FalseF))))))
(xs'@x#xs) apply (simp add:unfold) by metis
     show ?thesis
       using degree2 assms unfolding fields by (simp add: isovar-greater-degree
eval-or eval-and letbind insertion-mult insertion-const)
   qed
  qed
qed



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
```

using *assms*
proof(*cases A*)
case (*Less p*)
have ∃ *X*. *convert-poly var p* (*xs'* @ *x* # *xs*) = *Some X* **using** *assms Less*
apply(*cases MPoly-Type.degree p var* < *3*) **by** (*simp-all*)
then obtain *X* **where** *X-def*: *convert-poly var p* (*xs'* @ *x* # *xs*) = *Some X* **by** *auto*
then have *A'*: *A'* = *LessUni X* **using** *assms Less* **apply**(*cases MPoly-Type.degree p var* < *3*) **by** (*simp-all*)
show *?thesis* **unfolding** *Less substNegInfinity.simps*
unfolding *convert-alternateNegInfinity*[*OF X-def assms*(*2*)] *A'*
apply(*cases X*) **by** *simp*
next
case (*Eq p*)
then show *?thesis* **using** *assms convert-allZero* **by** *auto*
next
case (*Leq p*)
define *p'* **where** *p'* = (*case convert-poly var p* (*xs'*@*x*#*xs*) *of Some p'* ⇒ *p'*)
have *A'-simp* : *A'* = *LeqUni p'*
using *assms Leq*
using *p'-def* **by** *auto*
have *allZ* : *eval* (*allZero p var*) (*xs'*@*x*#*xs*) = *evalUni* (*allZero' p'*) *x* **using** *convert-allZero*
using *Leq assms p'-def* **by** *auto*
have *altNeg* : *eval* (*alternateNegInfinity p var*) (*xs'*@*x*#*xs*) = *evalUni* (*alternateNegInfinity' p'*) *x* **using** *convert-alternateNegInfinity*
using *Leq assms p'-def* **by** *auto*
show *?thesis*
unfolding *Leq substNegInfinity.simps eval-Or A'-simp substNegInfinityUni.simps evalUni.simps*
using *allZ altNeg* **by** *auto*
next
case (*Neq p*)
then show *?thesis* **using** *assms convert-allZero convert-neg* **by** *auto*
qed

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*)
using *assms* **by** *auto*
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 −
let *?p* = [:*c*, *b*, *a*:]
have *sign ?p x* ≠ *sign ?p y*

using *assms* **unfolding** *sign-def*
apply (*simp add*: *distrib-left mult.commute mult.left-commute power2-eq-square*)
by *linarith*
then have ($\exists$ *w* $\in$ (*root-list ?p*). *x* $\leq$ *w* $\wedge$ *w* $\leq$ *y*) **using** *changes-sign assms* **by** *auto*
then obtain *w* **where** *w-prop*: *w* $\in$ (*root-list ?p*) $\wedge$ *x* $\leq$ *w* $\wedge$ *w* $\leq$ *y* **by** *auto*
then have *a*$*$*w*^2 + *b*$*$*w* + *c* = *0* **unfolding** *root-list-def*
using *add.commute distrib-right mult.assoc mult.commute power2-eq-square*
using *quadratic-poly-eval* **by** *force*
then show *?thesis* **using** *w-prop* **by** *auto*
qed

lemma *no-change-eval-lt*:
fixes *x y*:: *real*
assumes *x* < *y*
assumes ¬($\exists$ *w*. *x* $\leq$ *w* $\wedge$ *w* $\leq$ *y* $\wedge$ *a*$*$*w*^2 + *b*$*$*w* + *c* = *0*)
shows ((*aEvalUni* (*LessUni* (*a*,*b*,*c*)) *x* = *aEvalUni* (*LessUni* (*a*,*b*,*c*)) *y*))
using *change-eval-lt*
using *assms*(*1*) *assms*(*2*) **by** *blast*

lemma *change-eval-neq*:
fixes *x y*:: *real*
assumes ((*aEvalUni* (*NeqUni* (*a*,*b*,*c*)) *x* $\neq$ *aEvalUni* (*NeqUni* (*a*,*b*,*c*)) *y*) $\wedge$ *x* < *y*)
shows ($\exists$ *w*. *x* $\leq$ *w* $\wedge$ *w* $\leq$ *y* $\wedge$ *a*$*$*w*^2 + *b*$*$*w* + *c* = *0*)
using *assms* **by** *auto*

lemma *change-eval-leq*:
fixes *x y*:: *real*
assumes ((*aEvalUni* (*LeqUni* (*a*,*b*,*c*)) *x* $\neq$ *aEvalUni* (*LeqUni* (*a*,*b*,*c*)) *y*) $\wedge$ *x* < *y*)
shows ($\exists$ *w*. *x* $\leq$ *w* $\wedge$ *w* $\leq$ *y* $\wedge$ *a*$*$*w*^2 + *b*$*$*w* + *c* = *0*)
proof −
let *?p* = [:*c*, *b*, *a*:]
have *sign ?p x* $\neq$ *sign ?p y*
using *assms* **unfolding** *sign-def*
apply (*simp add*: *distrib-left mult.commute mult.left-commute power2-eq-square*)
by *linarith*
then have ($\exists$ *w* $\in$ (*root-list ?p*). *x* $\leq$ *w* $\wedge$ *w* $\leq$ *y*) **using** *changes-sign assms* **by** *auto*
then obtain *w* **where** *w-prop*: *w* $\in$ (*root-list ?p*) $\wedge$ *x* $\leq$ *w* $\wedge$ *w* $\leq$ *y* **by** *auto*
then have *a*$*$*w*^2 + *b*$*$*w* + *c* = *0* **unfolding** *root-list-def*
using *add.commute distrib-right mult.assoc mult.commute power2-eq-square*
using *quadratic-poly-eval* **by** *force*
then show *?thesis* **using** *w-prop* **by** *auto*
qed

lemma *change-eval*:

fixes *x y*:: *real*
fixes *At*:: *atomUni*
assumes *xlt*: $x < y$
assumes *noteq*: $((aEvalUni\ At)\ x \neq aEvalUni\ (At)\ y)$
assumes $getPoly\ At = (a,\ b,\ c)$
shows $(\exists\, w.\ x \leq w \wedge w \leq y \wedge a*w\hat{}2 + b*w + c = 0)$
proof −
have *four-types*: $At = (EqUni\ (a,b,c)) \vee At = (LessUni\ (a,b,c)) \vee At = (LeqUni\ (a,b,c)) \vee At = (NeqUni\ (a,b,c))$
using *getPoly.elims assms*(3) **by** *force*
have *eq-h*: $At = (EqUni\ (a,b,c)) \longrightarrow (\exists\, w.\ x \leq w \wedge w \leq y \wedge a*w\hat{}2 + b*w + c = 0)$
using *assms*(1) *assms*(2) *change-eval-eq*
by *blast*
have *less-h*: $At = (LessUni(a,b,c)) \longrightarrow (\exists\, w.\ x \leq w \wedge w \leq y \wedge a*w\hat{}2 + b*w + c = 0)$
using *change-eval-lt assms*
by *blast*
have *leq-h*: $At = (LeqUni(a,b,c)) \longrightarrow (\exists\, w.\ x \leq w \wedge w \leq y \wedge a*w\hat{}2 + b*w + c = 0)$
using *change-eval-leq assms*
by *blast*
have *neq-h*: $At = (NeqUni(a,b,c)) \longrightarrow (\exists\, w.\ x \leq w \wedge w \leq y \wedge a*w\hat{}2 + b*w + c = 0)$
using *change-eval-neq assms*
by *blast*
show *?thesis*
using *four-types eq-h less-h leq-h neq-h*
by *blast*
qed

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 $((aEvalUni\ At)\ x = aEvalUni\ (At)\ y \wedge x < y)$
using *change-eval*
using *assms*(1) *assms*(2) *assms*(3) **by** *blast*

lemma *same-eval''* :
assumes $getPoly\ At = (a,\ b,\ c)$
assumes *nonz*: $a \neq 0 \vee b \neq 0 \vee c \neq 0$
shows $\exists\, x.\ \forall\, y{<}x.\ (aEvalUni\ At\ y = aEvalUni\ At\ x)$
proof −
let *?p* = [:*c*, *b*, *a*:]
have *poly-eval*: $\forall\, y.\ poly\ ?p\ y = a*y\hat{}2 + b*y + c$
by (*simp add*: *distrib-left power2-eq-square*)

have *?p* $\neq$ *0* **using** *nonz* **by** *auto*
then have *finite* {*y*. *poly ?p y* = *0*}
using *poly-roots-finite*
by *blast*
then have *finite* {*y*. *c* + (*a* ∗ *y*^{2} + *b* ∗ *y*) = *0*} ⟹
∀ *y*. *y* ∗ (*b* + *y* ∗ *a*) = *a* ∗ *y*^{2} + *b* ∗ *y* ⟹
finite {*y*. *a* ∗ *y*^{2} + *b* ∗ *y* + *c* = *0*}
proof −
assume *a1*: *finite* {*y*. *c* + (*a* ∗ *y*^{2} + *b* ∗ *y*) = *0*}
have ∀ *x0*. *c* + (*a* ∗ *x0*^{2} + *b* ∗ *x0*) = *a* ∗ *x0*^{2} + *b* ∗ *x0* + *c*
by *simp*
then show *?thesis*
using *a1* **by** *presburger*
qed
then have *finset*: *finite* {*y*. *a*∗*y*^2 + *b*∗*y* + *c* = *0*}
using *poly-eval*
by (*metis* ‹*finite* {*y*. *poly* [:*c*, *b*, *a*:] *y* = *0*}› *poly-roots-set-same*)
then have ∃ *x*. (∀ *y*. *a*∗*y*^2 + *b*∗*y* + *c* = *0* ⟶ *x* < *y*)
proof −
let *?l* = *sorted-list-of-set* {*y*. *a*∗*y*^2 + *b*∗*y* + *c* = *0*}
have ∃ *x*. *x* < *?l* ! *0*
using *infzeros nonz* **by** *blast*
then obtain *x* **where** *x-prop*: *x* < *?l*! *0* **by** *auto*
then have ∀ *y*. *List.member ?l y* ⟶ *x* < *y*
proof *clarsimp*
fix *y*
assume *List.member ?l y*
then have ∃ *n*. *n* < *length ?l* ∧ *?l* ! *n* = *y*
by (*meson in-set-conv-nth in-set-member*)
then obtain *n* **where** *n-prop*: *n* < *length ?l* ∧ *?l* ! *n* = *y*
by *auto*
have *h*: ∀ *n* < *length ?l*. *?l* ! *n* ≥ *?l* !*0* **using** *sorted-iff-nth-mono*
using *sorted-sorted-list-of-set* **by** *blast*
then have *h*: *y* ≥ *?l* ! *0* **using** *n-prop* **by** *auto*
then show *x* < *y* **using** *x-prop* **by** *auto*
qed
then show *?thesis*
using *finset set-sorted-list-of-set in-set-member*
by (*metis* (*mono-tags*, *lifting*) *mem-Collect-eq*)
qed
then obtain *x* **where** *x-prop*: (∀ *y*. *a*∗*y*^2 + *b*∗*y* + *c* = *0* ⟶ *x* < *y*) **by** *auto*
then have *same-as*: ∀ *y*<*x*. (*aEvalUni At y* = *aEvalUni At x*)
using *no-change-eval change-eval assms* **by** (*smt* (*verit*, *ccfv-SIG*))
then show *?thesis* **by** *auto*
qed

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−
let *?At* = (*LessUni* (*a*, *b*, *c*))
have *firsth* : $\bigwedge x.\ (\forall\, y{<}x.\ a * y^2 + b * y + c < 0 \Longrightarrow a{\leq}0)$
proof −
fix *x*
assume *lt*: $\forall\, y{<}x.\ a * y^2 + b * y + c < 0$
have $\exists\, w.\ \forall\, y < w.\ y\hat{}2 > (-b/a){*}y - c/a$ **using** *ysq-dom-y-plus-coeff*[**where** $b = -b/a$, **where** $c = -c/a$]
by *auto*
then have *gtdomhelp*: $a > 0 \Longrightarrow \exists\, w.\ \forall\, y < w.\ a{*}y\hat{}2 > a{*}((-b/a){*}y - c/a)$
by *auto*
have $\forall\, y.\ (a > 0 \longrightarrow a{*}((-b/a){*}y - c/a) = -b{*}y - c)$
by (*simp add*: *right-diff-distrib′*)
then have *gtdom*: $a > 0 \Longrightarrow \exists\, w.\forall\, y < w.\ a{*}y\hat{}2 > -b{*}y - c$
using *gtdomhelp*
by *simp*
then have $a > 0 \Longrightarrow$ *False*
proof −
assume $a > 0$
then have $\exists\, w.\forall\, y < w.\ a{*}y\hat{}2 > -b{*}y - c$ **using** *gtdom* **by** *auto*
then obtain *w* **where** *w-prop*: $\forall\, y < w.\ a{*}y\hat{}2 + b{*}y + c > 0$
by *fastforce*
let *?mn* = *min w x* − *1*
have *gtz*: $a{*}?mn\hat{}2 + b{*}?mn + c > 0$ **using** *w-prop*
by *auto*
have *ltz*: $a{*}?mn\hat{}2 + b{*}?mn + c < 0$ **using** *lt* **by** *auto*
then show *False* **using** *gtz ltz* **by** *auto*
qed
then show $a \leq 0$
by *fastforce*
qed
have *bleq0* : $\bigwedge x.\ (\forall\, y{<}x.\ b * y + c < 0 \Longrightarrow b{\geq}0)$
proof −
fix *x*
assume *lt*: $\forall\, y{<}x.\ b * y + c < 0$
have *gtdom*: $b < 0 \Longrightarrow \exists\, w.\forall\, y < w.\ b{*}y > -c$
by (*metis mult.commute neg-less-divide-eq*)
then have $b < 0 \Longrightarrow$ *False*
proof −
assume $b < 0$
then have $\exists\, w.\forall\, y < w.\ b{*}y > -c$ **using** *gtdom* **by** *auto*
then obtain *w* **where** *w-prop*: $\forall\, y < w\ .b{*}y + c > 0$
by *fastforce*
let *?mn* = *min w x* − *1*
have *gtz*: $b{*}?mn + c > 0$ **using** *w-prop*
by *auto*
have *ltz*: $b{*}?mn + c < 0$ **using** *lt* **by** *auto*
then show *False* **using** *gtz ltz* **by** *auto*

qed
then show $b \geq 0$
by *fastforce*
qed
have *secondh*: $\bigwedge x.\ (\forall y{<}x.\ a * y^2 + b * y + c < 0 \Longrightarrow \neg\ a < 0 \Longrightarrow \neg\ 0 < b \Longrightarrow b = 0)$
using *firsth bleq0*
by (*metis add.commute add.right-neutral less-eq-real-def mult-zero-class.mult-zero-left*)

have *thirdh* : $\bigwedge x.\ \forall y{<}x.\ a * y^2 + b * y + c < 0 \Longrightarrow \neg\ a < 0 \Longrightarrow \neg\ 0 < b \Longrightarrow c < 0$
using *firsth secondh add.commute add.right-neutral infzeros mult-zero-class.mult-zero-left not-numeral-le-zero order-refl*
by (*metis less-eq-real-def*)
have *fourthh* : $a < 0 \Longrightarrow \exists x.\ \forall y{<}x.\ a * y^2 + b * y + c < 0$
proof −
assume *aleq*: $a < 0$
have $\exists (w{::}real).\ \forall (y{::}real).\ (y < w \longrightarrow y\hat{}2 > (-b/a){*}y + (-c/a))$
using *ysq-dom-y-plus-coeff*[**where** $b = -b/a$, **where** $c = -c/a$]
by *blast*
then have *hyp*:$\exists (w{::}real).\ \forall (y{::}real).\ (y < w \longrightarrow a{*}y\hat{}2 \leq a{*}(-b/a){*}y + a{*}(-c/a))$
by (*metis* (*no-types*, *opaque-lifting*) ‹$a < 0$› *distrib-left less-eq-real-def linorder-not-le mult.assoc mult-less-cancel-left*)
have $\forall y.\ a{*}(-b/a){*}y + a{*}(-c/a) = -b{*}y - c$
using ‹$a < 0$› **by** *auto*
then have $\exists (w{::}real).\ \forall (y{::}real).\ (y < w \longrightarrow a{*}y\hat{}2 \leq -b{*}y - c)$
using *hyp* **by** *auto*
then have $\exists (w{::}real).\ \forall (y{::}real).\ (y < w \longrightarrow a{*}y\hat{}2 + b{*}y + c \leq 0)$
by (*metis add.commute add-uminus-conv-diff le-diff-eq mult-minus-left real-add-le-0-iff*)
then obtain *w* **where** *w-prop*: $\forall (y{::}real).\ (y < w \longrightarrow a{*}y\hat{}2 + b{*}y + c \leq 0)$ **by** *auto*
have $\exists x.\ \forall y < x.\ aEvalUni\ ?At\ x = aEvalUni\ ?At\ y$ **using** *same-eval''*
proof −
have *f1*: $\forall x0\ x1.\ ((x0{::}real) < x1) = (\neg\ 0 \leq x0 + -\ 1 * x1)$
by *linarith*
have $a \neq 0$
using ‹$a < 0$› **by** *force*
then obtain *rr* :: $atomUni \Rightarrow real$ **where**
$\forall r.\ 0 \leq r + -\ 1 * rr\ (LessUni\ (a,\ b,\ c)) \vee aEvalUni\ (LessUni\ (a,\ b,\ c))\ r = aEvalUni\ (LessUni\ (a,\ b,\ c))\ (rr\ (LessUni\ (a,\ b,\ c)))$
using *f1* **by** (*metis getPoly.simps*(4) *same-eval''*)
then show *?thesis*
using *f1* **by** *meson*
qed
then obtain *x* **where** *x-prop*: $\forall y < x.\ aEvalUni\ ?At\ x = aEvalUni\ ?At\ y$ **by** *auto*
let *?mn* = *min x w* − *1*
have $\forall y < ?mn.\ \ aEvalUni\ ?At\ y = True \vee aEvalUni\ ?At\ y = False$

```
    using x-prop by auto
  have ∀ y < ?mn. aEvalUni ?At y = False ⟶ a*y^2 + b*y + c ≥ 0
    by auto
  then have ⋀y. ∀y<w. a * y^2 + b * y + c ≤ 0 ⟹
      y < min x w − 1 ⟹
      ¬ a * y^2 + b * y + c < 0 ⟹
      a * y^2 + b * y + c = 0
  proof −
    fix y :: real
    assume a1: y < min x w − 1
    assume a2: ¬ a * y^2 + b * y + c < 0
    assume a3: ∀y<w. a * y^2 + b * y + c ≤ 0
    have y < w
      using a1 by linarith
    then show a * y^2 + b * y + c = 0
      using a3 a2 less-eq-real-def by blast
  qed
  then have ∀ y < ?mn. aEvalUni ?At y = False ⟶ a*y^2 + b*y + c = 0
    using w-prop by auto
  then have ∀ y < ?mn. aEvalUni ?At y = False ⟹ False using infzeros aleq
    by (metis power-zero-numeral zero-less-power2)
  then have ∀ y < ?mn. aEvalUni ?At y = True
  proof −
    { fix rr :: real
      have ∀ r ra. (ra::real) < r ∨ ¬ ra < r + − 1
        by linarith
      then have ¬ rr < min x w − 1 ∨ aEvalUni (LessUni (a, b, c)) rr
          by (metis (no-types) ‹∀y<min x w − 1. aEvalUni (LessUni (a, b,
c)) y = False ⟹ False› ab-group-add-class.ab-diff-conv-add-uminus less-eq-real-def
min-less-iff-disj not-le x-prop) }
    then show ?thesis
      by blast
  qed
  then show ?thesis by auto
qed
have fifthh : b > 0 ⟹ ∃ x. ∀y<x. b * y + c < 0
proof−
  assume bh : b > 0
  show ∃ x. ∀y<x. b * y + c < 0
    apply(rule exI[where x=−c/b])
    apply auto
    using bh
    by (simp add: mult.commute pos-less-minus-divide-eq)
qed
show ?thesis
  apply(auto)
  using firsth apply simp
  using secondh apply simp
  using thirdh apply simp
```

using *fourthh* **apply** *simp*
using *fifthh* **by** *simp*
qed

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 −
have *s1*: $\forall y.\ -1 * a * y^2 + -1 * b * y + -1 * c < 0 \longleftrightarrow\ a * y^2 + b * y + c > 0$
by *auto*
have *s2*: $(-1 * a < 0 \vee -1 * a = 0 \wedge (0 < -1 * b \vee -1 * b = 0 \wedge -1 * c < 0)) \longleftrightarrow$
$(a > 0 \vee a = 0 \wedge (0 > b \vee\ b = 0 \wedge c > 0))$
by *auto*
have $(\exists x.\ \forall y<x.\ -1 * a * y^2 + -1 * b * y + -1 * c < 0) =$
$(-1 * a < 0 \vee -1 * a = 0 \wedge (0 < -1 * b \vee -1 * b = 0 \wedge -1 * c < 0))$
using *inequality-case*[**where** $a = -1*a$, **where** $b = -1*b$, **where** $c= -1*c$]
by *auto*
then show *?thesis*
using *s1 s2* **by** *auto*
qed

lemma *infinity-evalUni-LessUni* : $(\exists x.\ \forall y<x.\ aEvalUni\ (LessUni\ p)\ y) = (evalUni\ (substNegInfinityUni\ (LessUni\ p))\ x)$
proof(*cases p*)
case (*fields a b c*)
show *?thesis*
unfolding *fields* **apply** *simp*
using *inequality-case*[*of a b c*] **.**
qed

lemma *infinity-evalUni-EqUni* : $(\exists x.\ \forall y<x.\ aEvalUni\ (EqUni\ p)\ y) = (evalUni\ (substNegInfinityUni\ (EqUni\ p))\ x)$
proof(*cases p*)
case (*fields a b c*)
show *?thesis*
using *infzeros*[*of - a b c*] **by**(*auto simp add*: *fields*)
qed

lemma *infinity-evalUni-NeqUni* : $(\exists x.\ \forall y<x.\ aEvalUni\ (NeqUni\ p)\ y) = (evalUni\ (substNegInfinityUni\ (NeqUni\ p))\ x)$
proof(*cases p*)
case (*fields a b c*)
show *?thesis*
unfolding *fields* **apply** *simp*
using *inequality-case*[*of a b c*]
using *inequality-case-geq*[*of a b c*]
by (*metis less-numeral-extra*(*3*) *linorder-neqE-linordered-idom mult-eq-0-iff*)

qed

lemma *infinity-evalUni-LeqUni* : $(\exists\, x.\ \forall\, y{<}x.\ aEvalUni\ (LeqUni\ p)\ y) = (evalUni\ (substNegInfinityUni\ (LeqUni\ p))\ x)$
proof(*cases p*)
case (*fields a b c*)
show *?thesis*
unfolding *fields* **apply** *simp*
proof −
have *h1*: $((\exists\, x.\ \forall\, y{<}x.\ a * y^2 + b * y + c < 0) \vee (\exists\, x.\ \forall\, y{<}x.\ a * y^2 + b * y + c = 0)) \longrightarrow (\exists\, x.\ \forall\, y{<}x.\ a * y^2 + b * y + c \leq 0)$
using *less-eq-real-def*
by *auto*
have *h2*: $(\exists\, x.\ \forall\, y{<}x.\ a * y^2 + b * y + c \leq 0) \Longrightarrow ((\exists\, x.\ \forall\, y{<}x.\ a * y^2 + b * y + c < 0) \vee (\exists\, x.\ \forall\, y{<}x.\ a * y^2 + b * y + c = 0))$
proof −
assume *a1*: $(\exists\, x.\ \forall\, y{<}x.\ a * y^2 + b * y + c \leq 0)$
have $\neg(\exists\, x.\ \forall\, y{<}x.\ a * y^2 + b * y + c = 0) \Longrightarrow (\exists\, x.\ \forall\, y{<}x.\ a * y^2 + b * y + c < 0)$
proof −
assume *a2*: $\neg(\exists\, x.\ \forall\, y{<}x.\ a * y^2 + b * y + c = 0)$
then have $a \neq 0 \vee b \neq 0 \vee c \neq 0$ **by** *auto*
then have $(a < 0 \vee a = 0 \wedge (0 < b \vee b = 0 \wedge c < 0))$
proof −
have *x1*: $a > 0 \Longrightarrow False$
proof −
assume $a > 0$
then have $(\exists\, (x{::}real).\ \forall\, (y{::}real){<}x.\ (a{::}real) * y^2 + (b{::}real) * y + (c{::}real) > 0)$ **using** *inequality-case-geq*
by *auto*
then show *?thesis*
using *a1*
by (*meson a2 linorder-not-le min-less-iff-conj*)
qed
then have *x2*: $a = 0 \wedge 0 > b \Longrightarrow False$
proof −
assume $a = 0 \wedge 0 > b$
then have $(\exists\, (x{::}real).\ \forall\, (y{::}real){<}x.\ (a{::}real) * y^2 + (b{::}real) * y + (c{::}real) > 0)$ **using** *inequality-case-geq*
by *blast*
then show *?thesis*
using *a1*
by (*meson a2 linorder-not-le min-less-iff-conj*)
qed
then have *x3*: $a = 0 \wedge b = 0 \wedge c > 0 \Longrightarrow False$
using *a1 a2* **by** *auto*
show *?thesis* **using** *x1 x2 x3*
by (*meson ‹a ≠ 0 ∨ b ≠ 0 ∨ c ≠ 0› linorder-neqE-linordered-idom*)

```
      qed
      then show (∃ x. ∀ y<x. a * y^2 + b * y + c < 0) using inequality-case
        by auto
    qed
    then show ?thesis
      by auto
  qed
  have (∃ x. ∀ y<x. a * y^2 + b * y + c ≤ 0) = (∃ x. ∀ y<x. a * y^2 + b * y + c
< 0) ∨ (∃ x. ∀ y<x. a * y^2 + b * y + c = 0)
    using h1 h2 by auto
  then show (∃ x. ∀ y<x. a * y^2 + b * y + c ≤ 0) =
  (a < 0 ∨ a = 0 ∧ (0 < b ∨ b = 0 ∧ c < 0) ∨ a = 0 ∧ b = 0 ∧ c = 0)
    using inequality-case[of a b c] infzeros[of - a b c] by auto
 qed
qed
```

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

```
lemma infinity-evalUni :
 shows (∃ x. ∀ y<x. aEvalUni At y) = (evalUni (substNegInfinityUni At) x)
proof(cases At)
 case (LessUni p)
 then show ?thesis using infinity-evalUni-LessUni by auto
next
 case (EqUni p)
 then show ?thesis using infinity-evalUni-EqUni by auto
next
 case (LeqUni p)
 then show ?thesis using infinity-evalUni-LeqUni by auto
next
 case (NeqUni p)
 then show ?thesis using infinity-evalUni-NeqUni by auto
qed

end
```

9.3 Infinitesimals

```
theory Infinitesimals
 imports ExecutiblePolyProps LinearCase QuadraticCase NegInfinity Debruijn
begin

lemma freeIn-substInfinitesimalQuadratic :
 assumes var ∉ vars a
   var ∉ vars b
   var ∉ vars c
   var ∉ vars d
 shows freeIn var (substInfinitesimalQuadratic var a b c d At)
proof(cases At)
```

```
  case (Less p)
  show ?thesis unfolding substInfinitesimalQuadratic.simps Less
    apply(rule free-in-quad-fm[of var a b c d (convertDerivative var p)])
    using assms by auto
next
  case (Eq p)
  then show ?thesis apply simp
    apply(rule freeIn-list-conj)
    apply auto
    using not-in-isovarspar by simp-all
next
  case (Leq p)
 then show ?thesis unfolding substInfinitesimalQuadratic.simps Leq freeIn.simps
   using free-in-quad-fm[of var a b c d (convertDerivative var p), OF assms] apply
simp
    apply(rule freeIn-list-conj)
    using not-in-isovarspar by simp-all
next
  case (Neq p)
  then show ?thesis apply (auto simp add:neg-def)
    apply(rule freeIn-list-conj)
    apply auto
    using not-in-isovarspar by simp-all
qed

lemma freeIn-substInfinitesimalQuadratic-fm : assumes var ∉ vars a
  var ∉ vars b
  var ∉ vars c
  var ∉ vars d
showsfreeIn var (substInfinitesimalQuadratic-fm var a b c d F)
proof−
  {fix z
    have freeIn (var+z)
     (liftmap
       (λx. substInfinitesimalQuadratic (var + x) (liftPoly 0 x a) (liftPoly 0 x b)
             (liftPoly 0 x c) (liftPoly 0 x d))
       F z)
      apply(induction F arbitrary:z) apply auto
         apply(rule freeIn-substInfinitesimalQuadratic)
            apply (simp-all add: assms not-in-lift)
        apply (metis (no-types, lifting) add-Suc-right)
       apply (metis (mono-tags, lifting) add-Suc-right)
      apply (simp add: ab-semigroup-add-class.add-ac(1))
     by (simp add: add.assoc)
  }then show ?thesis
    unfolding substInfinitesimalQuadratic-fm.simps
    by (metis (no-types, lifting) add.right-neutral)
qed
```

lemma *freeIn-substInfinitesimalLinear*:
assumes *var* ∉ *vars a var* ∉ *vars b*
shows *freeIn var* (*substInfinitesimalLinear var a b At*)
proof(*cases At*)
case (*Less p*)
show *?thesis* **unfolding** *Less substInfinitesimalLinear.simps*
using *var-not-in-linear-fm*[*of var a b* (*convertDerivative var p*), *OF assms*]
unfolding *linear-substitution-fm.simps linear-substitution-fm-helper.simps* **.**
next
case (*Eq p*)
then show *?thesis* **apply** *simp* **apply**(*rule freeIn-list-conj*)
apply *auto*
using *not-in-isovarspar* **by** *simp-all*
next
case (*Leq p*)
show *?thesis* **unfolding** *Leq substInfinitesimalLinear.simps freeIn.simps*
using *var-not-in-linear-fm*[*of var a b* (*convertDerivative var p*), *OF assms*]
unfolding *linear-substitution-fm.simps linear-substitution-fm-helper.simps* **ap-**
ply *simp* **apply**(*rule freeIn-list-conj*)
apply *auto*
using *not-in-isovarspar* **by** *simp-all*
next
case (*Neq p*)
then show *?thesis* **apply** (*auto simp add*:*neg-def*) **apply**(*rule freeIn-list-conj*)
apply *auto*
using *not-in-isovarspar* **by** *simp-all*
qed

lemma *freeIn-substInfinitesimalLinear-fm*:
assumes *var* ∉ *vars a var* ∉ *vars b*
shows *freeIn var* (*substInfinitesimalLinear-fm var a b F*)
proof−
{fix *z*
have *freeIn* (*var*+*z*)
(*liftmap* (λ*x*. *substInfinitesimalLinear* (*var* + *x*) (*liftPoly 0 x a*) (*liftPoly 0 x*
b)) *F z*)
apply(*induction F arbitrary*:*z*) **apply** *auto*
apply(*rule freeIn-substInfinitesimalLinear*)
apply (*simp-all add*: *assms not-in-lift*)
apply (*metis* (*full-types*) *Suc-eq-plus1 ab-semigroup-add-class.add-ac*(*1*))
apply (*metis* (*full-types*) *Suc-eq-plus1 ab-semigroup-add-class.add-ac*(*1*))
apply (*simp add*: *add.assoc*)
by (*simp add*: *add.assoc*)
}
then show *?thesis*
unfolding *substInfinitesimalLinear-fm.simps*
by (*metis* (*no-types*, *lifting*) *add.right-neutral*)
qed

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(*cases MPoly-Type.degree p var* = *0*)
case *True*
then show *?thesis* **using** *assms* **apply** (*simp add*: *isovar-greater-degree eval-or eval-and insertion-mult insertion-const*)
using *sum-over-zero*[*of p var*] **by** *auto*
next
case *False*
then have *nonzero*: *MPoly-Type.degree p var* ≠ *0* **by** *auto*
then show *?thesis* **proof**(*cases MPoly-Type.degree p var* = *1*)
case *True*
have *h1* : *MPoly-Type.degree p var* < *3* **using** *True* **by** *auto*
have *h2* : *get-coeffs var p* = (*isolate-variable-sparse p var 2*, *isolate-variable-sparse p var 1*, *isolate-variable-sparse p var 0*) **by** *auto*
have *h* : *insertion* (*nth-default 0* (*xs*′ @ *x* # *xs*)) *p* = *b* ∗ *x* + *c*
using *poly-to-univar*[*OF h1 h2 - - - assms*(*2*), *of a x xs b c x*] **using** *assms*(*1*)
apply(*cases MPoly-Type.degree p var* < *3*) **apply** *simp-all*
using *isovar-greater-degree*[*of p var*] **unfolding** *True* **by** *simp*
have *h3*: *MPoly-Type.degree* (*isolate-variable-sparse p var* (*Suc 0*) ∗ *Const 1*) *var* = *0*
using *degree-mult*[*of isolate-variable-sparse p var* (*Suc 0*) *Const 1 var*]
using *degree-isovarspar mult-one-right* **by** *presburger*
show *?thesis*
using *assms True*
unfolding *convertDerivative.simps*[*of - p*] *convertDerivative.simps*[*of -* (*derivative var p*)]

apply (*simp add*: *derivative-def isovar-greater-degree eval-or eval-and insertion-add insertion-mult insertion-const HOL.arg-cong*[*OF sum-over-zero*[*of p var*], *of insertion* (*nth-default var* (*xs'*@*x*#*xs*))] *insertion-var-zero del*:*convertDerivative.simps*)

unfolding *h* *h3* **by**(*simp del*:*convertDerivative.simps*)
next
case *False*
then have *deg2* : *MPoly-Type.degree p var = 2*
by (*metis One-nat-def assms*(*1*) *convert-poly.simps le-SucE less-Suc0 less-Suc-eq-le nonzero numeral-2-eq-2 numeral-3-eq-3 option.distinct*(*1*))
then have *first* : *isolate-variable-sparse p var* (*Suc* (*Suc 0*)) ≠ *0*
by (*metis MPoly-Type.degree-zero isolate-variable-sparse-degree-eq-zero-iff nat.distinct*(*1*) *numeral-2-eq-2*)
have *second* : (*isolate-variable-sparse p var* (*Suc* (*Suc 0*)) ∗ *Var var*)≠*0*
by (*metis MPoly-Type.degree-zero One-nat-def ExecutiblePolyProps.degree-one Zero-not-Suc first mult-eq-0-iff*)
have *other* : *Const* (*2*::*real*)≠*0*
by (*simp add*: *nonzero-const-is-nonzero*)
have *thing*: (*Var var*::*real mpoly*) ≠ *0*
using *second* **by** *auto*
have *degree*: *MPoly-Type.degree*
(*isolate-variable-sparse p var* (*Suc 0*) ∗ *Const 1* +
isolate-variable-sparse p var (*Suc* (*Suc 0*)) ∗ *Var var* ∗ *Const 2*)
var −
Suc 0 = *0*
apply *simp* **apply**(*rule Nat.eq-imp-le*) **apply**(*rule degree-less-sum'*[*of - - 0*])
apply (*simp add*: *degree-isovarspar mult-one-right*) **apply** *auto*
unfolding *degree-mult*[*OF second other*, *of var*] *degree-const* **apply** *simp*
unfolding *degree-mult*[*OF first thing*] *degree-one*
using *degree-isovarspar* **by** *force*
have *insertion*: *insertion* (*nth-default 0* (*xs'*@*x*#*xs*)) (∑ (*i*::*nat*)≤*2*. *isolate-variable-sparse p var i* ∗ *Var var* ^ *i*) = *a* ∗ *x*^*2* + *b* ∗ *x* + *c*
proof(*cases MPoly-Type.degree p var = 1*)
case *True*
then show *?thesis*
using *False* **by** *blast*
next
case *False*
then have *deg2* : *MPoly-Type.degree p var = 2*
by (*metis One-nat-def assms*(*1*) *convert-poly.simps le-SucE less-Suc0 less-Suc-eq-le nonzero numeral-2-eq-2 numeral-3-eq-3 option.distinct*(*1*))
then have *degless3* : *MPoly-Type.degree p var < 3* **by** *auto*
have *thing* : *var*<*length*(*xs'*@*x* # *xs*) **using** *assms* **by** *auto*
have *h* : (∑ *i*≤*2*. *isolate-variable-sparse p var i* ∗ *Var var* ^ *i*) = *p*
using *deg2*
by (*metis sum-over-zero*)
have *three*: (*3*::*nat*) = *Suc*(*Suc*(*Suc*(*0*))) **by** *auto*
have (∑ *i* = *0*..<*3*. *insertion* (*nth-default 0* (*xs'*@*x*#*xs*)) (*isolate-variable-sparse p var i*) ∗ *x* ^ *i*) =

(*insertion* (*nth-default 0* (*xs'@x#xs*)) (*isolate-variable-sparse p var 0*) ∗ *x* ^ *0*) +
(*insertion* (*nth-default 0* (*xs'@x#xs*)) (*isolate-variable-sparse p var* (*1*::*nat*)) ∗ *x* ^ (*1*::*nat*)) +
(*insertion* (*nth-default 0* (*xs'@x#xs*)) (*isolate-variable-sparse p var* (*2*::*nat*)) ∗ *x* ^ (*2*::*nat*))
unfolding *Set-Interval.comm-monoid-add-class.sum.atLeast0-lessThan-Suc three*
proof −
have *insertion* (*nth-default 0* (*xs'@x#xs*)) (*isolate-variable-sparse p var* (*MPoly-Type.degree p var*)) ∗ *x* ^ *MPoly-Type.degree p var* + (*x* ∗ *insertion* (*nth-default 0* (*xs'@x#xs*)) (*isolate-variable-sparse p var 1*) + (*insertion* (*nth-default 0* (*xs'@x#xs*)) (*isolate-variable-sparse p var 0*) + (∑ *n* = *0*..<*0*. *insertion* (*nth-default 0* (*xs'@x#xs*)) (*isolate-variable-sparse p var n*) ∗ *x* ^ *n*))) = *insertion* (*nth-default 0* (*xs'@x#xs*)) (*isolate-variable-sparse p var 0*) + *x* ∗ *insertion* (*nth-default 0* (*xs'@x#xs*)) (*isolate-variable-sparse p var 1*) + *insertion* (*nth-default 0* (*xs'@x#xs*)) (*isolate-variable-sparse p var* (*MPoly-Type.degree p var*)) ∗ *x* ^ *MPoly-Type.degree p var*
by *auto*
then show (∑ *n* = *0*..<*0*. *insertion* (*nth-default 0* (*xs'@x#xs*)) (*isolate-variable-sparse p var n*) ∗ *x* ^ *n*) + *insertion* (*nth-default 0* (*xs'@x#xs*)) (*isolate-variable-sparse p var 0*) ∗ *x* ^ *0* + *insertion* (*nth-default 0* (*xs'@x#xs*)) (*isolate-variable-sparse p var* (*Suc 0*)) ∗ *x* ^ *Suc 0* + *insertion* (*nth-default 0* (*xs'@x#xs*)) (*isolate-variable-sparse p var* (*Suc* (*Suc 0*))) ∗ *x* ^ *Suc* (*Suc 0*) = *insertion* (*nth-default 0* (*xs'@x#xs*)) (*isolate-variable-sparse p var 0*) ∗ *x* ^ *0* + *insertion* (*nth-default 0* (*xs'@x#xs*)) (*isolate-variable-sparse p var 1*) ∗ *x* ^ *1* + *insertion* (*nth-default 0* (*xs'@x#xs*)) (*isolate-variable-sparse p var 2*) ∗ x^2
by (*metis* (*no-types*) *One-nat-def Suc-1 add.commute deg2 mult.commute mult.left-neutral power-0 power-one-right*)
qed
also have ... = *a* ∗ x^2 + *b* ∗ *x* + *c*
unfolding *Power.power-class.power.power-0 Power.monoid-mult-class.power-one-right Groups.monoid-mult-class.mult-1-right*
using *assms* **unfolding** *convert-poly.simps* **using** *degless3* **by** *simp*
finally have *h2* :(∑ *i* = *0*..<*3*. *insertion* (*nth-default 0* (*xs'@x#xs*)) (*isolate-variable-sparse p var i*) ∗ *x* ^ *i*) = *a* ∗ x^2 + *b* ∗ *x* + *c*
.
show *?thesis* **using** *sum-over-degree-insertion*[*OF thing deg2*, *of x*] **apply** *auto* **using** *h h2* **using** *assms*(*2*) **by** *auto*
qed
have *insertionb*: *insertion* (*nth-default 0* (*xs'@x#xs*)) (*isolate-variable-sparse p var* (*Suc 0*)) = *b*
using *assms* **apply**(*cases MPoly-Type.degree p var* < *3*) **by** *simp-all*
have *insertiona* : *insertion* (*nth-default 0* (*xs'@x#xs*)) (*isolate-variable-sparse p var* (*Suc* (*Suc 0*))) = *a*
using *assms* **apply**(*cases MPoly-Type.degree p var* < *3*) **apply** *simp-all*
by (*simp add*: *numeral-2-eq-2*)
have *x* : *insertion* (*nth-default 0* (*xs'* @ *x* # *xs*)) (*Var var*) = *x* **using** *insertion-var*[*of var* (*xs'* @ *x* # *xs*) *x*] **using** *assms*(*2*) **by** *auto*
have *zero1* : *insertion* (*nth-default 0* (*xs'* @ *x* # *xs*))

```
    (isolate-variable-sparse (isolate-variable-sparse p var (Suc 0)) var (Suc 0)) =
0
     by (simp add: degree-isovarspar isovar-greater-degree)
   have zero2 : insertion (nth-default 0 (xs' @ x # xs))
    (isolate-variable-sparse (isolate-variable-sparse p var (Suc (Suc 0))) var 0) =
a
     using degree0isovarspar degree-isovarspar insertiona by presburger
    have zero3 : insertion (nth-default 0 (xs' @ x # xs)) (isolate-variable-sparse
(Var var) var (Suc 0)) = 1 using isolate-var-one
     using MPoly-Type.insertion-one by fastforce
   have zero4 : insertion (nth-default 0 (xs' @ x # xs))
    (isolate-variable-sparse (isolate-variable-sparse p var (Suc (Suc 0))) var (Suc
0)) = 0
     by (simp add: degree-isovarspar isovar-greater-degree)
   have insertion-deriv : insertion (nth-default 0 (xs'@x#xs))
     (isolate-variable-sparse
      (isolate-variable-sparse p var (Suc 0) * Const 1 +
       isolate-variable-sparse p var (Suc (Suc 0)) * Var var * Const 2)
      var (Suc 0)) = 2*a
     unfolding isovarspar-sum isolate-variable-sparse-mult apply auto
    unfolding const-lookup-suc const-lookup-zero Rings.mult-zero-class.mult-zero-right
      Groups.group-add-class.add.group-left-neutral
     unfolding insertion-add insertion-mult insertion-const apply auto
     unfolding zero1 zero2 zero3 zero4 by simp
  have deg2 : MPoly-Type.degree p var = 2 using degree-convert-eq[OF assms(1)]
False nonzero by auto
   then show ?thesis
     using assms
    unfolding convertDerivative.simps[of - p] convertDerivative.simps[of - (derivative
var p)] convertDerivative.simps[of - (derivative var (derivative var p))]
      apply (simp add: x insertionb insertiona insertion-deriv insertion degree
derivative-def isovar-greater-degree eval-or eval-and insertion-add insertion-mult
insertion-const HOL.arg-cong[OF sum-over-zero[of p var], of insertion (nth-default
0 (xs'@x#xs))] insertion-var-zero del:convertDerivative.simps)
     by (smt (z3) One-nat-def degree-isovarspar distrib-right insertion-deriv iso-
late-variable-sparse-ne-zeroD mult-one-right neq0-conv not-one-le-zero zero1)
 qed
qed

fun linearSubstitutionUni :: real ⇒ real ⇒ atomUni ⇒ atomUni fmUni where
 linearSubstitutionUni b c a = (if aEvalUni a (−c/b) then TrueFUni else False-
FUni)

lemma convert-linearSubstitutionUni:
 assumes convert-atom var a (xs'@x#xs) = Some(a')
 assumes insertion (nth-default 0 (xs'@x#xs)) b = B
 assumes insertion (nth-default 0 (xs'@x#xs)) c = C
 assumes B ≠ 0
 assumes var∉(vars b)
```

assumes *var∉(vars c)*
assumes *length xs′ = var*
shows *aEval (linear-substitution var (−c) b a) (xs′@x#xs) = evalUni (linearSubstitutionUni B C a′) x*
using *assms*
proof −
have *aEval a (xs′@(−C/B)#xs) = evalUni (linearSubstitutionUni B C a′) x*
using *assms(1)* **proof**(*cases a*)
case (*Less p*)
then have *MPoly-Type.degree p var < 3* **using** *assms(1)***apply**(*cases MPoly-Type.degree p var < 3*) **by** *auto*
then show *?thesis*
using *Less assms* **apply** *simp*
using *poly-to-univar* **by** *force*
next
case (*Eq p*)
then have *MPoly-Type.degree p var < 3* **using** *assms(1)***apply**(*cases MPoly-Type.degree p var < 3*) **by** *auto*
then show *?thesis*
using *Eq assms* **apply** *simp*
using *poly-to-univar* **by** *force*
next
case (*Leq p*)
then have *MPoly-Type.degree p var < 3* **using** *assms(1)***apply**(*cases MPoly-Type.degree p var < 3*) **by** *auto*
then show *?thesis*
using *Leq assms* **apply** *simp*
using *poly-to-univar* **by** *force*
next
case (*Neq p*)
then have *MPoly-Type.degree p var < 3* **using** *assms(1)***apply**(*cases MPoly-Type.degree p var < 3*) **by** *auto*
then show *?thesis*
using *Neq assms* **apply** *simp*
using *poly-to-univar* **by** *force*
qed
then have *subst* : *aEval a ((xs′@x#xs)[var := − C / B]) = evalUni (linearSubstitutionUni B C a′) x* **using** *assms* **by** *auto*
have *hlength* : *var< length (xs′@x#xs)* **using** *assms* **by** *auto*
have *inB* : *insertion (nth-default 0 ((xs′@x#xs)[var := − C / B])) b = B* **using** *assms* **apply** *auto* **apply**(*cases a*) **apply** *auto*
by (*simp add*: *insertion-lowerPoly1*)+
have *inC* : *insertion (nth-default 0 ((xs′@x#xs)[var := − C / B])) (−c) = −C* **using** *assms* **apply** *auto* **apply**(*cases a*) **apply** *auto*
by (*simp add*: *insertion-lowerPoly1 insertion-neg*)+
have *freenegc* : *var∉vars(−c)* **using** *assms not-in-neg* **by** *auto*
show *?thesis* **using** *linear*[*OF hlength assms(4) freenegc assms(5) inC inB*, *of a*] *subst*
using *var-not-in-eval3*[*OF var-not-in-linear*[*OF freenegc assms(5)*] *assms(7)*]

by (*metis assms*(7) *list-update-length*)
qed

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−
have *lb* : *insertion* (*nth-default 0* (*xs*′@*x*#*xs*)) (*liftPoly 0 0 b*) = *B* **unfolding** *lift00* **using** *assms*(2) **by** *auto*
have *lc* : *insertion* (*nth-default 0* (*xs*′@*x*#*xs*)) (*liftPoly 0 0 c*) = *C* **unfolding** *lift00* **using** *assms*(3) *insertion-neg* **by** *auto*
have *nb* : *var* ∉ *vars* (*liftPoly 0 0 b*) **using** *not-in-lift*[*OF assms*(5), *of 0*] **by** *auto*
have *nc* : *var* ∉ *vars* (*liftPoly 0 0 c*) **using** *not-in-lift*[*OF assms*(6)] *not-in-neg*
using *assms*(6) *lift00* **by** *auto*
then show *?thesis* **using** *assms* **using** *lb lc convert-linearSubstitutionUni*[*OF assms*(1) *lb lc assms*(4) *nb nc*]
by (*simp add*: *lift00*)
qed

lemma *evalUni-if* : *evalUni* (*if cond then TrueFUni else FalseFUni*) *x* = *cond*
by(*cases cond*)(*auto*)

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*
using *degree-less-sum*[*of q var m p n*]
by (*simp add*: *add.commute*)

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(*cases p'*)
case (*fields a' b' c'*)
have *h* : *convert-poly var p* (*xs'*@*x*#*xs*) = *Some* (*a'*, *b'*, *c'*)
using *assms fields* **by** *auto*
have *h2* : ∃ *F'*. *convert-fm var* (*convertDerivative var p*) *xs* = *Some F'*
unfolding *convertDerivative.simps*[*of - p*] *convertDerivative.simps*[*of - derivative var p*] *convertDerivative.simps*[*of - derivative var* (*derivative var p*)]
apply(*auto simp del*: *convertDerivative.simps*)
using *degree-convert-eq h* **apply** *blast*
using *assms*(*1*) *degree-convert-eq* **apply** *blast*
apply (*metis Suc-eq-plus1 degree-derivative gr-implies-not0 less-trans-Suc nat-neq-iff*)
using *assms*(*1*) *degree-convert-eq* **apply** *blast*
apply (*meson assms*(*1*) *degree-convert-eq*)
apply (*metis One-nat-def Suc-eq-plus1 degree-derivative less-2-cases less-Suc-eq nat-neq-iff numeral-3-eq-3 one-add-one*)
using *assms*(*1*) *degree-convert-eq* **apply** *blast*
using *degree-derivative* **apply** *force*
using *assms*(*1*) *degree-convert-eq* **apply** *blast*
apply (*meson assms*(*1*) *degree-convert-eq*)
apply (*metis degree-derivative eq-numeral-Suc less-add-one less-trans-Suc not-less-eq numeral-2-eq-2 pred-numeral-simps*(*3*))
apply (*meson assms*(*1*) *degree-convert-eq*)
using *degree-derivative* **apply** *fastforce*
by (*meson assms*(*1*) *degree-convert-eq*)
have *c'-insertion* : *insertion* (*nth-default 0* (*xs'*@*x*#*xs*)) (*isolate-variable-sparse p var 0*) = *c'*
using *assms fields* **unfolding** *convert-poly.simps* **apply**(*cases MPoly-Type.degree p var* < *3*) **by** *auto*
have *b'-insertion* : *insertion* (*nth-default 0* (*xs'*@*x*#*xs*)) (*isolate-variable-sparse p var* (*Suc 0*)) = *b'*
using *assms fields* **unfolding** *convert-poly.simps* **apply**(*cases MPoly-Type.degree p var* < *3*) **by** *auto*
then have *b'-insertion2* : *insertion* (*nth-default 0* (*xs'*@*x*#*xs*)) (*isolate-variable-sparse p var 1*) = *b'*
by *auto*

have *a′-insertion* : *insertion* (*nth-default 0* (*xs′@x#xs*)) (*isolate-variable-sparse p var 2*) = *a′*
using *assms fields* **unfolding** *convert-poly.simps* **apply**(*cases MPoly-Type.degree p var < 3*) **by** *auto*
have *liftb* : *liftPoly 0 0 b = b* **using** *lift00* **by** *auto*
have *liftc* : *liftPoly 0 0 c = c* **using** *lift00* **by** *auto*
have *b′-insertion′* : *insertion* (*nth-default 0* (*xs′@x#xs*)) (*isolate-variable-sparse* (*isolate-variable-sparse p var* (*Suc 0*)) *var 0*) = *b′*
using *assms fields* **unfolding** *convert-poly.simps* **apply**(*cases MPoly-Type.degree p var < 3*) **apply** *auto*
by (*simp add*: *degree0isovarspar degree-isovarspar*)
have *insertion-into-1* : *insertion* (*nth-default 0* (*xs′@x#xs*)) (*isolate-variable-sparse* (*Const 1*) *var 0*) = *1*
by (*simp add*: *const-lookup-zero insertion-const*)
have *twominusone* : ((*2−1*)::*nat*) = *1* **by** *auto*
show *?thesis*
proof(*cases MPoly-Type.degree p var = 0*)
case *True*
have *simp*: (*convertDerivative var p*)=*Atom*(*Less p*)
using *True*
by *auto*
have *azero* : *a′=0*
by (*metis MPoly-Type.insertion-zero True a′-insertion isolate-variable-sparse-ne-zeroD nat.simps*(*3*) *not-less numeral-2-eq-2 zero-less-iff-neq-zero*)
have *bzero* : *b′=0*
using *True b′-insertion isovar-greater-degree* **by** *fastforce*
show *?thesis* **unfolding** *fields substInfinitesimalLinearUni.simps*
convertDerivativeUni.simps linearSubstitutionUni.simps map-atomUni.simps evalUni.simps evalUni-if aEvalUni.simps
Rings.mult-zero-class.mult-zero-left Rings.mult-zero-class.mult-zero-right Groups.add-0 azero bzero
substInfinitesimalLinear.simps convertDerivative.simps[*of - p*] *True simp liftmap.simps*
linear-substitution.simps
apply (*auto simp add*:*True*)
unfolding *c′-insertion* **by** *auto*
next
case *False*
then have *degnot0* : *MPoly-Type.degree p var ≠ 0* **by** *auto*
then show *?thesis*
proof(*cases MPoly-Type.degree p var = 1*)
case *True*
then have *simp* : *convertDerivative var p = Or* (*fm.Atom* (*Less p*)) (*And* (*fm.Atom* (*Eq p*)) (*fm.Atom* (*Less* (*derivative var p*))))
by (*metis One-nat-def Suc-eq-plus1 add-right-imp-eq convertDerivative.simps degnot0 degree-derivative zero-less-one*)
have *azero* : *a′=0*
by (*metis MPoly-Type.insertion-zero One-nat-def True a′-insertion iso-var-greater-degree lessI numeral-2-eq-2*)

have *degderiv* : *MPoly-Type.degree* (*isolate-variable-sparse p var* (*Suc 0*) ∗ *Const 1*) *var* = *0*
using *degree-mult*
by (*simp add*: *degree-isovarspar mult-one-right*)
show *?thesis*
unfolding *fields substInfinitesimalLinearUni.simps convertDerivativeUni.simps linearSubstitutionUni.simps map-atomUni.simps evalUni.simps evalUni-if aEvalUni.simps Rings.mult-zero-class.mult-zero-left Rings.mult-zero-class.mult-zero-right Groups.add-0 azero substInfinitesimalLinear.simps True simp liftmap.simps linear-substitution.simps eval-Or eval-And liftb liftc*
apply *auto*
unfolding *derivative-def True insertion-sub insertion-mult c′-insertion b′-insertion assms lift00* **apply** *auto*
unfolding *insertion-sub insertion-mult c′-insertion b′-insertion assms lift00*
apply (*smt diff-divide-eq-iff divide-less-0-iff mult-less-0-iff*)
apply (*smt mult-imp-less-div-pos neg-less-divide-eq zero-le-mult-iff*)
using *assms*(*4*) *mult.commute nonzero-mult-div-cancel-left*
apply *smt*
unfolding *degderiv* **apply** *auto*
unfolding *isolate-variable-sparse-mult* **apply** *auto*
unfolding *insertion-mult* **defer**
apply (*smt assms*(*4*) *diff-divide-eq-iff divide-less-0-iff mult-less-0-iff*)
defer
using *assms*(*4*) **apply** *blast*
unfolding *b′-insertion′ insertion-into-1* **apply** *auto*
by (*smt assms*(*4*) *less-divide-eq mult-pos-neg2 no-zero-divisors zero-less-mult-pos*)
next
case *False*
then have *degreetwo* : *MPoly-Type.degree p var* = *2* **using** *degnot0*
by (*metis One-nat-def degree-convert-eq h less-2-cases less-Suc-eq numeral-2-eq-2 numeral-3-eq-3*)
have *two* : (*2*::*nat*) = *Suc*(*Suc 0*) **by** *auto*
have *sum* : (∑ *i* = *0*..<*2*. *isolate-variable-sparse p var i* ∗ (− *c*) ^ *i* ∗ *b* ^ (*2* − *i*)) =
isolate-variable-sparse p var 0 ∗ (− *c*) ^ *0* ∗ *b* ^ (*2* − *0*) + *isolate-variable-sparse p var 1* ∗ (− *c*) ^ *1* ∗ *b* ^ (*2* − *1*)
unfolding *Set-Interval.comm-monoid-add-class.sum.atLeast0-lessThan-Suc two* **by** *auto*
have *a* : *isolate-variable-sparse p var* (*Suc* (*Suc 0*)) ≠ *0*
by (*metis degnot0 degree-isovarspar degreetwo isolate-variable-sparse-degree-eq-zero-iff numeral-2-eq-2*)
have *b* : ((*Var var* ∗ *Const 2*) :: *real mpoly*) ≠ (*0*::*real mpoly*)
by (*metis MPoly-Type.degree-zero ExecutiblePolyProps.degree-one mult-eq-0-iff nonzero-const-is-nonzero zero-neq-numeral zero-neq-one*)
have *degreedeg1* : *MPoly-Type.degree*
(*isolate-variable-sparse p var* (*Suc 0*) ∗ *Const 1* +
isolate-variable-sparse p var (*Suc* (*Suc 0*)) ∗ *Var var* ∗ *Const 2*)

```
        var = 1
      apply(rule degree-less-sum'[where n =0])
      apply (simp add: degree-isovarspar mult-one-right) defer
      apply simp
      using degree-mult[OF a b, of var]
    by (metis (no-types, opaque-lifting) ExecutiblePolyProps.degree-one add.left-neutral
b degree-const degree-isovarspar degree-mult mult.commute mult-zero-class.mult-zero-right)
    have simp : (convertDerivative var p) = Or (fm.Atom (Less p))
   (And (fm.Atom (Eq p))
    (Or (fm.Atom (Less (derivative var p)))
       (And (fm.Atom (Eq (derivative var p))) (fm.Atom (Less (derivative var
(derivative var p)))))))
      using degreetwo
       by (metis One-nat-def Suc-1 Suc-eq-plus1 add-diff-cancel-right' convert-
Derivative.simps degree-derivative neq0-conv zero-less-Suc)
    have a : insertion (nth-default 0 (xs'@x#xs))
   (isolate-variable-sparse
    (isolate-variable-sparse p var (Suc 0) * Const 1 +
     isolate-variable-sparse p var (Suc (Suc 0)) * Var var * Const 2)
     var 0) = b' unfolding isovarspar-sum isolate-variable-sparse-mult apply
auto
    unfolding const-lookup-suc const-lookup-zero Rings.mult-zero-class.mult-zero-right
       Groups.group-add-class.add.group-left-neutral
      by (simp add: b'-insertion' isolate-var-0 mult-one-right)
    have b : insertion (nth-default 0 (xs'@x#xs))
   (isolate-variable-sparse
    (isolate-variable-sparse p var (Suc 0) * Const 1 +
     isolate-variable-sparse p var (Suc (Suc 0)) * Var var * Const 2)
    var (Suc 0)) = 2 * a'
     unfolding isovarspar-sum isolate-variable-sparse-mult apply auto
    unfolding const-lookup-suc const-lookup-zero Rings.mult-zero-class.mult-zero-right
       Groups.group-add-class.add.group-left-neutral
     unfolding insertion-add insertion-mult insertion-const
     by (metis MPoly-Type.insertion-one MPoly-Type.insertion-zero One-nat-def
a'-insertion add.commute add.right-neutral degree0isovarspar degree-isovarspar iso-
late-var-one isovar-greater-degree mult.commute mult.right-neutral mult-zero-class.mult-zero-right
numeral-2-eq-2 zero-less-one)
    have simp-insertion-blob : insertion (nth-default 0 (xs'@x#xs))
   (isolate-variable-sparse
     (isolate-variable-sparse p var (Suc 0) * Const 1 +
      isolate-variable-sparse p var (Suc (Suc 0)) * Var var * Const 2)
     var 0 *
    b -
    isolate-variable-sparse
     (isolate-variable-sparse p var (Suc 0) * Const 1 +
      isolate-variable-sparse p var (Suc (Suc 0)) * Var var * Const 2)
     var (Suc 0) *
    c) = b' * B - 2 * a' * C
     unfolding insertion-sub insertion-mult assms a b by auto
```

have a : *isolate-variable-sparse*
(*isolate-variable-sparse p var* (*Suc 0*) $*$ *Const 1* +
isolate-variable-sparse p var (*Suc* (*Suc 0*)) $*$ *Var var* $*$ *Const 2*)
var (*Suc 0*) $\neq$ *0*
by (*metis MPoly-Type.degree-zero One-nat-def degreedeg1 isolate-variable-sparse-degree-eq-zero-iff zero-neq-one*)
have b' : (*Const 1*::*real mpoly*) $\neq$ *0*
by (*simp add*: *nonzero-const-is-nonzero*)
have *degreeblob* : *MPoly-Type.degree*
(*isolate-variable-sparse*
(*isolate-variable-sparse p var* (*Suc 0*) $*$ *Const 1* +
isolate-variable-sparse p var (*Suc* (*Suc 0*)) $*$ *Var var* $*$ *Const 2*)
var (*Suc 0*) $*$
Const 1)
var = *0*
unfolding *degree-mult*[*OF a b'*, *of var*]
by (*simp add*: *degree-isovarspar degree-eq-iff monomials-Const*)
have *otherblob* : *insertion* (*nth-default 0* (*xs'*@*x*#*xs*))
(*isolate-variable-sparse*
(*isolate-variable-sparse*
(*isolate-variable-sparse p var* (*Suc 0*) $*$ *Const 1* +
isolate-variable-sparse p var (*Suc* (*Suc 0*)) $*$ *Var var* $*$ *Const 2*)
var (*Suc 0*) $*$
Const 1)
var 0) = *2* $*$ *a'* **using** *b*
by (*simp add*: *degree0isovarspar degree-isovarspar mult-one-right*)

have ($c' * B^2 - b' * C * B + a' * C^2 < 0$) = (($c' * B^2 - b' * C * B + a' * C^2$)/($B^2$) < *0*)
by (*simp add*: *assms*(*4*) *divide-less-0-iff*)
also have ... = ((($c' * B^2$)/(B^2) − ($b' * C * B$)/(B^2) + ($a' * C^2$)/(B^2)) < *0*)
by (*metis* (*no-types*, *lifting*) *add-divide-distrib diff-divide-distrib*)
also have ... = ($a' * (C / B)^2 - b' * C / B + c' < 0$)
proof −
{ assume $c' + a' * (C / B)^2 - b' * (C / B) < 0$
then have *?thesis*
by (*simp add*: *assms*(*4*) *power2-eq-square*) **}**
moreover
{ assume ¬ $c' + a' * (C / B)^2 - b' * (C / B) < 0$
then have *?thesis*
by (*simp add*: *power2-eq-square*) **}**
ultimately show *?thesis*
by *fastforce*
qed
finally have *h1*: ($c' * B^2 - b' * C * B + a' * C^2 < 0$) = ($a' * (C / B)^2 - b' * C / B + c' < 0$)
.
have ($c' * B^2 - b' * C * B + a' * C^2 = 0$) = (($c' * B^2 - b' * C * B + a'$

$* C^2)/(B^2) = 0$)
by (*simp add*: *assms*(*4*))
also have ... = $(((c' * B^2)/(B^2) - (b' * C * B)/(B^2) + (a' * C^2)/(B^2)) =$
0)
by (*metis* (*no-types*, *lifting*) *add-divide-distrib diff-divide-distrib*)
also have ... = $(a' * (C / B)^2 - b' * C / B + c' = 0)$
proof −
{ assume $c' + a' * (C * (C / (B * B))) - b' * (C / B) \neq 0$
then have *?thesis*
by (*simp add*: *assms*(*4*) *power2-eq-square*) **}**
moreover
{ assume $c' + a' * (C * (C / (B * B))) - b' * (C / B) = 0$
then have *?thesis*
by (*simp add*: *power2-eq-square*) **}**
ultimately show *?thesis*
by *fastforce*
qed
finally have *h2* : $(c' * B^2 - b' * C * B + a' * C^2 = 0) = (a' * (C / B)^2$
$- b' * C / B + c' = 0)$
.
have *h3* : $((b' * B - 2 * a' * C) * B < 0) = (b' < 2 * a' * C / B)$
by (*smt assms*(*4*) *less-divide-eq zero-le-mult-iff*)
have *h4* : $(b' * B = 2 * a' * C) = (b' = 2 * a' * C / B)$
by (*simp add*: *assms*(*4*) *nonzero-eq-divide-eq*)
show *?thesis* **unfolding** *fields substInfinitesimalLinearUni.simps*
convertDerivativeUni.simps linearSubstitutionUni.simps map-atomUni.simps
evalUni.simps evalUni-if aEvalUni.simps
Rings.mult-zero-class.mult-zero-left Rings.mult-zero-class.mult-zero-right
Groups.add-0
substInfinitesimalLinear.simps degreetwo simp liftmap.simps
linear-substitution.simps eval-Or eval-And liftb liftc
apply *simp*
unfolding *derivative-def degreetwo insertion-sub insertion-mult c′-insertion*
b′-insertion assms **apply** *simp*
unfolding *sum insertion-add insertion-mult insertion-pow insertion-neg*
assms
unfolding *b′-insertion2 c′-insertion a′-insertion*
unfolding *Power.power-class.power.power-0 Groups.monoid-mult-class.mult-1-right*
Groups.cancel-comm-monoid-add-class.diff-zero Power.monoid-mult-class.power-one-right
twominusone degreedeg1 **apply** *simp*
unfolding *insertion-mult assms simp-insertion-blob degreeblob*
unfolding *insertion-mult insertion-sub assms otherblob* **apply** *simp*
unfolding *otherblob h1 h2 h3 h4* **unfolding** *lift00 insertion-neg assms inser-*
tion-isovarspars-free insertion-sum insertion-mult insertion-sub degree0isovarspar
degree-isovarspar mult-one-right insertion-sum-var insertion-pow insertion-neg sum
unfolding *assms b′-insertion c′-insertion a′-insertion insertion-neg inser-*
tion-mult insertion-add insertion-pow **apply** *simp*
by (*smt assms*(*2*) *assms*(*3*) *b′-insertion h1 h2 h3 h4 insertion-mult inser-*
tion-sub mult-one-right simp-insertion-blob)

```
    qed
  qed
qed
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
  using assms
proof(cases a)
  case (Less p)
  have ∃ p'. convert-poly var p (xs'@x#xs) = Some p'
    using Less assms(1) apply(cases MPoly-Type.degree p var < 3) by auto
  then obtain p' where p'-def : convert-poly var p (xs'@x#xs) = Some p' by
auto
  have A'-simp :  a' = LessUni p'
    using assms Less
    using p'-def by auto
 have h1 : eval (convertDerivative var p) (xs'@x#xs) = evalUni (convertDerivativeUni
p') x using convert-convertDerivative
    apply ( cases p')
    using A'-simp Less assms by auto
  show ?thesis unfolding A'-simp using convert-substInfinitesimalLinear-less[OF
p'-def assms(2−7)] unfolding Less by auto
next
  case (Eq p)
  define p' where p' = (case convert-poly var p (xs'@x#xs) of Some p' ⇒ p')
  have A'-simp :  a' = EqUni p'
    using assms Eq
    using p'-def by auto
  show ?thesis
   unfolding Eq A'-simp substInfinitesimalLinear.simps substInfinitesimalLinearUni.simps
    using convert-allZero A'-simp Eq assms by auto
next
  case (Leq p)
  have ∃ p'. convert-poly var p (xs' @ x # xs) = Some p'
    using assms(1) unfolding Leq apply auto apply(cases MPoly-Type.degree p
var < 3) by auto
  then obtain p' where p'-def : convert-poly var p (xs' @ x # xs) = Some p' by
metis
  have A'-simp :  a' = LeqUni p'
    using assms Leq
    using p'-def by auto
 have h1 : eval (convertDerivative var p) (xs'@x#xs) = evalUni (convertDerivativeUni
```

p') x **using** *convert-convertDerivative*
apply(*cases* p')
using A'*-simp Leq assms* **by** *auto*
show *?thesis* **unfolding** A'*-simp Leq substInfinitesimalLinear.simps eval-Or substInfinitesimalLinearUni.simps evalUni.simps*
using *convert-substInfinitesimalLinear-less*[*OF* p'*-def assms*(2−7)]
convert-allZero[*OF* p'*-def assms*(7)] **by** *simp*
next
case (*Neq p*)
have $\exists p'$. *convert-poly var p* (xs' @ x # xs) = *Some* p'
using *assms*(1) **unfolding** *Neq* **apply** *auto* **apply**(*cases MPoly-Type.degree p var* < 3) **by** *auto*
then obtain p' **where** p'*-def* : *convert-poly var p* (xs' @ x # xs) = *Some* p' **by** *metis*
have A'*-simp* : $a' = NeqUni\ p'$
using *assms Neq*
using p'*-def* **by** *auto*
show *?thesis*
unfolding *Neq* A'*-simp substInfinitesimalLinear.simps substInfinitesimalLinearUni.simps*
using *convert-allZero*[*OF* p'*-def assms*(7)]
by (*metis* A'*-simp Neq assms*(1) *assms*(7) *convert-substNegInfinity eval.simps*(6) *eval-neg substNegInfinityUni.simps*(4) *substNegInfinity.simps*(4))
qed

lemma *either-or*:
fixes r :: *real*
assumes a: ($\exists y'>r.\ \forall x\in\{r<..y'\}.$ (*aEvalUni* (*EqUni* (a, b, c)) x) ∨ (*aEvalUni* (*LessUni* (a, b, c)) x))
shows ($\exists y'>r.\ \forall x\in\{r<..y'\}.$ (*aEvalUni* (*EqUni* (a, b, c)) x)) ∨
($\exists y'>r.\ \forall x\in\{r<..y'\}.$ (*aEvalUni* (*LessUni* (a, b, c)) x))
proof (*cases* $a = 0 \wedge b = 0 \wedge c = 0$)
case *True*
then have ($\exists y'>r.\ \forall x\in\{r<..y'\}.$ (*aEvalUni* (*EqUni* (a, b, c)) x))
using *assms* **by** *auto*
then show *?thesis*
by *blast*
next
case *False*
then have *noz*: $a\neq 0 \vee b\neq 0 \vee c\neq 0$ **by** *auto*
obtain *y1'* **where** *y1prop*: $y1' > r \wedge (\forall x\in\{r<..y1'\}.$ (*aEvalUni* (*EqUni* (a, b, c)) x) ∨ (*aEvalUni* (*LessUni* (a, b, c)) x))
using a **by** *auto*
obtain *y2'* **where** *y2prop*: $y2' > r \wedge (\forall x\in\{r<..y2'\}.\ a * x^2 + b * x + c \neq 0)$
using *noz nonzcoeffs*[*of a b c*] **by** *auto*
let *?y* = *min y1' y2'*
have *ygt*: *?y* > r **using** *y1prop y2prop* **by** *auto*
have $\forall x\in\{r<..?y\}.$ (*aEvalUni* (*LessUni* (a, b, c)) x)
using *y1prop y2prop greaterThanAtMost-iff*

```
    by force
  then show ?thesis using ygt
    by blast
qed

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 (cases At = LessUni p)
  case True
  then have LessUni: At = LessUni p by auto
  then show ?thesis proof(cases p)
    case (fields a b c)
    then show ?thesis
      unfolding LessUni fields
      using one-root-a-lt0[where r=C/B, where a= a, where b=b,where c= c]
apply(auto)
      using continuity-lem-lt0-expanded[where a= a, where b = 2 * a * C / B ,
where c = c]  apply (auto)
       using continuity-lem-gt0-expanded[where a = a, where b = 2 * a * C /
B,where c = c, where r = − (C / B)] apply (auto)
      apply (meson less-eq-real-def linorder-not-less)
       using one-root-a-gt0[where r = C/B, where a = a, where b=b, where
c=c] apply (auto)
      using continuity-lem-lt0-expanded[where a= a, where b = 2 * a * C / B,
where c= c]
      apply (auto)
       using continuity-lem-gt0-expanded[where a = a, where b = 2 * a * C /
B,where c = c, where r = − (C / B)]
      apply (auto) apply (meson less-eq-real-def linorder-not-less)
      using case-d1 apply (auto)
      using continuity-lem-lt0-expanded[where a= a, where b = b, where c= c]
      apply (auto)
     using continuity-lem-gt0-expanded[where a = a, where b = b,where c = c,
where r = − (C / B)]
      apply (auto) apply (meson less-eq-real-def linorder-not-less)
      using case-d4 apply (auto)
      using continuity-lem-lt0-expanded[where a= a, where b = b, where c= c]
      apply (auto)
     using continuity-lem-gt0-expanded[where a = a, where b = b,where c = c,
where r = − (C / B)]
      apply (auto)
      by (meson less-eq-real-def linorder-not-le)
  qed
next
  case False
  then have EqUni: At = EqUni p using at-is by auto
```

```
  then show ?thesis proof(cases p)
    case (fields a b c)
    show ?thesis
      apply(auto simp add:EqUni fields)
      using continuity-lem-eq0[where r= −(C/B)] apply blast
      using continuity-lem-eq0[where r= −(C/B)] apply blast
      using continuity-lem-eq0[where r= −(C/B)] apply blast
      using linordered-field-no-ub by blast
  qed
qed


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(cases At)
  case (LessUni p)
  then show ?thesis using infinitesimal-linear'-helper[of At p B C] assms by auto
next
  case (EqUni p)
  then show ?thesis  using infinitesimal-linear'-helper[of At p B C] assms by
(auto)
next
  case (LeqUni p)
  then show ?thesis proof(cases p)
    case (fields a b c)
   have same: ∀ x. aEvalUni (LeqUni p) x = (aEvalUni (EqUni p) x) ∨ (aEvalUni
(LessUni p) x)
      apply (simp add: fields)
      by force
    have ⋀a b c.
       At = LeqUni p ⟹
       p = (a, b, c) ⟹
       (∃ y'>− C / B. ∀ x∈{− C / B<..y'}. aEvalUni At x) =
       evalUni (substInfinitesimalLinearUni B C At) x
    proof −
      fix a b c
      assume atis: At = LeqUni p
      assume p-is:  p = (a, b, c)
      have s1: (∃ y'>− C / B. ∀ x∈{− C / B<..y'}. aEvalUni At x) = (∃ y'>− C
/ B. ∀ x∈{− C / B<..y'}.  (aEvalUni (EqUni p) x) ∨ (aEvalUni (LessUni p) x))
       using atis same aEvalUni.simps(2) aEvalUni.simps(3) fields less-eq-real-def
       by blast
      have s2: ... = (∃ y'>− C / B. ∀ x∈{− C / B<..y'}.  (aEvalUni (EqUni p)
x)) ∨ (∃ y'>− C / B. ∀ x∈{− C / B<..y'}. (aEvalUni (LessUni p) x))
       using either-or[where r = −C/B] p-is
       by blast
     have eq1: (∃ y'>− C / B. ∀ x∈{− C / B<..y'}.  (aEvalUni (EqUni p) x)) =
```

```
(evalUni (substInfinitesimalLinearUni B C (EqUni p)) x)
        using infinitesimal-linear'-helper[where At = EqUni p, where p = p,
where B = B, where C= C]
        assms by auto
      have eq2: (∃ y'>− C / B. ∀ x∈{− C / B<..y'}. (aEvalUni (LessUni p) x))
= (evalUni (substInfinitesimalLinearUni B C (LessUni p)) x)
        using infinitesimal-linear'-helper[where At = LessUni p, where p = p,
where B = B, where C= C]
        assms by auto
      have z1: (∃ y'>− C / B. ∀ x∈{− C / B<..y'}. aEvalUni At x) = ((evalUni
(substInfinitesimalLinearUni B C (EqUni p)) x) ∨ (evalUni (substInfinitesimalLinearUni
B C (LessUni p)) x))
        using s1 s2 eq1 eq2 by auto
     have z2: (evalUni (substInfinitesimalLinearUni B C (EqUni p)) x) ∨ (evalUni
(substInfinitesimalLinearUni B C (LessUni p)) x) = evalUni (substInfinitesimalLinearUni
B C (LeqUni p)) x
        by auto
         have z3: (evalUni (substInfinitesimalLinearUni B C At) x) = evalUni
(substInfinitesimalLinearUni B C (LeqUni p)) x
        using LeqUni by auto
       then have z4: (evalUni (substInfinitesimalLinearUni B C (EqUni p)) x) ∨
(evalUni (substInfinitesimalLinearUni B C (LessUni p)) x) = (evalUni (substInfinitesimalLinearUni
B C At) x)
        using z2 z3 by auto
     let ?a = (evalUni (substInfinitesimalLinearUni B C (EqUni p)) x) ∨ (evalUni
(substInfinitesimalLinearUni B C (LessUni p)) x)
      let ?b = (∃ y'>− C / B. ∀ x∈{− C / B<..y'}. aEvalUni At x)
      let ?c = (evalUni (substInfinitesimalLinearUni B C At) x)
      have t1: ?b = ?a using z1 by auto
      have t2: ?a = ?c using z4
        by (simp add: atis)
      then have ?b = ?c using t1 t2 by auto
      then show (∃ y'>− C / B. ∀ x∈{− C / B<..y'}. aEvalUni At x) = evalUni
(substInfinitesimalLinearUni B C At) x
        by auto
    qed
    then show ?thesis
      using LeqUni fields by blast
  qed
next
  case (NeqUni p)
  then show ?thesis proof(cases p)
    case (fields a b c)
    then show ?thesis unfolding NeqUni fields using nonzcoeffs  by (auto)
  qed
qed

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* ≠ (*0*::*real*)
shows
((*a*′::*real*) ∗ (((*A*::*real*) + (*B*::*real*) ∗ *sqrt* (*C*::*real*)) / (*D*::*real*))^2 + (*b*′::*real*) ∗ (*A* + *B* ∗ *sqrt C*) / *D* + *c*′ < *0* ∨
a′ ∗ ((*A* + *B* ∗ *sqrt C*) / *D*)^2 + *b*′ ∗ (*A* + *B* ∗ *sqrt C*) / *D* + (*c*′::*real*) = *0* ∧
(*b*′ + *a*′ ∗ (*A* + *B* ∗ *sqrt C*) ∗ *2* / *D* < *0* ∨
b′ + *a*′ ∗ (*A* + *B* ∗ *sqrt C*) ∗ *2* / *D* = *0* ∧ *2* ∗ *a*′ < *0*)) =
(*a*′ ∗ ((*A* + *B* ∗ *sqrt C*) / *D*)^2 + *b*′ ∗ (*A* + *B* ∗ *sqrt C*) / *D* + *c*′ < *0* ∨
a′ ∗ ((*A* + *B* ∗ *sqrt C*) / *D*)^2 + *b*′ ∗ (*A* + *B* ∗ *sqrt C*) / *D* + *c*′ = *0* ∧
(*2* ∗ *a*′ ∗ (*A* + *B* ∗ *sqrt C*) / *D* + *b*′ < *0* ∨
2 ∗ *a*′ ∗ (*A* + *B* ∗ *sqrt C*) / *D* + *b*′ = *0* ∧ *a*′ < *0*))
proof (*cases* (*a*′ ∗ ((*A* + *B* ∗ *sqrt C*) / *D*)^2 + *b*′ ∗ (*A* + *B* ∗ *sqrt C*) / *D* + *c*′ < *0*))
case *True*
then show *?thesis*
by *auto*
next
case *False*
have *a*′ ∗ (*A* + *B* ∗ *sqrt C*) ∗ *2* = *2* ∗ *a*′ ∗ (*A* + *B* ∗ *sqrt C*) **by** *auto*
then have *a*′ ∗ (*A* + *B* ∗ *sqrt C*) ∗ *2* / *D* =*2* ∗ *a*′ ∗ (*A* + *B* ∗ *sqrt C*) / *D*
using *dneq* **by** *simp*
then have *b*′ + *a*′ ∗ (*A* + *B* ∗ *sqrt C*) ∗ *2* / *D* = *2* ∗ *a*′ ∗ (*A* + *B* ∗ *sqrt C*) / *D* + *b*′
using *add.commute* **by** *simp*
then have (*b*′ + *a*′ ∗ (*A* + *B* ∗ *sqrt C*) ∗ *2* / *D* < *0* ∨ *b*′ + *a*′ ∗ (*A* + *B* ∗ *sqrt C*) ∗ *2* / *D* = *0* ∧ *a*′ < *0*)
= (*2* ∗ *a*′ ∗ (*A* + *B* ∗ *sqrt C*) / *D* + *b*′ < *0* ∨ *2* ∗ *a*′ ∗ (*A* + *B* ∗ *sqrt C*) / *D* + *b*′ = *0* ∧ *a*′ < *0*)
by (*simp add*: ‹*b*′ + *a*′ ∗ (*A* + *B* ∗ *sqrt C*) ∗ *2* / *D* = *2* ∗ *a*′ ∗ (*A* + *B* ∗ *sqrt C*) / *D* + *b*′›)
then have (*a*′ ∗ ((*A* + *B* ∗ *sqrt C*) / *D*)^2 + *b*′ ∗ (*A* + *B* ∗ *sqrt C*) / *D* + *c*′ = *0* ∧
(*b*′ + *a*′ ∗ (*A* + *B* ∗ *sqrt C*) ∗ *2* / *D* < *0* ∨ *b*′ + *a*′ ∗ (*A* + *B* ∗ *sqrt C*) ∗ *2* / *D* = *0* ∧ *a*′ < *0*)) =

$(a' * ((A + B * sqrt\ C)\ /\ D)^2 + b' * (A + B * sqrt\ C)\ /\ D + c' = 0 \wedge$
$(2 * a' * (A + B * sqrt\ C)\ /\ D + b' < 0 \vee 2 * a' * (A + B * sqrt\ C)\ /\ D + b' = 0 \wedge a' < 0))$
by *blast*
then show *?thesis* **using** *False* **by** *simp*
qed

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 $var \notin (vars\ a)$
assumes $var \notin (vars\ b)$
assumes $var \notin (vars\ c)$
assumes $var \notin (vars\ d)$
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(*cases* p')
case (*fields* a' b' c')
have h : *convert-poly var p* $(xs'@x\#xs)$ = *Some* (a', b', c')
using *assms fields* **by** *auto*
have *h2* : $\exists F'$. *convert-fm var* (*convertDerivative var p*) $(xs'@x\#xs)$ = *Some* F'
unfolding *convertDerivative.simps*[*of - p*] *convertDerivative.simps*[*of - derivative var p*] *convertDerivative.simps*[*of - derivative var* (*derivative var p*)]
apply (*auto simp del*: *convertDerivative.simps*)
using *degree-convert-eq h* **apply** *blast*
using *assms*(*1*) *degree-convert-eq* **apply** *blast*
using *degree-derivative* **apply** *fastforce*
apply (*metis degree-convert-eq h numeral-3-eq-3*)
apply (*metis* (*no-types*, *lifting*) *One-nat-def add.right-neutral add-Suc-right degree-derivative less-Suc-eq-0-disj less-Suc-eq-le neq0-conv numeral-3-eq-3*)
apply (*metis One-nat-def Suc-eq-plus1 degree-derivative less-2-cases less-Suc-eq nat-neq-iff numeral-3-eq-3 one-add-one*)
apply (*meson assms*(*1*) *degree-convert-eq*)
using *degree-derivative* **apply** *fastforce*
using *assms*(*1*) *degree-convert-eq* **apply** *blast*
apply (*meson assms*(*1*) *degree-convert-eq*)
apply (*metis degree-derivative less-Suc-eq less-add-one not-less-eq numeral-3-eq-3*)
apply (*meson assms*(*1*) *degree-convert-eq*)
apply (*metis* (*no-types*, *opaque-lifting*) *Suc-1 Suc-eq-plus1 degree-derivative less-2-cases less-Suc-eq numeral-3-eq-3*)
using *assms*(*1*) *degree-convert-eq* **by** *blast*
have c'-*insertion* : *insertion* (*nth-default 0* $(xs'@x\#xs)$) (*isolate-variable-sparse*

p var 0) = *c'*
using *assms fields* **unfolding** *convert-poly.simps* **apply**(*cases MPoly-Type.degree p var < 3*) **by** *auto*
then have *c'-insertion''*: ⋀*x*. *insertion* (*nth-default 0* (*xs'* @ *x* # *xs*)) (*isolate-variable-sparse p var 0*) = *c'*
using *assms*(*12*) *not-in-isovarspar*[*of p var 0 isolate-variable-sparse p var 0*, *OF HOL.refl*]
by (*metis list-update-length not-contains-insertion*)
have *b'-insertion* : *insertion* (*nth-default 0* (*xs'*@*x*#*xs*)) (*isolate-variable-sparse p var* (*Suc 0*)) = *b'*
using *assms fields* **unfolding** *convert-poly.simps* **apply**(*cases MPoly-Type.degree p var < 3*) **by** *auto*
then have *b'-insertion''*: ⋀*x*. *insertion* (*nth-default 0* (*xs'* @ *x* # *xs*)) (*isolate-variable-sparse p var* (*Suc 0*)) = *b'*
using *assms*(*12*) *not-in-isovarspar*[*of p var Suc 0 isolate-variable-sparse p var* (*Suc 0*), *OF HOL.refl*]
by (*metis list-update-length not-contains-insertion*)
then have *b'-insertion2* : *insertion* (*nth-default 0* (*xs'*@*x*#*xs*)) (*isolate-variable-sparse p var 1*) = *b'*
by *auto*
have *a'-insertion* : *insertion* (*nth-default 0* (*xs'*@*x*#*xs*)) (*isolate-variable-sparse p var 2*) = *a'*
using *assms fields* **unfolding** *convert-poly.simps* **apply**(*cases MPoly-Type.degree p var < 3*) **by** *auto*
then have *a'-insertion''*: ⋀*x*. *insertion* (*nth-default 0* (*xs'* @ *x* # *xs*)) (*isolate-variable-sparse p var 2*) = *a'*
using *assms*(*12*) *not-in-isovarspar*[*of p var 2 isolate-variable-sparse p var 2*, *OF HOL.refl*]
by (*metis list-update-length not-contains-insertion*)
have *liftb* : *liftPoly 0 0 b* = *b* **using** *lift00* **by** *auto*
have *liftc* : *liftPoly 0 0 c* = *c* **using** *lift00* **by** *auto*
have *b'-insertion'* : *insertion* (*nth-default 0* (*xs'*@*x*#*xs*)) (*isolate-variable-sparse* (*isolate-variable-sparse p var* (*Suc 0*)) *var 0*) = *b'*
using *assms fields* **unfolding** *convert-poly.simps* **apply**(*cases MPoly-Type.degree p var < 3*) **apply** *auto*
using *degree0isovarspar degree-isovarspar* **by** *auto*
have *insertion-into-1* : *insertion* (*nth-default 0* (*xs'*@*x*#*xs*)) (*isolate-variable-sparse* (*Const 1*) *var 0*) = *1*
by (*simp add*: *const-lookup-zero insertion-const*)
have *twominusone* : ((*2*−*1*)::*nat*) = *1* **by** *auto*
have *length0* : *var* < *length* (*xs'*@*x*#*xs*) **using** *assms* **by** *auto*
have *altinserta* : ∀ *xa*. *insertion* (*nth-default 0* ((*xs'*@*x*#*xs*)[*var* := *xa*])) *a* = *A*
using *assms* **by** (*metis list-update-length not-contains-insertion*)
have *altinserta'* : ⋀*xa*. *insertion* (*nth-default 0* ((*xs'*@*x*#*xs*)[*var* := *xa*])) *a* = *A*
using *assms* **by** (*metis list-update-length not-contains-insertion*)
have *altinsertb* : ∀ *xa*. *insertion* (*nth-default 0* ((*xs'*@*x*#*xs*)[*var* := *xa*])) *b* = *B*
using *assms* **by** (*metis list-update-length not-contains-insertion*)
have *altinsertb'* : ⋀*xa*. *insertion* (*nth-default 0* ((*xs'*@*x*#*xs*)[*var* := *xa*])) *b* = *B*
using *assms* **by** (*metis list-update-length not-contains-insertion*)

have *altinsertc* : $\forall$ *xa. insertion* (*nth-default 0* ((*xs'*@*x*#*xs*)[*var* := *xa*])) *c* = *C*
using *assms* **by** (*metis list-update-length not-contains-insertion*)
have *altinsertc'* : $\bigwedge$*xa. insertion* (*nth-default 0* ((*xs'*@*x*#*xs*)[*var* := *xa*])) *c* = *C*
using *assms* **by** (*metis list-update-length not-contains-insertion*)
have *altinsertd* : $\forall$ *xa. insertion* (*nth-default 0* ((*xs'*@*x*#*xs*)[*var* := *xa*])) *d* = *D*
using *assms* **by** (*metis list-update-length not-contains-insertion*)
have *altinsertd'* : $\bigwedge$*xa. insertion* (*nth-default 0* ((*xs'*@*x*#*xs*)[*var* := *xa*])) *d* = *D*

using *assms* **by** (*metis list-update-length not-contains-insertion*)
have *freeInQuadraticSub* : $\forall$ *At. eval* (*quadratic-sub var a b c d At*) ((*xs'*@*x*#*xs*)[*var* := *sqrt C*]) = *eval* (*quadratic-sub var a b c d At*) ((*xs'*@*x*#*xs*))
by (*metis assms*(*10*) *assms*(*11*) *assms*(*8*) *assms*(*9*) *free-in-quad list-update-id var-not-in-eval*)
have *quad* : $\bigwedge$*At.* (*eval* (*quadratic-sub var a b c d At*) (*xs'*@*x*#*xs*) =
aEval At ((*xs'*@*x*#*xs*)[*var* := (*A* + *B* ∗ *sqrt C*) / *D*]))
using *quadratic-sub*[*OF length0 assms*(*6*−*7*) *assms*(*10*) *altinserta altinsertb altinsertc altinsertd*, *symmetric*]
using *freeInQuadraticSub* **by** *auto*
show *?thesis*
proof(*cases MPoly-Type.degree p var* = *0*)
case *True*
then have *simp*: (*convertDerivative var p*)=*Atom*(*Less p*)
by *auto*
have *azero* : *a'*=*0*
by (*metis MPoly-Type.insertion-zero True a'-insertion isolate-variable-sparse-ne-zeroD nat.simps*(*3*) *not-less numeral-2-eq-2 zero-less-iff-neq-zero*)
have *bzero* : *b'*=*0*
using *True b'-insertion isovar-greater-degree* **by** *fastforce*
define *p1* **where** *p1* = *isolate-variable-sparse p var 0*
have *degree-p1*: *MPoly-Type.degree p1 var* = *0* **unfolding** *p1-def*
by (*simp add*: *degree-isovarspar*)
define *p2* **where** *p2* = *isolate-variable-sparse p1 var 0* ∗ *Const 0* ∗ *Var var* + *isolate-variable-sparse p1 var 0* ∗ *Const 1*
define *A* **where** *A* = *isolate-variable-sparse p2 var 0*
define *B* **where** *B* = *isolate-variable-sparse p2 var* (*Suc 0*)
show *?thesis*
unfolding *substInfinitesimalQuadratic.simps substInfinitesimalQuadratic-Uni.simps*
fields
convertDerivativeUni.simps map-atomUni.simps quadraticSubUni.simps aEvalUni.simps evalUni.simps evalUni-if
Rings.mult-zero-class.mult-zero-left Groups.add-0 Rings.mult-zero-class.mult-zero-right
True simp azero bzero
quadratic-sub-fm.simps quadratic-sub-fm-helper.simps liftmap.simps lift00
quad aEval.simps
apply (*simp add*:*True c'-insertion p1-def*[*symmetric*] *degree-p1 p2-def*[*symmetric*] *A-def*[*symmetric*] *B-def*[*symmetric*])
unfolding *A-def B-def p2-def p1-def degree0isovarspar*[*OF True*] *isovarspar-sum mult-one-right mult-zero-right mult-zero-left const-lookup-zero const-lookup-suc*

apply *simp*
unfolding *insertion-add insertion-sub insertion-mult insertion-pow insertion-const c'-insertion* **apply** *simp*
using ‹*isolate-variable-sparse p var 0 = p*› *b'-insertion2 bzero c'-insertion* **by** *force*
next
case *False*
then have *degreenonzero* : *MPoly-Type.degree p var ≠0* **by** *auto*
show *?thesis*
proof(*cases MPoly-Type.degree p var = 1*)
case *True*
then have *simp* : *convertDerivative var p = Or* (*fm.Atom* (*Less p*)) (*And* (*fm.Atom* (*Eq p*)) (*fm.Atom* (*Less* (*derivative var p*))))
by (*metis One-nat-def Suc-eq-plus1 add-right-imp-eq convertDerivative.simps degree-derivative degreenonzero less-numeral-extra*(*1*))
have *azero* : *a'=0*
by (*metis MPoly-Type.insertion-zero One-nat-def True a'-insertion isovar-greater-degree lessI numeral-2-eq-2*)
have *degderiv* : *MPoly-Type.degree* (*isolate-variable-sparse p var* (*Suc 0*) ∗ *Const 1*) *var = 0*
using *degree-mult*
by (*simp add*: *degree-isovarspar mult-one-right*)
have *thing* : *var<length* (*xs'*@((*A* + *B* ∗ *sqrt C*) / *D* # *xs*)) **using** *assms* **by** *auto*
have *insertp* : *insertion* (*nth-default 0* (*xs'*@((*A* + *B* ∗ *sqrt C*) / *D* # *xs*))) *p* = *b'* ∗ (*A* + *B* ∗ *sqrt C*) / *D* + *c'*
using *sum-over-degree-insertion*[*OF thing True*, *of* (*A* + *B* ∗ *sqrt C*) / *D*, *symmetric*] **unfolding** *list-update-length assms*(*12*)[*symmetric*] **apply** *simp*
unfolding *assms*(*12*) **unfolding** *c'-insertion'' b'-insertion''* **by** *auto*
have *insertb* : *insertion* (*nth-default 0* (*xs'*@(*A* + *B* ∗ *sqrt C*) / *D* # *xs*)) (*isolate-variable-sparse p var* (*Suc 0*) ∗ *Const 1*) = *b'*
unfolding *insertion-mult insertion-const b'-insertion''* **by** *simp*
show *?thesis*
unfolding *substInfinitesimalQuadratic.simps substInfinitesimalQuadratic-Uni.simps*
fields
convertDerivativeUni.simps map-atomUni.simps quadraticSubUni.simps aEvalUni.simps evalUni.simps evalUni-if
Rings.mult-zero-class.mult-zero-left Groups.add-0 Rings.mult-zero-class.mult-zero-right
True simp azero
quadratic-sub-fm.simps quadratic-sub-fm-helper.simps liftmap.simps lift00
quad aEval.simps eval.simps derivative-def Groups.monoid-add-class.add-0-right
apply *simp*
unfolding *insertp insertb insertion-mult insertion-const*
using *assms*(*12*) *b'-insertion'' insertp* **by** *force*
next
case *False*
then have *degree2* : *MPoly-Type.degree p var = 2* **using** *degreenonzero*
using *h less-Suc-eq* **by** *fastforce*

have *simp* : (*convertDerivative var p*) = *Or* (*fm.Atom* (*Less p*))
(*And* (*fm.Atom* (*Eq p*))
(*Or* (*fm.Atom* (*Less* (*derivative var p*)))
(*And* (*fm.Atom* (*Eq* (*derivative var p*))) (*fm.Atom* (*Less* (*derivative var* (*derivative var p*)))))))
by (*metis One-nat-def Suc-eq-plus1 add-diff-cancel-right' convertDerivative.simps degree2 degree-derivative degreenonzero neq0-conv one-add-one*)
have *insertionp* : *var* < *length* (*xs'*@(*A* + *B* ∗ *sqrt C*) / *D* # *xs*) **using** *assms* **by** *auto*
have *three* : *3* = *Suc*(*Suc*(*Suc*(*0*))) **by** *auto*
have *two* : *2* = *Suc*(*Suc*(*0*)) **by** *auto*
have *insertionp* : *insertion* (*nth-default 0* ((*xs'*@*x* # *xs*)[*var* := (*A* + *B* ∗ *sqrt C*) / *D*])) *p* = *a'* ∗ ((*A* + *B* ∗ *sqrt C*) / *D*)^2 + *b'* ∗ (*A* + *B* ∗ *sqrt C*) / *D* + *c'*
using *sum-over-degree-insertion*[*OF insertionp degree2*, *of* (*A* + *B* ∗ *sqrt C*) / *D*, *symmetric*] **unfolding**
a'-insertion[*symmetric*] *b'-insertion*[*symmetric*] *c'-insertion*[*symmetric*]
insertion-isovarspars-free[*of* - - (*A* + *B* ∗ *sqrt C*) / *D* - - *x*]
unfolding *two* **apply** *simp*
using *assms*(*12*) **by** *force*
have *insertion-simp* : *insertion* (*nth-default 0* ((*xs'* @ *x* # *xs*)[*var* := (*A* + *B* ∗ *sqrt C*) / *D*])) = *insertion* (*nth-default 0* ((*xs'* @ ((*A* + *B* ∗ *sqrt C*) / *D*) # *xs*)))
using *assms*
by (*metis list-update-length*)
have *degreeone* : *MPoly-Type.degree*
(*isolate-variable-sparse p var* (*Suc 0*) ∗ *Const 1* +
isolate-variable-sparse p var (*Suc* (*Suc 0*)) ∗ *Var var* ∗ *Const 2*)
var = *1*
apply(*rule degree-less-sum'*[**where** *n=0*])
apply (*simp add*: *degree-isovarspar mult-one-right*)
apply (*smt One-nat-def ExecutiblePolyProps.degree-one degree2 degree-const degree-isovarspar degree-mult degreenonzero isolate-variable-sparse-degree-eq-zero-iff mult.commute nonzero-const-is-nonzero numeral-2-eq-2 plus-1-eq-Suc*)
by *simp*
have *zero1* : *insertion* (*nth-default 0* (*xs'* @ (*A* + *B* ∗ *sqrt C*) / *D* # *xs*))
(*isolate-variable-sparse* (*isolate-variable-sparse p var* (*Suc 0*)) *var* (*Suc 0*)) = *0*
by (*simp add*: *degree-isovarspar isovar-greater-degree*)
have *zero2* : *insertion* (*nth-default 0* (*xs'* @ (*A* + *B* ∗ *sqrt C*) / *D* # *xs*))
(*isolate-variable-sparse* (*isolate-variable-sparse p var* (*Suc* (*Suc 0*))) *var 0*) = *a'*
using *a'-insertion''* *degree0isovarspar degree-isovarspar numeral-2-eq-2* **by** *force*
have *zero3* : *insertion* (*nth-default 0* (*xs'* @ (*A* + *B* ∗ *sqrt C*) / *D* # *xs*)) (*isolate-variable-sparse* (*Var var*) *var* (*Suc 0*)) = *1*
using *isolate-var-one* **by** *fastforce*
have *zero4* : *insertion* (*nth-default 0* (*xs'* @ (*A* + *B* ∗ *sqrt C*) / *D* # *xs*))
(*isolate-variable-sparse* (*isolate-variable-sparse p var* (*Suc* (*Suc 0*))) *var* (*Suc 0*)) = *0*

```
      by (simp add: degree-isovarspar isovar-greater-degree)
    have insertiona' : insertion (nth-default 0 (xs' @ (A + B * sqrt C) / D #
xs))
     (isolate-variable-sparse
      (isolate-variable-sparse p var (Suc 0) * Const 1 +
       isolate-variable-sparse p var (Suc (Suc 0)) * Var var * Const 2)
      var (Suc 0) *
      Const 1) = 2 * a'
      unfolding isovarspar-sum isolate-variable-sparse-mult apply auto
    unfolding const-lookup-suc const-lookup-zero Rings.mult-zero-class.mult-zero-right
      Groups.group-add-class.add.group-left-neutral
      unfolding insertion-add insertion-mult insertion-const b'-insertion' apply
auto
      unfolding zero1 zero2 zero3 zero4 by auto
     have a' : insertion (nth-default 0 (xs' @ (A + B * sqrt C) / D # xs))
(isolate-variable-sparse p var (Suc (Suc 0))) = a'
      unfolding two[symmetric] unfolding a'-insertion'' by auto
    have var: insertion (nth-default 0 (xs' @ (A + B * sqrt C) / D # xs)) (Var
var) = (A + B * sqrt C) / D using assms
      by (metis insertion-simp insertion-var length0)
    show ?thesis
      unfolding substInfinitesimalQuadratic.simps substInfinitesimalQuadratic-
Uni.simps
       fields
        convertDerivativeUni.simps map-atomUni.simps quadraticSubUni.simps
aEvalUni.simps evalUni.simps evalUni-if
      Rings.mult-zero-class.mult-zero-left Groups.add-0 Rings.mult-zero-class.mult-zero-right
       degree2 simp
        quadratic-sub-fm.simps quadratic-sub-fm-helper.simps liftmap.simps lift00
Groups.monoid-add-class.add-0-right
       quad aEval.simps eval.simps derivative-def apply (simp add:insertion-sum
insertion-add insertion-mult insertion-const insertion-var-zero)
      unfolding insertionp
      unfolding insertion-simp
      unfolding b'-insertion'' a'-insertion''
      unfolding
       degreeone apply simp
      unfolding a' var
      unfolding insertiona'
      using weird[OF assms(6)] by auto
   qed
  qed
qed

lemma convert-substInfinitesimalQuadratic:
  assumes convert-atom var At (xs'@ x#xs) = Some(At')
  assumes insertion (nth-default 0 (xs'@ x#xs)) a = A
  assumes insertion (nth-default 0 (xs'@ x#xs)) b = B
  assumes insertion (nth-default 0 (xs'@ x#xs)) c = C
```

assumes *insertion* (*nth-default 0* (*xs'*@ *x*#*xs*)) *d* = *D*
assumes *D* ≠ *0*
assumes *0* ≤ *C*
assumes *var*∉(*vars a*)
assumes *var*∉(*vars b*)
assumes *var*∉(*vars c*)
assumes *var*∉(*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*
using *assms*
proof(*cases At*)
case (*Less p*)
define *p'* **where** *p'* = (*case convert-poly var p* (*xs'*@ *x*#*xs*) *of Some p'* ⇒ *p'*)
have *At'-simp* : *At'* = *LessUni p'*
using *assms Less*
using *p'-def* **by** *auto*
show *?thesis*
using *convert-substInfinitesimalQuadratic-less*[*OF - assms*(*2*−*12*)]
by (*metis At'-simp Less None-eq-map-option-iff assms*(*1*) *convert-atom.simps*(*1*) *option.distinct*(*1*) *option.exhaust-sel option.the-def p'-def substInfinitesimalQuadraticUni.simps*(*2*) *substInfinitesimalQuadratic.simps*(*2*))
next
case (*Eq p*)
define *p'* **where** *p'* = (*case convert-poly var p* (*xs'*@ *x*#*xs*) *of Some p'* ⇒ *p'*)
have *At'-simp* : *At'* = *EqUni p'*
using *assms Eq*
using *p'-def* **by** *auto*
show *?thesis*
unfolding *At'-simp Eq substInfinitesimalQuadraticUni.simps substInfinitesimalQuadratic.simps*
using *At'-simp Eq assms*(*1*) *convert-substNegInfinity assms*(*12*) **by** *fastforce*
next
case (*Leq p*)
define *p'* **where** *p'* = (*case convert-poly var p* (*xs'*@ *x*#*xs*) *of Some p'* ⇒ *p'*)
have *At'-simp* : *At'* = *LeqUni p'*
using *assms Leq*
using *p'-def* **by** *auto*
have *allzero* : *eval* (*allZero p var*) (*xs'*@ *x*#*xs*) = *evalUni* (*allZero' p'*) *x*
using *Leq assms*(*1*) *convert-allZero p'-def assms*(*12*) **by** *force*
have *less* : *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*
using *convert-substInfinitesimalQuadratic-less*[*OF - assms*(*2*−*12*)]
by (*metis Leq assms*(*1*) *convert-atom.simps*(*3*) *option.distinct*(*1*) *option.exhaust-sel option.map*(*1*) *option.the-def p'-def*)
show *?thesis*
unfolding *At'-simp Leq substInfinitesimalQuadraticUni.simps substInfinitesimalQuadratic.simps*

eval.simps evalUni.simps
using *allzero less* **by** *auto*
next
case (*Neq p*)
define p' **where** $p' = (case\ convert\text{-}poly\ var\ p\ (xs'@\ x\#xs)\ of\ Some\ p' \Rightarrow p')$
have *At'-simp* : $At' = NeqUni\ p'$
using *assms Neq*
using *p'-def* **by** *auto*
show *?thesis*
unfolding *At'-simp Neq substInfinitesimalQuadraticUni.simps substInfinitesimalQuadratic.simps*
by (*metis assms*(*12*) *At'-simp Neq assms*(*1*) *convert-substNegInfinity eval.simps*(*6*) *eval-neg substNegInfinityUni.simps*(*4*) *substNegInfinity.simps*(*4*))
qed

lemma *infinitesimal-quad-helper*:
fixes *A B C D*:: *real*
assumes *at-is*: $At = LessUni\ p \vee At = EqUni\ p$
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*)
$= (evalUni\ (substInfinitesimalQuadraticUni\ A\ B\ C\ D\ At)\ x)$
proof(*cases At* = *LessUni p*)
case *True*
then have *LessUni*: *At* = *LessUni p* **by** *auto*
then show *?thesis* **proof**(*cases p*)
case (*fields a b c*)
show *?thesis*
proof(*cases* $2 * (a::real) * (A + B * sqrt\ C) / D + b = 0$)
case *True*
then have *True1* : $2 * a * (A + B * sqrt\ C) / D + b = 0$ **by** *auto*
show *?thesis* **proof**(*cases* $a * ((A + B * sqrt\ C) / D)^2 + b * (A + B * sqrt\ C) / D + c = 0$)
case *True*
then have *True2* : $a * ((A + B * sqrt\ C) / D)^2 + b * (A + B * sqrt\ C) / D + c = 0$ **by** *auto*
then show *?thesis* **proof**(*cases a*<*0*)
case *True*
then show *?thesis* **unfolding** *LessUni fields* **apply** (*simp add*:*True1 True2 True*)
using *True1 True2 True*
proof −
assume *beq*: $2 * a * (A + B * sqrt\ C) / D + b = 0$
assume *root*: $a * ((A + B * sqrt\ C) / D)^2 + b * (A + B * sqrt\ C) / D + c = 0$
assume *alt*: $a < 0$
let *?r* = $-((A + B * sqrt\ C) / D)$
have *beq-var*: $b = 2 * a * ?r$ **using** *beq*

by *auto*
have *root-var*: $a * ?r\hat{}2 - 2*a*?r*?r + c = 0$ **using** *root*
by (*simp add*: *beq-var*)
have $\exists\, y'>-\ ?r.\ \forall\, x\in\{-\ ?r<..y'\}.\ a * x^2 + 2 * a *?r * x + c < 0$
using *beq-var root-var alt one-root-a-lt0*[**where** $a = a$, **where** b=b, **where** c=c, **where** r=*?r*]
by *auto*
then show $\exists\, y'>(A + B * sqrt\ C)\ /\ D.\ \forall\, x\in\{(A + B * sqrt\ C)\ /\ D<..y'\}.\ a * x^2 + b * x + c < 0$
using *beq-var* **by** *auto*
qed
next
case *False*
then show *?thesis* **unfolding** *LessUni fields* **apply** (*simp add*:*True1 True2 False*)
using *True1 True2 False*
proof *clarsimp*
fix y'
assume *beq*: $2 * a * (A + B * sqrt\ C)\ /\ D + b = 0$
assume *root*: $a * ((A + B * sqrt\ C)\ /\ D)^2 + b * (A + B * sqrt\ C)\ /\ D + c = 0$
assume *agteq*: $\neg\ a < 0$
assume *y-prop*: $(A + B * sqrt\ C)\ /\ D < y'$
have *beq-var*: $b = 2 * a * (-\ A - B * sqrt\ C)\ /\ D$ **using** *beq*
by (*metis* (*no-types*, *opaque-lifting*) *ab-group-add-class.ab-diff-conv-add-uminus add.left-neutral add-diff-cancel-left' divide-inverse mult.commute mult-minus-right*)

have *root-var*: $a * ((-\ A - B * sqrt\ C)\ /\ D)^2 - 2 * a * (-\ A - B * sqrt\ C) * (-\ A - B * sqrt\ C)\ /\ (D * D) + c = 0$
using *root*
proof −
have *f1*: $\bigwedge r\ ra.\ -\ ((r::real) + ra) = -\ r - ra$
by *auto*
have *f2*: $\bigwedge r\ ra.\ r * (a * 2 * (-\ A - B * sqrt\ C))\ /\ (ra * D) = r\ /\ (ra\ /\ b)$
by (*simp add*: *beq-var*)
have *f3*: $c - 0 + a * ((A + B * sqrt\ C)\ /\ D)^2 = -\ (b * (A + B * sqrt\ C)\ /\ D)$
using *root* **by** *force*
have *f4*: $\bigwedge r\ ra\ rb.\ ((-\ (r::real) - ra)\ /\ rb)^2 = ((r + ra)\ /\ rb)^2$
using *f1* **by** (*metis* (*no-types*) *divide-minus-left power2-minus*)
have $\bigwedge r\ ra\ rb\ rc.\ -\ ((r::real) * (ra + rb)\ /\ rc) = r * (-\ ra - rb)\ /\ rc$
using *f1* **by** (*metis divide-divide-eq-right divide-minus-left mult.commute*)
then show *?thesis*
using *f4 f3 f2* **by** (*simp add*: *mult.commute*)
qed
have *y-prop-var*: $-\ ((-\ A - B * sqrt\ C)\ /\ D) < y'$ **using** *y-prop*
by (*metis add.commute diff-minus-eq-add divide-minus-left minus-diff-eq*)
have $\exists\, x\in\{-\ (-\ (A + B * sqrt\ C)\ /\ D)<..y'\}.\ \neg\ a * x^2 + 2 * a * (-$

```
(A + B * sqrt C) / D) * x + c < 0
            using y-prop-var beq-var root-var agteq one-root-a-gt0[where a = a,
where b =b, where c = c, where r= −(A + B * sqrt C) / D]
           by auto
         then show ∃x∈{(A + B * sqrt C) / D<..y'}. ¬ a * x^2 + b * x + c < 0
          proof −
           have f1: 2 * a * (A + B * sqrt C) * inverse D + b = 0
            by (metis True1 divide-inverse)
           obtain rr :: real where
            f2: rr ∈ {− (− (A + B * sqrt C) / D)<..y'} ∧ ¬ a * rr^2 + 2 * a *
(− (A + B * sqrt C) / D) * rr + c < 0
            using ‹∃x∈{− (− (A + B * sqrt C) / D)<..y'}. ¬ a * x^2 + 2 * a *
(− (A + B * sqrt C) / D) * x + c < 0› by blast
           have f3: a * ((A + B * sqrt C) * (inverse D * 2)) = − b
            using f1 by linarith
           have f4: ∀r. − (− (r::real)) = r
            by simp
           have f5: ∀r ra. (ra::real) * − r = r * − ra
            by simp
           have f6: a * ((A + B * sqrt C) * (inverse D * − 2)) = b
            using f3 by simp
           have f7: ∀r ra rb. (rb::real) * (ra * r) = r * (rb * ra)
            by auto
           have f8: ∀r ra. − (ra::real) * r = ra * − r
            by linarith
           then have f9: a * (inverse D * ((A + B * sqrt C) * − 2)) = b
            using f7 f6 f5 by presburger
           have f10: rr ∈ {inverse D * (A + B * sqrt C)<..y'}
                 using f4 f2 by (metis (no-types) divide-inverse mult.commute
mult-minus-right)
           have ¬ c + (rr * b + a * rr^2) < 0
              using f9 f8 f7 f2 by (metis (no-types) add.commute divide-inverse
mult.commute mult-minus-right)
           then show ?thesis
                 using f10 by (metis (no-types) add.commute divide-inverse
mult.commute)
          qed
         qed
        qed
      next
       case False
       then have False1 : a * ((A + B * sqrt C) / D)^2 + b * (A + B * sqrt C)
/ D + c ≠ 0 by auto
        show ?thesis proof(cases a * ((A + B * sqrt C) / D)^2 + b * (A + B *
sqrt C) / D + c < 0)
         case True
         show ?thesis unfolding LessUni fields apply (simp add: True1 True)
          using True1 True
         proof −
```

let *?r* = $(A + B * sqrt\ C)\ /\ D$
assume $2 * a * (A + B * sqrt\ C)\ /\ D + b = 0$
assume $a * ((A + B * sqrt\ C)\ /\ D)^2 + b * (A + B * sqrt\ C)\ /\ D + c < 0$
then have $\exists y'{>}(A + B * sqrt\ C)\ /\ D.\ \forall x{\in}\{(A + B * sqrt\ C)\ /\ D{<}..y'\}.\ poly\ [:c,\ b,\ a:]\ x < 0$ **using** *continuity-lem-lt0*[**where** $r{=}\ (A + B * sqrt\ C)\ /\ D$, **where** $c = c$, **where** $b = b$, **where** $a{=}a$]
quadratic-poly-eval[*of c b a ?r*] **by** *auto*
then show $\exists y'{>}(A + B * sqrt\ C)\ /\ D.\ \forall x{\in}\{(A + B * sqrt\ C)\ /\ D{<}..y'\}.\ a * x^2 + b * x + c < 0$
using *quadratic-poly-eval*[*of c b a*]
by *fastforce*
qed
next
case *False*
then have *False'* : $a * ((A + B * sqrt\ C)\ /\ D)^2 + b * (A + B * sqrt\ C)\ /\ D + c > 0$ **using** *False1* **by** *auto*
show *?thesis* **unfolding** *LessUni fields* **apply**(*simp add*: *True1 False False1*)
using *True1 False' continuity-lem-gt0-expanded*[**where** $a = a$, **where** $b = b$,**where** $c = c$, **where** $r = ((A + B * sqrt\ C)\ /\ D)$]
by (*metis mult-less-0-iff not-square-less-zero times-divide-eq-right*)
qed
qed
next
case *False*
then have *False1* : $2 * a * (A + B * sqrt\ C)\ /\ D + b \neq 0$ **by** *auto*
have *c1*: $a * ((A + B * sqrt\ C)\ /\ D)^2 + b * (A + B * sqrt\ C)\ /\ D + c = 0 \Longrightarrow$
$2 * a * (A + B * sqrt\ C)\ /\ D + b < 0 \Longrightarrow$
$\exists y'{>}(A + B * sqrt\ C)\ /\ D.\ \forall x{\in}\{(A + B * sqrt\ C)\ /\ D{<}..y'\}.\ a * x^2 + b * x + c < 0$
proof −
assume *root*: $a * ((A + B * sqrt\ C)\ /\ D)^2 + b * (A + B * sqrt\ C)\ /\ D + c = 0$
assume *blt*: $2 * a * (A + B * sqrt\ C)\ /\ D + b < 0$
let *?r* = $-(A + B * sqrt\ C)\ /\ D$
have *bltvar*: $b < 2 * a * ?r$ **using** *blt divide-minus-left mult-2 mult-minus-right real-add-less-0-iff*
by (*metis times-divide-eq-right*)
have *rootvar*: $a * ?r\hat{}2 - b * ?r + c = 0$ **using** *root*
proof −
have *f1*: $\forall r\ ra.\ -\ (ra{::}real) * r = ra * -\ r$
by *simp*
have *f2*: $\forall r\ ra.\ ((ra{::}real) * -\ r)^2 = (ra * r)^2$
by *simp*
have *f3*: $a * (inverse\ D * (A - B * -\ sqrt\ C))^2 - inverse\ D * (b * -\ (A - B * -\ sqrt\ C)) - -\ c = 0$
by (*metis* (*no-types*) *diff-minus-eq-add divide-inverse mult.commute*

mult-minus-left root)
have *f4*: $\forall\, r\ ra\ rb.\ (rb::real) * (ra * r) = ra * (r * rb)$
by *simp*
have $\forall\, r\ ra.\ (ra::real) * - r = r * - ra$
by *simp*
then have $a * (inverse\ D * (A - B * - sqrt\ C))^2 - b * (inverse\ D * - (A - B * - sqrt\ C)) - - c = 0$
using *f4 f3 f1* **by** (*metis* (*no-types*))
then have $a * (inverse\ D * - (A - B * - sqrt\ C))^2 - b * (inverse\ D * - (A - B * - sqrt\ C)) - - c = 0$
using *f2* **by** *presburger*
then show *?thesis*
by (*simp add*: *divide-inverse mult.commute*)
qed
have $\exists\, y'> ((A + B * sqrt\ C) / D).\ \forall\, x\in\{((A + B * sqrt\ C) / D)<..y'\}.\ a * x^2 + b * x + c < 0$
using *rootvar bltvar case-d1*[**where** $a= a$, **where** $b = b$, **where** $c = c$, **where** $r = ?r$]
by (*metis add.inverse-inverse divide-inverse mult-minus-left*)
then show *?thesis*
by *blast*
qed
have *c2*: $\bigwedge y'.\ a * ((A + B * sqrt\ C) / D)^2 + b * (A + B * sqrt\ C) / D + c = 0 \Longrightarrow$
$\neg\ 2 * a * (A + B * sqrt\ C) / D + b < 0 \Longrightarrow$
$(A + B * sqrt\ C) / D < y' \Longrightarrow$
$\exists\, x\in\{(A + B * sqrt\ C) / D<..y'\}.\ \neg\ a * x^2 + b * x + c < 0$
proof −
let $?r = (A + B * sqrt\ C) / D$
fix y'
assume *h1*: $a * ((A + B * sqrt\ C) / D)^2 + b * (A + B * sqrt\ C) / D + c = 0$
assume *h2*: $\neg\ 2 * a * (A + B * sqrt\ C) / D + b < 0$
assume *h3*: $(A + B * sqrt\ C) / D < y'$
have *eq*: $2 * a * (A + B * sqrt\ C) / D + b = 0 \Longrightarrow \exists\, x\in\{(A + B * sqrt\ C) / D..y'\}.\ \neg\ a * x^2 + b * x + c < 0$
using *False1* **by** *blast*
have $2 * a * (A + B * sqrt\ C) / D + b > 0 \Longrightarrow \exists\, x\in\{?r<..y'\}.\ \neg\ a * x^2 + b * x + c < 0$
using *case-d4*[**where** $a = a$, **where** $b = b$, **where** $c= c$, **where** $r = -?r$] *h1 h2 h3* **by** *auto*
then show $\exists\, x\in\{(A + B * sqrt\ C) / D<..y'\}.\ \neg\ a * x^2 + b * x + c < 0$ **using** *h2 eq*
using *False1* **by** *linarith*
qed
have *c3*: $((a::real) * ((A + B * sqrt\ C) / D)^2 + b * (A + B * sqrt\ C) / D + c < 0) \longrightarrow$
$(\exists\, y'>((A + B * sqrt\ C) / D).\ \forall\, x\in\{(A + B * sqrt\ C) / D<..y'\}.\ a * x^2 + b * x + c < 0)$

proof *clarsimp*
assume *assump*: $a * ((A + B * sqrt\ C) / D)^2 + b * (A + B * sqrt\ C) / D + c < 0$
have $a * ((A + B * sqrt\ C) / D)^2 + b * ((A + B * sqrt\ C) / D) + c < 0 \Longrightarrow$
$\exists y' > (A + B * sqrt\ C) / D.\ \forall x \in \{(A + B * sqrt\ C) / D <..y'\}.\ a * x^2 + b * x + c < 0$
using *continuity-lem-lt0-expanded*[**where** $a = a$, **where** $b = b$, **where** $c = c$, **where** $r = ((A + B * sqrt\ C) / D)$::*real*]
by *auto*
then have $\exists y' > (A + B * sqrt\ C) / D.\ \forall x \in \{(A + B * sqrt\ C) / D <..y'\}.\ a * x^2 + b * x + c < 0$ **using** *assump* **by** *auto*
then obtain *y* **where** *y-prop*: $y > (A + B * sqrt\ C) / D \wedge (\forall x \in \{(A + B * sqrt\ C) / D <..y\}.\ a * x^2 + b * x + c < 0)$ **by** *auto*
then have *h*: $\exists\ k.\ k > (A + B * sqrt\ C) / D \wedge k < y$ **using** *dense*
by *blast*
then obtain *k* **where** *k-prop*: $k > (A + B * sqrt\ C) / D\ \wedge k < y$ **by** *auto*
then have $\forall x \in \{(A + B * sqrt\ C) / D..k\}.\ a * x^2 + b * x + c < 0$ **using** *y-prop*
using *assump* **by** *force*
then show $\exists y' > ((A + B * sqrt\ C) / D$::*real*$).\ \forall x \in \{(A + B * sqrt\ C) / D <..y'\}.\ a * x^2 + b * x + c < 0$
using *k-prop* **by** *auto*
qed
have *c4*: $\bigwedge y'.\ a * ((A + B * sqrt\ C) / D)^2 + b * (A + B * sqrt\ C) / D + c \neq 0 \Longrightarrow$
$\neg\ a * ((A + B * sqrt\ C) / D)^2 + b * (A + B * sqrt\ C) / D + c < 0 \Longrightarrow$
$(A + B * sqrt\ C) / D < y' \Longrightarrow$
$\exists x \in \{(A + B * sqrt\ C) / D <..y'\}.\ \neg\ a * x^2 + b * x + c < 0$
using *continuity-lem-gt0-expanded*[**where** $a = a$, **where** $b = b$, **where** $c = c$, **where** $r = (A + B * sqrt\ C) / D$]
by (*metis Groups.mult-ac*(*1*) *divide-inverse less-eq-real-def linorder-not-le*)
show *?thesis* **unfolding** *LessUni fields* **apply**(*simp add*: *False1*)
using *c1 c2 c3 c4* **by** *auto*
qed
qed
next
case *False*
then have *EqUni*: $At = EqUni\ p$ **using** *at-is* **by** *auto*
then show *?thesis* **proof**(*cases p*)
case (*fields a b c*)
have $\bigwedge y'.\ (A + B * sqrt\ C) / D < y' \Longrightarrow$
$\forall x \in \{(A + B * sqrt\ C) / D <..y'\}.\ a * x^2 + b * x + c = 0 \Longrightarrow (a = 0 \wedge b = 0 \wedge c = 0)$
proof −
fix y'
assume $(A + B * sqrt\ C) / D < y'$
then show $\forall x \in \{(A + B * sqrt\ C) / D <..y'\}.\ a * x^2 + b * x + c = 0 \Longrightarrow (a = 0 \wedge b = 0 \wedge c = 0)$ **using** *assms continuity-lem-eq0*[**where** $r = (A + B *$

sqrt C) / *D*, **where** *p* = *y'*, **where** *a*= *a*, **where** *b* =*b*, **where** *c*=*c*]
by *auto*
qed
then show *?thesis*
apply (*auto simp add*:*EqUni fields*)
using *linordered-field-no-ub* **by** *blast*
qed
qed

lemma *infinitesimal-quad*:
fixes *A B C D*:: *real*
assumes *D*≠*0*
assumes *C*≥*0*
shows (∃ *y'*::*real*>((*A*+*B* ∗ *sqrt*(*C*))/(*D*)). ∀ *x*::*real* ∈{((*A*+*B* ∗ *sqrt*(*C*))/(*D*))<..*y'*}. *aEvalUni At x*)
= (*evalUni* (*substInfinitesimalQuadraticUni A B C D At*) *x*)
proof(*cases At*)
case (*LessUni p*)
then show *?thesis* **using** *infinitesimal-quad-helper assms*
by *blast*
next
case (*EqUni p*)
then show *?thesis*
using *infinitesimal-quad-helper assms*
by *blast*
next
case (*LeqUni p*)
then show *?thesis*
proof (*cases p*)
case (*fields a b c*)
have *same*: ∀ *x*. *aEvalUni* (*LeqUni p*) *x* = (*aEvalUni* (*EqUni p*) *x*) ∨ (*aEvalUni* (*LessUni p*) *x*)
apply (*simp add*: *fields*)
by *force*
let *?r* = (*A* + *B* ∗ *sqrt C*) / *D*
have ⋀*a b c*.
At = *LeqUni p* ⟹
p = (*a*, *b*, *c*) ⟹
(∃ *y'*>(*A* + *B* ∗ *sqrt C*) / *D*. ∀ *x*∈{(*A* + *B* ∗ *sqrt C*) / *D*<..*y'*}. *aEvalUni At x*) =
evalUni (*substInfinitesimalQuadraticUni A B C D At*) *x*
proof −
fix *a b c*
assume *atis*: *At* = *LeqUni p*
assume *p-is*: *p* = (*a*, *b*, *c*)
have *s1*: (∃ *y'*>*?r*. ∀ *x*∈{ *?r*<..*y'*}. *aEvalUni At x*) = (∃ *y'*>*?r*. ∀ *x*∈{ *?r*<..*y'*}. (*aEvalUni* (*EqUni p*) *x*) ∨ (*aEvalUni* (*LessUni p*) *x*))
using *atis same aEvalUni.simps*(*2*) *aEvalUni.simps*(*3*) *fields less-eq-real-def*

by *blast*
have *s2*: ... = $(\exists\, y'{>}?r.\ \forall\, x{\in}\{\, ?r{<}..y'\}.\ (aEvalUni\ (EqUni\ p)\ x)) \vee (\exists\, y'{>}?r.\ \forall\, x{\in}\{\, ?r{<}..y'\}.\ (aEvalUni\ (LessUni\ p)\ x))$
using *either-or*[**where** $r = ?r$] *p-is*
by *blast*
have *eq1*: $(\exists\, y'{>}?r.\ \forall\, x{\in}\{\, ?r{<}..y'\}.\ (aEvalUni\ (EqUni\ p)\ x)) = (evalUni\ (substInfinitesimalQuadraticUni\ A\ B\ C\ D\ (EqUni\ p))\ x)$
using *infinitesimal-quad-helper*[**where** $At = EqUni\ p$, **where** $p = p$, **where** $B = B$, **where** $C = C$, **where** $A = A$, **where** $D = D$]
assms **by** *auto*
have *eq2*: $(\exists\, y'{>}?r.\ \forall\, x{\in}\{\, ?r{<}..y'\}.\ (aEvalUni\ (LessUni\ p)\ x)) = (evalUni\ (substInfinitesimalQuadraticUni\ A\ B\ C\ D\ (LessUni\ p))\ x)$
using *infinitesimal-quad-helper*[**where** $At = LessUni\ p$, **where** $p = p$, **where** $B = B$, **where** $C = C$, **where** $A = A$, **where** $D = D$]
assms **by** *auto*
have *z1*: $(\exists\, y'{>}?r.\ \forall\, x{\in}\{\, ?r{<}..y'\}.\ aEvalUni\ At\ x) = ((evalUni\ (substInfinitesimalQuadraticUni\ A\ B\ C\ D\ (EqUni\ p))\ x) \vee (evalUni\ (substInfinitesimalQuadraticUni\ A\ B\ C\ D\ (LessUni\ p))\ x))$
using *s1 s2 eq1 eq2* **by** *auto*
have *z2*: $(evalUni\ (substInfinitesimalQuadraticUni\ A\ B\ C\ D\ (EqUni\ p))\ x) \vee (evalUni\ (substInfinitesimalQuadraticUni\ A\ B\ C\ D\ (LessUni\ p))\ x) = evalUni\ (substInfinitesimalQuadraticUni\ A\ B\ C\ D\ (LeqUni\ p))\ x$
by *auto*
have *z3*: $(evalUni\ (substInfinitesimalQuadraticUni\ A\ B\ C\ D\ At)\ x) = evalUni\ (substInfinitesimalQuadraticUni\ A\ B\ C\ D\ (LeqUni\ p))\ x$
using *LeqUni* **by** *auto*
then have *z4*: $(evalUni\ (substInfinitesimalQuadraticUni\ A\ B\ C\ D\ (EqUni\ p))\ x) \vee (evalUni\ (substInfinitesimalQuadraticUni\ A\ B\ C\ D\ (LessUni\ p))\ x) = (evalUni\ (substInfinitesimalQuadraticUni\ A\ B\ C\ D\ At)\ x)$
using *z2 z3* **by** *auto*
let *?a* = $(evalUni\ (substInfinitesimalQuadraticUni\ A\ B\ C\ D\ (EqUni\ p))\ x) \vee (evalUni\ (substInfinitesimalQuadraticUni\ A\ B\ C\ D\ (LessUni\ p))\ x)$
let *?b* = $(\exists\, y'{>}?r.\ \forall\, x{\in}\{\, ?r{<}..y'\}.\ aEvalUni\ At\ x)$
let *?c* = $(evalUni\ (substInfinitesimalQuadraticUni\ A\ B\ C\ D\ At)\ x)$
have *t1*: *?b* = *?a* **using** *z1* **by** *auto*
have *t2*: *?a* = *?c* **using** *z4*
using *atis* **by** *auto*
then have *?b* = *?c* **using** *t1 t2* **by** *auto*
then show $(\exists\, y'{>}?r.\ \forall\, x{\in}\{\, ?r{<}..y'\}.\ aEvalUni\ At\ x) = evalUni\ (substInfinitesimalQuadraticUni\ A\ B\ C\ D\ At)\ x$
by *auto*
qed
then show *?thesis*
using *LeqUni fields* **by** *blast*
qed
next
case (*NeqUni p*)
then show *?thesis*
proof (*cases p*)

```
    case (fields a b c)
    then show ?thesis unfolding NeqUni fields using nonzcoeffs by auto
  qed
qed


end
```

9.4 Overall General VS Proofs

```
theory DNFUni
  imports QE InfinitesimalsUni
begin

fun DNFUni :: atomUni fmUni ⇒ atomUni list list where
  DNFUni (AtomUni a) = [[a]]|
  DNFUni (TrueFUni) = [[]] |
  DNFUni (FalseFUni) = []|
  DNFUni (AndUni A B) = [A' @ B'. A' ← DNFUni A, B' ← DNFUni B]|
  DNFUni (OrUni A B) = DNFUni A @ DNFUni B

lemma eval-DNFUni : evalUni F x = evalUni (list-disj-Uni(map (list-conj-Uni o
(map AtomUni)) (DNFUni F))) x
proof(induction F)
  case TrueFUni
  then show ?case by auto
next
  case FalseFUni
  then show ?case by auto
next
  case (AtomUni x)
  then show ?case by auto
next
  case (AndUni F1 F2)
  show ?case unfolding DNFUni.simps eval-list-disj-Uni evalUni.simps AndUni
List.map-concat List.set-concat apply simp
    unfolding eval-list-conj-Uni-append
    by blast
next
  case (OrUni F1 F2)
  then show ?case  unfolding DNFUni.simps List.map-append eval-list-disj-Uni
List.set-append evalUni.simps
    by blast
qed

fun elimVarUni-atom :: atomUni list ⇒ atomUni ⇒ 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 (λA. 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))
  using assms proof(induction L arbitrary :a b c d)
  case Nil
  then show ?case by auto
next
  case (Cons At L)
  then have Cons1 : ⋀a b c d. separateAtoms L = (a, b, c, d) ⟹
    (∀ l∈set L. aEvalUni l x) =
    ((∀ a∈set a. case a of (a, ba, c) ⇒ a * x^2 + ba * x + c = 0) ∧
     (∀ a∈set b. case a of (a, ba, c) ⇒ a * x^2 + ba * x + c < 0)∧
     (∀ a∈set c. case a of (a, ba, c) ⇒ a * x^2 + ba * x + c ≤ 0) ∧
     (∀ a∈set d. case a of (a, ba, c) ⇒ a * x^2 + ba * x + c ≠ 0))
```

separateAtoms (*At* # *L*) = (*a*, *b*,*c*,*d*) **by** *auto*
then show *?case* **proof**(*cases At*)
case (*LessUni p*)
show *?thesis* **proof**(*cases b*)
case *Nil*
show *?thesis* **using** *Cons*(*2*) **unfolding** *LessUni separateAtoms.simps Nil*
apply(*cases separateAtoms L*) **by** *simp*
next
case (*Cons p' b'*)
then have *p-def* : *p'* = *p* **using** *Cons1*(*2*) **unfolding** *LessUni separateAtoms.simps*
apply(*cases separateAtoms L*) **by** *simp*
have *h1* : *separateAtoms L* = (*a*,*b'*,*c*,*d*) **using** *Cons Cons1*(*2*) **unfolding** *LessUni separateAtoms.simps*
apply(*cases separateAtoms L*) **by** *simp*
have *h2* : (∀ *a*∈*set* (*p* # *b'*). *case a of* (*a*, *ba*, *c*) ⇒ *a* ∗ x^2 + *ba* ∗ *x* + *c* < *0*) = (
(∀ *a*∈*set* (*b'*). *case a of* (*a*, *ba*, *c*) ⇒ *a* ∗ x^2 + *ba* ∗ *x* + *c* < *0*)∧ (*case p of* (*a*, *ba*, *c*) ⇒ *a* ∗ x^2 + *ba* ∗ *x* + *c* < *0*))
by *auto*
have *h3* : (∀ *l*∈*set* (*LessUni p* # *L*). *aEvalUni l x*) = ((∀ *l*∈*set* (*L*). *aEvalUni l x*)∧(*case p of* (*a*, *ba*, *c*) ⇒ *a* ∗ x^2 + *ba* ∗ *x* + *c* < *0*))
by *auto*
show *?thesis* **unfolding** *Cons LessUni p-def h2 h3* **using** *Cons1*(*1*)[*OF h1*]
by *auto*
qed
next
case (*EqUni p*)
show *?thesis* **proof**(*cases a*)
case *Nil*
show *?thesis* **using** *Cons*(*2*) **unfolding** *EqUni separateAtoms.simps Nil*
apply(*cases separateAtoms L*) **by** *simp*
next
case (*Cons p' a'*)
then have *p-def* : *p'* = *p* **using** *Cons1*(*2*) **unfolding** *EqUni separateAtoms.simps*
apply(*cases separateAtoms L*) **by** *simp*
have *h1* : *separateAtoms L* = (*a'*,*b*,*c*,*d*) **using** *Cons Cons1*(*2*) **unfolding** *EqUni separateAtoms.simps*
apply(*cases separateAtoms L*) **by** *simp*
have *h2* : (∀ *a*∈*set* (*p* # *a'*). *case a of* (*a*, *ba*, *c*) ⇒ *a* ∗ x^2 + *ba* ∗ *x* + *c* = *0*) = (
(∀ *a*∈*set* (*a'*). *case a of* (*a*, *ba*, *c*) ⇒ *a* ∗ x^2 + *ba* ∗ *x* + *c* = *0*)∧ (*case p of* (*a*, *ba*, *c*) ⇒ *a* ∗ x^2 + *ba* ∗ *x* + *c* = *0*))
by *auto*
have *h3* : (∀ *l*∈*set* (*EqUni p* # *L*). *aEvalUni l x*) = ((∀ *l*∈*set* (*L*). *aEvalUni l x*)∧(*case p of* (*a*, *ba*, *c*) ⇒ *a* ∗ x^2 + *ba* ∗ *x* + *c* = *0*))
by *auto*
show *?thesis* **unfolding** *Cons EqUni p-def h2 h3* **using** *Cons1*(*1*)[*OF h1*]
by *auto*
qed

next
case (*LeqUni p*)
then show *?thesis* **proof**(*cases c*)
case *Nil*
show *?thesis* **using** *Cons*(*2*) **unfolding** *LeqUni separateAtoms.simps Nil*
apply(*cases separateAtoms L*) **by** *simp*
next
case (*Cons p' a'*)
then have *p-def* : $p' = p$ **using** *Cons1*(*2*) **unfolding** *LeqUni separateAtoms.simps*
apply(*cases separateAtoms L*) **by** *simp*
have *h1* : *separateAtoms L* = (*a*,*b*,*a'*,*d*) **using** *Cons Cons1*(*2*) **unfolding** *LeqUni separateAtoms.simps*
apply(*cases separateAtoms L*) **by** *simp*
have *h2* : (∀ *a*∈*set* (*p* # *a'*). *case a of* (*a*, *ba*, *c*) ⇒ $a * x^2 + ba * x + c \leq 0$) = (
(∀ *a*∈*set* (*a'*). *case a of* (*a*, *ba*, *c*) ⇒ $a * x^2 + ba * x + c \leq 0$)∧ (*case p of* (*a*, *ba*, *c*) ⇒ $a * x^2 + ba * x + c \leq 0$))
by *auto*
have *h3* : (∀ *l*∈*set* (*LeqUni p* # *L*). *aEvalUni l x*) = ((∀ *l*∈*set* (*L*). *aEvalUni l x*)∧(*case p of* (*a*, *ba*, *c*) ⇒ $a * x^2 + ba * x + c \leq 0$))
by *auto*
show *?thesis* **unfolding** *Cons LeqUni p-def h2 h3* **using** *Cons1*(*1*)[*OF h1*]
by *auto*
qed
next
case (*NeqUni p*)
then show *?thesis* **proof**(*cases d*)
case *Nil*
show *?thesis* **using** *Cons*(*2*) **unfolding** *NeqUni separateAtoms.simps Nil*
apply(*cases separateAtoms L*) **by** *simp*
next
case (*Cons p' a'*)
then have *p-def* : $p' = p$ **using** *Cons1*(*2*) **unfolding** *NeqUni separateAtoms.simps*
apply(*cases separateAtoms L*) **by** *simp*
have *h1* : *separateAtoms L* = (*a*,*b*,*c*,*a'*) **using** *Cons Cons1*(*2*) **unfolding** *NeqUni separateAtoms.simps*
apply(*cases separateAtoms L*) **by** *simp*
have *h2* : (∀ *a*∈*set* (*p* # *a'*). *case a of* (*a*, *ba*, *c*) ⇒ $a * x^2 + ba * x + c \neq 0$) = (
(∀ *a*∈*set* (*a'*). *case a of* (*a*, *ba*, *c*) ⇒ $a * x^2 + ba * x + c \neq 0$)∧ (*case p of* (*a*, *ba*, *c*) ⇒ $a * x^2 + ba * x + c \neq 0$))
by *auto*
have *h3* : (∀ *l*∈*set* (*NeqUni p* # *L*). *aEvalUni l x*) = ((∀ *l*∈*set* (*L*). *aEvalUni l x*)∧(*case p of* (*a*, *ba*, *c*) ⇒ $a * x^2 + ba * x + c \neq 0$))
by *auto*
show *?thesis* **unfolding** *Cons NeqUni p-def h2 h3* **using** *Cons1*(*1*)[*OF h1*]
by *auto*
qed
qed

qed

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))∧
(∀ (*a*,*b*,*c*)∈*set d*.(∃ *x*. ∀ *y*<*x*. *a*∗*y*^2+*b*∗*y*+*c*≠0)))
using *assms* **proof**(*induction L arbitrary* :*a b c d*)
case *Nil*
then show *?case* **by** *auto*
next
case (*Cons At L*)
then have *Cons1* : ⋀*a b c d*. *separateAtoms L* = (*a*, *b*, *c*, *d*) ⟹
(∀ *l*∈*set L*. *evalUni* (*substNegInfinityUni l*) *x*) =
((∀ *a*∈*set a*. *case a of* (*a*, *ba*, *c*) ⇒ ∃ *x*. ∀ *y*<*x*. *a* ∗ *y*^2 + *ba* ∗ *y* + *c* = 0) ∧
(∀ *a*∈*set b*. *case a of* (*a*, *ba*, *c*) ⇒ ∃ *x*. ∀ *y*<*x*. *a* ∗ *y*^2 + *ba* ∗ *y* + *c* < 0)∧
(∀ *a*∈*set c*. *case a of* (*a*, *ba*, *c*) ⇒ ∃ *x*. ∀ *y*<*x*. *a* ∗ *y*^2 + *ba* ∗ *y* + *c* ≤ 0)∧
(∀ *a*∈*set d*. *case a of* (*a*, *ba*, *c*) ⇒ ∃ *x*. ∀ *y*<*x*. *a* ∗ *y*^2 + *ba* ∗ *y* + *c* ≠ 0))
separateAtoms (*At* # *L*) = (*a*, *b*, *c*, *d*) **by** *auto*
then show *?case* **proof**(*cases At*)
case (*LessUni p*)
show *?thesis* **using** *LessUni Cons* **proof**(*induction b rule* : *list.induct*)
case *Nil*
then have *Nil* : *b* = []
using *Cons.prems* **by** *auto*
show *?case* **using** *Cons*(*2*) **unfolding** *LessUni separateAtoms.simps Nil*
apply(*cases separateAtoms L*) **by** *simp*
next
case (*Cons p' b'*)
then have *p-def* : *p'* = *p* **using** *Cons1*(*2*) **unfolding** *LessUni separateAtoms.simps*
apply(*cases separateAtoms L*) **by** *simp*
have *h1* : *separateAtoms L* = (*a*,*b'*,*c*,*d*) **using** *Cons Cons1*(*2*) **unfolding** *LessUni separateAtoms.simps*
apply(*cases separateAtoms L*) **by** *simp*
have *h2* : (∀ *a*∈*set* (*p* # *b'*). *case a of* (*a*, *ba*, *c*) ⇒ ∃ *x*. ∀ *y*<*x*. *a* ∗ *y*^2 + *ba* ∗ *y* + *c* < 0) = (
(∀ *a*∈*set* (*b'*). *case a of* (*a*, *ba*, *c*) ⇒ ∃ *x*. ∀ *y*<*x*. *a* ∗ *y*^2 + *ba* ∗ *y* + *c* < 0)∧ (*case p of* (*a*, *ba*, *c*) ⇒ ∃ *x*. ∀ *y*<*x*. *a* ∗ *y*^2 + *ba* ∗ *y* + *c* < 0))
by *auto*
have *one*: (∃ *x*. ∀ *y*<*x*. *aEvalUni* (*LessUni p*) *y*) = (*case p of* (*a*, *ba*, *c*) ⇒ ∃ *x*. ∀ *y*<*x*. *a* ∗ *y*^2 + *ba* ∗ *y* + *c* < 0)
apply(*cases p*) **by** *simp*
have (∀ *l*∈*set* (*LessUni p* # *L*). *evalUni* (*substNegInfinityUni l*) *x*) = ((*evalUni* (*substNegInfinityUni* (*LessUni p*)) *x*)∧(∀ *l*∈*set* (*L*). *evalUni* (*substNegInfinityUni l*) *x*))
by *auto*
also have ... = (

```
    (case p of (a,ba,c) ⇒ ∃ x. ∀ y<x. a * y^2 + ba * y + c < 0)∧
    (∀ a∈set a. case a of (a, ba, c) ⇒ ∃ x. ∀ y<x. a * y^2 + ba * y + c = 0) ∧
   (∀ a∈set b'. case a of (a, ba, c) ⇒ ∃ x. ∀ y<x. a * y^2 + ba * y + c < 0)∧
   (∀ a∈set c. case a of (a, ba, c) ⇒ ∃ x. ∀ y<x. a * y^2 + ba * y + c ≤ 0)∧
   (∀ a∈set d. case a of (a, ba, c) ⇒ ∃ x. ∀ y<x. a * y^2 + ba * y + c ≠ 0))
       unfolding infinity-evalUni[of LessUni p x, symmetric] Cons(3)[OF h1]
LessUni one by simp
    finally have h3 : (∀ l∈set (LessUni p # L). evalUni (substNegInfinityUni l)
x) = (
    (case p of (a,ba,c) ⇒ ∃ x. ∀ y<x. a * y^2 + ba * y + c < 0)∧
    (∀ a∈set a. case a of (a, ba, c) ⇒ ∃ x. ∀ y<x. a * y^2 + ba * y + c = 0) ∧
   (∀ a∈set b'. case a of (a, ba, c) ⇒ ∃ x. ∀ y<x. a * y^2 + ba * y + c < 0)∧
   (∀ a∈set c. case a of (a, ba, c) ⇒ ∃ x. ∀ y<x. a * y^2 + ba * y + c ≤ 0)∧
   (∀ a∈set d. case a of (a, ba, c) ⇒ ∃ x. ∀ y<x. a * y^2 + ba * y + c ≠ 0) )
     by auto
    show ?case unfolding Cons LessUni p-def h2 h3 using Cons1(1)[OF h1]
     by auto
  qed
 next
  case (EqUni p)
  show ?thesis using EqUni Cons proof(induction a rule : list.induct)
   case Nil
   then have Nil : a = []
    using Cons.prems by auto
   show ?case using Cons(2) unfolding EqUni separateAtoms.simps Nil
    apply(cases separateAtoms L) by simp
  next
   case (Cons p' a')
  then have p-def : p' = p using Cons1(2) unfolding EqUni separateAtoms.simps
    apply(cases separateAtoms L) by simp
    have h1 : separateAtoms L = (a',b,c,d) using Cons Cons1(2) unfolding
EqUni separateAtoms.simps
    apply(cases separateAtoms L) by simp
    have h2 : (∀ a∈set (p # a'). case a of (a, ba, c) ⇒ ∃ x. ∀ y<x. a * y^2 + ba
* y + c = 0) = (
       (∀ a∈set ( a'). case a of (a, ba, c) ⇒ ∃ x. ∀ y<x. a * y^2 + ba * y + c =
0)∧ (case p of (a, ba, c) ⇒ ∃ x. ∀ y<x. a * y^2 + ba * y + c = 0))
    by auto
    have one: (∃ x. ∀ y<x. aEvalUni (EqUni p) y) = (case p of (a, ba, c) ⇒ ∃ x.
∀ y<x. a * y^2 + ba * y + c = 0)
    apply(cases p) by simp
    have (∀ l∈set (EqUni p # L). evalUni (substNegInfinityUni l) x) = ((evalUni
(substNegInfinityUni (EqUni p)) x)∧(∀ l∈set ( L). evalUni (substNegInfinityUni l)
x))
    by auto
    also have ... = (
    (case p of (a,ba,c) ⇒ ∃ x. ∀ y<x. a * y^2 + ba * y + c = 0)∧
    (∀ a∈set a'. case a of (a, ba, c) ⇒ ∃ x. ∀ y<x. a * y^2 + ba * y + c = 0) ∧
   (∀ a∈set b. case a of (a, ba, c) ⇒ ∃ x. ∀ y<x. a * y^2 + ba * y + c < 0)∧
```

```
    (∀ a∈set c. case a of (a, ba, c) ⇒ ∃ x. ∀ y<x. a * y^2 + ba * y + c ≤ 0)∧
    (∀ a∈set d. case a of (a, ba, c) ⇒ ∃ x. ∀ y<x. a * y^2 + ba * y + c ≠ 0))
     unfolding infinity-evalUni[of EqUni p x, symmetric] Cons(3)[OF h1] EqUni
one by simp
    finally have h3 : (∀ l∈set (EqUni p # L). evalUni (substNegInfinityUni l) x)
= (
    (case p of (a,ba,c) ⇒ ∃ x. ∀ y<x. a * y^2 + ba * y + c = 0)∧
    (∀ a∈set a'. case a of (a, ba, c) ⇒ ∃ x. ∀ y<x. a * y^2 + ba * y + c = 0) ∧
    (∀ a∈set b. case a of (a, ba, c) ⇒ ∃ x. ∀ y<x. a * y^2 + ba * y + c < 0)∧
    (∀ a∈set c. case a of (a, ba, c) ⇒ ∃ x. ∀ y<x. a * y^2 + ba * y + c ≤ 0)∧
    (∀ a∈set d. case a of (a, ba, c) ⇒ ∃ x. ∀ y<x. a * y^2 + ba * y + c ≠ 0))
     by auto
    show ?case unfolding Cons EqUni p-def h2 h3 using Cons1(1)[OF h1]
     by auto
   qed
  next
   case (LeqUni p)
   show ?thesis using LeqUni Cons proof(induction c rule : list.induct)
    case Nil
    then have Nil : c = []
     using Cons.prems by auto
    show ?case using Cons(2) unfolding LeqUni separateAtoms.simps Nil
     apply(cases separateAtoms L) by simp
   next
    case (Cons p' c')
   then have p-def : p' = p using Cons1(2) unfolding LeqUni separateAtoms.simps
     apply(cases separateAtoms L) by simp
    have h1 :  separateAtoms L = (a,b,c',d) using Cons Cons1(2) unfolding
LeqUni separateAtoms.simps
     apply(cases separateAtoms L) by simp
    have h2 : (∀ a∈set (p # c'). case a of (a, ba, c) ⇒ ∃ x. ∀ y<x. a * y^2 + ba
* y + c ≤ 0) = (
       (∀ a∈set ( c'). case a of (a, ba, c) ⇒ ∃ x. ∀ y<x. a * y^2 + ba * y + c ≤
0)∧ (case p of (a, ba, c) ⇒ ∃ x. ∀ y<x. a * y^2 + ba * y + c ≤ 0))
     by auto
    have one: (∃ x. ∀ y<x. aEvalUni (LeqUni p) y) = (case p of (a, ba, c) ⇒ ∃ x.
∀ y<x. a * y^2 + ba * y + c ≤ 0)
     apply(cases p) by simp
   have (∀ l∈set (LeqUni p # L). evalUni (substNegInfinityUni l) x) = ((evalUni
(substNegInfinityUni (LeqUni p)) x)∧(∀ l∈set ( L). evalUni (substNegInfinityUni l)
x))
     by auto
    also have ... = (
    (case p of (a,ba,c) ⇒ ∃ x. ∀ y<x. a * y^2 + ba * y + c ≤ 0)∧
    (∀ a∈set a. case a of (a, ba, c) ⇒ ∃ x. ∀ y<x. a * y^2 + ba * y + c = 0) ∧
    (∀ a∈set b. case a of (a, ba, c) ⇒ ∃ x. ∀ y<x. a * y^2 + ba * y + c < 0)∧
    (∀ a∈set c'. case a of (a, ba, c) ⇒ ∃ x. ∀ y<x. a * y^2 + ba * y + c ≤ 0)∧
    (∀ a∈set d. case a of (a, ba, c) ⇒ ∃ x. ∀ y<x. a * y^2 + ba * y + c ≠ 0))
       unfolding infinity-evalUni[of LeqUni p x, symmetric] Cons(3)[OF h1]
```

LeqUni one
by *simp*
finally have *h3* : (∀ *l*∈*set* (*LeqUni p # L*). *evalUni* (*substNegInfinityUni l*) *x*) = (
(*case p of* (*a*,*ba*,*c*) ⇒ ∃ *x*. ∀ *y*<*x*. *a* ∗ *y*^{2} + *ba* ∗ *y* + *c* ≤ *0*)∧
(∀ *a*∈*set a*. *case a of* (*a*, *ba*, *c*) ⇒ ∃ *x*. ∀ *y*<*x*. *a* ∗ *y*^{2} + *ba* ∗ *y* + *c* = *0*) ∧
(∀ *a*∈*set b*. *case a of* (*a*, *ba*, *c*) ⇒ ∃ *x*. ∀ *y*<*x*. *a* ∗ *y*^{2} + *ba* ∗ *y* + *c* < *0*)∧
(∀ *a*∈*set c*′. *case a of* (*a*, *ba*, *c*) ⇒ ∃ *x*. ∀ *y*<*x*. *a* ∗ *y*^{2} + *ba* ∗ *y* + *c* ≤ *0*)∧
(∀ *a*∈*set d*. *case a of* (*a*, *ba*, *c*) ⇒ ∃ *x*. ∀ *y*<*x*. *a* ∗ *y*^{2} + *ba* ∗ *y* + *c* ≠ *0*))
by *auto*
show *?case* **unfolding** *Cons LeqUni p-def h2 h3* **using** *Cons1*(*1*)[*OF h1*]
by *auto*
qed
next
case (*NeqUni p*)
show *?thesis* **using** *NeqUni Cons* **proof**(*induction d rule* : *list.induct*)
case *Nil*
then have *Nil* : *d* = []
using *Cons.prems* **by** *auto*
show *?case* **using** *Cons*(*2*) **unfolding** *NeqUni separateAtoms.simps Nil*
apply(*cases separateAtoms L*) **by** *simp*
next
case (*Cons p*′ *d*′)
then have *p-def* : *p*′ = *p* **using** *Cons1*(*2*) **unfolding** *NeqUni separateAtoms.simps*
apply(*cases separateAtoms L*) **by** *simp*
have *h1* : *separateAtoms L* = (*a*,*b*,*c*,*d*′) **using** *Cons Cons1*(*2*) **unfolding** *NeqUni separateAtoms.simps*
apply(*cases separateAtoms L*) **by** *simp*
have *h2* : (∀ *a*∈*set* (*p # d*′). *case a of* (*a*, *ba*, *c*) ⇒ ∃ *x*. ∀ *y*<*x*. *a* ∗ *y*^{2} + *ba* ∗ *y* + *c* ≠ *0*) = (
(∀ *a*∈*set* (*d*′). *case a of* (*a*, *ba*, *c*) ⇒ ∃ *x*. ∀ *y*<*x*. *a* ∗ *y*^{2} + *ba* ∗ *y* + *c* ≠ *0*)∧ (*case p of* (*a*, *ba*, *c*) ⇒ ∃ *x*. ∀ *y*<*x*. *a* ∗ *y*^{2} + *ba* ∗ *y* + *c* ≠ *0*))
by *auto*
have *one*: (∃ *x*. ∀ *y*<*x*. *aEvalUni* (*NeqUni p*) *y*) = (*case p of* (*a*, *ba*, *c*) ⇒ ∃ *x*. ∀ *y*<*x*. *a* ∗ *y*^{2} + *ba* ∗ *y* + *c* ≠ *0*)
apply(*cases p*) **by** *simp*
have (∀ *l*∈*set* (*NeqUni p # L*). *evalUni* (*substNegInfinityUni l*) *x*) = ((*evalUni* (*substNegInfinityUni* (*NeqUni p*)) *x*)∧(∀ *l*∈*set* (*L*). *evalUni* (*substNegInfinityUni l*) *x*))
by *auto*
also have ... = (
(*case p of* (*a*,*ba*,*c*) ⇒ ∃ *x*. ∀ *y*<*x*. *a* ∗ *y*^{2} + *ba* ∗ *y* + *c* ≠ *0*)∧
(∀ *a*∈*set a*. *case a of* (*a*, *ba*, *c*) ⇒ ∃ *x*. ∀ *y*<*x*. *a* ∗ *y*^{2} + *ba* ∗ *y* + *c* = *0*) ∧
(∀ *a*∈*set b*. *case a of* (*a*, *ba*, *c*) ⇒ ∃ *x*. ∀ *y*<*x*. *a* ∗ *y*^{2} + *ba* ∗ *y* + *c* < *0*)∧
(∀ *a*∈*set c*. *case a of* (*a*, *ba*, *c*) ⇒ ∃ *x*. ∀ *y*<*x*. *a* ∗ *y*^{2} + *ba* ∗ *y* + *c* ≤ *0*)∧
(∀ *a*∈*set d*′. *case a of* (*a*, *ba*, *c*) ⇒ ∃ *x*. ∀ *y*<*x*. *a* ∗ *y*^{2} + *ba* ∗ *y* + *c* ≠ *0*))
unfolding *infinity-evalUni*[*of NeqUni p x*, *symmetric*] *Cons*(*3*)[*OF h1*] *NeqUni one*
by *simp*

```
    finally have h3 : (∀ l∈set (NeqUni p # L). evalUni (substNegInfinityUni l)
x) = (
    (case p of (a,ba,c) ⇒ ∃ x. ∀ y<x. a * y^2 + ba * y + c ≠ 0)∧
    (∀ a∈set a. case a of (a, ba, c) ⇒ ∃ x. ∀ y<x. a * y^2 + ba * y + c = 0) ∧
    (∀ a∈set b. case a of (a, ba, c) ⇒ ∃ x. ∀ y<x. a * y^2 + ba * y + c < 0)∧
    (∀ a∈set c. case a of (a, ba, c) ⇒ ∃ x. ∀ y<x. a * y^2 + ba * y + c ≤ 0)∧
    (∀ a∈set d'. case a of (a, ba, c) ⇒ ∃ x. ∀ y<x. a * y^2 + ba * y + c ≠ 0) )
      by auto
    show ?case unfolding Cons NeqUni p-def h2 h3 using Cons1(1)[OF h1]
      by auto
   qed
  qed
qed

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)
  using assms proof(induction L arbitrary :eq les leq neq)
  case Nil
  then show ?case by auto
next
  case (Cons At L)
  then show ?case proof(cases At)
    case (LessUni p)
   have ∃ les'. p#les' = les ∧ separateAtoms L = (eq, les', leq, neq) using Cons(2)
unfolding LessUni apply (cases separateAtoms L) by auto
    then obtain les' where les' : p#les' = les separateAtoms L = (eq, les', leq,
neq) by auto
   show ?thesis unfolding LessUni les'(1)[symmetric] using Cons(1)[OF les'(2)]
by auto
  next
    case (EqUni p)
   have ∃ eq'. p#eq' = eq ∧ separateAtoms L = (eq', les, leq, neq) using Cons(2)
unfolding EqUni apply (cases separateAtoms L) by auto
   then obtain eq' where eq' : p#eq' = eq separateAtoms L = (eq', les, leq, neq)
by auto
    show ?thesis unfolding EqUni eq'(1)[symmetric] using Cons(1)[OF eq'(2)]
by auto
  next
    case (LeqUni p)
   have ∃ leq'. p#leq' = leq ∧ separateAtoms L = (eq, les, leq', neq) using Cons(2)
unfolding LeqUni apply (cases separateAtoms L)
     by auto
    then obtain leq' where leq' : p#leq' = leq separateAtoms L = (eq, les, leq',
neq) by auto
   show ?thesis unfolding LeqUni leq'(1)[symmetric] using Cons(1)[OF leq'(2)]
by auto
  next
```

case (*NeqUni p*)
have ∃ *neq'*. *p#neq'* = *neq* ∧ *separateAtoms L* = (*eq*, *les*, *leq*, *neq'*) **using** *Cons*(*2*) **unfolding** *NeqUni* **apply** (*cases separateAtoms L*)
by *auto*
then obtain *neq'* **where** *neq'* : *p#neq'* = *neq separateAtoms L* = (*eq*, *les*, *leq*, *neq'*) **by** *auto*
show *?thesis* **unfolding** *NeqUni neq'*(*1*)[*symmetric*] **using** *Cons*(*1*)[*OF neq'*(*2*)] **by** *auto*
qed
qed

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*)
unfolding *image-set*[*symmetric*] *set-split*[*OF assms*]
unfolding *image-set map-append map-map* **by** *auto*

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−
have *c1*: (∃ *l*∈*set eq*. *evalUni* (*elimVarUni-atom L'* (*EqUni l*)) *x*) = (∃ (*a'*, *b'*, *c'*)∈*set eq*. *evalUni* (*elimVarUni-atom L'* (*EqUni* (*a'*, *b'*, *c'*))) *x*)
by (*metis* (*no-types*, *lifting*) *case-prodE case-prodI2*)
have *c2*: (∃ *l*∈*set les*. *evalUni* (*elimVarUni-atom L'* (*LessUni l*)) *x*) = (∃ (*a'*, *b'*, *c'*)∈*set les*. *evalUni* (*elimVarUni-atom L'* (*LessUni* (*a'*, *b'*, *c'*))) *x*)
by (*metis* (*no-types*, *lifting*) *case-prodE case-prodI2*)
have *c3*: (∃ *l*∈*set leq*. *evalUni* (*elimVarUni-atom L'* (*LeqUni l*)) *x*) = (∃ (*a'*, *b'*, *c'*)∈*set leq*. *evalUni* (*elimVarUni-atom L'* (*LeqUni* (*a'*, *b'*, *c'*))) *x*)
by (*metis* (*no-types*, *lifting*) *case-prodE case-prodI2*)
have *c4*: (∃ *l*∈*set neq*. *evalUni* (*elimVarUni-atom L'* (*NeqUni l*)) *x*) = (∃ (*a'*, *b'*, *c'*)∈*set neq*. *evalUni* (*elimVarUni-atom L'* (*NeqUni* (*a'*, *b'*, *c'*))) *x*)
by (*metis* (*no-types*, *lifting*) *case-prodE case-prodI2*)
have *h* : ((∃ *l*∈*EqUni* ‘ *set eq*. *evalUni* (*elimVarUni-atom L' l*) *x*) ∨
(∃ *l*∈*LessUni* ‘ *set les*. *evalUni* (*elimVarUni-atom L' l*) *x*) ∨
(∃ *l*∈*LeqUni* ‘ *set leq*. *evalUni* (*elimVarUni-atom L' l*) *x*) ∨
(∃ *l*∈*NeqUni* ‘ *set neq*. *evalUni* (*elimVarUni-atom L' l*) *x*)
) =
((∃ *l*∈*set eq*. *evalUni* (*elimVarUni-atom L'* (*EqUni l*)) *x*) ∨
(∃ *l*∈*set les*. *evalUni* (*elimVarUni-atom L'* (*LessUni l*)) *x*) ∨
(∃ *l*∈*set leq*. *evalUni* (*elimVarUni-atom L'* (*LeqUni l*)) *x*) ∨
(∃ *l*∈*set neq*. *evalUni* (*elimVarUni-atom L'* (*NeqUni l*)) *x*)

)
by *auto*
then have ... = ((∃(*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*))
using *c1 c2 c3 c4* **by** *auto*
then show *?thesis*
unfolding *set-split*[*OF assms*] *set-append bex-Un image-set*[*symmetric*]
using *case-prodE case-prodI2* **by** *auto*
qed

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(*cases At*)
case (*LessUni p*)
then show *?thesis* **apply**(*cases p*) **apply** *simp* **unfolding** *eval-list-conj-Uni set-split'*[*OF assms*] **by** *simp*
next
case (*EqUni p*)
then show *?thesis* **apply**(*cases p*) **apply** *simp* **unfolding** *eval-list-conj-Uni set-split'*[*OF assms*] **by** *simp*
next
case (*LeqUni p*)
then show *?thesis* **apply**(*cases p*) **apply** *simp* **unfolding** *eval-list-conj-Uni set-split'*[*OF assms*] **by** *simp*
next
case (*NeqUni p*)
then show *?thesis* **apply**(*cases p*) **apply** *simp* **unfolding** *eval-list-conj-Uni set-split'*[*OF assms*] **by** *simp*
qed

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' ≠ *0* ∧

$$
\begin{array}{l}
-\ b'^2 + 4 * a' * c' \leq 0 \ \wedge \\
((\forall (d,\ e,\ f) \in set\ a. \\
\quad evalUni \\
\quad (substInfinitesimalQuadraticUni\ (-\ b')\ 1\ (b'^2 - 4 * a' * c')\ (2 * a') \\
\quad\ (EqUni\ (d,\ e,\ f))) \\
\quad x) \wedge \\
(\forall (d,\ e,\ f) \in set\ b. \\
\quad evalUni \\
\quad (substInfinitesimalQuadraticUni\ (-\ b')\ 1\ (b'^2 - 4 * a' * c')\ (2 * a') \\
\quad\ (LessUni\ (d,\ e,\ f))) \\
\quad x) \wedge \\
(\forall (d,\ e,\ f) \in set\ c. \\
\quad evalUni \\
\quad (substInfinitesimalQuadraticUni\ (-\ b')\ 1\ (b'^2 - 4 * a' * c')\ (2 * a') \\
\quad\ (LeqUni\ (d,\ e,\ f))) \\
\quad x) \wedge \\
(\forall (d,\ e,\ f) \in set\ d. \\
\quad evalUni \\
\quad (substInfinitesimalQuadraticUni\ (-\ b')\ 1\ (b'^2 - 4 * a' * c')\ (2 * a') \\
\quad\ (NeqUni\ (d,\ e,\ f))) \\
\quad x) \vee \\
(\forall (d,\ e,\ f) \in set\ a. \\
\quad evalUni \\
\quad (substInfinitesimalQuadraticUni\ (-\ b')\ (-\ 1)\ (b'^2 - 4 * a' * c')\ (2 * a') \\
\quad\ (EqUni\ (d,\ e,\ f))) \\
\quad x) \wedge \\
(\forall (d,\ e,\ f) \in set\ b. \\
\quad evalUni \\
\quad (substInfinitesimalQuadraticUni\ (-\ b')\ (-\ 1)\ (b'^2 - 4 * a' * c')\ (2 * a') \\
\quad\ (LessUni\ (d,\ e,\ f))) \\
\quad x) \wedge \\
(\forall (d,\ e,\ f) \in set\ c. \\
\quad evalUni \\
\quad (substInfinitesimalQuadraticUni\ (-\ b')\ (-\ 1)\ (b'^2 - 4 * a' * c')\ (2 * a') \\
\quad\ (LeqUni\ (d,\ e,\ f))) \\
\quad x) \wedge \\
(\forall (d,\ e,\ f) \in set\ d. \\
\quad evalUni \\
\quad (substInfinitesimalQuadraticUni\ (-\ b')\ (-\ 1)\ (b'^2 - 4 * a' * c')\ (2 * a') \\
\quad\ (NeqUni\ (d,\ e,\ f))) \\
\quad x))) = \\
\\
((a' = 0 \ \wedge \ b' \neq 0) \ \wedge \\
(\forall (d,\ e,\ f) \in set\ a. \\
\quad (\exists\, y'::real > -c'/b'.\ \forall x::real \in \{-c'/b' < .. y'\}.\ aEvalUni\ (EqUni\ (d,\ e,\ f))
\end{array}
$$

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

proof(*cases a'=0*)
case *True*
then have *a'*: *a'=0* **by** *auto*
then show *?thesis* **proof**(*cases b'=0*)
case *True*

```
    then show ?thesis using a' by auto
  next
    case False
     then show ?thesis using True unfolding infinitesimal-linear'[of b' c' - x,
symmetric, OF False] by auto
  qed
next
  case False
  then have a' : a' ≠ 0 by auto
  then have d : 2 * a' ≠ 0 by auto
  show ?thesis proof(cases 0 ≤ b'^2 - 4 * a' * c')
    case True
    then show ?thesis using False
      unfolding infinitesimal-quad[OF d True, of -b', symmetric] by auto
  next
    case False
    then show ?thesis using a' by auto
  qed
qed

lemma eq-inf : (∀(a, b, c)∈set (a::(real*real*real) list). ∃x. ∀y<x. a * y^2 + b *
y + c = 0) = (∀(a, b, c)∈set a. a=0∧b=0∧c=0)
  using infinity-evalUni-EqUni[of - x] by auto
```

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(cases separateAtoms L)
  case (fields a b c d)
   have a : ⋀ P. (∀l∈set (map EqUni a) ∪ (set (map LessUni b) ∪ (set (map
LeqUni c) ∪ set (map NeqUni d))).P l) =
              ((∀(d,e,f)∈set a. P (EqUni (d,e,f))) ∧ (∀(d,e,f)∈set b. P (LessUni
(d,e,f))) ∧ (∀(d,e,f)∈set c. P (LeqUni (d,e,f))) ∧ (∀(d,e,f)∈set d. P (NeqUni
(d,e,f))))
    by auto
  show ?thesis apply(simp add: eval-list-conj-Uni separate-aEval[OF fields]
       splitAtoms-negInfinity[OF fields] eval-list-disj-Uni
       del:elimVar.simps)
```

unfolding *eval-conj-atom generalVS-DNF.simps*
split-elimVar[*OF fields*]

split-elimvar[*OF fields*]

unfolding *elimVarUni-atom.simps evalUni.simps aEvalUni.simps*
Rings.mult-zero-class.mult-zero-left Groups.add-0 eval-list-conj-Uni Groups.group-add-class.minus-zero

eval-map-all linearSubstitutionUni.simps quadraticSubUni.simps evalUni-if
aEvalUni.simps
set-append a less eq-inf
using *qe* **by** *auto*
qed

lemma *forallx-substNegInf* : (¬*evalUni* (*map-atomUni substNegInfinityUni F*) *x*)
= (∀ *x*. ¬ *evalUni* (*map-atomUni substNegInfinityUni F*) *x*)
proof(*induction F*)
case *TrueFUni*
then show *?case* **by** *simp*
next
case *FalseFUni*
then show *?case* **by** *simp*
next
case (*AtomUni At*)
then show *?case* **apply**(*cases At*) **by** *auto*
next
case (*AndUni F1 F2*)
then show *?case* **by** *auto*
next
case (*OrUni F1 F2*)
then show *?case* **by** *auto*
qed

lemma *linear-subst-map*: *evalUni* (*map-atomUni* (*linearSubstitutionUni b c*) *F*) *x*
= *evalUni F* (−*c*/*b*)
apply(*induction F*)**by** *auto*

lemma *quadratic-subst-map* : *evalUni* (*map-atomUni* (*quadraticSubUni a b c d*)
F) *x* = *evalUni F* ((*a*+*b*∗*sqrt*(*c*))/*d*)
apply(*induction F*)**by** *auto*

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)
  apply(induction L)using convert-atom-change[OF assms] apply simp-all
  by (metis)

lemma negInf-convert :
  assumes convert-atom-list var L (xs′ @ x # xs) = Some L′
  assumes length xs′ = var
  shows (∀f∈set L. eval (substNegInfinity var f) (xs′ @ x # xs))
        = (∀f∈set L′. evalUni (substNegInfinityUni f) x)
  using assms
proof(induction L arbitrary : L′)
  case Nil
  then show ?case by auto
next
  case (Cons a L)
  then show ?case proof(cases a)
    case (Less p)
    have h: MPoly-Type.degree p var < 3 ⟹
          eval (substNegInfinity var (Less p)) (xs′ @ x # xs) = evalUni
           (substNegInfinityUni
             (LessUni
              (insertion (nth-default 0 (xs′ @ x # xs)) (isolate-variable-sparse p var
2),
               insertion (nth-default 0 (xs′ @ x # xs)) (isolate-variable-sparse p var
(Suc 0)),
               insertion (nth-default 0 (xs′ @ x # xs)) (isolate-variable-sparse p var
0))))
           x
     using convert-substNegInfinity[of var Less p xs′ x xs, OF - assms(2)] by simp
   show ?thesis using Cons(2)[symmetric] Cons(1) unfolding Less apply(cases
MPoly-Type.degree p var < 3)
     defer apply simp apply(cases convert-atom-list var L (xs′ @ x # xs)) apply
(simp-all del: substNegInfinity.simps substNegInfinityUni.simps)
      unfolding h
      using assms(2) by presburger
  next
    case (Eq p)
    have h: MPoly-Type.degree p var < 3 ⟹
```

```
        eval (substNegInfinity var (Eq p)) (xs' @ x # xs) = evalUni
         (substNegInfinityUni
           (EqUni
            (insertion (nth-default 0 (xs' @ x # xs)) (isolate-variable-sparse p var
2),
             insertion (nth-default 0 (xs' @ x # xs)) (isolate-variable-sparse p var
(Suc 0)),
             insertion (nth-default 0 (xs' @ x # xs)) (isolate-variable-sparse p var
0))))
         x
     using convert-substNegInfinity[of var Eq p, OF - assms(2)] by simp
   show ?thesis using Cons(2)[symmetric] Cons(1) unfolding Eq apply(cases
MPoly-Type.degree p var < 3)
    defer apply simp apply(cases convert-atom-list var L (xs' @ x # xs)) apply
(simp-all del: substNegInfinity.simps substNegInfinityUni.simps)
    unfolding h assms by auto
 next
   case (Leq p)
   have h: MPoly-Type.degree p var < 3 ⟹
        eval (substNegInfinity var (Leq p)) (xs' @ x # xs) = evalUni
         (substNegInfinityUni
           (LeqUni
            (insertion (nth-default 0 (xs' @ x # xs)) (isolate-variable-sparse p var
2),
             insertion (nth-default 0 (xs' @ x # xs)) (isolate-variable-sparse p var
(Suc 0)),
             insertion (nth-default 0 (xs' @ x # xs)) (isolate-variable-sparse p var
0))))
         x
     using convert-substNegInfinity[of var Leq p, OF - assms(2)] by simp
   show ?thesis using Cons(2) unfolding Leq apply(cases  MPoly-Type.degree
p var < 3)
     defer apply simp
     apply(cases convert-atom-list var L (xs' @ x # xs))
     apply (simp-all del: substNegInfinity.simps substNegInfinityUni.simps)
     unfolding h using Cons.IH assms by auto
 next
   case (Neq p)
   have h: MPoly-Type.degree p var < 3 ⟹
        eval (substNegInfinity var (Neq p)) (xs' @ x # xs) = evalUni
         (substNegInfinityUni
           (NeqUni
            (insertion (nth-default 0 (xs' @ x # xs)) (isolate-variable-sparse p var
2),
             insertion (nth-default 0 (xs' @ x # xs)) (isolate-variable-sparse p var
(Suc 0)),
             insertion (nth-default 0 (xs' @ x # xs)) (isolate-variable-sparse p var
0))))
         x
```

```
    using convert-substNegInfinity[of var Neq p, OF - assms(2)] by simp
   show ?thesis using Cons(2) unfolding Neq apply(cases  MPoly-Type.degree
p var < 3) defer apply simp
    apply(cases convert-atom-list var L (xs' @ x # xs))
    apply (simp-all del: substNegInfinity.simps substNegInfinityUni.simps)
    unfolding h using Cons.IH assms by auto
  qed
qed

lemma elimVar-atom-single :
  assumes convert-atom var A (xs' @ x # xs) = Some A'
  assumes convert-atom-list var L2 (xs' @ x # xs) = Some L2'
  assumes length xs' = var
  shows eval (elimVar var L2 [] A) (xs' @ x # xs) = evalUni (elimVarUni-atom
L2' A') x
proof(cases A)
  case (Less p)
 define a where a = insertion (nth-default 0 (xs' @ x # xs)) (isolate-variable-sparse
p var 2)
  have a-def' : a = insertion (nth-default 0 (xs' @ 0 # xs)) (isolate-variable-sparse
p var 2) unfolding a-def
    using insertion-isovarspars-free[of (xs' @ x # xs) var x p 2 0] assms by auto
 define b where b = insertion (nth-default 0 (xs' @ x # xs)) (isolate-variable-sparse
p var (Suc 0))
  have b-def' : b = insertion (nth-default 0 (xs' @ 0 # xs)) (isolate-variable-sparse
p var (Suc 0)) unfolding b-def
    using insertion-isovarspars-free[of (xs' @ x # xs) var x p (Suc 0) 0] assms by
auto
 define c where c = insertion (nth-default 0 (xs' @ x # xs)) (isolate-variable-sparse
p var 0)
  have c-def' : c = insertion (nth-default 0 (xs' @ 0 # xs)) (isolate-variable-sparse
p var 0) unfolding c-def
    using insertion-isovarspars-free[of (xs' @ x # xs) var x p 0 0] assms by auto
  have linear : b≠0 ⟹ (∀ f∈set L2.
        eval
       (substInfinitesimalLinear var (−isolate-variable-sparse p var 0) (isolate-variable-sparse
p var (Suc 0)) f)
         (xs' @ x # xs)) = (∀ l∈set L2'. evalUni (substInfinitesimalLinearUni b c
l) x)
    using assms(2) proof(induction L2 arbitrary : L2')
    case Nil
    then show ?case by auto
  next
    case (Cons At L2)
    have ∃ At'. convert-atom var At (xs' @ x # xs) = Some At' proof(cases At)
      case (Less p)
     then show ?thesis using Cons(3) apply simp apply(cases MPoly-Type.degree
p var < 3) by simp-all
    next
```

```
    case (Eq p)
   then show ?thesis using Cons(3) apply simp apply(cases MPoly-Type.degree
p var < 3) by simp-all
   next
    case (Leq p)
   then show ?thesis using Cons(3) apply simp apply(cases MPoly-Type.degree
p var < 3) by auto
   next
    case (Neq p)
   then show ?thesis using Cons(3) apply simp apply(cases MPoly-Type.degree
p var < 3) by auto
   qed
  then obtain At' where At' : convert-atom var At (xs' @ x # xs) = Some At'
by auto
   have ∃ L2's. convert-atom-list var L2 (xs' @ x # xs) = Some L2's
    using Cons(3) At'
    apply(cases convert-atom-list var L2 (xs' @ x # xs)) by auto
   then obtain L2's where L2's : convert-atom-list var L2 (xs' @ x # xs) =
Some L2's by auto
   have L2' : L2' = At' # L2's
    using Cons(3) At'
    by (simp-all add: L2's)
   have h : eval
    (substInfinitesimalLinear var (−isolate-variable-sparse p var 0) (isolate-variable-sparse
p var (Suc 0))
         At)
      (xs' @ x # xs) = evalUni (substInfinitesimalLinearUni b c At') x
   proof(cases convert-atom var At (xs' @ x # xs))
    case None
    then show ?thesis using At' apply(cases At)  by simp-all
   next
    case (Some a)
    have h1 : var ∉ vars (isolate-variable-sparse p var (Suc 0))
     by (simp add: not-in-isovarspar)
   have h2 : var ∉ vars (isolate-variable-sparse p var 0)by (simp add: not-in-isovarspar)

    have h :  evalUni (substInfinitesimalLinearUni b c a) x =
   evalUni (substInfinitesimalLinearUni b c At') x
    proof(cases At)
     case (Less p)
      then show ?thesis using At'[symmetric] Some[symmetric] apply(cases
MPoly-Type.degree p var < 3) by simp-all
    next
     case (Eq p)
      then show ?thesis using At'[symmetric] Some[symmetric] apply(cases
MPoly-Type.degree p var < 3) by simp-all
    next
     case (Leq x3)
     then show ?thesis using At' Some by auto
```

```
      next
        case (Neq x4)
        then show ?thesis using At' Some by auto
      qed
    show ?thesis unfolding convert-substInfinitesimalLinear[OF Some b-def[symmetric]
c-def[symmetric] Cons(2) h1 h2 assms(3)]
        using h .
    qed
    show ?case unfolding L2' using h Cons(1)[OF Cons(2) L2's] by auto
  qed
  have quadratic-1 : (a ≠ 0) ⟹
     (4 * a * c ≤ b^2) ⟹ (∀f∈set L2.
          eval
           (substInfinitesimalQuadratic var
             (− isolate-variable-sparse p var (Suc 0)) 1
             ((isolate-variable-sparse p var (Suc 0))^2 −
              4 * isolate-variable-sparse p var 2 * isolate-variable-sparse p var 0)
             (2 * isolate-variable-sparse p var 2) f)
           (xs' @ x # xs)) = (∀ l∈set L2'.
          evalUni
           (substInfinitesimalQuadraticUni (− b) 1 (b^2 − 4 * a * c) (2 * a) l)
           x)
    using assms(2) proof(induction L2 arbitrary: L2')
    case Nil
    then show ?case by auto
  next
    case (Cons At L2)
    have ∃ At'. convert-atom var At (xs' @ x # xs) = Some At' proof(cases At)
      case (Less p)
    then show ?thesis using Cons(4) apply simp apply(cases MPoly-Type.degree
p var < 3) by simp-all
    next
      case (Eq p)
    then show ?thesis using Cons(4) apply simp apply(cases MPoly-Type.degree
p var < 3) by simp-all
    next
      case (Leq p)
    then show ?thesis using Cons(4) apply simp apply(cases MPoly-Type.degree
p var < 3) by auto
    next
      case (Neq p)
    then show ?thesis using Cons(4) apply simp apply(cases MPoly-Type.degree
p var < 3) by auto
    qed
    then obtain At' where At' : convert-atom var At (xs' @ x # xs) = Some At'
by auto
    have ∃ L2's. convert-atom-list var L2 (xs' @ x # xs) = Some L2's
      using Cons(4) At'
      apply(cases convert-atom-list var L2 (xs' @ x # xs)) by auto
```

then obtain *L2's* **where** *L2's* : *convert-atom-list var L2* (*xs'* @ *x* # *xs*) = *Some L2's* **by** *auto*
have *L2'* : *L2'* = *At'* # *L2's*
using *Cons*(*4*) *At'* **apply**(*cases At*) **apply** *auto*
by (*simp-all add*: *L2's*)
have *h1* : *var* < *length* (*xs'* @ *x* # *xs*) **using** *assms* **by** *auto*
have *h2* : *2*∗*a* ≠*0* **using** *Cons* **by** *auto*
have *h3* : *0*≤*b*^*2*−*4*∗*a*∗*c* **using** *Cons*(*3*) **by** *auto*
have *h4* : *var*∉*vars* ((*isolate-variable-sparse p var* (*Suc 0*))^2 −
4 ∗ *isolate-variable-sparse p var 2* ∗ *isolate-variable-sparse p var 0*)
by (*metis add-uminus-conv-diff not-in-add not-in-isovarspar not-in-mult not-in-neg not-in-pow num-double numeral-times-numeral one-add-one power-0*)
have *h5* : ∀ *xa*. *insertion* (*nth-default 0* ((*xs'* @ *x* # *xs*)[*var* := *xa*])) (− *isolate-variable-sparse p var* (*Suc 0*)) = −*b*
unfolding *insertion-neg b-def*
by (*metis insertion-isovarspars-free list-update-id*)
have *h6* : ∀ *xa*. *insertion* (*nth-default 0* ((*xs'* @ *x* # *xs*)[*var* := *xa*])) *1* = *1* **by** *auto*
have *h7* : ∀ *xa*. *insertion* (*nth-default 0* ((*xs'* @ *x* # *xs*)[*var* := *xa*]))
((*isolate-variable-sparse p var* (*Suc 0*))^2 −
4 ∗ *isolate-variable-sparse p var 2* ∗ *isolate-variable-sparse p var 0*) =
b^2 − *4* ∗ *a* ∗ *c* **apply**(*simp add*: *insertion-four insertion-mult insertion-sub insertion-pow b-def a-def c-def*)
by (*metis insertion-isovarspars-free list-update-id*)
have ⋀*xa*. *insertion* (*nth-default 0* (*xs'* @ *xa* # *xs*)) (*2*::*real mpoly*) = (*2*::*real*)
by (*metis MPoly-Type.insertion-one insertion-add one-add-one*)
then have *h8* : ∀ *xa*. *insertion* (*nth-default 0* ((*xs'* @ *x* # *xs*)[*var* := *xa*])) (*2* ∗ *isolate-variable-sparse p var 2*) = *2* ∗ *a*
unfolding *insertion-mult a-def* **apply** *auto*
by (*metis assms*(*3*) *insertion-lowerPoly1 list-update-length not-in-isovarspar*)
have *h9* : *var*∉*vars*(− *isolate-variable-sparse p var* (*Suc 0*))
by (*simp add*: *not-in-isovarspar not-in-neg*)
have *h10* : *var*∉*vars*(*1*::*real mpoly*)
by (*metis h9 not-in-pow power.simps*(*1*))
have *h11* : *var*∉*vars*(*2* ∗ *isolate-variable-sparse p var 2*)
by (*metis isovarspar-sum mult-2 not-in-isovarspar*)
have *h* : *eval*
(*substInfinitesimalQuadratic var* (− *isolate-variable-sparse p var* (*Suc 0*)) *1*
((*isolate-variable-sparse p var* (*Suc 0*))^2 −
4 ∗ *isolate-variable-sparse p var 2* ∗ *isolate-variable-sparse p var 0*)
(*2* ∗ *isolate-variable-sparse p var 2*) *At*)
(*xs'* @ *x* # *xs*) = *evalUni*
(*substInfinitesimalQuadraticUni* (− *b*) *1* (*b*^{2} − *4* ∗ *a* ∗ *c*) (*2* ∗ *a*) *At'*) *x*
proof (*cases convert-atom var At* (*xs'* @ *x* # *xs*))
case *None*
then show *?thesis* **using** *At'* **apply**(*cases At*) **by** *auto*
next
case (*Some aT*)
have *h1* : *insertion* (*nth-default 0* (*xs'* @ *x* # *xs*)) (− *isolate-variable-sparse*

p var (Suc 0)) = (−b) **unfolding** *b-def insertion-neg* **by** *auto*
have *h2 : insertion (nth-default 0 (xs' @ x # xs)) 1 = 1* **by** *auto*
have *h3 : insertion (nth-default 0 (xs' @ x # xs)) (((isolate-variable-sparse p var (Suc 0))² −*
4 ∗ isolate-variable-sparse p var 2 ∗ isolate-variable-sparse p var 0)) = (b² − 4 ∗ a ∗ c)
unfolding *insertion-mult insertion-pow insertion-four insertion-neg insertion-sub a-def b-def c-def*
by *auto*
have *h4 : insertion (nth-default 0 (xs' @ x # xs)) (2 ∗ isolate-variable-sparse p var 2) = 2 ∗ a*
unfolding *insertion-mult a-def*
by *(metis insertion-add insertion-mult mult-2)*
have *h5 : 2 ∗ a ≠ 0* **using** *Cons* **by** *auto*
have *h6 : 0 ≤ b² − 4 ∗ a ∗ c* **using** *Cons* **by** *auto*
have *h7 : var∉vars(− isolate-variable-sparse p var (Suc 0))*
by *(simp add: not-in-isovarspar not-in-neg)*
have *h8 : var∉vars(1::real mpoly)*
by *(metis h9 not-in-pow power.simps(1))*
have *h9 : var ∉ vars ((isolate-variable-sparse p var (Suc 0))² −*
4 ∗ isolate-variable-sparse p var 2 ∗
isolate-variable-sparse p var 0)
by *(metis add-uminus-conv-diff not-in-add not-in-isovarspar not-in-mult not-in-neg not-in-pow num-double numeral-times-numeral one-add-one power-0)*
have *h10 : var∉vars(2 ∗ isolate-variable-sparse p var 2)*
by *(metis isovarspar-sum mult-2 not-in-isovarspar)*
have *h : evalUni (substInfinitesimalQuadraticUni (− b) 1 (b² − 4 ∗ a ∗ c) (2 ∗ a) aT)*
x =
evalUni (substInfinitesimalQuadraticUni (− b) 1 (b² − 4 ∗ a ∗ c) (2 ∗ a) At')
*x***proof***(cases At)*
case *(Less p)*
then show *?thesis* **using** *At'[symmetric] Some[symmetric]* **apply***(cases MPoly-Type.degree p var < 3)* **by** *auto*
next
case *(Eq p)*
then show *?thesis* **using** *At'[symmetric] Some[symmetric]* **apply***(cases MPoly-Type.degree p var < 3)* **by** *auto*
next
case *(Leq x3)*
then show *?thesis* **using** *At'* **using** *Some* **by** *auto*
next
case *(Neq x4)*
then show *?thesis* **using** *At'* **using** *Some* **by** *auto*
qed
show *?thesis* **unfolding** *convert-substInfinitesimalQuadratic[OF Some h1 h2 h3 h4 h5 h6 h7 h8 h9 h10 assms(3)]*
using *h* **.**
qed

show *?case*
unfolding *L2'* **apply**(*simp del* : *substInfinitesimalQuadratic.simps substInfinitesimalQuadraticUni.simps*)
unfolding
Cons(*1*)[*OF Cons*(*2*) *Cons*(*3*) *L2's*]
unfolding *h*
by *auto*
qed
have *quadratic-2* : $(a \neq 0) \Longrightarrow$
$(4 * a * c \leq b^2) \Longrightarrow (\forall f{\in}set\ L2.$
eval
(*substInfinitesimalQuadratic var*
(− *isolate-variable-sparse p var* (*Suc 0*)) (− *1*)
((*isolate-variable-sparse p var* (*Suc 0*))^2 −
4 ∗ *isolate-variable-sparse p var 2* ∗ *isolate-variable-sparse p var 0*)
(*2* ∗ *isolate-variable-sparse p var 2*) *f*)
(*xs'* @ *x* # *xs*)) = (∀ *l*∈*set L2'*.
evalUni
(*substInfinitesimalQuadraticUni* (− *b*) (− *1*) (b^2 − *4* ∗ *a* ∗ *c*) (*2* ∗ *a*)
l)
x)
using *assms*(*2*) **proof**(*induction L2 arbitrary*: *L2'*)
case *Nil*
then show *?case* **by** *auto*
next
case (*Cons At L2*)
have ∃ *At'*. *convert-atom var At* (*xs'* @ *x* # *xs*) = *Some At'* **proof**(*cases At*)
case (*Less p*)
then show *?thesis* **using** *Cons*(*4*) **apply** *simp* **apply**(*cases MPoly-Type.degree p var* < *3*) **by** *simp-all*
next
case (*Eq p*)
then show *?thesis* **using** *Cons*(*4*) **apply** *simp* **apply**(*cases MPoly-Type.degree p var* < *3*) **by** *simp-all*
next
case (*Leq p*)
then show *?thesis* **using** *Cons*(*4*) **apply** *simp* **apply**(*cases MPoly-Type.degree p var* < *3*) **by** *auto*
next
case (*Neq p*)
then show *?thesis* **using** *Cons*(*4*) **apply** *simp* **apply**(*cases MPoly-Type.degree p var* < *3*) **by** *auto*
qed
then obtain *At'* **where** *At'* : *convert-atom var At* (*xs'* @ *x* # *xs*) = *Some At'* **by** *auto*
have ∃ *L2's*. *convert-atom-list var L2* (*xs'* @ *x* # *xs*) = *Some L2's*
using *Cons*(*4*) *At'*

apply(*cases convert-atom-list var L2* (*xs'* @ *x* # *xs*)) **by** *auto*
then obtain *L2's* **where** *L2's* : *convert-atom-list var L2* (*xs'* @ *x* # *xs*) = *Some L2's* **by** *auto*
have *L2'* : *L2'* = *At'* # *L2's*
using *Cons*(*4*) *At'* **apply**(*cases At*) **apply** *auto*
by (*simp-all add*: *L2's*)
have *h1* : *var* < *length* (*xs'* @ *x* # *xs*) **using** *assms* **by** *auto*
have *h2* : *2*∗*a* ≠*0* **using** *Cons* **by** *auto*
have *h3* : *0*≤*b*^*2*−*4*∗*a*∗*c* **using** *Cons*(*3*) **by** *auto*
have *h4* : *var*∉*vars* ((*isolate-variable-sparse p var* (*Suc 0*))^2 −
4 ∗ *isolate-variable-sparse p var 2* ∗ *isolate-variable-sparse p var 0*)
by (*metis add-uminus-conv-diff not-in-add not-in-isovarspar not-in-mult not-in-neg not-in-pow num-double numeral-times-numeral one-add-one power-0*)
have *h5* : ∀ *xa*. *insertion* (*nth-default 0* ((*xs'* @ *x* # *xs*)[*var* := *xa*])) (− *isolate-variable-sparse p var* (*Suc 0*)) = −*b*
unfolding *insertion-neg b-def*
by (*metis insertion-isovarspars-free list-update-id*)
have *h6* : ∀ *xa*. *insertion* (*nth-default 0* ((*xs'* @ *x* # *xs*)[*var* := *xa*])) (*−1*) = (*−1*) **unfolding** *insertion-neg* **by** *auto*
have *h7* : ∀ *xa*. *insertion* (*nth-default 0* ((*xs'* @ *x* # *xs*)[*var* := *xa*]))
((*isolate-variable-sparse p var* (*Suc 0*))^2 −
4 ∗ *isolate-variable-sparse p var 2* ∗ *isolate-variable-sparse p var 0*) =
b^2 − *4* ∗ *a* ∗ *c* **apply**(*simp add*: *insertion-four insertion-mult insertion-sub insertion-pow b-def a-def c-def*) **using** *assms*
by (*metis insertion-isovarspars-free list-update-id*)
have ⋀*xa*. *insertion* (*nth-default 0* (*xs'* @ *xa* # *xs*)) (*2*::*real mpoly*) = (*2*::*real*)
by (*metis MPoly-Type.insertion-one insertion-add one-add-one*)
then have *h8* : ∀ *xa*. *insertion* (*nth-default 0* ((*xs'* @ *x* # *xs*)[*var* := *xa*])) (*2* ∗ *isolate-variable-sparse p var 2*) = *2* ∗ *a*
unfolding *insertion-mult a-def* **apply** *auto* **using** *assms*
by (*metis* (*no-types*, *opaque-lifting*) *MPoly-Type.insertion-one add.inverse-inverse add-uminus-conv-diff arith-special*(*3*) *insertion-isovarspars-free insertion-neg insertion-sub list-update-id*)
have *h9* : *var*∉*vars*(− *isolate-variable-sparse p var* (*Suc 0*))
by (*simp add*: *not-in-isovarspar not-in-neg*)
have *h10* : *var*∉*vars*(− *1*::*real mpoly*)
by (*metis h9 not-in-neg not-in-pow power.simps*(*1*))
have *h11* : *var*∉*vars*(*2* ∗ *isolate-variable-sparse p var 2*)
by (*metis isovarspar-sum mult-2 not-in-isovarspar*)
have *h* : *eval*
(*substInfinitesimalQuadratic var* (− *isolate-variable-sparse p var* (*Suc 0*)) (*−1*)
((*isolate-variable-sparse p var* (*Suc 0*))^2 −
4 ∗ *isolate-variable-sparse p var 2* ∗ *isolate-variable-sparse p var 0*)
(*2* ∗ *isolate-variable-sparse p var 2*) *At*)
(*xs'* @ *x* # *xs*) = *evalUni*
(*substInfinitesimalQuadraticUni* (− *b*) (*−1*) (*b*^{2} − *4* ∗ *a* ∗ *c*) (*2* ∗ *a*) *At'*) *x*
proof (*cases convert-atom var At* (*xs'* @ *x* # *xs*))
case *None*

then show *?thesis* **using** *At'* **apply**(*cases At*) **by** *auto*
next
case (*Some aT*)
have *h1* : *insertion* (*nth-default 0* (*xs'* @ *x* # *xs*)) (− *isolate-variable-sparse p var* (*Suc 0*)) = (−*b*) **unfolding** *b-def insertion-neg* **by** *auto*
have *h2* : *insertion* (*nth-default 0* (*xs'* @ *x* # *xs*)) (−*1*) = −*1* **unfolding** *insertion-neg* **by** *auto*
have *h3* : *insertion* (*nth-default 0* (*xs'* @ *x* # *xs*)) (((*isolate-variable-sparse p var* (*Suc 0*))^2 −
4 ∗ *isolate-variable-sparse p var 2* ∗ *isolate-variable-sparse p var 0*)) = (b^2 − *4* ∗ *a* ∗ *c*)
unfolding *insertion-mult insertion-pow insertion-four insertion-neg insertion-sub a-def b-def c-def* **using** *assms*
by *auto*
have *h4* : *insertion* (*nth-default 0* (*xs'* @ *x* # *xs*)) (*2* ∗ *isolate-variable-sparse p var 2*) = *2* ∗ *a*
unfolding *insertion-mult a-def*
by (*metis insertion-add insertion-mult mult-2*)
have *h5* : *2* ∗ *a* ≠ *0* **using** *Cons* **by** *auto*
have *h6* : *0* ≤ b^2 − *4* ∗ *a* ∗ *c* **using** *Cons* **by** *auto*
have *h7* : *var*∉*vars*(− *isolate-variable-sparse p var* (*Suc 0*))
by (*simp add*: *not-in-isovarspar not-in-neg*)
have *h8* : *var*∉*vars*(− *1*::*real mpoly*)
by (*simp add*: *h10 not-in-neg*)
have *h9* : *var* ∉ *vars* ((*isolate-variable-sparse p var* (*Suc 0*))^2 −
4 ∗ *isolate-variable-sparse p var 2* ∗
isolate-variable-sparse p var 0)
by (*metis add-uminus-conv-diff not-in-add not-in-isovarspar not-in-mult not-in-neg not-in-pow num-double numeral-times-numeral one-add-one power-0*)
have *h10* : *var*∉*vars*(*2* ∗ *isolate-variable-sparse p var 2*)
by (*metis isovarspar-sum mult-2 not-in-isovarspar*)
have *h* : *evalUni* (*substInfinitesimalQuadraticUni* (− *b*) (−*1*) (b^2 − *4* ∗ *a* ∗ *c*) (*2* ∗ *a*) *aT*)
x =
evalUni (*substInfinitesimalQuadraticUni* (− *b*) (−*1*) (b^2 − *4* ∗ *a* ∗ *c*) (*2* ∗ *a*) *At'*)
*x***proof**(*cases At*)
case (*Less p*)
then show *?thesis* **using** *At'*[*symmetric*] *Some*[*symmetric*] **apply**(*cases MPoly-Type.degree p var* < *3*) **by** *auto*
next
case (*Eq p*)
then show *?thesis* **using** *At'*[*symmetric*] *Some*[*symmetric*] **apply**(*cases MPoly-Type.degree p var* < *3*) **by** *auto*
next
case (*Leq x3*)
then show *?thesis* **using** *At'*
using *Some option.inject* **by** *auto*
next

```
      case (Neq x4)
      then show ?thesis using At'
        using Some by auto
    qed
    show ?thesis unfolding convert-substInfinitesimalQuadratic[OF Some h1 h2
h3 h4 h5 h6 h7 h8 h9 h10 assms(3)]
      using h .
  qed


  show ?case
    unfolding L2' apply(simp del : substInfinitesimalQuadratic.simps substIn-
finitesimalQuadraticUni.simps)
    unfolding
      Cons(1)[OF Cons(2) Cons(3) L2's]
    unfolding h
    by auto
 qed

 show ?thesis using assms(1)[symmetric] unfolding Less apply(cases MPoly-Type.degree
p var < 3) apply simp-all
  apply(simp del : substInfinitesimalLinear.simps substInfinitesimalLinearUni.simps
substInfinitesimalQuadratic.simps substInfinitesimalQuadraticUni.simps
      add: insertion-neg insertion-mult insertion-add insertion-pow insertion-sub
insertion-four
    a-def[symmetric] b-def[symmetric] c-def[symmetric] a-def'[symmetric] b-def'[symmetric]
c-def'[symmetric] eval-list-conj
      eval-list-conj-Uni
      ) using linear quadratic-1 quadratic-2 by smt
next
 case (Eq p)
 define a where a = insertion (nth-default 0 (xs' @ x # xs)) (isolate-variable-sparse
p var 2)
 have a-def' : a = insertion (nth-default 0 (xs' @ 0 # xs)) (isolate-variable-sparse
p var 2) unfolding a-def
  using insertion-isovarspars-free[of xs' @x#xs var x p 2 0] using assms by auto
 define b where b = insertion (nth-default 0 (xs' @ x # xs)) (isolate-variable-sparse
p var (Suc 0))
 have b-def' : b = insertion (nth-default 0 (xs' @ 0 # xs)) (isolate-variable-sparse
p var (Suc 0)) unfolding b-def
  using insertion-isovarspars-free[of xs' @x#xs var x p (Suc 0) 0] using assms
by auto
 define c where c = insertion (nth-default 0 (xs' @ x # xs)) (isolate-variable-sparse
p var 0)
 have c-def' : c = insertion (nth-default 0 (xs' @ 0 # xs)) (isolate-variable-sparse
p var 0) unfolding c-def
  using insertion-isovarspars-free[of xs' @x#xs var x p 0 0]using assms by auto
 have linear : a=0 ⟹ b≠0 ⟹ (∀f∈set L2.
      aEval
```

```
        (linear-substitution var
          (−isolate-variable-sparse p var 0) (isolate-variable-sparse p var (Suc 0))
f)
        (xs' @ x # xs)) = (∀ l∈set L2'. evalUni (linearSubstitutionUni b c l) x)

    using assms(2)
  proof(induction L2 arbitrary: L2')
    case Nil
    then show ?case by auto
  next
    case (Cons At L2)
    have ∃ At'. convert-atom var At (xs' @ x # xs) = Some At' proof(cases At)
      case (Less p)
    then show ?thesis using Cons(4) apply simp apply(cases MPoly-Type.degree
p var < 3) by simp-all
    next
      case (Eq p)
    then show ?thesis using Cons(4) apply simp apply(cases MPoly-Type.degree
p var < 3) by simp-all
    next
      case (Leq p)
    then show ?thesis using Cons(4) apply simp apply(cases MPoly-Type.degree
p var < 3) by auto
    next
      case (Neq p)
    then show ?thesis using Cons(4) apply simp apply(cases MPoly-Type.degree
p var < 3) by auto
    qed
    then obtain At' where At' : convert-atom var At (xs' @ x # xs) = Some At'
by auto
    have ∃ L2's. convert-atom-list var L2 (xs' @ x # xs) = Some L2's
      using Cons(4) At'
      apply(cases convert-atom-list var L2 (xs' @ x # xs)) by auto
    then obtain L2's where L2's : convert-atom-list var L2 (xs' @ x # xs) =
Some L2's by auto
    have L2' : L2' = At' # L2's
      using Cons(4) At' apply(cases At) apply auto
      by (simp-all add: L2's)
    have h1 : var ∉ vars (isolate-variable-sparse p var (Suc 0))
      by (simp add: not-in-isovarspar)
  have h2 : var ∉ vars (isolate-variable-sparse p var 0)by (simp add: not-in-isovarspar)

    have h : aEval
        (linear-substitution var
          (−isolate-variable-sparse p var 0) (isolate-variable-sparse p var (Suc 0))
At)
        (xs' @ x # xs) = evalUni (linearSubstitutionUni b c At') x
    proof(cases convert-atom var At (xs' @ x # xs))
      case None
```

```
    then show ?thesis using At' apply(cases At) by auto
  next
    case (Some a)
    have h : a=At'
      using At' Some by auto
  show ?thesis unfolding convert-linearSubstitutionUni[OF Some b-def[symmetric]
c-def[symmetric] Cons(3) h1 h2 assms(3)]
      unfolding h by auto
  qed
  have (∀f∈set (At # L2).
      aEval
       (linear-substitution var
         (−isolate-variable-sparse p var 0) (isolate-variable-sparse p var (Suc 0))
f)
       (xs' @ x # xs)) = (aEval
       (linear-substitution var
         (−isolate-variable-sparse p var 0)(isolate-variable-sparse p var (Suc 0))
At)
       (xs' @ x # xs)∧ (∀f∈set (L2).
      aEval
       (linear-substitution var
         (−isolate-variable-sparse p var 0) (isolate-variable-sparse p var (Suc 0))
f)
       (xs' @ x # xs))) by auto
  also have ... = (evalUni (linearSubstitutionUni b c At') x ∧
   (∀ l∈set L2's. evalUni (linearSubstitutionUni b c l) x))
    unfolding h Cons(1)[OF Cons(2) Cons(3) L2's] by auto
  finally show ?case  unfolding L2' by auto
 qed

 have quadratic-1 : (a ≠ 0) ⟹
   (4 * a * c ≤ b²) ⟹(∀f∈set L2.
       eval
        (quadratic-sub var (− isolate-variable-sparse p var (Suc 0)) 1
          ((isolate-variable-sparse p var (Suc 0))² −
           4 * isolate-variable-sparse p var 2 * isolate-variable-sparse p var 0)
          (2 * isolate-variable-sparse p var 2) f)
        (xs' @ x # xs)) = (∀ l∈set L2'.
       evalUni (quadraticSubUni (− b) 1 (b² − 4 * a * c) (2 * a) l) x)
  using assms(2) proof(induction L2 arbitrary: L2')
  case Nil
  then show ?case by auto
 next
  case (Cons At L2)
  have ∃ At'. convert-atom var At (xs' @ x # xs) = Some At' proof(cases At)
    case (Less p)
   then show ?thesis using Cons(4) apply simp apply(cases MPoly-Type.degree
p var < 3) by simp-all
  next
```

case (*Eq p*)
then show *?thesis* **using** *Cons*(*4*) **apply** *simp* **apply**(*cases MPoly-Type.degree p var < 3*) **by** *simp-all*
next
case (*Leq p*)
then show *?thesis* **using** *Cons*(*4*) **apply** *simp* **apply**(*cases MPoly-Type.degree p var < 3*) **by** *auto*
next
case (*Neq p*)
then show *?thesis* **using** *Cons*(*4*) **apply** *simp* **apply**(*cases MPoly-Type.degree p var < 3*) **by** *auto*
qed
then obtain *At'* **where** *At'* : *convert-atom var At* (*xs'* @ *x* # *xs*) = *Some At'* **by** *auto*
have ∃ *L2's. convert-atom-list var L2* (*xs'* @ *x* # *xs*) = *Some L2's*
using *Cons*(*4*) *At'*
apply(*cases convert-atom-list var L2* (*xs'* @ *x* # *xs*)) **by** *auto*
then obtain *L2's* **where** *L2's* : *convert-atom-list var L2* (*xs'* @ *x* # *xs*) = *Some L2's* **by** *auto*
have *L2'* : *L2'* = *At'* # *L2's*
using *Cons*(*4*) *At'* **apply**(*cases At*) **apply** *auto*
by (*simp-all add*: *L2's*)
have *h1* : *var* < *length* (*xs'* @ *x* # *xs*) **using** *assms* **by** *auto*
have *h2* : *2*∗*a* ≠*0* **using** *Cons* **by** *auto*
have *h3* : *0*≤*b*^*2*−*4*∗*a*∗*c* **using** *Cons*(*3*) **by** *auto*
have *h4* : *var*∉*vars* ((*isolate-variable-sparse p var* (*Suc 0*))^2 −
4 ∗ *isolate-variable-sparse p var 2* ∗ *isolate-variable-sparse p var 0*)
by (*metis add-uminus-conv-diff not-in-add not-in-isovarspar not-in-mult not-in-neg not-in-pow num-double numeral-times-numeral one-add-one power-0*)
have *h5* : ∀ *xa. insertion* (*nth-default 0* ((*xs'* @ *x* # *xs*)[*var* := *xa*])) (− *isolate-variable-sparse p var* (*Suc 0*)) = −*b*
unfolding *insertion-neg b-def*
by (*metis insertion-isovarspars-free list-update-id*)
have *h6* : ∀ *xa. insertion* (*nth-default 0* ((*xs'* @ *x* # *xs*)[*var* := *xa*])) *1* = *1* **by** *auto*
have *h7* : ∀ *xa. insertion* (*nth-default 0* ((*xs'* @ *x* # *xs*)[*var* := *xa*]))
((*isolate-variable-sparse p var* (*Suc 0*))^2 −
4 ∗ *isolate-variable-sparse p var 2* ∗ *isolate-variable-sparse p var 0*) =
b^2 − *4* ∗ *a* ∗ *c* **apply**(*simp add*: *insertion-four insertion-mult insertion-sub insertion-pow b-def a-def c-def*)
by (*metis insertion-isovarspars-free list-update-id*)
have ⋀*xa. insertion* (*nth-default 0* (*xs'* @ *xa* # *xs*)) (*2*::*real mpoly*) = (*2*::*real*)
by (*metis MPoly-Type.insertion-one insertion-add one-add-one*)
then have *h8* : ∀ *xa. insertion* (*nth-default 0* ((*xs'* @ *x* # *xs*)[*var* := *xa*])) (*2* ∗ *isolate-variable-sparse p var 2*) = *2* ∗ *a*
unfolding *insertion-mult a-def* **apply** *auto*
by (*metis assms*(*3*) *insertion-add insertion-isovarspars-free insertion-mult list-update-length mult-2*)
have *h9* : *var*∉*vars*(− *isolate-variable-sparse p var* (*Suc 0*))

```
    by (simp add: not-in-isovarspar not-in-neg)
  have h10 : var∉vars(1::real mpoly)
    by (metis h9 not-in-pow power.simps(1))
  have h11 : var∉vars(2 * isolate-variable-sparse p var 2)
    by (metis isovarspar-sum mult-2 not-in-isovarspar)
  have h :  eval
   (quadratic-sub var (− isolate-variable-sparse p var (Suc 0)) 1
     ((isolate-variable-sparse p var (Suc 0))^2 −
      4 * isolate-variable-sparse p var 2 * isolate-variable-sparse p var 0)
     (2 * isolate-variable-sparse p var 2) At)
   (xs' @ x # xs) =  aEval At (xs' @ (((− b + 1 * sqrt (b^2 − 4 * a * c)) / (2 *
a)) # xs))
    using quadratic-sub[OF h1 h2 h3 h4 h5 h6 h7 h8, symmetric, of At]
      free-in-quad[OF h9 h10 h4 h11]
    by (metis assms(3) list-update-length var-not-in-eval3)
  have h2 : aEval At (xs' @ (− b + 1 * sqrt (b^2 − 4 * a * c)) / (2 * a) # xs)
= evalUni (quadraticSubUni (− b) 1 (b^2 − 4 * a * c) (2 * a) At') x
  proof(cases At)
    case (Less p)
    then show ?thesis
    proof(cases convert-atom var At (xs' @ x # xs))
      case None
      then show ?thesis
          using At'[symmetric] apply(cases MPoly-Type.degree p var < 3) by
simp-all
    next
      case (Some aT)
      then have Some : ⋀x. convert-atom var At (xs' @ x # xs) = Some aT
        by (metis assms(3) convert-atom-change)
      show ?thesis unfolding aEval-aEvalUni[OF Some assms(3)]
        using At'[symmetric] Some[symmetric]
        unfolding Less apply(cases MPoly-Type.degree p var < 3)  by simp-all
    qed
  next
    case (Eq p)
    then show ?thesis proof(cases convert-atom var At (xs' @ x # xs))
      case None
      then show ?thesis
          using At'[symmetric] apply(cases MPoly-Type.degree p var < 3) by
simp-all
    next
      case (Some aT)
      then have Some : ⋀x. convert-atom var At (xs' @ x # xs) = Some aT
        by (metis assms(3) convert-atom-change)
      show ?thesis unfolding aEval-aEvalUni[OF Some assms(3)]
        using At'[symmetric] Some[symmetric]
        unfolding Eq apply(cases MPoly-Type.degree p var < 3)  by simp-all
    qed
  next
```

```
      case (Leq x3)
      then show ?thesis proof(cases convert-atom var At (xs' @ x # xs))
        case None
        then show ?thesis
            using At'[symmetric] apply(cases MPoly-Type.degree p var < 3) by
simp-all
      next
        case (Some aT)
        then have Some : ⋀x. convert-atom var At (xs' @ x # xs) = Some aT
          by (metis assms(3) convert-atom-change)
        show ?thesis unfolding aEval-aEvalUni[OF Some assms(3)]
          using At'[symmetric] Some[symmetric]
          unfolding Leq apply(cases MPoly-Type.degree p var < 3) by auto
      qed
    next
      case (Neq x4)
      then show ?thesis
      proof(cases convert-atom var At (xs' @ x # xs))
        case None
        then show ?thesis
            using At'[symmetric] apply(cases MPoly-Type.degree p var < 3) by
simp-all
      next
        case (Some aT)
        then have Some : ⋀x. convert-atom var At (xs' @ x # xs) = Some aT
          by (metis assms(3) convert-atom-change)
        show ?thesis unfolding aEval-aEvalUni[OF Some assms(3)]
          using At'[symmetric] Some[symmetric]
          unfolding Neq apply(cases MPoly-Type.degree p var < 3) by auto
      qed
    qed
    show ?case
      unfolding L2' apply(simp del : quadratic-sub.simps quadraticSubUni.simps)
      unfolding
        Cons(1)[OF Cons(2) Cons(3) L2's]
      unfolding h h2
      by auto
  qed
  have quadratic-2 : (a ≠ 0) ⟹
    (4 * a * c ≤ b^2) ⟹ (∀f∈set L2.
        eval
         (quadratic-sub var (- isolate-variable-sparse p var (Suc 0)) (- 1)
           ((isolate-variable-sparse p var (Suc 0))^2 -
            4 * isolate-variable-sparse p var 2 * isolate-variable-sparse p var 0)
           (2 * isolate-variable-sparse p var 2) f)
         (xs' @ x # xs)) = (∀l∈set L2'.
        evalUni (quadraticSubUni (- b) (- 1) (b^2 - 4 * a * c) (2 * a) l) x)
    using assms(2) proof(induction L2 arbitrary: L2')
    case Nil
```

then show *?case* **by** *auto*
next
case (*Cons At L2*)
have ∃ *At'*. *convert-atom var At* (*xs'* @ *x* # *xs*) = *Some At'* **proof**(*cases At*)
case (*Less p*)
then show *?thesis* **using** *Cons*(*4*) **apply** *simp* **apply**(*cases MPoly-Type.degree p var < 3*) **by** *simp-all*
next
case (*Eq p*)
then show *?thesis* **using** *Cons*(*4*) **apply** *simp* **apply**(*cases MPoly-Type.degree p var < 3*) **by** *simp-all*
next
case (*Leq p*)
then show *?thesis* **using** *Cons*(*4*) **apply** *simp* **apply**(*cases MPoly-Type.degree p var < 3*) **by** *auto*
next
case (*Neq p*)
then show *?thesis* **using** *Cons*(*4*) **apply** *simp* **apply**(*cases MPoly-Type.degree p var < 3*) **by** *auto*
qed
then obtain *At'* **where** *At'* : *convert-atom var At* (*xs'* @ *x* # *xs*) = *Some At'* **using** *assms* **by** *auto*
have ∃ *L2's*. *convert-atom-list var L2* (*xs'* @ *x* # *xs*) = *Some L2's*
using *Cons*(*4*) *At'*
apply(*cases convert-atom-list var L2* (*xs'* @ *x* # *xs*)) **by** *auto*
then obtain *L2's* **where** *L2's* : *convert-atom-list var L2* (*xs'* @ *x* # *xs*) = *Some L2's* **by** *auto*
have *L2'* : *L2'* = *At'* # *L2's*
using *Cons*(*4*) *At'* **apply**(*cases At*) **apply** *auto*
by (*simp-all add*: *L2's*)
have *h1* : *var* < *length* (*xs'* @ *x* # *xs*) **using** *assms* **by** *auto*
have *h2* : *2*∗*a* ≠*0* **using** *Cons* **by** *auto*
have *h3* : *0*≤*b*^*2*−*4*∗*a*∗*c* **using** *Cons*(*3*) **by** *auto*
have *h4* : *var*∉*vars* ((*isolate-variable-sparse p var* (*Suc 0*))^2 −
4 ∗ *isolate-variable-sparse p var 2* ∗ *isolate-variable-sparse p var 0*)
by (*metis add-uminus-conv-diff not-in-add not-in-isovarspar not-in-mult not-in-neg not-in-pow num-double numeral-times-numeral one-add-one power-0*)
have *h5* : ∀ *xa*. *insertion* (*nth-default 0* ((*xs'* @ *x* # *xs*)[*var* := *xa*])) (− *isolate-variable-sparse p var* (*Suc 0*)) = −*b*
unfolding *insertion-neg b-def*
by (*metis insertion-isovarspars-free list-update-id*)
have *h6* : ∀ *xa*. *insertion* (*nth-default 0* ((*xs'* @ *x* # *xs*)[*var* := *xa*])) (−*1*) = −*1*
unfolding *insertion-neg*
by *auto*
have *h7* : ∀ *xa*. *insertion* (*nth-default 0* ((*xs'* @ *x* # *xs*)[*var* := *xa*]))
((*isolate-variable-sparse p var* (*Suc 0*))^2 −
4 ∗ *isolate-variable-sparse p var 2* ∗ *isolate-variable-sparse p var 0*) =
b^2 − *4* ∗ *a* ∗ *c* **apply**(*simp add*: *insertion-four insertion-mult insertion-sub*

insertion-pow b-def a-def c-def)
by (*metis insertion-isovarspars-free list-update-id*)
have $\bigwedge xa$. *insertion* (*nth-default 0* (*xs'* @*xa* # *xs*)) (*2*::*real mpoly*) = (*2*::*real*)
by (*metis MPoly-Type.insertion-one insertion-add one-add-one*)
then have *h8* : $\forall$ *xa*. *insertion* (*nth-default 0* ((*xs'* @ *x* # *xs*)[*var* := *xa*])) (*2* ∗ *isolate-variable-sparse p var 2*) = *2* ∗ *a*
unfolding *insertion-mult a-def* **apply** *auto*
by (*metis assms*(*3*) *insertion-lowerPoly1 list-update-length not-in-isovarspar*)
have *h9* : *var*∉*vars*(− *isolate-variable-sparse p var* (*Suc 0*))
by (*simp add*: *not-in-isovarspar not-in-neg*)
have *h10* : *var*∉*vars*(−*1*::*real mpoly*)
by (*metis h9 not-in-neg not-in-pow power.simps*(*1*))
have *h11* : *var*∉*vars*(*2* ∗ *isolate-variable-sparse p var 2*)
by (*metis isovarspar-sum mult-2 not-in-isovarspar*)
have *h* : *eval*
(*quadratic-sub var* (− *isolate-variable-sparse p var* (*Suc 0*)) (−*1*)
((*isolate-variable-sparse p var* (*Suc 0*))^2 −
4 ∗ *isolate-variable-sparse p var 2* ∗ *isolate-variable-sparse p var 0*)
(*2* ∗ *isolate-variable-sparse p var 2*) *At*)
(*xs'* @ *x* # *xs*) = *aEval At* (*xs'* @ (((− *b* − *1* ∗ *sqrt* (*b*^{2} − *4* ∗ *a* ∗ *c*)) / (*2* ∗ *a*)) # *xs*))
using *quadratic-sub*[*OF h1 h2 h3 h4 h5 h6 h7 h8*, *symmetric*, *of At*]
var-not-in-eval3 free-in-quad[*OF h9 h10 h4 h11*]
using *assms*(*3*) **by** *fastforce*
have *h2* : *aEval At* (*xs'* @ (− *b* − *1* ∗ *sqrt* (*b*^{2} − *4* ∗ *a* ∗ *c*)) / (*2* ∗ *a*) # *xs*) = *evalUni* (*quadraticSubUni* (− *b*) (−*1*) (*b*^{2} − *4* ∗ *a* ∗ *c*) (*2* ∗ *a*) *At'*) *x*
proof(*cases At*)
case (*Less p*)
then show *?thesis*
proof(*cases convert-atom var At* (*xs'* @ *x* # *xs*))
case *None*
then show *?thesis*
using *At'*[*symmetric*] **apply**(*cases MPoly-Type.degree p var* < *3*) **by** *simp-all*
next
case (*Some aT*)
then have *Some* : $\bigwedge x$. *convert-atom var At* (*xs'* @ *x* # *xs*) = *Some aT*
by (*metis assms*(*3*) *convert-atom-change*)
show *?thesis* **unfolding** *aEval-aEvalUni*[*OF Some assms*(*3*)]
using *At'*[*symmetric*] *Some*[*symmetric*]
unfolding *Less* **apply**(*cases MPoly-Type.degree p var* < *3*) **by** *simp-all*
qed
next
case (*Eq p*)
then show *?thesis* **proof**(*cases convert-atom var At* (*xs'* @ *x* # *xs*))
case *None*
then show *?thesis*
using *At'*[*symmetric*] **apply**(*cases MPoly-Type.degree p var* < *3*) **by** *simp-all*

```
      next
        case (Some aT)
        then have Some : ⋀x. convert-atom var At (xs' @ x # xs) = Some aT
          by (metis assms(3) convert-atom-change)
        show ?thesis unfolding aEval-aEvalUni[OF Some assms(3)]
          using At'[symmetric] Some[symmetric]
          unfolding Eq apply(cases MPoly-Type.degree p var < 3) by simp-all
      qed
    next
      case (Leq x3)
      then show ?thesis proof(cases convert-atom var At (xs' @ x # xs))
        case None
        then show ?thesis
            using At'[symmetric] apply(cases MPoly-Type.degree p var < 3) by
simp-all
      next
        case (Some aT)
        then have Some : ⋀x. convert-atom var At (xs' @ x # xs) = Some aT
          by (metis assms(3) convert-atom-change)
        show ?thesis unfolding aEval-aEvalUni[OF Some assms(3)]
          using At'[symmetric] Some[symmetric]
          unfolding Leq apply(cases MPoly-Type.degree p var < 3) by auto
      qed
    next
      case (Neq x4)
      then show ?thesis proof(cases convert-atom var At (xs' @ x # xs))
        case None
        then show ?thesis
            using At'[symmetric] apply(cases MPoly-Type.degree p var < 3) by
simp-all
      next
        case (Some aT)
        then have Some : ⋀x. convert-atom var At (xs' @ x # xs) = Some aT
          by (metis assms(3) convert-atom-change)
        show ?thesis unfolding aEval-aEvalUni[OF Some assms(3)]
          using At'[symmetric] Some[symmetric]
          unfolding Neq apply(cases MPoly-Type.degree p var < 3) by auto
      qed
    qed
    show ?case
      unfolding L2' apply(simp del : quadratic-sub.simps quadraticSubUni.simps)
      unfolding
        Cons(1)[OF Cons(2) Cons(3) L2's]
      unfolding h h2
      by auto
  qed
  show ?thesis using assms(1)[symmetric] unfolding Eq apply(cases MPoly-Type.degree
p var < 3) apply simp-all
    apply(simp del : linearSubstitutionUni.simps quadraticSubUni.simps
```

```
        add: insertion-neg insertion-mult insertion-add insertion-pow insertion-sub
insertion-four
      a-def[symmetric] b-def[symmetric] c-def[symmetric] a-def'[symmetric] b-def'[symmetric]
c-def'[symmetric] eval-list-conj
        eval-list-conj-Uni )using linear
    using quadratic-1 quadratic-2
    by smt
next
  case (Leq p)
 define a where a = insertion (nth-default 0 (xs' @ x # xs)) (isolate-variable-sparse
p var 2)
  have a-def' : a = insertion (nth-default 0 (xs' @ 0 # xs)) (isolate-variable-sparse
p var 2) unfolding a-def
    using insertion-isovarspars-free[of xs'@ x#xs var x p 2 0] using assms by auto
 define b where b = insertion (nth-default 0 (xs' @ x # xs)) (isolate-variable-sparse
p var (Suc 0))
  have b-def' : b = insertion (nth-default 0 (xs'@ 0 # xs)) (isolate-variable-sparse
p var (Suc 0)) unfolding b-def
     using insertion-isovarspars-free[of xs'@x#xs var x p (Suc 0) 0] using assms
by auto
 define c where c = insertion (nth-default 0 (xs' @ x # xs)) (isolate-variable-sparse
p var 0)
  have c-def' : c = insertion (nth-default 0 (xs'@ 0 # xs)) (isolate-variable-sparse
p var 0) unfolding c-def
    using insertion-isovarspars-free[of xs'@ x#xs var x p 0 0] using assms by auto
  have linear : a=0 ⟹ b≠0 ⟹ (∀ f∈set L2.
        aEval
         (linear-substitution var
           (−isolate-variable-sparse p var 0) (isolate-variable-sparse p var (Suc 0))
f)
         (xs' @ x # xs)) = (∀ l∈set L2'. evalUni (linearSubstitutionUni b c l) x)
    using assms(2)
  proof(induction L2 arbitrary: L2')
    case Nil
    then show ?case by auto
  next
    case (Cons At L2)
    have ∃ At'. convert-atom var At (xs' @ x # xs) = Some At' proof(cases At)
      case (Less p)
    then show ?thesis using Cons(4) apply simp apply(cases MPoly-Type.degree
p var < 3) by simp-all
    next
      case (Eq p)
    then show ?thesis using Cons(4) apply simp apply(cases MPoly-Type.degree
p var < 3) by simp-all
    next
      case (Leq p)
    then show ?thesis using Cons(4) apply simp apply(cases MPoly-Type.degree
p var < 3) by auto
```

next
case (*Neq p*)
then show *?thesis* **using** *Cons*(*4*) **apply** *simp* **apply**(*cases MPoly-Type.degree p var < 3*) **by** *auto*
qed
then obtain *At'* **where** *At'* : *convert-atom var At* (*xs'* @ *x* # *xs*) = *Some At'* **by** *auto*
have ∃ *L2's. convert-atom-list var L2* (*xs'* @ *x* # *xs*) = *Some L2's*
using *Cons*(*4*) *At'*
apply(*cases convert-atom-list var L2* (*xs'* @ *x* # *xs*)) **by** *auto*
then obtain *L2's* **where** *L2's* : *convert-atom-list var L2* (*xs'* @ *x* # *xs*) = *Some L2's* **by** *auto*
have *L2'* : *L2'* = *At'* # *L2's*
using *Cons*(*4*) *At'* **apply**(*cases At*) **apply** *auto*
by (*simp-all add*: *L2's*)
have *h1* : *var* ∉ *vars* (*isolate-variable-sparse p var* (*Suc 0*))
by (*simp add*: *not-in-isovarspar*)
have *h2* : *var* ∉ *vars* (*isolate-variable-sparse p var 0*)**by** (*simp add*: *not-in-isovarspar*)

have *h* : *aEval*
(*linear-substitution var*
(−*isolate-variable-sparse p var 0*) (*isolate-variable-sparse p var* (*Suc 0*)) *At*)
(*xs'* @ *x* # *xs*) = *evalUni* (*linearSubstitutionUni b c At'*) *x*
proof(*cases convert-atom var At* (*xs'* @ *x* # *xs*))
case *None*
then show *?thesis* **using** *At'* **apply**(*cases At*) **by** *auto*
next
case (*Some a*)
have *h* : *a*=*At'*
using *At' Some* **by** *auto*
show *?thesis* **unfolding** *convert-linearSubstitutionUni*[*OF Some b-def*[*symmetric*] *c-def*[*symmetric*] *Cons*(*3*) *h1 h2 assms*(*3*)]
unfolding *h* **by** *auto*
qed
have (∀ *f*∈*set* (*At* # *L2*).
aEval
(*linear-substitution var*
(−*isolate-variable-sparse p var 0*) (*isolate-variable-sparse p var* (*Suc 0*)) *f*)
(*xs'* @ *x* # *xs*)) = (*aEval*
(*linear-substitution var*
(−*isolate-variable-sparse p var 0*) (*isolate-variable-sparse p var* (*Suc 0*)) *At*)
(*xs'* @ *x* # *xs*)∧ (∀ *f*∈*set* (*L2*).
aEval
(*linear-substitution var*
(−*isolate-variable-sparse p var 0*) (*isolate-variable-sparse p var* (*Suc 0*)) *f*)

```
      (xs' @ x # xs))) by auto
  also have ... = (evalUni (linearSubstitutionUni b c At') x ∧
   (∀ l∈set L2's. evalUni (linearSubstitutionUni b c l) x))
    unfolding h Cons(1)[OF Cons(2) Cons(3) L2's] by auto
  finally show ?case  unfolding L2' by auto
 qed

 have quadratic-1 : (a ≠ 0) ⟹
   (4 * a * c ≤ b^2) ⟹(∀ f∈set L2.
       eval
        (quadratic-sub var (- isolate-variable-sparse p var (Suc 0)) 1
          ((isolate-variable-sparse p var (Suc 0))^2 -
           4 * isolate-variable-sparse p var 2 * isolate-variable-sparse p var 0)
          (2 * isolate-variable-sparse p var 2) f)
        (xs' @ x # xs)) = (∀ l∈set L2'.
       evalUni (quadraticSubUni (- b) 1 (b^2 - 4 * a * c) (2 * a) l) x)
  using assms(2) proof(induction L2 arbitrary: L2')
  case Nil
  then show ?case by auto
 next
  case (Cons At L2)
  have ∃ At'. convert-atom var At (xs' @ x # xs) = Some At' proof(cases At)
   case (Less p)
  then show ?thesis using Cons(4) apply simp apply(cases MPoly-Type.degree
p var < 3) by simp-all
  next
   case (Eq p)
  then show ?thesis using Cons(4) apply simp apply(cases MPoly-Type.degree
p var < 3) by simp-all
  next
   case (Leq p)
  then show ?thesis using Cons(4) apply simp apply(cases MPoly-Type.degree
p var < 3) by auto
  next
   case (Neq p)
  then show ?thesis using Cons(4) apply simp apply(cases MPoly-Type.degree
p var < 3) by auto
  qed
  then obtain At' where At' : convert-atom var At (xs' @ x # xs) = Some At'
by auto
  have ∃ L2's. convert-atom-list var L2 (xs' @ x # xs) = Some L2's
   using Cons(4) At'
   apply(cases convert-atom-list var L2 (xs' @ x # xs)) by auto
  then obtain L2's where L2's : convert-atom-list var L2 (xs' @ x # xs) =
Some L2's by auto
  have L2' : L2' = At' # L2's
   using Cons(4) At' apply(cases At) apply auto
   by (simp-all add: L2's)
  have h1 :  var < length (xs' @ x # xs) using assms by auto
```

have *h2* : *2∗a ≠0* **using** *Cons* **by** *auto*
have *h3* : *0≤b^2−4∗a∗c* **using** *Cons(3)* **by** *auto*
have *h4* : *var∉vars* ((*isolate-variable-sparse p var* (*Suc 0*))^2 −
4 ∗ isolate-variable-sparse p var 2 ∗ isolate-variable-sparse p var 0)
by (*metis add-uminus-conv-diff not-in-add not-in-isovarspar not-in-mult not-in-neg not-in-pow num-double numeral-times-numeral one-add-one power-0*)
have *h5* : ∀ *xa. insertion* (*nth-default 0* ((*xs′* @ *x* # *xs*)[*var* := *xa*])) (− *isolate-variable-sparse p var* (*Suc 0*)) = −*b*
unfolding *insertion-neg b-def*
by (*metis insertion-isovarspars-free list-update-id*)
have *h6* : ∀ *xa. insertion* (*nth-default 0* ((*xs′* @ *x* # *xs*)[*var* := *xa*])) *1* = *1* **by** *auto*
have *h7* : ∀ *xa. insertion* (*nth-default 0* ((*xs′* @ *x* # *xs*)[*var* := *xa*]))
((*isolate-variable-sparse p var* (*Suc 0*))^2 −
4 ∗ isolate-variable-sparse p var 2 ∗ isolate-variable-sparse p var 0) =
b^2 − *4 ∗ a ∗ c* **apply**(*simp add*: *insertion-four insertion-mult insertion-sub insertion-pow b-def a-def c-def*)
by (*metis insertion-isovarspars-free list-update-id*)
have ⋀*xa. insertion* (*nth-default 0* (*xs′* @*xa* # *xs*)) (*2*::*real mpoly*) = (*2*::*real*)
by (*metis MPoly-Type.insertion-one insertion-add one-add-one*)
then have *h8* : ∀ *xa. insertion* (*nth-default 0* ((*xs′* @ *x* # *xs*)[*var* := *xa*])) (*2 ∗ isolate-variable-sparse p var 2*) = *2 ∗ a*
unfolding *insertion-mult a-def* **apply** *auto*
by (*metis assms(3) insertion-lowerPoly1 list-update-length not-in-isovarspar*)
have *h9* : *var∉vars*(− *isolate-variable-sparse p var* (*Suc 0*))
by (*simp add*: *not-in-isovarspar not-in-neg*)
have *h10* : *var∉vars*(*1*::*real mpoly*)
by (*metis h9 not-in-pow power.simps*(*1*))
have *h11* : *var∉vars*(*2 ∗ isolate-variable-sparse p var 2*)
by (*metis isovarspar-sum mult-2 not-in-isovarspar*)
have *h* : *eval*
(*quadratic-sub var* (− *isolate-variable-sparse p var* (*Suc 0*)) *1*
((*isolate-variable-sparse p var* (*Suc 0*))^2 −
4 ∗ isolate-variable-sparse p var 2 ∗ isolate-variable-sparse p var 0)
(*2 ∗ isolate-variable-sparse p var 2*) *At*)
(*xs′* @ *x* # *xs*) = *aEval At* (*xs′* @ (((− *b* + *1 ∗ sqrt* (*b*^{2} − *4 ∗ a ∗ c*)) / (*2 ∗ a*)) # *xs*))
using *quadratic-sub*[*OF h1 h2 h3 h4 h5 h6 h7 h8*, *symmetric*, *of At*]
var-not-in-eval3 free-in-quad[*OF h9 h10 h4 h11*]
by (*metis assms*(*3*) *list-update-length*)
have *h2* : *aEval At* (*xs′* @ (− *b* + *1 ∗ sqrt* (*b*^{2} − *4 ∗ a ∗ c*)) / (*2 ∗ a*) # *xs*) = *evalUni* (*quadraticSubUni* (− *b*) *1* (*b*^{2} − *4 ∗ a ∗ c*) (*2 ∗ a*) *At′*) *x*
proof(*cases At*)
case (*Less p*)
then show *?thesis*
proof(*cases convert-atom var At* (*xs′* @ *x* # *xs*))
case *None*
then show *?thesis*
using *At′*[*symmetric*] **apply**(*cases MPoly-Type.degree p var* < *3*) **by**

```
simp-all
      next
        case (Some aT)
        then have Some : ⋀x. convert-atom var At (xs' @ x # xs) = Some aT
          by (metis assms(3) convert-atom-change)
        show ?thesis unfolding aEval-aEvalUni[OF Some assms(3)]
          using At'[symmetric] Some[symmetric]
          unfolding Less apply(cases MPoly-Type.degree p var < 3) by simp-all
      qed
    next
      case (Eq p)
      then show ?thesis proof(cases convert-atom var At (xs' @ x # xs))
        case None
        then show ?thesis
            using At'[symmetric] apply(cases MPoly-Type.degree p var < 3) by
simp-all
      next
        case (Some aT)
        then have Some : ⋀x. convert-atom var At (xs' @ x # xs) = Some aT
          by (metis assms(3) convert-atom-change)
        show ?thesis unfolding aEval-aEvalUni[OF Some assms(3)]
          using At'[symmetric] Some[symmetric]
          unfolding Eq apply(cases MPoly-Type.degree p var < 3) by simp-all
      qed
    next
      case (Leq x3)
      then show ?thesis proof(cases convert-atom var At (xs' @ x # xs))
        case None
        then show ?thesis
            using At'[symmetric] apply(cases MPoly-Type.degree p var < 3) by
simp-all
      next
        case (Some aT)
        then have Some : ⋀x. convert-atom var At (xs' @ x # xs) = Some aT
          by (metis assms(3) convert-atom-change)
        show ?thesis unfolding aEval-aEvalUni[OF Some assms(3)]
          using At'[symmetric] Some[symmetric]
          unfolding Leq apply(cases MPoly-Type.degree p var < 3) by auto
      qed
    next
      case (Neq x4)
      then show ?thesis
      proof(cases convert-atom var At (xs' @ x # xs))
        case None
        then show ?thesis
            using At'[symmetric] apply(cases MPoly-Type.degree p var < 3) by
simp-all
      next
        case (Some aT)
```

```
      then have Some : ⋀x. convert-atom var At (xs' @ x # xs) = Some aT
        by (metis assms(3) convert-atom-change)
      show ?thesis unfolding aEval-aEvalUni[OF Some assms(3)]
        using At'[symmetric] Some[symmetric]
        unfolding Neq apply(cases MPoly-Type.degree p var < 3) by auto
    qed
  qed
  show ?case
    unfolding L2' apply(simp del : quadratic-sub.simps quadraticSubUni.simps)
    unfolding
      Cons(1)[OF Cons(2) Cons(3) L2's]
    unfolding h h2
    by auto
qed
have quadratic-2 : (a ≠ 0) ⟹
  (4 * a * c ≤ b^2) ⟹ (∀f∈set L2.
      eval
       (quadratic-sub var (− isolate-variable-sparse p var (Suc 0)) (− 1)
         ((isolate-variable-sparse p var (Suc 0))^2 −
          4 * isolate-variable-sparse p var 2 * isolate-variable-sparse p var 0)
         (2 * isolate-variable-sparse p var 2) f)
       (xs' @ x # xs)) = (∀ l∈set L2'.
      evalUni (quadraticSubUni (− b) (− 1) (b^2 − 4 * a * c) (2 * a) l) x)
  using assms(2) proof(induction L2 arbitrary: L2')
  case Nil
  then show ?case by auto
next
  case (Cons At L2)
  have ∃ At'. convert-atom var At (xs' @ x # xs) = Some At' proof(cases At)
    case (Less p)
   then show ?thesis using Cons(4) apply simp apply(cases MPoly-Type.degree
p var < 3) by simp-all
  next
    case (Eq p)
   then show ?thesis using Cons(4) apply simp apply(cases MPoly-Type.degree
p var < 3) by simp-all
  next
    case (Leq p)
   then show ?thesis using Cons(4) apply simp apply(cases MPoly-Type.degree
p var < 3) by auto
  next
    case (Neq p)
   then show ?thesis using Cons(4) apply simp apply(cases MPoly-Type.degree
p var < 3) by auto
  qed
  then obtain At' where At' : convert-atom var At (xs' @ x # xs) = Some At'
by auto
  have ∃ L2's. convert-atom-list var L2 (xs' @ x # xs) = Some L2's
    using Cons(4) At'
```

apply(*cases convert-atom-list var L2* (*xs'* @ *x* # *xs*)) **by** *auto*
then obtain *L2's* **where** *L2's* : *convert-atom-list var L2* (*xs'* @ *x* # *xs*) = *Some L2's* **by** *auto*
have *L2'* : *L2'* = *At'* # *L2's*
using *Cons*(*4*) *At'* **apply**(*cases At*) **apply** *auto*
by (*simp-all add*: *L2's*)
have *h1* : *var* < *length* (*xs'* @ *x* # *xs*) **using** *assms* **by** *auto*
have *h2* : *2*∗*a* ≠*0* **using** *Cons* **by** *auto*
have *h3* : *0*≤*b*^*2*−*4*∗*a*∗*c* **using** *Cons*(*3*) **by** *auto*
have *h4* : *var*∉*vars* ((*isolate-variable-sparse p var* (*Suc 0*))^2 −
4 ∗ *isolate-variable-sparse p var 2* ∗ *isolate-variable-sparse p var 0*)
by (*metis add-uminus-conv-diff not-in-add not-in-isovarspar not-in-mult not-in-neg not-in-pow num-double numeral-times-numeral one-add-one power-0*)
have *h5* : ∀ *xa. insertion* (*nth-default 0* ((*xs'* @ *x* # *xs*)[*var* := *xa*])) (− *isolate-variable-sparse p var* (*Suc 0*)) = −*b*
unfolding *insertion-neg b-def*
by (*metis insertion-isovarspars-free list-update-id*)
have *h6* : ∀ *xa. insertion* (*nth-default 0* ((*xs'* @ *x* # *xs*)[*var* := *xa*])) (−*1*) = −*1*
unfolding *insertion-neg*
by *auto*
have *h7* : ∀ *xa. insertion* (*nth-default 0* ((*xs'* @ *x* # *xs*)[*var* := *xa*]))
((*isolate-variable-sparse p var* (*Suc 0*))^2 −
4 ∗ *isolate-variable-sparse p var 2* ∗ *isolate-variable-sparse p var 0*) =
b^2 − *4* ∗ *a* ∗ *c* **apply**(*simp add*: *insertion-four insertion-mult insertion-sub insertion-pow b-def a-def c-def*)
by (*metis insertion-isovarspars-free list-update-id*)
have ⋀*xa. insertion* (*nth-default 0* (*xs'* @ *xa* # *xs*)) (*2*::*real mpoly*) = (*2*::*real*)
by (*metis MPoly-Type.insertion-one insertion-add one-add-one*)
then have *h8* : ∀ *xa. insertion* (*nth-default 0* ((*xs'* @ *x* # *xs*)[*var* := *xa*])) (*2* ∗ *isolate-variable-sparse p var 2*) = *2* ∗ *a*
unfolding *insertion-mult a-def* **apply** *auto*
by (*metis assms*(*3*) *insertion-lowerPoly1 list-update-length not-in-isovarspar*)
have *h9* : *var*∉*vars*(− *isolate-variable-sparse p var* (*Suc 0*))
by (*simp add*: *not-in-isovarspar not-in-neg*)
have *h10* : *var*∉*vars*(−*1*::*real mpoly*)
by (*metis h9 not-in-neg not-in-pow power.simps*(*1*))
have *h11* : *var*∉*vars*(*2* ∗ *isolate-variable-sparse p var 2*)
by (*metis isovarspar-sum mult-2 not-in-isovarspar*)
have *h* : *eval*
(*quadratic-sub var* (− *isolate-variable-sparse p var* (*Suc 0*)) (−*1*)
((*isolate-variable-sparse p var* (*Suc 0*))^2 −
4 ∗ *isolate-variable-sparse p var 2* ∗ *isolate-variable-sparse p var 0*)
(*2* ∗ *isolate-variable-sparse p var 2*) *At*)
(*xs'* @ *x* # *xs*) = *aEval At* (*xs'* @(((− *b* − *1* ∗ *sqrt* (*b*^{2} − *4* ∗ *a* ∗ *c*)) / (*2* ∗ *a*)) # *xs*))
using *quadratic-sub*[*OF h1 h2 h3 h4 h5 h6 h7 h8*, *symmetric*, *of At*]
var-not-in-eval3 free-in-quad[*OF h9 h10 h4 h11*]
using *assms*(*3*) **by** *fastforce*

```
    have h2 : aEval At (xs' @(- b - 1 * sqrt (b^2 - 4 * a * c)) / (2 * a) # xs)
= evalUni (quadraticSubUni (- b) (-1) (b^2 - 4 * a * c) (2 * a) At') x
    proof(cases At)
      case (Less p)
      then show ?thesis
      proof(cases convert-atom var At (xs' @ x # xs))
        case None
        then show ?thesis
            using At'[symmetric] apply(cases MPoly-Type.degree p var < 3) by
simp-all
      next
        case (Some aT)
        then have Some : ⋀x. convert-atom var At (xs' @ x # xs) = Some aT
          by (metis assms(3) convert-atom-change)
        show ?thesis unfolding aEval-aEvalUni[OF Some assms(3)]
          using At'[symmetric] Some[symmetric]
          unfolding Less apply(cases MPoly-Type.degree p var < 3)  by simp-all
      qed
    next
      case (Eq p)
      then show ?thesis proof(cases convert-atom var At (xs' @ x # xs))
        case None
        then show ?thesis
            using At'[symmetric] apply(cases MPoly-Type.degree p var < 3) by
simp-all
      next
        case (Some aT)
        then have Some : ⋀x. convert-atom var At (xs' @ x # xs) = Some aT
          by (metis assms(3) convert-atom-change)
        show ?thesis unfolding aEval-aEvalUni[OF Some assms(3)]
          using At'[symmetric] Some[symmetric]
          unfolding Eq apply(cases MPoly-Type.degree p var < 3) by simp-all
      qed
    next
      case (Leq x3)
      then show ?thesis proof(cases convert-atom var At (xs' @ x # xs))
        case None
        then show ?thesis
            using At'[symmetric] apply(cases MPoly-Type.degree p var < 3) by
simp-all
      next
        case (Some aT)
        then have Some : ⋀x. convert-atom var At (xs' @ x # xs) = Some aT
          by (metis assms(3) convert-atom-change)
        show ?thesis unfolding aEval-aEvalUni[OF Some assms(3)]
          using At'[symmetric] Some[symmetric]
          unfolding Leq apply(cases MPoly-Type.degree p var < 3)
          by (auto)
      qed
```

```
    next
      case (Neq x4)
      then show ?thesis proof(cases convert-atom var At (xs' @ x # xs))
        case None
        then show ?thesis
            using At'[symmetric] apply(cases MPoly-Type.degree p var < 3) by
simp-all
      next
        case (Some aT)
        then have Some : ⋀x. convert-atom var At (xs' @ x # xs) = Some aT
          by (metis assms(3) convert-atom-change)
        show ?thesis unfolding aEval-aEvalUni[OF Some assms(3)]
          using At'[symmetric] Some[symmetric]
          unfolding Neq apply(cases MPoly-Type.degree p var < 3) by auto
      qed
    qed
    show ?case
     unfolding L2' apply(simp del : quadratic-sub.simps quadraticSubUni.simps)
      unfolding
        Cons(1)[OF Cons(2) Cons(3) L2's]
      unfolding h h2
      by auto
  qed
 show ?thesis using assms(1)[symmetric] unfolding Leq apply(cases MPoly-Type.degree
p var < 3) apply simp-all
    apply(simp del : linearSubstitutionUni.simps quadraticSubUni.simps
        add: insertion-neg insertion-mult insertion-add insertion-pow insertion-sub
insertion-four
      a-def[symmetric] b-def[symmetric] c-def[symmetric] a-def'[symmetric] b-def'[symmetric]
c-def'[symmetric] eval-list-conj
        eval-list-conj-Uni ) using linear
    using quadratic-1 quadratic-2
    by smt
next
  case (Neq p)
 define a where a = insertion (nth-default 0 (xs' @ x # xs)) (isolate-variable-sparse
p var 2)
 have a-def' : a = insertion (nth-default 0 (xs' @ 0 # xs)) (isolate-variable-sparse
p var 2) unfolding a-def
    using insertion-isovarspars-free[of xs'  @x#xs var x p 2 0] using assms by
auto
 define b where b = insertion (nth-default 0 (xs' @ x # xs)) (isolate-variable-sparse
p var (Suc 0))
 have b-def' : b = insertion (nth-default 0 (xs' @ 0 # xs)) (isolate-variable-sparse
p var (Suc 0)) unfolding b-def
    using insertion-isovarspars-free[of xs'@x#xs var x p (Suc 0) 0] using assms
by auto
 define c where c = insertion (nth-default 0 (xs' @ x # xs)) (isolate-variable-sparse
p var 0)
```

have *c-def'* : *c* = *insertion* (*nth-default 0* (*xs'*@*0* # *xs*)) (*isolate-variable-sparse p var 0*) **unfolding** *c-def*
using *insertion-isovarspars-free*[*of xs'*@*x*#*xs var x p 0 0*] **using** *assms* **by** *auto*
have *linear* : *b*≠*0* ⟹ (∀ *f*∈*set L2*.
eval
(*substInfinitesimalLinear var*
(−*isolate-variable-sparse p var 0*) (*isolate-variable-sparse p var* (*Suc 0*)) *f*)
(*xs'* @ *x* # *xs*)) = (∀ *l*∈*set L2'*. *evalUni* (*substInfinitesimalLinearUni b c l*) *x*)
using *assms*(*2*) **proof**(*induction L2 arbitrary* : *L2'*)
case *Nil*
then show *?case* **by** *auto*
next
case (*Cons At L2*)
have ∃ *At'*. *convert-atom var At* (*xs'* @ *x* # *xs*) = *Some At'* **proof**(*cases At*)
case (*Less p*)
then show *?thesis* **using** *Cons*(*3*) **apply** *simp* **apply**(*cases MPoly-Type.degree p var* < *3*) **by** *simp-all*
next
case (*Eq p*)
then show *?thesis* **using** *Cons*(*3*) **apply** *simp* **apply**(*cases MPoly-Type.degree p var* < *3*) **by** *simp-all*
next
case (*Leq p*)
then show *?thesis* **using** *Cons*(*3*) **apply** *simp* **apply**(*cases MPoly-Type.degree p var* < *3*) **by** *auto*
next
case (*Neq p*)
then show *?thesis* **using** *Cons*(*3*) **apply** *simp* **apply**(*cases MPoly-Type.degree p var* < *3*) **by** *auto*
qed
then obtain *At'* **where** *At'* : *convert-atom var At* (*xs'* @ *x* # *xs*) = *Some At'* **by** *auto*
have ∃ *L2's*. *convert-atom-list var L2* (*xs'* @ *x* # *xs*) = *Some L2's*
using *Cons*(*3*) *At'*
apply(*cases convert-atom-list var L2* (*xs'* @ *x* # *xs*)) **by** *auto*
then obtain *L2's* **where** *L2's* : *convert-atom-list var L2* (*xs'* @ *x* # *xs*) = *Some L2's* **by** *auto*
have *L2'* : *L2'* = *At'* # *L2's*
using *Cons*(*3*) *At'*
by (*simp-all add*: *L2's*)
have *h* : *eval*
(*substInfinitesimalLinear var*
(−*isolate-variable-sparse p var 0*) (*isolate-variable-sparse p var* (*Suc 0*)) *At*)
(*xs'* @ *x* # *xs*) = *evalUni* (*substInfinitesimalLinearUni b c At'*) *x*
proof(*cases convert-atom var At* (*xs'* @ *x* # *xs*))
case *None*

```
    then show ?thesis using At' apply(cases At) by simp-all
   next
    case (Some a)
    have h1 : var ∉ vars (isolate-variable-sparse p var (Suc 0))
     by (simp add: not-in-isovarspar)
   have h2 : var ∉ vars (isolate-variable-sparse p var 0)by (simp add: not-in-isovarspar)

    have h :  evalUni (substInfinitesimalLinearUni b c a) x =
   evalUni (substInfinitesimalLinearUni b c At') x
    proof(cases At)
     case (Less p)
      then show ?thesis using At'[symmetric] Some[symmetric] apply(cases
MPoly-Type.degree p var < 3) by simp-all
    next
     case (Eq p)
      then show ?thesis using At'[symmetric] Some[symmetric] apply(cases
MPoly-Type.degree p var < 3) by simp-all
    next
     case (Leq x3)
     then show ?thesis using At' Some by auto
    next
     case (Neq x4)
     then show ?thesis using At' Some by auto
    qed
   show ?thesis unfolding convert-substInfinitesimalLinear[OF Some b-def[symmetric]
c-def[symmetric] Cons(2) h1 h2 assms(3)]
     using h .
  qed
  show ?case unfolding L2' using h Cons(1)[OF Cons(2) L2's] by auto
 qed
 have quadratic-1 : (a ≠ 0) ⟹
   (4 * a * c ≤ b^2) ⟹ (∀ f∈set L2.
       eval
        (substInfinitesimalQuadratic var
          (− isolate-variable-sparse p var (Suc 0)) 1
          ((isolate-variable-sparse p var (Suc 0))^2 −
           4 * isolate-variable-sparse p var 2 * isolate-variable-sparse p var 0)
          (2 * isolate-variable-sparse p var 2) f)
        (xs' @ x # xs)) = (∀ l∈set L2'.
       evalUni
        (substInfinitesimalQuadraticUni (− b) 1 (b^2 − 4 * a * c) (2 * a) l)
        x)
  using assms(2) proof(induction L2 arbitrary: L2')
  case Nil
  then show ?case by auto
 next
  case (Cons At L2)
  have ∃ At'. convert-atom var At (xs' @ x # xs) = Some At' proof(cases At)
   case (Less p)
```

then show *?thesis* **using** *Cons*(*4*) **apply** *simp* **apply**(*cases MPoly-Type.degree p var* < *3*) **by** *simp-all*
next
case (*Eq p*)
then show *?thesis* **using** *Cons*(*4*) **apply** *simp* **apply**(*cases MPoly-Type.degree p var* < *3*) **by** *simp-all*
next
case (*Leq p*)
then show *?thesis* **using** *Cons*(*4*) **apply** *simp* **apply**(*cases MPoly-Type.degree p var* < *3*) **by** *auto*
next
case (*Neq p*)
then show *?thesis* **using** *Cons*(*4*) **apply** *simp* **apply**(*cases MPoly-Type.degree p var* < *3*) **by** *auto*
qed
then obtain *At'* **where** *At'* : *convert-atom var At* (*xs'* @ *x* # *xs*) = *Some At'* **by** *auto*
have ∃ *L2's. convert-atom-list var L2* (*xs'* @ *x* # *xs*) = *Some L2's*
using *Cons*(*4*) *At'*
apply(*cases convert-atom-list var L2* (*xs'* @ *x* # *xs*)) **by** *auto*
then obtain *L2's* **where** *L2's* : *convert-atom-list var L2* (*xs'* @ *x* # *xs*) = *Some L2's* **by** *auto*
have *L2'* : *L2'* = *At'* # *L2's*
using *Cons*(*4*) *At'* **apply**(*cases At*) **apply** *auto*
by (*simp-all add*: *L2's*)
have *h1* : *var* < *length* (*xs'* @ *x* # *xs*) **using** *assms* **by** *auto*
have *h2* : *2*∗*a* ≠*0* **using** *Cons* **by** *auto*
have *h3* : *0*≤*b*^*2*−*4*∗*a*∗*c* **using** *Cons*(*3*) **by** *auto*
have *h4* : *var*∉*vars* ((*isolate-variable-sparse p var* (*Suc 0*))^2 −
4 ∗ *isolate-variable-sparse p var 2* ∗ *isolate-variable-sparse p var 0*)
by (*metis add-uminus-conv-diff not-in-add not-in-isovarspar not-in-mult not-in-neg not-in-pow num-double numeral-times-numeral one-add-one power-0*)
have *h5* : ∀ *xa. insertion* (*nth-default 0* ((*xs'* @ *x* # *xs*)[*var* := *xa*])) (− *isolate-variable-sparse p var* (*Suc 0*)) = −*b*
unfolding *insertion-neg b-def*
by (*metis insertion-isovarspars-free list-update-id*)
have *h6* : ∀ *xa. insertion* (*nth-default 0* ((*xs'* @ *x* # *xs*)[*var* := *xa*])) *1* = *1* **by** *auto*
have *h7* : ∀ *xa. insertion* (*nth-default 0* ((*xs'* @ *x* # *xs*)[*var* := *xa*]))
((*isolate-variable-sparse p var* (*Suc 0*))^2 −
4 ∗ *isolate-variable-sparse p var 2* ∗ *isolate-variable-sparse p var 0*) =
b^2 − *4* ∗ *a* ∗ *c* **apply**(*simp add*: *insertion-four insertion-mult insertion-sub insertion-pow b-def a-def c-def*)
by (*metis insertion-isovarspars-free list-update-id*)
have ⋀*xa. insertion* (*nth-default 0* (*xs'* @*xa* # *xs*)) (*2*::*real mpoly*) = (*2*::*real*)
by (*metis MPoly-Type.insertion-one insertion-add one-add-one*)
then have *h8* : ∀ *xa. insertion* (*nth-default 0* ((*xs'* @ *x* # *xs*)[*var* := *xa*])) (*2* ∗ *isolate-variable-sparse p var 2*) = *2* ∗ *a*
unfolding *insertion-mult a-def* **apply** *auto*

by (*metis assms(3) insertion-lowerPoly1 list-update-length not-in-isovarspar*)
have *h9* : *var∉vars*(− *isolate-variable-sparse p var* (*Suc 0*))
by (*simp add*: *not-in-isovarspar not-in-neg*)
have *h10* : *var∉vars*(*1*::*real mpoly*)
by (*metis h9 not-in-pow power.simps*(*1*))
have *h11* : *var∉vars*(*2* ∗ *isolate-variable-sparse p var 2*)
by (*metis isovarspar-sum mult-2 not-in-isovarspar*)
have *h* : *eval*
(*substInfinitesimalQuadratic var* (− *isolate-variable-sparse p var* (*Suc 0*)) *1*
((*isolate-variable-sparse p var* (*Suc 0*))^2 −
4 ∗ *isolate-variable-sparse p var 2* ∗ *isolate-variable-sparse p var 0*)
(*2* ∗ *isolate-variable-sparse p var 2*) *At*)
(*xs′* @ *x* # *xs*) = *evalUni*
(*substInfinitesimalQuadraticUni* (− *b*) *1* (b^2 − *4* ∗ *a* ∗ *c*) (*2* ∗ *a*) *At′*) *x*
proof (*cases convert-atom var At* (*xs′* @ *x* # *xs*))
case *None*
then show *?thesis* **using** *At′* **apply**(*cases At*) **by** *auto*
next
case (*Some aT*)
have *h1* : *insertion* (*nth-default 0* (*xs′* @ *x* # *xs*)) (− *isolate-variable-sparse p var* (*Suc 0*)) = (−*b*) **unfolding** *b-def insertion-neg* **by** *auto*
have *h2* : *insertion* (*nth-default 0* (*xs′* @ *x* # *xs*)) *1* = *1* **by** *auto*
have *h3* : *insertion* (*nth-default 0* (*xs′* @ *x* # *xs*)) (((*isolate-variable-sparse p var* (*Suc 0*))^2 −
4 ∗ *isolate-variable-sparse p var 2* ∗ *isolate-variable-sparse p var 0*)) = (b^2 − *4* ∗ *a* ∗ *c*)
unfolding *insertion-mult insertion-pow insertion-four insertion-neg insertion-sub a-def b-def c-def*
by *auto*
have *h4* : *insertion* (*nth-default 0* (*xs′* @ *x* # *xs*)) (*2* ∗ *isolate-variable-sparse p var 2*) = *2* ∗ *a*
unfolding *insertion-mult a-def*
by (*metis insertion-add insertion-mult mult-2*)
have *h5* : *2* ∗ *a* ≠ *0* **using** *Cons* **by** *auto*
have *h6* : *0* ≤ b^2 − *4* ∗ *a* ∗ *c* **using** *Cons* **by** *auto*
have *h7* : *var∉vars*(− *isolate-variable-sparse p var* (*Suc 0*))
by (*simp add*: *not-in-isovarspar not-in-neg*)
have *h8* : *var∉vars*(*1*::*real mpoly*)
by (*metis h9 not-in-pow power.simps*(*1*))
have *h9* : *var* ∉ *vars* ((*isolate-variable-sparse p var* (*Suc 0*))^2 −
4 ∗ *isolate-variable-sparse p var 2* ∗
isolate-variable-sparse p var 0)
by (*metis add-uminus-conv-diff not-in-add not-in-isovarspar not-in-mult not-in-neg not-in-pow num-double numeral-times-numeral one-add-one power-0*)
have *h10* : *var∉vars*(*2* ∗ *isolate-variable-sparse p var 2*)
by (*metis isovarspar-sum mult-2 not-in-isovarspar*)
have *h* : *evalUni* (*substInfinitesimalQuadraticUni* (− *b*) *1* (b^2 − *4* ∗ *a* ∗ *c*) (*2* ∗ *a*) *aT*)
x =

```
    evalUni (substInfinitesimalQuadraticUni (− b) 1 (b^2 − 4 ∗ a ∗ c) (2 ∗ a) At')
     xproof(cases At)
       case (Less p)
        then show ?thesis using At'[symmetric] Some[symmetric] apply(cases
MPoly-Type.degree p var < 3) by auto
      next
       case (Eq p)
        then show ?thesis using At'[symmetric] Some[symmetric] apply(cases
MPoly-Type.degree p var < 3) by auto
      next
       case (Leq x3)
       then show ?thesis using At' using Some by auto
      next
       case (Neq x4)
       then show ?thesis using At' using Some by auto
      qed
      show ?thesis unfolding convert-substInfinitesimalQuadratic[OF Some h1 h2
h3 h4 h5 h6 h7 h8 h9 h10 assms(3)]
       using h .
    qed


    show ?case
     unfolding L2' apply(simp del : substInfinitesimalQuadratic.simps substIn-
finitesimalQuadraticUni.simps)
      unfolding
       Cons(1)[OF Cons(2) Cons(3) L2's]
      unfolding h
      by auto
  qed
  have quadratic-2 : (a ≠ 0) ⟹
    (4 ∗ a ∗ c ≤ b^2) ⟹ (∀ f∈set L2.
        eval
         (substInfinitesimalQuadratic var
           (− isolate-variable-sparse p var (Suc 0)) (− 1)
           ((isolate-variable-sparse p var (Suc 0))^2 −
            4 ∗ isolate-variable-sparse p var 2 ∗ isolate-variable-sparse p var 0)
           (2 ∗ isolate-variable-sparse p var 2) f)
         (xs' @ x # xs)) = (∀ l∈set L2'.
        evalUni
         (substInfinitesimalQuadraticUni (− b) (− 1) (b^2 − 4 ∗ a ∗ c) (2 ∗ a)
           l)
         x)
    using assms(2) proof(induction L2 arbitrary: L2')
    case Nil
    then show ?case by auto
  next
    case (Cons At L2)
    have ∃ At'. convert-atom var At (xs' @ x # xs) = Some At' proof(cases At)
```

case (*Less p*)
then show *?thesis* **using** *Cons*(*4*) **apply** *simp* **apply**(*cases MPoly-Type.degree p var < 3*) **by** *simp-all*
next
case (*Eq p*)
then show *?thesis* **using** *Cons*(*4*) **apply** *simp* **apply**(*cases MPoly-Type.degree p var < 3*) **by** *simp-all*
next
case (*Leq p*)
then show *?thesis* **using** *Cons*(*4*) **apply** *simp* **apply**(*cases MPoly-Type.degree p var < 3*) **by** *auto*
next
case (*Neq p*)
then show *?thesis* **using** *Cons*(*4*) **apply** *simp* **apply**(*cases MPoly-Type.degree p var < 3*) **by** *auto*
qed
then obtain *At'* **where** *At'* : *convert-atom var At* (*xs'* @ *x* # *xs*) = *Some At'* **by** *auto*
have ∃ *L2's. convert-atom-list var L2* (*xs'* @ *x* # *xs*) = *Some L2's*
using *Cons*(*4*) *At'*
apply(*cases convert-atom-list var L2* (*xs'* @ *x* # *xs*)) **by** *auto*
then obtain *L2's* **where** *L2's* : *convert-atom-list var L2* (*xs'* @ *x* # *xs*) = *Some L2's* **by** *auto*
have *L2'* : *L2'* = *At'* # *L2's*
using *Cons*(*4*) *At'* **apply**(*cases At*) **apply** *auto*
by (*simp-all add*: *L2's*)
have *h1* : *var* < *length* ((*xs'* @ *x* # *xs*)) **using** *assms* **by** *auto*
have *h2* : *2*∗*a* ≠*0* **using** *Cons* **by** *auto*
have *h3* : *0*≤*b*^*2*−*4*∗*a*∗*c* **using** *Cons*(*3*) **by** *auto*
have *h4* : *var*∉*vars* ((*isolate-variable-sparse p var* (*Suc 0*))^2 −
4 ∗ *isolate-variable-sparse p var 2* ∗ *isolate-variable-sparse p var 0*)
by (*metis add-uminus-conv-diff not-in-add not-in-isovarspar not-in-mult not-in-neg not-in-pow num-double numeral-times-numeral one-add-one power-0*)
have *h5* : ∀ *xa. insertion* (*nth-default 0* ((*xs'* @ *x* # *xs*)[*var* := *xa*])) (− *isolate-variable-sparse p var* (*Suc 0*)) = −*b*
unfolding *insertion-neg b-def*
by (*metis insertion-isovarspars-free list-update-id*)
have *h6* : ∀ *xa. insertion* (*nth-default 0* ((*xs'* @ *x* # *xs*)[*var* := *xa*])) (−*1*) = (−*1*) **unfolding** *insertion-neg* **by** *auto*
have *h7* : ∀ *xa. insertion* (*nth-default 0* ((*xs'* @ *x* # *xs*)[*var* := *xa*]))
((*isolate-variable-sparse p var* (*Suc 0*))^2 −
4 ∗ *isolate-variable-sparse p var 2* ∗ *isolate-variable-sparse p var 0*) =
b^2 − *4* ∗ *a* ∗ *c* **apply**(*simp add*: *insertion-four insertion-mult insertion-sub insertion-pow b-def a-def c-def*)
by (*metis insertion-isovarspars-free list-update-id*)
have ⋀*xa. insertion* (*nth-default 0* (*xs'* @ *xa* # *xs*)) (*2*::*real mpoly*) = (*2*::*real*)
by (*metis MPoly-Type.insertion-one insertion-add one-add-one*)
then have *h8* : ∀ *xa. insertion* (*nth-default 0* ((*xs'* @ *x* # *xs*)[*var* := *xa*])) (*2* ∗ *isolate-variable-sparse p var 2*) = *2* ∗ *a*

unfolding *insertion-mult a-def* **apply** *auto*
by (*metis assms*(*3*) *insertion-lowerPoly1 list-update-length not-in-isovarspar*)
have *h9* : *var*∉*vars*(− *isolate-variable-sparse p var* (*Suc 0*))
by (*simp add*: *not-in-isovarspar not-in-neg*)
have *h10* : *var*∉*vars*(− *1*::*real mpoly*)
by (*metis h9 not-in-neg not-in-pow power.simps*(*1*))
have *h11* : *var*∉*vars*(*2* ∗ *isolate-variable-sparse p var 2*)
by (*metis isovarspar-sum mult-2 not-in-isovarspar*)
have *h* : *eval*
(*substInfinitesimalQuadratic var* (− *isolate-variable-sparse p var* (*Suc 0*)) (−*1*)
((*isolate-variable-sparse p var* (*Suc 0*))^2 −
4 ∗ *isolate-variable-sparse p var 2* ∗ *isolate-variable-sparse p var 0*)
(*2* ∗ *isolate-variable-sparse p var 2*) *At*)
(*xs*′ @ *x* # *xs*) = *evalUni*
(*substInfinitesimalQuadraticUni* (− *b*) (−*1*) (*b*^{2} − *4* ∗ *a* ∗ *c*) (*2* ∗ *a*) *At*′) *x*
proof (*cases convert-atom var At* (*xs*′ @ *x* # *xs*))
case *None*
then show *?thesis* **using** *At*′ **apply**(*cases At*) **by** *auto*
next
case (*Some aT*)
have *h1* : *insertion* (*nth-default 0* (*xs*′ @ *x* # *xs*)) (− *isolate-variable-sparse p var* (*Suc 0*)) = (−*b*) **unfolding** *b-def insertion-neg* **by** *auto*
have *h2* : *insertion* (*nth-default 0* (*xs*′ @ *x* # *xs*)) (−*1*) = −*1* **unfolding** *insertion-neg* **by** *auto*
have *h3* : *insertion* (*nth-default 0* (*xs*′ @ *x* # *xs*)) (((*isolate-variable-sparse p var* (*Suc 0*))^2 −
4 ∗ *isolate-variable-sparse p var 2* ∗ *isolate-variable-sparse p var 0*)) = (*b*^{2} − *4* ∗ *a* ∗ *c*)
unfolding *insertion-mult insertion-pow insertion-four insertion-neg insertion-sub a-def b-def c-def*
by *auto*
have *h4* : *insertion* (*nth-default 0* (*xs*′ @ *x* # *xs*)) (*2* ∗ *isolate-variable-sparse p var 2*) = *2* ∗ *a*
unfolding *insertion-mult a-def*
by (*metis insertion-add insertion-mult mult-2*)
have *h5* : *2* ∗ *a* ≠ *0* **using** *Cons* **by** *auto*
have *h6* : *0* ≤ *b*^{2} − *4* ∗ *a* ∗ *c* **using** *Cons* **by** *auto*
have *h7* : *var*∉*vars*(− *isolate-variable-sparse p var* (*Suc 0*))
by (*simp add*: *not-in-isovarspar not-in-neg*)
have *h8* : *var*∉*vars*(− *1*::*real mpoly*)
by (*simp add*: *h10 not-in-neg*)
have *h9* : *var* ∉ *vars* ((*isolate-variable-sparse p var* (*Suc 0*))^2 −
4 ∗ *isolate-variable-sparse p var 2* ∗
isolate-variable-sparse p var 0)
by (*metis add-uminus-conv-diff not-in-add not-in-isovarspar not-in-mult not-in-neg not-in-pow num-double numeral-times-numeral one-add-one power-0*)
have *h10* : *var*∉*vars*(*2* ∗ *isolate-variable-sparse p var 2*)
by (*metis isovarspar-sum mult-2 not-in-isovarspar*)

have h : *evalUni* (*substInfinitesimalQuadraticUni* $(-\ b)$ (-1) $(b^2 - 4 * a * c)$ $(2 * a)$ aT)
$x =$
evalUni (*substInfinitesimalQuadraticUni* $(-\ b)$ (-1) $(b^2 - 4 * a * c)$ $(2 * a)$ At')
*x***proof**(*cases At*)
case (*Less p*)
then show *?thesis* **using** *At'*[*symmetric*] *Some*[*symmetric*] **apply**(*cases MPoly-Type.degree p var* < 3) **by** *auto*
next
case (*Eq p*)
then show *?thesis* **using** *At'*[*symmetric*] *Some*[*symmetric*] **apply**(*cases MPoly-Type.degree p var* < 3) **by** *auto*
next
case (*Leq x3*)
then show *?thesis* **using** *At'*
using *Some option.inject* **by** *auto*
next
case (*Neq x4*)
then show *?thesis* **using** *At'*
using *Some* **by** *auto*
qed
show *?thesis* **unfolding** *convert-substInfinitesimalQuadratic*[*OF Some h1 h2 h3 h4 h5 h6 h7 h8 h9 h10 assms*(*3*)]
using *h* **.**
qed

show *?case*
unfolding *L2'* **apply**(*simp del* : *substInfinitesimalQuadratic.simps substInfinitesimalQuadraticUni.simps*)
unfolding
Cons(*1*)[*OF Cons*(*2*) *Cons*(*3*) *L2's*]
unfolding *h*
by *auto*
qed

show *?thesis* **using** *assms*(*1*)[*symmetric*] **unfolding** *Neq* **apply**(*cases MPoly-Type.degree p var* < 3) **apply** *simp-all*
apply(*simp del* : *substInfinitesimalLinear.simps substInfinitesimalLinearUni.simps substInfinitesimalQuadratic.simps substInfinitesimalQuadraticUni.simps*
add: *insertion-neg insertion-mult insertion-add insertion-pow insertion-sub insertion-four*
a-def[*symmetric*] *b-def*[*symmetric*] *c-def*[*symmetric*] *a-def'*[*symmetric*] *b-def'*[*symmetric*] *c-def'*[*symmetric*] *eval-list-conj*
eval-list-conj-Uni
) **using** *linear quadratic-1 quadratic-2* **by** *smt*
qed

lemma *convert-list* :
assumes *convert-atom-list var L* (*xs'* @ *x* # *xs*) = *Some L'*
assumes *l*∈*set*(*L*)
shows ∃ *l'*∈ *set L'*. *convert-atom var l* (*xs'* @ *x* # *xs*) = *Some l'*
using *assms*
proof(*induction L arbitrary* : *L'*)
case *Nil*
then show *?case* **by** *auto*
next
case (*Cons At L*)
then show *?case* **proof**(*cases At*)
case (*Less p*)
then show *?thesis* **using** *Cons*(*2*)[*symmetric*] *Cons*(*1*) *Cons*(*3*) **unfolding** *Less* **apply** *simp* **apply**(*cases MPoly-Type.degree p var* < *3*) **apply** *simp-all*
apply(*cases convert-atom-list var L* (*xs'* @ *x* # *xs*)) **apply** *simp-all* **apply**(*cases l* = *Less p*) **by** *simp-all*
next
case (*Eq p*)
show *?thesis* **using** *Cons*(*2*)[*symmetric*] *Cons*(*1*) *Cons*(*3*) **unfolding** *Eq* **apply** *simp* **apply**(*cases MPoly-Type.degree p var* < *3*) **apply** *simp-all*
apply(*cases convert-atom-list var L* (*xs'* @ *x* # *xs*)) **apply** *simp-all* **apply**(*cases l* = *Eq p*) **by** *simp-all*
next
case (*Leq p*)
then show *?thesis* **using** *Cons*(*2*)[*symmetric*] *Cons*(*1*) *Cons*(*3*) **unfolding** *Leq* **apply** *simp* **apply**(*cases MPoly-Type.degree p var* < *3*) **apply** *simp-all*
apply(*cases convert-atom-list var L* (*xs'* @ *x* # *xs*)) **apply** *simp-all* **apply**(*cases l* = *Leq p*) **by** *simp-all*
next
case (*Neq p*)
then show *?thesis* **using** *Cons*(*2*)[*symmetric*] *Cons*(*1*) *Cons*(*3*) **unfolding** *Neq* **apply** *simp* **apply**(*cases MPoly-Type.degree p var* < *3*) **apply** *simp-all*
apply(*cases convert-atom-list var L* (*xs'* @ *x* # *xs*)) **apply** *simp-all* **apply**(*cases l* = *Neq p*) **by** *simp-all*
qed
qed

lemma *convert-list2* :
assumes *convert-atom-list var L* (*xs'* @ *x* # *xs*) = *Some L'*
assumes *l'*∈*set*(*L'*)
shows ∃ *l*∈ *set L*. *convert-atom var l* (*xs'* @ *x* # *xs*) = *Some l'*
using *assms*
proof(*induction L arbitrary* : *L'*)
case *Nil*
then show *?case* **by** *auto*
next
case (*Cons At L*)
then show *?case* **proof**(*cases At*)
case (*Less p*)

then show *?thesis* **using** *Cons(2)*[*symmetric*] *Cons(1) Cons(3)* **unfolding** *Less* **apply** *simp* **apply**(*cases MPoly-Type.degree p var < 3*) **apply** *simp-all*
apply(*cases convert-atom-list var L* (*xs'* @ *x* # *xs*)) **apply** *simp-all*
by *blast*
next
case (*Eq p*)
show *?thesis* **using** *Cons(2)*[*symmetric*] *Cons(1) Cons(3)* **unfolding** *Eq* **apply** *simp* **apply**(*cases MPoly-Type.degree p var < 3*) **apply** *simp-all*
apply(*cases convert-atom-list var L* (*xs'* @ *x* # *xs*)) **apply** *simp-all* **by** *blast*
next
case (*Leq p*)
then show *?thesis* **using** *Cons(2)*[*symmetric*] *Cons(1) Cons(3)* **unfolding** *Leq* **apply** *simp* **apply**(*cases MPoly-Type.degree p var < 3*) **apply** *simp-all*
apply(*cases convert-atom-list var L* (*xs'* @ *x* # *xs*)) **apply** *simp-all* **by** *blast*
next
case (*Neq p*)
then show *?thesis* **using** *Cons(2)*[*symmetric*] *Cons(1) Cons(3)* **unfolding** *Neq* **apply** *simp* **apply**(*cases MPoly-Type.degree p var < 3*) **apply** *simp-all*
apply(*cases convert-atom-list var L* (*xs'* @ *x* # *xs*)) **apply** *simp-all* **by** *blast*
qed
qed

lemma *elimVar-atom-convert* :
assumes *convert-atom-list var L* (*xs'* @ *x* # *xs*) = *Some L'*
assumes *convert-atom-list var L2* (*xs'* @ *x* # *xs*) = *Some L2'*
assumes *length xs'* = *var*
shows (∃*f*∈*set L*. *eval* (*elimVar var L2* [] *f*) (*xs'* @ *x* # *xs*))
= (∃*f*∈*set L'*. *evalUni* (*elimVarUni-atom L2' f*) *x*)
proof *safe*
fix *f*
assume *h* : *f* ∈ *set L*
eval (*elimVar var L2* [] *f*) (*xs'* @ *x* # *xs*)
have ∃*f'*∈*set L'*. *convert-atom var f* (*xs'* @ *x* # *xs*) = *Some f'*
using *convert-list h assms* **by** *auto*
then obtain *f'* **where** *f'* : *f'*∈*set L' convert-atom var f* (*xs'* @ *x* # *xs*) = *Some f'* **by** *metis*
show ∃*f*∈*set L'*. *evalUni* (*elimVarUni-atom L2' f*) *x*
apply(*rule bexI*[**where** *x*=*f'*]) **using** *f' elimVar-atom-single*[*OF f'*(*2*) *assms*(*2*) *assms*(*3*)] *h* **by** *auto*
next
fix *f'*
assume *h* : *f'* ∈ *set L'*
evalUni (*elimVarUni-atom L2' f'*) *x*
have ∃*f*∈*set L*. *convert-atom var f* (*xs'* @ *x* # *xs*) = *Some f'* **using** *convert-list2 h assms* **by** *auto*
then obtain *f* **where** *f* : *f*∈*set L convert-atom var f* (*xs'* @ *x* # *xs*) = *Some f'* **by** *metis*
show ∃*f*∈*set L*. *eval* (*elimVar var L2* [] *f*) (*xs'* @ *x* # *xs*)
apply(*rule bexI*[**where** *x*=*f*]) **using** *f elimVar-atom-single*[*OF f*(*2*) *assms*(*2*)

assms(*3*)] *h* **by** *auto*
qed

lemma *eval-convert* :
assumes *convert-atom-list var L* (*xs'* @ *x* # *xs*) = *Some L'*
assumes *length xs'* = *var*
shows (∀ *f*∈*set L*. *aEval f* (*xs'* @ *x* # *xs*)) = (∀ *f*∈*set L'*. *aEvalUni f x*)
using *assms*
proof(*induction L arbitrary* : *L'*)
case *Nil*
then show *?case* **by** *auto*
next
case (*Cons a L*)
then show *?case* **proof**(*cases a*)
case (*Less p*)
then show *?thesis* **using** *Cons*(*2*)[*symmetric*] *Cons*(*1*) *Cons*(*3*) **unfolding** *Less* **apply**(*cases MPoly-Type.degree p var* < *3*)
apply *simp-all* **apply**(*cases convert-atom-list var L* (*xs'* @ *x* # *xs*)) **apply** *simp-all*
by (*simp add*: *poly-to-univar*)
next
case (*Eq p*)
then show *?thesis* **using** *Cons*(*2*)[*symmetric*] *Cons*(*1*) *Cons*(*3*) **unfolding** *Eq* **apply**(*cases MPoly-Type.degree p var* < *3*)
apply *simp-all* **apply**(*cases convert-atom-list var L* (*xs'* @ *x* # *xs*)) **apply** *simp-all*
by (*simp add*: *poly-to-univar*)
next
case (*Leq p*)
show *?thesis* **using** *Cons*(*2*)[*symmetric*] *Cons*(*1*) *Cons*(*3*) **unfolding** *Leq* **apply**(*cases MPoly-Type.degree p var* < *3*)
apply *simp-all* **apply**(*cases convert-atom-list var L* (*xs'* @ *x* # *xs*)) **apply** *simp-all*
by (*simp add*: *poly-to-univar*)
next
case (*Neq p*)
show *?thesis* **using** *Cons*(*2*)[*symmetric*] *Cons*(*1*) *Cons*(*3*) **unfolding** *Neq* **apply**(*cases MPoly-Type.degree p var* < *3*)
apply *simp-all* **apply**(*cases convert-atom-list var L* (*xs'* @ *x* # *xs*)) **apply** *simp-all*
by (*simp add*: *poly-to-univar*)
qed
qed
lemma *all-degree-2-convert* :
assumes *all-degree-2 var L*
shows ∃ *L'*. *convert-atom-list var L xs* = *Some L'*
using *assms*
proof(*induction L*)

case *Nil*
then show *?case* **by** *auto*
next
case (*Cons a L*)
then show *?case* **proof**(*cases a*)
case (*Less p*)
show *?thesis* **using** *Cons* **unfolding** *Less all-degree-2.simps convert-atom-list.simps convert-atom.simps*
using *degree-convert-eq*[*of var p xs*] **by** *auto*
next
case (*Eq p*)
then show *?thesis* **using** *Cons* **unfolding** *Eq all-degree-2.simps convert-atom-list.simps convert-atom.simps*
using *degree-convert-eq*[*of var p xs*] **by** *auto*
next
case (*Leq x3*)
then show *?thesis* **using** *Cons* **by** *auto*
next
case (*Neq x4*)
then show *?thesis* **using** *Cons* **by** *auto*
qed
qed
lemma *gen-qe-eval* :
assumes *hlength* : *length xs* = *var*
shows (∃ *x*. (*eval* (*list-conj* ((*map Atom L*) @ *F*)) (*xs* @ (*x*#Γ)))) = (∃ *x*.(*eval* (*gen-qe var L F*) (*xs* @ (*x*#Γ))))
proof(*cases luckyFind var L* [])
case *None*
then have *notLucky* : *luckyFind var L* [] = *None* **by** *auto*
then show *?thesis* **proof**(*cases F*)
case *Nil*
then show *?thesis* **proof**(*cases all-degree-2 var L*)
case *True*
then have ⋀*x*.∃ *L*′. *convert-atom-list var L* (*xs*@*x*#Γ) = *Some L*′ **using** *all-degree-2-convert*[*of var L xs*@-#Γ] **by** *auto*
then obtain *L*′ **where** *L*′ : *convert-atom-list var L* (*xs*@*x*#Γ) = *Some L*′ **by** *metis*
then have *L*′ : ⋀*x*. *convert-atom-list var L* (*xs*@*x*#Γ) = *Some L*′
by (*metis convert-atom-list-change hlength*)
show *?thesis*
unfolding *Nil* **apply** (*simp add*:*eval-list-conj eval-list-disj True del*:*luckyFind.simps*)
unfolding *notLucky* **apply** (*simp add*:*eval-list-conj eval-list-disj*)
using *negInf-convert*[*OF L*′ *assms*] *elimVar-atom-convert*[*OF L*′ *L*′ *assms*] *eval-convert*[*OF L*′ *assms*]
using *eval-generalVS*′′[*of L*′] **unfolding** *eval-list-conj-Uni generalVS-DNF.simps eval-list-conj-Uni eval-list-disj-Uni eval-append eval-map eval-map-all evalUni.simps*

by *auto*

```
    next
      case False
      then show ?thesis using notLucky unfolding Nil  False apply simp
        by (metis append-Nil2 hlength notLucky option.simps(4) qe-eq-repeat.simps
qe-eq-repeat-eval)
    qed
  next
    case (Cons a list)
    show ?thesis
      apply(simp add:Cons del:qe-eq-repeat.simps)
      apply(rule qe-eq-repeat-eval[of xs var L a # list Γ])
      using assms .
  qed
next
  case (Some a)
  then show ?thesis
    using luckyFind-eval[OF Some assms] apply(cases F) apply simp
    apply(simp add:Cons del:qe-eq-repeat.simps)
    using qe-eq-repeat-eval[of xs var L - Γ]
    using assms  by auto
qed


lemma freeIn-elimVar : freeIn var (elimVar var L F A)
proof(cases A)
  case (Less p)
  have two: 2 = Suc(Suc 0) by auto
  have notIn4: var ∉ vars (4::real mpoly)
     by (metis isolate-var-one not-in-add not-in-isovarspar numeral-plus-numeral
one-add-one semiring-norm(2) semiring-norm(6))
  show ?thesis using Less apply auto
    using not-in-isovarspar apply force+
    apply (rule freeIn-list-conj)
    apply auto
    defer defer
    using not-in-isovarspar apply force+
   using not-in-sub[OF not-in-mult[of var 4, OF - not-in-mult[of var isolate-variable-sparse
p var 2 isolate-variable-sparse p var 0]], of (isolate-variable-sparse p var (Suc 0))^2]
    apply (simp add:not-in-isovarspar two)
   using not-in-mult[of var isolate-variable-sparse p var (Suc 0) isolate-variable-sparse
p var (Suc 0)]
    apply (simp add:not-in-isovarspar notIn4)
    apply (simp add: ideal.scale-scale)
    apply(rule freeIn-list-conj)
    apply auto
    defer defer
    apply(rule freeIn-list-conj)
    apply auto
    apply(rule freeIn-substInfinitesimalQuadratic) apply auto
```

using *not-in-isovarspar not-in-neg* **apply** *blast*
apply (*metis not-in-isovarspar not-in-neg not-in-pow power-0*)
using *notIn4 not-in-isovarspar not-in-mult not-in-pow not-in-sub* **apply** *auto*[*1*]
apply (*metis isovarspar-sum mult-2 not-in-isovarspar*)
using *freeIn-substInfinitesimalQuadratic-fm*[*of var* (− *isolate-variable-sparse p var* (*Suc 0*)) −*1* ((*isolate-variable-sparse p var* (*Suc 0*))^2 − *4* ∗ *isolate-variable-sparse p var 2* ∗ *isolate-variable-sparse p var 0*) (*2* ∗ *isolate-variable-sparse p var 2*)] **apply** *auto*[*1*]
apply (*metis* (*no-types*, *lifting*) *mult-2 notIn4 not-in-add not-in-isovarspar not-in-mult not-in-neg not-in-pow not-in-sub power-0*)
apply(*rule freeIn-substInfinitesimalLinear*)
apply (*meson not-in-isovarspar not-in-neg*)
apply (*simp add*: *not-in-isovarspar*)
using *freeIn-substInfinitesimalLinear-fm*
using *not-in-isovarspar not-in-neg* **apply** *force*
apply (*metis* (*no-types*, *lifting*) ‹⟦*var* ∉ *vars 4*; *var* ∉ *vars* (*isolate-variable-sparse p var 2*); *var* ∉ *vars* (*isolate-variable-sparse p var 0*); *var* ∉ *vars* ((*isolate-variable-sparse p var* (*Suc 0*))^2)⟧ ⟹ *var* ∉ *vars* (*4* ∗ (*isolate-variable-sparse p var 2* ∗ *isolate-variable-sparse p var 0*) − (*isolate-variable-sparse p var* (*Suc 0*))^2)› *freeIn-substInfinitesimalQuadratic minus-diff-eq mult.assoc mult-2 notIn4 not-in-add not-in-isovarspar not-in-neg not-in-pow power-0*)
using *freeIn-substInfinitesimalQuadratic-fm*[*of var* (− *isolate-variable-sparse p var* (*Suc 0*)) *1* ((*isolate-variable-sparse p var* (*Suc 0*))^2 − *4* ∗ *isolate-variable-sparse p var 2* ∗ *isolate-variable-sparse p var 0*) (*2* ∗ *isolate-variable-sparse p var 2*)]
apply *auto*
by (*metis* (*no-types*, *lifting*) ‹⟦*var* ∉ *vars 4*; *var* ∉ *vars* (*isolate-variable-sparse p var 2*); *var* ∉ *vars* (*isolate-variable-sparse p var 0*); *var* ∉ *vars* ((*isolate-variable-sparse p var* (*Suc 0*))^2)⟧ ⟹ *var* ∉ *vars* (*4* ∗ (*isolate-variable-sparse p var 2* ∗ *isolate-variable-sparse p var 0*) − (*isolate-variable-sparse p var* (*Suc 0*))^2)› *ideal.scale-scale minus-diff-eq mult-2 notIn4 not-in-add not-in-isovarspar not-in-neg not-in-pow power-0*)
next
case (*Eq p*)
then show *?thesis* **using** *freeIn-elimVar-eq* **by** *auto*
next
case (*Leq p*)
then show *?thesis* **using** *freeIn-elimVar-eq* **by** *auto*
next
case (*Neq p*)
have *two*: *2* = *Suc*(*Suc 0*) **by** *auto*
have *notIn4*: *var* ∉ *vars* (*4*::*real mpoly*)
by (*metis isolate-var-one not-in-add not-in-isovarspar numeral-plus-numeral one-add-one semiring-norm*(*2*) *semiring-norm*(*6*))
show *?thesis* **using** *Neq* **apply** *auto*
using *not-in-isovarspar* **apply** *force*+
apply (*rule freeIn-list-conj*)
apply *auto*
defer defer
using *not-in-isovarspar* **apply** *force*+

using *not-in-sub*[*OF not-in-mult*[*of var 4*, *OF - not-in-mult*[*of var isolate-variable-sparse p var 2 isolate-variable-sparse p var 0*]], *of* (*isolate-variable-sparse p var* (*Suc 0*))^2]
apply (*simp add*:*not-in-isovarspar two*)
using *not-in-mult*[*of var isolate-variable-sparse p var* (*Suc 0*) *isolate-variable-sparse p var* (*Suc 0*)]
apply (*simp add*:*not-in-isovarspar notIn4*)
apply (*simp add*: *ideal.scale-scale*)
apply(*rule freeIn-list-conj*)
apply *auto*
defer defer
apply(*rule freeIn-list-conj*)
apply *auto*
apply(*rule freeIn-substInfinitesimalQuadratic*) **apply** *auto*
using *not-in-isovarspar not-in-neg* **apply** *blast*
apply (*metis not-in-isovarspar not-in-neg not-in-pow power-0*)
using *notIn4 not-in-isovarspar not-in-mult not-in-pow not-in-sub* **apply** *auto*[*1*]
apply (*metis isovarspar-sum mult-2 not-in-isovarspar*)
using *freeIn-substInfinitesimalQuadratic-fm*[*of var* (− *isolate-variable-sparse p var* (*Suc 0*)) −*1* ((*isolate-variable-sparse p var* (*Suc 0*))^2 − *4* ∗ *isolate-variable-sparse p var 2* ∗ *isolate-variable-sparse p var 0*) (*2* ∗ *isolate-variable-sparse p var 2*)] **apply** *auto*[*1*]
apply (*metis* (*no-types*, *lifting*) *mult-2 notIn4 not-in-add not-in-isovarspar not-in-mult not-in-neg not-in-pow not-in-sub power-0*)
apply(*rule freeIn-substInfinitesimalLinear*)
apply (*meson not-in-isovarspar not-in-neg*)
apply (*simp add*: *not-in-isovarspar*)
using *freeIn-substInfinitesimalLinear-fm*
using *not-in-isovarspar not-in-neg* **apply** *force*
apply (*metis* (*no-types*, *lifting*) ‹⟦*var* ∉ *vars 4*; *var* ∉ *vars* (*isolate-variable-sparse p var 2*); *var* ∉ *vars* (*isolate-variable-sparse p var 0*); *var* ∉ *vars* ((*isolate-variable-sparse p var* (*Suc 0*))^2)⟧ ⟹ *var* ∉ *vars* (*4* ∗ (*isolate-variable-sparse p var 2* ∗ *isolate-variable-sparse p var 0*) − (*isolate-variable-sparse p var* (*Suc 0*))^2)› *freeIn-substInfinitesimalQuadratic minus-diff-eq mult.assoc mult-2 notIn4 not-in-add not-in-isovarspar not-in-neg not-in-pow power-0*)
using *freeIn-substInfinitesimalQuadratic-fm*[*of var* (− *isolate-variable-sparse p var* (*Suc 0*)) *1* ((*isolate-variable-sparse p var* (*Suc 0*))^2 − *4* ∗ *isolate-variable-sparse p var 2* ∗ *isolate-variable-sparse p var 0*) (*2* ∗ *isolate-variable-sparse p var 2*)]
apply *auto*
by (*metis* (*no-types*, *lifting*) ‹⟦*var* ∉ *vars 4*; *var* ∉ *vars* (*isolate-variable-sparse p var 2*); *var* ∉ *vars* (*isolate-variable-sparse p var 0*); *var* ∉ *vars* ((*isolate-variable-sparse p var* (*Suc 0*))^2)⟧ ⟹ *var* ∉ *vars* (*4* ∗ (*isolate-variable-sparse p var 2* ∗ *isolate-variable-sparse p var 0*) − (*isolate-variable-sparse p var* (*Suc 0*))^2)› *ideal.scale-scale minus-diff-eq mult-2 notIn4 not-in-add not-in-isovarspar not-in-neg not-in-pow power-0*)
qed

lemma *freeInDisj*: *freeIn var* (*list-disj* (*list-conj* (*map* (*substNegInfinity var*) *L*) # *map* (*elimVar var L* []) *L*))
apply(*rule freeIn-list-disj*)

apply(*auto*)
apply(*rule freeIn-list-conj*)
apply *simp*

using *freeIn-substNegInfinity*[*of var*]
apply *simp*
using *freeIn-elimVar*
by *simp*

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−
have *h* : (∃ *x*. (*eval* (*list-conj* (*map Atom L*)) (*xs′*@*x*#Γ))) = (∃ *x*. *eval* (*gen-qe var L* []) (*xs′*@*x* # Γ))
using *gen-qe-eval*[*OF assms*(*2*), *of L* [] Γ] **unfolding** *List.append.left-neutral* **by** *auto*
show (∃ *x*. (*eval* (*list-conj* (*map Atom L*)) (*xs′*@*x*#Γ))) = (∀ *x*.(*eval* (*gen-qe var L* []) (*xs′*@*x*#Γ)))
unfolding *h*
apply (*simp add*:*assms*)
apply(*cases find-lucky-eq var L*)
apply *simp* **using** *freeInDisj*[*of var L*]
using *var-not-in-eval3*[*OF - assms*(*2*)] **apply** *blast*
subgoal for *a*
using *freeIn-elimVar-eq*[*of var L* [] *a*]
apply(*simp del*:*elimVar.simps*)
using *var-not-in-eval3*[*OF - assms*(*2*)] **by** *blast*
done
next
show *freeIn var* (*gen-qe var L* [])
apply(*simp add*:*assms*)
apply(*cases find-lucky-eq var L*) **apply** (*simp add*:*freeInDisj*)
subgoal for *a*
using *freeIn-elimVar-eq*[*of var L* [] *a*]
by(*simp del*:*elimVar.simps*)
done
qed

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(*cases convert-atom-list var L* (*xs'*@*x*#Γ))
case *None*
then show *?thesis* **using** *all-degree-2-convert*[*OF assms*(*1*), *of* (*xs'* @ *x* # Γ)]
by *auto*
next
case (*Some a*)
then have *Some* : ⋀*x*. *convert-atom-list var L* (*xs'*@*x*#Γ) = *Some a* **using** *convert-atom-list-change*[*OF assms*(*2*), *of L x* Γ]
by *fastforce*

show *?thesis*
apply (*simp add*: *eval-list-conj eval-list-disj*)
using *negInf-convert*[*OF Some assms*(*2*)] *elimVar-atom-convert*[*OF Some Some assms*(*2*)] *eval-convert*[*OF Some assms*(*2*)]
using *eval-generalVS''*[*of a*] **unfolding** *eval-list-conj-Uni generalVS-DNF.simps eval-list-conj-Uni eval-list-disj-Uni eval-append eval-map eval-map-all evalUni.simps*
by *auto*
qed

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(*induction* φ)
case (*And* φ*1* φ*2*)
define *f* **where** *f* = (λ*a*. *case a of*
(*al*, *fl*) ⇒ (∀*a*∈*set al*. *aEval a xs*) ∧ (∀*f*∈*set fl*. *eval f xs*))
have *h1*:(∃*a*∈*set* (*dnf* (*And* φ*1* φ*2*)). *f a*) = (∃*a*∈*set* (*dnf* φ*1*). ∃*a'*∈*set*(*dnf* φ*2*). *f a* ∧ *f a'*)
apply *simp* **unfolding** *f-def* **apply** *auto*
apply *blast*
apply *blast*
subgoal for *a b c d*
apply(*rule bexI*[**where** *x*=(*a*,*b*)])
apply(*rule exI*[**where** *x*=*a*@*c*])

apply(*rule exI*[**where** *x*=*b*@*d*])
by *auto*
done
also have *h2* : ... = ((∃ *a*∈*set* (*dnf φ1*). *f a*)∧(∃ *a*∈*set*(*dnf φ2*). *f a*))
by *auto*
show *?case* **unfolding** *f-def*[*symmetric*] **unfolding** *h1 h2* **unfolding** *f-def* **using** *And* **by** *auto*
qed *auto*

theorem *dnf-modified-eval* :
(∃ (*al*,*fl*,*n*)∈*set* (*dnf-modified φ*).
(∃ *L*. (*length L* = *n*
∧ (∀ *a*∈*set al*. *aEval a* (*L*@*xs*))
∧ (∀ *f*∈*set fl*. *eval f* (*L*@*xs*))))) = *eval φ xs*
proof(*induction φ arbitrary*:*xs*)
case (*Atom x*)
then show *?case*
by (*cases x*, *auto*)
next
case (*And φ1 φ2*)
{fix *d1 d2 A A′ B B′ L1 L2*
assume *A* : (*A*,*A′*,*length* (*L1*::*real list*))∈*set* (*dnf-modified φ1*) **and** (*B*,*B′*,*length* (*L2*::*real list*))∈*set* (*dnf-modified φ2*)
have (
(∀ *a*∈*set* ((*map* (*liftAtom* (*length L1*) (*length L2*)) *A* @ *map* (*liftAtom 0* (*length L1*)) *B*)). *aEval a* ((*L1*@*L2*) @ *xs*))
∧ (∀ *f*∈*set* (*map* (*liftFm* (*length L1*) (*length L2*)) *A′* @ *map* (*liftFm 0* (*length L1*)) *B′*). *eval f* ((*L1*@*L2*) @ *xs*))) =
(
(∀ *a*∈*set*(*map* (*liftAtom* (*length L1*) (*length L2*)) *A*) ∪ *set*(*map* (*liftAtom 0* (*length L1*)) *B*). *aEval a* ((*L1*@*L2*) @ *xs*))
∧ (∀ *f*∈*set*(*map* (*liftFm* (*length L1*) (*length L2*)) *A′*) ∪ *set*(*map* (*liftFm 0* (*length L1*)) *B′*). *eval f* ((*L1*@*L2*) @ *xs*)))
by *auto*
also have ... = (
(∀ *a*∈*set*(*map* (*liftAtom* (*length L1*) (*length L2*)) *A*).*aEval a* ((*L1*@*L2*) @ *xs*))
∧ (∀ *a*∈*set*(*map* (*liftAtom 0* (*length L1*)) *B*). *aEval a* ((*L1*@*L2*) @ *xs*))
∧ (∀ *f*∈*set*(*map* (*liftFm* (*length L1*) (*length L2*)) *A′*).*eval f* ((*L1*@*L2*) @ *xs*))
∧ (∀ *f*∈*set*(*map* (*liftFm 0* (*length L1*)) *B′*). *eval f* ((*L1*@*L2*) @ *xs*)))
by *blast*
also have ... = (
(∀ *a*∈*set A*. *aEval* (*liftAtom* (*length L1*) (*length L2*) *a*) ((*L1*@*L2*) @ *xs*))
∧ (∀ *a*∈*set B*. *aEval* (*liftAtom 0* (*length L1*) *a*) ((*L1*@*L2*) @ *xs*))
∧ (∀ *f*∈*set A′*. *eval* (*liftFm* (*length L1*) (*length L2*) *f*) ((*L1*@*L2*) @ *xs*))
∧ (∀ *f*∈*set B′*. *eval* (*liftFm 0* (*length L1*) *f*) ((*L1*@*L2*) @ *xs*)))
by *simp*
also have ... = (
(∀ *a*∈*set A*. *aEval* (*liftAtom* (*length L1*) (*length L2*) *a*) (*insert-into* (*L1* @ *xs*)

(*length L1*) *L2*))
∧ (∀ *a*∈*set B*. *aEval* (*liftAtom 0* (*length L1*) *a*) (*insert-into* (*L2* @ *xs*) *0 L1*))
∧ (∀ *f*∈*set A′*. *eval* (*liftFm* (*length L1*) (*length L2*) *f*) (*insert-into* (*L1* @ *xs*) (*length L1*) *L2*))
∧ (∀ *f*∈*set B′*. *eval* (*liftFm 0* (*length L1*) *f*) (*insert-into* (*L2* @ *xs*) *0 L1*)))
by *auto*
also have ... = (
((∀ *a*∈*set A*. *aEval a* (*L1* @ *xs*)) ∧ (∀ *f*∈*set A′*. *eval f* (*L1* @ *xs*))) ∧
((∀ *a*∈*set B*. *aEval a* (*L2* @ *xs*)) ∧ (∀ *f*∈*set B′*. *eval f* (*L2* @ *xs*))))
proof *safe*
fix *a*
show ∀ *a*∈*set A*. *aEval* (*liftAtom* (*length L1*) (*length L2*) *a*) (*insert-into* (*L1* @ *xs*) (*length L1*) *L2*) ⟹
a ∈ *set A* ⟹ *aEval a* (*L1* @ *xs*)
using *eval-liftFm*[*of L2 length L2 length L1 L1* @ *xs Atom a*, *OF refl*]
by *auto*
next
fix *f*
show ∀ *f*∈*set A′*. *eval* (*liftFm* (*length L1*) (*length L2*) *f*) (*insert-into* (*L1* @ *xs*) (*length L1*) *L2*) ⟹
f ∈ *set A′* ⟹ *eval f* (*L1* @ *xs*)
using *eval-liftFm*[*of L2 length L2 length L1 L1* @ *xs f*, *OF refl*] **by** *auto*
next
fix *a*
show ∀ *a*∈*set B*. *aEval* (*liftAtom 0* (*length L1*) *a*) (*insert-into* (*L2* @ *xs*) *0 L1*) ⟹
a ∈ *set B* ⟹ *aEval a* (*L2* @ *xs*)
using *eval-liftFm*[*of L1 length L1 0 L2*@*xs Atom a*, *OF refl*] **by** *auto*
next
fix *f*
show ∀ *f*∈*set B′*. *eval* (*liftFm 0* (*length L1*) *f*) (*insert-into* (*L2* @ *xs*) *0 L1*) ⟹ *f* ∈ *set B′* ⟹ *eval f* (*L2* @ *xs*)
using *eval-liftFm*[*of L1 length L1 0 L2* @ *xs f*, *OF refl*] **by** *auto*
next
fix *a*
show ∀ *a*∈*set A*. *aEval a* (*L1* @ *xs*) ⟹
a ∈ *set A* ⟹ *aEval* (*liftAtom* (*length L1*) (*length L2*) *a*) (*insert-into* (*L1* @ *xs*) (*length L1*) *L2*)
using *eval-liftFm*[*of L2 length L2 length L1 L1* @ *xs Atom a*, *OF refl*] **by** *auto*
next
fix *a*
show ∀ *a*∈*set B*. *aEval a* (*L2* @ *xs*) ⟹ *a* ∈ *set B* ⟹ *aEval* (*liftAtom 0* (*length L1*) *a*) (*insert-into* (*L2* @ *xs*) *0 L1*)
using *eval-liftFm*[*of L1 length L1 0 L2*@*xs Atom a*, *OF refl*] **by** *auto*
next
fix *f*
show ∀ *f*∈*set A′*. *eval f* (*L1* @ *xs*) ⟹
f ∈ *set A′* ⟹ *eval* (*liftFm* (*length L1*) (*length L2*) *f*) (*insert-into* (*L1* @

xs) (*length L1*) *L2*)
using *eval-liftFm*[*of L2 length L2 length L1 L1 @ xs f*, *OF refl*] **by** *auto*
next
fix *f*
show $\forall f{\in}set\ B'.\ eval\ f\ (L2\ @\ xs) \Longrightarrow f \in set\ B' \Longrightarrow eval\ (liftFm\ 0\ (length\ L1)\ f)\ (insert\text{-}into\ (L2\ @\ xs)\ 0\ L1)$
using *eval-liftFm*[*of L1 length L1 0 L2 @ xs f*, *OF refl*] **by** *auto*
qed
finally have (
$(\forall a{\in}set\ ((map\ (liftAtom\ (length\ L1)\ (length\ L2))\ A\ @\ map\ (liftAtom\ 0\ (length\ L1))\ B)).\ aEval\ a\ ((L1@L2)\ @\ xs))$
$\wedge\ (\forall f{\in}set\ (\ map\ (liftFm\ (length\ L1)\ (length\ L2))\ A'\ @\ map\ (liftFm\ 0\ (length\ L1))\ B').\ eval\ f\ ((L1@L2)\ @\ xs))) = ($
$((\forall a{\in}set\ A.\ aEval\ a\ (L1\ @\ xs)) \wedge (\forall f{\in}set\ A'.\ eval\ f\ (L1\ @\ xs))) \wedge$
$((\forall a{\in}set\ B.\ aEval\ a\ (L2\ @\ xs)) \wedge (\forall f{\in}set\ B'.\ eval\ f\ (L2\ @\ xs))))$
by *simp*
}
then have $h : (\exists (A,A',d1){\in}set\ (dnf\text{-}modified\ \varphi 1).\ \exists (B,B',d2){\in}set\ (dnf\text{-}modified\ \varphi 2).$
$(\exists L1.\exists L2.\ length\ L1 = d1 \wedge length\ L2 = d2 \wedge$
$(\forall a{\in}set\ ((map\ (liftAtom\ d1\ d2)\ A\ @\ map\ (liftAtom\ 0\ d1)\ B)).\ aEval\ a\ ((L1@L2)\ @\ xs))$
$\wedge\ (\forall f{\in}set\ (\ map\ (liftFm\ d1\ d2)\ A'\ @\ map\ (liftFm\ 0\ d1)\ B').\ eval\ f\ ((L1@L2)\ @\ xs)))) = ((\exists (A,A',d1){\in}set\ (dnf\text{-}modified\ \varphi 1).\ \exists (B,B',d2){\in}set(dnf\text{-}modified\ \varphi 2).$
$(\exists L1.\ length\ L1 = d1 \wedge (\forall a{\in}set\ A.\ aEval\ a\ (L1\ @\ xs)) \wedge (\forall f{\in}set\ A'.\ eval\ f\ (L1\ @\ xs))) \wedge$
$(\exists L2.\ length\ L2 = d2 \wedge (\forall a{\in}set\ B.\ aEval\ a\ (L2\ @\ xs)) \wedge (\forall f{\in}set\ B'.\ eval\ f\ (L2\ @\ xs))))\ ))$
proof *safe*
fix *A A' B B' L1 L2*
assume *prev* : $(\bigwedge A\ A'\ L1\ B\ B'\ L2.$
$(A,\ A',\ length\ L1) \in set\ (dnf\text{-}modified\ \varphi 1) \Longrightarrow$
$(B,\ B',\ length\ L2) \in set\ (dnf\text{-}modified\ \varphi 2) \Longrightarrow$
$((\forall a{\in}set\ (map\ (liftAtom\ (length\ L1)\ (length\ L2))\ A\ @\ map\ (liftAtom\ 0\ (length\ L1))\ B).$
$aEval\ a\ ((L1\ @\ L2)\ @\ xs)) \wedge$
$(\forall f{\in}set\ (map\ (liftFm\ (length\ L1)\ (length\ L2))\ A'\ @\ map\ (liftFm\ 0\ (length\ L1))\ B').$
$eval\ f\ ((L1\ @\ L2)\ @\ xs))) =$
$(((\forall a{\in}set\ A.\ aEval\ a\ (L1\ @\ xs)) \wedge (\forall f{\in}set\ A'.\ eval\ f\ (L1\ @\ xs))) \wedge$
$(\forall a{\in}set\ B.\ aEval\ a\ (L2\ @\ xs)) \wedge (\forall f{\in}set\ B'.\ eval\ f\ (L2\ @\ xs))))$
and *A*: $(A,A',length\ L1){\in}set\ (dnf\text{-}modified\ \varphi 1)$ **and** *B*: $(B,B',length\ L2){\in}set\ (dnf\text{-}modified\ \varphi 2)$
and *h1* : $\forall a{\in}set\ (map\ (liftAtom\ (length\ L1)\ (length\ L2))\ A\ @\ map\ (liftAtom\ 0\ (length\ L1))\ B).$
$aEval\ a\ ((L1\ @\ L2)\ @\ xs)$
and *h2* : $\forall f{\in}set\ (map\ (liftFm\ (length\ L1)\ (length\ L2))\ A'\ @\ map\ (liftFm\ 0\ (length\ L1))\ B').$
$eval\ f\ ((L1\ @\ L2)\ @\ xs)$

have h : (($\forall$ a∈*set* (*map* (*liftAtom* (*length L1*) (*length L2*)) A @ *map* (*liftAtom* 0 (*length L1*)) B).
aEval a ((*L1* @ *L2*) @ *xs*)) ∧
($\forall$ f∈*set* (*map* (*liftFm* (*length L1*) (*length L2*)) A' @ *map* (*liftFm* 0 (*length L1*)) B').
eval f ((*L1* @ *L2*) @ *xs*))) =
((($\forall$ a∈*set A*. *aEval a* (*L1* @ *xs*)) ∧ ($\forall$ f∈*set* A'. *eval f* (*L1* @ *xs*))) ∧
($\forall$ a∈*set B*. *aEval a* (*L2* @ *xs*)) ∧ ($\forall$ f∈*set* B'. *eval f* (*L2* @ *xs*)))
using *prev*[**where** A=A, **where** A'=A', **where** B=B, **where** B'=B'] *A B* **by** *simp*
show ∃ (A, A', *d1*)∈*set* (*dnf-modified* φ*1*).
∃ (B, B', *d2*)∈*set* (*dnf-modified* φ*2*).
(∃ *L1*. *length L1* = *d1* ∧ ($\forall$ a∈*set A*. *aEval a* (*L1* @ *xs*)) ∧ ($\forall$ f∈*set* A'. *eval f* (*L1* @ *xs*))) ∧
(∃ *L2*. *length L2* = *d2* ∧ ($\forall$ a∈*set B*. *aEval a* (*L2* @ *xs*)) ∧ ($\forall$ f∈*set* B'. *eval f* (*L2* @ *xs*)))
apply (*rule bexI*[**where** x=(A, A', *length L1*), *OF* - A])
apply *auto* **defer**
apply (*rule bexI*[**where** x=(B, B', *length L2*), *OF* - B])
apply *auto*
using *h h1 h2*
by *auto*
next
fix *A A' d1 B B' d2 L1 L2*
assume *prev* : ($\bigwedge$*A A' L1 B B' L2*.
(*A*, *A'*, *length L1*) ∈ *set* (*dnf-modified* φ*1*) ⟹
(*B*, *B'*, *length L2*) ∈ *set* (*dnf-modified* φ*2*) ⟹
(($\forall$ a∈*set* (*map* (*liftAtom* (*length L1*) (*length L2*)) *A* @ *map* (*liftAtom* 0 (*length L1*)) *B*).
aEval a ((*L1* @ *L2*) @ *xs*)) ∧
($\forall$ f∈*set* (*map* (*liftFm* (*length L1*) (*length L2*)) A' @ *map* (*liftFm* 0 (*length L1*)) B').
eval f ((*L1* @ *L2*) @ *xs*))) =
((($\forall$ a∈*set A*. *aEval a* (*L1* @ *xs*)) ∧ ($\forall$ f∈*set* A'. *eval f* (*L1* @ *xs*))) ∧
($\forall$ a∈*set B*. *aEval a* (*L2* @ *xs*)) ∧ ($\forall$ f∈*set* B'. *eval f* (*L2* @ *xs*))))
and *A*: (*A*,*A'*,*length L1*)∈*set* (*dnf-modified* φ*1*) **and** *B*: (*B*,*B'*,*length L2*)∈*set* (*dnf-modified* φ*2*)
and *h1* : $\forall$ a∈*set A*. *aEval a* (*L1* @ *xs*) $\forall$ f∈*set* A'. *eval f* (*L1* @ *xs*)
$\forall$ a∈*set B*. *aEval a* (*L2* @ *xs*) $\forall$ f∈*set* B'. *eval f* (*L2* @ *xs*)
have h : (($\forall$ a∈*set* (*map* (*liftAtom* (*length L1*) (*length L2*)) A @ *map* (*liftAtom* 0 (*length L1*)) B).
aEval a ((*L1* @ *L2*) @ *xs*)) ∧
($\forall$ f∈*set* (*map* (*liftFm* (*length L1*) (*length L2*)) A' @ *map* (*liftFm* 0 (*length L1*)) B').
eval f ((*L1* @ *L2*) @ *xs*))) =
((($\forall$ a∈*set A*. *aEval a* (*L1* @ *xs*)) ∧ ($\forall$ f∈*set* A'. *eval f* (*L1* @ *xs*))) ∧
($\forall$ a∈*set B*. *aEval a* (*L2* @ *xs*)) ∧ ($\forall$ f∈*set* B'. *eval f* (*L2* @ *xs*)))
using *prev*[**where** A=A, **where** A'=A', **where** B=B, **where** B'=B'] *h1 A B* **by** *simp*

show $\exists (A,\ A',\ d1) \in set\ (dnf\text{-}modified\ \varphi 1).$
$\exists (B,\ B',\ d2) \in set\ (dnf\text{-}modified\ \varphi 2).$
$\exists\, L1\ L2.$
$length\ L1 = d1\ \wedge$
$length\ L2 = d2\ \wedge$
$(\forall\, a \in set\ (map\ (liftAtom\ d1\ d2)\ A\ @\ map\ (liftAtom\ 0\ d1)\ B).\ aEval\ a\ ((L1\ @\ L2)\ @\ xs))\ \wedge$
$(\forall\, f \in set\ (map\ (liftFm\ d1\ d2)\ A'\ @\ map\ (liftFm\ 0\ d1)\ B').\ eval\ f\ ((L1\ @\ L2)\ @\ xs))$
apply (*rule bexI*[**where** $x=(A,\ A',\ length\ L1)$, *OF - A*])
apply *auto* **defer**
apply (*rule bexI*[**where** $x=(B,\ B',\ length\ L2)$, *OF - B*])
apply *auto*
apply (*rule exI*[**where** $x=L1$])
apply *auto*
apply (*rule exI*[**where** $x=L2$])
apply *auto*
using *h h1* **by** *auto*
qed

define f **where** $f\ (x::\ (atom\ list * atom\ fm\ list * nat)) = (case\ x\ of\ (al,fl,n) \Rightarrow (\exists\, L.\ length\ L = n\ \wedge\ (\forall\, a \in set\ al.\ aEval\ a\ (L\ @\ xs))\ \wedge\ (\forall\, f \in set\ fl.\ eval\ f\ (L\ @\ xs))))$ **for** x
define g **where** $((g\ (uuaa::atom\ list)\ (uua::atom\ fm\ list)\ (uu::nat)\ x)::\ (atom\ list * atom\ fm\ list * nat)) = ($
case x of
$(B,\ B',\ d2) \Rightarrow$
$(map\ (liftAtom\ uu\ d2)\ uuaa\ @\ map\ (liftAtom\ 0\ uu)\ B,$
$map\ (\lambda x.\ map\text{-}fm\text{-}binders\ (\lambda a\ x.\ liftAtom\ (uu + x)\ d2\ a)\ x\ 0)\ uua\ @$
$map\ (\lambda x.\ map\text{-}fm\text{-}binders\ (\lambda a\ x.\ liftAtom\ (0 + x)\ uu\ a)\ x\ 0)\ B',$
$uu + d2))$ **for** *uuaa uua uu x*

define f' **where** $f'\ L\ A\ A'\ d1\ B\ B'\ d2 = ((\forall\, a \in set\ ((map\ (liftAtom\ d1\ d2)\ A\ @\ map\ (liftAtom\ 0\ d1)\ B)).\ aEval\ a\ (L\ @\ xs))$
$\wedge\ (\forall\, f \in set\ (\ map\ (liftFm\ d1\ d2)\ A'\ @\ map\ (liftFm\ 0\ d1)\ B').\ eval\ f\ (L\ @\ xs)))$
for *L A A' d1 B B' d2*
have $(\exists (al,\ fl,\ n) \in set\ (dnf\text{-}modified\ (And\ \varphi 1\ \varphi 2)).$
$\exists\, L.\ length\ L = n\ \wedge\ (\forall\, a \in set\ al.\ aEval\ a\ (L\ @\ xs))\ \wedge\ (\forall\, f \in set\ fl.\ eval\ f\ (L\ @\ xs)))$
$= (\exists\, y \in set\ (dnf\text{-}modified\ (And\ \varphi 1\ \varphi 2)).\ f\ y)$
unfolding *f-def* **by** *simp*
also have ... $= (\exists\, y \in set\ (dnf\text{-}modified\ \varphi 1).$
$\exists\, a\ aa\ b.$
$(\exists\, uu\ uua\ uuaa.$
$(uuaa,\ uua,\ uu) = y\ \wedge$
$(a,\ aa,\ b)$
$\in (g\ uuaa\ uua\ uu)$ ‘

set (*dnf-modified* $\varphi 2$)) $\wedge$
f (*a*, *aa*, *b*))
using *g-def* **by** *simp*
also have ... = ($\exists$ (*A*,*A'*,*d1*)$\in$*set* (*dnf-modified* $\varphi 1$).
$\exists$ *x*$\in$*set* (*dnf-modified* $\varphi 2$).
f (*g A A' d1 x*))
by (*smt case-prodE f-def imageE image-eqI prod.simps*(2))
also have ... = ($\exists$ (*A*,*A'*,*d1*)$\in$*set* (*dnf-modified* $\varphi 1$).
$\exists$ *x*$\in$*set* (*dnf-modified* $\varphi 2$).
f (*case x of* (*B*,*B'*,*d2*) $\Rightarrow$ (*map* (*liftAtom d1 d2*) *A* @ *map* (*liftAtom 0 d1*) *B*,
map (λx. *liftFm d1 d2 x*) *A'* @
map (λx. *liftFm 0 d1 x*) *B'*,
d1 + *d2*)))
using *g-def* **by** *simp*
also have ... = ($\exists$ (*A*,*A'*,*d1*)$\in$*set* (*dnf-modified* $\varphi 1$). $\exists$ *x*$\in$*set* (*dnf-modified* $\varphi 2$).
(*case* (*case x of* (*B*,*B'*,*d2*) $\Rightarrow$ (*map* (*liftAtom d1 d2*) *A* @ *map* (*liftAtom 0 d1*) *B*,
map (λx. *liftFm d1 d2 x*) *A'* @ *map* (λx. *liftFm 0 d1 x*) *B'*,
d1 + *d2*)) *of* (*al*,*fl*,*n*) $\Rightarrow$ ($\exists$ *L*. *length L* = *n* $\wedge$ ($\forall$ *a*$\in$*set al*. *aEval a* (*L* @ *xs*)) $\wedge$ ($\forall$ *f*$\in$*set fl*. *eval f* (*L* @ *xs*))))
)
using *f-def* **by** *simp*
also have ... = ($\exists$ (*A*,*A'*,*d1*)$\in$*set* (*dnf-modified* $\varphi 1$). $\exists$ (*B*,*B'*,*d2*)$\in$*set* (*dnf-modified* $\varphi 2$).
(*case* ((*map* (*liftAtom d1 d2*) *A* @ *map* (*liftAtom 0 d1*) *B*,
map (λx. *liftFm d1 d2 x*) *A'* @ *map* (λx. *liftFm 0 d1 x*) *B'*,
d1 + *d2*)) *of* (*al*,*fl*,*n*) $\Rightarrow$ ($\exists$ *L*. *length L* = *n* $\wedge$ ($\forall$ *a*$\in$*set al*. *aEval a* (*L* @ *xs*)) $\wedge$ ($\forall$ *f*$\in$*set fl*. *eval f* (*L* @ *xs*))))
)
by(*smt case-prodE case-prodE2 old.prod.case*)
also have ... = ($\exists$ (*A*,*A'*,*d1*)$\in$*set* (*dnf-modified* $\varphi 1$). $\exists$ (*B*,*B'*,*d2*)$\in$*set* (*dnf-modified* $\varphi 2$).
($\exists$ *L*. *length L* = *d1* + *d2* $\wedge$
($\forall$ *a*$\in$*set* ((*map* (*liftAtom d1 d2*) *A* @ *map* (*liftAtom 0 d1*) *B*)). *aEval a* (*L* @ *xs*))
$\wedge$ ($\forall$ *f*$\in$*set* (*map* (*liftFm d1 d2*) *A'* @ *map* (*liftFm 0 d1*) *B'*). *eval f* (*L* @ *xs*))))

by(*smt case-prodE case-prodE2 old.prod.case*)
also have ... = ($\exists$ (*A*,*A'*,*d1*)$\in$*set* (*dnf-modified* $\varphi 1$). $\exists$ (*B*,*B'*,*d2*)$\in$*set* (*dnf-modified* $\varphi 2$).
($\exists$ *L*. *length L* = *d1* + *d2* $\wedge$ (*f'* *L A A' d1 B B' d2*)))
using *f'-def* **by** *simp*
also have ... = ($\exists$ (*A*,*A'*,*d1*)$\in$*set* (*dnf-modified* $\varphi 1$). $\exists$ (*B*,*B'*,*d2*)$\in$*set* (*dnf-modified* $\varphi 2$).
($\exists$ *L1*.$\exists$ *L2*. *length L1* = *d1* $\wedge$ *length L2* = *d2* $\wedge$ (*f'* (*L1*@*L2*) *A A' d1 B B' d2*)))
proof *safe*
fix *A A' d1 B B' d2 L*

assume A: $(A,A',d1)\in set\ (dnf\text{-}modified\ \varphi 1)$ **and** B: $(B,B',d2)\in set\ (dnf\text{-}modified\ \varphi 2)$
and L: $length\ L = d1 + d2\ (f'\ L\ A\ A'\ d1\ B\ B'\ d2)$
show $\exists (A,\ A',\ d1)\in set\ (dnf\text{-}modified\ \varphi 1).$
$\exists (B,\ B',\ d2)\in set\ (dnf\text{-}modified\ \varphi 2).\ \exists\, L1\ L2.\ length\ L1 = d1 \wedge length\ L2 = d2 \wedge f'\ (L1\ @\ L2)\ A\ A'\ d1\ B\ B'\ d2$
apply $(rule\ bexI[\textbf{where}\ x=(A,\ A',\ d1),\ OF\ -\ A])$
apply $auto$
apply $(rule\ bexI[\textbf{where}\ x=(B,\ B',\ d2),\ OF\ -\ B])$
apply $auto$
apply $(rule\ exI[\textbf{where}\ x=take\ d1\ L])$
apply $auto$ **defer**
apply $(rule\ exI[\textbf{where}\ x=drop\ d1\ L])$
using L
by $auto$
next
fix $A\ A'\ d1\ B\ B'\ d2\ L1\ L2$
assume A: $(A,A',\ length\ L1)\in set\ (dnf\text{-}modified\ \varphi 1)$ **and** B: $(B,B',length\ L2)\in set\ (dnf\text{-}modified\ \varphi 2)$
and L: $(f'\ (L1\ @\ L2)\ A\ A'\ (length\ L1)\ B\ B'\ (length\ L2))$
thm exI
thm $bexI$
show $\exists (A,\ A',\ d1)\in set\ (dnf\text{-}modified\ \varphi 1).\ \exists (B,\ B',\ d2)\in set\ (dnf\text{-}modified\ \varphi 2).\ \exists\, L.\ length\ L = d1 + d2 \wedge f'\ L\ A\ A'\ d1\ B\ B'\ d2$
apply $(rule\ bexI[\textbf{where}\ x=(A,\ A',\ length\ L1),\ OF\ -\ A])$
apply $auto$
apply $(rule\ bexI[\textbf{where}\ x=(B,\ B',\ length\ L2),\ OF\ -\ B])$
apply $auto$
apply $(rule\ exI[\textbf{where}\ x=L1\ @\ L2])$
using L
by $auto$
qed

also have $... = (\exists (A,A',d1)\in set\ (dnf\text{-}modified\ \varphi 1).\ \exists (B,B',d2)\in set\ (dnf\text{-}modified\ \varphi 2).$
$(\exists\, L1.\exists\, L2.\ length\ L1 = d1 \wedge length\ L2 = d2 \wedge$
$(\forall\, a\in set\ ((map\ (liftAtom\ d1\ d2)\ A\ @\ map\ (liftAtom\ 0\ d1)\ B)).\ aEval\ a\ ((L1@L2)\ @\ xs))$
$\wedge\ (\forall f\in set\ (\ map\ (liftFm\ d1\ d2)\ A'\ @\ map\ (liftFm\ 0\ d1)\ B').\ eval\ f\ ((L1@L2)\ @\ xs))))$
unfolding $f'\text{-}def$ **by** $simp$

also have $... = ((\exists (A,A',d1)\in set\ (dnf\text{-}modified\ \varphi 1).\ \exists (B,B',d2)\in set(dnf\text{-}modified\ \varphi 2).$
$(\exists\, L.\ length\ L = d1 \wedge (\forall\, a\in set\ A.\ aEval\ a\ (L\ @\ xs)) \wedge (\forall f\in set\ A'.\ eval\ f\ (L\ @\ xs))) \wedge$
$(\exists\, L.\ length\ L = d2 \wedge (\forall\, a\in set\ B.\ aEval\ a\ (L\ @\ xs)) \wedge (\forall f\in set\ B'.\ eval\ f\ (L\ @\ xs))))$
using h **by** $simp$

also have ... = ((∃(A,A′,d1)∈*set* (*dnf-modified* φ1). ∃(B,B′,d2)∈*set*(*dnf-modified* φ2).
f (A,A′,d1) ∧
f (B,B′,d2)))
using *f-def* **by** *simp*
also have ... = ((∃ a∈*set* (*dnf-modified* φ1). ∃ a1∈*set*(*dnf-modified* φ2). *f a* ∧ *f* a1))
by (*simp add*: *Bex-def-raw*)
also have ... = ((∃ a∈*set* (*dnf-modified* φ1). *f a*) ∧ (∃ a∈*set* (*dnf-modified* φ2). *f a*))
by *blast*
also have ... = *eval* (*And* φ1 φ2) *xs*
using *And f-def* **by** *simp*
finally have (∃(*al*, *fl*, *n*)∈*set* (*dnf-modified* (*And* φ1 φ2)).
∃ *L*. *length L* = *n* ∧ (∀ a∈*set al*. *aEval a* (*L* @ *xs*)) ∧ (∀ *f*∈*set fl*. *eval f* (*L* @ *xs*))) =
eval (*And* φ1 φ2) *xs*
by *simp*
then show *?case* **by** *simp*
next
case (*Or* φ1 φ2)
have *h1* : *eval* (*Or* φ1 φ2) *xs* = *eval* φ1 *xs* ∨ *eval* φ2 *xs*
using *eval.simps*(*5*) **by** *blast*
have *h2* : *set* (*dnf-modified* (*Or* φ1 φ2)) = *set*(*dnf-modified* φ1) ∪ *set*(*dnf-modified* φ2)
by *simp*
show *?case* **using** *Or h1 h2*
by (*metis* (*no-types*, *lifting*) *Un-iff eval.simps*(*5*))
next
case (*ExQ* φ)
have *h1* : ((∃ *x*. (∃(*al*, *fl*, *n*)∈*set* (*dnf-modified* φ).
∃ *L*. *length L* = *n* ∧ (∀ a∈*set al*. *aEval a* (*L* @ (*x*#*xs*))) ∧ (∀ *f*∈*set fl*. *eval f* (*L* @ (*x*#*xs*)))))
=
(∃(*al*, *fl*, *n*)∈*set* (*dnf-modified* φ).
(∃ *x*.∃ *L*. *length L* = *n* ∧ (∀ a∈*set al*. *aEval a* ((*L*@[*x*]) @ *xs*)) ∧ (∀ *f*∈*set fl*. *eval f* ((*L*@[*x*]) @ *xs*)))))
apply *simp* **by** *blast*
{ fix *n al fl*
define *f* **where** *f L* = (*length* (*L*:: *real list*) = ((*n*::*nat*)+*1*) ∧ (∀ a∈*set al*. *aEval a* (*L* @ *xs*)) ∧ (∀ *f*∈*set fl*. *eval f* (*L* @ *xs*))) **for** *L*
have (∃ *x*.∃ *L*. *f* (*L*@[*x*])) = (∃ *L*. *f L*)
by (*metis* (*full-types*) *One-nat-def add-Suc-right f-def list.size*(*3*) *nat.simps*(*3*) *rev-exhaust*)
then have ((∃ *x*. ∃ *L*. *length* (*L*@[*x*]) = (*n*+*1*) ∧ (∀ a∈*set al*. *aEval a* ((*L*@[*x*]) @ *xs*)) ∧ (∀ *f*∈*set fl*. *eval f* ((*L*@[*x*]) @ *xs*)))
=
(∃ *L*. *length L* = (*n*+*1*) ∧ (∀ a∈*set al*. *aEval a* (*L* @ *xs*)) ∧ (∀ *f*∈*set fl*. *eval f* (*L* @ *xs*))))

```
    unfolding f-def by simp
  }
  then have h2 : ∀ n al fl. (
            (∃ x. ∃ L. length (L@[x]) = (n+1) ∧ (∀ a∈set al. aEval a ((L@[x]) @
xs)) ∧ (∀ f∈set fl. eval f ((L@[x]) @ xs)))
            =
            (∃ L. length L = (n+1) ∧ (∀ a∈set al. aEval a (L @ xs)) ∧ (∀ f∈set fl.
eval f (L @ xs)))
          )
    by simp
  { fix al fl n
    define f where f al fl n = (∃ L. length L = n ∧ (∀ a∈set al. aEval a (L @ xs))
∧ (∀ f∈set fl. eval f (L @ xs))) for al fl n
    have f al fl (n+1) = (case (case (al, fl, n) of (A, A', d) ⇒ (A, A',d+1)) of
          (al, fl, n) ⇒ f al fl n)
      by simp
    then have (∃ L. length L = (n+1) ∧ (∀ a∈set al. aEval a (L @ xs)) ∧ (∀ f∈set
fl. eval f (L @ xs)))
            = (
            case (case (al, fl, n) of (A, A', d) ⇒ (A, A',d+1)) of
            (al, fl, n) ⇒
             ∃ L. length L = n ∧ (∀ a∈set al. aEval a (L @ xs)) ∧ (∀ f∈set fl. eval
f (L @ xs)))
      unfolding f-def by simp
  }
  then have h3 :
            (∃ (al, fl, n)∈set (dnf-modified φ).
             ∃ L. length L = (n+1) ∧ (∀ a∈set al. aEval a (L @ xs)) ∧ (∀ f∈set fl.
eval f (L @ xs)))
            = (∃ a∈set (dnf-modified φ).
            case (case a of (A, A', d) ⇒ (A, A',d+1)) of
            (al, fl, n) ⇒
             ∃ L. length L = n ∧ (∀ a∈set al. aEval a (L @ xs)) ∧ (∀ f∈set fl. eval
f (L @ xs)))
    by (smt case-prodE case-prodI2)
  show ?case using ExQ h1 h2 h3 by simp
next
  case (ExN x1 φ)

  show ?case
    apply simp proof safe
    fix a aa b L
    have takedrop: (take b L @ drop b L @ xs) = (L @ xs) by auto
   assume h: (a, aa, b) ∈ set (dnf-modified φ) length L = b + x1 ∀ a∈set a. aEval
a (L @ xs) ∀ f∈set aa. eval f (L @ xs)
    show ∃ l. length l = x1 ∧ eval φ (l @ xs)
      apply(rule exI[where x=drop b L])
      apply auto
      using h(2) apply simp
```

unfolding *ExN*[*symmetric*]
apply(*rule bexI*[**where** *x*=(*a*,*aa*,*b*)])
apply *simp*
apply(*rule exI*[**where** *x*=*take b L*])
apply *auto*
using *h* **apply** *simp*
unfolding *takedrop*
using *h* **by** *auto*
next
fix *l*
assume *h*: *eval* φ (*l* @ *xs*) *x1* = *length l*
obtain *al fl n* **where** *h1* : (*al*, *fl*, *n*)∈*set* (*dnf-modified* φ) ∃ *L*. *length L* = *n* ∧ (∀ *a*∈*set al*. *aEval a* (*L* @ *l* @ *xs*)) ∧ (∀ *f*∈*set fl*. *eval f* (*L* @ *l* @ *xs*))
using *h*(*1*) **unfolding** *ExN*[*symmetric*]
by *blast*
obtain *L* **where** *h2* : *length L* = *n* (∀ *a*∈*set al*. *aEval a* (*L* @ *l* @ *xs*)) (∀ *f*∈*set fl*. *eval f* (*L* @ *l* @ *xs*)) **using** *h1*(*2*) **by** *metis*
show ∃ *x*∈*set* (*dnf-modified* φ).
case case x of (*A*, *A'*, *d*) ⇒ (*A*, *A'*, *d* + *length l*) *of*
(*al*, *fl*, *n*) ⇒ ∃ *L*. *length L* = *n* ∧ (∀ *a*∈*set al*. *aEval a* (*L* @ *xs*)) ∧ (∀ *f*∈*set fl*. *eval f* (*L* @ *xs*))
apply(*rule bexI*[**where** *x*=(*al*,*fl*,*n*)])
apply *simp*
apply(*rule exI*[**where** *x*=*L*@*l*])
apply *auto*
using *h2 h1* **by** *auto*
qed
qed *auto*
end

10.2 Recursive QE

theory *VSQuad*
imports *EqualityVS GeneralVSProofs Reindex OptimizationProofs DNF*
begin

lemma *existN-eval* : ∀ *xs*. *eval* (*ExN n* φ) *xs* = (∃ *L*. (*length L* = *n* ∧ *eval* φ (*L*@*xs*)))
proof(*induction n*)
case *0*
then show *?case* **by** *simp*
next
case (*Suc n*)
{fix *xs*
have *eval* (*ExN* (*Suc n*) φ) *xs* = (∃ *l*. *length l* = *Suc n* ∧ *eval* φ (*l* @ *xs*))
by *simp*
also have ... = (∃ *x*.∃ *L*. (*length L* = *n* ∧ *eval* φ (*L*@(*x*#*xs*))))
proof *safe*
fix *l*

```
    assume h : length l = Suc n eval φ (l @ xs)
    show ∃ x L. length L = n ∧ eval φ (L @ x # xs)
      apply(rule exI[where x=l ! n])
      apply(rule exI[where x=take n l])
      using h apply auto
     by (metis Cons-nth-drop-Suc append.assoc append-Cons append-take-drop-id
lessI order-refl self-append-conv self-append-conv2 take-all)
  next
    fix x L
    assume h : eval φ (L @ x # xs) n = length L
    show ∃ l. length l = Suc (length L) ∧ eval φ (l @ xs)
      apply(rule exI[where x=L@[x]])
      using h by auto
  qed
  also have ... = (∃ x.∃ L. (length L = n ∧ eval φ ((L@[x])@xs)))
    by simp
  also have ... = (∃ x.∃ L. (length (L@[x]) = (Suc n) ∧ eval φ ((L@[x])@xs)))
    by simp
  also have ... = (∃ L. (length L = (Suc n) ∧ eval φ (L@xs)))
    by (metis append-butlast-last-id length-0-conv nat.simps(3))
  finally have eval (ExN (Suc n) φ) xs = (∃ L. (length L = (Suc n) ∧ eval φ
(L@xs)))
    by simp
 }
 then show ?case by simp
qed



lemma boundedFlipNegQuantifier : (¬(∀ x∈A. ¬ P x)) = (∃ x∈A. P x)
 by blast


theorem QE-dnf'-eval:
 assumes steph : ⋀amount F Γ.
   (∃ 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 (step amount F) Γ)
 assumes opt : ⋀xs F . eval (opt F) xs = eval F xs
 shows eval (QE-dnf' opt step φ) xs = eval φ xs
proof(induction φ arbitrary : xs)
 case (Atom x)
 then show ?case by (simp add: simp-atom-eval)
next
 case (And φ1 φ2)
 then show ?case by (simp add: eval-and)
next
 case (Or φ1 φ2)
 then show ?case by (simp add: eval-or)
```

```
next
  case (Neg φ)
  then show ?case  apply simp
    by (metis  eval-neg )
next
  case (ExQ φ)
  have h1 : ⋀F. (∃ xs. length xs = Suc 0 ∧
        F xs) = (∃ x.
        F [x])
    by (metis length-0-conv length-Suc-conv)
  show ?case
    apply simp
    unfolding steph[symmetric] apply(simp add: eval-list-disj)
    unfolding h1 apply(rule ex-cong1)
    unfolding ExQ[symmetric]
    unfolding opt[symmetric, of (QE-dnf' opt step φ)]
    unfolding dnf-modified-eval[symmetric, of (opt (QE-dnf' opt step φ))]
    apply(rule bex-cong) apply simp
    subgoal for x f
      apply(cases f)
      apply (auto simp add:eval-list-conj)
      by (metis Un-iff eval.simps(1) imageI)
    done
next
  case (AllQ φ)
  have h1 : ⋀F. (∀ xs::real list. (length xs = Suc 0 ⟶
        F xs)) = (∀ x.
        F [x])
    by (metis length-0-conv length-Suc-conv)
  show ?case
    apply simp
    unfolding steph[symmetric] apply(simp add: eval-list-disj)
    unfolding h1 apply(rule all-cong1)
    unfolding AllQ[symmetric]
    unfolding eval-neg[symmetric, of (QE-dnf' opt step φ)]
    unfolding opt[symmetric, of neg(QE-dnf' opt step φ)]
    unfolding Set.bex-simps(8)[symmetric] HOL.Not-eq-iff
    unfolding dnf-modified-eval[symmetric, of (opt (neg(QE-dnf' opt step φ)))]
    apply(rule bex-cong) apply simp
    subgoal for x f
      apply(cases f)
      apply (auto simp add:eval-list-conj)
      by (metis Un-iff eval.simps(1) imageI)
    done
next
  case (ExN amount φ)
  show ?case
    apply(cases amount)
    apply (simp-all add: ExN)
```

```
    unfolding steph[symmetric] apply(simp add: eval-list-disj)
    unfolding ExN[symmetric]
    unfolding opt[of (QE-dnf' opt step φ),symmetric]
    unfolding dnf-modified-eval[of (opt (QE-dnf' opt step φ)),symmetric]
    apply(rule ex-cong1)
    subgoal for nat xs
      apply(cases length xs = Suc nat)
      apply simp-all
      apply(rule bex-cong)
      apply simp-all
      subgoal for f
        apply(cases f)
        apply simp
        apply(rule ex-cong1)
        unfolding eval-list-conj
        apply auto
        by (meson Un-iff eval.simps(1) imageI)
      done
    done
next
  case (AllN amount φ)
  show ?case
    apply(cases amount)
    apply (simp-all add: AllN)
    unfolding steph[symmetric] apply(simp add: eval-list-disj)
    unfolding AllN[symmetric]
    unfolding eval-neg[symmetric, of (QE-dnf' opt step φ)]
    unfolding opt[symmetric, of neg(QE-dnf' opt step φ)]
    unfolding Set.bex-simps(8)[symmetric]
    unfolding HOL.imp-conv-disj
    unfolding HOL.de-Morgan-conj[symmetric]
    unfolding HOL.not-ex[symmetric]
    unfolding  HOL.Not-eq-iff
    unfolding dnf-modified-eval[symmetric, of (opt (neg(QE-dnf' opt step φ)))]
    apply(rule ex-cong1)
    subgoal for nat xs
      apply(cases length xs = Suc nat)
      apply simp-all
      apply(rule bex-cong)
      apply simp-all
      subgoal for f
        apply(cases f)
        apply simp
        apply(rule ex-cong1)
        unfolding eval-list-conj
        apply auto
        by (meson Un-iff eval.simps(1) imageI)
      done
    done
```

qed *auto*

theorem *QE-dnf-eval*:
assumes *steph* : $\bigwedge$*var amount new 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 amount var L F*) (*xs* @ Γ)))
assumes *opt* : $\bigwedge$*xs F* . *eval* (*opt F*) *xs* = *eval F xs*
shows *eval* (*QE-dnf opt step* φ) *xs* = *eval* φ *xs*
proof(*induction* φ *arbitrary*:*xs*)
case (*Atom x*)
then show *?case* **by** (*simp add*: *simp-atom-eval*)
next
case (*And* φ*1* φ*2*)
then show *?case* **by** (*simp add*: *eval-and*)
next
case (*Or* φ*1* φ*2*)
then show *?case* **by** (*simp add*: *eval-or*)
next
case (*Neg* φ)
then show *?case*
by (*metis eval.simps*(*6*) *eval-neg QE-dnf.simps*(*3*))
next
case (*ExQ* φ)
have *h* : (∃ *x*. ∃ (*al*, *fl*, *n*)∈*set* (*dnf-modified* (*opt* (*QE-dnf opt step* φ))).
∃ *L*. *length L* = *n* ∧ (∀ *a*∈*set al*. *aEval a* (*L* @ *x* # *xs*)) ∧ (∀ *f*∈*set fl*.
eval f (*L* @ *x* # *xs*))) =
(∃ (*al*, *fl*, *n*)∈*set* (*dnf-modified* (*opt* (*QE-dnf opt step* φ))). ∃ *x*.
∃ *L*. *length L* = *n* ∧ (∀ *a*∈*set al*. *aEval a* (*L* @ *x* # *xs*)) ∧ (∀ *f*∈*set fl*.
eval f (*L* @ *x* # *xs*)))
apply *safe*
by *blast*+
have *lessThan* : $\bigwedge$*c*. *Suc 0* ≤ *c* + *1*
by *simp*
show *?case* **apply** (*simp add*:*eval-list-disj*)
unfolding *ExQ*[*symmetric*]
unfolding *opt*[*symmetric*, *of* (*QE-dnf opt step* φ)]
unfolding *dnf-modified-eval*[*symmetric*, *of opt*(*QE-dnf opt step* φ)]
unfolding *h*
apply(*rule bex-cong*)
apply *simp*
subgoal for *f*
apply(*cases f*)
apply *simp*
subgoal for *a b c*
using *steph*[*of Suc 0 c a b xs*, *symmetric*, *OF lessThan*] **apply** (*simp*
add:*eval-list-conj*)

```
        apply safe
        subgoal for xs' l' l''
          apply(rule exI[where x=l''!c])
          apply(rule exI[where x=take c l''])
          apply auto
           apply (metis Un-iff append.assoc append-Cons append-Nil eval.simps(1)
image-eqI lessI order-refl take-Suc-conv-app-nth take-all)
            by (metis Un-iff append.assoc append-Cons append-Nil lessI order-refl
take-Suc-conv-app-nth take-all)
        subgoal for A B C D
          apply(rule exI[where x=D@[C]]) by auto
        subgoal for A B
          apply(rule exI[where x=B@[A]]) by auto
        done
      done
    done
next
  case (AllQ φ)
  have h : (∃ x. ∃ (al, fl, n)∈set (dnf-modified (opt (neg(QE-dnf opt step φ))))).
            ∃ L. length L = n ∧ (∀ a∈set al. aEval a (L @ x # xs)) ∧ (∀ f∈set fl.
eval f (L @ x # xs))) =
        (∃ (al, fl, n)∈set (dnf-modified (opt (neg(QE-dnf opt step φ))))). ∃ x.
            ∃ L. length L = n ∧ (∀ a∈set al. aEval a (L @ x # xs)) ∧ (∀ f∈set fl.
eval f (L @ x # xs)))
    apply safe
    by blast+
  have lessThan : ⋀c. Suc 0 ≤ c + 1
    by simp
  show ?case
    apply (simp add:eval-list-disj)
    unfolding AllQ[symmetric]
    unfolding eval-neg[symmetric, of (QE-dnf opt step φ)]
    unfolding opt[symmetric, of neg(QE-dnf opt step φ)]
   unfolding HOL.Not-eq-iff[symmetric, of (∀ f∈set (dnf-modified (opt (neg (QE-dnf
opt step φ))))). ¬ eval (case f of (al, fl, n) ⇒ ExN (Suc n) (step (Suc 0) n al fl))
xs)]
    unfolding SMT.verit-connective-def(3)[symmetric]
    unfolding boundedFlipNegQuantifier
    unfolding dnf-modified-eval[symmetric, of opt(neg(QE-dnf opt step φ))]
    unfolding h
    apply(rule bex-cong)
    apply simp
    subgoal for f
      apply(cases f)
      apply simp
      subgoal for a b c
          using steph[of Suc 0 c a b xs, symmetric,OF lessThan] apply (simp
add:eval-list-conj)
        apply safe
```

```
        subgoal for xs' l' l''
          apply(rule exI[where x=l'!c])
          apply(rule exI[where x=take c l'])
          apply auto
           apply (metis Un-iff append.assoc append-Cons append-Nil eval.simps(1)
image-eqI lessI order-refl take-Suc-conv-app-nth take-all)
            by (metis Un-iff append.assoc append-Cons append-Nil lessI order-refl
take-Suc-conv-app-nth take-all)
        subgoal for A B C D
          apply(rule exI[where x=D@[C]]) by auto
        subgoal for A B
          apply(rule exI[where x=B@[A]]) by auto
        done
      done
    done
next
  case (ExN x1 φ)
  show ?case
  proof(cases x1)
    case 0
    then show ?thesis using ExN by simp
  next
    case (Suc nat)
    have h : (∃ l. length l = Suc nat ∧
         (∃ (al, fl, n)∈set (dnf-modified (opt (QE-dnf opt step φ))).
              ∃ L. length L = n ∧ (∀ a∈set al. aEval a (L @ l @ xs)) ∧ (∀ f∈set fl.
eval f (L @ l @ xs)))) =
         (∃ (al, fl, n)∈set (dnf-modified (opt (QE-dnf opt step φ))). (∃ l. length l =
Suc nat ∧
              (∃ L. length L = n ∧ (∀ a∈set al. aEval a (L @ l @ xs)) ∧ (∀ f∈set fl.
eval f (L @ l @ xs)))))
      apply safe
      by blast+
    have lessThan : ⋀c. Suc nat ≤ c + nat + 1 by simp
    show ?thesis
      apply (simp add:eval-list-disj Suc)
      unfolding ExN[symmetric]
      unfolding opt[symmetric, of (QE-dnf opt step φ)]
      unfolding dnf-modified-eval[symmetric, of (opt (QE-dnf opt step φ))]
      unfolding h
      apply(rule bex-cong)
      apply simp
      subgoal for f
        apply(cases f)
        subgoal for a b c
          apply simp
          using steph[of Suc nat c+nat,symmetric, OF lessThan]
          apply (auto simp add:eval-list-conj)
          subgoal for L
```

```
          apply(rule exI[where x=drop c L])
          apply auto
          apply(rule exI[where x=take c L])
          apply auto
            apply (metis Un-iff append.assoc append-take-drop-id eval.simps(1)
image-eqI)
          by (metis Un-iff append.assoc append-take-drop-id)
        subgoal for L l
          apply(rule exI[where x=l@L])
          by auto
        done
      done
    done
  qed
next
  case (AllN x1 φ)
  then show ?case
  proof(cases x1)
    case 0
    then show ?thesis using AllN by simp
  next
    case (Suc nat)
    have h : (∃ l. length l = Suc nat ∧
        (∃ (al, fl, n)∈set (dnf-modified (opt (neg(QE-dnf opt step φ)))).
            ∃ L. length L = n ∧ (∀ a∈set al. aEval a (L @ l @ xs)) ∧ (∀ f∈set fl.
eval f (L @ l @ xs)))) =
       (∃ (al, fl, n)∈set (dnf-modified (opt (neg(QE-dnf opt step φ)))). (∃ l. length
l = Suc nat ∧
            (∃ L. length L = n ∧ (∀ a∈set al. aEval a (L @ l @ xs)) ∧ (∀ f∈set fl.
eval f (L @ l @ xs)))))
      apply safe
      by blast+
    have lessThan : ⋀c. Suc nat ≤ c + nat + 1 by simp
    show ?thesis
      apply (simp add:eval-list-disj Suc)
      unfolding AllN[symmetric]
      unfolding eval-neg[symmetric, of QE-dnf opt step φ]
      unfolding HOL.imp-conv-disj
      unfolding HOL.de-Morgan-conj[symmetric]
      unfolding opt[symmetric, of neg(QE-dnf opt step φ)]
      unfolding dnf-modified-eval[symmetric, of (opt (neg(QE-dnf opt step φ)))]
       unfolding HOL.Not-eq-iff[symmetric, of (∀ f∈set (dnf-modified (opt (neg
(QE-dnf opt step φ))))).
        ¬ eval (case f of (al, fl, n) ⇒ ExN (Suc (n + nat)) (step (Suc nat) (n +
nat) al fl)) xs)]
      unfolding SMT.verit-connective-def(3)[symmetric]
      unfolding boundedFlipNegQuantifier
      unfolding h
      apply(rule bex-cong)
```

```
    apply simp
    subgoal for f
      apply(cases f)
      subgoal for a b c
        apply simp
        using steph[of Suc nat c+nat,symmetric, OF lessThan]
        apply (auto simp add:eval-list-conj)
        subgoal for L
          apply(rule exI[where x=drop c L])
          apply auto
          apply(rule exI[where x=take c L])
          apply auto
            apply (metis Un-iff append.assoc append-take-drop-id eval.simps(1)
image-eqI)
          by (metis Un-iff append.assoc append-take-drop-id)
        subgoal for L l
          apply(rule exI[where x=l@L])
          by auto
        done
      done
    done
  qed
qed auto

lemma opt: eval ((push-forall ∘ nnf ∘ unpower 0 o groupQuantifiers o clearQuan-
tifiers) F) L= eval F L
  using push-forall-eval eval-nnf unpower-eval groupQuantifiers-eval clearQuanti-
fiers-eval by auto

lemma opt': eval ((push-forall ( nnf ( unpower 0 ( groupQuantifiers (clearQuantifiers
F)))))) L= eval F L
  using push-forall-eval eval-nnf unpower-eval groupQuantifiers-eval clearQuanti-
fiers-eval by auto

lemma opt-no-group: eval ((push-forall ∘ nnf ∘ unpower 0 o clearQuantifiers) F)
L= eval F L
  using push-forall-eval eval-nnf unpower-eval clearQuantifiers-eval by auto



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
  apply(induction n arbitrary : F)
  apply simp-all
  subgoal for n F
    using assms[of F xs] by auto
  done
```

lemma *repeatAmountOfQuantifiers-eval* :
assumes $\bigwedge xs\ F.\ eval\ F\ xs = eval\ (step\ F)\ xs$
shows $eval\ F\ xs = eval\ (repeatAmountOfQuantifiers\ step\ F)\ xs$
proof−
define F' **where** $F' = step\ F$
have *h*: $eval\ F\ xs = eval\ F'\ xs$
using *assms* **unfolding** F'*-def* **by** *auto*
show *?thesis*
apply (*simp add*: F'*-def*[*symmetric*] *h*)
using *repeatAmountOfQuantifiers-helper-eval*[*OF assms*] **by** *auto*
qed

end

10.3 Heuristic Proofs

theory *HeuristicProofs*
imports *VSQuad Heuristic OptimizationProofs*
begin

lemma *the-real-step-augment*:
assumes *steph* : $\bigwedge xs\ var\ L\ F\ \Gamma.\ length\ xs = var \Longrightarrow (\exists x.\ eval\ (list\text{-}conj\ (map\ fm.Atom\ L\ @\ F))\ (xs\ @\ x\ \#\ \Gamma)) = (\exists x.\ eval\ (step\ var\ L\ F)\ (xs\ @\ x\ \#\ \Gamma))$
shows $(\exists xs.\ (length\ xs = amount \wedge eval\ (list\text{-}disj\ (map(\lambda(L,F,n).\ ExN\ n\ (list\text{-}conj\ (map\ fm.Atom\ L\ @\ F)))\ F))\ (xs\ @\ \Gamma))) = (eval\ (the\text{-}real\text{-}step\text{-}augment\ step\ amount\ F)\ \Gamma)$
proof(*induction amount arbitrary*: *F* Γ)
case *0*
then show *?case* **by** *auto*
next
case (*Suc amount*)
have *h1* : $\bigwedge F.\ (\exists x\ xs.\ length\ xs = amount \wedge F\ (xs\ @\ x\ \#\ \Gamma)) = (\exists xs.\ length\ xs = Suc\ amount \wedge F\ (xs\ @\ \Gamma))$
by (*smt* (*z3*) *Suc-inject append.assoc append-Cons append-Nil2 append-eq-conv-conj length-append-singleton lessI self-append-conv2 take-hd-drop*)

have *h2*: $\bigwedge X\ x\ \Gamma.\ (\exists f{\in}set\ (dnf\text{-}modified\ X).$
$eval\ (case\ f\ of\ (L,\ F,\ n) \Rightarrow ExN\ n\ (list\text{-}conj\ (map\ fm.Atom\ L\ @\ F)))\ (x\ @\ \Gamma)) = (\exists (al,\ fl,\ n){\in}set\ (dnf\text{-}modified\ X).\ \exists L.\ length\ L = n \wedge (\forall a{\in}set\ al.\ aEval\ a\ (L\ @\ (x\ @\ \Gamma))) \wedge (\forall f{\in}set\ fl.\ eval\ f\ (L\ @\ (x\ @\ \Gamma))))$
subgoal for *X x* Γ
apply(*rule bex-cong*)
apply *simp-all*
subgoal for *f*
apply(*cases f*)
apply(*auto simp add*:*eval-list-conj*)
by (*metis Un-iff eval.simps*(*1*) *imageI*)
done

```
  done
 have h3 : ⋀G. (∃ x. ∃ f∈set F. G x f) = (∃ f∈set F. ∃ x. G x f)
  by blast
 show ?case
  apply simp
  unfolding Suc[symmetric]
  unfolding eval-list-disj
  apply simp
  unfolding h1[symmetric, of λx. (∃ f∈set F. eval (case f of (L, F, n) ⇒ ExN
n (list-conj (map fm.Atom L @ F))) x)]
  unfolding HOL.ex-comm[of λx xs. length xs = amount ∧ (∃ f∈set F. eval (case
f of (L, F, n) ⇒ ExN n (list-conj (map fm.Atom L @ F))) (xs @ x # Γ))]
  unfolding HOL.ex-comm[of λx xs. length xs = amount ∧
     (∃ f∈set (dnf-modified (push-forall
              (nnf (unpower 0
                   (groupQuantifiers
                     (clearQuantifiers(list-disj (map (λ(L, F, n). ExN n (step
(n + amount) L F)) F)))))))).
       eval (case f of (L, F, n) ⇒ ExN n (list-conj (map fm.Atom L @ F))) (xs
@ x # Γ))]
  apply(rule ex-cong1)
  apply simp
  subgoal for xs
   unfolding h2
   unfolding dnf-modified-eval
   unfolding opt'
   unfolding eval-list-disj
   unfolding List.set-map Set.bex-simps(7)
   unfolding h3
   apply(cases length xs = amount)
   apply (simp-all add:opt)
   apply(rule bex-cong)
   apply simp-all
   subgoal for f
    apply(cases f)
    apply simp
    subgoal for a b c
      unfolding HOL.ex-comm[of λx l. length l = c ∧ eval (list-conj (map
fm.Atom a @ b)) (l @ xs @ x # Γ)]
     unfolding HOL.ex-comm[of λx l. length l = c ∧ eval (step (c + amount)
a b) (l @ xs @ x # Γ)]
      apply(rule ex-cong1)
      apply simp
      subgoal for l
       apply(cases length l = c)
       apply simp-all
       using steph[of l @ xs c + amount a b Γ]
       by simp
      done
```

```
      done
    done
  done
qed

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 safe
  fix var L F Γ xs
  assume h : length xs = var + 1
    eval (list-conj (map fm.Atom L @ F)) (xs @ Γ)
  have h1 : length (take var xs) = var using h by auto
  have h2 : (∃x. eval (step var L F) (take var xs @ x # Γ))
    using h steph[OF h1]
    by (metis Cons-nth-drop-Suc One-nat-def add.right-neutral add-Suc-right ap-
pend.assoc append-Cons append-Nil append-take-drop-id drop-all lessI order-refl)

  then obtain x where h3: eval (step var L F) (take var xs @ x # Γ) by auto
  show ∃xs. length xs = var + 1 ∧ eval (step var L F) (xs @ Γ)
    apply(rule exI[where x=take var xs @[x]])
    apply (auto)
    using h(1) apply simp
    using h3 by simp
next
  fix var L F Γ xs
  assume h: length xs = var + 1
    eval (step var L F) (xs @ Γ)
  have h1 : length (take var xs) = var using h by auto
  have h2 : (∃x. eval (list-conj (map fm.Atom L @ F)) (take var xs @ x # Γ))
    using h steph[OF h1]
    by (metis Cons-nth-drop-Suc One-nat-def add.right-neutral add-Suc-right ap-
pend.assoc append-Cons append-Nil append-take-drop-id drop-all lessI order-refl)

  then obtain x where h3: eval (list-conj (map fm.Atom L @ F)) (take var xs @
x # Γ) by auto
  show ∃xs. length xs = var + 1 ∧ eval (list-conj (map fm.Atom L @ F)) (xs @
Γ)
    apply(rule exI[where x=take var xs @[x]])
    apply (auto)
    using h(1) apply simp
    using h3 by simp
qed

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*))
subgoal for *var amount L F Γ*
proof(*induction var arbitrary*: *L F Γ amount*)
case *0*
then have *amount* = *0* ∨ *amount* = *Suc 0* **by** *auto*
then show *?case* **apply** *simp* **using** *steph*[*of* [] *0 L F Γ*] **apply** *auto*
apply (*metis append-Cons length-Cons list.size*(*3*) *self-append-conv2*)
apply (*metis append-Cons length-Cons list.size*(*3*) *self-append-conv2*)
apply (*metis Suc-length-conv append-Cons length-0-conv self-append-conv2*)
by (*metis Suc-length-conv append-Cons append-self-conv2 length-0-conv*)
next
case (*Suc var*)
define *heu* **where** *heu* = *heuristic* (*Suc var*) *L F*
have *heurange* : *heu* ≤ *Suc var* **unfolding** *heu-def*
by (*simp add*: *heuristic*)
have *lessThan1* : *1* ≤ *var* + *1* **by** *auto*

{
fix *amount*
assume *amountLessThan*: *amount* ≤ *var* + *1*
have (∃ *xs*. *length xs* = *Suc* (*Suc var*) ∧
eval (*list-conj* (*map fm.Atom L @ F*)) (*xs @ Γ*)) = (∃ *xs*. *length xs* = *Suc* (*Suc var*) ∧
eval
(*step* (*Suc var*) (*map* (*swap-atom* (*Suc var*) *heu*) *L*)
(*map* (*swap-fm* (*Suc var*) *heu*) *F*))
(*xs @ Γ*))
proof(*safe*)
fix *xs*
assume *h*: *length* (*xs*::*real list*) = *Suc* (*Suc var*) *eval* (*list-conj* (*map fm.Atom L @ F*)) (*xs @ Γ*)
then have *length* : *length* (*take* (*Suc var*) (*swap-list* (*Suc var*) *heu xs*)) = *Suc var* **by** *auto*
have *take*: (*take* (*Suc var*) (*swap-list* (*Suc var*) *heu xs*) @ *xs* ! *heu* # *Γ*) = (*swap-list* (*Suc var*) *heu* (*xs @ Γ*)) **using** *h*(*1*)
unfolding *swap-list.simps*
by (*smt* (*verit*, *ccfv-threshold*) *Cons-nth-drop-Suc append.right-neutral append-Nil2 append-assoc append-eq-conv-conj append-self-conv2 append-take-drop-id drop0 heu-def heurange le-imp-less-Suc length-greater-0-conv length-list-update lessI list.sel*(*1*) *list.sel*(*3*) *list.simps*(*3*) *list.size*(*3*) *list-update-append nth-Cons-0 nth-append nth-append-length nth-list-update-eq take0 take-hd-drop*)
have *length1* : *Suc var* < *length* (*xs @ Γ*) **using** *h* **by** *auto*

have *length2* : *heu* < *length* (*xs* @ Γ) **using** *h heurange* **by** *auto*
have *h1*: (∃ *x*. *eval*
(*step* (*Suc var*) (*map* (*swap-atom* (*Suc var*) *heu*) *L*)
(*map* (*swap-fm* (*Suc var*) *heu*) *F*))
(*take* (*Suc var*) (*swap-list* (*Suc var*) *heu xs*) @ *x* # Γ))
unfolding *steph*[*OF length*, *symmetric*]
apply(*rule exI*[**where** *x*=*nth xs heu*])
using *h* **unfolding** *eval-list-conj take* **apply** (*auto simp del*:*swap-list.simps*)
unfolding *swap-fm*[*OF length1 length2*,*symmetric*] *swap-atom*[*OF length1 length2*,*symmetric*]
by (*meson UnCI eval.simps*(*1*) *imageI*)+
then obtain *x* **where** *heval*: *eval*
(*step* (*Suc var*) (*map* (*swap-atom* (*Suc var*) *heu*) *L*)
(*map* (*swap-fm* (*Suc var*) *heu*) *F*))
(*take* (*Suc var*) (*swap-list* (*Suc var*) *heu xs*) @ *x* # Γ) **by** *auto*
show ∃ *xs*. *length xs* = *Suc* (*Suc var*) ∧
eval
(*step* (*Suc var*) (*map* (*swap-atom* (*Suc var*) *heu*) *L*)
(*map* (*swap-fm* (*Suc var*) *heu*) *F*))
(*xs* @ Γ)
apply(*rule exI*[**where** *x*=*take* (*Suc var*) (*swap-list* (*Suc var*) *heu xs*) @ [*x*]])
apply *auto*
using *h* **apply** *simp*
using *heval* **by** *auto*
next
fix *xs*
assume *h* : *length xs* = *Suc* (*Suc var*)
eval
(*step* (*Suc var*) (*map* (*swap-atom* (*Suc var*) *heu*) *L*)
(*map* (*swap-fm* (*Suc var*) *heu*) *F*))
(*xs* @ Γ)
define *choppedXS* **where** *choppedXS* = *take* (*Suc var*) *xs*
then have *length* : *length choppedXS* = *Suc var*
using *h*(*1*) **by** *force*
have (∃ *x*. *eval* (*step* (*Suc var*) (*map* (*swap-atom* (*Suc var*) *heu*) *L*) (*map* (*swap-fm* (*Suc var*) *heu*) *F*)) (*choppedXS* @ *x* # Γ))
using *h*(*2*) *choppedXS-def*
by (*metis append.assoc append-Cons append-Nil2 append-eq-conv-conj h*(*1*) *lessI take-hd-drop*)
then have ∃ *x*. (∀ *l*∈ *set L*. *aEval* (*swap-atom* (*Suc var*) *heu l*) (*choppedXS*@*x*#Γ)) ∧ (∀ *f*∈ *set F*. *eval* (*swap-fm* (*Suc var*) *heu f*) (*choppedXS*@*x*#Γ))
unfolding *steph*[*symmetric*, *OF length*, *of* (*map* (*swap-atom* (*Suc var*) *heu*) *L*) (*map* (*swap-fm* (*Suc var*) *heu*) *F*) Γ] *eval-list-conj* **apply** *auto*
by (*metis Un-iff eval.simps*(*1*) *imageI*)
then obtain *x* **where** *x* : (∀ *l*∈*set L*. *aEval* (*swap-atom* (*Suc var*) *heu l*) (*choppedXS* @ *x* # Γ)) ∧
(∀ *f*∈*set F*. *eval* (*swap-fm* (*Suc var*) *heu f*) (*choppedXS* @ *x* # Γ)) **by** *auto*
have *length1* : *Suc var* < *length* (*swap-list* (*Suc var*) *heu* (*choppedXS* @

[x]) @ Γ)
by (*simp add*: *length*)
have *length2* : *heu* < *length* (*swap-list* (*Suc var*) *heu* (*choppedXS* @ [*x*]) @ Γ)
using ‹*Suc var* < *length* (*swap-list* (*Suc var*) *heu* (*choppedXS* @ [*x*]) @ Γ)› *heurange* **by** *linarith*
have *swapswap* : (*swap-list* (*Suc var*) *heu* (*swap-list* (*Suc var*) *heu* (*choppedXS* @ [*x*]) @ Γ)) = (*choppedXS* @ [*x*]) @ Γ **apply** *auto*
by (*smt* (*z3*) *Cons-nth-drop-Suc append-eq-conv-conj append-same-eq heurange id-take-nth-drop le-neq-implies-less length length1 length-append-singleton lessI list.sel*(*1*) *list-update-append1 list-update-length list-update-swap nth-append nth-append-length nth-list-update-neq swap-list.simps take-hd-drop take-update-swap upd-conv-take-nth-drop*)
show ∃ *xs*. *length xs* = *Suc* (*Suc var*) ∧
eval (*list-conj* (*map fm.Atom L* @ *F*)) (*xs* @ Γ)
apply(*rule exI*[**where** *x*=*swap-list* (*Suc var*) *heu* (*choppedXS* @ [*x*])])
apply(*auto simp add*: *eval-list-conj simp del*: *swap-list.simps*)
apply(*simp add* :*length*)
unfolding *swap-atom*[*OF length1 length2*] *swap-fm*[*OF length1 length2*] *swapswap*
using *x* **by** *auto*
qed
also have ... = (∃ *xs*. *length xs* = *Suc* (*Suc var*) ∧
(∃ *f*∈*set* (*dnf* ((*push-forall* ∘ *nnf* ∘ *unpower 0 o groupQuantifiers o clearQuantifiers*)(*step* (*Suc var*) (*map* (*swap-atom* (*Suc var*) *heu*) *L*)
(*map* (*swap-fm* (*Suc var*) *heu*) *F*)))).
eval (*case f of* (*x*, *xa*) ⇒ *step-augment step heuristic amount var x xa*)
(*xs* @ Γ)))
unfolding *opt*[*of* (*step* (*Suc var*) (*map* (*swap-atom* (*Suc var*) *heu*) *L*) (*map* (*swap-fm* (*Suc var*) *heu*) *F*)), *symmetric*]
unfolding *dnf-eval*[*of* (*push-forall* ∘ *nnf* ∘ *unpower 0 o groupQuantifiers o clearQuantifiers*)(*step* (*Suc var*) (*map* (*swap-atom* (*Suc var*) *heu*) *L*)
(*map* (*swap-fm* (*Suc var*) *heu*) *F*)), *symmetric*]
proof(*safe*)
fix *xs a b*
assume *h*: *length xs* = *Suc* (*Suc var*)
(*a*, *b*)
∈ *set* (*dnf* ((*push-forall* ∘ *nnf* ∘ *unpower 0 o groupQuantifiers o clearQuantifiers*)(*step* (*Suc var*) (*map* (*swap-atom* (*Suc var*) *heu*) *L*)
(*map* (*swap-fm* (*Suc var*) *heu*) *F*))))
∀ *a*∈*set a*. *aEval a* (*xs* @ Γ)
∀ *f*∈*set b*. *eval f* (*xs* @ Γ)
have (∃ *xs*′. *length xs*′ = *var* + *1* ∧
eval (*step-augment step heuristic amount var a b*) (*xs*′ @ *xs* ! *Suc var* # Γ))
unfolding *Suc*(*1*)[*of amount a b nth xs* (*Suc var*)#Γ, *OF amountLessThan*, *symmetric*]
apply(*rule exI*[**where** *x*=*take* (*Suc var*) *xs*])
using *h*(*1*) *h*(*3*−*4*) **apply**(*auto simp add*: *eval-list-conj*)
apply (*metis Cons-nth-drop-Suc append-Cons append-eq-append-conv2*

append-eq-conv-conj append-take-drop-id lessI)
by (*metis Cons-nth-drop-Suc append-Cons append-eq-append-conv2 append-eq-conv-conj append-take-drop-id lessI*)
then obtain *xs'* **where** *xs'*: *length xs' = var + 1 eval* (*step-augment step heuristic amount var a b*) (*xs' @ xs ! Suc var # Γ*)
by *auto*

show ∃ *xs*. *length xs = Suc* (*Suc var*) ∧
(∃ *f*∈*set* (*dnf* ((*push-forall* ∘ *nnf* ∘ *unpower 0 o groupQuantifiers o clearQuantifiers*)(*step* (*Suc var*) (*map* (*swap-atom* (*Suc var*) *heu*) *L*)
(*map* (*swap-fm* (*Suc var*) *heu*) *F*)))).
eval (*case f of* (*x*, *xa*) ⇒ *step-augment step heuristic amount var x xa*)
(*xs* @ Γ))
apply(*rule exI*[**where** *x*=*xs'* @[*xs ! Suc var*]])
apply *auto*
using *xs'* **apply** *simp*
apply(*rule bexI*[**where** *x*=(*a*,*b*)])
using *xs' h* **apply**(*cases amount*) **apply** (*simp-all add*:*eval-list-conj*)
using *h*(*2*) **by** *auto*
next
fix *xs a b*
assume *h*: *length xs = Suc* (*Suc var*)
(*a*, *b*)
∈ *set* (*dnf* ((*push-forall* ∘ *nnf* ∘ *unpower 0 o groupQuantifiers o clearQuantifiers*)(*step* (*Suc var*) (*map* (*swap-atom* (*Suc var*) *heu*) *L*)
(*map* (*swap-fm* (*Suc var*) *heu*) *F*))))
eval (*step-augment step heuristic amount var a b*) (*xs* @ Γ)
have (∃ *xs'*. *length xs' = var + 1* ∧
eval (*list-conj* (*map fm.Atom a* @ *b*)) (*xs'* @ *xs* ! *Suc var* # Γ))
unfolding *Suc*(*1*)[*of amount a b nth xs* (*Suc var*)#Γ, *OF amountLessThan*]
apply(*rule exI*[**where** *x*=*take* (*Suc var*) *xs*])
using *h*(*1*) *h*(*3*) **apply** *auto*
by (*metis Cons-nth-drop-Suc append.right-neutral append-Cons append-assoc append-eq-conv-conj append-self-conv2 append-take-drop-id lessI*)
then obtain *xs'* **where** *xs'*: *length xs' = var + 1 eval* (*list-conj* (*map fm.Atom a* @ *b*)) (*xs'* @ *xs* ! *Suc var* # Γ)
by *auto*
show ∃ *xs*. *length xs = Suc* (*Suc var*) ∧
(∃ (*al*, *fl*)
∈*set* (*dnf* ((*push-forall* ∘ *nnf* ∘ *unpower 0 o groupQuantifiers o clearQuantifiers*)(*step* (*Suc var*) (*map* (*swap-atom* (*Suc var*) *heu*) *L*)
(*map* (*swap-fm* (*Suc var*) *heu*) *F*)))).
(∀ *a*∈*set al*. *aEval a* (*xs* @ Γ)) ∧
(∀ *f*∈*set fl*. *eval f* (*xs* @ Γ)))
apply(*rule exI*[**where** *x*=*xs'* @[*xs ! Suc var*]])
apply *auto*
using *xs'* **apply** *simp*
apply(*rule bexI*[**where** *x*=(*a*,*b*)])
using *xs' h* **apply** (*simp-all add*: *eval-list-conj*)

proof −
assume $\forall f{\in}fm.Atom\ `\ set\ a \cup set\ b.\ eval\ f\ (xs' @ xs\ !\ Suc\ var\ \#\ \Gamma)$
then have $\forall f.\ f \in fm.Atom\ `\ set\ a \cup set\ b \longrightarrow eval\ f\ (xs' @ xs\ !\ Suc\ var\ \#\ \Gamma)$
by *meson*
then have *f1*: $v \notin set\ a \vee eval\ (fm.Atom\ v)\ (xs' @ xs\ !\ Suc\ var\ \#\ \Gamma)$ **for** v
by *blast*
obtain $aa :: atom$ **where**
$(\exists v0.\ v0 \in set\ a \wedge \neg\ eval\ (fm.Atom\ v0)\ (xs' @ xs\ !\ Suc\ var\ \#\ \Gamma)) = (aa \in set\ a \wedge \neg\ eval\ (fm.Atom\ aa)\ (xs' @ xs\ !\ Suc\ var\ \#\ \Gamma))$
by *blast*
then show $\forall a{\in}set\ a.\ aEval\ a\ (xs' @ xs\ !\ Suc\ var\ \#\ \Gamma)$
using *f1 eval.simps(1)* **by** *auto*
qed

qed
finally have $(\exists xs.\ length\ xs = Suc\ (Suc\ var) \wedge eval\ (list\text{-}conj\ (map\ fm.Atom\ L @ F))\ (xs @ \Gamma)) =$
$(\exists xs.\ length\ xs = Suc\ (Suc\ var) \wedge$
$(\exists f{\in}set\ (dnf\ ((push\text{-}forall \circ nnf \circ unpower\ 0\ o\ groupQuantifiers\ o\ clear\text{-}Quantifiers)\ (step\ (Suc\ var)\ (map\ (swap\text{-}atom\ (Suc\ var)\ heu)\ L)\ (map\ (swap\text{-}fm\ (Suc\ var)\ heu)\ F)))).$
$eval\ (case\ f\ of\ (x,\ xa) \Rightarrow step\text{-}augment\ step\ heuristic\ amount\ var\ x\ xa)\ (xs @ \Gamma)))$
by *auto*
}then show *?case* **apply**(*cases amount*) **using** *Suc(2)* **by** (*simp-all add*:*eval-list-disj heu-def*[*symmetric*])
qed
done

lemma *qe-eq-repeat-eval-augment* : $amount \leq var{+}1 \Longrightarrow$
$(\exists xs.\ (length\ xs = var + 1) \wedge eval\ (list\text{-}conj\ (map\ fm.Atom\ L @ F))\ (xs @ \Gamma)) =$
$(\exists xs.\ (length\ xs = var + 1) \wedge eval\ (step\text{-}augment\ qe\text{-}eq\text{-}repeat\ IdentityHeuristic\ amount\ var\ L\ F)\ (xs @ \Gamma))$
apply(*rule step-augmenter-eval*[*of qe-eq-repeat IdentityHeuristic amount var L F* Γ])
using *qe-eq-repeat-eval* **apply** *blast* **by** *auto*

lemma *qe-eq-repeat-eval'* :
$(\exists xs.\ (length\ xs = var + 1) \wedge eval\ (list\text{-}conj\ (map\ fm.Atom\ L @ F))\ (xs @ \Gamma)) =$
$(\exists xs.\ (length\ xs = var + 1) \wedge eval\ (qe\text{-}eq\text{-}repeat\ var\ L\ F)\ (xs @ \Gamma))$
apply(*rule step-converter*[*of qe-eq-repeat var L F* Γ])
using *qe-eq-repeat-eval* **by** *blast*

lemma *gen-qe-eval-augment* : $amount \leq var{+}1 \Longrightarrow$
$(\exists xs.\ (length\ xs = var + 1) \wedge eval\ (list\text{-}conj\ (map\ fm.Atom\ L @ F))\ (xs @$

Γ)) =
(∃ *xs*. (*length xs* = *var* + *1*) ∧ *eval* (*step-augment gen-qe IdentityHeuristic amount var L F*) (*xs* @ Γ))
apply(*rule step-augmenter-eval*[*of gen-qe IdentityHeuristic amount var L F* Γ])
using *gen-qe-eval* **apply** *blast* **by** *auto*

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))
apply(*rule step-converter*[*of gen-qe var L F* Γ])
using *gen-qe-eval* **by** *blast*

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))
apply(*rule step-augmenter-eval*[*of luckyFind′ IdentityHeuristic amount var L F* Γ])
using *luckyFind′-eval* **apply** *blast* **by** *auto*

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))
apply(*rule step-converter*[*of luckyFind′ var L F* Γ])
using *luckyFind′-eval* **by** *blast*

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))
apply(*rule step-converter*[*of luckiestFind var L F* Γ])
using *luckiestFind-eval* **by** *blast*

lemma *sortedListMember* : *sorted-list-of-fset b* = *var* # *list* ⟹ *fmember var b*
by (*metis fset-of-list-elem list.set-intros*(*1*) *sorted-list-of-fset-simps*(*2*))

lemma *rangeHeuristic* :
assumes *heuristicPicker n L F* = *Some* (*var*, *step*)
shows *var*≤*n*
proof(*cases aquireData n L*)
case (*fields a b c*)
then show *?thesis* **using** *assms* **apply**(*simp-all del*: *aquireData.simps getBest.simps*)
apply(*cases getBest a L*)
apply(*simp-all del*: *aquireData.simps getBest.simps*)
apply(*cases F*) **apply**(*simp-all del*: *aquireData.simps getBest.simps*)

apply(*cases getBest c L*) **apply**(*simp-all del*: *aquireData.simps getBest.simps*)
apply(*cases getBest b L*)**apply**(*simp-all del*: *aquireData.simps getBest.simps*)
apply (*metis not-le-imp-less option.distinct*(*1*) *option.inject prod.inject*)
apply (*metis not-le-imp-less option.distinct*(*1*) *option.inject prod.inject*)
apply(*cases getBest b L*)**apply**(*simp-all del*: *aquireData.simps getBest.simps*)
by (*metis not-le-imp-less option.distinct*(*1*) *option.inject prod.inject*)+
qed

lemma *pickedOneOfThem* :
assumes *heuristicPicker n L F* = *Some* (*var*, *step*)
shows *step* = *qe-eq-repeat* ∨ *step* = *gen-qe* ∨ *step* = *luckyFind′*
using *assms*
apply(*cases aquireData n L*)
subgoal for *l e g*
using *assms* **apply**(*simp-all del*: *aquireData.simps getBest.simps*)
apply(*cases getBest l L*)
apply(*simp-all del*: *aquireData.simps getBest.simps*)
apply(*cases F*) **apply**(*simp-all del*: *aquireData.simps getBest.simps*)
apply(*cases getBest g L*) **apply**(*simp-all del*: *aquireData.simps getBest.simps*)
apply(*cases getBest e L*)**apply**(*simp-all del*: *aquireData.simps getBest.simps*)
apply (*metis option.distinct*(*1*) *option.inject prod.inject*)
apply (*metis option.distinct*(*1*) *option.inject prod.inject*)
apply(*cases getBest e L*)**apply**(*simp-all del*: *aquireData.simps getBest.simps*)
by (*metis option.distinct*(*1*) *option.inject prod.inject*)+
done

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(*induction var arbitrary* : *L F* Γ *amount*)
case *0*
then show *?case* **apply**(*simp del*:*heuristicPicker.simps*)
apply(*cases heuristicPicker 0 L F*) **apply**(*cases amount*)
apply (*simp-all del*:*heuristicPicker.simps*)
subgoal for *a*
apply(*cases a*)
apply (*simp-all del*:*heuristicPicker.simps*)
subgoal for *var step*
apply(*cases var*) **apply**(*cases amount*)
apply(*simp-all del*:*heuristicPicker.simps*)
proof−
assume *h*: *heuristicPicker 0 L F* = *Some* (*0*, *step*)
show (∃ *xs*. *length xs* = *Suc 0* ∧ *eval* (*list-conj* (*map fm.Atom L* @ *F*)) (*xs* @ Γ)) =
(∃ *xs*. *length xs* = *Suc 0* ∧ *eval* (*step 0 L F*) (*xs* @ Γ))
using *pickedOneOfThem*[*OF h*]
using *qe-eq-repeat-eval′*[*of 0 L F* Γ] *gen-qe-eval′*[*of 0 L F* Γ] *luckyFind-eval′*[*of 0 L F* Γ]

```
        by auto
      next
        show ⋀nat. amount ≤ Suc 0 ⟹
          heuristicPicker 0 L F = Some (Suc nat, step) ⟹
          a = (Suc nat, step) ⟹
          var = Suc nat ⟹
          (∃ xs. length xs = Suc 0 ∧ eval (list-conj (map fm.Atom L @ F)) (xs @
Γ)) =
          (∃ xs. length xs = Suc 0 ∧ eval (superPicker amount 0 L F) (xs @ Γ))
        apply(cases amount) by(simp-all del:heuristicPicker.simps)
      qed
      done
    done
next
  case (Suc i)
  then show ?case apply(cases heuristicPicker (Suc i) L F) apply(cases amount)
    apply(simp-all del:heuristicPicker.simps)
    subgoal for a
      apply(cases a)
      apply(simp-all del:heuristicPicker.simps) apply(cases amount) apply simp
      apply(cases amount) apply(simp-all del:heuristicPicker.simps)
      subgoal for var step amountPred amountPred'
      proof−
        assume amountPred : amountPred ≤ Suc i
        assume ih: (⋀amount L F Γ.
        amount ≤ Suc i ⟹
        (∃ xs. length xs = Suc i ∧ eval (list-conj (map fm.Atom L @ F)) (xs @ Γ))
=
        (∃ xs. length xs = Suc i ∧ eval (superPicker amount i L F) (xs @ Γ)))
        assume h0 : heuristicPicker (Suc i) L F = Some (var, step)
        have h1: ⋀xs X F. (∃ f∈set (map (λ(x, y). F x y)
                    (dnf X)).
              eval f (xs)) = (∃ (al,fl)∈set(dnf X).
              eval (F al fl) (xs))
          subgoal for xs X F
            apply auto
            subgoal for a b
              apply(rule bexI[where x=(a,b)])
              apply simp-all
              done
            done
          done
        have eval-map : ⋀al fl xs Γ.(∀ f∈set (map fm.Atom al @ fl). eval f (xs @
Γ)) = ((∀ a∈set al. aEval a (xs @ Γ)) ∧ (∀ f∈set fl. eval f (xs @ Γ)))
          apply auto
          by (meson Un-iff eval.simps(1) imageI)
        have rearangeExists : ⋀ X F.((∃ xs. length xs = Suc (Suc i) ∧
              (∃ (al, fl)∈set (dnf X). F al fl xs)) =
            (∃ (al,fl)∈set (dnf X).(∃ xs. length xs = Suc (Suc i) ∧
```

F al fl xs)))
by *blast*
have *dropTheEnd* : $\bigwedge F\ \Gamma.(\exists\, xs.\ length\ xs = Suc\ (Suc\ i) \wedge F\ (xs\ @\ \Gamma)) = (\exists\, x.\ (\exists\, xs.\ length\ xs = i+1 \wedge F\ (xs\ @\ x\#\Gamma)))$
apply(*safe*)
subgoal for *F* Γ *xs*
apply(*rule exI*[**where** *x*=*nth xs* (*i*+*1*)])
apply(*rule exI*[**where** *x*=*take* (*i*+*1*) *xs*]) **apply** *auto*
by (*metis Cons-nth-drop-Suc append.right-neutral append-Cons append-assoc append-eq-conv-conj append-self-conv2 append-take-drop-id lessI*)
subgoal for *F* Γ *x xs*
apply(*rule exI*[**where** *x*=*xs*@[*x*]])
by *auto*
done
have *h2* : $\bigwedge X\ \Gamma\ amount.\ amount \leq Suc\ i \Longrightarrow ((\exists\, xs.\ length\ xs = Suc\ (Suc\ i) \wedge$
$(\exists\, (al, fl) \in set\ (dnf\ X).$
$eval\ (superPicker\ amount\ i\ al\ fl)\ (xs\ @\ \Gamma)))$
$= (\exists\, xs.\ length\ xs = Suc\ (Suc\ i) \wedge$
$(\exists\, (al, fl) \in set\ (dnf\ X).$
$(\forall\, a \in set\ al.\ aEval\ a\ (xs@\Gamma)) \wedge (\forall\, f \in set\ fl.\ eval\ f\ (xs@\Gamma)))))$
subgoal for *X* Γ *amount*
unfolding *rearangeExists*
apply(*rule bex-cong*)
apply *simp*
subgoal for *x*
apply (*cases x*)
apply *simp*
subgoal for *al fl*
unfolding *dropTheEnd*
unfolding *dropTheEnd*[*of* $\lambda xs.\ (\forall\, a \in set\ al.\ aEval\ a\ xs) \wedge (\forall\, f \in set\ fl.\ eval\ f\ xs)$]
apply *simp*
unfolding *ih*[*of amount al fl* -#Γ,*symmetric*]
unfolding *eval-list-conj*
apply(*rule ex-cong1*)
subgoal for *xa*
apply(*rule ex-cong1*)
subgoal for *xab* **apply** *auto*
by (*meson Un-iff eval.simps*(*1*) *image-eqI*)
done
done
done
done
done
have *h3* : $\bigwedge L\ F.\ (\exists\, xs.\ length\ xs = Suc\ (Suc\ i) \wedge eval\ (step\ (Suc\ i)\ L\ F)\ (xs@\Gamma)) = (\exists\, xs.\ length\ xs = Suc\ (Suc\ i) \wedge eval\ (list\text{-}conj\ (map\ fm.Atom\ L\ @\ F))\ (xs\ @\ \Gamma))$
subgoal for *L F*

```
          using pickedOneOfThem[OF h0]
           using  qe-eq-repeat-eval'[of Suc i L F Γ] gen-qe-eval'[of Suc i L F Γ]
luckyFind-eval'[of Suc i L F Γ]
          by auto
        done
      have heurange : var≤ Suc i using rangeHeuristic[OF h0] by auto
      show ?thesis
        unfolding eval-list-disj
        unfolding h1
        unfolding h2[OF amountPred]
        unfolding dnf-eval
        unfolding opt'
        unfolding h3
      proof(safe)
        fix xs
       assume h : length xs = Suc (Suc i) eval (list-conj (map fm.Atom L @ F))
(xs @ Γ)
        have h3 : var < length (xs @ Γ)  using h heurange by auto
       have h1: (swap-list (Suc i) var (xs @ Γ)) = (swap-list (Suc i) var xs @ Γ)
          using h(1) heurange apply simp
          by (simp add: list-update-append nth-append)
        have h2 : Suc i < length (xs @ Γ) using h by auto

        show ∃ xs. length xs = Suc (Suc i) ∧
            eval (list-conj (map fm.Atom (map (swap-atom (Suc i) var) L) @ map
(swap-fm (Suc i) var) F)) (xs @ Γ)
          apply(rule exI[where x=swap-list (Suc i) var xs])
          apply(auto simp add:h eval-list-conj simp del:swap-list.simps)
          apply(simp add: h)
          using swap-fm[OF h2 h3] swap-atom[OF h2 h3] unfolding h1
          using h(2) unfolding eval-list-conj
          apply auto

          by (meson Un-iff eval.simps(1) imageI)
      next
        fix xs
         assume h : length xs = Suc (Suc i)eval (list-conj (map fm.Atom (map
(swap-atom (Suc i) var) L) @ map (swap-fm (Suc i) var) F)) (xs @ Γ)
        have h3 : var < length (swap-list (Suc i) var xs @ Γ)  using h heurange
by auto
        have h1: swap-list (Suc i) var (swap-list (Suc i) var xs @ Γ) = xs @ Γ
          apply auto
          using h(1) heurange
           by (smt (z3) le-imp-less-Suc length-list-update lessI list-update-append
list-update-id list-update-overwrite list-update-swap nth-append nth-list-update-eq)
        have h2 : Suc i < length (swap-list (Suc i) var xs @ Γ) using h by auto
         show ∃ xs. length xs = Suc (Suc i) ∧ eval (list-conj (map fm.Atom L @
F)) (xs @ Γ)
          apply(rule exI[where x=swap-list (Suc i) var xs])
```

```
          apply(auto simp add:eval-list-conj simp del:swap-list.simps)
          apply(simp add: h)
          unfolding swap-fm[OF h2 h3] swap-atom[OF h2 h3]
          unfolding h1
          using h(2) unfolding eval-list-conj
          apply auto
          apply (meson Un-iff eval.simps(1) imageI)
          done
       qed
     qed
     done
   done
qed


lemma brownHueristic-less-than: brownsHeuristic n L F = var ⟹ var≤ n
  apply simp
  apply(cases sorted-list-of-fset
         ((λx. case foldl
                    (λ(maxdeg, totaldeg, appearancecount) l.
                       let deg = MPoly-Type.degree (case l of Less p ⇒ p | Eq p ⇒
p | Leq p ⇒ p | Neq p ⇒ p) x
                        in (max maxdeg deg, totaldeg + deg, appearancecount + (if
0 < deg then 1 else 0)))
                    (0, 0, 0) L of
               (a, b, c) ⇒ Quad (a, b, c, x)) |`|
          fset-of-list [0..<n]))
  apply auto
  subgoal for a apply(cases a)
    by auto
  done
end
```

10.4 Top-Level Algorithm Proofs

```
theory ExportProofs
  imports HeuristicProofs Exports

   HOL.String HOL−Library.Code-Target-Int HOL−Library.Code-Target-Nat Pret-
tyPrinting Show.Show-Real
begin


theorem eval (Unpower f) L = eval f L unfolding unpower-eval Unpower-def by
auto


theorem VSLuckiest: ∀ xs. eval (VSLuckiest φ) xs = eval φ xs
  unfolding VSLuckiest-def Unpower-def opt-def
```

using *QE-dnf-eval*[*OF luckiestFind-eval′ opt-no-group*] *opt-no-group*
by *fastforce*

theorem *VSLuckiestBlocks* : ∀ *xs*. *eval* (*VSLuckiestBlocks* φ) *xs* = *eval* φ *xs*
unfolding *VSLuckiestBlocks-def Unpower-def opt-group-def*
using *QE-dnf′-eval*[*OF the-real-step-augment*[*OF luckiestFind-eval*, *of* λ*x* - -. *x*] *opt*]
using *opt*
by *fastforce*

theorem *VSEquality* : ∀ *xs*. *eval* (*VSEquality* φ) *xs* = *eval* φ *xs*
unfolding *VSEquality-def Unpower-def opt-def*
using *QE-dnf-eval*[*OF qe-eq-repeat-eval′ opt-no-group*]
using *opt-no-group VSLuckiest*
by *fastforce*

theorem *VSEqualityBlocks* : ∀ *xs*. *eval* (*VSEqualityBlocks* φ) *xs* = *eval* φ *xs*
unfolding *VSEqualityBlocks-def Unpower-def opt-group-def*
using *QE-dnf′-eval*[*OF the-real-step-augment*[*OF qe-eq-repeat-eval*, *of* λ*x* - -. *x*] *opt*]
using *opt VSLuckiestBlocks*
by *fastforce*

theorem *VSGeneralBlocks* : ∀ *xs*. *eval* (*VSGeneralBlocks* φ) *xs* = *eval* φ *xs*
unfolding *VSGeneralBlocks-def Unpower-def opt-group-def*
using *QE-dnf′-eval*[*OF the-real-step-augment*[*OF gen-qe-eval*, *of* λ*x* - -. *x*] *opt*]
using *opt VSLuckiestBlocks*
by *fastforce*

theorem *VSLuckyBlocks* : ∀ *xs*. *eval* (*VSLuckyBlocks* φ) *xs* = *eval* φ *xs*
unfolding *VSLuckyBlocks-def Unpower-def opt-group-def*
using *QE-dnf′-eval*[*OF the-real-step-augment*[*OF luckyFind′-eval*, *of* λ*x* - -. *x*] *opt*]
using *opt VSLuckiestBlocks*
by *fastforce*

theorem *VSLEGBlocks* : ∀ *xs*. *eval* (*VSLEGBlocks* φ) *xs* = *eval* φ *xs*
unfolding *VSLEGBlocks-def opt-group-def*
using *VSEqualityBlocks VSGeneralBlocks VSLuckyBlocks*
by *fastforce*

theorem *VSEqualityBlocksLimited* : ∀ *xs*. *eval* (*VSEqualityBlocksLimited* φ) *xs* = *eval* φ *xs*
unfolding *VSEqualityBlocksLimited-def Unpower-def opt-group-def*
using *QE-dnf-eval*[*OF qe-eq-repeat-eval-augment opt*] *opt VSLuckiestBlocks*
by *fastforce*

theorem *VSEquality-3-times* : $\forall$ *xs. eval* (*VSEquality-3-times* φ) *xs* = *eval* φ *xs*
using *VSEquality* **unfolding** *VSEquality-3-times-def* **by** *auto*

theorem *VSGeneral*: $\forall$ *xs. eval* (*VSGeneral* φ) *xs* = *eval* φ *xs*
unfolding *VSGeneral-def Unpower-def Unpower-def opt-def*
using *QE-dnf-eval*[*OF gen-qe-eval*′ *opt-no-group*]
using *opt-no-group VSLuckiest*
by *fastforce*

theorem *VSGeneralBlocksLimited*: $\forall$ *xs. eval* (*VSGeneralBlocksLimited* φ) *xs* = *eval* φ *xs*
unfolding *VSGeneralBlocksLimited-def Unpower-def opt-group-def*
using *QE-dnf-eval*[*OF gen-qe-eval-augment opt*] *opt VSLuckiestBlocks*
by *fastforce*

theorem *VSBrowns*: $\forall$ *xs. eval* (*VSBrowns* φ) *xs* = *eval* φ *xs*
unfolding *VSBrowns-def Unpower-def opt-group-def*
using *QE-dnf-eval*[*OF step-augmenter-eval*[*of gen-qe brownsHeuristic*, *OF gen-qe-eval brownHueristic-less-than*] *opt*] *opt VSLuckiestBlocks*
by *fastforce*

theorem *VSGeneral-3-times* : $\forall$ *xs. eval* (*VSGeneral-3-times* φ) *xs* = *eval* φ *xs*
unfolding *VSGeneral-3-times-def* **using** *VSGeneral*
by *auto*

theorem *VSLucky*: $\forall$ *xs. eval* (*VSLucky* φ) *xs* = *eval* φ *xs*
unfolding *VSLucky-def Unpower-def opt-def*
using *QE-dnf-eval*[*OF luckyFind-eval*′ *opt-no-group*] *opt-no-group VSLuckiest*
by *fastforce*

theorem *VSLuckyBlocksLimited*: $\forall$ *xs. eval* (*VSLuckyBlocksLimited* φ) *xs* = *eval* φ *xs*
unfolding *VSLuckyBlocksLimited-def Unpower-def opt-group-def*
using *QE-dnf-eval*[*OF luckyFind-eval-augment opt*] *opt VSLuckiestBlocks*
by *fastforce*

theorem *VSLEG*: $\forall$ *xs. eval* (*VSLEG* φ) *xs* = *eval* φ *xs*
unfolding *VSLEG-def*
using *VSLucky VSEquality VSGeneral* **by** *auto*

theorem *VSHeuristic* : $\forall$ *xs. eval*(*VSHeuristic* φ) *xs* = *eval* φ *xs*
unfolding *VSHeuristic-def Unpower-def opt-group-def*
using *QE-dnf-eval*[*OF superPicker-eval opt*] *opt VSLuckiestBlocks*
by *fastforce*

theorem *VSLuckiestRepeat* : $\forall$ *xs. eval* (*VSLuckiestRepeat* φ) *xs* = *eval* φ *xs*
unfolding *VSLuckiestRepeat-def* **using** *repeatAmountOfQuantifiers-eval*[*OF*] **us-**

ing *VSLuckiest*
by *blast*

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.