A First Complete Algorithm for Real Quantifier Elimination in Isabelle/HOL

Katherine Kosaian, Yong Kiam Tan, and André Platzer

March 17, 2025

Abstract

We formalize a multivariate quantifier elimination (QE) algorithm in the theorem prover Isabelle/HOL. Our algorithm is complete, in that it is able to reduce *any* quantified formula in the first-order logic of real arithmetic to a logically equivalent quantifier-free formula. The algorithm we formalize is a hybrid mixture of Tarski's original QE algorithm [8] and the Ben-Or, Kozen, and Reif [2] algorithm, and it is the first complete multivariate QE algorithm formalized in Isabelle/HOL.

Remark

This is the AFP entry associated with a corresponding paper by Kosaian, Tan, and Platzer [6]. Various auxiliary sources [1, 7] beyond the original BKR paper were helpful in the development of this AFP entry. The most closely related works are by Cyril Cohen and Assia Mahboubi [4, 3, 5].

Contents

theory *Multiv-Poly-Props*
imports
HOL−Computational-Algebra.Computational-Algebra
Polynomial-Interpolation.Ring-Hom-Poly
Virtual-Substitution.ExecutiblePolyProps
Sturm-Tarski.Pseudo-Remainder-Sequence

Factor-Algebraic-Polynomial.Poly-Connection
Virtual-Substitution.VSQuad

begin

1 Some definitions for lists of polynomials

```
abbreviation lead-coeffs:: 'a::zero Polynomial.poly list ⇒ 'a list
  where lead-coeffs p-list ≡ map Polynomial.lead-coeff p-list

definition coeffs-list:: 'a::zero Polynomial.poly list ⇒ 'a list
  where coeffs-list p-list ≡ concat (map Polynomial.coeffs p-list)

value lead-coeffs [[:((Var 0 +(Const (3::real))*((Var 1)^2)):: real mpoly), 0, (1::real
mpoly):]]

abbreviation degrees:: 'a::zero Polynomial.poly list ⇒ nat list
  where degrees polys ≡ map Polynomial.degree polys

value degrees [[:((Var 0 +(Const (3::real))*((Var 1)^2)):: real mpoly), 0, (1::real
mpoly):]]

fun variables:: real mpoly list ⇒ nat set
  where variables [] = {}
  | variables (h#T) = (vars h) ∪ (variables T)

fun variables-list:: real mpoly list ⇒ nat list
  where variables-list qs = sorted-list-of-set (variables qs)

lemma variables-prop:
  shows v ∈ variables qs ⟷ (∃ q ∈ set qs. v ∈ vars q)
proof (induct qs)
  case Nil
  then show ?case by auto
next
  case (Cons a qs)
  then show ?case by simp
qed

lemma variables-finite:
  shows finite (variables qs)
proof (induct qs)
  case Nil
  then show ?case by auto
next
  case (Cons a qs)
  then show ?case
    by (simp add: vars-finite)
qed

lemma variables-list-prop:
```

shows *v* ∈ *set* (*variables-list qs*) ⟷ (∃ *q* ∈ *set qs*. *v* ∈ *vars q*)
using *variables-finite*
by (*simp add*: *member-def variables-prop*)

2 Evaluating multivariate polynomials

definition *eval-mpoly*:: *real list* ⇒ *real mpoly* ⇒ *real*
where *eval-mpoly L p* = *insertion* (*nth-default 0 L*) *p*

value *eval-mpoly* [*4*, *1*, *2*] ((*Var 0* +(*Const* (*3*::*real*))∗((*Var 1*)^*2*)):: *real mpoly*)

definition *eval-mpoly-poly*:: *real list* ⇒ *real mpoly Polynomial.poly* ⇒ *real Polynomial.poly*
where *eval-mpoly-poly L p* = *map-poly* (*eval-mpoly L*) *p*

lemma *eval-mpoly-poly-coeff1*: *n* ≤ *Polynomial.degree* (*eval-mpoly-poly L p*) ⟹ *Polynomial.coeff* (*eval-mpoly-poly L p*) *n* = *eval-mpoly L* (*Polynomial.coeff p n*)
unfolding *eval-mpoly-poly-def*
by (*simp add*: *coeff-map-poly eval-mpoly-def*)

lemma *eval-mpoly-poly-coeff2*: ∀ *n* > *Polynomial.degree* (*eval-mpoly-poly L p*). *Polynomial.coeff* (*eval-mpoly-poly L p*) *n* = *0*
using *coeff-eq-0* **by** *auto*

value *eval-mpoly-poly* [*4*, *1*, *2*] [:((*Var 0* +(*Const* (*3*::*real*))∗((*Var 1*)^*2*)):: *real mpoly*), *0*, (*1*::*real mpoly*):]

definition *eval-mpoly-poly-list*:: *real list* ⇒ *real mpoly Polynomial.poly list* ⇒ *real Polynomial.poly list*
where *eval-mpoly-poly-list L p-list* = *map* (λ*x*. (*eval-mpoly-poly L x*)) *p-list*

interpretation *eval-mpoly-map-poly-comm-ring-hom*: *map-poly-comm-ring-hom eval-mpoly val*
apply (*unfold-locales*)
by (*auto simp add*: *eval-mpoly-def insertion-add insertion-mult*)

interpretation *eval-mpoly-map-poly-idom-hom*: *map-poly-idom-hom eval-mpoly val*..

interpretation *eval-mpoly-poly-comm-ring-hom*: *comm-ring-hom eval-mpoly-poly val*
apply *unfold-locales*
apply (*auto simp add*: *eval-mpoly-poly-def*)
using *eval-mpoly-map-poly-comm-ring-hom.base.map-poly-hom-add* **apply** *force*
using *eval-mpoly-map-poly-comm-ring-hom.hom-mult* **by** *auto*

interpretation *eval-mpoly-poly-map-poly-idom-hom*: *map-poly-idom-hom eval-mpoly-poly val*..

3 Removing highest degree monomial

definition *one-less-degree*:: *real mpoly Polynomial.poly* ⇒ *real mpoly Polynomial.poly*
where *one-less-degree p* = *p* − *Polynomial.monom* (*Polynomial.lead-coeff p*) (*Polynomial.degree p*)

lemma *one-less-degree-degree*:
assumes *Polynomial.degree p* > *0*
shows *Polynomial.degree*(*one-less-degree p*) < *Polynomial.degree p*
proof −
obtain *q* **where** *q*:*Polynomial.degree p* = *Suc q*
using *assms not0-implies-Suc* **by** *blast*
from *poly-as-sum-of-monoms*[*of p*]
have *p-is*: *p* = ($\sum$ *i*≤*q*. *Polynomial.monom* (*poly.coeff p i*) *i*) + *Polynomial.monom* (*Polynomial.lead-coeff p*) (*Polynomial.degree p*)
by (*simp add*: *q*)
then have *e*: *p* − *Polynomial.monom* (*Polynomial.lead-coeff p*) (*Polynomial.degree p*) = ($\sum$ *i*≤*q*. *Polynomial.monom* (*poly.coeff p i*) *i*)
using *diff-eq-eq* **by** *blast*
show *?thesis* **unfolding** *one-less-degree-def e*
by (*smt* (*z3*) *Polynomial.coeff-diff Polynomial.lead-coeff-monom Zero-not-Suc p-is cancel-comm-monoid-add-class.diff-cancel degree-0 degree-add-eq-left degree-monom-eq e leading-coeff-0-iff linorder-neqE-nat q*)
qed

lemma *sublist-prefix-property*:
assumes *length og-list* ≥ *length sub-list*
assumes ∀ *i* < *length sub-list*. *sub-list* ! *i* = *og-list* ! *i*
shows *Sublist.prefix sub-list og-list*
using *assms*
proof (*induct length sub-list arbitrary*: *sub-list*)
case *0*
then show *?case* **by** *auto*
next
case (*Suc x*)
then have ∃ *a l*. *sub-list* = *append l* [*a*]
by (*metis length-greater-0-conv rev-exhaust zero-less-Suc*)
then obtain *a l* **where** *al-prop*: *sub-list* = *append l* [*a*]
by *auto*
then have *length-l*: *List.length l* = *x*
using *Suc.hyps*(*2*) **by** *auto*
then have *prefix l og-list*
by (*metis Suc.hyps*(*1*) *Suc.hyps*(*2*) *Suc.prems*(*1*) *Suc.prems*(*2*) *Suc-leD al-prop butlast-snoc lessI nth-butlast order.strict-trans*)
then have ∃ *t*. *og-list* = *l* @ *t*
using *prefix-def* **by** *blast*
then obtain *t* **where** *t-prop*: *og-list* = *l* @ *t* **by** *auto*
then have *t-first*: *og-list* ! (*length sub-list* − *1*) = (*t* ! *0*)
by (*metis Suc.hyps*(*2*) *diff-Suc-1 diff-self-eq-0 length-l less-irrefl nth-append*)

have *sub-list* ! (*length sub-list* − *1*) = *a*
using *al-prop*
by *fastforce*
then have *og-list* ! (*length sub-list* − *1*) = *a*
using *Suc.hyps*(*2*) *Suc.prems* **by** *fastforce*
then have *t-zero*: *t* ! *0* = *a*
using *t-prop t-first* **by** *auto*
have *length l* = *length sub-list* − *1*
using *Suc.hyps*(*2*) *length-l* **by** *presburger*
then have *length t* > *0*
using *assms*
by (*metis Suc.hyps*(*2*) *Suc.prems*(*1*) *append.right-neutral bot-nat-0.not-eq-extremum diff-is-0-eq length-0-conv length-l lessI t-prop zero-less-diff*)
then have ∃ *t1*. *t* = *a* # *t1*
using *t-zero*
by (*metis length-0-conv less-nat-zero-code list.exhaust nth-Cons-0*)
then show *?case*
using *t-prop al-prop*
by *force*
qed

lemma *one-less-degree-is-prefix*:
assumes *Polynomial.degree q* > *0*
shows *Sublist.prefix* (*Polynomial.coeffs* (*one-less-degree q*)) (*Polynomial.coeffs q*)
proof −
let *?sub-list* = *Polynomial.coeffs* (*Multiv-Poly-Props.one-less-degree q*)
have ∀ *i* < *length ?sub-list*. *Polynomial.coeff* (*one-less-degree q*) *i* = *Polynomial.coeff q i*
by (*metis* (*no-types*, *lifting*) *Multiv-Poly-Props.one-less-degree-def Polynomial.coeff-diff Polynomial.coeff-monom assms coeffs-eq-Nil diff-zero length-0-conv length-coeffs-degree less-Suc-eq-0-disj not-less-eq one-less-degree-degree*)
then have ∀ *i* < *length ?sub-list*. *?sub-list* ! *i* = (*Polynomial.coeffs q*) ! *i*
unfolding *Polynomial.coeffs-def*
by (*smt* (*verit*, *ccfv-SIG*) *add-0 assms degree-0 diff-zero length-map length-upt less-Suc-eq less-nat-zero-code list.size*(*3*) *nth-map-upt one-less-degree-degree order.strict-trans*)

then show *?thesis*
using *sublist-prefix-property assms*
by (*smt* (*verit*, *del-insts*) *Suc-mono bot-nat-0.not-eq-extremum coeffs-eq-Nil degree-0 length-coeffs-degree less-imp-le-nat list.size*(*3*) *one-less-degree-degree order-less-subst1*)
qed

lemma *one-less-degree-is-strict-prefix*:
assumes *Polynomial.degree q* > *0*
shows *Sublist.strict-prefix* (*Polynomial.coeffs* (*Multiv-Poly-Props.one-less-degree q*)) (*Polynomial.coeffs q*)
proof −
have *length* (*Polynomial.coeffs* (*Multiv-Poly-Props.one-less-degree q*)) < *length*

(*Polynomial.coeffs q*)
  using *one-less-degree-degree*[*of q*]
    assms
 by (*metis coeffs-0-eq-Nil degree-0 length-coeffs-degree less-nat-zero-code list.size*(*3*) *not-less-eq*)
 then show *?thesis*
  using *one-less-degree-is-prefix assms*
  by (*metis less-irrefl-nat prefix-order.order-iff-strict*)
qed

lemma *coeff-one-less-degree-var*:
 assumes $0 < Polynomial.degree\ p$
 assumes *one-less-degree* $p \neq 0$
 shows $i \leq Polynomial.degree$ (*one-less-degree p*) $\Longrightarrow$
   poly.coeff p i = *poly.coeff* (*one-less-degree p*) *i*
proof −
 fix *i*
 assume *i-leq*: $i \leq Polynomial.degree$ (*one-less-degree p*)
 then have *i-lt*: *i* < *length* (*Polynomial.coeffs* (*Multiv-Poly-Props.one-less-degree p*))
  unfolding *Polynomial.coeffs-def* **using** *assms* **by** *auto*
 have *i-lt2*: *i* < (*Polynomial.degree p*)
  using *i-leq one-less-degree-degree*[*of p*] *assms*(*1*) **by** *auto*
 have *len-map1*: *length* (*map* (*poly.coeff p*) [*0..<Suc* (*Polynomial.degree p*)]) = *Polynomial.degree p* + *1*
  by *auto*
 have *len-map2*: *length* (*Polynomial.coeffs* (*Multiv-Poly-Props.one-less-degree p*))
  = *Polynomial.degree* (*one-less-degree p*) + *1*
  unfolding *Polynomial.coeffs-def* **using** *assms*(*2*) **by** *auto*
 have $p \neq 0$
  using *assms*(*1*) **by** *auto*
 then have *same-coeff*: *poly.coeff p i* = (*Polynomial.coeffs*) *p* ! *i*
  unfolding *Polynomial.coeffs-def* **using** *assms i-lt2 len-map1*
  by (*smt* (*verit*, *best*) *Suc-eq-plus1 add-0 le-simps*(*2*) *length-map less-imp-le-nat nth-map nth-upt*)
 obtain *zs* **where** *zs-prop*: *Polynomial.coeffs p* =
   (*Polynomial.coeffs* (*Multiv-Poly-Props.one-less-degree p*)) @ *zs*
  using *assms i-leq one-less-degree-is-prefix*[*of p*] **unfolding** *prefix-def*
  by *auto*
 have *same-coeff-2*: ((*Polynomial.coeffs* (*Multiv-Poly-Props.one-less-degree p*)) @ *zs*) ! *i* =
  (*Polynomial.coeffs* (*one-less-degree p*)) ! *i*
  using *i-lt*
  by (*simp add*: *nth-append*)
 show *poly.coeff p i* = *poly.coeff* (*one-less-degree p*) *i*
  using *same-coeff same-coeff-2 zs-prop*
  by (*simp add*: *i-lt nth-coeffs-coeff*)
qed

lemma *coeff-one-less-degree*:
assumes *one-less-degree p* $\neq$ *0*
shows $i \leq$ *Polynomial.degree* (*one-less-degree p*) $\Longrightarrow$
poly.coeff p i = *poly.coeff* (*one-less-degree p*) *i*
proof −
have *Polynomial.degree p* > *0*
using *assms*
by (*metis* (*no-types*, *lifting*) *Multiv-Poly-Props.one-less-degree-def Polynomial.coeff-diff Polynomial.coeff-monom diff-zero eq-iff-diff-eq-0 eq-iff-diff-eq-0 le-degree leading-coeff-0-iff linorder-not-less zero-less-iff-neq-zero*)
then show $\bigwedge i$. $i \leq$ *Polynomial.degree* (*one-less-degree p*) $\Longrightarrow$
poly.coeff p i = *poly.coeff* (*one-less-degree p*) *i*
using *coeff-one-less-degree-var assms*
by *auto*
qed

lemma *coeff-one-less-degree-subset*:
assumes *one-less-degree q* $\neq$ *0*
shows *set* (*Polynomial.coeffs* (*Multiv-Poly-Props.one-less-degree q*)) $\subseteq$ *set* (*Polynomial.coeffs q*)
proof *clarsimp*
fix *x*
assume $x \in$ *set* (*Polynomial.coeffs* (*Multiv-Poly-Props.one-less-degree q*))
then obtain *i* **where** *i-prop*: $i \leq$ *Polynomial.degree* (*one-less-degree q*)
x = *poly.coeff* (*one-less-degree q*) *i*
unfolding *Polynomial.coeffs-def* **using** *assms*
using *atMost-upto* **by** *auto*
have *Polynomial.degree q* > *0*
using *assms*
by (*metis* (*no-types*, *lifting*) *Multiv-Poly-Props.one-less-degree-def Polynomial.coeff-diff Polynomial.coeff-monom diff-zero eq-iff-diff-eq-0 eq-iff-diff-eq-0 le-degree leading-coeff-0-iff linorder-not-less zero-less-iff-neq-zero*)
then show $x \in$ *set* (*Polynomial.coeffs q*)
using *coeff-one-less-degree assms i-prop*
unfolding *Polynomial.coeffs-def* **using** *one-less-degree-degree*
by (*smt* (*verit*, *best*) *Polynomial.coeffs-def coeff-in-coeffs degree-0 le-trans less-imp-le-nat less-numeral-extra*(*3*))
qed

lemma *coeffs-between-one-less-degree*:
assumes *0* < *Polynomial.degree p*
assumes *igt*: *i* > *Polynomial.degree* (*one-less-degree p*)
assumes *ilt*: *i* < *Polynomial.degree p*
shows *poly.coeff p i* = *0*
using *assms*
using *Multiv-Poly-Props.one-less-degree-def coeff-eq-0* **by** *fastforce*

lemma *poly-p-altdef-one-less-degree*:

assumes *deg-gt*: *Polynomial.degree p* > *0*
assumes *deg-is*: *Polynomial.degree p* = *d*
shows *poly p x* = ($\sum$ *i*≤*Polynomial.degree* (*one-less-degree p*). *Polynomial.coeff* (*one-less-degree p*) *i* ∗ *x* ^ *i*)
+ (*Polynomial.coeff p d*)∗(*x*^*d*)
proof −
let *?lp* = (*one-less-degree p*)
let *?deg-lp* = *Polynomial.degree ?lp*
have *poly p x* = ($\sum$ *i*≤*d*. *Polynomial.coeff p i* ∗ *x* ^ *i*)
using *assms poly-altdef* **by** *auto*
then have *p-is*: *poly p x* = ($\sum$ *i*≤(*d* − *1*). *Polynomial.coeff p i* ∗ *x* ^ *i*) + (*Polynomial.coeff p d*)∗(*x*^*d*)
using *deg-gt*
by (*metis* (*no-types*, *lifting*) *One-nat-def Suc-pred assms*(*2*) *sum.atMost-Suc*)
have *well-def*: *?deg-lp* ≤ *d* − *1*
using *one-less-degree-degree deg-gt deg-is*
by (*metis One-nat-def Suc-pred less-Suc-eq-le*)
then have *sum1*: ($\sum$ *i*≤(*d* − *1*). *Polynomial.coeff p i* ∗ *x* ^ *i*) = ($\sum$ *i*≤*Polynomial.degree ?lp*. *Polynomial.coeff p i* ∗ *x* ^ *i*)
proof −
have ⋀*i*. (*i* > *Polynomial.degree ?lp* ∧ *i* ≤ *d* − *1*) ⟹ *Polynomial.coeff p i* = *0*
using *deg-is coeffs-between-one-less-degree*
by (*metis One-nat-def Suc-pred deg-gt less-Suc-eq-le*)
then show *?thesis* **using** *well-def*
proof (*induct d* − *1* − *?deg-lp arbitrary*: *d*)
case *0*
then show *?case*
by (*metis* (*no-types*, *lifting*) *diff-is-0-eq le-antisym*)
next
case (*Suc xa*)
then have *sum*: ($\sum$ *i*≤*d* − *1*. *poly.coeff p i* ∗ *x* ^ *i*) =
($\sum$ *i*≤*d* − *2*. *poly.coeff p i* ∗ *x* ^ *i*) + (*poly.coeff p* (*d*−*1*) ∗ *x* ^ (*d*−*1*))
using *Nat.diff-diff-right One-nat-def Suc-le-mono Suc-pred add-diff-cancel-right′ bot-nat-0.not-eq-extremum diff-is-0-eq le-numeral-extra*(*4*) *nat-1-add-1 sum.atMost-Suc zero-le*
by (*smt* (*verit*, *del-insts*))
have *h0*: *xa* = *d* − *1* − *1* − *Polynomial.degree* (*Multiv-Poly-Props.one-less-degree p*)
using *Suc.hyps*(*2*) **by** *auto*
have *h1*: (⋀*i*. *Polynomial.degree* (*Multiv-Poly-Props.one-less-degree p*) < *i* ∧
i ≤ *d* − *1* − *1* ⟹
poly.coeff p i = *0*)
using *Suc.prems*(*1*) **by** *auto*
have *h2*: *Polynomial.degree* (*Multiv-Poly-Props.one-less-degree p*) ≤ *d* − *1* − *1*
using *Suc.prems Suc.hyps*(*2*) **by** *auto*
have *eq1*: ($\sum$ *i*≤*d* − *2*. *poly.coeff p i* ∗ *x* ^ *i*) = ($\sum$ *i*≤*Polynomial.degree ?lp*. *Polynomial.coeff p i* ∗ *x* ^ *i*)

using *Suc.hyps*(*1*)[*OF h0 h1 h2*]
by (*metis* (*no-types*, *lifting*) *diff-diff-left one-add-one*)
have *dgt*: *d* − *1* > *Polynomial.degree* (*one-less-degree p*)
using *Suc.prems Suc.hyps*(*2*) **by** *auto*
have *eq2*: *poly.coeff p* (*d*−*1*) = *0*
using *Suc.prems*
using *dgt* **by** *blast*
then show *?case* **using** *sum eq1 eq2*
by *fastforce*
qed
qed
have *sum2*: (∑ *i*≤*Polynomial.degree* (*one-less-degree p*). *Polynomial.coeff p i* ∗ *x* ^ *i*)
= (∑ *i*≤*Polynomial.degree* (*one-less-degree p*). *Polynomial.coeff* (*one-less-degree p*) *i* ∗ *x* ^ *i*)
using *coeff-one-less-degree*
by (*smt* (*verit*, *del-insts*) *Multiv-Poly-Props.one-less-degree-def Polynomial.coeff-monom atMost-iff bot-nat-0.extremum-uniqueI deg-gt degree-0 eq-iff-diff-eq-0 leading-coeff-0-iff nat-neq-iff sum.cong*)
then show *?thesis*
using *p-is sum1 sum2*
by *presburger*
qed

4 Expressing as univariate

definition *transform*:: *real mpoly list* ⇒ *real mpoly Polynomial.poly list*
where *transform qs* = (*let vs* = *variables-list qs in*
map (λ*q*. (*mpoly-to-mpoly-poly* (*nth vs* (*length vs* − *1*)) *q*)) *qs*)

definition *mpoly-to-mpoly-poly-alt* :: *nat* ⇒ *'a* :: *comm-ring-1 mpoly* ⇒ *'a mpoly Polynomial.poly* **where**
mpoly-to-mpoly-poly-alt x p = (∑ *i*∈{*0*..*MPoly-Type.degree p x*} .
Polynomial.monom (*isolate-variable-sparse p x i*) *i*)

definition *univariate-in*:: *real mpoly list* ⇒ *nat* ⇒ *real mpoly Polynomial.poly list*
where *univariate-in qs i* = *map* (*mpoly-to-mpoly-poly-alt i*) *qs*

lemma *degree-less-sum-max*:
shows *MPoly-Type.degree* (*p*+*q*) *var* ≤ *max* (*MPoly-Type.degree p var*) (*MPoly-Type.degree q var*)
by (*simp add*: *degree-add-leq*)

lemma *mpoly-to-mpoly-poly-alt-sum-aux* :
shows (∑ *i* = *0*..*b*.
Polynomial.monom (*isolate-variable-sparse* (*p* + *q*) *x i*) *i*) =
(∑ *i* = *0*..*b*. *Polynomial.monom* (*isolate-variable-sparse p x i*) *i*) +
(∑ *i* = *0*..*b*. *Polynomial.monom* (*isolate-variable-sparse q x i*) *i*)
proof (*induct b*)

case *0*
then show *?case* **using** *isovarspar-sum*[*of p q x 0*]
by (*simp add*: *add-monom*)
next
case (*Suc x*)
then show *?case*
using *isovarspar-sum*[*of p q x Suc x*]
proof −
have *f1*: ($\sum$ *n* = *0*..*x*. *Polynomial.monom* (*isolate-variable-sparse* (*p* + *q*) *x n*) *n*) = ($\sum$ *n* = *0*..*x*. *Polynomial.monom* (*isolate-variable-sparse p x n*) *n* + *Polynomial.monom* (*isolate-variable-sparse q x n*) *n*)
by (*simp add*: *isovarspar-sum monom-hom.hom-add*)
have *Polynomial.monom* (*isolate-variable-sparse* (*p* + *q*) *x* (*Suc x*)) (*Suc x*) = *Polynomial.monom* (*isolate-variable-sparse p x* (*Suc x*)) (*Suc x*) + *Polynomial.monom* (*isolate-variable-sparse q x* (*Suc x*)) (*Suc x*)
by (*simp add*: *add-monom isovarspar-sum*)
then have ($\sum$ *n* = *0*..*x*. *Polynomial.monom* (*isolate-variable-sparse* (*p* + *q*) *x n*) *n*) + *Polynomial.monom* (*isolate-variable-sparse* (*p* + *q*) *x* (*Suc x*)) (*Suc x*) = ($\sum$ *n* = *0*..*x*. *Polynomial.monom* (*isolate-variable-sparse p x n*) *n* + *Polynomial.monom* (*isolate-variable-sparse q x n*) *n*) + (*Polynomial.monom* (*isolate-variable-sparse p x* (*Suc x*)) (*Suc x*) + *Polynomial.monom* (*isolate-variable-sparse q x* (*Suc x*)) (*Suc x*))
using *f1* **by** *presburger*
then have ($\sum$ *n* = *0*..*Suc x*. *Polynomial.monom* (*isolate-variable-sparse* (*p* + *q*) *x n*) *n*) = ($\sum$ *n* = *0*..*x*. *Polynomial.monom* (*isolate-variable-sparse p x n*) *n* + *Polynomial.monom* (*isolate-variable-sparse q x n*) *n*) + (*Polynomial.monom* (*isolate-variable-sparse p x* (*Suc x*)) (*Suc x*) + *Polynomial.monom* (*isolate-variable-sparse q x* (*Suc x*)) (*Suc x*))
by (*smt* (*z3*) *sum.atLeast0-atMost-Suc*)
then have ($\sum$ *n* = *0*..*Suc x*. *Polynomial.monom* (*isolate-variable-sparse* (*p* + *q*) *x n*) *n*) = ($\sum$ *n* = *0*..*Suc x*. *Polynomial.monom* (*isolate-variable-sparse p x n*) *n* + *Polynomial.monom* (*isolate-variable-sparse q x n*) *n*)
by (*smt* (*z3*) *sum.atLeast0-atMost-Suc*)
then show *?thesis*
by (*smt* (*z3*) *Suc add-monom isovarspar-sum sum.atLeast0-atMost-Suc sum.distrib*)
qed
qed

lemma *isovar-sum-to-higher-degree*:
assumes *b* ≥ (*MPoly-Type.degree p x*)
shows ($\sum$ *i* = *0*..(*MPoly-Type.degree p x*). *Polynomial.monom* (*isolate-variable-sparse p x i*) *i*) =
($\sum$ *i* = *0*..*b*. *Polynomial.monom* (*isolate-variable-sparse p x i*) *i*)
using *assms*
proof (*induct b*)
case *0*
then show *?case*
by *auto*
next

case (*Suc b*)
then show *?case* **using** *isovar-greater-degree*
by (*smt* (*z3*) *Orderings.order-eq-iff add.commute add-0 isolate-variable-sparse-ne-zeroD le-SucE monom-hom.hom-zero sum.atLeast0-atMost-Suc*)
qed

lemma *mpoly-to-mpoly-poly-alt-sum* :
shows *mpoly-to-mpoly-poly-alt x* (*p*+*q*) = (*mpoly-to-mpoly-poly-alt x p*) + (*mpoly-to-mpoly-poly-alt x q*)
proof −
let *?deg* = *max* (*MPoly-Type.degree p x*) (*MPoly-Type.degree q x*)
have ($\sum$ *i* = *0*..*MPoly-Type.degree* (*p* + *q*) *x*.
Polynomial.monom (*isolate-variable-sparse* (*p* + *q*) *x i*) *i*) =
($\sum$ *i* = *0*..*MPoly-Type.degree p x*. *Polynomial.monom* (*isolate-variable-sparse p x i*) *i*) +
($\sum$ *i* = *0*..*MPoly-Type.degree q x*. *Polynomial.monom* (*isolate-variable-sparse q x i*) *i*)
proof (*induct ?deg arbitrary*: *p q*)
case *0*
then have *MPoly-Type.degree* (*p* + *q*) *x* = *0* ∧ *MPoly-Type.degree p x* = *0* ∧ *MPoly-Type.degree q x* = *0*
using *degree-add-leq*[*of p x 0 q*]
by (*metis le-zero-eq max.cobounded1 max.cobounded2*)
then show *?case*
using *isovarspar-sum*[*of p q x*]
by (*simp add*: *add-monom*)
next
case (*Suc xa*)
let *?deg-sum* = *MPoly-Type.degree* (*p*+*q*) *x*
let *?deg-p* = (*MPoly-Type.degree p x*)
let *?deg-q* = (*MPoly-Type.degree q x*)
have *?deg-sum* = *max ?deg-p ?deg-q* ∨ *?deg-sum* < *max ?deg-p ?deg-q*
using *degree-less-sum-max*[*of p q x*] **by** *auto*
moreover {
assume ∗: *?deg-sum* = *max ?deg-p ?deg-q*
have *eq1*: ($\sum$ *i* = *0*..*?deg-p*. *Polynomial.monom* (*isolate-variable-sparse p x i*) *i*) =
($\sum$ *i* = *0*..*?deg-sum*. *Polynomial.monom* (*isolate-variable-sparse p x i*) *i*)
using ∗ *isovar-sum-to-higher-degree*
by (*simp add*: *isovar-sum-to-higher-degree*)
have *eq2*: ($\sum$ *i* = *0*..*?deg-q*. *Polynomial.monom* (*isolate-variable-sparse q x i*) *i*) =
($\sum$ *i* = *0*..*?deg-sum*. *Polynomial.monom* (*isolate-variable-sparse q x i*) *i*)
using ∗ *isovar-sum-to-higher-degree*
by (*simp add*: *isovar-sum-to-higher-degree*)
then have ($\sum$ *i* = *0*..*?deg-sum*.
Polynomial.monom (*isolate-variable-sparse* (*p* + *q*) *x i*) *i*) =
($\sum$ *i* = *0*..*?deg-p*. *Polynomial.monom* (*isolate-variable-sparse p x i*) *i*) +
($\sum$ *i* = *0*..*?deg-q*. *Polynomial.monom* (*isolate-variable-sparse q x i*) *i*)

by (*simp add*: *eq1 eq2 mpoly-to-mpoly-poly-alt-sum-aux*)
}
moreover {
assume ∗: *?deg-sum* < *max ?deg-p ?deg-q*
let *?mx* = *max ?deg-p ?deg-q*
have *eq1*: ($\sum$ *i* = *0..?deg-p*. *Polynomial.monom* (*isolate-variable-sparse p x i*) *i*) =
($\sum$ *i* = *0..?mx*. *Polynomial.monom* (*isolate-variable-sparse p x i*) *i*)
using ∗ *isovar-sum-to-higher-degree*
by (*simp add*: *isovar-sum-to-higher-degree*)
have *eq2*: ($\sum$ *i* = *0..?deg-q*. *Polynomial.monom* (*isolate-variable-sparse q x i*) *i*) =
($\sum$ *i* = *0..?mx*. *Polynomial.monom* (*isolate-variable-sparse q x i*) *i*)
using ∗ *isovar-sum-to-higher-degree*
by (*simp add*: *isovar-sum-to-higher-degree*)
have *eq3*: ($\sum$ *i* = *0..?deg-sum*. *Polynomial.monom* (*isolate-variable-sparse* (*p* + *q*) *x i*) *i*) =
($\sum$ *i* = *0..?mx*. *Polynomial.monom* (*isolate-variable-sparse* (*p* + *q*) *x i*) *i*)
using ∗ *isovar-sum-to-higher-degree*
by (*simp add*: *isovar-sum-to-higher-degree*)
then have ($\sum$ *i* = *0..?deg-sum*.
Polynomial.monom (*isolate-variable-sparse* (*p* + *q*) *x i*) *i*) =
($\sum$ *i* = *0..?deg-p*. *Polynomial.monom* (*isolate-variable-sparse p x i*) *i*) +
($\sum$ *i* = *0..?deg-q*. *Polynomial.monom* (*isolate-variable-sparse q x i*) *i*)
using *mpoly-to-mpoly-poly-alt-sum-aux eq1 eq2 eq3*
by *auto*
}
ultimately have ($\sum$ *i* = *0..?deg-sum*.
Polynomial.monom (*isolate-variable-sparse* (*p* + *q*) *x i*) *i*) =
($\sum$ *i* = *0..?deg-p*. *Polynomial.monom* (*isolate-variable-sparse p x i*) *i*) +
($\sum$ *i* = *0..?deg-q*. *Polynomial.monom* (*isolate-variable-sparse q x i*) *i*)
by *fastforce*
then show *?case*
by *fastforce*
qed
then show *?thesis*
by (*simp add*: *mpoly-to-mpoly-poly-alt-def*)
qed

lemma *multivar-as-univar*: *mpoly-to-mpoly-poly-alt x p* = *mpoly-to-mpoly-poly x p*
proof (*induct p rule*: *mpoly-induct*)
case (*monom m a*)
have *a* = *0* ∨ *a* ≠ *0* **by** *auto*
moreover {
assume ∗: *a* = *0*
then have *mpoly-to-mpoly-poly-alt x* (*MPoly-Type.monom m a*) =
mpoly-to-mpoly-poly x (*MPoly-Type.monom m a*)
using *mpoly-to-mpoly-poly-monom*[*of x m a*]
unfolding *mpoly-to-mpoly-poly-alt-def*

isolate-variable-sparse-monom[*of m a*]
by *auto*
}
moreover {
assume ∗: $a \neq 0$
then have *monomials* (*MPoly-Type.monom m a*) = {*m*}
using *MPolyExtension.monomials-monom* **by** *auto*
then have (*lookup m x*) = *MPoly-Type.degree* (*MPoly-Type.monom m a*) *x*
using *degree-eq-iff*[*of* (*MPoly-Type.monom m a*) *x*]
by (*simp add*: ∗ *degree-monom*)
then have *h1*: ($\sum$ *i* = *0*..*MPoly-Type.degree* (*MPoly-Type.monom m a*) *x*.
Polynomial.monom (*isolate-variable-sparse* (*MPoly-Type.monom m a*) *x i*)
i) =
($\sum$ *i* = *0*..(*lookup m x*).
Polynomial.monom (*isolate-variable-sparse* (*MPoly-Type.monom m a*) *x i*)
i)
by *auto*
have $\bigwedge$*i*. *i* < (*lookup m x*) $\Longrightarrow$
Polynomial.monom (*isolate-variable-sparse* (*MPoly-Type.monom m a*) *x i*) *i* =
0
using *isolate-variable-sparse-monom*[*of m a*]
by *auto*
then have ($\sum$ *i* = *0*..<(*lookup m x*).
Polynomial.monom (*isolate-variable-sparse* (*MPoly-Type.monom m a*) *x i*)
i) = *0*
by *simp*
then have *h2*: ($\sum$ *i* = *0*..(*lookup m x*).
Polynomial.monom (*isolate-variable-sparse* (*MPoly-Type.monom m a*) *x i*) *i*)
= *Polynomial.monom* (*isolate-variable-sparse* (*MPoly-Type.monom m a*) *x* (*lookup*
m x))
(*lookup m x*)
by (*simp add*: *sum.head-if*)
have *k1*: ($\sum$ *i* = *0*..*MPoly-Type.degree* (*MPoly-Type.monom m a*) *x*.
Polynomial.monom (*isolate-variable-sparse* (*MPoly-Type.monom m a*) *x i*)
i) =
Polynomial.monom (*isolate-variable-sparse* (*MPoly-Type.monom m a*) *x*
(*lookup m x*))
(*lookup m x*)
using *h1 h2* **by** *auto*
have *Abs-poly-mapping* ((*lookup m*)(*x* := *0*)) =
Abs-poly-mapping (λ*k*. *lookup m k when k* ≠ *x*)
by (*metis* (*full-types*) *when*(*1*) *when*(*2*) *fun-upd-apply*)
then have (*Poly-Mapping.update x 0 m*) = (*remove-key x m*)
unfolding *remove-key-def update-def*
by (*auto*)
then have *MPoly-Type.monom* (*Poly-Mapping.update x 0 m*) *a* = *MPoly-Type.monom*
(*remove-key x m*) *a*
by *auto*
then have *isolate-variable-sparse* (*MPoly-Type.monom m a*) *x* (*lookup m x*) =

MPoly-Type.monom (*remove-key x m*) *a*
using *ExecutiblePolyProps.isolate-variable-sparse-monom*[*of m a x* (*lookup m x*)] ∗
by *auto*
then have *k2*: *Polynomial.monom* (*isolate-variable-sparse* (*MPoly-Type.monom m a*) *x* (*lookup m x*)) (*lookup m x*) =
Polynomial.monom (*MPoly-Type.monom* (*remove-key x m*) *a*) (*lookup m x*)
by *auto*
have *mpoly-to-mpoly-poly-alt x* (*MPoly-Type.monom m a*) =
mpoly-to-mpoly-poly x (*MPoly-Type.monom m a*) **unfolding** *mpoly-to-mpoly-poly-alt-def*

using *k1 k2 mpoly-to-mpoly-poly-monom*[*of x m a*]
by *auto*
}
ultimately have *mpoly-to-mpoly-poly-alt x* (*MPoly-Type.monom m a*) =
mpoly-to-mpoly-poly x (*MPoly-Type.monom m a*)
by *blast*
then show *?case*
by *blast*
next
case (*sum p1 p2 m a*)
then show *?case*
using *mpoly-to-mpoly-poly-add*[*of x p1 p2*] *mpoly-to-mpoly-poly-alt-sum*[*of x p1 p2*]
by *auto*
qed

5 Same mpoly eval means same polynomials

lemma *var-in-some-coeff*:
fixes *p*::*real mpoly Polynomial.poly*
fixes *w*::*real mpoly*
assumes $x \in$ *vars* ((*poly p w*)::*real mpoly*)
shows $x \in$ *vars w* $\vee$ ($\exists$ *i*. $x \in$ *vars* (*poly.coeff p i*))
using *assms*
proof (*induct Polynomial.degree p arbitrary*: *p rule*: *less-induct*)
case *less*
{assume ∗: *Polynomial.degree p* = *0*
then have $x \in$ *vars w* $\vee$ ($\exists$ *i*. $x \in$ *vars* (*poly.coeff p i*))
using *poly-altdef*[*of p w*]
using *local.less*(*2*) **by** *force*
} moreover
{assume ∗: *Polynomial.degree p* > *0*
then obtain *xa* **where** *deg-is*: *Polynomial.degree p* = *xa* + *1*
by (*metis Suc-eq-plus1 less-numeral-extra*(*3*) *not0-implies-Suc*)
then have *poly-p-w*: *poly p w* = ($\sum$ *i*≤*xa*. *poly.coeff p i* ∗ *w* ^ *i*) + (*poly.coeff p* (*xa* + *1*) ∗ *w* ^ (*xa* + *1*))
using *poly-altdef*[*of p w*]
by (*metis* (*no-types*, *lifting*) *Suc-eq-plus1 sum.atMost-Suc*)

```
    then have xin: x ∈ vars (∑ i≤xa. poly.coeff p i * w ^ i) ∨ x ∈ vars (poly.coeff
p (xa + 1) * w ^ (xa + 1))
      using vars-add
      using local.less(2) by auto
    let ?q = one-less-degree p
    have less-deg: (poly (one-less-degree p) w) = (∑ i≤xa. poly.coeff p i * w ^ i)
      using coeff-one-less-degree poly-altdef deg-is
      by (smt (verit, best) local.less(2) Suc-eq-plus1 poly-p-w add-diff-cancel-right'
poly-p-altdef-one-less-degree sum.cong zero-less-Suc)
    {assume **: x ∈ vars (poly.coeff p (xa + 1) * w ^ (xa + 1))
      then have x ∈ vars w ∨ (∃ i. x ∈ vars (poly.coeff p i))
        using vars-mult[of poly.coeff p (xa + 1) w ^ (xa + 1)]
        by (meson not-in-mult not-in-pow)
    }
    moreover {assume **:x ∈ vars (∑ i≤xa. poly.coeff p i * w ^ i)
      then have x ∈ vars (poly (one-less-degree p) w)
        using less-deg by auto
      then have x ∈ vars w ∨ (∃ i. x ∈ vars (poly.coeff (one-less-degree p) i))
        using less.hyps one-less-degree-degree *
        by simp
      then have x ∈ vars w ∨ (∃ i. x ∈ vars (poly.coeff p i))
        using coeff-one-less-degree
        by (metis coeff-eq-0 le-degree lessI zero-poly.rep-eq)
    }
    ultimately have  x ∈ vars w ∨ (∃ i. x ∈ vars (poly.coeff p i))
      using xin
      by auto
  } ultimately show ?case
    by auto
qed

fun zero-list:: nat ⇒ ('a::zero) list
  where zero-list 0 = []
  | zero-list (Suc n) = (0::'a)#(zero-list n)

lemma zero-list-len:
  shows length (zero-list n) = n
proof (induct n)
  case 0
  then show ?case by auto
next
  case (Suc n)
  then show ?case by auto
qed

lemma zero-list-member:
```

```
  shows m < n ⟹ (zero-list n) ! m = 0
proof -
  assume m < n
  then show (zero-list n) ! m = 0
    proof (induct n arbitrary: m)
      case 0
      then show ?case by auto
    next
      case (Suc n)
      then show ?case
        using less-Suc-eq-0-disj by force
    qed
qed

lemma eval-mpoly-zero-is-zero:
  assumes all-same: ⋀ L. eval-mpoly L p = 0
  shows p = 0
  using assms
proof (induct List.length (sorted-list-of-set (vars p)) arbitrary: p rule: less-induct)
  case less
  {assume *: vars p = {}
    then obtain k where k-prop: p = Const k
      using vars-empty-iff
      by blast
    then have k ≠ 0 ⟹ False using all-same
    proof -
      assume *:k≠ 0
      have eval-mpoly [1] p = k
        using k-prop unfolding eval-mpoly-def
        by auto
      then show False
        using less.prems * by auto
    qed
    then have  p = 0
      using k-prop
      using mpoly-Const-0
      by blast
  }
  moreover {assume *: vars p ≠ {}
    then obtain k where k ∈ vars p
      by blast
    then have MPoly-Type.degree p k > 0
      using degree-pos-iff by blast
    let ?uni-conn = mpoly-to-mpoly-poly-alt k p
    have Polynomial.degree ?uni-conn > 0
      by (simp add: ‹0 < MPoly-Type.degree p k› multivar-as-univar)
    then have ?uni-conn ≠ 0
      by auto
    then have finset: finite {x. Polynomial.poly ?uni-conn x = 0}
```

using *poly-roots-finite*[*of ?uni-conn*]
by *blast*
then have ∃ (*w*::*real*). *Polynomial.poly ?uni-conn* (*Const w*) ≠ *0*
proof −
have (∀ (*w*::*real*). *Const w* ∈ {*x*. *Polynomial.poly ?uni-conn x* = *0*}) ⟹ *False*
proof −
assume ∀ *w*. *Const w* ∈ {*x*. *Polynomial.poly ?uni-conn x* = *0*}
then have *subset*: {*x*. ∃ (*w*::*real*). *Const w* = *x*} ⊆{*x*. *Polynomial.poly ?uni-conn x* = *0*}
by *auto*
have *bij-betw* (λ*x*::*real*. *Const x*) {*x*. ∃ (*w*::*real*). *w* = *x*} {*x*. ∃ (*w*::*real*). *Const w* = *x*}
proof −
have *h1*: *inj-on Const* {*x*. ∃ *w*. *w* = *x*}
unfolding *inj-on-def* **by** *auto*
have *h2*: *Const* ‘ {*x*. ∃ *w*. *w* = *x*} = {*x*. ∃ *w*. *Const w* = *x*}
unfolding *image-def* **by** *auto*
show *?thesis* **using** *h1 h2* **unfolding** *bij-betw-def*
by *auto*
qed
then have *set-eq*: *finite* {*x*. ∃ (*w*::*real*). *w* = *x*} = *finite* {*x*. ∃ (*w*::*real*). *Const w* = *x*}
using *bij-betw-finite*[*of* - {*x*. ∃ (*w*::*real*). *w* = *x*} {*x*. ∃ *w*. *Const w* = *x*}]
by *auto*
have *h1*: ¬ (*finite* {*x*. ∃ (*w*::*real*). *w* = *x*})
using *infinite-UNIV-char-0*
by *auto*
then have ¬ (*finite* {*x*. ∃ (*w*::*real*). *Const w* = *x*})
using *set-eq h1* **by** *auto*
then have ¬ (*finite* {*x*. *Polynomial.poly ?uni-conn x* = *0*})
using *subset infinite-super* **by** *auto*
then show *False* **using** *finset*
by *auto*
qed
then show *?thesis*
by *auto*
qed
then obtain *w* **where** *w-prop*: ((*Polynomial.poly ?uni-conn* (*Const w*))::*real mpoly*) ≠ *0*
by *auto*
have *vars*: *vars* ((*Polynomial.poly ?uni-conn* (*Const w*))::*real mpoly*) ⊆ *vars p* − {*k*}
proof
fix *x*
assume ∗: *x* ∈ *vars* (*poly* (*mpoly-to-mpoly-poly-alt k p*) (*Const w*))
have *vars* (*Const w*) = {} **by** *auto*
then have *ex-i*: ∃ *i*. *x* ∈ *vars* (*poly.coeff* (*mpoly-to-mpoly-poly k p*) *i*)
using *var-in-some-coeff* ∗
by (*metis empty-iff multivar-as-univar*)

show $x \in$ *vars p* $- \{k\}$
using *multivar-as-univar vars-coeff-mpoly-to-mpoly-poly*
using *ex-i* **by** *blast*
qed
then have *card* (*vars* (*poly* (*mpoly-to-mpoly-poly-alt k p*) (*Const w*))) < *card* (*vars p*)
by (*metis Diff-subset* ‹*k* ∈ *vars p*› *card-Diff1-less order-neq-le-trans psubset-card-mono psubset-subset-trans vars-finite*)
then have *vars-len*: *length* (*sorted-list-of-set* (*vars* (*Polynomial.poly ?uni-conn* (*Const w*))))
< *length* (*sorted-list-of-set* (*vars p*))
by *auto*
then have ∃ *z*. *eval-mpoly z* (*Polynomial.poly ?uni-conn* (*Const w*)) ≠ *0*
proof −
have (⋀*L*. *eval-mpoly L* (*Polynomial.poly ?uni-conn* (*Const w*)) = *0*) ⟹ (*Polynomial.poly ?uni-conn* (*Const w*)) = *0*
using *vars-len less.hyps* **by** *auto*
then show *?thesis* **using** *w-prop* **by** *auto*
qed
then obtain *z* **where** *z-prop*: *eval-mpoly z* (*Polynomial.poly ?uni-conn* (*Const w*)) ≠ *0*
by *auto*
obtain *update-z* **where** *update-z-prop*: *update-z* = (*if length z* > *k then z else append z* (*zero-list* (*k* − (*length z*) + *1*)))
by *simp*
then have *update-z-len*: *length update-z* ≥ *k*
using *zero-list-len*
by (*smt* (*z3*) *add-diff-inverse-nat length-append less-or-eq-imp-le linorder-le-less-linear nat-add-left-cancel-less not-add-less1*)
obtain *ell* **where** *ell-prop*:
ell = *list-update update-z k w*
(∀ *i* ≠ *k*. *ell* ! *i* = *update-z* ! *i*) ∧ *ell* ! *k* = *w*
using *update-z-len*
by (*simp add*: *update-z-prop zero-list-len*)
have *k-notin*: *k* ∉ *vars* (*Polynomial.poly ?uni-conn* (*Const w*))
by (*meson DiffE singletonI subsetD vars*)
then have *eval-mpoly update-z* (*Polynomial.poly ?uni-conn* (*Const w*)) =
eval-mpoly ell (*Polynomial.poly ?uni-conn* (*Const w*))
unfolding *eval-mpoly-def*
by (*metis ell-prop*(*1*) *list-update-id not-contains-insertion*)
have *same-except1*: (⋀*y*. *y* ≠ *k* ⟹ *nth-default 0 update-z y* = *nth-default 0 ell y*)
using *update-z-prop*
by (*metis ell-prop*(*1*) *ell-prop*(*2*) *length-list-update nth-default-eq-dflt-iff nth-default-nth*)

have *same-except2*: (⋀*y*. *y* ≠ *k* ⟹ *nth-default 0 update-z y* = *nth-default 0 z y*)
proof −
fix *y*

assume *y-not*: $y \neq k$
{assume $*$: $y \geq (length\ z)$
then have *h1*: *nth-default 0 z y = 0*
unfolding *nth-default-def*
by *auto*
have *h2*: *nth-default 0 update-z y = 0*
unfolding *nth-default-def* **using** *y-not update-z-prop zero-list-member*
by (*smt* (*z3*) $*$ *add-diff-cancel-left′ diff-less-mono length-append linorder-not-le nth-append zero-list-len*)
then have *nth-default 0 update-z y = nth-default 0 z y*
using *h1 h2*
by *auto*
} moreover
{assume $*$: $y < (length\ z)$
then have *nth-default 0 update-z y = nth-default 0 z y*
using *update-z-prop y-not*
by (*metis nth-default-append nth-default-nth*)
}
ultimately show *nth-default 0 update-z y = nth-default 0 z y* **using** *update-z-prop zero-list-member*
using *linorder-le-less-linear* **by** *blast*
qed
have *h*: ($\bigwedge y.\ y \neq k \Longrightarrow$ *nth-default 0 z y = nth-default 0 ell y*)
using *same-except1 same-except2*
by *auto*
then have *same1*: *poly* (*map-poly* (*insertion* (*nth-default 0 z*)) (*mpoly-to-mpoly-poly k p*)) *w =*
(*insertion* (*nth-default 0 ell*)) *p*
using *insertion-mpoly-to-mpoly-poly*[*of k* (*nth-default 0 z*) (*nth-default 0 ell*) *p*]
ell-prop(*2*)
by (*smt* (*verit*, *del-insts*) *One-nat-def add-Suc-right add-diff-inverse-nat ell-prop*(*1*) *length-append length-list-update lessI nth-default-def semiring-norm*(*51*) *update-z-prop zero-list-len*)
have *same2*: *poly* (*map-poly* (*insertion* (*nth-default 0 z*)) (*mpoly-to-mpoly-poly k p*)) *w*
= eval-mpoly z (*Polynomial.poly ?uni-conn* (*Const w*))
unfolding *eval-mpoly-def poly-def*
by (*smt* (*verit*, *best*) *One-nat-def h* ‹*eval-mpoly z* (*poly* (*mpoly-to-mpoly-poly-alt k p*) (*Const w*)) $\neq$ *0*› *k-notin add-diff-inverse-nat arith-extra-simps*(*6*) *ell-prop*(*1*) *eval-mpoly-def eval-mpoly-map-poly-comm-ring-hom.base.poly-map-poly insertion-const insertion-irrelevant-vars insertion-var length-append less.prems lessI multivar-as-univar nat-add-left-cancel-less poly-mpoly-to-mpoly-poly update-z-prop zero-list-len*)
have *same3*: *eval-mpoly ell p = eval-mpoly z* (*Polynomial.poly ?uni-conn* (*Const w*))
using *same1 same2*
using *eval-mpoly-def* **by** *force*
then have *eval-mpoly ell p* $\neq$ *0*
using *z-prop*

by *presburger*
then have *mpoly-to-mpoly-poly-alt k p* $\neq$ *0* $\Longrightarrow$ *False*
using *insertion-mpoly-to-mpoly-poly w-prop*
by (*meson local.less*(*2*))
then have *mpoly-to-mpoly-poly-alt k p* = *0*
by *auto*
then have *p* = *0*
using *multivar-as-univar*
by (*metis mpoly-to-mpoly-poly-0 mpoly-to-mpoly-poly-eq-iff*)
}
ultimately show *?case*
by *auto*
qed

6 Useful properties for decision proofs

lemma *eval-mpoly-same*:
assumes *all-same*: ($\bigwedge$ *L*. *eval-mpoly L p* = *eval-mpoly L q*)
shows *p* = *q*
proof −
have $\bigwedge$ *L*. *eval-mpoly L* (*p* − *q*) = *0*
using *all-same*
using *eval-mpoly-map-poly-comm-ring-hom.base.hom-minus* **by** *fastforce*
then have *p* − *q* = *0*
using *eval-mpoly-zero-is-zero*
by *blast*
then show *?thesis*
by *auto*
qed

lemma *univariate-in-eval*:
fixes *qs*:: *real mpoly list*
fixes *x y*:: *real*
shows (*map* (λ*p*. (*Polynomial.poly p x*)) (*map* (λ*q*. *eval-mpoly-poly* (*y*#*xs*) *q*) (*univariate-in qs 0*))
= *map* (*eval-mpoly* (*x*#*xs*)) *qs*)
proof −
have $\bigwedge$*xa*. *xa* ∈ *set qs* $\Longrightarrow$
poly (*eval-mpoly-poly* (*y* # *xs*) (*mpoly-to-mpoly-poly-alt 0 xa*)) *x* =
eval-mpoly (*x* # *xs*) *xa*
proof −
fix *xa*
assume *xa* ∈ *set qs*
have ($\bigwedge$*ya*. *ya* $\neq$ *0* $\Longrightarrow$ *nth-default 0* (*y* # *xs*) *ya* = *nth-default 0* (*x* # *xs*) *ya*)
by (*metis nat.exhaust nth-default-Cons-Suc*)
then show *poly* (*eval-mpoly-poly* (*y* # *xs*) (*mpoly-to-mpoly-poly-alt 0 xa*)) *x* =
eval-mpoly (*x* # *xs*) *xa*
unfolding *eval-mpoly-poly-def eval-mpoly-def*

using *insertion-mpoly-to-mpoly-poly*[*of 0* (*nth-default 0* (*y* # *xs*))
(*nth-default 0* (*x* # *xs*)) *xa*]
multivar-as-univar[*of 0 xa*]
by *auto*
qed
then show *?thesis* **unfolding** *univariate-in-def*
by *auto*
qed

lemma *lowering-poly-eval-var*:
fixes *q*:: *real mpoly Polynomial.poly*
assumes *not-in-vars*: ∀ *c* ∈ *set* (*Polynomial.coeffs q*). *0* ∉ *vars c*
assumes *nonz*: *q* ≠ *0*
fixes *x y*:: *real*
shows *eval-mpoly-poly xs* (*map-poly* (*lowerPoly 0 1*) *q*)
= *eval-mpoly-poly* (*y*#*xs*) *q*
proof −
let *?map-lp-q* = *map-poly* (*lowerPoly 0 1*) *q*
have *zero-prop*: ⋀*p*. *p* ∈ *set* (*Polynomial.coeffs q*) ⟹ (*lowerPoly 0 1 p* = *0* ⟷
p = *0*)
proof −
fix *p*
assume ∗: *p* ∈ *set* (*Polynomial.coeffs q*)
let *?lp* = *lowerPoly 0 1 p*
have ∀ *r*::*real mpoly*. *r* = *0* ⟷ (∀ *val*. *eval-mpoly val r* = *0*)
using *eval-mpoly-zero-is-zero* **by** *auto*
have *0* ∉ *vars p*
using ∗ *not-in-vars* **by** *auto*
then have *match*: ∀ *y*. *eval-mpoly* (*y* # *xs*) *p* = *eval-mpoly xs ?lp*
by (*simp add*: *eval-mpoly-def insertion-lowerPoly01*)
then have *d1*: *lowerPoly 0 1 p* = *0* ⟹ *p* = *0*
proof −
assume *is-zero*: *lowerPoly 0 1 p* = *0*
then have ⋀ *L*. *eval-mpoly L ?lp* = *0*
using *eval-mpoly-zero-is-zero* **by** *auto*
then have ⋀ *L*. *eval-mpoly L p* = *0*
proof −
fix *L*:: *real list*
{ **assume** ∗: *L* = []
then have *same-eval*: *eval-mpoly* (*0* # *L*) *p* = *eval-mpoly L p*
unfolding *eval-mpoly-def*
by (*smt* (*verit*, *ccfv-threshold*) *add-0 insertion-irrelevant-vars less-Suc0*
list.size(*3*) *list.size*(*4*) *nth-Cons-0 nth-default-Nil nth-default-def*)
then have *eval-mpoly* (*0* # *L*) *p* = *eval-mpoly L ?lp*
using *match*
by (*simp add*: ‹*0* ∉ *vars p*› *eval-mpoly-def insertion-lowerPoly01*)
then have *eval-mpoly L p* = *0*

```
          using is-zero same-eval by auto
        } moreover { assume ∗: length L > 0
          then obtain h T where L = h # T
            by (metis length-greater-0-conv neq-Nil-conv)
          then have same-eval: eval-mpoly L p = eval-mpoly T ?lp
            using match
            by (simp add: ‹0 ∉ vars p› eval-mpoly-def insertion-lowerPoly01)
          then have eval-mpoly L p = 0
            using is-zero same-eval by auto
        }
        ultimately show eval-mpoly L p = 0
          by blast
      qed
      then show p = 0
        using eval-mpoly-zero-is-zero by auto
    qed
    have d2: p = 0 =⇒ lowerPoly 0 1 p = 0
      using match eval-mpoly-zero-is-zero
      using lower0 by blast
    then show (lowerPoly 0 1 p = 0 ←→ p = 0)
      using d1 d2 by auto
  qed
  then have map-nonz: ?map-lp-q ≠ 0
    using nonz
    by (simp add: lower0 map-poly-eq-0-iff)
  have Polynomial.degree q = length (Polynomial.coeffs q) − 1
    using nonz degree-eq-length-coeffs by auto
  then have deg1: Polynomial.degree ?map-lp-q ≤ Polynomial.degree q
    unfolding map-poly-def
    by (metis degree-map-poly-le map-poly-def)
  have deg2: Polynomial.degree ?map-lp-q ≥ Polynomial.degree q
    using zero-prop unfolding map-poly-def
   by (metis Ring-Hom-Poly.degree-map-poly coeff-in-coeffs le-degree leading-coeff-neq-0
map-poly-def nonz)
  then have same-deg: Polynomial.degree q = Polynomial.degree ?map-lp-q
    using deg1 deg2 by auto
  have q = poly-of-list (Polynomial.coeffs q)
   (map-poly (lowerPoly 0 1) q) = poly-of-list (Polynomial.coeffs (map-poly (lowerPoly
0 1) q))
    using Poly-coeffs by auto
 have same-len: length (Polynomial.coeffs q) = length (Polynomial.coeffs (map-poly
(lowerPoly 0 1) q))
    using nonz same-deg degree-eq-length-coeffs map-nonz
    by (simp add: length-coeffs-degree)
 have ⋀i. i < length (Polynomial.coeffs q) =⇒ eval-mpoly (y#xs) ((Polynomial.coeffs
q) ! i) = eval-mpoly xs ((Polynomial.coeffs (map-poly (lowerPoly 0 1) q)) ! i)
  proof − fix i
    assume ∗: i < length (Polynomial.coeffs q)
    then have not-in-vars: 0 ∉ vars ((Polynomial.coeffs q) ! i)
```

```
      using not-in-vars in-set-member
      by auto
   then have same-eval: eval-mpoly (y#xs) ((Polynomial.coeffs q) ! i) = eval-mpoly
xs ((lowerPoly 0 1) ((Polynomial.coeffs q) ! i))
      by (simp add: eval-mpoly-def insertion-lowerPoly01)
    have (Polynomial.coeffs (map-poly (lowerPoly 0 1) q)) ! i =
      (lowerPoly 0 1) ((Polynomial.coeffs q) ! i)
      using coeff-map-poly lower0 same-len *
      by (metis nth-coeffs-coeff)
    then show eval-mpoly (y#xs) ((Polynomial.coeffs q) ! i) = eval-mpoly xs
((Polynomial.coeffs (map-poly (lowerPoly 0 1) q)) ! i)
      using same-eval by auto
  qed
  then show ?thesis unfolding eval-mpoly-poly-def map-poly-def
    using same-len
    by (metis map-poly-def nth-map-conv)
qed


lemma lowering-poly-eval:
  fixes q:: real mpoly Polynomial.poly
  assumes ∀ c ∈ set (Polynomial.coeffs q). 0 ∉ vars c
  fixes x y:: real
  shows eval-mpoly-poly xs (map-poly (lowerPoly 0 1) q)
    = eval-mpoly-poly (y#xs) q
  using lowering-poly-eval-var
  by (metis assms eval-mpoly-poly-comm-ring-hom.hom-zero map-poly-0)

lemma reindexed-univ-qs-eval:
  assumes univ-qs = univariate-in qs 0
  assumes reindexed-univ-qs = map (map-poly (lowerPoly 0 1)) univ-qs
  shows map (eval-mpoly (x#xs)) qs =
 (map (λp. (Polynomial.poly p x )) (map (λq. eval-mpoly-poly xs q) reindexed-univ-qs))
proof −
  have same: ⋀i. i < length reindexed-univ-qs ⟹ (map (λp. (Polynomial.poly p
x )) (map (λq. eval-mpoly-poly xs q) reindexed-univ-qs)) ! i
  = (Polynomial.poly (eval-mpoly-poly xs (reindexed-univ-qs ! i)) x)
    by simp
  have len1: length univ-qs = length qs
    using assms(1) unfolding univariate-in-def
    by auto
  have len2: length reindexed-univ-qs = length univ-qs
    using assms(2) by auto
  have len: length reindexed-univ-qs = length qs
    using len1 len2 by auto
 have ⋀i. i < length qs ⟹ (Polynomial.poly (eval-mpoly-poly xs (reindexed-univ-qs
! i)) x) =
  (eval-mpoly (x#xs) (qs ! i))
  proof −
```

fix *i*
assume *i* < *length qs*
have *reindexed-univ-qs* ! *i* = (*map-poly* (*lowerPoly 0 1*)) ((*univariate-in qs 0*) ! *i*)
using *assms*
using ‹*i* < *length qs*› *len1* **by** *auto*
let *?q* = (*univariate-in qs 0*) ! *i*
let *?q1* = *mpoly-to-mpoly-poly 0* (*qs* ! *i*)
have ∀ *c*∈*set* (*Polynomial.coeffs ?q1*). *0* ∉ *vars c*
using *vars-coeff-mpoly-to-mpoly-poly*[*of 0 qs* ! *i*] *in-set-member*
unfolding *Polynomial.coeffs-def*
by *auto*
then have ∀ *c*∈*set* (*Polynomial.coeffs ?q*). *0* ∉ *vars c*
using *multivar-as-univar*
by (*metis* ‹*i* < *length qs*› *nth-map univariate-in-def*)

then show (*Polynomial.poly* (*eval-mpoly-poly xs* (*reindexed-univ-qs* ! *i*)) *x*) =
(*eval-mpoly* (*x#xs*) (*qs* ! *i*))
using *univariate-in-eval lowering-poly-eval assms*
by (*smt* (*verit*, *ccfv-SIG*) ‹*i* < *length qs*› *length-map nth-map*)
qed
then show *?thesis*
using *same len*
by (*simp add*: *nth-map-conv*)
qed

value *variables-list* [((*Var 0* +(*Const* (*3*::*real*))∗((*Var 1*)^*2*)):: *real mpoly*)]

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

value (*mpoly-to-mpoly-poly-alt* (*1*::*nat*) ((*Var 0* +(*Const* (*3*::*real*))∗((*Var 1*)^*2*)):: *real mpoly*))::
real mpoly Polynomial.poly

end

theory *Multiv-Consistent-Sign-Assignments*
imports
Multiv-Poly-Props
Datatype-Order-Generator.Order-Generator

begin

derive *linorder rat list*

7 Define satisfies evaluation and proofs

definition *satisfies-evaluation-alternate*:: *real list* ⇒ *real mpoly* ⇒ *rat* ⇒ *bool*
where
  satisfies-evaluation-alternate val f n =
  (*sgn* (*eval-mpoly val f*) = *real-of-rat* (*sgn n*))

definition *satisfies-evaluation*:: *real list* ⇒ *real mpoly* ⇒ *rat* ⇒ *bool* **where**
  satisfies-evaluation val f n =
  ((*Sturm-Tarski.sign* (*eval-mpoly val f*)::*real*) = (*Sturm-Tarski.sign n*::*real*))

lemma *satisfies-evaluation-alternate*:
  shows *satisfies-evaluation-alternate val f n* = *satisfies-evaluation val f n*
proof –
 have *h1*: *sgn* (*eval-mpoly val f*) = *real-of-rat* (*sgn n*) ⟹ *of-int* (*Sturm-Tarski.sign* (*eval-mpoly val f*)) = *of-int* (*Sturm-Tarski.sign n*)
   by (*metis* (*no-types*, *lifting*) *Sturm-Tarski.sign-def of-rat-1 of-rat-eq-0-iff of-rat-eq-iff rel-simps*(*91*) *sgn-eq-0-iff sgn-if sign-simps*(*2*))
 have *h2*: (*of-int*(*Sturm-Tarski.sign* (*eval-mpoly val f*))::*real*) = (*of-int*(*Sturm-Tarski.sign n*)::*real*) ⟹ *sgn* (*eval-mpoly val f*) = *real-of-rat* (*sgn n*)
   by (*smt* (*verit*) *Sturm-Tarski.sign-def of-int-hom.injectivity of-rat-eq-0-iff of-rat-hom.hom-1-iff of-rat-neg-one sgn-if sign-simps*(*2*))
 then show *?thesis* **unfolding** *satisfies-evaluation-alternate-def satisfies-evaluation-def*
using *h1 h2*
   by *blast*
qed

lemma *eval-mpoly-poly-one-less-degree*:
  assumes *satisfies-evaluation val* (*Polynomial.lead-coeff q*) *0*
  shows *eval-mpoly-poly val* (*one-less-degree q*) =
     eval-mpoly-poly val q
  using *assms*
  unfolding *one-less-degree-def satisfies-evaluation-def*
 by (*metis assms diff-zero eval-mpoly-map-poly-comm-ring-hom.base.map-poly-hom-monom eval-mpoly-poly-comm-ring-hom.hom-minus eval-mpoly-poly-def monom-eq-0 of-rat-0 satisfies-evaluation-alternate satisfies-evaluation-alternate-def sgn-0-0*)

lemma *degree-valuation-le*:
  shows *Polynomial.degree* (*eval-mpoly-poly val p*) ≤ *Polynomial.degree p*
  unfolding *eval-mpoly-poly-def*
  by (*simp add*: *degree-map-poly-le*)

lemma *satisfies-evaluation-nonzero*:
  assumes *satisfies-evaluation val p n*
  assumes *n* ≠ *0*
  shows *eval-mpoly val p* ≠ *0*
  using *assms* **unfolding** *satisfies-evaluation-def*
  by (*smt* (*verit*, *ccfv-threshold*) *Sturm-Tarski.sign-def of-int-eq-iff*)

lemma *degree-valuation*:
  assumes *satisfies-evaluation val* (*Polynomial.lead-coeff p*) *n*
  assumes *n* ≠ *0*
  shows *Polynomial.degree p* = *Polynomial.degree* (*eval-mpoly-poly val p*)
  using *satisfies-evaluation-nonzero*[*OF assms*]
  unfolding *eval-mpoly-poly-def satisfies-evaluation-def*
  by (*metis Ring-Hom-Poly.degree-map-poly degree-0 eval-mpoly-map-poly-comm-ring-hom.hom-zero*)

lemma *lead-coeff-valuation*:
  assumes *satisfies-evaluation val* (*Polynomial.lead-coeff p*) *n*
  assumes *n* ≠ *0*
  shows *eval-mpoly val* (*Polynomial.lead-coeff p*) =
      Polynomial.lead-coeff (*eval-mpoly-poly val p*)
  using *satisfies-evaluation-nonzero*[*OF assms*]
  unfolding *eval-mpoly-poly-def satisfies-evaluation-def*
  using *assms*(*1*) *assms*(*2*) *degree-valuation eval-mpoly-poly-def* **by** *force*

lemma *eval-commutes*:
  fixes *p*:: *real mpoly Polynomial.poly*
  assumes *eval-mpoly val* (*Polynomial.lead-coeff p*) ≠ *0*
  shows *eval-mpoly val* (*Polynomial.lead-coeff p*) = *Polynomial.lead-coeff* (*eval-mpoly-poly val p*)
proof −
  have (*Polynomial.degree p*) = (*Polynomial.degree* (*eval-mpoly-poly val p*))
    using *assms*
    using *degree-valuation satisfies-evaluation-def*
    by (*metis degree-valuation-le eval-mpoly-map-poly-comm-ring-hom.base.coeff-map-poly-hom eval-mpoly-poly-def le-antisym le-degree*)
  then have *eval-mpoly val* (*Polynomial.coeff p* (*Polynomial.degree p*)) = *Polynomial.coeff* (*eval-mpoly-poly val p*) (*Polynomial.degree* (*eval-mpoly-poly val p*))
    using *assms lead-coeff-valuation satisfies-evaluation-def*
    using *eval-mpoly-poly-coeff1 le-refl* **by** *presburger*
  then show *?thesis*
    by *auto*
qed

lemma *eval-mpoly-poly-pseudo-divmod*:
  assumes *satisfies-evaluation val* (*Polynomial.lead-coeff p*) *n*
  assumes *n* ≠ *0*
  assumes *satisfies-evaluation val* (*Polynomial.lead-coeff q*) *m*
  assumes *m* ≠ *0*
  shows *pseudo-divmod* (*eval-mpoly-poly val p*) (*eval-mpoly-poly val q*) =
   (*map-prod* (*eval-mpoly-poly val*) (*eval-mpoly-poly val*) (*pseudo-divmod p q*))
proof −
  from *satisfies-evaluation-nonzero*[*OF assms*(*3*−*4*)]
  have *1*: *eval-mpoly-poly val q* ≠ *0* **using** *assms* **unfolding** *satisfies-evaluation-def*
    by (*metis assms*(*3*−*4*) *lead-coeff-valuation leading-coeff-0-iff*)
  then have *2*: *q* ≠ *0*

using *eval-mpoly-poly-def* **by** *fastforce*
show *?thesis* **unfolding** *pseudo-divmod-def*
using *1 2* **apply** *auto*
unfolding *eval-mpoly-poly-def*
by (*smt* (*verit*) *assms degree-valuation eval-mpoly-map-poly-comm-ring-hom.base.pseudo-divmod-main-hom eval-mpoly-poly-def lead-coeff-valuation leading-coeff-0-iff length-coeffs-degree map-poly-0 satisfies-evaluation-nonzero*)
qed

lemma *eval-mpoly-poly-pseudo-mod*:
assumes *satisfies-evaluation val* (*Polynomial.lead-coeff p*) *n*
assumes $n \neq 0$
assumes *satisfies-evaluation val* (*Polynomial.lead-coeff q*) *m*
assumes $m \neq 0$
shows *pseudo-mod* (*eval-mpoly-poly val p*)
(*eval-mpoly-poly val q*) =
eval-mpoly-poly val (*pseudo-mod p q*)
unfolding *pseudo-mod-def*
using *assms eval-mpoly-poly-pseudo-divmod* **by** *auto*

lemma *eval-mpoly-poly-smult*:
shows *Polynomial.smult* (*eval-mpoly val m*) (*eval-mpoly-poly val r*) =
eval-mpoly-poly val (*Polynomial.smult m r*)
apply (*intro poly-eqI*)
apply (*simp-all add*:*eval-mpoly-poly-def coeff-map-poly eval-mpoly-def*)
by (*simp add*: *insertion-mult*)

8 Consistent Sign Assignments for mpoly type

definition *mpoly-sign*:: *real list* ⇒ *real mpoly* ⇒ *rat* **where**
mpoly-sign val f = *of-int* (*Sturm-Tarski.sign* (*eval-mpoly val f*))

lemma *mpoly-sign-lemma-valuation-length*:
$\{x.\ \exists(val::real\ list).\ mpoly\text{-}sign\ val\ q = x\} =$
$\{x.\ \exists(val::real\ list).\ ((\forall v \in vars\ q.\ length\ val > v) \wedge mpoly\text{-}sign\ val\ q = x)\}$
proof −
have *subset1*: $\{x.\ \exists(val::real\ list).\ mpoly\text{-}sign\ val\ q = x\}$
$\subseteq \{x.\ \exists(val::real\ list).\ ((\forall v \in vars\ q.\ length\ val > v) \wedge mpoly\text{-}sign\ val\ q = x)\}$
proof *clarsimp*
fix *val*::*real list*
{
assume ∗: $(\forall v{\in}vars\ q.\ v < length\ val)$
then have $(\forall v{\in}vars\ q.\ v < length\ val) \wedge mpoly\text{-}sign\ val\ q = mpoly\text{-}sign\ val\ q$
by *auto*
then have $\exists vala.\ (\forall v{\in}vars\ q.\ v < length\ vala) \wedge mpoly\text{-}sign\ vala\ q = mpoly\text{-}sign\ val\ q$
by *auto*
}
moreover {

assume ∗: (∃ *v*∈*vars q*. *v* ≥ *length val*)
have *finite* {*vars q*}
by *simp*
then have ∃ *k*. ∀ *v*∈*vars q*. *k* > *v*
by (*meson finite-nat-set-iff-bounded vars-finite*)
then obtain *k* **where** *k-prop*: ∀ *v*∈*vars q*. *k* > *v*
by *auto*
then have *gtz*: *k* − *length val* > *0*
using ∗ **by** *auto*
let *?app-len* = *k* − *length val*
have ∃ (*l*::*real list*). *length l* = *?app-len* ∧ (∀ *i* < *?app-len*. *l* ! *i* = *0*)
using *gtz Polynomial.coeff-monom length-map length-upt nth-map*
proof −
{ fix *nn* :: *real list* ⇒ *nat*
{ assume ∃ *r n na nb*. *nn* (*map* (*poly.coeff* (*Polynomial.monom r n*)) [*na*..<*nb*]) < *nb* − *na* ∧ *nb* − *na* = *k* − *length val* ∧ *r* = *0*
then have ∃ *r n ns*. *nn* (*map* (*poly.coeff* (*Polynomial.monom r n*)) *ns*) < *length ns* ∧ *length ns* = *k* − *length val* ∧ *r* = *0*
by (*metis length-upt*)
then have ∃ *rs*. ¬ *nn rs* < *k* − *length val* ∨ *length rs* = *k* − *length val* ∧ *rs* ! *nn rs* = *0*
by (*metis* (*no-types*) *Polynomial.coeff-monom length-map nth-map*) **}**
then have ∃ *rs*. ¬ *nn rs* < *k* − *length val* ∨ *length rs* = *k* − *length val* ∧ *rs* ! *nn rs* = *0*
by *blast* **}**
then have ∃ *rs*. ∀ *n*. ¬ *n* < *k* − *length val* ∨ *length rs* = *k* − *length val* ∧ *rs* ! *n* = (*0*::*real*)
by *meson*
then show *?thesis*
using *gtz* **by** *blast*
qed
then obtain *l*::*real list* **where** *l-prop*: *length l* = *?app-len* ∧ (∀ *i* < *?app-len*. *l* ! *i* = *0*)
by *auto*
let *?new-list* = *append val l*
have *length ?new-list* = *k* **using** *l-prop*
by (*metis add-diff-cancel-left′ gtz length-append less-imp-add-positive zero-less-diff*)
then have *p1*: ∀ *v*∈*vars q*. *v* < *length ?new-list*
using *k-prop*
by *meson*
have *p2*: *mpoly-sign ?new-list q* = *mpoly-sign val q*
using *l-prop* **unfolding** *mpoly-sign-def eval-mpoly-def*
by (*smt* (*verit*, *best*) *insertion-irrelevant-vars nth-default-append nth-default-def*)

then have ∃ *vala*. (∀ *v*∈*vars q*. *v* < *length vala*) ∧ *mpoly-sign vala q* = *mpoly-sign val q*
using *p1 p2*
by *blast*
}

ultimately have ∃*vala*. (∀ *v*∈*vars q*. *v* < *length vala*) ∧ *mpoly-sign vala q* = *mpoly-sign val q*
using *not-le-imp-less* **by** *blast*
then show ∃*vala*. (∀ *v*∈*vars q*. *v* < *length vala*) ∧ *mpoly-sign vala q* = *mpoly-sign val q*
by *blast*
qed
have *subset2*: {*x*. ∃(*val*::*real list*). ((∀ *v* ∈ *vars q*. *length val* > *v*) ∧ *mpoly-sign val q* = *x*)}
⊆ {*x*. ∃(*val*::*real list*). *mpoly-sign val q* = *x*}
by *blast*
show *?thesis* **using** *subset1 subset2* **by** *auto*
qed

definition *map-mpoly-sign*::*real mpoly list* ⇒ *real list* ⇒ *rat list*
where *map-mpoly-sign qs val* ≡
map ((*rat-of-int* ∘ *Sturm-Tarski.sign*) ∘ (*eval-mpoly val*)) *qs*

definition *all-lists*:: *nat* ⇒ *real list set* **where**
all-lists n = {(*ls*::*real list*). *length ls* = *n*}

definition *mpoly-consistent-sign-vectors*::*real mpoly list* ⇒ *real list set* ⇒ *rat list set*
where *mpoly-consistent-sign-vectors qs S* = (*map-mpoly-sign qs*) ‘ *S*

definition *mpoly-csv*::*real mpoly list* ⇒ *rat list set*
where *mpoly-csv qs* = {*sign-vec*. (∃ *val*. *map-mpoly-sign qs val* = *sign-vec*)}

9 Data structure definitions

definition *mpoly-sign-data*:: *real list* ⇒ *real mpoly* ⇒ (*real mpoly* × *rat*) **where**
mpoly-sign-data val f = (*f*, *mpoly-sign val f*)

definition *map-mpoly-sign-data*:: *real list* ⇒ *real mpoly list* ⇒ (*real mpoly* × *rat*) *list* **where**
map-mpoly-sign-data val qs = *map* (λ*x*. *mpoly-sign-data val x*) *qs*

definition *mpoly-csv-data*::*real mpoly list* ⇒ (*real mpoly* × *rat*) *list set*
where *mpoly-csv-data qs* = {*sign-vec*. (∃ *val*. *map-mpoly-sign-data val qs* = *sign-vec*)}

definition *all-coeffs*:: *real mpoly Polynomial.poly list* ⇒ *real mpoly list*
where *all-coeffs qs* = *concat* (*map Polynomial.coeffs qs*)

primrec *all-coeffs-alt*:: *real mpoly Polynomial.poly list* ⇒ *real mpoly list*
where *all-coeffs-alt* [] = []
| *all-coeffs-alt* (*h*#*T*) = *append* (*Polynomial.coeffs h*) (*all-coeffs T*)

lemma *all-coeffs-alt*: *all-coeffs qs* = *all-coeffs-alt qs*
by (*metis* (*no-types*, *opaque-lifting*) *all-coeffs-alt.simps*(*1*) *all-coeffs-alt.simps*(*2*) *all-coeffs-def concat.simps*(*1*) *concat.simps*(*2*) *list.exhaust list.simps*(*8*) *list.simps*(*9*))

definition *all-coeffs-csv-data*::*real mpoly Polynomial.poly list* ⇒ (*real mpoly* × *rat*) *list set*
where *all-coeffs-csv-data qs* = *mpoly-csv-data* (*all-coeffs qs*)

primrec

lookup-assump-aux:: *'k* ⇒ (*'k* × *'a*) *list* ⇒ *'a option*
where *lookup-assump-aux p* [] = *None*
| *lookup-assump-aux p* (*h* # *T*) =
(*if* (*fst h* = *p*) *then Some* (*snd h*) *else lookup-assump-aux p T*)

fun *lookup-assump*:: *real mpoly* ⇒ (*real mpoly* × *rat*) *list* ⇒ *rat*
where *lookup-assump p q* = (*case* (*lookup-assump-aux p q*) *of*
None ⇒ *1000*
| *Some i* ⇒ *i*)

10 Lemmas about first nonzero coefficient helper

primrec *first-nonzero-coefficient-helper*:: (*real mpoly* × *rat*) *list* ⇒ *real mpoly list* ⇒ *rat*
where *first-nonzero-coefficient-helper assumps* [] = *0*
| *first-nonzero-coefficient-helper assumps* (*h* # *T*) =
(*case lookup-assump-aux h assumps of*
(*Some i*) ⇒ (*if i* ≠ *0 then i else first-nonzero-coefficient-helper assumps T*)
| *None* ⇒ *first-nonzero-coefficient-helper assumps T*)

definition *sign-of-first-nonzero-coefficient*:: (*real mpoly* × *rat*) *list* ⇒ *real mpoly Polynomial.poly* ⇒ *rat*
where *sign-of-first-nonzero-coefficient assumps q* = *first-nonzero-coefficient-helper assumps* (*rev* (*Polynomial.coeffs q*))

definition *sign-of-first-nonzero-coefficient-aux*:: (*real mpoly* × *rat*) *list* ⇒ *real mpoly list* ⇒ *rat*
where *sign-of-first-nonzero-coefficient-aux assumps coeffl* = *first-nonzero-coefficient-helper assumps coeffl*

lemma *sign-of-first-nonzero-coefficient-aux*:*sign-of-first-nonzero-coefficient-aux assumps* (*rev* (*Polynomial.coeffs q*)) = *sign-of-first-nonzero-coefficient assumps q*
by (*simp add*: *sign-of-first-nonzero-coefficient-aux-def sign-of-first-nonzero-coefficient-def*)

definition *sign-of-first-nonzero-coefficient-list*:: *real mpoly Polynomial.poly list* ⇒ (*real mpoly* × *rat*) *list* ⇒ *rat list*

where *sign-of-first-nonzero-coefficient-list qs assumps* = *map* (λ*q*. *sign-of-first-nonzero-coefficient assumps q*) *qs*

lemma *all-coeffs-member*:
fixes *qs*:: *real mpoly Polynomial.poly list*
fixes *q*:: *real mpoly Polynomial.poly*
fixes *coeff* :: *real mpoly*
assumes *q* ∈ *set qs*
assumes *inset*: *coeff* ∈ *set* (*Polynomial.coeffs q*)
shows *coeff* ∈ *set* (*all-coeffs qs*)
proof −
have *coeff* ∈ *set* (*all-coeffs qs*)
using *assms*
proof (*induction qs*)
case *Nil*
then show *?case* **by** *auto*
next
case (*Cons a qs*)
then show *?case*
by (*metis Un-iff all-coeffs-alt all-coeffs-alt.simps*(*2*) *set-ConsD set-append*)
qed
then show *?thesis* **using** *all-coeffs-alt* **by** *auto*
qed

lemma *map-mpoly-sign-data-duplicates*:
fixes *qs*:: *real mpoly list*
fixes *val*:: *real list*
fixes *coeff* :: *real mpoly*
shows ((*coeff*, *i*) ∈ *set* (*map-mpoly-sign-data val qs*) ∧ (*coeff*, *k*) ∈ *set* (*map-mpoly-sign-data val qs*)
⟹ *i* = *k*)
proof *clarsimp*
assume *m1*: (*coeff*, *i*) ∈ *set* (*map-mpoly-sign-data val qs*)
assume *m2*: (*coeff*, *k*) ∈ *set* (*map-mpoly-sign-data val qs*)
show *i* = *k*
using *m1 m2* **unfolding** *map-mpoly-sign-data-def mpoly-sign-data-def*
by (*smt* (*verit*, *del-insts*) *fst-conv imageE list.set-map snd-conv*)
qed

lemma *lookup-assump-aux-property*:
fixes *i*:: *rat*
fixes *l*:: (*real mpoly* × *rat*) *list*
assumes (*c*, *i*) ∈ *set l*
assumes *no-duplicates*: ∀ *j k*.
((*c*, *j*) ∈ *set l* ∧ (*c*, *k*) ∈ *set l* ⟶ *j* = *k*)
shows *lookup-assump-aux c l* = *Some i*
using *assms*
proof (*induct l*)
case *Nil*

then show *?case*
by (*simp add*: *member-rec*(*2*))
next
case (*Cons a l*)
then show *?case*
by *force*
qed

value *Polynomial.coeffs* ([:*1*, *2*, *3*:]::*real Polynomial.poly*)

lemma *lookup-assump-aux-eo*:
shows *lookup-assump-aux p assumps* = *None* ∨ (∃ *k*. *lookup-assump-aux p assumps* = *Some k*)
using *option.exhaust-sel* **by** *blast*

lemma *lookup-assump-means-inset*:
assumes *lookup-assump-aux p assumps* = *Some k*
shows (*p*, *k*) ∈ *set assumps*
using *assms* **proof** (*induct assumps*)
case *Nil*
then show *?case* **by** *auto*
next
case (*Cons a assumps*)
then show *?case*
by (*metis list.set-intros*(*1*) *list.set-intros*(*2*) *lookup-assump-aux.simps*(*2*) *option.inject prod.collapse*)
qed

lemma *inset-means-lookup-assump-some*:
assumes (*p*, *k*) ∈ *set assumps*
shows ∃ *j*. *lookup-assump-aux p assumps* = *Some j*
using *assms*
proof (*induct assumps*)
case *Nil*
then show *?case*
by (*simp add*: *member-rec*(*2*))
next
case (*Cons a assumps*)
then show *?case*
by *force*
qed

value *List.drop 2* [(*0*::*int*), *0*, *3*, *2*, *1*]

lemma *sign-of-first-nonzero-coefficient-drop*:
assumes *list-len* = *length ell*
assumes *k* < *list-len*
assumes ⋀*i*. ((*i* ≥ *k* ∧ *i* < *list-len*) ⟹ (*lookup-assump-aux* (*ell* ! *i*) *assumps* = *None* ∨ *lookup-assump-aux* (*ell* ! *i*) *assumps* = *Some 0*))

shows *first-nonzero-coefficient-helper assumps* (*rev ell*) = *first-nonzero-coefficient-helper assumps* (*drop* (*list-len* − *k*) (*rev ell*))
using *assms*
proof (*induct length ell arbitrary*: *ell list-len k*)
case *0*
then show *?case*
by *auto*
next
case (*Suc x*)
then have ∃ *sub-ell*. *ell* = *sub-ell* @[*nth ell* (*length ell* − *1*)]
by (*metis cancel-comm-monoid-add-class.diff-cancel diff-Suc-1 diff-is-0-eq lessI take-Suc-conv-app-nth take-all*)
then obtain *sub-ell* **where** *sub-ell*: *ell* = *sub-ell* @ [*nth ell* (*length ell* − *1*)] **by** *auto*
then have *len-sub-ell*: *length sub-ell* = *x*
by (*metis Suc.hyps*(*2*) *diff-Suc-1 length-append-singleton*)
have *rev-prop*: *rev ell* = (*nth ell* (*length ell* − *1*))#(*rev sub-ell*)
using *sub-ell*
by *simp*
then have *drop-prop*: (*drop* (*x* − (*k*−*1*)) (*rev sub-ell*)) = (*drop* (*x* − *k* + *2*) (*rev ell*))
by (*smt* (*verit*, *best*) *Suc.hyps*(*2*) *Suc.prems*(*1*) *Suc.prems*(*2*) *Suc-1 Suc-diff-Suc Suc-eq-plus1 add-diff-cancel-right add-diff-inverse-nat diff-Suc-1 diff-diff-left diff-is-0-eq drop-Suc-Cons drop-rev less-imp-le-nat less-one*)
then have *drop-prop2*: (*drop* (*x* − (*k*−*1*)) (*rev sub-ell*)) = (*drop* (*x* − *k* + *1*) (*drop 1* (*rev ell*)))
by *simp*
have ∃ *i*. (*i* ≥ *k* ∧ *i* < *list-len*) ⟶
lookup-assump-aux (*ell* ! *i*) *assumps* = *None* ∨ *lookup-assump-aux* (*ell* ! *i*) *assumps* = *Some 0*
using *Suc.prems*(*3*) **by** *presburger*
let *?i* = (*length ell* − *1*)
have *lookup-assump-aux* (*ell* ! *?i*) *assumps* = *None* ∨ *lookup-assump-aux* (*ell* ! *?i*) *assumps* = *Some 0*
using *assms*(*3*) *assms*(*2*)
by (*metis Suc.hyps*(*2*) *Suc.prems*(*1*) *Suc.prems*(*2*) *Suc.prems*(*3*) *diff-Suc-1 lessI linorder-not-le not-less-eq*)
then have *lookup-assump-aux* ((*rev ell*) ! *0*) *assumps* = *None* ∨ *lookup-assump-aux* ((*rev ell*) ! *0*) *assumps* = *Some 0*
using *rev-prop*
by (*metis nth-Cons-0*)
then have *key-prop*: *first-nonzero-coefficient-helper assumps* (*rev ell*) =
first-nonzero-coefficient-helper assumps (*drop 1* (*rev ell*))
unfolding *first-nonzero-coefficient-helper-def*
by (*smt* (*verit*, *ccfv-SIG*) *One-nat-def drop0 drop-Suc-Cons list.simps*(*7*) *nth-Cons-0 option.simps*(*4*) *option.simps*(*5*) *rev-prop*)
then have *key-prop2*: *first-nonzero-coefficient-helper assumps* (*rev ell*) =
first-nonzero-coefficient-helper assumps (*rev sub-ell*)
using *rev-prop*

by (*metis One-nat-def drop0 drop-Suc-Cons*)
have *eo*: $k = length\ ell - 1 \lor k < length\ sub\text{-}ell$
using *len-sub-ell*
using *Suc.hyps*(*2*) *Suc.prems*(*1*) *Suc.prems*(*2*) **by** *linarith*
moreover {
assume ∗: $k = length\ ell - 1$
then have *len*: $list\text{-}len - k = 1$
using *len-sub-ell Suc.prems*(*1*) *Suc.hyps*(*2*)
by *simp*
then have *first-nonzero-coefficient-helper assumps* (*rev ell*) =
first-nonzero-coefficient-helper assumps (*drop* (*list-len* − *k*) (*rev ell*))
using *key-prop* **by** *auto*
}
moreover {
assume ∗: $k < length\ sub\text{-}ell$
have *impl*: $x = length\ sub\text{-}ell \Longrightarrow$
$k < length\ sub\text{-}ell \Longrightarrow$
($\bigwedge i.\ k \le i \land i < length\ sub\text{-}ell \Longrightarrow$
lookup-assump-aux (*sub-ell* ! *i*) *assumps* = *None* ∨
lookup-assump-aux (*sub-ell* ! *i*) *assumps* = *Some 0*) ⟹
first-nonzero-coefficient-helper assumps (*rev sub-ell*) =
first-nonzero-coefficient-helper assumps (*drop* ((*length sub-ell*) − *k*) (*rev sub-ell*))
using *Suc.prems Suc.hyps sub-ell*
by *blast*
have $\bigwedge i.\ (i \ge k \land i < list\text{-}len) \Longrightarrow$
lookup-assump-aux (*ell* ! *i*) *assumps* = *None* ∨ *lookup-assump-aux* (*ell* ! *i*) *assumps* = *Some 0*
using *Suc.prems*(*3*) **by** *auto*
then have *sub-ell-prop*: ($\bigwedge i.\ (i \ge k \land i < (length\ sub\text{-}ell)) \Longrightarrow$
lookup-assump-aux (*sub-ell* ! *i*) *assumps* = *None* ∨
lookup-assump-aux (*sub-ell* ! *i*) *assumps* = *Some 0*)
using *sub-ell*
by (*metis Suc.hyps*(*2*) *Suc.prems*(*1*) *len-sub-ell less-SucI nth-append*)
then have *first-nonzero-coefficient-helper assumps* (*rev sub-ell*) =
first-nonzero-coefficient-helper assumps (*drop* ((*length sub-ell*) − *k*) (*rev sub-ell*))
using ∗ *len-sub-ell sub-ell-prop impl*
by *blast*
then have *first-nonzero-coefficient-helper assumps* (*rev ell*) =
first-nonzero-coefficient-helper assumps (*drop* (*list-len* − *k*) (*rev ell*))
using *key-prop2 drop-prop2*
by (*metis* (*full-types*) *Suc.hyps*(*2*) *Suc.prems*(*1*) *diff-Suc-1 diff-commute drop-Cons′ len-sub-ell rev-prop*)
}
ultimately have *first-nonzero-coefficient-helper assumps* (*rev ell*) =
first-nonzero-coefficient-helper assumps (*drop* (*list-len* − *k*) (*rev ell*))
using *eo*
by *fastforce*
then show *?case* **by** *auto*
qed

value *Polynomial.coeffs* ([:*0*, *1*, *2*, *3*:]::*real Polynomial.poly*)
value *Polynomial.degree* ([:*0*, *1*, *2*, *3*:]::*real Polynomial.poly*)

lemma *helper-two*:
  assumes *deg-gt*: *Polynomial.degree q* > *0*
  assumes *sat-eval*: $\bigwedge$*p n*. (*p*,*n*) ∈ *set assumps* ⟹ *satisfies-evaluation val p n*
  assumes *lc-zero*: *lookup-assump-aux* (*Polynomial.lead-coeff q*) *assumps* = *Some 0*
  shows *sign-of-first-nonzero-coefficient assumps q* = *sign-of-first-nonzero-coefficient assumps* (*one-less-degree q*)
proof −
  let *?coeffs-q* = *Polynomial.coeffs q*
  let *?coeffs-less* = *Polynomial.coeffs* (*one-less-degree q*)
  obtain *r* **where** *r*:*Polynomial.degree q* = *Suc r*
    using *deg-gt not0-implies-Suc*
    by *blast*
  from *poly-as-sum-of-monoms*[*of q*]
  have *lc-is*: (*Polynomial.lead-coeff q*) = (*Polynomial.coeffs q*) ! (*r* + *1*)
    using *r*
    by (*metis Suc-eq-plus1 coeffs-nth deg-gt degree-0 le-refl less-numeral-extra*(*3*))
  have *qis*: *q* = ($\sum$ *i*≤*r*. *Polynomial.monom* (*poly.coeff q i*) *i*) + *Polynomial.monom* (*Polynomial.lead-coeff q*) (*Polynomial.degree q*)
    using ‹($\sum$ *i*≤*Polynomial.degree q*. *Polynomial.monom* (*poly.coeff q i*) *i*) = *q*› *r* **by** *auto*
  then have *oldis*: *q* − *Polynomial.monom* (*Polynomial.lead-coeff q*) (*Polynomial.degree q*) = ($\sum$ *i*≤*r*. *Polynomial.monom* (*poly.coeff q i*) *i*)
    using *diff-eq-eq* **by** *blast*
  have *same-coeffs*: ∀ *i* ≤ *r*. *Polynomial.coeff q i* = *Polynomial.coeff* (*one-less-degree q*) *i*
    using *qis oldis*
    by (*metis* (*no-types*, *lifting*) *Multiv-Poly-Props.one-less-degree-def Orderings.order-eq-iff Polynomial.coeff-diff Polynomial.coeff-monom diff-zero not-less-eq-eq r*)
  then have *coeff-zer*: ∀ *i* ≤ *r* . (*i* > (*length* (*Polynomial.coeffs* (*one-less-degree q*)) −*1*) ⟶ (*Polynomial.coeffs q*) ! *i* = *0*)
    by (*metis coeffs-between-one-less-degree coeffs-nth deg-gt degree-0 degree-eq-length-coeffs le-imp-less-Suc nat-less-le r*)
  then have $\bigwedge$*i*. (*i* < *r* + *1* ∧ *i* ≥ (*length* (*Polynomial.coeffs* (*one-less-degree q*)))) ⟹ (*lookup-assump-aux* ((*Polynomial.coeffs q*) ! *i*) *assumps* = *None* ∨ *lookup-assump-aux* ((*Polynomial.coeffs q*) ! *i*) *assumps* = *Some 0*)
  proof −
    fix *i*
    let *?coeff* = (*Polynomial.coeffs q*) ! *i*
    assume *i* < *r* + *1* ∧ *i* ≥ (*length* (*Polynomial.coeffs* (*one-less-degree q*)))
    then have (*Polynomial.coeffs q*) ! *i* = *0* **using** *coeff-zer*
      by (*metis* (*no-types*, *lifting*) *Multiv-Poly-Props.one-less-degree-def Polynomial.coeff-monom Suc-eq-plus1 add-diff-cancel-left′ coeffs-eq-Nil coeffs-nth degree-0 diff-less diff-zero length-0-conv length-greater-0-conv less-Suc-eq-le less-imp-diff-less oldis order-le-less qis r*)

then have $\bigwedge k$. *Polynomial.coeffs q* ! *i* = *0* ⟹ *k* ≠ *0* ⟹ (*0*, *k*) ∈ *set assumps* ⟹ *False*
using *sat-eval* **unfolding** *satisfies-evaluation-def*
using *eval-mpoly-map-poly-comm-ring-hom.base.hom-zero sat-eval satisfies-evaluation-nonzero* **by** *blast*
then have ¬(∃ *k*. *k* ≠ *0* ∧ (*?coeff*, *k*) ∈ *set assumps*)
by (*metis* ‹*Polynomial.coeffs q* ! *i* = *0*›)
then have ¬(∃ *k*. *k* ≠ *0* ∧ *lookup-assump-aux* ((*Polynomial.coeffs q*) ! *i*) *assumps* = *Some k*)
using *lookup-assump-means-inset*
by *metis*
then show (*lookup-assump-aux* ((*Polynomial.coeffs q*) ! *i*) *assumps* = *None* ∨ *lookup-assump-aux* ((*Polynomial.coeffs q*) ! *i*) *assumps* = *Some 0*)
using *lookup-assump-aux-eo*
by *metis*
qed
then have *lookup-inbtw*: $\bigwedge i$. (*i* < *r* + *2* ∧ *i* ≥ (*length* (*Polynomial.coeffs* (*one-less-degree q*)))) ⟹ (*lookup-assump-aux* ((*Polynomial.coeffs q*) ! *i*) *assumps* = *None* ∨ *lookup-assump-aux* ((*Polynomial.coeffs q*) ! *i*) *assumps* = *Some 0*)
using *lc-is lc-zero nat-less-le*
by (*metis Suc-1 add-Suc-right less-antisym*)
let *?ell* = (*Polynomial.coeffs q*)
let *?list-len* = *length ?ell*
let *?k* = (*length* (*Polynomial.coeffs* (*one-less-degree q*)))
have *sublist*: *Sublist.strict-prefix* (*Polynomial.coeffs* (*Multiv-Poly-Props.one-less-degree q*)) (*Polynomial.coeffs q*)
using *deg-gt one-less-degree-is-strict-prefix assms* **by** *auto*
then have *drop-is*: (*drop* (*?list-len* − *?k*) (*rev ?ell*)) = (*rev* (*Polynomial.coeffs* (*Multiv-Poly-Props.one-less-degree q*)))
using *same-coeffs one-less-degree-is-strict-prefix*
by (*metis append-eq-conv-conj deg-gt one-less-degree-is-prefix prefix-def rev-take*)

let *?full-list* = (*rev* (*Polynomial.coeffs q*))
let *?sub-list* = (*rev* (*Polynomial.coeffs* (*Multiv-Poly-Props.one-less-degree q*)))
have *length* (*Polynomial.coeffs q*) = *r* + *2*
using *r* **unfolding** *Polynomial.coeffs-def*
by *auto*
have *fnz*: *first-nonzero-coefficient-helper assumps* (*?full-list*) = *first-nonzero-coefficient-helper assumps* (*drop* (*length ?full-list* − *length ?sub-list*) (*?full-list*))
using *lookup-inbtw sign-of-first-nonzero-coefficient-drop*[*of r* + *2 Polynomial.coeffs q length ?sub-list assumps*]
by (*metis* ‹*length* (*Polynomial.coeffs q*) = *r* + *2*› *diff-is-0-eq drop0 length-rev linorder-not-le*)
have *?full-list* = *append* (*take* (*length ?full-list* − *length ?sub-list*) *?full-list*) *?sub-list*
using *sublist drop-is*
by (*metis append-take-drop-id length-rev*)
then have *first-nonzero-coefficient-helper assumps* (*rev* (*Polynomial.coeffs q*)) =

```
    first-nonzero-coefficient-helper assumps
    (rev (Polynomial.coeffs (Multiv-Poly-Props.one-less-degree q)))
    using fnz
    by (simp add: drop-is)
  then show ?thesis
    unfolding sign-of-first-nonzero-coefficient-def by auto
qed

lemma sign-fnz-aux-helper:
  assumes ∀ elem. elem ∈ set coeffl ⟶ lookup-assump-aux elem ell  = Some 0
  shows sign-of-first-nonzero-coefficient-aux ell coeffl = 0 using assms
proof (induct coeffl)
  case Nil
  then show ?case
    by (simp add: sign-of-first-nonzero-coefficient-aux-def)
next
  case (Cons a x)
  have firsth: lookup-assump-aux a ell = Some 0
    using Cons.prems
    by (simp add: member-rec(1))
  have ∀ elem. elem ∈ set x ⟶ lookup-assump-aux elem ell = Some 0
    using Cons.prems
    by (simp add: member-rec(1))
  then have bigh: sign-of-first-nonzero-coefficient-aux ell x = 0
    using Cons.hyps by auto
  then have first-nonzero-coefficient-helper ell x = 0
    unfolding sign-of-first-nonzero-coefficient-aux-def
    by auto
 then show ?case using firsth unfolding sign-of-first-nonzero-coefficient-aux-def
    by auto
qed

lemma sign-fnz-helper:
  assumes ∀ coeff. coeff ∈ set (Polynomial.coeffs q) ⟶ lookup-assump-aux coeff
(map-mpoly-sign-data val (all-coeffs qs))
    = Some 0
   shows sign-of-first-nonzero-coefficient (map-mpoly-sign-data val (all-coeffs qs))
q = 0 using assms
proof −
 have sign-of-first-nonzero-coefficient-aux (map-mpoly-sign-data val (all-coeffs qs))
(rev (Polynomial.coeffs q)) = 0
     using sign-fnz-aux-helper[of (Polynomial.coeffs q) (map-mpoly-sign-data val
(all-coeffs qs))] assms
    by (metis in-set-member set-rev sign-fnz-aux-helper)
  then show ?thesis using
      sign-of-first-nonzero-coefficient-aux by auto
qed

lemma sign-of-first-nonzero-coefficient-zer:
```

assumes *qin*: *q* ∈ *set qs*
assumes (*eval-mpoly-poly val q*) = *0*
shows *sign-of-first-nonzero-coefficient* (*map-mpoly-sign-data val* (*all-coeffs qs*)) *q* =
Sturm-Tarski.sign (*Polynomial.lead-coeff* (*eval-mpoly-poly val q*))
proof −
have *st-zero*: *Sturm-Tarski.sign* (*Polynomial.lead-coeff* (*eval-mpoly-poly val q*)) = *0*
using *assms*
by *simp*
have *h1*: ⋀*coeff*.
q ∈ *set qs* ⟹
Poly (*map* (*eval-mpoly val*) (*Polynomial.coeffs q*)) = *0* ⟹
coeff ∈ *set* (*Polynomial.coeffs q*) ⟹ *0* < *eval-mpoly val coeff* ⟹ *False*
by (*metis* (*no-types*, *opaque-lifting*) *Poly-eq-0 image-eqI image-set in-set-replicate less-numeral-extra*(*3*))
have *h2*: ⋀*coeff*.
q ∈ *set qs* ⟹
Poly (*map* (*eval-mpoly val*) (*Polynomial.coeffs q*)) = *0* ⟹
coeff ∈ *set* (*Polynomial.coeffs q*) ⟹
¬ *0* < *eval-mpoly val coeff* ⟹ *eval-mpoly val coeff* = *0*
by (*metis* (*no-types*, *opaque-lifting*) *Poly-eq-0 image-eqI image-set in-set-replicate*)
have *coeff-zero*: ∀ *coeff* ∈ *set* (*Polynomial.coeffs q*). *mpoly-sign val coeff* = *0*
using *assms* **unfolding** *eval-mpoly-poly-def map-poly-def mpoly-sign-def*
using *h1 h2* **using** *sign-simps*(*2*) **by** *blast*
have ∀ *coeff* ∈ *set* (*Polynomial.coeffs q*). *lookup-assump-aux coeff* (*map-mpoly-sign-data val* (*all-coeffs qs*))
= *Some 0*
proof *clarsimp*
fix *coeff*
assume *inset*: *coeff* ∈ *set* (*Polynomial.coeffs q*)
then have *h1*: *mpoly-sign val coeff* = *0*
using *coeff-zero* **by** *auto*
have *h2*: *coeff* ∈ *set* (*all-coeffs qs*)
using *qin inset all-coeffs-member all-coeffs-alt*
by *blast*
have (*coeff*, *0*) ∈ *set* (*map-mpoly-sign-data val* (*all-coeffs qs*))
unfolding *map-mpoly-sign-data-def mpoly-sign-data-def*
using *h1 h2*
by (*simp add*: *list.set-map member-def*)
then show *lookup-assump-aux coeff* (*map-mpoly-sign-data val* (*all-coeffs qs*)) = *Some 0*
using *lookup-assump-aux-property map-mpoly-sign-data-duplicates* **by** *presburger*
qed
then have ∀ *coeff*. *coeff* ∈ *set* (*Polynomial.coeffs q*) ⟶ *lookup-assump-aux coeff* (*map-mpoly-sign-data val* (*all-coeffs qs*))
= *Some 0*
by *simp*

then show *?thesis* **using** *st-zero sign-fnz-helper*
by *simp*
qed

lemma *sign-of-first-nonzero-coefficient-nonzer*:
assumes *inset*: $q \in set\ qs$
assumes *nonz*: $(eval\text{-}mpoly\text{-}poly\ val\ q) \neq 0$
assumes *sat-eval*: $\bigwedge p\ n.\ (p,n) \in set\ (map\text{-}mpoly\text{-}sign\text{-}data\ val\ (all\text{-}coeffs\ qs)) \Longrightarrow satisfies\text{-}evaluation\ val\ p\ n$
shows *sign-of-first-nonzero-coefficient* (*map-mpoly-sign-data val* (*all-coeffs qs*)) *q* =
Sturm-Tarski.sign (*Polynomial.lead-coeff* (*eval-mpoly-poly val q*))
proof −
let *?assumps* = (*map-mpoly-sign-data val* (*all-coeffs qs*))
have *qnonz*: $q \neq 0$ **using** *nonz* **by** *auto*
let *?eval-q* = (*eval-mpoly-poly val q*)
have ∀ *x* > (*Polynomial.degree ?eval-q*). *Polynomial.coeff ?eval-q x* = *0*
using *coeff-eq-0* **by** *blast*
have *deg-leq*: (*Polynomial.degree ?eval-q*) ≤ *Polynomial.degree q*
by (*simp add*: *degree-map-poly-le eval-mpoly-poly-def*)
let *?deg-eq* = *Polynomial.degree ?eval-q*
let *?deg-q* = *Polynomial.degree q*
have *st-nonzero*: *Sturm-Tarski.sign* (*Polynomial.lead-coeff ?eval-q*) ≠ *0*
using *nonz*
by (*simp add*: *Sturm-Tarski.sign-def*)
then have *coi-sign*: *mpoly-sign val* (*Polynomial.coeff q ?deg-eq*) = *Sturm-Tarski.sign* (*Polynomial.lead-coeff ?eval-q*)
unfolding *mpoly-sign-def eval-mpoly-poly-def*
by *auto*
let *?coi* = *Polynomial.coeff q ?deg-eq*
have *mem*: ((*mpoly-sign-data val ?coi*)::*real mpoly* × *rat*) ∈ *set ?assumps*
unfolding *map-mpoly-sign-data-def* **using** *all-coeffs-member*[*of q qs ?coi*]
by (*metis* ‹*Polynomial.degree* (*eval-mpoly-poly val q*) ≤ *Polynomial.degree q*› *coeff-in-coeffs eval-mpoly-poly-comm-ring-hom.hom-zero image-eqI image-set inset nonz*)
then obtain *elem1 elem2* **where** *elems-prop*: (*elem1*, *elem2*) = *mpoly-sign-data val ?coi*
using *mpoly-sign-data-def* **by** *force*
then have *elem2-prop*: *elem2* = *Sturm-Tarski.sign* (*Polynomial.lead-coeff ?eval-q*)
using *coi-sign*
by (*simp add*: *mpoly-sign-data-def*)
have *key1*: *lookup-assump-aux ?coi ?assumps* = *Some* (*rat-of-int* (*Sturm-Tarski.sign* (*Polynomial.lead-coeff ?eval-q*)))
using *sat-eval elems-prop mem elem2-prop st-nonzero*
by (*simp add*: *eval-mpoly-poly-coeff1 lookup-assump-aux-property map-mpoly-sign-data-duplicates mpoly-sign-data-def mpoly-sign-def*)
have *len-coeffs-q*: *length* (*Polynomial.coeffs q*) = *?deg-q* + *1*
unfolding *Polynomial.coeffs-def* **using** *qnonz*
by *simp*

have *len-coeffs-eq*: *length* (*Polynomial.coeffs ?eval-q*) = *?deg-eq* + *1*
using *length-coeffs nonz* **by** *blast*
moreover {
assume ∗: (*Polynomial.degree ?eval-q*) = *Polynomial.degree q*
then have *coi-is*: *?coi* = *Polynomial.lead-coeff q*
by *simp*
have ∃ *h T*. (*rev* (*Polynomial.coeffs q*)) = (*h*#*T*)
by (*meson neq-Nil-conv not-0-coeffs-not-Nil qnonz rev-is-Nil-conv*)
then obtain *h T* **where** *ht-prop*: (*rev* (*Polynomial.coeffs q*)) = (*h*#*T*)
by *auto*
have (*rev* (*Polynomial.coeffs q*)) ! *0* = (*Polynomial.coeffs q*) ! (*Polynomial.degree q*)
by (*simp add*: *degree-eq-length-coeffs qnonz rev-nth*)
then have *revis*: (*rev* (*Polynomial.coeffs q*)) ! *0* = *?coi*
using *coi-is*
by (*simp add*: *coeffs-nth qnonz*)
then have *lookup-assump-aux* ((*rev* (*Polynomial.coeffs q*)) ! *0*) *?assumps* = *Some* (*rat-of-int* (*Sturm-Tarski.sign* (*Polynomial.lead-coeff ?eval-q*)))
using *key1 st-nonzero coi-sign revis*
unfolding *sign-of-first-nonzero-coefficient-def first-nonzero-coefficient-helper-def*
by *auto*
then have *?thesis*
using *ht-prop st-nonzero* **unfolding** *sign-of-first-nonzero-coefficient-def*
by *auto*
}
moreover {
assume ∗: (*Polynomial.degree ?eval-q*) < *Polynomial.degree q*
then have *lt*: *Polynomial.degree* (*eval-mpoly-poly val q*) + *1* < *Polynomial.degree q* + *1*
using *len-coeffs-q len-coeffs-eq add-less-cancel-right* **by** *blast*
have *key2*: ⋀*x*. ((*x* ≥ *?deg-eq* + *1* ∧ *x* < *?deg-q* + *1*) ⟹
lookup-assump-aux ((*Polynomial.coeffs q*) ! *x*) *?assumps* = *Some 0*)
proof −
fix *x*
assume *xgeq*: (*x* ≥ *?deg-eq* + *1* ∧ *x* < *?deg-q* + *1*)
let *?coi2* = ((*Polynomial.coeffs q*) ! *x*)
have *mem*: ((*mpoly-sign-data val ?coi2*)::*real mpoly* × *rat*) ∈ *set ?assumps*
unfolding *map-mpoly-sign-data-def* **using** *all-coeffs-member*[*of q qs ?coi2*]
by (*metis image-eqI image-set inset len-coeffs-q nth-mem xgeq*)
then obtain *newelem2* **where** *newelems-prop*: (*?coi2*, *newelem2*) = *mpoly-sign-data val ?coi2*
using *mpoly-sign-data-def* **by** *force*
have *xzer*: *eval-mpoly val ?coi2* = *0*
using *xgeq*
by (*metis Suc-eq-plus1 coeff-eq-0 coeffs-nth eval-mpoly-map-poly-comm-ring-hom.base.coeff-map-poly-hom eval-mpoly-poly-def linorder-not-le not-less-eq-eq qnonz*)
have *elem2-prop*: (*sgn* (*eval-mpoly val ?coi2*) = *real-of-rat* (*sgn newelem2*))
using *sat-eval*[*of ?coi2 newelem2*] *mem newelems-prop*
using *satisfies-evaluation-alternate* **unfolding** *satisfies-evaluation-alternate-def*

satisfies-evaluation-def
by *metis*
then have *key11*: *lookup-assump-aux ?coi2 ?assumps = Some 0*
using *xzer*
by (*metis lookup-assump-aux-property map-mpoly-sign-data-duplicates mem newelems-prop of-rat-0 of-rat-hom.injectivity sgn-0-0*)
then show *lookup-assump-aux* ((*Polynomial.coeffs q*) ! *x*) *?assumps = Some 0* **by** *auto*
qed
then have ($\bigwedge i$. *Polynomial.degree* (*eval-mpoly-poly val q*) + *1* ≤ *i* ∧ *i* < *Polynomial.degree q* + *1* ⟹
lookup-assump-aux (*Polynomial.coeffs q* ! *i*) *?assumps = None* ∨
lookup-assump-aux (*Polynomial.coeffs q* ! *i*) *?assumps = Some 0*)
by *blast*
then have *fnz*: *first-nonzero-coefficient-helper ?assumps* (*rev* (*Polynomial.coeffs q*))
= *first-nonzero-coefficient-helper ?assumps* (*drop* ((*?deg-q* + *1*) − (*?deg-eq* + *1*)) (*rev* (*Polynomial.coeffs q*)))
using *sign-of-first-nonzero-coefficient-drop*[*of ?deg-q* + *1 Polynomial.coeffs q ?deg-eq* + *1 ?assumps*]
using *len-coeffs-q lt* **by** *auto*
have *coeff-deg-eq*: (*Polynomial.coeffs q*) ! *?deg-eq* = *?coi*
unfolding *Polynomial.coeffs-def*
by (*smt* (*verit*, *ccfv-threshold*) *Polynomial.coeffs-def coeffs-nth deg-leq qnonz*)
have (*Polynomial.coeffs q*) ! *?deg-eq* = (*Polynomial.coeff q ?deg-eq*)
unfolding *Polynomial.coeffs-def*
by (*smt* (*verit*, *ccfv-SIG*) *Polynomial.coeffs-def coeff-deg-eq*)
then have *drop-zer-is*: (*drop* (*?deg-q* − *?deg-eq*) (*rev* (*Polynomial.coeffs q*))) ! *0* = (*Polynomial.coeff q ?deg-eq*)
using *len-coeffs-q*
by (*metis* (*no-types*, *lifting*) *add.right-neutral add-Suc-right deg-leq degree-eq-length-coeffs diff-diff-cancel diff-diff-left diff-le-self diff-less-Suc length-rev nth-drop plus-1-eq-Suc rev-nth*)
let *?loi* = (*drop* (*?deg-q* − *?deg-eq*) (*rev* (*Polynomial.coeffs q*)))
have ∃ *h T*. *?loi* = *h*#*T*
by (*metis Suc-eq-plus1 append.right-neutral append-take-drop-id diff-is-0-eq diff-less-Suc diff-less-mono le-refl len-coeffs-q length-rev length-take less-nat-zero-code min.cobounded2 variables.cases*)
then have *first-nonzero-coefficient-helper ?assumps* (*drop* ((*?deg-q* + *1*) − (*?deg-eq* + *1*)) (*rev* (*Polynomial.coeffs q*))) =
(*Sturm-Tarski.sign* (*Polynomial.lead-coeff ?eval-q*))
using *key1 st-nonzero drop-zer-is* **unfolding** *first-nonzero-coefficient-helper-def*

by *auto*
then have *?thesis* **unfolding** *sign-of-first-nonzero-coefficient-def*
using *fnz*
by *auto*
}
ultimately have *?thesis*

using *deg-leq nat-less-le* **by** *blast*
then show *?thesis* **by** *auto*
qed

lemma *sign-of-first-nonzero-coefficient*:
assumes *inset*: $q \in$ *set qs*
assumes *sat-eval*: $\bigwedge p\ n.\ (p,n) \in$ *set* (*map-mpoly-sign-data val* (*all-coeffs qs*)) $\Longrightarrow$ *satisfies-evaluation val p n*
shows *sign-of-first-nonzero-coefficient* (*map-mpoly-sign-data val* (*all-coeffs qs*)) $q =$
Sturm-Tarski.sign (*Polynomial.lead-coeff* (*eval-mpoly-poly val q*))
using *assms sign-of-first-nonzero-coefficient-zer sign-of-first-nonzero-coefficient-nonzer*
by *blast*

11 Relating multiple definitions

lemma *csv-as-expected-left*:
fixes *qs*:: *real mpoly list*
assumes *n-is*: $n =$ *length* (*variables-list qs*)
assumes *biggest-var-is*: *biggest-var* $=$ *nth* (*variables-list qs*) $(n-1) + 1$

assumes *qs-signs*: *qs-signs* $=$ *mpoly-consistent-sign-vectors qs* (*all-lists biggest-var*)
shows (*sign-val* $\in$ *qs-signs*) $\Longrightarrow$ ($\exists$ *val*. (*map* (*rat-of-int* $\circ$ *Sturm-Tarski.sign* $\circ$ (λp. *eval-mpoly val p*)) *qs* $=$ *sign-val*))
proof $-$
assume *inset*: *sign-val* $\in$ *qs-signs*
have $\exists$(*l*::*real list*). (*List.length l* $=$ *biggest-var* $\wedge$ *sign-val* $=$ *map-mpoly-sign qs l*)
using *inset qs-signs* **unfolding** *mpoly-consistent-sign-vectors-def all-lists-def*
by *blast*
then show ($\exists$ *val*. (*map* (*rat-of-int* $\circ$ *Sturm-Tarski.sign* $\circ$ (λp. *eval-mpoly val p*)) *qs* $=$ *sign-val*))
using *map-mpoly-sign-def* **by** *auto*
qed

lemma *in-list-lemma*:
assumes $n =$ *length l*
shows *inlist*: ($v \in$ *set l* $\Longrightarrow$ ($\exists k \leq n-1$. $v =$ *nth l k*))
using *assms*
proof (*induct l arbitrary*: *n v*)
case *Nil*
then show *?case*
by (*simp add*: *member-rec*(*2*))
next
case (*Cons a l*)
then have $n - 1 =$ *length l*
by *simp*
then have *ex-l*: $v \in$ *set l* $\longrightarrow$ ($\exists k \leq n-2$. $v = l\ !\ k$)
using *Cons.hyps*

by (*metis diff-diff-left one-add-one*)
have *eo*: $v \in set\ (a\#l) \Longrightarrow v = a \vee v \in set\ l$
by (*simp add*: *member-rec*(*1*))
show *?case*
proof −
have *h1*: $v = a \Longrightarrow \exists k{\le}n - Suc\ 0.\ v = (a\ \#\ l)\ !\ k$
by *auto*
have *h2*: $v \in set\ l \Longrightarrow \exists k{\le}n - Suc\ 0.\ v = (a\ \#\ l)\ !\ k$
proof −
assume ∗: $v \in set\ l$
then have $(\exists k{\le}n{-}2.\ v = l\ !\ k)$
using *ex-l*
by (*metis nat-1-add-1 plus-1-eq-Suc*)
then obtain *k* **where** $k \le n - 2 \wedge v = l\ !\ k$
by *auto*
then have $l\ !\ k = (a\ \#\ l)\ !\ (k{+}1)$
by *force*
then show $\exists k{\le}n - Suc\ 0.\ v = (a\ \#\ l)\ !\ k$
by (*metis* ∗ *One-nat-def Suc-leI* ‹$n - 1 = length\ l$› *in-set-conv-nth nth-Cons-Suc*)
qed
show *?thesis*
using *h1 h2 Cons*
by (*metis One-nat-def eo*)
qed
qed

lemma *eval-list-longer-than-degree*:
assumes *gt-than*: $\forall i \in vars\ q.\ length\ val > i$
assumes $length\ ell \ge length\ val$
assumes $\forall i < length\ val.\ ell\ !\ i = val\ !\ i$
shows $eval\text{-}mpoly\ ell\ q = eval\text{-}mpoly\ val\ q$
proof −
have $\bigwedge m.\ lookup\ (mapping\text{-}of\ q)\ m \neq 0 \Longrightarrow (\prod v.\ nth\text{-}default\ 0\ ell\ v\ \widehat{}\ lookup\ m\ v) = (\prod v.\ nth\text{-}default\ 0\ val\ v\ \widehat{}\ lookup\ m\ v)$
proof −
fix *m*
assume ∗: $lookup\ (mapping\text{-}of\ q)\ m \neq 0$
then have $\forall v.\ (lookup\ m\ v \neq 0 \longrightarrow v \in vars\ q)$
by (*metis coeff-def coeff-isolate-variable-sparse isovarsparNotIn*)
then have *zer-lookup*: $\forall\ v \ge length\ val.\ lookup\ m\ v = 0$
using ∗ *assms*
using *linorder-not-less* **by** *blast*
have *h1*: $\forall\ v \ge length\ val.\ nth\text{-}default\ 0\ ell\ v\ \widehat{}\ lookup\ m\ v = 1$
using *zer-lookup* **by** *auto*
have *h2*: $\forall\ v \ge length\ val.\ nth\text{-}default\ 0\ val\ v\ \widehat{}\ lookup\ m\ v = 1$
using *zer-lookup* **by** *auto*
have *h3*: $\forall\ v < length\ val.\ nth\text{-}default\ 0\ ell\ v\ \widehat{}\ lookup\ m\ v = nth\text{-}default\ 0\ val\ v\ \widehat{}\ lookup\ m\ v$

using *assms*
by (*metis nth-default-def order-less-le-trans*)
then show ($\prod$ *v. nth-default 0 ell v* ^ *lookup m v*) = ($\prod$ *v. nth-default 0 val v* ^ *lookup m v*)
using *h1 h2 h3*
by (*metis linorder-not-le*)
qed
then show *?thesis*
using *assms* **unfolding** *eval-mpoly-def* **unfolding** *insertion-def insertion-aux-def*
unfolding *insertion-fun-def*
by (*smt* (*verit*, *best*) *Prod-any.cong Sum-any.cong id-apply map-fun-apply mult-cancel-left*)
qed

lemma *same-eval-list-tailing-zeros*:
assumes *length ell* > *length val*
assumes ∀ *i* < *length val. ell* ! *i* = *val* ! *i*
assumes *ell-zeros*: ∀ *i* < *length ell.* (*i* ≥ *length val* ⟶ *ell* ! *i* = *0*)
shows *eval-mpoly ell q* = *eval-mpoly val q*
proof −
have ⋀*m. lookup* (*mapping-of q*) *m* ≠ *0* ⟹ ($\prod$ *v. nth-default 0 ell v* ^ *lookup m v*) = ($\prod$ *v. nth-default 0 val v* ^ *lookup m v*)
proof −
fix *m*
assume *lookup* (*mapping-of q*) *m* ≠ *0*
have *h1-val*: ∀ *v* ≥ *length val. nth-default 0 val v* = *0*
by (*simp add*: *nth-default-beyond*)
have *h1-ell*: ⋀ *v* . *v* ≥ *length val* ⟹ *nth-default 0 ell v* = *0*
by (*simp add*: *ell-zeros nth-default-eq-dflt-iff*)
have *h1*: ∀ *v* ≥ *length val. nth-default 0 ell v* ^ *lookup m v* = *nth-default 0 val v* ^ *lookup m v*
using *h1-val h1-ell*
by *presburger*
have *h2*: ∀ *v* < *length val. nth-default 0 ell v* ^ *lookup m v* = *nth-default 0 val v* ^ *lookup m v*
using *assms*
by (*metis nth-default-nth order-less-trans*)
then show ($\prod$ *v. nth-default 0 ell v* ^ *lookup m v*) = ($\prod$ *v. nth-default 0 val v* ^ *lookup m v*)
using *h1 h2*
by (*metis linorder-not-le*)
qed
then show *?thesis*
using *assms* **unfolding** *eval-mpoly-def* **unfolding** *insertion-def insertion-aux-def*
unfolding *insertion-fun-def*
by (*smt* (*verit*, *best*) *Prod-any.cong Sum-any.cong id-apply map-fun-apply mult-cancel-left*)
qed

```
lemma biggest-variable-in-sorted-list:
  assumes length-nonz: variables-list qs ≠ []
  assumes n-is: n = length (variables-list qs)
  shows (m ∈ set (variables-list qs) ⟹ (nth (variables-list qs) (n−1)) ≥ m)
proof −
  have allk: ∀ k < n−1. (nth (variables-list qs) k) ≤ (nth (variables-list qs) (k+1))
    using n-is
    by (simp add: sorted-wrt-nth-less)
  then have ⋀v. ∀ k<n − Suc 0. sorted-list-of-set (variables qs) ! k ≤ sorted-list-of-set
(variables qs) ! Suc k ⟹
        (⋀n l v. n = length l ⟹ v ∈ set l ⟹ ∃ k≤n − Suc 0. v = l ! k) ⟹
        v ∈ set (sorted-list-of-set (variables qs)) ⟹
        v ≤ sorted-list-of-set (variables qs) ! (n − Suc 0)
   by (metis One-nat-def Suc-less-eq Suc-pred diff-less in-list-lemma length-greater-0-conv
length-nonz less-numeral-extra(1) n-is sorted-iff-nth-Suc sorted-nth-mono variables-list.simps)

  then have allin: ∀ v. (v ∈ set (variables-list qs) ⟶ v ≤ nth (variables-list qs)
(n−1))
    using inlist allk
    by (auto)
  then show ⋀ m. (m ∈ set (variables-list qs) ⟹ (nth (variables-list qs) (n−1))
≥ m)
  proof −
    fix m
    assume mem: m ∈ set (variables-list qs)
    let ?len = length (variables-list qs)
    have ∃ w < ?len. m = (variables-list qs) ! w
      using mem
      by (metis in-set-conv-nth)
    then obtain w where w-prop: w < ?len ∧m = (variables-list qs) ! w
      by auto
    then have leq: w ≤ n−1 using n-is
      by auto
    have sorted (variables-list qs)
      using sorted-sorted-list-of-set
      by simp
    then show nth (variables-list qs) (n−1) ≥ m
      using leq w-prop allin mem
      by blast
  qed
qed

lemma csv-as-expected-right:
  fixes qs:: real mpoly list
  assumes length-nonz: length (variables-list qs) > 0
  assumes n-is: n = length (variables-list qs)
  assumes biggest-var-is: biggest-var = nth (variables-list qs) (n−1) + 1

 assumes qs-signs: qs-signs = mpoly-consistent-sign-vectors qs (all-lists biggest-var)
```

shows (∃ *val*. (*map* (*rat-of-int* ∘ *Sturm-Tarski.sign* ∘ (λ*p*. *eval-mpoly val p*)) *qs* = *sign-val*)) ⟹ (*sign-val* ∈ *qs-signs*)
proof −
assume (∃ *val*. (*map* (*rat-of-int* ∘ *Sturm-Tarski.sign* ∘ (λ*p*. *eval-mpoly val p*)) *qs* = *sign-val*))
then obtain *val* **where** *val-prop*: (*map* (*rat-of-int* ∘ *Sturm-Tarski.sign* ∘ (λ*p*. *eval-mpoly val p*)) *qs* = *sign-val*)
by *auto*
then have *length sign-val* = *length qs*
using *List.length-map* **by** *auto*
have *inlist*: ∀ *v*. (*v* ∈ *set* (*variables-list qs*) ⟶ (∃ *k*≤*n*−*1*. *v* = *nth* (*variables-list qs*) *k*))
using *in-list-lemma n-is*
by *metis*
have *allin*: ∀ *v*. (*v* ∈ *set* (*variables-list qs*) ⟶ *v* ≤ *nth* (*variables-list qs*) (*n*−*1*))
using *inlist*
by (*metis biggest-variable-in-sorted-list length-nonz less-numeral-extra*(*3*) *list.size*(*3*) *n-is*)
have *sorted-list-of-set* (*variables qs*) = *remdups* (*sorted-list-of-set* (*variables qs*))
by (*metis remdups-id-iff-distinct sorted-list-of-set*(*3*))
then have *remdups*: *variables-list qs* = *remdups* (*variables-list qs*)
by *simp*
have (*nth* (*variables-list qs*) (*n*−*1*)) ∈ *set* (*variables-list qs*)
using *n-is length-nonz*
by (*metis Suc-pred′ in-set-conv-nth lessI*)

then have *biggest*: ⋀ *m*. (*m* ∈ *set* (*variables-list qs*) ⟹ (*nth* (*variables-list qs*) (*n*−*1*)) ≥ *m*)
using *assms biggest-variable-in-sorted-list*
using *allin* **by** *presburger*
have *gtthan-to-zero*: ∀ *m* ≥ *biggest-var*. ∀ (*q*::*real mpoly*) ∈ *set*(*qs*). *MPoly-Type.degree* (*q*::*real mpoly*) *m* = *0*
proof *clarsimp*
fix *m q*
assume *m*: *biggest-var* ≤ *m*
assume *qin*: *q* ∈ *set qs*
have ∀ *v*. (*v* ∈ *set* (*variables-list qs*) ⟶ *m* > *v*)
using *biggest m*
using *biggest-var-is* **by** *fastforce*
then have ∀ *v* ∈ *vars q*. *m* > *v*
using *qin variables-list-prop*
by *blast*
then show *MPoly-Type.degree q m* = *0*
using *biggest-var-is n-is degree-eq-0-iff*
by *blast*
qed

moreover {
assume ∗: *length val* ≥ *biggest-var*

```
    let ?ell = take biggest-var val
    have ell-prop: length ?ell = biggest-var
      by (simp add: *)
    have ⋀(q::real mpoly). q∈ set qs ⟹  eval-mpoly ?ell q = eval-mpoly val q
    proof − fix q
      assume q ∈ set (qs)
      have h1: ∀ i∈vars q. i < length (take biggest-var val)
      by (metis ‹q ∈ set qs› degree-eq-0-iff ell-prop gtthan-to-zero linorder-le-less-linear)
      have h2: length (take biggest-var val) ≤ length val
        using * ell-prop by auto
      have h3: ∀ i<length (take biggest-var val). val ! i = take biggest-var val ! i
        by simp
      show eval-mpoly ?ell q = eval-mpoly val q
        using h1 h2 h3 eval-list-longer-than-degree[of q ?ell val]
        by auto
    qed
     then have map (rat-of-int ∘ Sturm-Tarski.sign ∘ eval-mpoly ?ell) qs = map
(rat-of-int ∘ Sturm-Tarski.sign ∘ eval-mpoly val) qs
      by auto
    then have ∃ ell. length ell = biggest-var ∧ map (rat-of-int ∘ Sturm-Tarski.sign
∘ eval-mpoly ell) qs = sign-val
      using ell-prop val-prop
      by blast
  }
  moreover {
    assume *: length val < biggest-var
    let ?ell = val @ (zero-list (biggest-var − length val))
    have len: length ?ell = biggest-var
      using * zero-list-len
     by (metis add-diff-cancel-left′ eq-diff-iff length-append less-or-eq-imp-le zero-less-diff)

    then have p1: (∀  n < length val. ?ell ! n = val ! n)
      using *
      by (meson nth-append)
    have p2: (∀ n ≥ length val. n < biggest-var ⟶ ?ell ! n = 0)
      using * zero-list-member
      by (metis diff-less-mono leD nth-append)
    have ⋀q. (q::real mpoly) ∈ set(qs) ⟹ eval-mpoly ?ell q = eval-mpoly val q
    proof −
      fix q
      assume q ∈ set qs
      show eval-mpoly ?ell q = eval-mpoly val q
        using p1 p2  same-eval-list-tailing-zeros[of val ?ell q] * len
        by presburger
    qed
    then have ∃ ell. length ell = biggest-var ∧ map (rat-of-int ∘ Sturm-Tarski.sign
∘ eval-mpoly ell) qs = sign-val
      using val-prop len
      by (smt (verit) comp-apply map-eq-conv)
```

```
  }
 ultimately have ∃ ell. length ell = biggest-var ∧ map (rat-of-int ∘ Sturm-Tarski.sign
∘ eval-mpoly ell) qs = sign-val
    by (meson linorder-le-less-linear)
  then show ?thesis using qs-signs unfolding mpoly-consistent-sign-vectors-def
map-mpoly-sign-def
    using all-lists-def by auto
qed

lemma csv-as-expected:
  assumes length-nonz: length (variables-list qs) > 0
  assumes n-is: n = length (variables-list qs)
  assumes biggest-var-is: biggest-var = nth (variables-list qs) (n−1) + 1

 assumes qs-signs: qs-signs = mpoly-consistent-sign-vectors qs (all-lists biggest-var)
  shows (sign-val ∈ qs-signs) ⟷ (∃ val. (map (rat-of-int ∘ Sturm-Tarski.sign ∘
(eval-mpoly val)) qs = sign-val))
 using assms csv-as-expected-left[of n qs biggest-var qs-signs sign-val] csv-as-expected-right
  by blast

definition dim-poly:: real mpoly ⇒ nat
  where dim-poly q = Max (vars q)

definition dim-poly-list:: real mpoly list ⇒ nat
  where dim-poly-list qs = Max (variables qs)

lemma dim-poly-list-prop:
  assumes length-nonz: variables-list qs ≠ []
  assumes n-is: n = length (variables-list qs)
  shows dim-poly-list qs = nth (variables-list qs) (n−1)
proof −
  let ?biggest-var = nth (variables-list qs) (n−1)
  have ?biggest-var ∈ set (variables-list qs)
    using assms
    by (meson diff-less length-greater-0-conv member-def nth-mem zero-less-one)
  then have h1: nth (variables-list qs) (n−1) ∈ variables qs
    using variables-prop assms
    using variables-list-prop by blast
  have h2: ∀ x ∈ variables qs. x ≤ nth (variables-list qs) (n−1)
    using assms biggest-variable-in-sorted-list variables-list-prop variables-prop
    by presburger
  show ?thesis using h1 h2 variables-finite unfolding dim-poly-list-def
    by (meson Max-eqI)
qed

lemma lookup-assump-aux-subset-consistency:
  assumes val: ⋀p n. (p,n) ∈ set branch-assms ⟹ satisfies-evaluation val p n
  assumes subset: set new-assumps ⊆ set branch-assms
  assumes i-assm: (∃ i. lookup-assump-aux (Polynomial.lead-coeff r) new-assumps
```

= *Some i* ∧ *i* ≠ *0*)
  shows (∃ *i*. *lookup-assump-aux* (*Polynomial.lead-coeff r*) *branch-assms* = *Some i* ∧ *i* ≠ *0*)
proof −
 obtain *i* **where** *i-prop*: *lookup-assump-aux* (*Polynomial.lead-coeff r*) *new-assumps* = *Some i*
    i ≠ *0*
    using *i-assm*
    by *auto*
  then have (*Polynomial.lead-coeff r*, *i*) ∈ *set new-assumps*
    by (*meson lookup-assump-means-inset*)
  then have *in-set*: (*Polynomial.lead-coeff r*, *i*) ∈ *set branch-assms*
    using *subset* **by** *auto*
  then have *satisfies-evaluation val* (*Polynomial.lead-coeff r*) *i*
    using *val* **by** *auto*
  then have ¬ (*satisfies-evaluation val* (*Polynomial.lead-coeff r*) *0*)
    using *i-prop*(*2*) **unfolding** *satisfies-evaluation-def*
   by (*metis linorder-neqE-linordered-idom of-int-hom.hom-0-iff one-neq-zero sign-simps*(*1*) *sign-simps*(*2*) *sign-simps*(*3*) *zero-neq-neg-one*)
  then have *not-in-set*: (*Polynomial.lead-coeff r*, *0*) ∉ *set branch-assms*
    using *val*
    by *blast*
  show *?thesis* **using** *in-set not-in-set i-prop*(*2*)
    by (*metis inset-means-lookup-assump-some lookup-assump-means-inset*)
qed

lemma *lookup-assump-aux-subset-consistent-sign*:
  assumes *val*: ⋀*p n*. (*p*,*n*) ∈ *set branch-assms* ⟹ *satisfies-evaluation val p n*
  assumes *subset*: *set new-assumps* ⊆ *set branch-assms*
  assumes *i1*: *lookup-assump-aux* (*Polynomial.lead-coeff r*) *new-assumps* = *Some i1*
  assumes *i2*: *lookup-assump-aux* (*Polynomial.lead-coeff r*) *branch-assms* = *Some i2*
  shows *Sturm-Tarski.sign i1* = *Sturm-Tarski.sign i2*
proof −
  have (*Polynomial.lead-coeff r*, *i1*) ∈ *set new-assumps*
    using *i1*
    by (*simp add*: *lookup-assump-means-inset*)
  then have *in-set*: (*Polynomial.lead-coeff r*, *i1*) ∈ *set branch-assms*
    using *subset* **by** *auto*
  then have *sat-eval*: *satisfies-evaluation val* (*Polynomial.lead-coeff r*) *i1*
    using *val* **by** *auto*
  have *Sturm-Tarski.sign i1* ≠ *Sturm-Tarski.sign i2* ⟹ *False*
  proof −
    assume *Sturm-Tarski.sign i1* ≠ *Sturm-Tarski.sign i2*
    then have ¬ (*satisfies-evaluation val* (*Polynomial.lead-coeff r*) *i2*)
      using *sat-eval* **unfolding** *satisfies-evaluation-def*
      by *presburger*
    then have *not-in-set*: (*Polynomial.lead-coeff r*, *i2*) ∉ *set branch-assms*

using *val*
by *blast*
then show *False* **using** *i2*
by (*meson lookup-assump-means-inset member-def*)
qed
then show *?thesis*
by *blast*
qed

lemma *lookup-assump-aux-subset-not-none*:
assumes *val*: $\bigwedge p\ n.\ (p,n) \in$ *set branch-assms* $\Longrightarrow$ *satisfies-evaluation val p n*
assumes *subset*: *set new-assumps* $\subseteq$ *set branch-assms*
assumes *i1*: *lookup-assump-aux* (*Polynomial.lead-coeff r*) *new-assumps* = *Some i1*
shows $\exists$ *i2*. *lookup-assump-aux* (*Polynomial.lead-coeff r*) *branch-assms* = *Some i2*
proof −
have (*Polynomial.lead-coeff r*, *i1*) $\in$ *set new-assumps*
using *i1*
by (*simp add*: *lookup-assump-means-inset*)
then have *in-set*: (*Polynomial.lead-coeff r*, *i1*) $\in$ *set branch-assms*
using *subset* **by** *auto*
then show *?thesis*
by (*simp add*: *inset-means-lookup-assump-some member-def*)
qed

end

theory *Multiv-Pseudo-Remainder-Sequence*
imports
Multiv-Consistent-Sign-Assignments

begin

12 Functions

definition *mul-pseudo-mod*:: *'a*::{*comm-ring-1*,*semiring-1-no-zero-divisors*} *Polynomial.poly* ⇒ *'a Polynomial.poly* ⇒ *'a Polynomial.poly* **where**
mul-pseudo-mod p q = (
let m =
(*if even*(*Polynomial.degree p*+*1*−*Polynomial.degree q*)
then −*1*
else −*Polynomial.lead-coeff q*) *in*
Polynomial.smult m (*pseudo-mod p q*))

```
function spmods-multiv-aux::
  real mpoly Polynomial.poly ⇒
   real mpoly Polynomial.poly ⇒
   (real mpoly × rat) list ⇒
   ((real mpoly × rat) list × real mpoly Polynomial.poly list) list where
  spmods-multiv-aux p q assumps = (
  if q = 0 then [(assumps, [p])] else
    (case (lookup-assump-aux (Polynomial.lead-coeff q) assumps) of
      None ⇒
        let left = spmods-multiv-aux p (one-less-degree q) ((Polynomial.lead-coeff q,
(0::rat)) # assumps) in
      let res-one = spmods-multiv-aux q (mul-pseudo-mod p q) ((Polynomial.lead-coeff
q, (1::rat)) # assumps) in
      let res-minus-one = spmods-multiv-aux q (mul-pseudo-mod p q) ((Polynomial.lead-coeff
q, (−1::rat)) # assumps) in
        let right-one = map (λx. (fst x, Cons p (snd x))) res-one in
        let right-minus-one = map (λx. (fst x, Cons p (snd x))) res-minus-one in
          append left (append right-one right-minus-one)
      | (Some i) ⇒
        (if i = 0 then  spmods-multiv-aux p (one-less-degree q) assumps
        else
          let res = spmods-multiv-aux q (mul-pseudo-mod p q) assumps in
          map (λx. (fst x, Cons p (snd x))) res
        )
  )) using prod-cases3
   apply blast
  by fastforce
termination
  apply (relation measure (λ(p,q,r). if q = [:0:] then 1 else 2 + Polynomial.degree
q))
       apply blast
      apply (auto)
  using one-less-degree-degree
        apply (metis one-less-degree-def cancel-comm-monoid-add-class.diff-cancel
degree-0-id gr0I monom-0)
  unfolding mul-pseudo-mod-def
  using pseudo-mod(2)
     apply auto[1]
    apply (simp add: degree-pseudo-mod-less)
  apply (metis Multiv-Poly-Props.one-less-degree-def cancel-comm-monoid-add-class.diff-cancel
degree-0-id monom-0 not-gr-zero one-less-degree-degree)
  by (metis degree-pseudo-mod-less degree-smult-eq smult-eq-0-iff)


function spmods-multiv::
  real mpoly Polynomial.poly ⇒
   real mpoly Polynomial.poly ⇒
   (real mpoly × rat) list ⇒
   ((real mpoly × rat) list × (real mpoly Polynomial.poly list)) list
```

where *spmods-multiv p q assumps* = (
if p = *0 then* [(*assumps*,[])] *else*
(*case* (*lookup-assump-aux* (*Polynomial.lead-coeff p*) *assumps*) *of*
None ⇒
let left = *spmods-multiv* (*one-less-degree p*) *q* ((*Polynomial.lead-coeff p*, (*0*::*rat*)) # *assumps*) *in*
let right-one = *spmods-multiv-aux p q* ((*Polynomial.lead-coeff p*, (*1*::*rat*)) # *assumps*) *in*
let right-minus-one = *spmods-multiv-aux p q* ((*Polynomial.lead-coeff p*, (−*1*::*rat*)) # *assumps*) *in*
left @ (*right-one* @ *right-minus-one*)
| (*Some i*) ⇒
(*if i* = *0 then spmods-multiv* (*one-less-degree p*) *q assumps*
else
spmods-multiv-aux p q assumps
)
))
using *spmods-multiv-aux.cases* **apply** *blast*
by *force*
termination
apply (*relation measure* (λ(*p*,*q*). *if p* = [:*0*:] *then 1 else 2* + *Polynomial.degree p*))
apply *blast*
apply (*auto*)
using *one-less-degree-degree*
apply (*metis Multiv-Poly-Props.one-less-degree-def cancel-comm-monoid-add-class.diff-cancel degree-0-id monom-0 not-gr-zero*)
by (*metis Multiv-Poly-Props.one-less-degree-def cancel-comm-monoid-add-class.diff-cancel degree-0-id monom-0 not-gr-zero one-less-degree-degree*)

declare *spmods-multiv-aux.simps*[*simp del*]
declare *spmods-multiv.simps*[*simp del*]

13 Proofs

lemma *mul-pseudo-mod-valuation*:
assumes *satisfies-evaluation val* (*Polynomial.lead-coeff p*) *n*
assumes *n* ≠ *0*
assumes *satisfies-evaluation val* (*Polynomial.lead-coeff q*) *m*
assumes *m* ≠ *0*
shows *mul-pseudo-mod* (*eval-mpoly-poly val p*) (*eval-mpoly-poly val q*) =
eval-mpoly-poly val (*mul-pseudo-mod p q*)
proof −
from *degree-valuation*[*OF assms*(*1*−*2*)] **have**
1: *Polynomial.degree p* = *Polynomial.degree* (*eval-mpoly-poly val p*) **.**
from *degree-valuation*[*OF assms*(*3*−*4*)] **have**
2: *Polynomial.degree q* = *Polynomial.degree* (*eval-mpoly-poly val q*) **.**
from *lead-coeff-valuation*[*OF assms*(*1*−*2*)] **have**
3: *eval-mpoly val* (*Polynomial.lead-coeff p*) = *Polynomial.lead-coeff* (*eval-mpoly-poly*

val p) **.**
from *lead-coeff-valuation*[*OF assms*(*3*−*4*)] **have**
4: *eval-mpoly val* (*Polynomial.lead-coeff q*) = *Polynomial.lead-coeff* (*eval-mpoly-poly val q*) **.**
show *?thesis*
using *assms*
by (*smt* (*verit*, *ccfv-SIG*) *1 2 4 eval-mpoly-map-poly-comm-ring-hom.base.hom-one eval-mpoly-map-poly-comm-ring-hom.base.hom-uminus eval-mpoly-poly-pseudo-mod eval-mpoly-poly-smult mul-pseudo-mod-def of-int-hom.hom-one of-int-hom.hom-uminus*)

qed

lemma *spmods-multiv-aux-induct*[*case-names Base Rec Lookup0 LookupN0*]:
fixes *p q* :: *real mpoly Polynomial.poly*
fixes *assumps* ::(*real mpoly* × *rat*) *list*
assumes *base*: ⋀*p q assumps*. *q* = *0* ⟹ *P p q assumps*
and *rec*: ⋀*p q assumps*.
⟦*q* ≠ *0*;
lookup-assump-aux (*Polynomial.lead-coeff q*) *assumps* = *None*;
P p (*one-less-degree q*) ((*Polynomial.lead-coeff q*, *0*) # *assumps*);
P q (*mul-pseudo-mod p q*) ((*Polynomial.lead-coeff q*, *1*) # *assumps*);
P q (*mul-pseudo-mod p q*) ((*Polynomial.lead-coeff q*, −*1*) # *assumps*)⟧ ⟹
P p q assumps
and *lookup0*: ⋀*p q assumps*.
⟦*q* ≠ *0*;
lookup-assump-aux (*Polynomial.lead-coeff q*) *assumps* = *Some 0*;
P p (*one-less-degree q*) *assumps*⟧ ⟹ *P p q assumps*
and *lookupN0*: ⋀*p q assumps r*.
⟦*q* ≠ *0*;
lookup-assump-aux (*Polynomial.lead-coeff q*) *assumps* = *Some r*;
r ≠ *0*;
P q (*mul-pseudo-mod p q*) *assumps*⟧ ⟹ *P p q assumps*
shows *P p q assumps*
apply(*induct p q assumps rule*: *spmods-multiv-aux.induct*)
by (*metis base rec lookup0 lookupN0 not-None-eq*)

lemma *spmods-multiv-induct*[*case-names Base Rec Lookup0 LookupN0*]:
fixes *p q* :: *real mpoly Polynomial.poly*
fixes *assumps* ::(*real mpoly* × *rat*) *list*
assumes *base*: ⋀*p q assumps*. *p* = *0* ⟹ *P p q assumps*
and *rec*: ⋀*p q assumps*.
⟦*p* ≠ *0*;
lookup-assump-aux (*Polynomial.lead-coeff p*) *assumps* = *None*;
P (*one-less-degree p*) *q* ((*Polynomial.lead-coeff p*, *0*) # *assumps*)⟧ ⟹
P p q assumps
and *lookup0*: ⋀*p q assumps*.
⟦*p* ≠ *0*;
lookup-assump-aux (*Polynomial.lead-coeff p*) *assumps* = *Some 0*;

P (*one-less-degree p*) *q assumps*⟧ ⟹ *P p q assumps*
and *lookupN0*: ⋀*p q assumps r*.
⟦*p* ≠ *0*;
lookup-assump-aux (*Polynomial.lead-coeff p*) *assumps* = *Some r*;
r ≠ *0*⟧ ⟹ *P p q assumps*
shows *P p q assumps*
apply(*induct p q assumps rule*: *spmods-multiv.induct*)
by (*metis base rec lookup0 lookupN0 not-None-eq*)

lemma *spmods-multiv-aux-assum-acc*:
assumes (*acc′*,*seq′*) ∈ *set* (*spmods-multiv-aux p q acc*)
shows *set acc* ⊆ *set acc′*
using *assms*
proof (*induct p q acc arbitrary*:*acc′ seq′ rule*: *spmods-multiv-aux-induct*)
case (*Base p q assumps*)
then show *?case* **by** (*auto simp add*: *spmods-multiv-aux.simps*)
next
case (*Rec p q assumps*)
then show *?case* **using** *spmods-multiv-aux.simps*[*of p q assumps*]
by (*smt* (*z3*) *Un-iff imageE insert-subset list.set*(*2*) *list.set-map old.prod.inject option.simps*(*4*) *prod.collapse set-append*)

next
case (*Lookup0 p q assumps*)
then show *?case*
by (*auto simp add*: *spmods-multiv-aux.simps*[*of p q assumps*])
next
case (*LookupN0 p q assumps r*)
then show *?case* **using** *spmods-multiv-aux.simps*[*of p q assumps*]
using *option.simps*(*5*) *prod.collapse* **by** *fastforce*
qed

lemma *spmods-multiv-assum-acc*:
assumes (*acc′*,*seq′*) ∈ *set* (*spmods-multiv p q acc*)
shows *set acc* ⊆ *set acc′*
using *assms*
proof (*induct p q acc arbitrary*:*acc′ seq′ rule*: *spmods-multiv-induct*)
case (*Base p q assumps*)
then show *?case* **by** (*auto simp add*: *spmods-multiv.simps*)
next
case (*Rec p q assumps*)
then show *?case*
using *spmods-multiv-aux-assum-acc spmods-multiv.simps*[*of p q assumps*]
by (*metis Un-iff insert-subset list.set*(*2*) *option.simps*(*4*) *set-append*)
next
case (*Lookup0 p q assumps*)
then show *?case*
by (*auto simp add*: *spmods-multiv.simps*[*of p q assumps*])

next
  case (*LookupN0 p q assumps r*)
  then show *?case*
    using *spmods-multiv-aux-assum-acc*
    by (*auto simp add*: *spmods-multiv.simps*[*of p q assumps*])
qed

lemma *lookup-assum-aux-mem*:
  assumes *lookup-assump-aux x ls* = *Some i*
  shows (*x*,*i*) ∈ *set ls*
  using *assms*
  apply (*induction ls*)
   apply *force*
  by (*metis fst-conv list.set-intros*(*1*) *list.set-intros*(*2*) *lookup-assump-aux.simps*(*2*) *old.prod.exhaust option.inject prod.sel*(*2*))

lemma *matches-ss*:
  assumes (*Polynomial.lead-coeff p*,*m*) ∈ *set assumps m* ≠ *0*
  assumes (*assumps*, *sturm-seq*) ∈ *set* (*spmods-multiv-aux p q acc*)
  assumes ⋀*p n*. (*p*,*n*) ∈ *set assumps* ⟹ *satisfies-evaluation val p n*
  shows *map* (*eval-mpoly-poly val*) *sturm-seq* =
    spmods (*eval-mpoly-poly val p*) (*eval-mpoly-poly val q*)
  using *assms*
proof (*induct p q acc arbitrary*:*assumps sturm-seq m rule*: *spmods-multiv-aux-induct*)
  case (*Base p q assumps*)
  then show *?case*
    using *lead-coeff-valuation satisfies-evaluation-nonzero spmods-multiv-aux.simps*
by *fastforce*
next
  case *ih*: (*Rec p q acc*)
   let *?left* = *spmods-multiv-aux p* (*one-less-degree q*) ((*Polynomial.lead-coeff q*, (*0*::*rat*)) # *acc*)
  let *?res-one* = *spmods-multiv-aux q* (*mul-pseudo-mod p q*) ((*Polynomial.lead-coeff q*, (*1*::*rat*)) # *acc*)
  let *?res-minus-one* = *spmods-multiv-aux q* (*mul-pseudo-mod p q*) ((*Polynomial.lead-coeff q*, (−*1*::*rat*)) # *acc*)
  have *rec*: (*assumps*, *sturm-seq*) ∈ *set ?left* ∨
    (*assumps*, *sturm-seq*) ∈ *set* (*map* (λ*x*. (*fst x*, *Cons p* (*snd x*))) *?res-one*) ∨
    (*assumps*, *sturm-seq*) ∈ *set* (*map* (λ*x*. (*fst x*, *Cons p* (*snd x*))) *?res-minus-one*)
    using *ih* **by** (*auto simp add*: *spmods-multiv-aux.simps*[*of p q acc*])
  moreover {
    assume (*assumps*, *sturm-seq*) ∈ *set ?left*
    then have *map* (*eval-mpoly-poly val*) *sturm-seq* = *spmods* (*eval-mpoly-poly val p*) (*eval-mpoly-poly val q*)
      by (*metis eval-mpoly-poly-one-less-degree ih.hyps*(*3*) *ih.prems*(*1*) *ih.prems*(*2*) *ih.prems*(*4*) *insert-subset list.set*(*2*) *spmods-multiv-aux-assum-acc*)
  }
  moreover {

assume ∗∗:
(assumps, sturm-seq) ∈ *set (map (λx. (fst x, Cons p (snd x))) ?res-one)* ∨
(assumps, sturm-seq) ∈ *set (map (λx. (fst x, Cons p (snd x))) ?res-minus-one)*
then obtain *s ss* **where** *ss:sturm-seq = s#ss*
and *rec:(assumps,ss)* ∈ *set ?res-one* ∨ *(assumps,ss)* ∈ *set ?res-minus-one*
by *auto*
have *lead-coeff-inset: (Polynomial.lead-coeff q,1)* ∈ *set assumps* ∨ *(Polynomial.lead-coeff q,−1)* ∈ *set assumps*
using ∗∗ *spmods-multiv-aux-assum-acc* **by** *fastforce*
then have *A:map (eval-mpoly-poly val) ss =*
spmods (eval-mpoly-poly val q) (eval-mpoly-poly val (mul-pseudo-mod p q))
by *(metis ih.hyps(4) ih.hyps(5) ih.prems(4) local.rec zero-neq-neg-one zero-neq-one)*
have *B:spmods (eval-mpoly-poly val p) (eval-mpoly-poly val q) =*
((eval-mpoly-poly val p) # (spmods (eval-mpoly-poly val q) (mul-pseudo-mod (eval-mpoly-poly val p) (eval-mpoly-poly val q))))
by *(metis ih.prems(1) ih.prems(2) ih.prems(4) lead-coeff-valuation leading-coeff-0-iff mul-pseudo-mod-def satisfies-evaluation-nonzero spmods.simps)*
have *C: mul-pseudo-mod (eval-mpoly-poly val p) (eval-mpoly-poly val q) = eval-mpoly-poly val (mul-pseudo-mod p q)*
by *(metis lead-coeff-inset ih.prems(1) ih.prems(2) ih.prems(4) mul-pseudo-mod-valuation zero-neq-neg-one zero-neq-one)*
have *map (eval-mpoly-poly val) sturm-seq = spmods (eval-mpoly-poly val p) (eval-mpoly-poly val q)*
using *A B C rec ss* ∗∗ **by** *auto*
}
ultimately show *?case*
using *local.rec* **by** *blast*
next
case *(Lookup0 p q acc)*
then have *rec: (assumps, sturm-seq)* ∈ *set (spmods-multiv-aux p (one-less-degree q) acc)*
by *(auto simp add: spmods-multiv-aux.simps[of p q acc])*
have *(Polynomial.lead-coeff q,0)* ∈ *set acc*
by *(simp add: Lookup0.hyps(2) lookup-assum-aux-mem)*
then have *satisfies-evaluation val (Polynomial.lead-coeff q) 0*
using *Lookup0.prems(3) Lookup0.prems(4) spmods-multiv-aux-assum-acc* **by** *blast*
then have *eval-mpoly-poly val (one-less-degree q) = (eval-mpoly-poly val q)*
by *(auto simp add: eval-mpoly-poly-one-less-degree)*
then show *?case*
using *Lookup0.hyps(3) Lookup0.prems(1) Lookup0.prems(2) Lookup0.prems(4) local.rec* **by** *presburger*
next
case *ih:(LookupN0 p q acc r)*
then have *asm:(assumps, sturm-seq)* ∈ *set (*
map (λx. (fst x, Cons p (snd x)))
(spmods-multiv-aux q (mul-pseudo-mod p q) acc))
by *(auto simp add: spmods-multiv-aux.simps[of p q acc])*
then obtain *s ss* **where** *ss:sturm-seq = s#ss*

and *rec*:(*assumps*,*ss*) ∈ *set* (*spmods-multiv-aux q* (*mul-pseudo-mod p q*) *acc*)
by *auto*
have *A*:*map* (*eval-mpoly-poly val*) *ss* = *spmods* (*eval-mpoly-poly val q*) (*eval-mpoly-poly val* (*mul-pseudo-mod p q*))
using *ih*(*4*)[*OF* - - *rec*]
by (*meson ih.hyps*(*2*) *ih.hyps*(*3*) *ih.prems*(*4*) *in-mono local.rec lookup-assump-means-inset spmods-multiv-aux-assum-acc*)
have *B*:*spmods* (*eval-mpoly-poly val p*) (*eval-mpoly-poly val q*) =
((*eval-mpoly-poly val p*) # (*spmods* (*eval-mpoly-poly val q*) (*mul-pseudo-mod* (*eval-mpoly-poly val p*) (*eval-mpoly-poly val q*))))
by (*metis ih*(*5*) *ih*(*6*) *ih.prems*(*4*) *lead-coeff-valuation leading-coeff-0-iff mul-pseudo-mod-def satisfies-evaluation-nonzero spmods.simps*)
have *C*: *mul-pseudo-mod* (*eval-mpoly-poly val p*) (*eval-mpoly-poly val q*) = *eval-mpoly-poly val* (*mul-pseudo-mod p q*)
by (*meson ih.hyps*(*2*) *ih.hyps*(*3*) *ih.prems*(*1*) *ih.prems*(*2*) *ih.prems*(*4*) *local.rec lookup-assum-aux-mem mul-pseudo-mod-valuation spmods-multiv-aux-assum-acc subsetD*)
show *?case*
using *A B C rec ss asm* **by** *force*
qed

lemma *spmods-multiv-aux-sturm-lc*:
assumes (*Polynomial.lead-coeff p*,*m*) ∈ *set acc m* ≠ *0*
assumes (*acc′*,*seq′*) ∈ *set* (*spmods-multiv-aux p q acc*)
assumes *el* ∈ *set seq′*
shows ∃ *r*. (*Polynomial.lead-coeff el*,*r*) ∈ *set acc′* ∧ *r* ≠ *0*
using *assms*
proof (*induct p q acc arbitrary*:*acc′ seq′ el m rule*: *spmods-multiv-aux-induct*)
case (*Base p q acc*)
then show *?case*
using *empty-iff fst-conv list.set*(*1*) *prod.sel*(*2*) *set-ConsD spmods-multiv-aux.simps*
by *auto*
next
case (*Rec p q acc*)
then show *?case*
apply (*auto simp add*: *spmods-multiv-aux.simps*[*of p q acc*])
apply (*meson Rec.prems*(*3*) *spmods-multiv-aux-assum-acc subset-eq*)
apply (*metis zero-neq-one*)
apply (*meson Rec.prems*(*3*) *spmods-multiv-aux-assum-acc subset-iff*)
by (*metis zero-neq-neg-one*)

next
case (*Lookup0 p q acc*)
then show *?case*
by (*auto simp add*: *spmods-multiv-aux.simps*[*of p q acc*])
next
case (*LookupN0 p q acc r*)
then show *?case*
apply (*auto simp add*: *spmods-multiv-aux.simps*[*of p q acc*])

using *spmods-multiv-aux-assum-acc* **apply** *blast*
by (*meson lookup-assum-aux-mem*)
qed

lemma *spmods-multiv-sturm-lc*:
assumes (*acc′*,*seq′*) ∈ *set* (*spmods-multiv p q acc*)
assumes *el* ∈ *set seq′*
shows ∃ *r*. (*Polynomial.lead-coeff el*,*r*) ∈ *set acc′* ∧ *r* ≠ *0*
using *assms*
proof (*induct p q acc arbitrary*:*acc′ seq′ rule*: *spmods-multiv-induct*)
case (*Base p q assumps*)
then show *?case*
using *spmods-multiv.simps*
by *simp*
next
case (*Rec p q assumps*)
then show *?case*
apply (*auto simp add*: *spmods-multiv.simps*[*of p q assumps*])
using *spmods-multiv-aux-sturm-lc*
apply (*metis list.set-intros*(*1*) *zero-neq-one*)
by (*metis list.set-intros*(*1*) *spmods-multiv-aux-sturm-lc zero-neq-neg-one*)
next
case (*Lookup0 p q assumps*)
then show *?case*
by (*auto simp add*: *spmods-multiv.simps*[*of p q assumps*])
next
case (*LookupN0 p q assumps r*)
then show *?case* **using** *spmods-multiv.simps*[*of p q assumps*]
spmods-multiv-aux-sturm-lc lookup-assum-aux-mem
by (*smt* (*verit*, *ccfv-threshold*) *option.simps*(*5*))

qed

lemma *matches-len-complete*:
assumes ⋀*p n*. (*p*,*n*) ∈ *set acc* ⟹ *satisfies-evaluation val p n*
obtains *assumps sturm-seq* **where**
(*assumps*, *sturm-seq*) ∈ *set* (*spmods-multiv-aux p q acc*)
⋀ *p n*. (*p*,*n*) ∈ *set assumps* ⟹ *satisfies-evaluation val p n*
using *assms*
proof (*induct p q acc arbitrary*: *thesis rule*: *spmods-multiv-aux-induct*)
case (*Base p q acc*)
then show *?case*
by (*simp add*: *spmods-multiv-aux.simps*)
next
case *ih*: (*Rec p q acc*)
let *?left* = *spmods-multiv-aux p* (*one-less-degree q*) ((*Polynomial.lead-coeff q*, (*0*::*rat*)) # *acc*)
let *?res-one* = *spmods-multiv-aux q* (*mul-pseudo-mod p q*) ((*Polynomial.lead-coeff*

```
q, (1::rat)) # acc)
 let ?res-minus-one = spmods-multiv-aux q (mul-pseudo-mod p q) ((Polynomial.lead-coeff
q, (−1::rat)) # acc)
  have satisfies-evaluation val (Polynomial.lead-coeff q) 0 ∨
    satisfies-evaluation val (Polynomial.lead-coeff q) 1 ∨
    satisfies-evaluation val (Polynomial.lead-coeff q) (−1)
    unfolding satisfies-evaluation-def
    apply auto
    using Sturm-Tarski.sign-cases
    by (metis of-int-hom.hom-one of-int-minus)
  then have q:
  (∀ p n. (p,n) ∈ set ((Polynomial.lead-coeff q, 0) # acc) ⟶ satisfies-evaluation
val p n) ∨
  (∀ p n. (p,n) ∈ set ((Polynomial.lead-coeff q, 1) # acc) ⟶ satisfies-evaluation
val p n) ∨
  (∀ p n. (p,n) ∈ set ((Polynomial.lead-coeff q, −1) # acc) ⟶ satisfies-evaluation
val p n)
    by (simp add: ih.prems(2))
  moreover {
     assume ∗:(∀ p n. (p,n) ∈ set ((Polynomial.lead-coeff q, 0) # acc) ⟶ satis-
fies-evaluation val p n)
    then have ?case using ∗ ih(3)
        by (metis (no-types, lifting) Un-iff ih.hyps(1) ih.hyps(2) ih.prems(1) op-
tion.case(1) set-append spmods-multiv-aux.simps)
  }
  moreover {
     assume ∗:(∀ p n. (p,n) ∈ set ((Polynomial.lead-coeff q, 1) # acc) ⟶ satis-
fies-evaluation val p n)
    then have ?case using ∗ ih(4)
    by (smt (z3) Un-iff fst-conv ih.hyps(1) ih.hyps(2) ih.prems(1) in-set-conv-decomp
list.simps(9) map-append option.case(1) set-append spmods-multiv-aux.simps)
  }
  moreover {
    assume ∗:(∀ p n. (p,n) ∈ set ((Polynomial.lead-coeff q, −1) # acc) ⟶ satis-
fies-evaluation val p n)
    then have ?case using ∗ ih(5)
    by (smt (z3) Un-iff fst-conv ih.hyps(1) ih.hyps(2) ih.prems(1) in-set-conv-decomp
list.simps(9) map-append option.case(1) set-append spmods-multiv-aux.simps)
  }
  ultimately show ?case
    by fastforce
next
  case (Lookup0 p q acc)
  then show ?case
    by (auto simp add: spmods-multiv-aux.simps[of p q acc])
next
  case ih: (LookupN0 p q assumps r)
  then obtain assumps′ sturm-seq′ where
    (assumps′, sturm-seq′) ∈ set (spmods-multiv-aux q (mul-pseudo-mod p q) as-
```

sumps)
  (⋀*p n*. (*p*, *n*) ∈ *set assumps'* ⟹ *satisfies-evaluation val p n*)
  by *blast*
 then show *?case* **using** *ih spmods-multiv-aux.simps*[*of p q assumps*] *fst-conv image-eqI*
  by (*smt* (*verit*) *list.set-map option.simps*(*5*))

qed

lemma *spmods-multiv-nonz-some*:
 fixes *p*:: *real mpoly Polynomial.poly*
 fixes *q*:: *real mpoly Polynomial.poly*
 assumes *inset*: (*assumps*, *sturm-seq*) ∈ *set* (*spmods-multiv p q acc*)
 shows *p* ≠ *0* ⟹ ∃ *i*. *lookup-assump-aux* (*Polynomial.lead-coeff p*) *assumps* = *Some i*
 using *assms*
proof (*induct p q acc rule*: *spmods-multiv-induct*)
 case (*Base p q acc*)
 then show *?case* **by** *auto*
next
 case (*Rec p q acc*)
 then have (*lookup-assump-aux* (*Polynomial.lead-coeff p*) *acc*) = *None*
  by *meson*
 then have *set* (*spmods-multiv p q acc*) =
   set (*spmods-multiv* (*one-less-degree p*) *q* ((*Polynomial.lead-coeff p*, (*0*::*rat*)) # *acc*))
   ∪ *set* (*spmods-multiv-aux p q* ((*Polynomial.lead-coeff p*, (*1*::*rat*)) # *acc*))
   ∪ *set* (*spmods-multiv-aux p q* ((*Polynomial.lead-coeff p*, (−*1*::*rat*)) # *acc*))
  by (*simp add*: *Rec.prems*(*1*) *spmods-multiv.simps sup-assoc*)
 then have (*assumps*, *sturm-seq*) ∈ *set* (*spmods-multiv* (*one-less-degree p*) *q* ((*Polynomial.lead-coeff p*, (*0*::*rat*)) # *acc*))
   ∨ (*assumps*, *sturm-seq*) ∈ *set* (*spmods-multiv-aux p q* ((*Polynomial.lead-coeff p*, (*1*::*rat*)) # *acc*))
   ∨ (*assumps*, *sturm-seq*) ∈ *set* (*spmods-multiv-aux p q* ((*Polynomial.lead-coeff p*, (−*1*::*rat*)) # *acc*))
  using *Rec.prems*(*2*) **by** *blast*
 then show *?case*
  using *spmods-multiv-assum-acc spmods-multiv-aux-assum-acc*
  by (*metis insert-subset inset-means-lookup-assump-some list.set*(*2*))
next
 case (*Lookup0 p q acc*)
 then show *?case*
  by (*meson inset-means-lookup-assump-some lookup-assum-aux-mem spmods-multiv-assum-acc subset-eq*)
next
 case (*LookupN0 p q acc r*)
 then show *?case*
  by (*meson in-set-member inset-means-lookup-assump-some lookup-assump-means-inset spmods-multiv-assum-acc subsetD*)

qed

lemma *spmods-multiv-aux-nonz-some*:
  fixes *p*:: *real mpoly Polynomial.poly*
  fixes *q*:: *real mpoly Polynomial.poly*
  assumes *inset*: (*assumps*, *sturm-seq*) ∈ *set* (*spmods-multiv-aux p q acc*)
  shows $q \neq 0 \Longrightarrow \exists\, i.$ *lookup-assump-aux* (*Polynomial.lead-coeff q*) *assumps* = *Some i*
  using *assms*
proof (*induct p q acc rule*: *spmods-multiv-aux-induct*)
  case (*Base p q acc*)
  then show *?case* **by** *auto*
next
  case (*Rec p q acc*)
  then have (*lookup-assump-aux* (*Polynomial.lead-coeff q*) *acc*) = *None*
    by *meson*
  then have *set* (*spmods-multiv-aux p q acc*) =
     set (*spmods-multiv-aux p* (*one-less-degree q*) ((*Polynomial.lead-coeff q*, (*0*::*rat*)) # *acc*))
    ∪ *set* (*map* (λ*x*. (*fst x*, *Cons p* (*snd x*))) (*spmods-multiv-aux q* (*mul-pseudo-mod p q*) ((*Polynomial.lead-coeff q*, (*1*::*rat*)) # *acc*)))
    ∪ *set* (*map* (λ*x*. (*fst x*, *Cons p* (*snd x*))) (*spmods-multiv-aux q* (*mul-pseudo-mod p q*) ((*Polynomial.lead-coeff q*, (−*1*::*rat*)) # *acc*)))
    using *Rec.prems*(*1*) *spmods-multiv-aux.simps*
    by (*simp add*: *Rec.prems*(*1*) *spmods-multiv-aux.simps sup-assoc*)
  then have *bigh*: (*assumps*, *sturm-seq*) ∈ *set* (*spmods-multiv-aux p* (*one-less-degree q*) ((*Polynomial.lead-coeff q*, (*0*::*rat*)) # *acc*))
    ∨ (*assumps*, *sturm-seq*) ∈ *set* (*map* (λ*x*. (*fst x*, *Cons p* (*snd x*))) (*spmods-multiv-aux q* (*mul-pseudo-mod p q*) ((*Polynomial.lead-coeff q*, (*1*::*rat*)) # *acc*)))
    ∨ (*assumps*, *sturm-seq*) ∈ *set* (*map* (λ*x*. (*fst x*, *Cons p* (*snd x*))) (*spmods-multiv-aux q* (*mul-pseudo-mod p q*) ((*Polynomial.lead-coeff q*, (−*1*::*rat*)) # *acc*)))
    using *Rec.prems*(*2*)
    by *blast*
  have *h1*: (⋀*acc′ seq′ p q acc*. (*acc′*, *seq′*) ∈ *set* (*spmods-multiv-aux p q acc*) ⟹ *set acc* ⊆ *set acc′*) ⟹
    (*assumps*, *sturm-seq*)
    ∈ *set* (*spmods-multiv-aux p* (*Multiv-Poly-Props.one-less-degree q*)
          ((*Polynomial.lead-coeff q*, *0*) # *acc*)) ⟹
    ∃ *i*. *lookup-assump-aux* (*Polynomial.lead-coeff q*) *assumps* = *Some i*
     by (*meson in-set-member inset-means-lookup-assump-some list.set-intros*(*1*) *subsetD*)
  have *h2*: ⋀*b*. (⋀*acc′ seq′ p q acc*. (*acc′*, *seq′*) ∈ *set* (*spmods-multiv-aux p q acc*) ⟹ *set acc* ⊆ *set acc′*) ⟹
        (*assumps*, *b*)
        ∈ *set* (*spmods-multiv-aux q* (*mul-pseudo-mod p q*) ((*Polynomial.lead-coeff q*, *1*) # *acc*)) ⟹
        sturm-seq = *p* # *b* ⟹ ∃ *i*. *lookup-assump-aux* (*Polynomial.lead-coeff q*) *assumps* = *Some i*
     by (*meson in-set-member inset-means-lookup-assump-some list.set-intros*(*1*)

subsetD)
have *h3*: $\bigwedge$*b*. ($\bigwedge$*acc′ seq′ p q acc*. (*acc′*, *seq′*) ∈ *set* (*spmods-multiv-aux p q acc*)
⟹ *set acc* ⊆ *set acc′*) ⟹
(*assumps*, *b*)
∈ *set* (*spmods-multiv-aux q* (*mul-pseudo-mod p q*) ((*Polynomial.lead-coeff*
q, − *1*) # *acc*)) ⟹
sturm-seq = *p* # *b* ⟹ ∃ *i*. *lookup-assump-aux* (*Polynomial.lead-coeff q*)
assumps = *Some i*
by (*meson in-set-member inset-means-lookup-assump-some list.set-intros*(*1*)
subsetD)
show *?case*
using *bigh spmods-multiv-aux-assum-acc h1 h2 h3*
by *auto*
next
case (*Lookup0 p q acc*)
then show *?case*
by (*meson inset-means-lookup-assump-some lookup-assum-aux-mem spmods-multiv-aux-assum-acc*
subsetD)
next
case (*LookupN0 p q acc r*)
then show *?case*
by (*meson in-set-member inset-means-lookup-assump-some lookup-assump-means-inset*
spmods-multiv-aux-assum-acc subsetD)
qed

lemma *spmods-multiv-sound*:
assumes (*assumps*, *sturm-seq*) ∈ *set* (*spmods-multiv p q acc*)
assumes $\bigwedge$*p n*. (*p*,*n*) ∈ *set assumps* ⟹ *satisfies-evaluation val p n*
shows *map* (*eval-mpoly-poly val*) *sturm-seq* =
spmods (*eval-mpoly-poly val p*) (*eval-mpoly-poly val q*)
using *assms*
proof (*induct p q acc arbitrary*:*assumps sturm-seq rule*: *spmods-multiv-induct*)
case (*Base p q assumps*)
then show *?case*
by (*simp add*: *spmods-multiv.simps*)
next
case (*Rec p q assumps*)
then show *?case*
by (*smt* (*verit*, *best*) *UnE eval-mpoly-poly-one-less-degree list.set-intros*(*1*)
matches-ss option.simps(*4*) *set-append spmods-multiv.simps spmods-multiv-assum-acc*
spmods-multiv-aux-assum-acc subsetD zero-neq-neg-one zero-neq-one)
next
case (*Lookup0 p q assumps*)
define *pval* **where** *pval*:*pval* = *eval-mpoly-poly val p*
define *qval* **where** *qval*:*qval* = *eval-mpoly-poly val q*
have (*Polynomial.lead-coeff p*,*0*) ∈ *set assumps*
using *Lookup0.hyps Lookup0.prems*
by (*meson lookup-assum-aux-mem spmods-multiv-assum-acc subset-code*(*1*))
then have *satisfies-evaluation val* (*Polynomial.lead-coeff p*) *0*

```
    using Lookup0.hyps Lookup0.prems by blast
  from eval-mpoly-poly-one-less-degree[OF this] have
    eval-prop: eval-mpoly-poly val (one-less-degree p) = pval using pval qval
    by auto
  have map (eval-mpoly-poly val) sturm-seq = spmods pval qval
    using Lookup0.hyps Lookup0.prems pval qval
    by (simp add: eval-prop spmods-multiv.simps)
  then show ?case
    using pval qval by blast
next
  case (LookupN0 p q assumps r)
  define pval where pval:pval = eval-mpoly-poly val p
  define qval where qval:qval = eval-mpoly-poly val q
  {
     assume right: ∃ k. lookup-assump-aux (Polynomial.lead-coeff p) assumps =
Some k ∧k ≠ 0
     then obtain k where k-prop: lookup-assump-aux (Polynomial.lead-coeff p)
assumps = Some k ∧k ≠ 0
      by auto
    then have (Polynomial.lead-coeff p,k) ∈ set assumps
      using spmods-multiv-aux-assum-acc
      by (simp add: lookup-assum-aux-mem)
    have k ≠ 0
      using k-prop by auto
    then have
      map (eval-mpoly-poly val) sturm-seq = spmods pval qval
      using matches-ss[of p k assumps sturm-seq q - val] right LookupN0.prems(2)
LookupN0.prems LookupN0.hyps
      using ‹(Polynomial.lead-coeff p, k) ∈ set assumps› pval qval
      by (simp add: spmods-multiv.simps)
  }
  then have map (eval-mpoly-poly val) sturm-seq = spmods pval qval
    using LookupN0.hyps LookupN0.prems
      lookup-assump-aux-subset-consistency option.case(2) spmods-multiv.simps sp-
mods-multiv-aux-assum-acc
   by (smt (verit) Sturm-Tarski.sign-def in-mono lookup-assum-aux-mem of-int-hom.injectivity
satisfies-evaluation-def sign-simps(2) spmods-multiv-nonz-some)
  then show ?case
    using pval qval by blast
qed

end


theory Hybrid-Multiv-Matrix
  imports

    Factor-Algebraic-Polynomial.Poly-Connection
    Multiv-Pseudo-Remainder-Sequence
```

BenOr-Kozen-Reif.More-Matrix
HOL−Library.Mapping
BenOr-Kozen-Reif.Renegar-Algorithm

begin

14 Find CSAS to qs at zeros of p

14.1 Towards Tarski Queries

fun *sminus*:: *nat list* ⇒ *rat list* ⇒ *int* **where**
sminus degree-list sturm-seq = *changes* (*map* (λ*i*. (−*1*)^(*nth degree-list i*)∗(*nth sturm-seq i*)) [*0*..< *length degree-list*])

definition *changes-R-smods-multiv*:: *rat list* ⇒ *nat list* ⇒ *int*
where *changes-R-smods-multiv signs-list degree-list* ≡ (*sminus degree-list signs-list*) − (*changes signs-list*)

definition *changes-R-smods-multiv-val*:: *real mpoly Polynomial.poly list* ⇒ *real list* ⇒ *int* **where**
changes-R-smods-multiv-val sturm-seq val ≡ (*let* (*eval-ss*::*real Polynomial.poly list*) = (*eval-mpoly-poly-list val sturm-seq*) *in* (*changes-poly-neg-inf eval-ss* − *changes-poly-pos-inf eval-ss*))

14.2 Building the Matrix Equation

type-synonym *rmpoly* = *real mpoly Polynomial.poly*
type-synonym *assumps* = (*real mpoly* × *rat*) *list*
type-synonym *matrix-equation* = (*rat mat* × ((*nat list* ∗ *nat list*) *list* × *rat list list*))

definition *base-case-info-M*:: (*assumps* × *matrix-equation*) *list*
where *base-case-info-M* = [([], *base-case-info-R*)]

definition *base-case-info-M-assumps*:: *assumps* ⇒ (*assumps* × *matrix-equation*) *list*
where *base-case-info-M-assumps init-assumps* = [(*init-assumps*, *base-case-info-R*)]

fun *combine-systems-single-M*:: *rmpoly* ⇒ *rmpoly list* ⇒ (*assumps* × *matrix-equation*) ⇒ *rmpoly list* ⇒ (*assumps* × *matrix-equation*) ⇒ (*assumps* × *matrix-equation*)
where *combine-systems-single-M p q1* (*a1*, *m1*) *q2* (*a2*,*m2*) =
(*append a1 a2*, *snd* (*combine-systems-R p* (*q1*, *m1*) (*q2*, *m2*)))

fun *combine-systems-M*:: *rmpoly* ⇒ *rmpoly list* ⇒ (*assumps* × *matrix-equation*) *list* ⇒ *rmpoly list* ⇒
(*assumps* × *matrix-equation*) *list* => *rmpoly list* × ((*assumps* × *matrix-equation*) *list*)
where *combine-systems-M p q1 list1 q2 list2* =
(*append q1 q2*, *concat* (*map* (λ*l1*. (*map* (λ*l2*. *combine-systems-single-M p q1 l1 q2*

l2) *list2*)) *list1*))

definition *construct-NofI-R-spmods*:: *rmpoly* ⇒ *assumps* ⇒ *rmpoly list* ⇒ *rmpoly list* ⇒ (*assumps* × (*rmpoly list*)) *list*
  where *construct-NofI-R-spmods p assumps I1 I2* = (
    let new-p = *sum-list* (*map* (λ*x*. *x*^2) (*p* # *I1*)) *in*
    spmods-multiv new-p ((*pderiv new-p*)∗(*prod-list I2*))) *assumps*

fun *construct-NofI-single-M*:: (*assumps* × (*rmpoly list*)) ⇒ (*assumps* × *rat*)
  where *construct-NofI-single-M* (*input-assumps*, *ss*) =
  (*let lcs* = *lead-coeffs ss*;
    sa-list = *map* (λ*lc*. *lookup-assump lc input-assumps*) *lcs*;
    degrees-list = *degrees ss in*
  (*input-assumps*, *rat-of-int* (*changes-R-smods-multiv sa-list degrees-list*)))

fun *construct-NofI-M*:: *rmpoly* ⇒ *assumps* ⇒ *rmpoly list* ⇒ *rmpoly list* => (*assumps* × *rat*) *list*
  where *construct-NofI-M p assumps I1 I2* =
(*let ss-list* = *construct-NofI-R-spmods p assumps I1 I2 in*
  map construct-NofI-single-M ss-list)

fun *pull-out-pairs*:: *rmpoly list* ⇒ (*nat list* ∗ *nat list*) *list* ⇒ (*rmpoly list* × *rmpoly list*) *list*
  where *pull-out-pairs qs Is* =
  map (λ(*I1*, *I2*). ((*retrieve-polys qs I1*), (*retrieve-polys qs I2*))) *Is*

fun *construct-rhs-vector-rec-M*:: *rmpoly* ⇒ *assumps* ⇒ (*rmpoly list* × *rmpoly list*) *list* ⇒ (*assumps* × *rat list*) *list*
  where *construct-rhs-vector-rec-M p assumps* [] = [(*assumps*, [])]
  | *construct-rhs-vector-rec-M p assumps* ((*qs1*, *qs2*)#[]) =
    (*let TQ-list* = *construct-NofI-M p assumps qs1 qs2 in*
    map (λ(*new-assumps*, *tq*). (*new-assumps*, [*tq*])) *TQ-list*)
  | *construct-rhs-vector-rec-M p assumps* ((*qs1*, *qs2*)#*T*) =
    concat (*let TQ-list* = *construct-NofI-M p assumps qs1 qs2 in*
   (*map* (λ(*new-assumps*, *tq*). (*let rec* = *construct-rhs-vector-rec-M p new-assumps T in*
     map (λ*r*. (*fst r*, *tq*#*snd r*)) *rec*)) *TQ-list*))

definition *construct-rhs-vector-M*:: *rmpoly* ⇒ *assumps* ⇒ *rmpoly list* ⇒ (*nat list* ∗ *nat list*) *list* ⇒ (*assumps* × *rat vec*) *list*
  where *construct-rhs-vector-M p assumps qs Is* =
  map (λ*res*. (*fst res*, *vec-of-list* (*snd res*))) (*construct-rhs-vector-rec-M p assumps* (*pull-out-pairs qs Is*))

definition *solve-for-lhs-single-M*:: *rmpoly* ⇒ *rmpoly list* ⇒ (*nat list* ∗ *nat list*) *list* ⇒ *rat mat* ⇒ *rat vec* ⇒ *rat vec*
  where *solve-for-lhs-single-M p qs subsets matr rhs-vector* =

mult-mat-vec (*matr-option* (*dim-row matr*) (*mat-inverse-var matr*)) *rhs-vector*

definition *solve-for-lhs-M*:: *rmpoly* ⇒ *assumps* ⇒ *rmpoly list* ⇒ (*nat list* ∗ *nat list*) *list* ⇒ *rat mat* ⇒ (*assumps* × *rat vec*) *list*
where *solve-for-lhs-M p assumps qs subsets matr* =
map (*λrhs*. (*fst rhs*, *solve-for-lhs-single-M p qs subsets matr* (*snd rhs*))) (*construct-rhs-vector-M p assumps qs subsets*)

14.3 Reduction

fun *reduce-system-single-M*:: *rmpoly* ⇒ *rmpoly list* ⇒ (*assumps* × *matrix-equation*) ⇒ (*assumps* × *matrix-equation*) *list*
where *reduce-system-single-M p qs* (*assumps*, (*m*,*subs*,*signs*)) =
map (*λlhs*. (*fst lhs*, *reduction-step-R m signs subs* (*snd lhs*))) (*solve-for-lhs-M p assumps qs subs m*)

fun *reduce-system-M*:: *rmpoly* ⇒ *rmpoly list* ⇒ (*assumps* × *matrix-equation*) *list* ⇒ (*assumps* × *matrix-equation*) *list*
where *reduce-system-M p qs input-list* = *concat* (*map* (*reduce-system-single-M p qs*) *input-list*)

14.4 Top-level Function

fun *calculate-data-M*:: *rmpoly* ⇒ *rmpoly list* ⇒ (*assumps* × *matrix-equation*) *list*
where
calculate-data-M p qs =
(*let len* = *length qs in*
if len = *0 then map* (*λ*(*assumps*,(*a*,(*b*,*c*))). (*assumps*, (*a*,*b*,*map* (*drop 1*) *c*))) (*reduce-system-M p* [*1*] *base-case-info-M*)
else if len ≤ *1 then reduce-system-M p qs base-case-info-M*
else
(*let q1* = *take* (*len div 2*) *qs*; *left* = *calculate-data-M p q1*;
q2 = *drop* (*len div 2*) *qs*; *right* = *calculate-data-M p q2*;
comb = *combine-systems-M p q1 left q2 right in*
reduce-system-M p (*fst comb*) (*snd comb*)
)
)

fun *calculate-data-assumps-M*:: *rmpoly* ⇒ *rmpoly list* ⇒ *assumps* ⇒ (*assumps* × *matrix-equation*) *list*
where
calculate-data-assumps-M p qs init-assumps =
(*let len* = *length qs in*
if len = *0 then map* (*λ*(*assumps*,(*a*,(*b*,*c*))). (*assumps*, (*a*,*b*,*map* (*drop 1*) *c*))) (*reduce-system-M p* [*1*] (*base-case-info-M-assumps init-assumps*))
else if len ≤ *1 then reduce-system-M p qs* (*base-case-info-M-assumps init-assumps*)
else
(*let q1* = *take* (*len div 2*) *qs*; *left* = *calculate-data-assumps-M p q1 init-assumps*;

```
    q2 = drop (len div 2) qs; right = calculate-data-assumps-M p q2 init-assumps;
      comb = combine-systems-M p q1 left q2 right in
      reduce-system-M p (fst comb) (snd comb)
  )
 )
```

end

theory *Hybrid-Multiv-Algorithm*

imports *Hybrid-Multiv-Matrix*
Virtual-Substitution.ExportProofs

begin

15 Most recent code

```
function lc-assump-generation:: rmpoly ⇒ assumps⇒ (assumps × rmpoly) list
  where lc-assump-generation q assumps =
    (if q = 0 then [(assumps, 0)] else
    (case (lookup-assump-aux (Polynomial.lead-coeff q) assumps) of
     None ⇒
       let zero = lc-assump-generation (one-less-degree q) ((Polynomial.lead-coeff
q, (0::rat)) # assumps);
           one  = ((Polynomial.lead-coeff q, (1::rat)) # assumps, q);
           minus-one  = ((Polynomial.lead-coeff q, (−1::rat)) # assumps, q) in
         one#minus-one#zero
     | (Some i) ⇒
       (if i = 0 then lc-assump-generation (one-less-degree q) assumps
       else
        [(assumps, q)]
       )
  ))
  by auto
termination apply (relation  measure (λ(q, assumps). (let w = (if q ≠ 0 then 1
else 0) in w + Polynomial.degree q)))
    apply (auto) using one-less-degree-degree
  apply (smt (verit, del-insts) Multiv-Poly-Props.one-less-degree-def Polynomial.coeff-diff
Polynomial.lead-coeff-monom cancel-comm-monoid-add-class.diff-cancel coeff-eq-0
degree-monom-eq leading-coeff-neq-0 linorder-neqE-nat)
  using one-less-degree-degree
  by (smt (verit) Multiv-Poly-Props.one-less-degree-def Polynomial.coeff-diff Poly-
nomial.lead-coeff-monom cancel-comm-monoid-add-class.diff-cancel coeff-eq-0 de-
gree-monom-eq leading-coeff-neq-0 linorder-neqE-nat)
```

declare *lc-assump-generation.simps*[*simp del*]

value *lc-assump-generation* ([:*Var 1*:]::*rmpoly*) [(*Var 1*, *1*)]

fun *lc-assump-generation-list*:: *rmpoly list* ⇒ *assumps* ⇒ (*assumps* × *rmpoly list*) *list*
  where *lc-assump-generation-list* [] *assumps* = [(*assumps*, [])]
  | *lc-assump-generation-list* (*q*#*qs*) *assumps* = (*let rec* = *lc-assump-generation q assumps in*
    concat (*map* (
      λ(*new-assumps*, *r*). (*let list-rec* = *lc-assump-generation-list qs new-assumps in*
      map (λ*elem*. (*fst elem*, *r*#(*snd elem*))) *list-rec*)) *rec*))

declare *lc-assump-generation-list.simps*[*simp del*]

value *lc-assump-generation-list* [([:*Var 1*:]::*rmpoly*), ([:*Var 1*:]::*rmpoly*)] []

value (*lc-assump-generation-list* [([:*Var 1*:]::*rmpoly*)] []) ! *1*

definition *poly-p*:: *rmpoly list* ⇒ *rmpoly*
  where *poly-p qs* = (*let prod-list* = *prod-list qs in*
    prod-list∗(*pderiv prod-list*))

primrec *check-all-const-deg-gen*:: (′*a*::*zero*) *Polynomial.poly list* ⇒ *bool*
  where *check-all-const-deg-gen* [] = *True*
  | *check-all-const-deg-gen* (*h*#*T*) = (*if Polynomial.degree h* = *0 then* (*check-all-const-deg-gen T*) *else False*)

primrec *prod-list-var-gen*:: (′*a*::*idom*) *list* ⇒ (′*a*::*idom*)
  where *prod-list-var-gen* [] = *1*
  | *prod-list-var-gen* (*h*#*T*) = (*if h* = *0 then* (*prod-list-var-gen T*) *else* (*h*∗ *prod-list-var-gen T*))

fun *poly-p-in-branch*:: (*assumps* × *rmpoly list*) ⇒ *rmpoly*
  where *poly-p-in-branch* (*assumps*, *qs*) =
 (*if* (*check-all-const-deg-gen qs* = *True*) *then* [:*0*, *1*:] *else*
  (*pderiv* (*prod-list-var-gen qs*)) ∗ (*prod-list-var-gen qs*)
)

fun *limit-points-on-branch*:: (*assumps* × *rmpoly list*) ⇒ (*rat list* × *rat list*)
  where *limit-points-on-branch* (*assumps*, *qs*) =
    (*map* (λ*q*. *if q* = *0 then 0 else* (*rat-of-int* ∘ *Sturm-Tarski.sign*) (*lookup-assump* (*Polynomial.lead-coeff q*) *assumps*)) *qs*,
    map (λ*q*. *if q* = *0 then 0 else* (*rat-of-int* ∘ *Sturm-Tarski.sign*) (*lookup-assump* (*Polynomial.lead-coeff q*) *assumps*)∗(−*1*)^(*Polynomial.degree q*)) *qs*)

fun *extract-signs*:: (*assumps* × *matrix-equation*) *list* ⇒ (*assumps* × *rat list list*)

list
where *extract-signs qs* = *map* (λ(*assumps*, (*mat* , (*subs*, *signs*))). (*assumps*, *signs*)) *qs*

fun *sign-determination-inner*:: *rmpoly list* ⇒ *assumps* ⇒ (*assumps* × *rat list list*) *list*
where *sign-determination-inner qs assumps* =
(*let branches* = *lc-assump-generation-list qs assumps in*
concat (*map* (λ*branch*.
let poly-p-branch = *poly-p-in-branch branch*;
(*pos-limit-branch*, *neg-limit-branch*) = *limit-points-on-branch branch*;
calculate-data-branch = *extract-signs* (*calculate-data-assumps-M poly-p-branch* (*snd branch*) (*fst branch*))
in map (λ(*a*, *signs*). (*a*, *pos-limit-branch*#*neg-limit-branch*#*signs*)) *calculate-data-branch*
) *branches*
))

fun *extract-polys*:: *atom fm* ⇒ *real mpoly list*
where *extract-polys* (*Atom* (*Less p*)) = [*p*] |
extract-polys (*Atom* (*Leq p*)) = [*p*] |
extract-polys (*Atom* (*Eq p*)) = [*p*] |
extract-polys (*Atom* (*Neq p*)) = [*p*] |
extract-polys (*TrueF*) = [] |
extract-polys (*FalseF*) = [] |
extract-polys (*And* φ ψ) = (*extract-polys* φ)@ (*extract-polys* ψ) |
extract-polys (*Or* φ ψ) = (*extract-polys* φ)@(*extract-polys* ψ) |
extract-polys (*Neg* φ) = (*extract-polys* φ) |
extract-polys (*ExN 0* φ) = (*extract-polys* φ) |
extract-polys (*AllN 0* φ) = (*extract-polys* φ) |
extract-polys - = []

fun *lookup-sem-M* :: *atom fm* ⇒ (*real mpoly* × *rat*) *list* ⇒ *bool option*
where
lookup-sem-M TrueF ls = *Some* (*True*)
| *lookup-sem-M FalseF ls* = *Some* (*False*)
| *lookup-sem-M* (*And l r*) *ls* = (*case* (*lookup-sem-M l ls*, *lookup-sem-M r ls*)
of (*Some i*, *Some j*) ⇒ *Some* (*i* ∧ *j*)
| - ⇒ *None*)
| *lookup-sem-M* (*Or l r*) *ls* = (*case* (*lookup-sem-M l ls*, *lookup-sem-M r ls*)
of (*Some i*, *Some j*) ⇒ *Some* (*i* ∨ *j*)
| - ⇒ *None*)
| *lookup-sem-M* (*Neg l*) *ls* = (*case* (*lookup-sem-M l ls*)
of Some i ⇒ *Some* ((¬*i*))
| - ⇒ *None*)
| *lookup-sem-M* (*Atom* (*Less p*)) *ls* =
(*case* (*lookup-assump-aux p ls*) *of*
Some i ⇒ *Some* (*i* < *0*)
| - ⇒ *None*

```
    )
  | lookup-sem-M (Atom (Leq p)) ls =
    (case (lookup-assump-aux p ls) of
    Some i ⇒ Some (i ≤ 0)
   | - ⇒ None
    )
  | lookup-sem-M (Atom (Eq p)) ls =
    (case (lookup-assump-aux p ls) of
    Some i ⇒ Some (i = 0)
   | - ⇒ None
    )
  | lookup-sem-M (Atom (Neq p)) ls =
    (case (lookup-assump-aux p ls) of
    Some i ⇒ Some (i ≠ 0)
   | - ⇒ None
    )
  | lookup-sem-M (ExN 0 l) ls = lookup-sem-M l ls
  | lookup-sem-M (AllN 0 l) ls = lookup-sem-M l ls
  | lookup-sem-M - ls = None

fun assump-to-atom:: (real mpoly × rat) ⇒ atom
  where assump-to-atom (p, r) =
      (if r = 0 then (Eq p)
    else (if r < 0 then (Less p)
    else (Less (−p))
  ))

fun assump-to-atom-fm:: assumps ⇒ atom fm
  where assump-to-atom-fm [] = TrueF
 | assump-to-atom-fm ((p, r)#T) = And (Atom (assump-to-atom (p, r))) (assump-to-atom-fm
T)

fun create-disjunction:: (assumps × rat list list) list  ⇒ atom fm
  where create-disjunction [] = FalseF
  | create-disjunction ((a, -)#T) = Or (assump-to-atom-fm a) (create-disjunction
T)

fun elim-forall:: atom fm ⇒ atom fm
  where elim-forall F =
  (
    let qs = extract-polys F;
        univ-qs = univariate-in qs 0;
        reindexed-univ-qs = map (map-poly (lowerPoly 0 1)) univ-qs;
        data = sign-determination-inner reindexed-univ-qs [];
        new-data = filter (λ(assumps, signs-list).
          list-all (λ signs.
          let paired-signs = zip qs signs in
            lookup-sem-M F paired-signs = (Some True))
          signs-list
```

```
        ) data
  in create-disjunction new-data
  )

definition elim-exist:: atom fm ⇒ atom fm
  where elim-exist F = Neg (elim-forall (Neg F))

fun structural-complexity:: atom fm ⇒ (nat × nat)
  where
    structural-complexity TrueF = (0, 1)
  | structural-complexity FalseF = (0, 1)
  | structural-complexity (Atom a) = (0, 1)
  | structural-complexity (And F1 F2) =
  (let (qF1, sF1) = structural-complexity F1;
    (qF2, sF2) = structural-complexity F2
  in (qF1 + qF2, 1 + sF1 + sF2))
  | structural-complexity (Or F1 F2) =
  (let (qF1, sF1) = structural-complexity F1;
    (qF2, sF2) = structural-complexity F2
  in (qF1 + qF2, 1 + sF1 + sF2))
  | structural-complexity (Neg F) =
  (let (qF, sF) = structural-complexity F
  in (qF, 1 + sF))
  | structural-complexity (ExQ F) =
  (let (qF, sF) = structural-complexity F
  in (1+ qF, 1 + sF))
  | structural-complexity (AllQ F) =
  (let (qF, sF) = structural-complexity F
  in (1+ qF, 1 + sF))
  | structural-complexity (ExN n F) =
  (let (qF, sF) = structural-complexity F
  in (2 + n+qF, 2+n+sF))
  | structural-complexity (AllN n F) =
  (let (qF, sF) = structural-complexity F
  in (2 + n+qF, 2+n+sF))

declare structural-complexity.simps[simp del]


fun qe:: atom fm ⇒ atom fm
  where
    qe TrueF = TrueF
  | qe FalseF = FalseF
  | qe (Atom a) = (Atom a)
  | qe (And F1 F2) = And (qe F1) (qe F2)
  | qe (Or F1 F2) = Or (qe F1) (qe F2)
  | qe (Neg F) = Neg (qe F)
  | qe (ExQ F) = elim-exist (qe F)
  | qe (AllQ F) = elim-forall (qe F)
```

| *qe* (*AllN n F*) = (*elim-forall* ^^ *n*) (*qe F*)
| *qe* (*ExN n F*) = (*elim-exist* ^^ *n*) (*qe F*)

definition *qe-with-VS*:: *atom fm* ⇒ *atom fm*
where *qe-with-VS F* = (*qe* ∘ *VSLEG*) *F*

value ((*MPoly* (*Pm-fmap* (*fmap-of-list* [(*Pm-fmap* (*fmap-of-list* []), *1*)])))::*real mpoly*)
= *Const 1*

fun *eval-ground* :: *atom fm* ⇒ *real list* ⇒ *bool* **where**
eval-ground (*Atom a*) Γ = *aEval a* Γ |
eval-ground (*TrueF*) - = *True* |
eval-ground (*FalseF*) - = *False* |
eval-ground (*And* φ ψ) Γ = ((*eval-ground* φ Γ) ∧ (*eval-ground* ψ Γ)) |
eval-ground (*Or* φ ψ) Γ = ((*eval-ground* φ Γ) ∨ (*eval-ground* ψ Γ)) |
eval-ground (*Neg* φ) Γ = (¬ (*eval-ground* φ Γ))

value *VSLEG* (*ExQ* (*ExQ* (*Atom* (*Less* (*Var 0*^*2*∗*Var 1* ::*real mpoly*)))))
value (*qe-with-VS* (*ExQ* (*ExQ* (*Atom* (*Less* (*Var 0*^*2*∗*Var 1* ::*real mpoly*))))))

16 Decision Portion

fun *extract-polys-from-assumps*:: *assumps* ⇒ *real mpoly list*
where *extract-polys-from-assumps* [] = []
| *extract-polys-from-assumps* ((*p*, *i*)#*T*) = *p*#(*extract-polys-from-assumps T*)

fun *assumps-are-consistent*:: *assumps* ⇒ *rat list list* ⇒ *bool*
where *assumps-are-consistent assump ls* = ((*map snd assump*) ∈ *set ls*)

fun *find-consistent-signs-at-roots-single-M*:: (*assumps* × *matrix-equation*) ⇒ *rat list list*
where *find-consistent-signs-at-roots-single-M* (*assumps*, (*M*, (*subsets*, *signs*))) = *signs*

fun *find-consistent-signs-at-roots-M*:: (*assumps* × *matrix-equation*) *list* ⇒ *rat list list*
where *find-consistent-signs-at-roots-M l* = *concat* (*map find-consistent-signs-at-roots-single-M l*)

16.1 Limit Points and Helper Functions

definition *expand-signs-list*:: *real mpoly list* ⇒ *rat list list* ⇒ (*real mpoly* × *rat*) *list list*
  where *expand-signs-list qs csas* = *map* (*λcsa. zip qs csa*) *csas*

fun *first-nonzero-coefficient-degree-helper*:: (*real mpoly* × *rat*) *list* ⇒ *real mpoly list* ⇒ *nat* ⇒ (*nat* × *rat*)
  where *first-nonzero-coefficient-degree-helper assumps* [] *n* = (*n*, *0*)
  | *first-nonzero-coefficient-degree-helper assumps* (*h* # *T*) *n* =
    (*case lookup-assump-aux h assumps of*
        (*Some i*) ⇒ (*if i* ≠ *0 then* (*n*, *i*) *else first-nonzero-coefficient-degree-helper assumps T* (*n*−*1*))
        | *None* ⇒ *first-nonzero-coefficient-degree-helper assumps T* (*n*−*1*))

fun *first-nonzero-coefficient-degree-helper-simp*:: (*real mpoly* × *rat*) *list* ⇒ *real mpoly list* ⇒ (*nat* × *rat*)
  where *first-nonzero-coefficient-degree-helper-simp assumps* [] = (*0*, *0*)
  | *first-nonzero-coefficient-degree-helper-simp assumps* (*h* # *T*) =
    (*case lookup-assump-aux h assumps of*
      (*Some i*) ⇒ (*if i* ≠ *0 then* (*length T*, *i*) *else first-nonzero-coefficient-degree-helper-simp assumps T*)
        | *None* ⇒ *first-nonzero-coefficient-degree-helper-simp assumps T*)

lemma *first-nonzero-coefficient-degree-helper-simp*:
  shows *first-nonzero-coefficient-degree-helper-simp assumps ell*
    = *first-nonzero-coefficient-degree-helper assumps ell* (*length ell* − *1*)
proof (*induct ell*)
  case *Nil*
  then show *?case*
    by *auto*
next
  case (*Cons a ell*)
  moreover {
    assume ∗: *lookup-assump-aux a assumps* = *Some 0*
    then have *first-nonzero-coefficient-degree-helper-simp assumps* (*a* # *ell*) =
      first-nonzero-coefficient-degree-helper assumps (*a* # *ell*) (*length* (*a* # *ell*) − *1*)
      using *Cons.hyps*
      by *simp*
  }
  moreover {
    assume ∗: ∃ *k*≠*0*. *lookup-assump-aux a assumps* = *Some k*
    then obtain *k* **where** *k-prop*: *k*≠*0* ∧ *lookup-assump-aux a assumps* = *Some k*
      by *auto*
    then have *first-nonzero-coefficient-degree-helper-simp assumps* (*a* # *ell*) =
      first-nonzero-coefficient-degree-helper assumps (*a* # *ell*) (*length* (*a* # *ell*) − *1*)
      using *Cons.hyps*

by *simp*
}
moreover {
assume ∗: *lookup-assump-aux a assumps* = *None*
then have *first-nonzero-coefficient-degree-helper-simp assumps* (*a* # *ell*) =
first-nonzero-coefficient-degree-helper assumps (*a* # *ell*) (*length* (*a* # *ell*) − *1*)
by (*simp add*: *local.Cons*)
}
ultimately have *first-nonzero-coefficient-degree-helper-simp assumps* (*a* # *ell*) =
first-nonzero-coefficient-degree-helper assumps (*a* # *ell*) (*length* (*a* # *ell*) − *1*)
by *fastforce*
then show *?case* **by** *auto*
qed

declare *pull-out-pairs.simps* [*simp del*]
declare *construct-rhs-vector-rec-M.simps* [*simp del*]

declare *first-nonzero-coefficient-degree-helper.simps*[*simp del*]
declare *first-nonzero-coefficient-degree-helper-simp.simps*[*simp del*]

definition *sign-and-degree-of-first-nonzero-coefficient*:: (*real mpoly* × *rat*) *list* ⇒ *rmpoly* ⇒ (*nat* × *rat*)
where *sign-and-degree-of-first-nonzero-coefficient assumps q* =
first-nonzero-coefficient-degree-helper assumps (*rev* (*Polynomial.coeffs q*)) ((*length* (*Polynomial.coeffs q*)) − *1*)

definition *sign-and-degree-of-first-nonzero-coefficient-simp*:: (*real mpoly* × *rat*) *list* ⇒ *rmpoly* ⇒ (*nat* × *rat*)
where *sign-and-degree-of-first-nonzero-coefficient-simp assumps q* =
first-nonzero-coefficient-degree-helper-simp assumps (*rev* (*Polynomial.coeffs q*))

lemma *sign-and-degree-of-first-nonzero-coefficient-simp*:
sign-and-degree-of-first-nonzero-coefficient assumps q = *sign-and-degree-of-first-nonzero-coefficient-simp assumps q*
using *first-nonzero-coefficient-degree-helper-simp*
by (*simp add*: *sign-and-degree-of-first-nonzero-coefficient-def sign-and-degree-of-first-nonzero-coefficient-simp-*

definition *sign-and-degree-of-first-nonzero-coefficient-list*:: *rmpoly list* ⇒ (*real mpoly* × *rat*) *list* ⇒ (*nat* × *rat*) *list*
where *sign-and-degree-of-first-nonzero-coefficient-list qs assumps* =
map (*λq. sign-and-degree-of-first-nonzero-coefficient-simp assumps q*) *qs*

fun *all-pos-limit-points*:: *rmpoly list* ⇒ *rat list list* ⇒ *rat list list*
where *all-pos-limit-points qs coeffs-signs* =
(*if qs* = [] *then* []

else (*if* (*all-coeffs qs* = []) *then* ([*map* (λ*x*. *0*) *qs*])
  else
  (*let expand-coeffs-signs* = *expand-signs-list* (*all-coeffs qs*) *coeffs-signs in*
  map ((*map snd*) ∘ *sign-and-degree-of-first-nonzero-coefficient-list qs*) *expand-coeffs-signs*)))

fun *all-neg-limit-points-aux*:: (*nat* × *rat*) *list* ⇒ *rat list*
  where *all-neg-limit-points-aux deg-sign-list* = *map* (λ(*deg*, *sgn*). (−*1*)^*deg*∗*sgn*) *deg-sign-list*

fun *all-neg-limit-points*:: *rmpoly list* ⇒ *rat list list* ⇒ *rat list list*
  where *all-neg-limit-points qs coeffs-signs* =
  (*let expand-coeffs-signs* = *expand-signs-list* (*all-coeffs qs*) *coeffs-signs*;
  (*sgn-and-deg-list*::(*nat* × *rat*) *list list*) = *map* (*sign-and-degree-of-first-nonzero-coefficient-list qs*) *expand-coeffs-signs*
    in map all-neg-limit-points-aux sgn-and-deg-list)

16.2 Top-level functions QE

definition *transform*:: *real mpoly list* ⇒ *real mpoly Polynomial.poly list*
  where *transform qs* = (*let vs* = *variables-list qs in*
   map (λ*q*. (*mpoly-to-mpoly-poly-alt* (*nth vs* (*length vs* − *1*)) *q*)) *qs*)

fun *calculate-data-to-signs*:: (*assumps* × *matrix-equation*) *list* ⇒ (*assumps* × *rat list list*) *list*
  where *calculate-data-to-signs ell* = *map* (λ*x*. (*fst x*, *snd* (*snd* (*snd x*)))) *ell*

fun *sum-list*:: *nat list* ⇒ *nat*
  where *sum-list* [] = *0*
  | *sum-list* (*a* # *ell*) = *a* + (*sum-list ell*)

fun *limit-point-data*:: (*rat* × *nat*) *list* ⇒ (*rat list* × *rat list*)
  where *limit-point-data ell* = (*map fst ell*, *map* (λ*x*. *fst x* ∗ (−*1*)^(*snd x*)) *ell*)

fun *generate-signs-and-assumptions*:: *rmpoly list* ⇒ (*assumps* × *rat list list*) *list*
  where *generate-signs-and-assumptions qs-univ* =
    (*let p* = *poly-p qs-univ*; *calc-data* = *calculate-data-M p qs-univ in* [])

export-code *calculate-data-assumps-M qe VSLEG add mult C V pow minus*
  real-of-int real-mult real-plus real-minus real-div print-mpoly
  eval-ground
  in *SML* **module-name** *export*

end

theory *Multiv-Tarski-Query*
  imports
    Multiv-Pseudo-Remainder-Sequence

Hybrid-Multiv-Matrix

begin

definition *sign-rat*::′*a*::{*zero*,*linorder*} ⇒ *rat* **where**
sign-rat n = *rat-of-int* (*Sturm-Tarski.sign n*)

17 Connect multivariate Tarski queries to univariate

lemma *cast-sgn-same-map*:
shows *map of-rat* (*map sgn ell*) = *map sgn ell*
by *simp*

lemma *changes-cast-sgn-same-map*:
shows *changes* ((*map of-rat ell*)::*real list*) = *changes* (*ell*::*rat list*)
proof (*induct length ell arbitrary*: *ell rule*: *less-induct*)
case *less*
then have *indhyp*: ∀ *ell1*. *length ell1* < *length ell* ⟶ *changes* (*map real-of-rat ell1*) = *changes ell1*
by *blast*
{
assume ∗: *length ell* = *0*
then have *changes* (*map real-of-rat ell*) = *changes ell*
by *auto*
}
moreover {
assume ∗: *length ell* = *1*
then have ∃ *h*. *ell* = [*h*]
by (*simp add*: *length-Suc-conv*)
then have *changes* (*map real-of-rat ell*) = *changes ell*
by *auto*
}
moreover {
assume ∗: *length ell* > *1*
then have ∃ *elem1 elem2 ell1*. *ell* = *elem1* # (*elem2* # *ell1*)
by (*metis One-nat-def Suc-le-length-iff le-simps*(*1*) *length-Cons less-Suc-eq-le*)
then obtain *elem1 elem2 ell1* **where** *ell-prop*: *ell* = *elem1* # (*elem2* # *ell1*)
by *auto*
have *len-lt*: *length* (*elem1* # *ell1*) < *length ell*
by (*simp add*: *ell-prop*)
then have *h1*: *changes* (*map real-of-rat* (*elem1* # *ell1*)) = *changes* (*elem1* # *ell1*)
using *indhyp*
by *blast*
have *h2*: *changes* (*map real-of-rat* (*elem2* # *ell1*)) = *changes* (*elem2* # *ell1*)
using *indhyp*
by (*metis len-lt list.size*(*4*))

have *h3*: *real-of-rat elem1* ∗ *real-of-rat elem2* < *0* ⟹ *elem2* ≠ *0* ⟹ *elem1* ∗ *elem2* < *0*
by (*metis of-rat-less-0-iff of-rat-mult*)
have *h4*: ¬ *real-of-rat elem1* ∗ *real-of-rat elem2* < *0* ⟹ *elem2* ≠ *0* ⟹ *elem1* ∗ *elem2* < *0* ⟹ *False*
by (*metis of-rat-less-0-iff of-rat-mult*)
have *changes* (*map real-of-rat* (*elem1* # (*elem2* # *ell1*))) = *changes* (*elem1* # (*elem2* # *ell1*))
using *h1 h2 h3 h4* **by** *auto*
then have *changes* (*map real-of-rat ell*) = *changes ell*
using *ell-prop* **by** *auto*
}
ultimately show *?case*
by (*meson less-one linorder-neqE-nat*)
qed

lemma *spmods-multiv-lc-auxNone*:
assumes *inset*: (*assumps*, *sturm-seq*) ∈ *set* (*spmods-multiv p q acc*)
assumes *pnonz*: *p* ≠ *0*
assumes *lookup-none*: (*lookup-assump-aux* (*Polynomial.lead-coeff p*) *acc*) = *None*
shows (*assumps*, *sturm-seq*) ∈ *set* (*spmods-multiv* (*one-less-degree p*) *q* ((*Polynomial.lead-coeff p*, (*0*::*rat*)) # *acc*))
∨ (∃ *k* ≠ *0*. (*assumps*, *sturm-seq*) ∈ *set* (*spmods-multiv-aux p q* ((*Polynomial.lead-coeff p*, *k*) # *acc*)))
proof −
have (*assumps*, *sturm-seq*) ∈ *set* (*spmods-multiv* (*one-less-degree p*) *q* ((*Polynomial.lead-coeff p*, (*0*::*rat*)) # *acc*))
∨ ((*assumps*, *sturm-seq*) ∈ *set* (*spmods-multiv-aux p q* ((*Polynomial.lead-coeff p*, (*1*::*rat*)) # *acc*))) ∨
(*assumps*, *sturm-seq*) ∈ *set* (*spmods-multiv-aux p q* ((*Polynomial.lead-coeff p*, (−*1*::*rat*)) # *acc*))
using *lookup-none inset pnonz spmods-multiv.simps*[*of p q acc*]
by *auto*
then show *?thesis*
using *class-field.zero-not-one class-field.neg-1-not-0*
by *metis*
qed

lemma *spmods-multiv-lc-auxSome1*:
assumes *inset*: (*assumps*, *sturm-seq*) ∈ *set* (*spmods-multiv p q acc*)
assumes *pnonz*: *p* ≠ *0*
assumes *lookup-some*: (*lookup-assump-aux* (*Polynomial.lead-coeff p*) *acc*) = *Some 0*
shows (((*Polynomial.lead-coeff p*), *0*) ∈ *set acc* ∧ (*assumps*, *sturm-seq*) ∈ *set* (*spmods-multiv* (*one-less-degree p*) *q acc*))
using *assms spmods-multiv.simps*[*of p q acc*]
using *in-set-member lookup-assum-aux-mem option.simps*(*5*) **by** *fastforce*

lemma *spmods-multiv-lc-auxSome2*:

assumes *inset*: (*assumps*, *sturm-seq*) ∈ *set* (*spmods-multiv p q acc*)
assumes *pnonz*: $p \neq 0$
assumes *lookup-some*: (*lookup-assump-aux* (*Polynomial.lead-coeff p*) *acc*) = *Some i* ∧ $i \neq 0$
shows (∃ $k \neq 0$. (((*Polynomial.lead-coeff p*), *k*) ∈ *set acc* ∧ (*assumps*, *sturm-seq*) ∈ *set* (*spmods-multiv-aux p q acc*)))
using *assms spmods-multiv.simps*[*of p q acc*] *lookup-assum-aux-mem*
by (*smt* (*verit*, *best*) *in-set-member option.simps*(*5*))

lemma *spmods-multiv-lc-aux*:
assumes *inset*: (*assumps*, *sturm-seq*) ∈ *set* (*spmods-multiv p q acc*)
assumes *pnonz*: $p \neq 0$
shows (∃ *accum*. (((*Polynomial.lead-coeff p*), *0*) ∈ *set accum* ∧ (*assumps*, *sturm-seq*) ∈ *set* (*spmods-multiv* (*one-less-degree p*) *q accum*)))
∨ (∃ *accum*. (∃ $k \neq 0$. ((((*Polynomial.lead-coeff p*), *k*) ∈ *set accum*) ∧ (*assumps*, *sturm-seq*) ∈ *set* (*spmods-multiv-aux p q accum*))))
proof −
have (*lookup-assump-aux* (*Polynomial.lead-coeff p*) *acc*) = *None* ∨ (∃ *k*. (*lookup-assump-aux* (*Polynomial.lead-coeff p*) *acc*) = *Some k*)
by *auto*
{**assume** ∗: (*lookup-assump-aux* (*Polynomial.lead-coeff p*) *acc*) = *None*
then have (∃ *accum*. (((*Polynomial.lead-coeff p*), *0*) ∈ *set accum* ∧ (*assumps*, *sturm-seq*) ∈ *set* (*spmods-multiv* (*one-less-degree p*) *q accum*)))
∨ (∃ *accum*. (∃ $k \neq 0$. ((((*Polynomial.lead-coeff p*), *k*) ∈ *set accum*) ∧ (*assumps*, *sturm-seq*) ∈ *set* (*spmods-multiv-aux p q accum*))))
using *spmods-multiv-lc-auxNone*
by (*meson inset list.set-intros*(*1*) *pnonz*)
} **moreover** {**assume** ∗: (∃ *k*. (*lookup-assump-aux* (*Polynomial.lead-coeff p*) *acc*) = *Some k*)
then have (∃ *accum*. (((*Polynomial.lead-coeff p*), *0*) ∈ *set accum* ∧ (*assumps*, *sturm-seq*) ∈ *set* (*spmods-multiv* (*one-less-degree p*) *q accum*)))
∨ (∃ *accum*. (∃ $k \neq 0$. ((((*Polynomial.lead-coeff p*), *k*) ∈ *set accum*) ∧ (*assumps*, *sturm-seq*) ∈ *set* (*spmods-multiv-aux p q accum*))))
using *spmods-multiv-lc-auxSome2*
by (*metis inset pnonz spmods-multiv-lc-auxSome1*)
} **moreover** {**assume** ∗: (*lookup-assump-aux* (*Polynomial.lead-coeff p*) *acc*) = *Some 0*
then have (∃ *accum*. (((*Polynomial.lead-coeff p*), *0*) ∈ *set accum* ∧ (*assumps*, *sturm-seq*) ∈ *set* (*spmods-multiv* (*one-less-degree p*) *q accum*)))
∨ (∃ *accum*. (∃ $k \neq 0$. ((((*Polynomial.lead-coeff p*), *k*) ∈ *set accum*) ∧ (*assumps*, *sturm-seq*) ∈ *set* (*spmods-multiv-aux p q accum*))))
by (*metis inset pnonz spmods-multiv-lc-auxSome1*)
}
ultimately show *?thesis*
by *blast*
qed

lemma *spmods-multiv-lc*:

assumes *inset*: (*assumps*, *sturm-seq*) ∈ *set* (*spmods-multiv p q acc*)
assumes *lc-inset*: *lc* ∈ *set* (*lead-coeffs sturm-seq*)
shows∃ *r*. (*lc*,*r*) ∈ *set assumps* ∧ *r* ≠ *0*
using *assms*
proof (*induct p q acc arbitrary*:*assumps sturm-seq lc rule*: *spmods-multiv.induct*)
case *ih*: (*1 p q acc*)
have *p* = *0* ∨ *p* ≠ *0* **by** *auto*
moreover {
assume ∗: *p* = *0*
then have *sturm-seq* = []
using *ih.prems*(*1*) *spmods-multiv.simps* **by** *fastforce*
then have *set* (*lead-coeffs sturm-seq*) = {} **by** *auto*
then have *False*
using *ih.prems*(*2*) **by** *force*
}
moreover {
assume ∗: *p* ≠ *0*
moreover {
assume ∗∗: (*lookup-assump-aux* (*Polynomial.lead-coeff p*) *acc*) = *None*
then have (*assumps*, *sturm-seq*) ∈ *set* (*spmods-multiv* (*one-less-degree p*) *q* ((*Polynomial.lead-coeff p*, (*0*::*rat*)) # *acc*))
∨ (∃ *k* ≠ *0*. (*assumps*, *sturm-seq*) ∈ *set* (*spmods-multiv-aux p q* ((*Polynomial.lead-coeff p*, *k*) # *acc*)))
using ∗ *spmods-multiv-lc-auxNone*[*of assumps sturm-seq p q acc*]
ih(*3*) **by** *auto*
moreover {
assume *eo*: (*assumps*, *sturm-seq*) ∈ *set* (*spmods-multiv* (*one-less-degree p*) *q* ((*Polynomial.lead-coeff p*, (*0*::*rat*)) # *acc*))
then have ∃ *r*. (*lc*,*r*) ∈ *set assumps* ∧ *r* ≠ *0*
using *ih* ∗ ∗∗ **by** *blast*
}
moreover {
assume *eo*: (∃ *k* ≠ *0*. (*assumps*, *sturm-seq*) ∈ *set* (*spmods-multiv-aux p q* ((*Polynomial.lead-coeff p*, *k*) # *acc*)))
then obtain *k* **where** *k-prop*: *k* ≠ *0* ∧ (*assumps*, *sturm-seq*) ∈ *set* (*spmods-multiv-aux p q* ((*Polynomial.lead-coeff p*, *k*) # *acc*))
by *auto*
then have ∃ *r*. (*lc*,*r*) ∈ *set assumps* ∧ *r* ≠ *0*
using ∗ *spmods-multiv-aux-sturm-lc*[*of p k* ((*Polynomial.lead-coeff p*, *k*) # *acc*) *assumps sturm-seq q*] *lc-inset*
using *ih.prems*(*2*) **by** *auto*
}
ultimately have ∃ *r*. (*lc*,*r*) ∈ *set assumps* ∧ *r* ≠ *0*
by *blast*
}
moreover {
assume ∗∗: ∃ *i*. (*lookup-assump-aux* (*Polynomial.lead-coeff p*) *acc*) = *Some i*
then obtain *i* **where** *i-prop*: (*lookup-assump-aux* (*Polynomial.lead-coeff p*) *acc*) = *Some i*

```
      by auto
    moreover {
      assume i: i = 0
     then have (((Polynomial.lead-coeff p), 0) ∈ set acc  ∧ (assumps, sturm-seq)
∈ set (spmods-multiv (one-less-degree p) q acc))
        using ∗ i-prop spmods-multiv-lc-auxSome1[of assumps sturm-seq p q acc]
ih(3)
        by auto
      then have ∃ r. (lc,r) ∈ set assumps ∧ r ≠ 0 using ∗ ih(4) i-prop i
          ih(2)[of i assumps sturm-seq lc] by auto
    }
    moreover {
      assume i: i ≠ 0
       then have h1: (∃ accum. (∃ k ≠ 0. ((((Polynomial.lead-coeff p), k)) ∈ set
accum ∧ (assumps, sturm-seq) ∈ set (spmods-multiv-aux p q accum))))
        using ∗ i-prop spmods-multiv-lc-auxSome2[of assumps sturm-seq p q acc]
ih(3)
        by auto
      then have ∃ r. (lc,r) ∈ set assumps ∧ r ≠ 0
        using  spmods-multiv-aux-sturm-lc[of p i acc assumps sturm-seq]
      proof −
          have f1: ∀ p r ps psa psb pa pb. ((Polynomial.lead-coeff p, r) ∉ set ps
∨ r = 0 ∨ (psa, psb) ∉ set (spmods-multiv-aux p pa ps) ∨ pb ∉ set psb) ∨ (∃ r.
(Polynomial.lead-coeff pb, r) ∈ set psa ∧ r ≠ 0)
          using spmods-multiv-aux-sturm-lc by blast
        obtain rr :: real mpoly Polynomial.poly ⇒ (real mpoly × rat) list ⇒ rat
where
            ∀ x0 x3. (∃ v7. (Polynomial.lead-coeff x0, v7) ∈ set x3 ∧ v7 ≠ 0) =
((Polynomial.lead-coeff x0, rr x0 x3) ∈ set x3 ∧ rr x0 x3 ≠ 0)
          by moura
          then have f2: ∀ p r ps psa psb pa pb. ((Polynomial.lead-coeff p, r) ∉
set ps ∨ r = 0 ∨ (psa, psb) ∉ set (spmods-multiv-aux p pa ps) ∨ pb ∉ set psb) ∨
(Polynomial.lead-coeff pb, rr pb psa) ∈ set psa ∧ rr pb psa ≠ 0
          using f1 by presburger
       obtain pp :: real mpoly Polynomial.poly set ⇒ (real mpoly Polynomial.poly
⇒ real mpoly) ⇒ real mpoly ⇒ real mpoly Polynomial.poly where
          f3: ∀ x0 x1 x2. (∃ v3. x2 = x1 v3 ∧ v3 ∈ x0) = (x2 = x1 (pp x0 x1 x2)
∧ pp x0 x1 x2 ∈ x0)
          by moura
        have lc ∈ Polynomial.lead-coeff ‘ set sturm-seq
          by (smt (z3) ih.prems(2) list.set-map)
          then have f4: lc = Polynomial.lead-coeff (pp (set sturm-seq) Polyno-
mial.lead-coeff lc) ∧ pp (set sturm-seq) Polynomial.lead-coeff lc ∈ set sturm-seq
          using f3 by (meson imageE)
       then have (Polynomial.lead-coeff (pp (set sturm-seq) Polynomial.lead-coeff
lc), rr (pp (set sturm-seq) Polynomial.lead-coeff lc) assumps) ∈ set assumps ∧ rr
(pp (set sturm-seq) Polynomial.lead-coeff lc) assumps ≠ 0
          using f2 by (meson h1)
        then show ?thesis
```

        using *f4* **by** *auto*
      qed
    }
    ultimately have ∃ *r*. (*lc*,*r*) ∈ *set assumps* ∧ *r* ≠ *0*
      by *blast*
  }
  ultimately have ∃ *r*. (*lc*,*r*) ∈ *set assumps* ∧ *r* ≠ *0*
    by *blast*
}
ultimately show *?case* **by** *blast*
qed

lemma *map-eq-2*:
  assumes ∀ *i* < *n*. *f i* = *g i*
  shows *map* (λ*i*. *f i*) [*0*..<*n*] = *map* (λ*i*. *g i*) [*0*..<*n*]
  by (*simp add*: *assms*)

lemma *changes-eq*:
  shows *changes q* = *changes* (*map real-of-int q*)
proof (*induct length q arbitrary*: *q*)
  case *0*
  then show *?case* **by** *auto*
next
  case (*Suc x*)
  then have *ind*: ∀ *q*. *x* = *length q* ⟶ *changes q* = *changes* (*map real-of-int q*)
    by *auto*
  have *Suc x* = *length q*
    using *Suc.hyps*(*2*) **by** *blast*
  have *x-zer*: *x* = *0* ⟹ *changes q* = *changes* (*map real-of-int q*)
  proof −
    assume *x* = *0*
    then have ∃ *a*. *q* = [*a*]
      using *Suc.hyps*(*2*)
      by (*metis length-Suc-conv nat.distinct*(*1*) *remdups-adj.cases*)
    then obtain *a* **where** *a-prop*: *q* = [*a*] **by** *auto*
    have *changes* [*a*] = *changes* (*map real-of-int* [*a*]) **by** *auto*
    then show *changes q* = *changes* (*map real-of-int q*)
      using *a-prop* **by** *auto*
  qed
  have *x-nonz*: *x* ≠ *0* ⟹ *changes q* = *changes* (*map real-of-int q*)
  proof −
    assume *x* ≠ *0*
    then have ∃ *a b c*. *q* = *a*#*b*#*c*
      using *Suc.hyps*(*2*)
      by (*metis Suc-length-conv list-decode.cases*)
    then obtain *a b c* **where** *abc-prop*: *q* = *a*#*b*#*c* **by** *auto*
    have *changes* (*a*#*b*#*c*) = *changes* (*map real-of-int* (*a*#*b*#*c*))
      apply (*auto*)
      using *Suc.hyps*(*1*) *Suc.hyps*(*2*) *abc-prop* **apply** *force*

apply (*simp add*: *mult-less-0-iff*)
apply (*simp add*: *Suc.hyps*(*1*) *Suc.hyps*(*2*) *Suc-inject abc-prop*)
apply (*metis of-int-less-0-iff of-int-mult*)
by (*simp add*: *Suc.hyps*(*1*) *Suc.hyps*(*2*) *Suc-inject abc-prop*)
then show *changes q* = *changes* (*map real-of-int q*)
using *abc-prop* **by** *auto*
qed
then show *?case* **using** *x-zer x-nonz* **by** *auto*
qed

lemma *eval-mpoly-commutes-helper*:
assumes *val-sat*: $\bigwedge$*p n*. (*p*,*n*) ∈ *set assumps* ⟹ *satisfies-evaluation val p n*
assumes *inset*: (*assumps*, *sturm-seq*) ∈ *set* (*spmods-multiv p q acc*)
shows *i* < *length sturm-seq* ⟹ *eval-mpoly val* (*Polynomial.lead-coeff* (*sturm-seq* ! *i*)) = *Polynomial.lead-coeff* (*eval-mpoly-poly val* (*sturm-seq* ! *i*))
proof −
fix *i*
assume *i-lt*: *i* < *length sturm-seq*
have *s1*: *eval-mpoly val* (*map Polynomial.lead-coeff sturm-seq* ! *i*) = *eval-mpoly val* (*Polynomial.lead-coeff* (*sturm-seq* ! *i*))
by (*simp add*: *i-lt*)
have *s2*: *Polynomial.lead-coeff* (*map* (*eval-mpoly-poly val*) *sturm-seq* ! *i*) = *Polynomial.lead-coeff* (*eval-mpoly-poly val* (*sturm-seq* ! *i*))
using *i-lt* **by** *force*
let *?ssi* = (*sturm-seq* ! *i*)
have *Polynomial.lead-coeff* (*sturm-seq* ! *i*) ∈ *set* (*lead-coeffs sturm-seq*)
using *i-lt* **by** *force*
then have *ex-s*: ∃ *s* ≠ *0*. (*Polynomial.lead-coeff ?ssi*, *s*) ∈ *set assumps*
using *spmods-multiv-aux-sturm-lc*
by (*metis* (*no-types*, *lifting*) *inset spmods-multiv-lc*)
then have *eval-mpoly val* (*Polynomial.lead-coeff ?ssi*) ≠ *0*
using *val-sat*[*of Polynomial.lead-coeff* (*sturm-seq* ! *i*) *0*]
unfolding *satisfies-evaluation-def*
using *satisfies-evaluation-nonzero val-sat* **by** *blast*
then show *eval-mpoly val* (*Polynomial.lead-coeff ?ssi*) = *Polynomial.lead-coeff* (*eval-mpoly-poly val ?ssi*)
by (*metis ex-s degree-valuation eval-mpoly-map-poly-comm-ring-hom.base.coeff-map-poly-hom eval-mpoly-poly-def val-sat*)
qed

lemma *changes-R-smods-multiv-connect-aux*:
assumes *inset*: (*assumps*, *sturm-seq*) ∈ *set* (*spmods-multiv p q acc*)
assumes *degree-list*: *degree-list* = *degrees sturm-seq*

assumes *signs-list*: *signs-list* ∈ *mpoly-consistent-sign-vectors* (*lead-coeffs sturm-seq*) (*all-lists* (*length val*))

assumes *val-sat*: ∀ *p n*. ((*p*,*n*) ∈ *set assumps* ⟶ *satisfies-evaluation val p n*)

assumes *key*: *signs-list* = *map* (*λx*. *sign-rat* (*eval-mpoly val x*)) (*lead-coeffs sturm-seq*)
shows *changes-R-smods-multiv signs-list degree-list* = *changes-R-smods-multiv-val sturm-seq val*
proof −
let *?eval-ss* = *eval-mpoly-poly-list val sturm-seq*
have *signs-list* ∈ (*map-mpoly-sign* (*lead-coeffs sturm-seq*)) ' {(*ls*::*real list*). *length ls* = *length val*}
using *signs-list* **unfolding** *mpoly-consistent-sign-vectors-def all-lists-def*
by *auto*
then have ∃ *l* ∈{(*ls*::*real list*). *length ls* = *length val*}. *signs-list* = *map-mpoly-sign* (*lead-coeffs sturm-seq*) *l*
by *auto*
then obtain *l*::*real list* **where** *l-prop*: *length l* = *length val* ∧ *signs-list* = *map-mpoly-sign* (*lead-coeffs sturm-seq*) *l*
by *auto*
then have *l1*: *length signs-list* = *length sturm-seq*
unfolding *map-mpoly-sign-def* **by** *auto*
have *l2*: *length degree-list* = *length sturm-seq*
using *degree-list* **by** *auto*
have *len-eq*: *length degree-list* = *length signs-list* ∧ *length sturm-seq* = *length signs-list*
using *l1 l2* **by** *auto*
have *same-len*: *length degree-list* = *length signs-list*
using *l1 l2* **by** *auto*
have *sminus*: *sminus degree-list signs-list* = *changes-poly-neg-inf ?eval-ss*
proof −
have *len-eq2*: *length* (*eval-mpoly-poly-list val sturm-seq*) = *length degree-list*
using *l2* **unfolding** *eval-mpoly-poly-list-def*
by *auto*
have *dhelp*:⋀*i*. *i* < *length degree-list* ⟹ (*sgn* (*signs-list* ! *i*)) = *sgn* ((*Polynomial.lead-coeff* (*eval-mpoly-poly-list val sturm-seq* ! *i*))::*real*)
proof −
fix *i*
assume *i-lt*: *i* < *length degree-list*
have (*signs-list* ! *i*) = *sign-rat* (*eval-mpoly val* ((*lead-coeffs sturm-seq*) ! *i*))
using *signs-list*
using *i-lt len-eq*
by (*metis* (*no-types*, *lifting*) *key length-map nth-map*)
then have *h1*: (*sgn* (*signs-list* ! *i*)) = *sign-rat* (*eval-mpoly val* ((*lead-coeffs sturm-seq*) ! *i*))
by (*simp add*: *Sturm-Tarski.sign-def sign-rat-def*)
have *helper1*: *sign-rat* (*eval-mpoly val* ((*lead-coeffs sturm-seq*) ! *i*)) =
sign-rat (*eval-mpoly val* (*Polynomial.lead-coeff* (*sturm-seq* ! *i*)))
using *i-lt len-eq* **by** *auto*
have *helper2*: *sign-rat* (*Polynomial.lead-coeff* (*map* (*map-poly* (*eval-mpoly val*)) *sturm-seq* ! *i*)) =
sign-rat (*Polynomial.lead-coeff* (*eval-mpoly-poly val* (*sturm-seq* ! *i*)))

using *i-lt eval-mpoly-poly-def eval-mpoly-poly-list-def len-eq2* **by** *force*
have *helper3*: *sign-rat* (*eval-mpoly val* (*Polynomial.lead-coeff* (*sturm-seq* ! *i*)))
= *sign-rat* (*Polynomial.lead-coeff* (*eval-mpoly-poly val* (*sturm-seq* ! *i*)))
using *val-sat inset eval-mpoly-commutes-helper*
by (*metis i-lt len-eq*)
then have *sign-rat* (*eval-mpoly val* (*Polynomial.lead-coeff* (*sturm-seq* ! *i*)))
= *sign-rat* ((*Polynomial.lead-coeff* (*eval-mpoly-poly-list val sturm-seq* ! *i*))::*real*)
unfolding *eval-mpoly-poly-list-def eval-mpoly-poly-def*
by (*simp add*: *helper2 helper3*)
then have (*of-rat* ∘ *sign-rat*) (*eval-mpoly val* (*Polynomial.lead-coeff* (*sturm-seq*
! *i*))) = *sgn* ((*Polynomial.lead-coeff* (*eval-mpoly-poly-list val sturm-seq* ! *i*))::*real*)
using *sgn-sign-eq*
by (*simp add*: *Sturm-Tarski.sign-def sgn-real-def sign-rat-def*)
then have (*of-rat* ∘ *sign-rat*) (*eval-mpoly val* ((*lead-coeffs sturm-seq*) ! *i*)) =
sgn ((*Polynomial.lead-coeff* (*eval-mpoly-poly-list val sturm-seq* ! *i*))::*real*)
by (*simp add*: *helper1*)
then show (*sgn* (*signs-list* ! *i*)) = *sgn* ((*Polynomial.lead-coeff* (*eval-mpoly-poly-list*
val sturm-seq ! *i*))::*real*)
using *h1 of-int-hom.hom-one of-int-minus sign-rat-def*
by (*smt* (*z3*) *comp-eq-dest-lhs of-rat-0 of-rat-1 of-rat-neg-one of-real-0 of-real-1*
of-real-minus sgn-if)
qed
have *dhelp2*: $\bigwedge$*i*. *i* < *length degree-list* ⟹ *Polynomial.degree* (*eval-mpoly-poly-list*
val sturm-seq ! *i*) = *Polynomial.degree* (*sturm-seq* ! *i*)
proof −
fix *i*
assume *i* < *length degree-list*
then have *i-lt*: *i* < *length sturm-seq* **using** *len-eq* **by** *auto*
then have *s1*: *eval-mpoly val* (*map Polynomial.lead-coeff sturm-seq* ! *i*) =
eval-mpoly val (*Polynomial.lead-coeff* (*sturm-seq* ! *i*))
by *simp*
have *s2*: *Polynomial.lead-coeff* (*map* (*eval-mpoly-poly val*) *sturm-seq* ! *i*) =
Polynomial.lead-coeff (*eval-mpoly-poly val* (*sturm-seq* ! *i*))
using *i-lt* **by** *force*
let *?ssi* = (*sturm-seq* ! *i*)
have *Polynomial.lead-coeff* (*sturm-seq* ! *i*) ∈ *set* (*lead-coeffs sturm-seq*)
using *i-lt* **by** *force*
then have ∃ *k*≠*0*. (*Polynomial.lead-coeff ?ssi*, *k*) ∈ *set assumps*
using *spmods-multiv-lc*[*of assumps sturm-seq p q acc Polynomial.lead-coeff*
(*sturm-seq* ! *i*)] *inset*
by *blast*
then have *eval-mpoly val* (*Polynomial.lead-coeff ?ssi*) ≠ *0*
using *val-sat* **unfolding** *satisfies-evaluation-def*
using *satisfies-evaluation-nonzero val-sat* **by** *blast*
then show *Polynomial.degree* (*eval-mpoly-poly-list val sturm-seq* ! *i*) = *Poly-*
nomial.degree (*sturm-seq* ! *i*)
unfolding *eval-mpoly-poly-list-def eval-mpoly-poly-def*
by (*metis* (*no-types*, *lifting*) *Ring-Hom-Poly.degree-map-poly i-lt degree-0*
map-poly-0 nth-map)

qed
have *d1*: $\bigwedge i$. *even* (*Polynomial.degree* (*eval-mpoly-poly-list val sturm-seq* ! *i*)) $\Longrightarrow$
i < *length degree-list* $\Longrightarrow$
(*sgn* ((− *1*) ^ *degree-list* ! *i* ∗ *signs-list* ! *i*)) = *sgn* (*Polynomial.lead-coeff* (*eval-mpoly-poly-list val sturm-seq* ! *i*))
proof −
fix *i*
assume *even*: *even* (*Polynomial.degree* (*eval-mpoly-poly-list val sturm-seq* ! *i*))
assume *i-lt-deg*: *i* < *length degree-list*
have *ev*: *even* (*degree-list* ! *i*)
using *dhelp2 even degree-list*
using *i-lt-deg* **by** *auto*
then show (*sgn* ((− *1*) ^ *degree-list* ! *i* ∗ *signs-list* ! *i*)) = *sgn* (*Polynomial.lead-coeff* (*eval-mpoly-poly-list val sturm-seq* ! *i*))
using *dhelp ev i-lt-deg*
by *simp*
qed
have *d2*: $\bigwedge i$. *odd* (*Polynomial.degree* (*eval-mpoly-poly-list val sturm-seq* ! *i*)) $\Longrightarrow$
i < *length degree-list* $\Longrightarrow$
of-rat (*sgn* ((− *1*) ^ *degree-list* ! *i* ∗ *signs-list* ! *i*)) = − *sgn* (*Polynomial.lead-coeff* (*eval-mpoly-poly-list val sturm-seq* ! *i*))
proof −
fix *i*
assume *odd*: *odd* (*Polynomial.degree* (*eval-mpoly-poly-list val sturm-seq* ! *i*))
assume *i-lt-deg*: *i* < *length degree-list*
have *odd1*: *odd* (*degree-list* ! *i*)
using *dhelp2 odd degree-list*
using *i-lt-deg* **by** *auto*
then have (*sgn* ((− *1*) ^ *degree-list* ! *i* ∗ *signs-list* ! *i*)) = − *sgn* (*Polynomial.lead-coeff* (*eval-mpoly-poly-list val sturm-seq* ! *i*))
using *dhelp odd*
by (*simp add*: *i-lt-deg of-rat-hom.hom-uminus*)
then show *real-of-rat* (*sgn* ((− *1*) ^ *degree-list* ! *i* ∗ *signs-list* ! *i*)) =
− *sgn* (*Polynomial.lead-coeff* (*eval-mpoly-poly-list val sturm-seq* ! *i*))
by (*smt* (*verit*, *ccfv-SIG*) *of-rat-0 of-rat-1 of-rat-neg-one of-real-0 of-real-1 of-real-eq-iff of-real-minus sgn-rat-def*)
qed
have ∀ *i* < *length degree-list*. (*of-rat* ((*sgn* ((− *1*) ^ *degree-list* ! *i* ∗ *signs-list* ! *i*))) =
sgn (*if even* (*Polynomial.degree* ((*eval-mpoly-poly-list val sturm-seq*)!*i*)) *then sgn* (*Polynomial.lead-coeff* ((*eval-mpoly-poly-list val sturm-seq*)!*i*)) *else* − *sgn* (*Polynomial.lead-coeff* ((*eval-mpoly-poly-list val sturm-seq*)!*i*))))
using *d1 d2*
by (*smt* (*verit*) *minus-of-real-eq-of-real-iff of-rat-0 of-rat-1 of-rat-hom.eq-iff of-real-0 of-real-1 real-of-rat-sgn sgn-real-def*)
then have *map of-rat* (*map* (λ*i*. (*sgn* ((− *1*) ^ *degree-list* ! *i* ∗ *signs-list* ! *i*)))

[*0..<length degree-list*]) =
map
(λ*i. sgn* (*if even* (*Polynomial.degree* ((*eval-mpoly-poly-list val sturm-seq*)!*i*))
then sgn (*Polynomial.lead-coeff* ((*eval-mpoly-poly-list val sturm-seq*)!*i*)) *else* − *sgn*
(*Polynomial.lead-coeff* ((*eval-mpoly-poly-list val sturm-seq*)!*i*))))
[*0..< length degree-list*]
using *map-eq-2 add.left-neutral length-map map-upt nth-map nth-upt of-int-minus*
by *auto*
then have *map of-rat* (*map sgn* ((*map* (λ*i.* ((− *1*) ^ *degree-list* ! *i* ∗ *signs-list*
! *i*)) [*0..<length degree-list*]))) =
(*map sgn*
(*map* (λ*i. if even* (*Polynomial.degree* ((*eval-mpoly-poly-list val sturm-seq*)!*i*))
then sgn (*Polynomial.lead-coeff* ((*eval-mpoly-poly-list val sturm-seq*)!*i*)) *else*
− *sgn* (*Polynomial.lead-coeff* ((*eval-mpoly-poly-list val sturm-seq*)!*i*)))
[*0..< length degree-list*])::*real list*)
by *auto*
then have *h1*:*map of-rat* (*map sgn* ((*map* (λ*i.* ((− *1*) ^ *degree-list* ! *i* ∗ *signs-list*
! *i*)) [*0..<length degree-list*]))) =
map sgn
(*map* (λ*i. if even* (*Polynomial.degree* ((*eval-mpoly-poly-list val sturm-seq*)!*i*))
then sgn (*Polynomial.lead-coeff* ((*eval-mpoly-poly-list val sturm-seq*)!*i*)) *else*
− *sgn* (*Polynomial.lead-coeff* ((*eval-mpoly-poly-list val sturm-seq*)!*i*)))
[*0..< length* (*eval-mpoly-poly-list val sturm-seq*)])
using *len-eq2*
by *presburger*
have *h2*: (*map* (λ*i. if even* (*Polynomial.degree* ((*eval-mpoly-poly-list val sturm-seq*)!*i*))

then sgn (*Polynomial.lead-coeff* ((*eval-mpoly-poly-list val sturm-seq*)!*i*)) *else*
− *sgn* (*Polynomial.lead-coeff* ((*eval-mpoly-poly-list val sturm-seq*)!*i*)))
[*0..< length* (*eval-mpoly-poly-list val sturm-seq*)]) =
(*map* (λ*p. if even* (*Polynomial.degree p*)
then sgn (*Polynomial.lead-coeff p*) *else*
− *sgn* (*Polynomial.lead-coeff p*))
(*eval-mpoly-poly-list val sturm-seq*))
using *map-upt* **by** *fastforce*
let *?m1* = ((*map* (λ*i.* ((− *1*) ^ *degree-list* ! *i* ∗ *signs-list* ! *i*)) [*0..<length
degree-list*]))
let *?m2* = ((*map* (λ*p. if even* (*Polynomial.degree p*) *then sgn* (*Polynomial.lead-coeff
p*) *else* − *sgn* (*Polynomial.lead-coeff p*))
(*eval-mpoly-poly-list val sturm-seq*)))::*real list*
have *map of-rat* ((*map sgn ?m1*)::*rat list*) = ((*map sgn ?m2*)::*real list*)
unfolding *changes-poly-neg-inf-def sgn-neg-inf-def*
using *h1 h2*
by *presburger*
then have *c1*: *changes* ((*map of-rat*(*map sgn ?m1*))::*real list*) = *changes* ((*map
sgn ?m2*)::*real list*)
using *arg-cong*
by *auto*

have *c2*: *changes* (*map sgn ?m1*) = *changes ?m1*
using *changes-map-sgn-eq*[*of ?m1*] *changes-eq*
by *auto*
have *c3*: *changes* (*map sgn ?m2*::*real list*) = *changes ?m2*
using *changes-map-sgn-eq*[*of ?m2*] *changes-eq*
by *auto*
have *c0*: *changes* (*map* (*real-of-rat* ∘ *sgn*) *?m1*) = *changes* (*map sgn ?m1*)
using *changes-cast-sgn-same-map*
by (*metis list.map-comp*)
then have *changes ?m1* = *changes ?m2*
using *c1 c2 c3 cast-sgn-same-map list.map-comp*
by *metis*
then have *changes* (*map* (λ*i*. (−*1*) ^(*nth degree-list i*)∗(*nth signs-list i*)) [*0*..< *length degree-list*]) =
changes-poly-neg-inf (*eval-mpoly-poly-list val sturm-seq*)
unfolding *changes-poly-neg-inf-def sgn-neg-inf-def*
using *changes-map-sgn-eq*
by *force*
then show *sminus degree-list signs-list* = *changes-poly-neg-inf* (*eval-mpoly-poly-list val sturm-seq*)
by *auto*
qed
have *splus*: *changes signs-list* = *changes-poly-pos-inf ?eval-ss*
proof −

have *eq1*: *map-mpoly-sign* (*lead-coeffs sturm-seq*) *l* =
map (λ*x*. *sign-rat* (*eval-mpoly val x*)) (*lead-coeffs sturm-seq*)
using *l-prop key*
by *auto*
have ∀ *i* < *length sturm-seq*. ((*sign-rat* (*eval-mpoly val* (*nth* (*lead-coeffs sturm-seq*) *i*))) =
sign-rat (*sgn-pos-inf* (*nth* (*eval-mpoly-poly-list val sturm-seq*) *i*)))
proof *clarsimp*
fix *i*
assume *i* < *length sturm-seq*
have *s1*: *eval-mpoly val* (*map Polynomial.lead-coeff sturm-seq* ! *i*) = *eval-mpoly val* (*Polynomial.lead-coeff* (*sturm-seq* ! *i*))
by (*simp add*: ‹*i* < *length sturm-seq*›)
have *s2*: *Polynomial.lead-coeff* (*map* (*eval-mpoly-poly val*) *sturm-seq* ! *i*) = *Polynomial.lead-coeff* (*eval-mpoly-poly val* (*sturm-seq* ! *i*))
using ‹*i* < *length sturm-seq*› **by** *force*
let *?ssi* = (*sturm-seq* ! *i*)
have *s3*: *eval-mpoly val* (*Polynomial.lead-coeff ?ssi*) = *Polynomial.lead-coeff* (*eval-mpoly-poly val ?ssi*)
using *eval-mpoly-commutes-helper*[*of assumps val sturm-seq*] *inset val-sat*
using ‹*i* < *length sturm-seq*› **by** *blast*
have *sign-rat* (*eval-mpoly val* (*map Polynomial.lead-coeff sturm-seq* ! *i*)) = *sign-rat* (*sgn* (*Polynomial.lead-coeff* (*eval-mpoly-poly-list val sturm-seq* ! *i*)))

using *s1 s2 s3*
using *eval-mpoly-poly-list-def*
by (*simp add*: *sign-rat-def*)
then show *sign-rat* (*eval-mpoly val* (*Polynomial.lead-coeff* (*sturm-seq* ! *i*)))
= *sign-rat* (*sgn-pos-inf* (*eval-mpoly-poly-list val sturm-seq* ! *i*))
unfolding *sgn-pos-inf-def*
by (*simp add*: *s1*)
qed
then have *eq2*: *map* (*λx*. *sign-rat* (*eval-mpoly val x*)) (*lead-coeffs sturm-seq*)
=
(*map* (*sign-rat* ∘ *sgn-pos-inf*) (*eval-mpoly-poly-list val sturm-seq*))
by (*smt* (*verit*, *ccfv-threshold*) *comp-eq-dest-lhs eval-mpoly-poly-def eval-mpoly-poly-list-def key l2 length-map list.map-comp nth-map nth-map-conv same-len*)
have *map-mpoly-sign* (*lead-coeffs sturm-seq*) *l* =
(*map* (*sign-rat* ∘ *sgn-pos-inf*) (*eval-mpoly-poly-list val sturm-seq*))
using *eq1 eq2* **by** *auto*
then have *changes*
(*map* (*Sturm-Tarski.sign* ∘
eval-mpoly l)
(*lead-coeffs sturm-seq*)) =
changes
(*map* (*λp*. *sgn* (*Polynomial.lead-coeff p*))
(*eval-mpoly-poly-list val sturm-seq*))
unfolding *changes-poly-pos-inf-def sign-rat-def map-mpoly-sign-def sgn-pos-inf-def*
using *changes-map-sign-of-int-eq list.map-comp*
by (*metis* (*no-types*, *lifting*) *changes-map-sign-eq comp-apply nth-map-conv*)
then have *changes* (*map-mpoly-sign* (*lead-coeffs sturm-seq*) *l*) =
changes-poly-pos-inf (*eval-mpoly-poly-list val sturm-seq*)
by (*metis* (*no-types*, *lifting*) *changes-map-sign-eq changes-map-sign-of-int-eq changes-poly-pos-inf-def list.map-comp map-eq-conv map-mpoly-sign-def sgn-pos-inf-def*)
then show *?thesis* **using** *l-prop* **unfolding** *map-mpoly-sign-def* **by** *auto*
qed
show *?thesis* **using** *sminus splus* **unfolding** *changes-R-smods-multiv-def changes-R-smods-multiv-val-def*
by (*metis*)
qed

lemma *changes-R-smods-multiv-connect*:
assumes *inset*: (*assumps*, *sturm-seq*) ∈ *set* (*spmods-multiv p q acc*)
assumes *degree-list*: *degree-list* = *degrees sturm-seq*

assumes *val-sat*: ∀ *p n*. ((*p*,*n*) ∈ *set assumps* ⟶ *satisfies-evaluation val p n*)

assumes *key*: *signs-list* = *map* (*λx*. *sign-rat* (*eval-mpoly val x*)) (*lead-coeffs sturm-seq*)
shows *changes-R-smods-multiv signs-list degree-list* = *changes-R-smods-multiv-val sturm-seq val*
proof −

have *l1*: *length signs-list* = *length sturm-seq*
using *key* **by** *auto*
then have ((*map* ((λ*x*. *sign-rat* (*eval-mpoly val x*)) ∘ *Polynomial.lead-coeff*) *sturm-seq*)::*rat list*)
∈ (((λ*x*. *map* (*sign-rat* ∘ *eval-mpoly x* ∘ *Polynomial.lead-coeff*) *sturm-seq*) ‘
{*ls*. *length ls* = *length val*})::*rat list set*)
by *auto*
then have *signs-list* ∈ *mpoly-consistent-sign-vectors* (*lead-coeffs sturm-seq*) (*all-lists* (*length val*))
using *key*
unfolding *mpoly-consistent-sign-vectors-def map-mpoly-sign-def*
by (*smt* (*verit*) *all-lists-def comp-apply image-eqI map-eq-conv mem-Collect-eq sign-rat-def*)
then show *?thesis* **using** *assms changes-R-smods-multiv-connect-aux*
by *auto*
qed

lemma *changes-R-smods-multiv-val-univariate*:
assumes (*assumps*, *sturm-seq*) ∈ *set* (*spmods-multiv p q acc*)
assumes ⋀*p n*. (*p*,*n*) ∈ *set assumps* ⟹ *satisfies-evaluation val p n*
shows *changes-R-smods-multiv-val sturm-seq val* = *changes-R-smods* (*eval-mpoly-poly val p*) (*eval-mpoly-poly val q*)
using *matches-ss* **unfolding** *changes-R-smods-def changes-R-smods-multiv-val-def*
using *assms changes-poly-neg-inf-def changes-poly-pos-inf-def eval-mpoly-poly-list-def spmods-smods-sgn-map-eq*(*2*) *spmods-smods-sgn-map-eq*(*3*)
spmods-multiv-sound
by *metis*

lemma *changes-R-smods-multiv-signs-list-connect*:
assumes *len-same*: *length signs-list* = *length degree-list*
assumes *key*: ((*map sign-rat signs-list*)::*rat list*) = (*signs-list-var*::*rat list*)
shows *changes-R-smods-multiv signs-list degree-list* = *changes-R-smods-multiv signs-list-var degree-list*
proof −
have *changes-same*: *changes signs-list* = *changes signs-list-var*
using *key*
using *changes-map-sign-of-int-eq map-eq-conv*
unfolding *sign-rat-def*
by (*metis* (*mono-tags*, *lifting*) *comp-apply*)
let *?ell1* = (*map* (λ*i*. (−*1*)^(*nth degree-list i*)∗(*nth signs-list i*)) [*0*..< *length degree-list*])
let *?ell2* = (*map* (λ*i*. (−*1*)^(*nth degree-list i*)∗(*nth signs-list-var i*)) [*0*..< *length degree-list*])
have ⋀*x*. *x* < *length degree-list* ⟹
sgn ((− *1*) ^ *degree-list* ! *x* ∗ *signs-list* ! *x*) = *sgn* ((− *1*) ^ *degree-list* ! *x* ∗ *map sign-rat signs-list* ! *x*)
proof −
fix *x*

assume *x* < *length degree-list*
have *h1*: *sgn* ((− *1*) ^ *degree-list* ! *x* ∗ *signs-list* ! *x*) = *sgn* ((− *1*) ^ *degree-list* ! *x*) ∗ *sgn* (*signs-list* ! *x*)
using *sgn-mult* **by** *blast*
have *h2*: *sgn* ((− *1*) ^ *degree-list* ! *x* ∗ *map sign-rat signs-list* ! *x*) = *sgn* ((− *1*) ^ *degree-list* ! *x*) ∗ *sgn* (*map sign-rat signs-list* ! *x*)
using *sgn-mult*
by *blast*
have *h3*: *sgn* (*map sign-rat signs-list* ! *x*) = *sgn* (*signs-list* ! *x*)
using *key* **unfolding** *sign-rat-def*
using ‹*x* < *length degree-list*› *len-same sgn-rat-def* **by** *fastforce*
show *sgn* ((− *1*) ^ *degree-list* ! *x* ∗ *signs-list* ! *x*) = *sgn* ((− *1*) ^ *degree-list* ! *x* ∗ *map sign-rat signs-list* ! *x*)
using *h1 h2 h3*
by *metis*
qed
then have *map sgn ?ell1* = *map sgn ?ell2*
using *key*
by *auto*
then have *changes ?ell1* = *changes ?ell2*
using *changes-map-sgn-eq*
by (*metis* (*no-types*, *lifting*))
then show *?thesis*
using *changes-R-smods-multiv-def changes-same sminus.simps* **by** *presburger*
qed

lemma *changes-R-smods-multiv-univariate*:
assumes (*assumps*, *sturm-seq*) ∈ *set* (*spmods-multiv p q acc*)
assumes *degree-list*: *degree-list* = *degrees sturm-seq*

assumes *val-sat*: ∀ *p n*. ((*p*,*n*) ∈ *set assumps* ⟶ *satisfies-evaluation val p n*)

assumes *key*: *map* (*sign-rat*::*rat*⇒*rat*) *signs-list* = *map* (λ*x*. *sign-rat* (*eval-mpoly val x*)) (*lead-coeffs sturm-seq*)
assumes ⋀*p n*. (*p*,*n*) ∈ *set assumps* ⟹ *satisfies-evaluation val p n*
shows *changes-R-smods-multiv signs-list degree-list* = *changes-R-smods* (*eval-mpoly-poly val p*) (*eval-mpoly-poly val q*)
proof −
have *same-len*: *length signs-list* = *length degree-list*
using *degree-list key*
by (*metis length-map*)
from *changes-R-smods-multiv-signs-list-connect*[*OF same-len key*]
have *h2*:
changes-R-smods-multiv signs-list degree-list =
changes-R-smods-multiv ((*map* (λ*x*. *sign-rat* (*eval-mpoly val x*)) (*lead-coeffs sturm-seq*))) *degree-list*
by *auto*
have *h1*: *changes-R-smods-multiv* (*map* (λ*x*. *sign-rat* (*eval-mpoly val x*)) (*lead-coeffs sturm-seq*)) *degree-list* = *changes-R-smods* (*eval-mpoly-poly val p*) (*eval-mpoly-poly*

val q)
  using *changes-R-smods-multiv-connect changes-R-smods-multiv-val-univariate assms* **by** *auto*
 then show *?thesis* **using** *h1 h2* **by** *auto*
qed

theorem *pderiv-commutes*:
 fixes *p*:: *real mpoly Polynomial.poly*
 fixes *val*:: *real list*
 shows *pderiv* (*eval-mpoly-poly val p*) = (*eval-mpoly-poly val* (*pderiv p*))
 by (*simp add*: *eval-mpoly-map-poly-idom-hom.base.map-poly-pderiv eval-mpoly-poly-def*)

theorem *sturm-R-multiv-comm*:
 shows *card* {*x. Polynomial.poly* (*eval-mpoly-poly val p*) *x=0*} = *changes-R-smods* (*eval-mpoly-poly val p*) ((*eval-mpoly-poly val* (*pderiv p*)))
 using *pderiv-commutes Sturm-Tarski.sturm-R*
 by *auto*

theorem *sturm-R-multiv2*:
 assumes *q* = *pderiv p*
 assumes (*assumps*, *sturm-seq*) ∈ *set* (*spmods-multiv p q acc*)
 assumes ⋀*p n.* (*p*,*n*) ∈ *set assumps* ⟹ *satisfies-evaluation val p n*
 shows *card* {*x. Polynomial.poly* (*eval-mpoly-poly val p*) *x=0*} = *changes-R-smods-multiv-val sturm-seq val*
 using *changes-R-smods-multiv-val-univariate Sturm-Tarski.sturm-R sturm-R-multiv-comm*
 using *assms*(*1*) *assms*(*2*) *assms*(*3*)
 by *force*

theorem *restate-tarski-multiv*:
 fixes *p*:: *real mpoly Polynomial.poly*
 fixes *q*:: *real mpoly Polynomial.poly*
 assumes (*eval-mpoly-poly val p*) ≠ *0*
 assumes (*assumps*, *sturm-seq*) ∈ *set* (*spmods-multiv p* ((*pderiv p*)∗*q*) *acc*)
 assumes ⋀*p n.* (*p*,*n*) ∈ *set assumps* ⟹ *satisfies-evaluation val p n*
 shows *changes-R-smods-multiv-val sturm-seq val* =
  int (*card* {*x. Polynomial.poly* (*eval-mpoly-poly val p*) *x=0* ∧ *Polynomial.poly* (*eval-mpoly-poly val q*) *x>0*})
  − *int*(*card* {*x. Polynomial.poly* (*eval-mpoly-poly val p*) *x=0* ∧ *Polynomial.poly* (*eval-mpoly-poly val q*) *x<0*})
proof −
 have *0*: *changes-R-smods-multiv-val sturm-seq val* = *changes-R-smods* (*eval-mpoly-poly val p*) (*eval-mpoly-poly val* ((*pderiv p*)∗*q*))
  using *changes-R-smods-multiv-val-univariate assms*
  by *blast*
 let *?p* = (*eval-mpoly-poly val p*)::*real Polynomial.poly*
 let *?q* = (*eval-mpoly-poly val q*)::*real Polynomial.poly*
 have *h1*: *taq* {*x. poly ?p x=0*} *?q* = *changes-R-smods ?p* (*pderiv ?p* ∗ *?q*)
  using *sturm-tarski-R*[*symmetric*] **by** *auto*

have *pd-comm*: *pderiv* (*eval-mpoly-poly val p*) ∗ *eval-mpoly-poly val q* = *eval-mpoly-poly val* (*pderiv p* ∗ *q*)
using *eval-mpoly-poly-comm-ring-hom.hom-mult pderiv-commutes* **by** *auto*
let *?all* = {*x. Polynomial.poly ?p x=0*}
let *?lt* = {*x. Polynomial.poly ?p x=0* ∧ *Polynomial.poly ?q x* < *0*}
let *?gt* = {*x. Polynomial.poly ?p x=0* ∧ *Polynomial.poly ?q x* > *0*}
let *?eq* = {*x. Polynomial.poly ?p x=0* ∧ *Polynomial.poly ?q x* = *0*}
have *cardlt*: ($\sum$ *x* | *poly* (*eval-mpoly-poly val p*) *x* = *0* ∧
poly (*eval-mpoly-poly val q*) *x* < *0*.
if 0 < *poly* (*eval-mpoly-poly val q*) *x then 1*
else if poly (*eval-mpoly-poly val q*) *x* = *0 then 0*
else − *1*) = −*int* (*card ?lt*)
by *auto*
have *cardgt*: ($\sum$ *x* | *poly* (*eval-mpoly-poly val p*) *x* = *0* ∧
0 < *poly* (*eval-mpoly-poly val q*) *x*.
if 0 < *poly* (*eval-mpoly-poly val q*) *x then 1*
else if poly (*eval-mpoly-poly val q*) *x* = *0 then 0*
else − *1*) = *int* (*card ?gt*)
by *auto*
have *empty*: {*x. poly* (*eval-mpoly-poly val p*) *x* = *0* ∧
poly (*eval-mpoly-poly val q*) *x* < *0*} ∩
{*x. poly* (*eval-mpoly-poly val p*) *x* = *0* ∧
0 < *poly* (*eval-mpoly-poly val q*) *x*} = {}
by *auto*
then have *cardzer*: ($\sum$ *x*∈{*x. poly* (*eval-mpoly-poly val p*) *x* = *0* ∧
poly (*eval-mpoly-poly val q*) *x* < *0*} ∩
{*x. poly* (*eval-mpoly-poly val p*) *x* = *0* ∧
0 < *poly* (*eval-mpoly-poly val q*) *x*}.
if 0 < *poly* (*eval-mpoly-poly val q*) *x then 1*
else if poly (*eval-mpoly-poly val q*) *x* = *0 then 0*
else − *1*) = *0* **by** *auto*
have *eq*: *?all* = *?lt* ∪ *?gt* ∪ *?eq* **by** *force*
from *poly-roots-finite*[*OF assms*(*1*)] **have** *fin*: *finite ?all* **.**
have ($\sum$ *x* | *poly ?p x* = *0*. *Sturm-Tarski.sign* (*poly ?q x*)) = *int* (*card ?gt*) − *int* (*card ?lt*)
unfolding *eq Sturm-Tarski.sign-def*
apply (*subst sum-Un*)
apply (*auto simp add*:*fin*)
apply (*subst sum-Un*)
apply (*simp add*: *fin*)
apply (*simp add*: *fin*)
using *cardzer cardlt cardgt empty*
by *auto*
then have *changes-R-smods* (*eval-mpoly-poly val p*) (*eval-mpoly-poly val* ((*pderiv p*)∗*q*)) = *int* (*card ?gt*) − *int* (*card ?lt*)
using *h1 taq-def pd-comm*
by *metis*
then have *changes-R-smods-multiv-val sturm-seq val* = *int* (*card ?gt*) − *int* (*card ?lt*)

```
    using 0 by auto
  then show ?thesis
    by presburger
qed

lemma sminus-map-sign:
  assumes same-len: length signs-list = length degree-list
  shows sminus degree-list signs-list =
    sminus degree-list (map sign-rat signs-list)
proof −
  let ?xs = (map (λi. (− 1) ^ degree-list ! i ∗ signs-list ! i) [0..<length degree-list])
  have changes-same: changes ?xs = changes (map sign-rat ?xs)
    using changes-map-sign-eq[of ?xs]
    unfolding sign-rat-def
   by (smt (verit, best) Sturm-Tarski.sign-def changes-map-sign-of-int-eq map-eq-conv
o-apply)
  have (map (λi. (−1)^(nth degree-list i)∗(nth (map sign-rat signs-list) i)) [0..<
length degree-list])
 = (map sign-rat ?xs)
  proof clarsimp
    fix x
    assume x < length degree-list
    then have x < length signs-list
      using same-len
      by auto
    then have p2: map sign-rat signs-list ! x =
        sign-rat (signs-list ! x)
      using nth-map by blast
   have p1: (− 1) ^ degree-list ! x = (1::int) ∨ (− 1) ^ degree-list ! x = (−1::int)
      using neg-one-even-power neg-one-odd-power by blast
    show (− 1) ^ degree-list ! x ∗ map sign-rat signs-list ! x =
      (sign-rat ((− 1) ^ degree-list ! x ∗ signs-list ! x))
      using p1 p2
     by (metis (no-types, opaque-lifting) mult-1 mult-minus-left neg-one-even-power
neg-one-odd-power of-int-minus sign-rat-def sign-uminus)
  qed
  then have  changes (map sign-rat ?xs)
= changes (map (λi. (−1)^(nth degree-list i)∗(nth (map sign-rat signs-list) i))
[0..< length degree-list])
    unfolding sign-rat-def
    by presburger
  then have changes (map (λi. (−1)^(nth degree-list i)∗(nth signs-list i)) [0..<
length degree-list])
    = changes (map (λi. (−1)^(nth degree-list i)∗(nth (map sign-rat signs-list) i))
[0..< length degree-list])
    using changes-same
    by (metis (no-types, lifting))
  then show ?thesis by auto
qed
```

lemma *changes-R-smods-multiv-map-sign*:
assumes *length signs-list* = *length degree-list*
shows *changes-R-smods-multiv signs-list degree-list* =
changes-R-smods-multiv (*map sign-rat signs-list*) *degree-list*
using *assms sminus-map-sign*
unfolding *changes-R-smods-multiv-def*
using *changes-R-smods-multiv-def changes-R-smods-multiv-signs-list-connect* **by** *presburger*

lemma *construct-NofI-single-M-univariate-superset*:
assumes *new-p*: *new-p* = *sum-list* (*map* (λ*x*. *x*^*2*) (*p* # *I1*))
assumes *new-q*: *new-q* = ((*pderiv new-p*)∗(*prod-list I2*))
assumes *seq-in*: (*assumps*, *sturm-seq*) ∈ *set* (*spmods-multiv new-p new-q acc*)
assumes *superset*: *set assumps* ⊆ *set assumps-superset*
assumes *good-val*: ⋀*p n*. (*p*,*n*) ∈ *set assumps-superset* ⟹ *satisfies-evaluation val p n*
shows *construct-NofI-single-M* (*assumps-superset*, *sturm-seq*) =
(*assumps-superset*, *construct-NofI-R* (*eval-mpoly-poly val p*) (*eval-mpoly-poly-list val I1*) (*eval-mpoly-poly-list val I2*))
proof −
let *?other-p* = *sum-list* (*map power2* (*eval-mpoly-poly val p* # *eval-mpoly-poly-list val I1*))
have *eval-mpoly-poly val* (*sum-list* (*map power2 I1*)) = *sum-list* (*map power2* (*eval-mpoly-poly-list val I1*))
proof (*induct I1*)
case *Nil*
then show *?case*
by (*simp add*: *eval-mpoly-poly-list-def*)
next
case (*Cons a I1*)
then show *?case*
by (*simp add*: *eval-mpoly-poly-comm-ring-hom.hom-add eval-mpoly-poly-comm-ring-hom.hom-power eval-mpoly-poly-list-def*)
qed
then have *eval1*: *?other-p* = *eval-mpoly-poly val new-p*
using *new-p eval-mpoly-poly-comm-ring-hom.hom-add eval-mpoly-poly-comm-ring-hom.hom-power*
by *auto*
then have *eval2*: (*pderiv ?other-p* ∗ *prod-list* (*eval-mpoly-poly-list val I2*)) = *eval-mpoly-poly val new-q*
using *new-q eval-mpoly-poly-comm-ring-hom.hom-mult*
by (*simp add*: *eval-mpoly-poly-list-def pderiv-commutes*)
let *?new-signs* = (*map* ((λ*lc*. *case lookup-assump-aux lc assumps of None* ⇒ *1000* | *Some i* ⇒ *i*) ∘
Polynomial.lead-coeff) *sturm-seq*)

have *lookup-some*: ⋀*ss-poly*.
ss-poly ∈ *set sturm-seq* ⟹ ∃ *i*. *lookup-assump-aux* (*Polynomial.lead-coeff ss-poly*) *assumps* = *Some i*

proof −
fix *ss-poly*
assume *ss-poly* ∈ *set sturm-seq*
then show ∃ *i*. *lookup-assump-aux* (*Polynomial.lead-coeff ss-poly*) *assumps* = *Some i*
using *seq-in spmods-multiv-sturm-lc*[*of assumps sturm-seq new-p new-q acc ss-poly*]
inset-means-lookup-assump-some
by *metis*
qed
have ⋀*ss-poly*. *ss-poly* ∈ *set sturm-seq* ⟹
(∃ *i*. *lookup-assump-aux* (*Polynomial.lead-coeff ss-poly*) *assumps* = *Some i* ∧
sign-rat i = *sign-rat* (*eval-mpoly val* (*Polynomial.lead-coeff ss-poly*)))
proof −
fix *ss-poly*
assume *ss-poly* ∈ *set sturm-seq*
then have ∃ *i*. *lookup-assump-aux* (*Polynomial.lead-coeff ss-poly*) *assumps* = *Some i*
using *lookup-some* **by** *auto*
then obtain *i* **where** *i-prop*: *lookup-assump-aux* (*Polynomial.lead-coeff ss-poly*) *assumps* = *Some i*
by *auto*
then have ((*Polynomial.lead-coeff ss-poly*), *i*) ∈ *set assumps*
using *lookup-assump-means-inset*[*of* (*Polynomial.lead-coeff ss-poly*) *assumps*]
by (*simp add*: *lookup-assum-aux-mem*)
then have (*sign-rat* (*i*)) = *sign-rat* (*eval-mpoly val* (*Polynomial.lead-coeff ss-poly*))
using *good-val*[*of* (*Polynomial.lead-coeff ss-poly*) *i*] **unfolding** *satisfies-evaluation-def*
by (*metis of-int-hom.injectivity sign-rat-def subsetD superset*)
then show (∃ *i*. *lookup-assump-aux* (*Polynomial.lead-coeff ss-poly*) *assumps* = *Some i* ∧
sign-rat i = *sign-rat* (*eval-mpoly val* (*Polynomial.lead-coeff ss-poly*)))
using *i-prop* **by** *auto*
qed
then have *help1*: ⋀*ss-poly*. *ss-poly* ∈ *set sturm-seq* ⟹
(∃ *i*. *lookup-assump-aux* (*Polynomial.lead-coeff ss-poly*) *assumps-superset* = *Some i*)
using *superset sign-rat-def good-val in-set-member inset-means-lookup-assump-some lookup-assum-aux-mem satisfies-evaluation-def subset-code*(*1*)
by (*smt* (*verit*, *ccfv-SIG*))
then have *help2*: ⋀*ss-poly*. ⋀ *i*. (*ss-poly* ∈ *set sturm-seq* ∧ *lookup-assump-aux* (*Polynomial.lead-coeff ss-poly*) *assumps-superset* = *Some i*) ⟹ *sign-rat i* = *sign-rat* (*eval-mpoly val* (*Polynomial.lead-coeff ss-poly*))
proof −
fix *ss-poly*
fix *i*
assume (*ss-poly* ∈ *set sturm-seq* ∧ *lookup-assump-aux* (*Polynomial.lead-coeff ss-poly*) *assumps-superset* = *Some i*)
then show *sign-rat i* = *sign-rat* (*eval-mpoly val* (*Polynomial.lead-coeff ss-poly*))

using *superset sign-rat-def good-val lookup-assum-aux-mem satisfies-evaluation-def*
by (*metis of-int-hom.eq-iff*)
qed
then have *all-ex-i*: ⋀*ss-poly. ss-poly* ∈ *set sturm-seq* ⟹
(∃ *i. lookup-assump-aux* (*Polynomial.lead-coeff ss-poly*) *assumps-superset* = *Some i* ∧
sign-rat i = *sign-rat* (*eval-mpoly val* (*Polynomial.lead-coeff ss-poly*)))
using *help1 help2*
by *blast*
then have *helper*: (*map* ((λ*lc. case lookup-assump-aux lc assumps-superset of None* ⇒ *1000* | *Some i* ⇒ *sign-rat i*) ∘
Polynomial.lead-coeff)
sturm-seq) = *map* (λ*x. sign-rat* (*eval-mpoly val x*)) (*lead-coeffs sturm-seq*)
by *force*
then have *rel1*: (*changes-R-smods-multiv*
(*map* ((λ*lc. case lookup-assump-aux lc assumps-superset of None* ⇒ *1000* | *Some i* ⇒ *sign-rat i*) ∘
Polynomial.lead-coeff)
sturm-seq)
(*degrees sturm-seq*)) = *changes-R-smods-multiv-val sturm-seq val*
using *helper changes-R-smods-multiv-connect*[*of assumps sturm-seq new-p new-q acc degrees sturm-seq val ?new-signs*]
using *assms*
by (*simp add*: *changes-R-smods-multiv-connect subset-code*(*1*))
have *rel2*: *changes-R-smods-multiv-val sturm-seq val* =
changes-R-smods (*eval-mpoly-poly val new-p*) (*eval-mpoly-poly val new-q*)
using *assms changes-R-smods-multiv-val-univariate*[*of assumps sturm-seq new-p new-q acc val*]
eval1 eval2
by *blast*
have *c1*: (*changes-R-smods-multiv*
(*map* ((λ*lc. case lookup-assump-aux lc assumps-superset of None* ⇒ *1000* | *Some i* ⇒ *sign-rat i*) ∘
Polynomial.lead-coeff)
sturm-seq)
(*degrees sturm-seq*)) =
(*changes-R-smods ?other-p* (*pderiv ?other-p* ∗ *prod-list* (*eval-mpoly-poly-list val I2*)))
using *rel1 rel2*
using *eval1 eval2*
by *presburger*
have ⋀*ss-poly. ss-poly* ∈ *set sturm-seq* ⟹
(*case lookup-assump-aux* (*Polynomial.lead-coeff ss-poly*) *assumps-superset of None* ⇒ *1000* | *Some i* ⇒ *sign-rat i*) =
sign-rat (*case lookup-assump-aux* (*Polynomial.lead-coeff ss-poly*) *assumps-superset of None* ⇒ *1000* | *Some i* ⇒ *i*)
proof −
fix *ss-poly*
assume *ss-poly* ∈ *set sturm-seq*

then have ∃ *i*. *lookup-assump-aux* (*Polynomial.lead-coeff ss-poly*) *assumps-superset* = *Some i*
using *lookup-some superset*
using *all-ex-i* **by** *blast*
then obtain *i* **where** *i-prop*: *lookup-assump-aux* (*Polynomial.lead-coeff ss-poly*) *assumps-superset* = *Some i*
by *auto*
then have *eq1*: (*case lookup-assump-aux* (*Polynomial.lead-coeff ss-poly*) *assumps-superset of None* ⇒ *1000* | *Some i* ⇒ *sign-rat i*) = *sign-rat i*
by *auto*
have *eq2*: *sign-rat* (*case lookup-assump-aux* (*Polynomial.lead-coeff ss-poly*) *assumps-superset of None* ⇒ *1000* | *Some i* ⇒ *i*) = *sign-rat i*
using *i-prop* **by** *auto*
then show (*case lookup-assump-aux* (*Polynomial.lead-coeff ss-poly*)
assumps-superset of
None ⇒ *1000* | *Some i* ⇒ *sign-rat i*) =
sign-rat
(*case lookup-assump-aux* (*Polynomial.lead-coeff ss-poly*)
assumps-superset of
None ⇒ *1000* | *Some i* ⇒ *i*)
using *eq1 eq2*
by *fastforce*
qed
then have (*map* ((λ*lc*. *case lookup-assump-aux lc assumps-superset of None* ⇒ *1000* | *Some i* ⇒ *sign-rat i*) ∘
Polynomial.lead-coeff)
sturm-seq)
= (*map sign-rat* (*map* ((λ*lc*. *case lookup-assump-aux lc assumps-superset of None* ⇒ *1000* | *Some i* ⇒ *i*) ∘
Polynomial.lead-coeff)
sturm-seq))
by *auto*
then have (*changes-R-smods-multiv*
(*map* ((λ*lc*. *case lookup-assump-aux lc assumps-superset of None* ⇒ *1000* | *Some i* ⇒ *sign-rat i*) ∘
Polynomial.lead-coeff)
sturm-seq)
(*degrees sturm-seq*))
= (*changes-R-smods-multiv*
(*map sign-rat* (*map* ((λ*lc*. *case lookup-assump-aux lc assumps-superset of None* ⇒ *1000* | *Some i* ⇒ *i*) ∘
Polynomial.lead-coeff)
sturm-seq))
(*degrees sturm-seq*))
by (*smt* (*verit*, *ccfv-threshold*))
then have (*changes-R-smods-multiv*
(*map* ((λ*lc*. *case lookup-assump-aux lc assumps-superset of None* ⇒ *1000* | *Some i* ⇒ *sign-rat i*) ∘
Polynomial.lead-coeff)

sturm-seq)
(degrees sturm-seq)) =
(changes-R-smods-multiv
(map ((λlc. case lookup-assump-aux lc assumps-superset of None ⇒ 1000 | Some i ⇒ i) ∘
Polynomial.lead-coeff)
sturm-seq)
(degrees sturm-seq))
using changes-R-smods-multiv-map-sign
by (metis (no-types, lifting) length-map)
then have c2: (changes-R-smods-multiv
(map ((λlc. case lookup-assump-aux lc assumps-superset of None ⇒ 1000 | Some i ⇒ i) ∘
Polynomial.lead-coeff)
sturm-seq)
(degrees sturm-seq)) =
(changes-R-smods ?other-p (pderiv ?other-p * prod-list (eval-mpoly-poly-list val I2)))
using c1
by presburger
show ?thesis **using** c2
unfolding construct-NofI-R-def
apply (simp) **unfolding** construct-NofI-R-def
by (metis)
qed

lemma construct-NofI-single-M-univariate:
assumes new-p: new-p = sum-list (map (λx. x^2) (p # I1))
assumes new-q: new-q = ((pderiv new-p)*(prod-list I2))
assumes seq-in: (assumps, sturm-seq) ∈ set (spmods-multiv new-p new-q acc)
assumes good-val: ⋀p n. (p,n) ∈ set assumps ⟹ satisfies-evaluation val p n
shows construct-NofI-single-M (assumps, sturm-seq) =
(assumps, construct-NofI-R (eval-mpoly-poly val p) (eval-mpoly-poly-list val I1) (eval-mpoly-poly-list val I2))
proof −
let ?other-p = sum-list (map power2 (eval-mpoly-poly val p # eval-mpoly-poly-list val I1))
have eval-mpoly-poly val (sum-list (map power2 I1)) = sum-list (map power2 (eval-mpoly-poly-list val I1))
proof (induct I1)
case Nil
then show ?case
by (simp add: eval-mpoly-poly-list-def)
next
case (Cons a I1)
then show ?case
by (simp add: eval-mpoly-poly-comm-ring-hom.hom-add eval-mpoly-poly-comm-ring-hom.hom-power eval-mpoly-poly-list-def)

qed
then have *eval1*: *?other-p* = *eval-mpoly-poly val new-p*
using *new-p eval-mpoly-poly-comm-ring-hom.hom-add eval-mpoly-poly-comm-ring-hom.hom-power*
by *auto*
then have *eval2*: (*pderiv ?other-p* ∗ *prod-list* (*eval-mpoly-poly-list val I2*)) = *eval-mpoly-poly val new-q*
using *new-q eval-mpoly-poly-comm-ring-hom.hom-mult*
by (*simp add*: *eval-mpoly-poly-list-def pderiv-commutes*)
let *?new-signs* = (*map* ((λ*lc. case lookup-assump-aux lc assumps of None* ⇒ *1000* | *Some i* ⇒ *i*) ∘
Polynomial.lead-coeff) *sturm-seq*)

have *lookup-some*: ⋀*ss-poly.*
ss-poly ∈ *set sturm-seq* ⟹ ∃ *i. lookup-assump-aux* (*Polynomial.lead-coeff ss-poly*) *assumps* = *Some i*
proof −
fix *ss-poly*
assume *ss-poly* ∈ *set sturm-seq*
then show ∃ *i. lookup-assump-aux* (*Polynomial.lead-coeff ss-poly*) *assumps* = *Some i*
using *seq-in spmods-multiv-sturm-lc*[*of assumps sturm-seq new-p new-q acc ss-poly*]
inset-means-lookup-assump-some
by *metis*
qed
have ⋀*ss-poly. ss-poly* ∈ *set sturm-seq* ⟹
(∃ *i. lookup-assump-aux* (*Polynomial.lead-coeff ss-poly*) *assumps* = *Some i* ∧
sign-rat i = *sign-rat* (*eval-mpoly val* (*Polynomial.lead-coeff ss-poly*)))
proof −
fix *ss-poly*
assume *ss-poly* ∈ *set sturm-seq*
then have ∃ *i. lookup-assump-aux* (*Polynomial.lead-coeff ss-poly*) *assumps* = *Some i*
using *lookup-some* **by** *auto*
then obtain *i* **where** *i-prop*: *lookup-assump-aux* (*Polynomial.lead-coeff ss-poly*) *assumps* = *Some i*
by *auto*
then have ((*Polynomial.lead-coeff ss-poly*), *i*) ∈ *set assumps*
using *lookup-assump-means-inset*[*of* (*Polynomial.lead-coeff ss-poly*) *assumps*]
by (*simp add*: *lookup-assum-aux-mem*)
then have (*sign-rat* (*i*)) = *sign-rat* (*eval-mpoly val* (*Polynomial.lead-coeff ss-poly*))
using *good-val*[*of* (*Polynomial.lead-coeff ss-poly*) *i*] **unfolding** *satisfies-evaluation-def*
by (*metis of-int-hom.injectivity sign-rat-def*)
then show (∃ *i. lookup-assump-aux* (*Polynomial.lead-coeff ss-poly*) *assumps* = *Some i* ∧
sign-rat i = *sign-rat* (*eval-mpoly val* (*Polynomial.lead-coeff ss-poly*)))
using *i-prop* **by** *auto*
qed

then have (*map* ((*λlc. case lookup-assump-aux lc assumps of None* ⇒ *1000* | *Some i* ⇒ *sign-rat i*) ∘
Polynomial.lead-coeff)
sturm-seq) = *map* (*λx. sign-rat* (*eval-mpoly val x*)) (*lead-coeffs sturm-seq*)
by *force*
then have *rel1*: (*changes-R-smods-multiv*
(*map* ((*λlc. case lookup-assump-aux lc assumps of None* ⇒ *1000* | *Some i* ⇒ *sign-rat i*) ∘
Polynomial.lead-coeff)
sturm-seq)
(*degrees sturm-seq*)) = *changes-R-smods-multiv-val sturm-seq val*
using *changes-R-smods-multiv-connect*[*of assumps sturm-seq new-p new-q acc degrees sturm-seq val ?new-signs*]
using *assms*(*3*) *assms*(*4*)
using *changes-R-smods-multiv-connect* **by** *blast*
have *rel2*: *changes-R-smods-multiv-val sturm-seq val* =
changes-R-smods (*eval-mpoly-poly val new-p*) (*eval-mpoly-poly val new-q*)
using *assms changes-R-smods-multiv-val-univariate*[*of assumps sturm-seq new-p new-q acc val*]
eval1 eval2
by *blast*
have *c1*: (*changes-R-smods-multiv*
(*map* ((*λlc. case lookup-assump-aux lc assumps of None* ⇒ *1000* | *Some i* ⇒ *sign-rat i*) ∘
Polynomial.lead-coeff)
sturm-seq)
(*degrees sturm-seq*)) =
(*changes-R-smods ?other-p* (*pderiv ?other-p* ∗ *prod-list* (*eval-mpoly-poly-list val I2*)))
using *rel1 rel2*
using *eval1 eval2*
by *presburger*
have ⋀*ss-poly. ss-poly* ∈ *set sturm-seq* ⟹
(*case lookup-assump-aux* (*Polynomial.lead-coeff ss-poly*) *assumps of None* ⇒ *1000* | *Some i* ⇒ *sign-rat i*) =
sign-rat (*case lookup-assump-aux* (*Polynomial.lead-coeff ss-poly*) *assumps of None* ⇒ *1000* | *Some i* ⇒ *i*)
proof −
fix *ss-poly*
assume *ss-poly* ∈ *set sturm-seq*
then have ∃ *i. lookup-assump-aux* (*Polynomial.lead-coeff ss-poly*) *assumps* = *Some i*
using *lookup-some* **by** *auto*
then obtain *i* **where** *i-prop*: *lookup-assump-aux* (*Polynomial.lead-coeff ss-poly*) *assumps* = *Some i*
by *auto*
then have *eq1*: (*case lookup-assump-aux* (*Polynomial.lead-coeff ss-poly*) *assumps of None* ⇒ *1000* | *Some i* ⇒ *sign-rat i*) = *sign-rat i*
by *auto*

have *eq2*: *sign-rat* (*case lookup-assump-aux* (*Polynomial.lead-coeff ss-poly*) *assumps of None* ⇒ *1000* | *Some i* ⇒ *i*) = *sign-rat i*
using *i-prop* **by** *auto*
then show (*case lookup-assump-aux* (*Polynomial.lead-coeff ss-poly*) *assumps of None* ⇒ *1000* | *Some i* ⇒ *sign-rat i*) =
sign-rat (*case lookup-assump-aux* (*Polynomial.lead-coeff ss-poly*) *assumps of None* ⇒ *1000* | *Some i* ⇒ *i*)
using *eq1 eq2*
by *metis*
qed
then have (*map* ((λ*lc. case lookup-assump-aux lc assumps of None* ⇒ *1000* | *Some i* ⇒ *sign-rat i*) ∘
Polynomial.lead-coeff)
sturm-seq)
= (*map sign-rat* (*map* ((λ*lc. case lookup-assump-aux lc assumps of None* ⇒ *1000* | *Some i* ⇒ *i*) ∘
Polynomial.lead-coeff)
sturm-seq))
by *auto*
then have *changes-1*: (*changes-R-smods-multiv*
(*map* ((λ*lc. case lookup-assump-aux lc assumps of None* ⇒ *1000* | *Some i* ⇒ *sign-rat i*) ∘
Polynomial.lead-coeff)
sturm-seq)
(*degrees sturm-seq*))
= (*changes-R-smods-multiv*
(*map sign-rat* (*map* ((λ*lc. case lookup-assump-aux lc assumps of None* ⇒ *1000* | *Some i* ⇒ *i*) ∘
Polynomial.lead-coeff)
sturm-seq))
(*degrees sturm-seq*))
by (*smt* (*verit*, *ccfv-threshold*))
then have *changes2*: (*changes-R-smods-multiv*
(*map* ((λ*lc. case lookup-assump-aux lc assumps of None* ⇒ *1000* | *Some i* ⇒ *sign-rat i*) ∘
Polynomial.lead-coeff)
sturm-seq)
(*degrees sturm-seq*)) =
(*changes-R-smods-multiv*
(*map* ((λ*lc. case lookup-assump-aux lc assumps of None* ⇒ *1000* | *Some i* ⇒ *i*) ∘
Polynomial.lead-coeff)
sturm-seq)
(*degrees sturm-seq*))
using *changes-R-smods-multiv-map-sign*
by (*metis* (*no-types*, *lifting*) *length-map*)
then have (*changes-R-smods-multiv*
(*map* ((λ*lc. case lookup-assump-aux lc assumps of None* ⇒ *1000* | *Some i* ⇒ *i*) ∘

Polynomial.lead-coeff)
sturm-seq)
(*degrees sturm-seq*)) =
(*changes-R-smods ?other-p* (*pderiv ?other-p* ∗ *prod-list* (*eval-mpoly-poly-list val I2*)))
using *c1*
by *presburger*
then show *?thesis*
unfolding *construct-NofI-R-def* **using** *c1 changes2*
by (*smt* (*verit*, *ccfv-SIG*) *construct-NofI-R-def construct-NofI-single-M-univariate-superset dual-order.refl good-val new-p new-q seq-in*)

qed

lemma *construct-NofI-M-univariate-tarski-query*:
assumes *inset*: (*assumps*, *tarski-query*) ∈ *set* (*construct-NofI-M p acc I1 I2*)
assumes *val*: ⋀*p n*. (*p*,*n*) ∈ *set assumps* ⟹ *satisfies-evaluation val p n*
shows *tarski-query* = *construct-NofI-R* (*eval-mpoly-poly val p*) (*eval-mpoly-poly-list val I1*) (*eval-mpoly-poly-list val I2*)
proof −
let *?ell-map* = *map construct-NofI-single-M* (*construct-NofI-R-spmods p acc I1 I2*)
let *?ell* = *construct-NofI-R-spmods p acc I1 I2*
have *same-len*: *length ?ell* = *length ?ell-map*
by *auto*
have (*assumps*, *tarski-query*) ∈ *set* (*map construct-NofI-single-M* (*construct-NofI-R-spmods p acc I1 I2*))
using *inset*
by (*metis construct-NofI-M.simps*)
then have ∃ *n* < *length ?ell-map*. *?ell-map* ! *n* = (*assumps*, *tarski-query*)
by (*metis construct-NofI-M.simps in-set-conv-nth inset length-map*)
then obtain *n* **where** *n-prop*: *n* < *length ?ell-map* ∧ *?ell-map* ! *n* = (*assumps*, *tarski-query*)
by *auto*
then have *?ell-map* ! *n* = *construct-NofI-single-M* (*?ell* ! *n*)
by *force*
then have *assumps-tq*: *construct-NofI-single-M* (*?ell* ! *n*) = (*assumps*, *tarski-query*)
using *n-prop* **by** *auto*
obtain *input-assumps ss* **where** *tuple-prop*: (*input-assumps*, *ss*) = *?ell* ! *n*
using *n-prop same-len*
by (*metis old.prod.exhaust*)
then have *atq-is*: (*assumps*, *tarski-query*) =
(*let lcs* = *lead-coeffs ss*;
sa-list = *map* (*λlc*. *lookup-assump lc input-assumps*) *lcs*;
degrees-list = *degrees ss in*
(*input-assumps*, *rat-of-int* (*changes-R-smods-multiv sa-list degrees-list*)))
by (*metis assumps-tq construct-NofI-single-M.simps*)
then have *as-is*: *assumps* = *input-assumps*
by *auto*

have *in-spmods-multiv*: (*assumps*, *ss*) ∈ *set* ((*let new-p* = *sum-list* (*map* (λ*x*. *x*^2) (*p* # *I1*)) *in*
spmods-multiv new-p ((*pderiv new-p*)∗(*prod-list I2*))) *acc*)
using *tuple-prop in-set-member*
using *as-is construct-NofI-R-spmods-def n-prop* **by** *auto*
let *?multiv-p* = *sum-list* (*map power2* (*p* # *I1*))
let *?multiv-q* = (*pderiv ?multiv-p* ∗ *prod-list I2*)
let *?univ-p* = *sum-list* (*map power2* (*eval-mpoly-poly val p* # *eval-mpoly-poly-list val I1*))
let *?univ-q* = (*pderiv ?univ-p* ∗ *prod-list* (*eval-mpoly-poly-list val I2*))
let *?signs-list* = *map* (λ*lc*. *lookup-assump lc assumps*) (*lead-coeffs ss*)
have *map-poly* (*eval-mpoly val*) (p^2 + *sum-list* (*map power2 I1*)) =
(*map-poly* (*eval-mpoly val*) *p*)^2 + *sum-list* (*map* (*power2* ∘ *map-poly* (*eval-mpoly val*)) *I1*)
using *eval-mpoly-map-poly-comm-ring-hom.hom-mult eval-mpoly-map-poly-comm-ring-hom.hom-sum*
proof −
have *h1*: *map-poly* (*eval-mpoly val*) (p^2 + *sum-list* (*map power2 I1*)) = (*map-poly* (*eval-mpoly val*) *p*)^2 + *map-poly* (*eval-mpoly val*) (*sum-list* (*map power2 I1*))
using *eval-mpoly-map-poly-comm-ring-hom.hom-add eval-mpoly-map-poly-comm-ring-hom.hom-power* **by** *presburger*
have *map-poly* (*eval-mpoly val*) (*sum-list* (*map power2 I1*)) =
(*sum-list* (*map* (*map-poly* (*eval-mpoly val*)∘*power2*) *I1*))
using *eval-mpoly-map-poly-comm-ring-hom.hom-add*
by (*simp add*: *eval-mpoly-map-poly-comm-ring-hom.hom-sum-list*)
then have *h2*: *map-poly* (*eval-mpoly val*) (*sum-list* (*map power2 I1*)) = *sum-list* (*map* (*power2* ∘ *map-poly* (*eval-mpoly val*)) *I1*)
using *eval-mpoly-map-poly-comm-ring-hom.hom-add eval-mpoly-map-poly-comm-ring-hom.hom-power*
by (*metis* (*mono-tags*, *lifting*) *comp-apply map-eq-conv*)
then show *?thesis*
using *h1 h2* **by** *auto*
qed
then have *p-connect*: *eval-mpoly-poly val ?multiv-p* = *?univ-p*
unfolding *eval-mpoly-poly-def eval-mpoly-poly-list-def*
by *auto*
then have *eval-mpoly-poly val* (*pderiv ?multiv-p*) = *pderiv ?univ-p*
by (*metis pderiv-commutes*)
then have *q-connect*: *eval-mpoly-poly val ?multiv-q* = *?univ-q*
using *eval-mpoly-map-poly-comm-ring-hom.hom-mult*
unfolding *eval-mpoly-poly-list-def*
using *eval-mpoly-poly-comm-ring-hom.hom-mult eval-mpoly-poly-comm-ring-hom.prod-list-map-hom* **by** *presburger*
have *tq-is*: *tarski-query* =
(*let lcs* = *lead-coeffs ss*;
sa-list = *map* (λ*lc*. *lookup-assump lc assumps*) *lcs*;
degrees-list = *degrees ss in*
rat-of-int (*changes-R-smods-multiv sa-list degrees-list*))
using *as-is atq-is* **by** *auto*
have ⋀*x*. *x* ∈ *set ss* ⟹

```
        sign-rat
          (case lookup-assump-aux (Polynomial.lead-coeff x) assumps of None ⇒
1000 | Some i ⇒ i) =
        sign-rat (eval-mpoly val (Polynomial.lead-coeff x))
  proof −
    fix x
    assume x ∈ set ss
    then have ∃ i. (Polynomial.lead-coeff x, i) ∈ set assumps
      using in-spmods-multiv spmods-multiv-sturm-lc[of assumps ss - - acc x]
      by meson
    then obtain i where i-prop: (Polynomial.lead-coeff x, i) ∈ set assumps
      by auto
    then have satisfies-evaluation val (Polynomial.lead-coeff x) i
      using val
      by auto
     then have ⋀j. (Polynomial.lead-coeff x, j) ∈ set assumps ⟹ sign-rat i =
sign-rat j
      using val unfolding satisfies-evaluation-def
      by (metis of-int-hom.injectivity sign-rat-def)
    then have ∃ j. (case lookup-assump-aux (Polynomial.lead-coeff x) assumps of
None ⇒ 1000 | Some i ⇒ i) = j ∧ sign-rat i = sign-rat j
     by (smt (verit, del-insts) i-prop in-set-member inset-means-lookup-assump-some
lookup-assum-aux-mem option.case(2))
    then show sign-rat
          (case lookup-assump-aux (Polynomial.lead-coeff x) assumps of None ⇒
1000 | Some i ⇒ i) =
        sign-rat (eval-mpoly val (Polynomial.lead-coeff x))
      using i-prop val
      by (metis of-int-hom.injectivity satisfies-evaluation-def sign-rat-def)
  qed
  then have map sign-rat (map (λlc. lookup-assump lc assumps) (lead-coeffs ss))
=
    map (λx. sign-rat (eval-mpoly val x)) (lead-coeffs ss)
    using val by auto
 then have changes-R-smods-multiv (map (λlc. lookup-assump lc assumps) (lead-coeffs
ss)) (degrees ss) =
    changes-R-smods (eval-mpoly-poly val (sum-list (map power2 (p # I1))))
     (eval-mpoly-poly val (pderiv (sum-list (map power2 (p # I1))) ∗ prod-list I2))
   using changes-R-smods-multiv-univariate[of assumps ss ?multiv-p ?multiv-q acc
degrees ss val ?signs-list] in-spmods-multiv val
    by (smt (verit, ccfv-SIG))
  then show ?thesis
    unfolding construct-NofI-R-def
    using p-connect q-connect tq-is
    by metis
qed

end
```

theory *Renegar-Modified*

imports *BenOr-Kozen-Reif.Renegar-Decision*

begin

definition *poly-f-nocrb* :: *real poly list* ⇒ *real poly*
  where
    poly-f-nocrb ps =
  (*if* (*check-all-const-deg ps* = *True*) *then* [:*0*, *1*:] *else*
  (*pderiv* (*prod-list-var ps*)) ∗ (*prod-list-var ps*))

lemma *root-set-nocrb*:
  assumes *is-not-const*: *check-all-const-deg qs* = *False*
  shows {*x*. *poly* (*poly-f qs*) *x* = *0*}
 = {*x*. *poly* (*poly-f-nocrb qs*) *x* = *0*} ∪ {−(*crb* (*prod-list-var qs*)), (*crb* (*prod-list-var qs*))}
proof −
  have ∀ *x* ∈ {*x*. *poly* (*poly-f qs*) *x* = *0*}. *poly* (*poly-f qs*) *x* = *0* **by** *auto*
  then have *all*: ∀ *x* ∈ {*x*. *poly* (*poly-f qs*) *x* = *0*}. *poly* ((*pderiv* (*prod-list-var qs*)) ∗ (*prod-list-var qs*)∗ ([:−(*crb* (*prod-list-var qs*)),*1*:]) ∗ ([:(*crb* (*prod-list-var qs*)),*1*:])) *x* = *0*
    unfolding *poly-f-def* **using** *is-not-const* **by** *presburger*
  have *all1*: ∀ *x* ∈ {*x*. *poly* (*poly-f qs*) *x* = *0*}.
    ((*poly* ((*pderiv* (*prod-list-var qs*)) ∗ (*prod-list-var qs*)) *x* = *0*)
     ∨ (*poly* (([:−(*crb* (*prod-list-var qs*)),*1*:]) ∗ ([:(*crb* (*prod-list-var qs*)),*1*:])) *x* = *0*))
  proof *clarsimp*
    fix *x*
    assume *inset*: *poly* (*poly-f qs*) (*real-of-int x*) = *0*
    assume *poly* (*pderiv* (*prod-list-var qs*)) (*real-of-int x*) ≠ *0*
    assume *x* ∗ *x* ≠ *crb* (*prod-list-var qs*) ∗ *crb* (*prod-list-var qs*)
   have *poly* ((*pderiv* (*prod-list-var qs*)) ∗ (*prod-list-var qs*)∗ ([:−(*crb* (*prod-list-var qs*)),*1*:]) ∗ ([:(*crb* (*prod-list-var qs*)),*1*:])) *x* = *0*
      using *all inset* **by** *auto*
    then show *poly* (*prod-list-var qs*) (*real-of-int x*) = *0* **using** *assms*
      by (*smt* (*verit*, *del-insts*) ‹*poly* (*pderiv* (*prod-list-var qs*)) (*real-of-int x*) ≠ *0*› ‹*x* ∗ *x* ≠ *crb* (*prod-list-var qs*) ∗ *crb* (*prod-list-var qs*)› *mult.commute mult-eq-0-iff mult-minus-left of-int-eq-iff of-int-minus poly-mult poly-root-factor*(*3*))
  qed
  have ∀ *x*. (*poly* (([:−(*crb* (*prod-list-var qs*)),*1*:]) ∗ ([:(*crb* (*prod-list-var qs*)),*1*:])) *x* = *0* ⟶
  (*x* = −(*crb* (*prod-list-var qs*)) ∨ *x* = (*crb* (*prod-list-var qs*))))
    by (*simp add*: *square-eq-iff*)
  then have ∀ *x* ∈ {*x*. *poly* (*poly-f qs*) *x* = *0*}.
    ((*poly* (*poly-f-nocrb qs*) *x* = *0*)
    ∨ (*x* = −(*crb* (*prod-list-var qs*)) ∨ *x* = (*crb* (*prod-list-var qs*))))
    using *all1* **unfolding** *poly-f-nocrb-def*

by (*smt* (*verit*, *del-insts*) *all is-not-const of-int-minus poly-root-factor*(*2*))
then have *subset1*: {*x*. *poly* (*poly-f qs*) *x* = *0*}
⊆ {*x*. *poly* (*poly-f-nocrb qs*) *x* = *0*} ∪ {−(*crb* (*prod-list-var qs*)), (*crb* (*prod-list-var qs*))}
using *UnCI is-not-const mem-Collect-eq* **by** *fastforce*
have *subset2*: {*x*. *poly* (*poly-f-nocrb qs*) *x* = *0*} ∪ {−(*crb* (*prod-list-var qs*)), (*crb* (*prod-list-var qs*))} ⊆
{*x*. *poly* (*poly-f qs*) *x* = *0*}
using *Un-insert-right insert-iff is-not-const mem-Collect-eq of-int-minus poly-f-def poly-f-nocrb-def* **by** *auto*
then show *?thesis* **using** *subset1 subset2* **by** *auto*
qed

lemma *nonzcrb-helper*:
assumes *q-in*: *q* ∈ *set qs*
assumes *qnonz*: *q* ≠ *0*
assumes *lengt*: *length* (*sorted-list-of-set* {(*x*::*real*). (∃ *q* ∈ *set*(*qs*). (*q* ≠ *0* ∧ *poly q x* = *0*))} :: *real list*) > *0*
shows ¬(∃ *x* ≥ (*real-of-int* (*crb* (*prod-list-var qs*))). *poly q x* = *0*)
proof *clarsimp*
fix *x*
assume *xgt*: *real-of-int* (*crb* (*prod-list-var qs*)) ≤ *x*
assume *pqz*: *poly q x* = *0*
let *?zer-list* = *sorted-list-of-set* {(*x*::*real*). (∃ *q* ∈ *set*(*qs*). (*q* ≠ *0* ∧ *poly q x* = *0*))} :: *real list*
have *strict-sorted-h*: *sorted-wrt* (<) *?zer-list* **using** *sorted-sorted-list-of-set strict-sorted-iff* **by** *auto*
have *finset*: *finite* {*x*. ∃ *q*∈*set qs*. *q* ≠ *0* ∧ *poly q x* = *0*}
proof −
have ∀ *q* ∈ *set qs*. *q*≠ *0* ⟶ *finite* {*x*. *poly q x* = *0*}
using *poly-roots-finite* **by** *auto*
then show *?thesis* **by** *auto*
qed
have *all-prop*: ∀ *x* ∈ {*x*. ∃ *q*∈*set qs*. *q* ≠ *0* ∧ *poly q x* = *0*}. *poly* (*prod-list-var qs*) *x* = *0*
using *q-dvd-prod-list-var-prop*
by *fastforce*
then have *poly* (*prod-list-var qs*) (*sorted-list-of-set* {*x*. ∃ *q*∈*set qs*. *q* ≠ *0* ∧ *poly q x* = *0*} ! *0*) = *0*
using *finset set-sorted-list-of-set*
by (*metis* (*no-types*, *lifting*) *lengt nth-mem*)
then have *crbgt*: *crb* (*prod-list-var qs*) > *?zer-list* ! (*length ?zer-list* − *1*) **using** *prod-list-var-nonzero crb-lem-pos*[*of prod-list-var qs ?zer-list* ! (*length ?zer-list* − *1*)]
by (*metis* (*no-types*, *lifting*) *all-prop diff-less finset lengt less-numeral-extra*(*1*) *nth-mem set-sorted-list-of-set*)
have *x-in*: *x* ∈ {*x*. ∃ *q*∈*set qs*. *q* ≠ *0* ∧ *poly q x* = *0*}
using *pqz q-in qnonz*
by *auto*

then have *x* ∈ *set ?zer-list*
  by (*smt* (*verit*, *best*) *finset in-set-member mem-Collect-eq set-sorted-list-of-set*)

then have $x \leq$ *?zer-list* ! (*length ?zer-list* − *1*) **using** *strict-sorted-h*
  by (*meson all-prop x-in crb-lem-pos not-less prod-list-var-nonzero xgt*)
then show *False* **using** *xgt crbgt*
  by *auto*
qed

lemma *root-set-nocrb-var*:
  assumes *is-not-const*: *check-all-const-deg qs* = *False*
  shows ({*x*. *poly* (*poly-f qs*) *x* = *0*}::*real set*)
= {*x*. *poly* (*poly-f-nocrb qs*) *x* = *0*} ∪ (({−(*crb* (*prod-list-var qs*)), (*crb* (*prod-list-var qs*))})::*real set*)
  using *root-set-nocrb* **apply** (*auto*)
    apply (*smt* (*verit*) *of-int-minus poly-f-def poly-f-nocrb-def poly-root-factor*(*2*))
   apply (*simp add*: *is-not-const poly-f-def*)
  using *is-not-const* **apply** *blast*
  by (*metis* (*no-types*, *lifting*) *poly-f-def poly-f-nocrb-def poly-root-factor*(*2*))

lemma *nonzcrb*:
  assumes *q-in*: *q* ∈ *set qs*
  assumes *qnonz*: $q \neq 0$
  shows ¬(∃ *x* ≥ (*real-of-int* (*crb* (*prod-list-var qs*))). *poly q x* = *0*)
proof −
  let *?zer-list* = (*sorted-list-of-set* {(*x*::*real*). (∃ *q* ∈ *set*(*qs*). (*q* ≠ *0* ∧ *poly q x* = *0*))} :: *real list*)
  have *eo*: *length ?zer-list* = *0* ∨ *length ?zer-list* > *0*
    by *auto*
  have *h1*: *length ?zer-list* = *0* ⟶ ¬(∃ *x* ≥ (*real-of-int* (*crb* (*prod-list-var qs*))). *poly q x* = *0*)
   by (*metis crb-lem-pos dvdE linorder-not-less mult-zero-left poly-mult prod-list-var-nonzero q-dvd-prod-list-var-prop q-in qnonz*)
  have *h2*: *length ?zer-list* > *0* ⟶ ¬(∃ *x* ≥ (*real-of-int* (*crb* (*prod-list-var qs*))). *poly q x* = *0*)
    using *nonzcrb-helper assms*
    by *blast*
  show *?thesis* **using** *eo h1 h2*
    by *auto*
qed

definition *sgn-pos-inf-rat-list*:: *real poly list* ⇒ *int list*
  where *sgn-pos-inf-rat-list l* = *map* (λ*x*. (*Sturm-Tarski.sign* (*sgn-pos-inf x*))) *l*

definition *sgn-neg-inf-rat-list*:: *real poly list* ⇒ *int list*
  where *sgn-neg-inf-rat-list l* = *map* (λ*x*. (*Sturm-Tarski.sign* (*sgn-neg-inf x*))) *l*

definition *sgn-neg-inf-rat-list2*:: *real poly list* ⇒ *rat list*
  where *sgn-neg-inf-rat-list2 l* = *map* (λ*x*. ((*rat-of-int* ∘ *Sturm-Tarski.sign*) (*sgn-neg-inf*

x))) *l*

definition *sgn-pos-inf-rat-list2*:: *real poly list* ⇒ *rat list*
where *sgn-pos-inf-rat-list2 l* = *map* (λ*x*. ((*rat-of-int* ∘ *Sturm-Tarski.sign*) (*sgn-pos-inf x*))) *l*

lemma *root-ub-restate*:
fixes *p*:: *real poly*
assumes *pnonz*: *p* ≠ *0*
fixes *z*::*real*
assumes *zgt*: ∀ *x*. *poly p x*=*0* ⟶ *x* < *z*
shows *x*≥*z* ⟹ *sgn* (*poly p x*) = *Sturm-Tarski.sign* (*sgn-pos-inf p*)
proof −
assume *xgt*: *x*≥*z*
have *allx*: ∀ *x* ≥ *z*. *sgn* (*poly p x*) = *sgn* (*poly p z*)
proof *clarsimp*
fix *x*
assume *zleq*: *z* ≤ *x*
then have *xnonz*: *sgn* (*poly p x*) ≠ *0*
using *zgt* **unfolding** *sgn-def* **by** *auto*
have *znonz*: *sgn* (*poly p z*) ≠ *0*
using *zgt* **unfolding** *sgn-def* **by** *auto*
have *Matrix-Equation-Construction.sgn* (*poly p x*) ≠ *Matrix-Equation-Construction.sgn* (*poly p z*) ⟹ *False*
proof −
assume *neq*: *Matrix-Equation-Construction.sgn* (*poly p x*) ≠ *Matrix-Equation-Construction.sgn* (*poly p z*)
then have *z-lt*: *z* < *x* **using** *zleq*
using *less-eq-real-def* **by** *force*
have *h1*: *z* < *x* ⟹
Matrix-Equation-Construction.sgn (*poly p z*) ≠ *0* ⟹
Matrix-Equation-Construction.sgn (*poly p x*) ≠ *0* ⟹
¬ *poly p x* < *0* ⟹ *0* < *poly p x*
using *zgt* **by** *fastforce*
have *h2*: *z* < *x* ⟹
Matrix-Equation-Construction.sgn (*poly p z*) ≠ *0* ⟹
Matrix-Equation-Construction.sgn (*poly p x*) ≠ *0* ⟹
¬ *poly p x* < *0* ⟹ *poly p z* < *0*
by (*metis neq sgn-def*)
have *h3*: *z* < *x* ⟹
Matrix-Equation-Construction.sgn (*poly p z*) ≠ *0* ⟹
Matrix-Equation-Construction.sgn (*poly p x*) ≠ *0* ⟹
¬ *0* < *poly p z* ⟹ *0* < *poly p x*
by (*metis neq sgn-def*)
have *h4*: *z* < *x* ⟹
Matrix-Equation-Construction.sgn (*poly p z*) ≠ *0* ⟹
Matrix-Equation-Construction.sgn (*poly p x*) ≠ *0* ⟹
¬ *0* < *poly p z* ⟹ *poly p z* < *0*
using *h2 h3* **by** *linarith*

```
      have ((poly p x) > 0 ∧ (poly p z) < 0) ∨ ((poly p x) < 0 ∧ (poly p z) > 0)
        using z-lt znonz xnonz h1 h2 h3 h4
        by auto
      then have ∃ w. w > x ∧ w < z ∧ (poly p w = 0)
        using poly-IVT-pos[of z x] poly-IVT-neg[of z x]
        by (metis ‹z < x› not-less-iff-gr-or-eq zgt)
      then show False
        using zgt
        using zleq by linarith
    qed
   then show Matrix-Equation-Construction.sgn (poly p x) = Matrix-Equation-Construction.sgn
(poly p z)
      by auto
  qed
  have (∃ ub.
        (∀ x≥ub. sgn-class.sgn (poly p x) = sgn-pos-inf p))
    using root-ub[of p] pnonz
    by meson
  then obtain ub where ub-prop: (∀ x≥ub. sgn-class.sgn (poly p x) = sgn-pos-inf
p)
    by auto
  let ?ub2 = max ub (z+1)
  have (∀ x≥?ub2. sgn-class.sgn (poly p x) = sgn-pos-inf p) ∧ (?ub2 > z)
    using ub-prop by auto
  then have h1: (Sturm-Tarski.sign(sgn-class.sgn (poly p ?ub2)::real)) = Sturm-Tarski.sign(sgn-pos-inf
p) ∧ (?ub2 > z)
    by auto
  have (Sturm-Tarski.sign(sgn-class.sgn (poly p ?ub2)::real)) = (sgn (poly p ?ub2))
    unfolding sign-def sgn-def by auto
  then have (sgn (poly p ?ub2)) = (Sturm-Tarski.sign(sgn-pos-inf p)) ∧ (?ub2 >
z)
    using h1
    by metis
  then have ∀ x≥z. sgn (poly p x) = Sturm-Tarski.sign (sgn-pos-inf p)
    using allx
    by (metis dual-order.refl max.absorb3 max.bounded-iff)
  then show ?thesis using xgt
    by auto
qed

lemma limit-pos-infinity-helper1:
  assumes q-in: q ∈ set qs
  assumes qnonz: q ≠ 0
  assumes x = (crb (prod-list-var qs))
  shows (if (poly q x > 0) then (1::int) else (if (poly q x = 0) then (0::rat) else
(−1::rat)))
    = ((Sturm-Tarski.sign (sgn-pos-inf q))::int)
  using poly-sgn-eventually-at-top[unfolded eventually-at-top-linorder]
proof −
```

```
  have (∃ ub. (∀ x. poly q x = 0 ⟶ x < ub ⟶
         (∀ x≥ub. sgn-class.sgn (poly q x) = sgn-pos-inf q)))
    using root-ub[of q] qnonz
    by meson
  then obtain ub where ub-prop: (∀ x. poly q x = 0 ⟶ x < ub ⟶
         (∀ x≥ub. sgn-class.sgn (poly q x) = sgn-pos-inf q))
    by auto
  show ?thesis
    proof −
    have q ∈ set qs ∧ q ≠ 0
      using q-in qnonz by blast
    then have f1: ∀ r. ¬ real-of-int (crb (prod-list-var qs)) ≤ r ∨ 0 ≠ poly q r
      by (metis (no-types) nonzcrb[of q])
    have f2: real-of-int (crb (prod-list-var qs)) ≤ real-of-int x
      using assms(3) by force
    then have f3: 0 ≠ poly q (real-of-int x)
      using f1 by blast
     have comb1: ⋀z x. ⟦q ≠ 0; ∀ x. poly q x = 0 ⟶ x < z; z ≤ x⟧ ⟹ of-rat
(Matrix-Equation-Construction.sgn (poly q x)) = complex-of-int (Sturm-Tarski.sign
(sgn-pos-inf q))
       using assms(2) root-ub-restate[of q] by auto
    obtain rr :: real where
     q ≠ 0 ∧ (0 = poly q rr ⟶ rr < real-of-int (crb (prod-list-var qs))) ∧ real-of-int
(crb (prod-list-var qs)) ≤ real-of-int x ⟶ Sturm-Tarski.sign (sgn-pos-inf q) =
Matrix-Equation-Construction.sgn (poly q (real-of-int x))
      by (metis comb1)
   then have f4: Sturm-Tarski.sign (sgn-pos-inf q) = Matrix-Equation-Construction.sgn
(poly q (real-of-int x))
      using f2 f1 linorder-not-less qnonz by blast
    have Matrix-Equation-Construction.sgn (poly q (real-of-int x)) = (if 0 < poly
q (real-of-int x) then 1 else if poly q (real-of-int x) < 0 then − 1 else 0) ∧ (if 0
< poly q (real-of-int x) then (1::rat) = (if 0 < poly q (real-of-int x) then 1 else if
poly q (real-of-int x) < 0 then − 1 else 0) else (if 0 < poly q (real-of-int x) then
1::rat else if poly q (real-of-int x) < 0 then − 1 else 0) = (if poly q (real-of-int x)
< 0 then − 1 else 0)) ∧ (if poly q (real-of-int x) < 0 then − (1::rat) = (if poly
q (real-of-int x) < 0 then − 1 else 0) else (0::rat) = (if poly q (real-of-int x) < 0
then − 1 else 0))
      by (simp add: sgn-def)
    moreover
    { assume − (1::int) ≠ (if poly q (real-of-int x) < 0 then − 1 else 0)
      then have 0 < poly q (real-of-int x)
        using f3 by (metis (no-types) not-less-iff-gr-or-eq) }
    ultimately show ?thesis
      by (smt (verit) Rat.of-rat-1 f4 of-int-hom.hom-one)
  qed
qed

lemma limit-pos-infinity-helper2:
  assumes q-in: q ∈ set qs
```

assumes *qnonz*: *q* = *0*
assumes *x* = (*crb* (*prod-list-var qs*))
shows (*if* (*poly q x* > *0*) *then* (*1*::*rat*) *else* (*if* (*poly q x* = *0*) *then* (*0*::*rat*) *else* (−*1*::*rat*)))
= ((*Sturm-Tarski.sign* (*sgn-pos-inf q*))::*int*)
proof −
have *0* = ((*Sturm-Tarski.sign* (*sgn-pos-inf q*)))
by (*simp add*: *qnonz sgn-pos-inf-def*)
then show *?thesis* **using** *qnonz*
by *auto*
qed

lemma *limit-pos-infinity-helper*:
assumes *q-in*: *q* ∈ *set qs*
assumes *x* = (*crb* (*prod-list-var qs*))
shows (*if* (*poly q x* > *0*) *then* (*1*::*int*) *else* (*if* (*poly q x* = *0*) *then 0 else* −*1*))
= ((*Sturm-Tarski.sign* (*sgn-pos-inf q*)))
using *limit-pos-infinity-helper1 limit-pos-infinity-helper2 assms*(*2*) *q-in*
by (*smt* (*verit*) *Rat.of-rat-1 Sturm-Tarski.sign-cases of-int-eq-iff of-int-hom.hom-one of-int-hom.hom-zero of-rat-0 of-rat-neg-one one-neq-neg-one zero-neq-neg-one*)

lemma *Sturm-Tarski-casting*:
shows ((*Sturm-Tarski.sign x*)) = *rat-of-int* (*Sturm-Tarski.sign x*)
by (*simp add*: *Sturm-Tarski.sign-def*)

lemma *limit-pos-infinity*:
shows *consistent-sign-vec qs* (*crb* (*prod-list-var qs*)) = *sgn-pos-inf-rat-list qs*
unfolding *consistent-sign-vec-def sgn-pos-inf-rat-list-def*
using *limit-pos-infinity-helper Sturm-Tarski-casting*
apply (*auto*)
apply *fastforce*
apply *presburger*
by (*metis of-int-hom.hom-one of-int-minus*)

lemma *nonzcrb-helper-neg*:
assumes *q-in*: *q* ∈ *set qs*
assumes *qnonz*: *q* ≠ *0*
assumes *lengt*: *length* (*sorted-list-of-set* {(*x*::*real*). (∃ *q* ∈ *set*(*qs*). (*q* ≠ *0* ∧ *poly q x* = *0*))} :: *real list*) > *0*
shows ¬(∃ *x* ≤ (*real-of-int* (−*crb* (*prod-list-var qs*))). *poly q x* = *0*)
proof *clarsimp*
fix *x*
assume *xlt*: *x* ≤ − *real-of-int* (*crb* (*prod-list-var qs*))
assume *pqz*: *poly q x* = *0*
let *?zer-list* = *sorted-list-of-set* {(*x*::*real*). (∃ *q* ∈ *set*(*qs*). (*q* ≠ *0* ∧ *poly q x* = *0*))} :: *real list*
have *strict-sorted-h*: *sorted-wrt* (<) *?zer-list* **using** *sorted-sorted-list-of-set strict-sorted-iff* **by** *auto*
have *finset*: *finite* {*x*. ∃ *q*∈*set qs*. *q* ≠ *0* ∧ *poly q x* = *0*}

proof −
have *all-q*: ∀ *q* ∈ *set qs*. *q*≠ *0* ⟶ *finite* {*x*. *poly q x* = *0*}
using *poly-roots-finite* **by** *auto*
then show *?thesis* **by** *auto*
qed
have *all-x*: ∀ *x* ∈ {*x*. ∃ *q*∈*set qs*. *q* ≠ *0* ∧ *poly q x* = *0*}. *poly* (*prod-list-var qs*) *x* = *0*
using *q-dvd-prod-list-var-prop*
by *fastforce*
then have *poly* (*prod-list-var qs*) (*sorted-list-of-set* {*x*. ∃ *q*∈*set qs*. *q* ≠ *0* ∧ *poly q x* = *0*} ! *0*) = *0*
using *finset set-sorted-list-of-set*
by (*metis* (*no-types*, *lifting*) *lengt nth-mem*)
then have *crbgt*: −*crb* (*prod-list-var qs*) < *?zer-list* ! *0* **using** *prod-list-var-nonzero crb-lem-pos*[*of prod-list-var qs ?zer-list* ! (*length ?zer-list* − *1*)]
using *crb-lem-neg* **by** *blast*
have *x-in*: *x* ∈ {*x*. ∃ *q*∈*set qs*. *q* ≠ *0* ∧ *poly q x* = *0*}
using *pqz q-in qnonz*
by *auto*
then have *x* ∈ *set ?zer-list*
by (*smt* (*verit*, *best*) *finset in-set-member mem-Collect-eq set-sorted-list-of-set*)

then have *x* ≥ *?zer-list* ! *0* **using** *strict-sorted-h*
by (*smt* (*verit*) *all-x x-in crb-lem-neg linorder-not-le of-int-hom.hom-uminus prod-list-var-nonzero xlt*)
then show *False* **using** *xlt crbgt*
by *auto*
qed

lemma *nonzcrb-neg*:
assumes *q-in*: *q* ∈ *set qs*
assumes *qnonz*: *q* ≠ *0*
shows ¬(∃ *x* ≤ (*real-of-int* (−*crb* (*prod-list-var qs*))). *poly q x* = *0*)
proof −
let *?zer-list* = (*sorted-list-of-set* {(*x*::*real*). (∃ *q* ∈ *set*(*qs*). (*q* ≠ *0* ∧ *poly q x* = *0*))} :: *real list*)
have *eo*: *length ?zer-list* = *0* ∨ *length ?zer-list* > *0*
by *auto*
have *h1*: *length ?zer-list* = *0* ⟶ ¬(∃ *x* ≤ (*real-of-int* (−*crb* (*prod-list-var qs*))). *poly q x* = *0*)
by (*metis crb-lem-neg dvdE linorder-not-less mult-zero-left poly-mult prod-list-var-nonzero q-dvd-prod-list-var-prop q-in qnonz*)
have *h2*: *length ?zer-list* > *0* ⟶ ¬(∃ *x* ≤ (*real-of-int* (−*crb* (*prod-list-var qs*))). *poly q x* = *0*)
using *nonzcrb-helper-neg assms*
by *blast*
show *?thesis* **using** *eo h1 h2*
by *auto*
qed

```
lemma root-lb-restate:
  fixes p:: real poly
  assumes pnonz: p ≠ 0
  fixes z::real
  assumes zgt: ∀x. poly p x=0 ⟶ x > z
  shows x≤z ⟹ sgn (poly p x) = Sturm-Tarski.sign (sgn-neg-inf p)
proof −
  assume xlt: x ≤ z
  have allx: ∀x ≤ z. sgn (poly p x) = sgn (poly p z)
  proof clarsimp
    fix x
    assume zleq: x ≤ z
    then have xnonz: sgn (poly p x) ≠ 0
      using zgt unfolding sgn-def by auto
    have znonz: sgn (poly p z) ≠ 0
      using zgt unfolding sgn-def by auto
   have Matrix-Equation-Construction.sgn (poly p x) ≠ Matrix-Equation-Construction.sgn
(poly p z) ⟹ False
    proof −
     assume neq: Matrix-Equation-Construction.sgn (poly p x) ≠ Matrix-Equation-Construction.sgn
(poly p z)
      then have x < z using zleq
        using less-eq-real-def by force
      then have ((poly p x) > 0 ∧ (poly p z) < 0) ∨ ((poly p x) < 0 ∧ (poly p z)
> 0)
        using znonz xnonz zgt neq sgn-def
        by metis
      then have ∃w. w < x ∧ w > z ∧ (poly p w = 0)
        using poly-IVT-pos[of x z] poly-IVT-neg[of x z]
        by (metis ‹x < z› not-less-iff-gr-or-eq zgt)
      then show False
        using zgt zleq
        by linarith
    qed
   then show Matrix-Equation-Construction.sgn (poly p x) = Matrix-Equation-Construction.sgn
(poly p z)
      by auto
  qed
  have (∃ub. (∀x≤ub. sgn-class.sgn (poly p x) = sgn-neg-inf p))
    using root-lb[of p] pnonz
    by meson
  then obtain lb where ub-prop: (∀x≤lb. sgn-class.sgn (poly p x) = sgn-neg-inf
p)
    by auto
  let ?ub2 = min lb (z−1)
  have (∀x≤?ub2. sgn-class.sgn (poly p x) = sgn-neg-inf p) ∧ (?ub2 < z)
    using ub-prop by auto
 then have h1: (Sturm-Tarski.sign(sgn-class.sgn (poly p ?ub2)::real)) = Sturm-Tarski.sign(sgn-neg-inf
```

p) ∧ (*?ub2* < *z*)
  by *auto*
 have (*Sturm-Tarski.sign*(*sgn-class.sgn* (*poly p ?ub2*)::*real*)) = (*sgn* (*poly p ?ub2*))
  unfolding *sign-def sgn-def* **by** *auto*
 then have (*sgn* (*poly p ?ub2*)) = (*Sturm-Tarski.sign*(*sgn-neg-inf p*)) ∧ (*?ub2* < *z*)
  using *h1* **by** *metis*
 then show *?thesis* **using** *allx xlt*
  by (*metis min.absorb3 min.cobounded2*)
qed

lemma *limit-neg-infinity-helper1*:
 assumes *q-in*: *q* ∈ *set qs*
 assumes *qnonz*: *q* ≠ *0*
 assumes *x* = −(*crb* (*prod-list-var qs*))
 shows (*if* (*poly q x* > *0*) *then 1 else* (*if* (*poly q x* = *0*) *then 0 else* −*1*))
  = ((*Sturm-Tarski.sign* (*sgn-neg-inf q*)))
proof −
 have (∃ *ub*. (∀ *x*. *poly q x* = *0* ⟶ *x* < *ub* ⟶
    (∀ *x*≥*ub*. *sgn-class.sgn* (*poly q x*) = *sgn-neg-inf q*)))
  using *root-lb*[*of q*] *qnonz*
  by (*meson not-less-iff-gr-or-eq*)
 then obtain *ub* **where** *ub-prop*: (∀ *x*. *poly q x* = *0* ⟶ *x* < *ub* ⟶
    (∀ *x*≥*ub*. *sgn-class.sgn* (*poly q x*) = *sgn-neg-inf q*))
  by *auto*
 show *?thesis*
 proof −
  have *q* ∈ *set qs* ∧ *q* ≠ *0*
   using ‹*q* ∈ *set qs*› ‹*q* ≠ *0*› **by** *blast*
  then have *f1*: ∀ *r*. ¬ *r* ≤ *real-of-int* (− *crb* (*prod-list-var qs*)) ∨ *0* ≠ *poly q r*
   using *nonzcrb-neg*[*of q*] **by** *auto*
  have *f2*: *real-of-int x* ≤ *real-of-int* (− *crb* (*prod-list-var qs*))
   using ‹*x* = − *crb* (*prod-list-var qs*)› **by** *fastforce*
  obtain *rr* :: *real* **where**
   f3: *q* ≠ *0* ∧ (*0* = *poly q rr* ⟶ *real-of-int* (− *crb* (*prod-list-var qs*)) < *rr*) ∧ *real-of-int x* ≤ *real-of-int* (− *crb* (*prod-list-var qs*)) ⟶ *Sturm-Tarski.sign* (*sgn-neg-inf q*) = *Matrix-Equation-Construction.sgn* (*poly q* (*real-of-int x*))
   by (*metis assms*(*2*) *root-lb-restate*[*of q*])
  have ¬ *rr* ≤ *real-of-int* (− *crb* (*prod-list-var qs*)) ∨ *0* ≠ *poly q rr*
   using *f1* **by** *blast*
  then have *Sturm-Tarski.sign* (*sgn-neg-inf q*) = *Matrix-Equation-Construction.sgn* (*poly q* (*real-of-int x*))
   using *f3 f2* ‹*q* ≠ *0*› **by** *linarith*
  then show *?thesis*
   by (*smt* (*verit*, *best*) *Sturm-Tarski.sign-def of-int-eq-iff of-int-hom.hom-one of-int-hom.hom-zero of-rat-hom.hom-0-iff of-rat-hom.hom-1-iff of-rat-neg-one sgn-def*)
 qed

qed

lemma *limit-neg-infinity-helper2*:
  assumes *q-in*: *q* ∈ *set qs*
  assumes *qnonz*: *q* = *0*
  assumes *x* = (−*crb* (*prod-list-var qs*))
  shows (*if* (*poly q x* > *0*) *then 1 else* (*if* (*poly q x* = *0*) *then 0 else* −*1*))
    = *Sturm-Tarski.sign* (*sgn-neg-inf q*)
proof −
  have *0* = *Sturm-Tarski.sign* (*sgn-neg-inf q*)
    by (*simp add*: *qnonz sgn-neg-inf-def*)
  then show *?thesis* **using** *qnonz*
    by *auto*
qed

lemma *limit-neg-infinity-helper-var*:
  assumes *q-in*: *q* ∈ *set qs*
  assumes *x* = (−*crb* (*prod-list-var qs*))
  shows (*if* (*poly q x* > *0*) *then 1 else* (*if* (*poly q x* = *0*) *then 0 else* −*1*))
    = *Sturm-Tarski.sign* (*sgn-neg-inf q*)
  using *limit-neg-infinity-helper1 limit-neg-infinity-helper2 assms*(*2*) *q-in*
  by *blast*

lemma *limit-neg-infinity-helper*:
  assumes *q-in*: *q* ∈ *set qs*
  assumes *x* = (−*crb* (*prod-list-var qs*))
  shows (*if* (*poly q x* > *0*) *then 1 else* (*if* (*poly q x* = *0*) *then 0 else* −*1*))
    = (*Sturm-Tarski.sign* (*sgn-neg-inf q*))
  using *limit-neg-infinity-helper-var Sturm-Tarski-casting*[*of* (*sgn-neg-inf q*)]
  using *assms*(*2*) *q-in* **by** *presburger*

lemma *limit-neg-infinity*:
  shows *consistent-sign-vec qs* (−(*crb* (*prod-list-var qs*))) = *sgn-neg-inf-rat-list qs*
  using *limit-neg-infinity-helper Sturm-Tarski-casting*
    consistent-sign-vec-def sgn-neg-inf-rat-list-def
  apply (*auto*) **apply** *fastforce* **apply** *presburger*
  by (*metis* (*mono-tags*, *opaque-lifting*) *of-int-eq-1-iff of-int-minus*)

lemma *csv-signs-at-same*:
  shows *consistent-sign-vec qs x* = *signs-at qs x*
  unfolding *consistent-sign-vec-def signs-at-def squash-def* **by** *auto*

lemma *complex-real-int-casting*:
  fixes *z*:: *int*
  shows (*complex-of-real* ∘ *real-of-int*) *z* = *complex-of-int z*
  by *auto*

lemma *poly-f-ncrb-constant-connection*:
  assumes *is-const*: *check-all-const-deg qs* = *True*
  shows *set* (*characterize-consistent-signs-at-roots* (*poly-f qs*) *qs*)

= *set* (*characterize-consistent-signs-at-roots* (*poly-f-nocrb qs*) *qs*) ∪ {*sgn-neg-inf-rat-list qs*, *sgn-pos-inf-rat-list qs*}
proof −
have *h1*: (*poly-f qs*) = [:*0*, *1*:]
using *assms* **unfolding** *poly-f-def* **by** *auto*
have *h2*: (*poly-f-nocrb qs*) = [:*0*, *1*:]
using *assms* **unfolding** *poly-f-nocrb-def* **by** *auto*
then have *same-set1*: *set* (*characterize-consistent-signs-at-roots* (*poly-f qs*) *qs*)
= *set* (*characterize-consistent-signs-at-roots* (*poly-f-nocrb qs*) *qs*)
using *h1 h2*
by *auto*
have {*x*. *poly* (*poly-f qs*) *x* = *0*} = {*0*}
using *h1* **by** *auto*
then have (*characterize-root-list-p* (*poly-f qs*)) = [*0*]
using *h1* **unfolding** *characterize-root-list-p-def* **by** *auto*
then have (*characterize-consistent-signs-at-roots* (*poly-f qs*) *qs*)
= (*remdups* (*map* (*signs-at qs*) [*0*]))
unfolding *characterize-consistent-signs-at-roots-def* **by** *auto*
then have (*characterize-consistent-signs-at-roots* (*poly-f qs*) *qs*) = [*signs-at qs 0*]
by *auto*
then have *same-set2*: *set* (*characterize-consistent-signs-at-roots* (*poly-f-nocrb qs*) *qs*) = {*signs-at qs 0*}
using *same-set1*
by *auto*
have *same1*: *sgn-neg-inf-rat-list qs* = *sgn-pos-inf-rat-list qs*
using *is-const* **unfolding** *sgn-neg-inf-rat-list-def sgn-pos-inf-rat-list-def*
unfolding *sgn-neg-inf-def sgn-pos-inf-def*
using *check-all-const-deg-prop* **by** *auto*
have $\bigwedge q.\ q \in set\ qs \Longrightarrow poly\ q\ 0 = lead\text{-}coeff\ q$
using *is-const check-all-const-deg-prop*
by (*metis poly-0-coeff-0*)
then have *same2-h*: *map* (λ*x*. *rat-of-int* (*Sturm-Tarski.sign* (*sgn-class.sgn* (*lead-coeff x*)))) *qs* =
map ((λ*x*. *if 0* < *x then 1 else if x* < *0 then* − *1 else 0*) ∘ (λ*q*. *poly q 0*)) *qs*
by *simp*
then have *map rat-of-int* (*sgn-pos-inf-rat-list qs*) = (*signs-at qs 0*)
using *is-const check-all-const-deg-prop*
unfolding *sgn-pos-inf-rat-list-def sgn-pos-inf-def signs-at-def squash-def*
by *auto*
then have *map real-of-rat* (*map rat-of-int* (*sgn-pos-inf-rat-list qs*)) =
map of-rat (*signs-at qs 0*)
by *auto*
then have *map real-of-int* (*sgn-pos-inf-rat-list qs*) = *map of-rat* (*signs-at qs 0*)
by *auto*
then have *map complex-of-real* (*map real-of-int* (*sgn-pos-inf-rat-list qs*)) = *map of-rat* (*signs-at qs 0*)
by *auto*
then have *same2*: (*sgn-pos-inf-rat-list qs*) = (*signs-at qs 0*)
using *complex-real-int-casting*

by (*metis list.map-comp map-eq-conv*)
show *?thesis*
using *same-set1 same-set2 same1 same2*
by *auto*
qed

lemma *poly-f-ncrb-nonconstant-connection*:
assumes *is-not-const*: *check-all-const-deg qs* = *False*
shows *set* (*characterize-consistent-signs-at-roots* (*poly-f qs*) *qs*)
= *set* (*characterize-consistent-signs-at-roots* (*poly-f-nocrb qs*) *qs*) ∪ {*sgn-neg-inf-rat-list qs*, *sgn-pos-inf-rat-list qs*}
proof −
let *?s1* = {*x*. *poly* (*poly-f qs*) *x* = *0*}
let *?s2* = ({*x*. *poly* (*poly-f-nocrb qs*) *x* = *0*} ∪ ({−(*crb* (*prod-list-var qs*)), (*crb* (*prod-list-var qs*))}::*real set*))
have *same-set*: *?s1* = *?s2*
using *root-set-nocrb-var*[*of qs*] *is-not-const*
by *auto*
then have *same-map*: (λ*x*. (*signs-at qs x*)) ‘ *?s1* = (λ*x*. (*signs-at qs x*)) ‘ *?s2*
by *presburger*
have *set* (*characterize-consistent-signs-at-roots* (*poly-f qs*) *qs*) =
set (*map* (*signs-at qs*) (*characterize-root-list-p* (*poly-f qs*)))
using *characterize-consistent-signs-at-roots-def* **by** *auto*
then have *set* (*characterize-consistent-signs-at-roots* (*poly-f qs*) *qs*) =
(λ*x*. (*signs-at qs x*)) ‘ {*x*. *poly* (*poly-f qs*) *x* = *0*}
by (*simp add*: *characterize-root-list-p-def poly-f-nonzero poly-roots-finite*)
then have *set* (*characterize-consistent-signs-at-roots* (*poly-f qs*) *qs*) =
(λ*x*. (*signs-at qs x*)) ‘ ({*x*. *poly* (*poly-f-nocrb qs*) *x* = *0*} ∪ ({−(*crb* (*prod-list-var qs*)), (*crb* (*prod-list-var qs*))}::*real set*))
using *same-map* **by** *auto*
then have *bigeq*: *set* (*characterize-consistent-signs-at-roots* (*poly-f qs*) *qs*) =
(λ*x*. (*signs-at qs x*)) ‘ {*x*. *poly* (*poly-f-nocrb qs*) *x* = *0*} ∪ (λ*x*. (*signs-at qs x*)) ‘ ({−(*crb* (*prod-list-var qs*)), (*crb* (*prod-list-var qs*))}::*real set*)
by (*metis image-Un*)
have *crb-seteq*: (λ*x*. (*signs-at qs x*)) ‘ ({−(*crb* (*prod-list-var qs*)), (*crb* (*prod-list-var qs*))}::*real set*) =
(λ*x*. (*signs-at qs x*)) ‘ ({−(*crb* (*prod-list-var qs*))}::*real set*) ∪
(λ*x*. (*signs-at qs x*)) ‘ ({(*crb* (*prod-list-var qs*))}::*real set*)
by *blast*
have *neg*: (λ*x*. (*signs-at qs x*)) ‘ ({−(*crb* (*prod-list-var qs*))}::*real set*) ={*sgn-neg-inf-rat-list qs*}
using *limit-neg-infinity*[*of qs*] *csv-signs-at-same* **by** *auto*
have *pos*: (λ*x*. (*signs-at qs x*)) ‘ ({(*crb* (*prod-list-var qs*))}::*real set*) ={*sgn-pos-inf-rat-list qs*}
using *limit-pos-infinity*[*of qs*] *csv-signs-at-same* **by** *auto*
have (λ*x*. (*signs-at qs x*)) ‘ ({−(*crb* (*prod-list-var qs*)), (*crb* (*prod-list-var qs*))}::*real set*) = {*sgn-neg-inf-rat-list qs*} ∪ {*sgn-pos-inf-rat-list qs*}
using *crb-seteq neg pos* **by** *auto*
then have (λ*x*. (*signs-at qs x*)) ‘ ({−(*crb* (*prod-list-var qs*)), (*crb* (*prod-list-var*

qs))}::*real set*)
  = {*sgn-neg-inf-rat-list qs*, *sgn-pos-inf-rat-list qs*}
  by *auto*
 then have *biggereq*: *set* (*characterize-consistent-signs-at-roots* (*poly-f qs*) *qs*) =
  signs-at qs ‘ {*x*. *poly* (*poly-f-nocrb qs*) *x* = *0*} ∪
  {*sgn-neg-inf-rat-list qs*, *sgn-pos-inf-rat-list qs*}
  using *bigeq* **by** *auto*
 have *key*: *signs-at qs* ‘ {*x*. *poly* (*poly-f-nocrb qs*) *x* = *0*} = *set* (*characterize-consistent-signs-at-roots* (*poly-f-nocrb qs*) *qs*)
  using *characterize-consistent-signs-at-roots-def*
  Groups.mult-ac(*2*) *characterize-root-list-p-def is-not-const list.set-map mult-cancel-left mult-cancel-left1 poly-f-def poly-f-nocrb-def poly-f-nonzero poly-roots-finite set-remdups sorted-list-of-set*(*1*)
 proof −
  have *poly-f-nocrb qs* ≠ *0*
   by (*metis mult-cancel-left1 poly-f-def poly-f-nocrb-def poly-f-nonzero*)
  then have *set* (*remdups* (*map* (*signs-at qs*) (*sorted-list-of-set* {*r*. *poly* (*poly-f-nocrb qs*) *r* = *0*}))) = *signs-at qs* ‘ {*r*. *poly* (*poly-f-nocrb qs*) *r* = *0*}
   by (*simp add*: *poly-roots-finite*)
  then show *?thesis*
   using *characterize-consistent-signs-at-roots-def characterize-root-list-p-def* **by** *presburger*
 qed
 then show *?thesis* **using** *biggereq*
  by (*metis image-image list.set-map*)
qed

lemma *poly-f-ncrb-connection*:
 shows *set* (*characterize-consistent-signs-at-roots* (*poly-f qs*) *qs*)
  = *set* (*characterize-consistent-signs-at-roots* (*poly-f-nocrb qs*) *qs*) ∪ {*sgn-neg-inf-rat-list qs*, *sgn-pos-inf-rat-list qs*}
 using *poly-f-ncrb-nonconstant-connection poly-f-ncrb-constant-connection*
 by *blast*

end

theory *Hybrid-Multiv-Matrix-Proofs*
 imports
  BenOr-Kozen-Reif.Matrix-Equation-Construction
  Multiv-Tarski-Query
  BenOr-Kozen-Reif.Renegar-Proofs
  Hybrid-Multiv-Matrix
  Hybrid-Multiv-Algorithm
  Renegar-Modified

begin

hide-const *BKR-Decision.And*
hide-const *BKR-Decision.Or*

hide-const *UnivPoly.eval*

17.1 Connect multivariate Tarski queries to univariate

lemma *pull-out-pairs-length*:
shows *length* (*pull-out-pairs qs Is*) = *length Is*
using *pull-out-pairs.simps* **by** *force*

lemma *construct-NofI-M-subset-prop*:
assumes (*assumps*, *tq*) ∈ *set* (*construct-NofI-M p init-assumps qs1 qs2*)
shows *set init-assumps* ⊆ *set assumps*
proof −
have (*assumps*, *tq*) ∈ *set* (*map construct-NofI-single-M* (*construct-NofI-R-spmods p init-assumps qs1 qs2*))
using *assms*
by *auto*
then obtain *mid-assumps tq-list* **where** *tuple-prop2*: (*assumps*, *tq*) = *construct-NofI-single-M* (*mid-assumps*, *tq-list*)
(*mid-assumps*, *tq-list*) ∈ *set* (*construct-NofI-R-spmods p init-assumps qs1 qs2*)
by *force*
have *s1*: *mid-assumps* = *assumps*
using *tuple-prop2*(*1*)
by *simp*
have *s2*: *set init-assumps* ⊆ *set mid-assumps*
using *tuple-prop2*(*2*) *spmods-multiv-assum-acc*
by (*metis construct-NofI-R-spmods-def*)
then show *set init-assumps* ⊆ *set assumps*
using *s1 s2*
by *blast*
qed

17.2 Connect multivariate RHS vector to univariate

lemma *construct-rhs-vector-rec-M-subset-prop-len1*:
assumes (*assumps*, *rhs-list*) ∈ *set* (*construct-rhs-vector-rec-M p init-assumps* [*a*])
shows *set init-assumps* ⊆ *set assumps*
proof −
obtain *qs1 qs2* **where** *a-prop*: *a* = (*qs1*, *qs2*)
using *prod.exhaust* **by** *blast*
have *tuple-prop*: (*assumps*, *rhs-list*) ∈ *set* (*map* (λ(*new-assumps*, *tq*). (*new-assumps*, [*tq*])) (*construct-NofI-M p init-assumps qs1 qs2*))
using *a-prop assms* **by** *auto*
then obtain *tq* **where** *tq-prop*: *rhs-list* = [*tq*]
by *auto*
let *?ell* = (*map* (λ(*new-assumps*, *tq*). (*new-assumps*, [*tq*])) (*construct-NofI-M p init-assumps qs1 qs2*))
have *tuple-in-list*: (*assumps*, *rhs-list*) ∈ *set ?ell*

using *tuple-prop*
by *auto*
then have (*assumps*, *tq*) ∈ *set* (*construct-NofI-M p init-assumps qs1 qs2*)
using *tq-prop*
by (*smt* (*verit*, *best*) *imageE list.inject list.set-map old.prod.case old.prod.simps*(*1*)
prod.collapse)
then have (*assumps*, *tq*) ∈ *set* (*construct-NofI-M p init-assumps qs1 qs2*)
using *tq-prop*
by *metis*
then have (*assumps*, *tq*) ∈ *set*(*map construct-NofI-single-M*
(*construct-NofI-R-spmods p init-assumps qs1 qs2*))
by *force*
then have (*assumps*, *tq*) ∈ *set* (*construct-NofI-M p init-assumps qs1 qs2*)
using ‹(*assumps*, *tq*) ∈ *set* (*construct-NofI-M p init-assumps qs1 qs2*)› **by** *force*
then show *?thesis* **using** *construct-NofI-M-subset-prop*
by *blast*
qed

lemma *construct-rhs-vector-rec-M-subset-prop*:
assumes (*assumps*, *rhs-list*) ∈ *set* (*construct-rhs-vector-rec-M p init-assumps*
qs-list)
shows *set init-assumps* ⊆ *set assumps*
using *assms*
proof (*induct qs-list arbitrary*: *assumps rhs-list init-assumps*)
case *Nil*
then show *?case*
using *construct-rhs-vector-rec-M.simps* **by** *auto*
next
case (*Cons a qs-list*)
obtain *qs1 qs2* **where** *a-prop*: *a* = (*qs1*, *qs2*)
using *Cons.prems Cons.hyps prod.exhaust*
by *fastforce*
{ **assume** ∗: *qs-list* = []
then have *set init-assumps* ⊆ *set assumps* **using** *construct-rhs-vector-rec-M-subset-prop-len1*

Cons.prems
by *blast*
}
moreover { **assume** ∗: *length qs-list* ≥ *1*
then obtain *v va* **where** *qs-list-prop*: *qs-list* = *v* # *va*
by (*metis One-nat-def Suc-le-length-iff*)
let *?TQ-list* = *construct-NofI-M p init-assumps qs1 qs2*
have (*assumps*, *rhs-list*) ∈ *set* (*construct-rhs-vector-rec-M p init-assumps* ((*qs1*,
qs2)#*qs-list*))
using *Cons.prems*(*1*) ∗ *a-prop* **by** *auto*
then have (*assumps*, *rhs-list*) ∈ *set* (*concat* ((*map* (λ(*new-assumps*, *tq*).
(*let rec* = *construct-rhs-vector-rec-M p new-assumps qs-list in*
map (λ*r*. (*fst r*, *tq*#*snd r*)) *rec*)) *?TQ-list*)))
using ∗ *a-prop qs-list-prop*

by (*simp add*: *split-def*)
then obtain *new-assumps tq* **where** *tq-prop*: (*new-assumps*,*tq*) ∈ *set* (*?TQ-list*)
(*assumps*, *rhs-list*) ∈ *set* (*let rec* = *construct-rhs-vector-rec-M p new-assumps qs-list in*
map (λ*r*. (*fst r*, *tq*#*snd r*)) *rec*)
by *auto*
then obtain *rhs-rest* **where** *rhs-list-prop*: *rhs-list* = *tq*#*rhs-rest*
(*assumps*, *rhs-rest*) ∈ *set* (*construct-rhs-vector-rec-M p new-assumps qs-list*)
by *auto*
then have *s1*: *set new-assumps* ⊆ *set assumps*
using *Cons.hyps*
by *auto*
have *s2*: *set init-assumps* ⊆ *set new-assumps*
using *construct-NofI-M-subset-prop tq-prop*(*1*)
by *auto*
have *set init-assumps* ⊆ *set assumps*
using *s1 s2*
by *auto*
}
ultimately show *?case*
using *Cons.prems*
by (*metis length-0-conv less-one linorder-neqE-nat nat-less-le rel-simps*(*47*))
qed

lemma *construct-rhs-vector-rec-M-univariate*:

assumes *rhs-list-is*: (*assumps*, *rhs-list*) ∈ *set*(*construct-rhs-vector-rec-M p init-assumps qs-list*)
assumes *val*: ⋀*p n*. (*p*,*n*) ∈ *set assumps* ⟹ *satisfies-evaluation val p n*
shows *rhs-list* = *map* (λ(*qs1*,*qs2*).
(*construct-NofI-R* (*eval-mpoly-poly val p*) (*eval-mpoly-poly-list val qs1*) (*eval-mpoly-poly-list val qs2*))) *qs-list*
using *assms*
proof (*induct qs-list arbitrary*: *assumps rhs-list init-assumps*)
case *Nil*
then show *?case*
using *construct-rhs-vector-rec-M.simps*
by *auto*
next
case (*Cons a qs-list*)
obtain *qs1 qs2* **where** *a-prop*: *a* = (*qs1*, *qs2*)
using *Cons.prems Cons.hyps*
using *prod.exhaust* **by** *blast*
{ assume ∗: *qs-list* = []
let *?tq* = *construct-NofI-R* (*eval-mpoly-poly val p*) (*eval-mpoly-poly-list val qs1*) (*eval-mpoly-poly-list val qs2*)
have (*assumps*, *rhs-list*) ∈ *set* (*construct-rhs-vector-rec-M p init-assumps* [(*qs1*, *qs2*)])
using *Cons.prems*(*1*) ∗ *a-prop* **by** *auto*

then have
(*assumps*, *rhs-list*) ∈ *set* (*let TQ-list* = *construct-NofI-M p init-assumps qs1 qs2 in*
map (λ(*new-assumps*, *tq*). (*new-assumps*, [*tq*])) *TQ-list*)
by (*metis construct-rhs-vector-rec-M.simps*(*2*))
then have *tuple-prop*:
(*assumps*, *rhs-list*) ∈ *set* (*map* (λ(*new-assumps*, *tq*). (*new-assumps*, [*tq*])) (*construct-NofI-M p init-assumps qs1 qs2*))
by *auto*
then obtain *tq* **where** *tq-prop*: *rhs-list* = [*tq*]
by *auto*
let *?ell* = (*map* (λ(*new-assumps*, *tq*). (*new-assumps*, [*tq*])) (*construct-NofI-M p init-assumps qs1 qs2*))
have *tuple-in-list*: (*assumps*, *rhs-list*) ∈ *set ?ell*
using *tuple-prop*
by *auto*
then have (*assumps*, *tq*) ∈ *set* (*construct-NofI-M p init-assumps qs1 qs2*)
using *tq-prop*
by (*smt* (*verit*, *best*) *imageE list.inject list.set-map old.prod.case old.prod.simps*(*1*) *prod.collapse*)
then have (*assumps*, *tq*) ∈ *set* (*construct-NofI-M p init-assumps qs1 qs2*)
using *tq-prop*
by *metis*
then have *rhs-list* = [*construct-NofI-R* (*eval-mpoly-poly val p*) (*eval-mpoly-poly-list val qs1*) (*eval-mpoly-poly-list val qs2*)]
using *construct-NofI-M-univariate-tarski-query*[*of assumps tq p init-assumps qs1 qs2 val*]
Cons.prems(*2*) *tq-prop*
by *auto*
then have *rhs-list* =
map (λ(*qs1*, *qs2*).
construct-NofI-R (*eval-mpoly-poly val p*) (*eval-mpoly-poly-list val qs1*)
(*eval-mpoly-poly-list val qs2*))
[(*qs1*, *qs2*)]
by *auto*
then have *rhs-list* =
map (λ(*qs1*, *qs2*).
construct-NofI-R (*eval-mpoly-poly val p*) (*eval-mpoly-poly-list val qs1*)
(*eval-mpoly-poly-list val qs2*))
(*a* # *qs-list*)
using ∗ *a-prop* **by** *auto*
}
moreover {
assume ∗: *qs-list* ≠ []
then obtain *v va* **where** *qs-list-prop*: *qs-list* = *v* # *va*
by (*meson neq-Nil-conv*)
let *?tq* = *construct-NofI-R* (*eval-mpoly-poly val p*) (*eval-mpoly-poly-list val qs1*) (*eval-mpoly-poly-list val qs2*)
let *?TQ-list* = *construct-NofI-M p init-assumps qs1 qs2*

have (*assumps*, *rhs-list*) ∈ *set* (*construct-rhs-vector-rec-M p init-assumps* ((*qs1*, *qs2*)#*qs-list*))
  using *Cons.prems*(*1*) ∗ *a-prop* **by** *auto*
 then have (*assumps*, *rhs-list*) ∈ *set* (*concat* ((*map* (λ(*new-assumps*, *tq*).
 (*let rec* = *construct-rhs-vector-rec-M p new-assumps qs-list in*
  map (λ*r*. (*fst r*, *tq*#*snd r*)) *rec*)) *?TQ-list*)))
  using ∗ *a-prop qs-list-prop*
  by (*simp add*: *split-def*)
 then obtain *new-assumps tq* **where** *tq-prop*: (*new-assumps*,*tq*) ∈ *set* (*?TQ-list*)
  (*assumps*, *rhs-list*) ∈ *set* (*let rec* = *construct-rhs-vector-rec-M p new-assumps qs-list in*
  map (λ*r*. (*fst r*, *tq*#*snd r*)) *rec*)
  by *auto*
 then obtain *rhs-rest* **where** *rhs-list-prop*: *rhs-list* = *tq*#*rhs-rest*
  (*assumps*, *rhs-rest*) ∈ *set* (*construct-rhs-vector-rec-M p new-assumps qs-list*)
  by *auto*
 then have *subset*: *set new-assumps* ⊆ *set assumps* **using**
   construct-rhs-vector-rec-M-subset-prop[*of assumps rhs-rest p new-assumps qs-list*]
  by *auto*
 then have *val-2*: ⋀ *p n*. (*p*, *n*) ∈ *set new-assumps* ⟹ *satisfies-evaluation val p n*
  using *val*
  by (*meson Set.basic-monos*(*7*) *local.Cons*(*3*))
 have *tq-is*: *tq* = *?tq*
 using *construct-NofI-M-univariate-tarski-query*[*of new-assumps tq p init-assumps qs1 qs2 val*]
  tq-prop(*1*) *Cons.prems*(*2*) *subset*
  by *blast*
 have *ih*: *rhs-rest* =
   map (λ(*qs1*, *qs2*).
    construct-NofI-R (*eval-mpoly-poly val p*) (*eval-mpoly-poly-list val qs1*)
    (*eval-mpoly-poly-list val qs2*))
   qs-list
  using *rhs-list-prop Cons.prems*(*2*) *val-2*
  by (*simp add*: *local.Cons*(*1*))
 then have *rhs-list* =
  map (λ(*qs1*, *qs2*).
   construct-NofI-R (*eval-mpoly-poly val p*) (*eval-mpoly-poly-list val qs1*)
   (*eval-mpoly-poly-list val qs2*))
  (*a* # *qs-list*)
  using *a-prop tq-is ih rhs-list-prop*
  by *simp*
}
ultimately have *rhs-list* =
 map (λ(*qs1*, *qs2*).
  construct-NofI-R (*eval-mpoly-poly val p*) (*eval-mpoly-poly-list val qs1*)
  (*eval-mpoly-poly-list val qs2*))

(*a* # *qs-list*)
by *blast*
then show *?case*
by *blast*
qed

lemma *retrieve-polys-prop*:
assumes $\bigwedge x.\ x \in set\ ns \Longrightarrow x < length\ qs$
shows (*eval-mpoly-poly-list val* (*retrieve-polys qs ns*)) = (*retrieve-polys* (*map* (*eval-mpoly-poly val*) *qs*) *ns*)
using *assms* **unfolding** *eval-mpoly-poly-list-def retrieve-polys-def* **by** *auto*

lemma *construct-rhs-vector-M-univariate*:

assumes *rhs-vec-is*: (*assumps*, *rhs-vec*) ∈ *set*(*construct-rhs-vector-M p init-assumps qs Is*)
assumes $\bigwedge p\ n.\ (p,n) \in set\ assumps \Longrightarrow satisfies\text{-}evaluation\ val\ p\ n$
assumes *well-def-subsets*: $\bigwedge$ *Is1 Is2 n*. (*Is1*, *Is2*) ∈ *set Is* ⟹
(*n* ∈ *set Is1* ∨ *n* ∈ *set Is2*) ⟹ *n* < *length qs*
shows *rhs-vec* = *construct-rhs-vector-R* (*eval-mpoly-poly val p*) (*map* (*eval-mpoly-poly val*) *qs*) *Is*
proof −
have (*assumps*, *rhs-vec*) ∈ *set* (*map* (λ*res*. (*fst res*, *vec-of-list* (*snd res*))) (*construct-rhs-vector-rec-M p init-assumps* (*pull-out-pairs qs Is*)))
using *rhs-vec-is* **unfolding** *construct-rhs-vector-M-def* **by** *auto*
then have ∃ *rhs-list*. *rhs-vec* = *vec-of-list rhs-list* ∧ (*assumps*, *rhs-list*) ∈ *set* (*map* (λ*res*. (*fst res*, *snd res*)) (*construct-rhs-vector-rec-M p init-assumps* (*pull-out-pairs qs Is*)))
by *auto*
then obtain *rhs-list* **where** *rhs-list-prop*: *rhs-vec* = *vec-of-list rhs-list* ∧ (*assumps*, *rhs-list*) ∈ *set* (*map* (λ*res*. (*fst res*, *snd res*)) (*construct-rhs-vector-rec-M p init-assumps* (*pull-out-pairs qs Is*)))
by *auto*
then have *rhs-list-char*: *rhs-list* = *map* (λ(*qs1*,*qs2*).
(*construct-NofI-R* (*eval-mpoly-poly val p*) (*eval-mpoly-poly-list val qs1*) (*eval-mpoly-poly-list val qs2*))) (*pull-out-pairs qs Is*)
using *assms construct-rhs-vector-rec-M-univariate*

by (*smt* (*verit*, *del-insts*) *case-prod-beta map-eq-conv map-idI prod.exhaust-sel*)
have *lov-1*: *list-of-vec* (*map-vec*
(λ(*I1*, *I2*).
construct-NofI-R (*eval-mpoly-poly val p*) (*retrieve-polys* (*map* (*eval-mpoly-poly val*) *qs*) *I1*)
(*retrieve-polys* (*map* (*eval-mpoly-poly val*) *qs*) *I2*))
(*vec-of-list Is*)) = *map* (λ(*I1*, *I2*).
construct-NofI-R (*eval-mpoly-poly val p*) (*retrieve-polys* (*map* (*eval-mpoly-poly val*) *qs*) *I1*)
(*retrieve-polys* (*map* (*eval-mpoly-poly val*) *qs*) *I2*)) *Is*

by (*metis list-vec vec-of-list-map*)
have *lov-2*: *list-of-vec rhs-vec* = *rhs-list*
using *rhs-list-prop*
using *list-vec* **by** *blast*
let *?rhs-list-var* = *map* (λ(*I1*, *I2*).
construct-NofI-R (*eval-mpoly-poly val p*) (*retrieve-polys* (*map* (*eval-mpoly-poly val*) *qs*) *I1*)
(*retrieve-polys* (*map* (*eval-mpoly-poly val*) *qs*) *I2*)) *Is*
have *rhs-list-is*: *rhs-list* = *?rhs-list-var*
proof −
have *len-is1*: *length* (*pull-out-pairs qs Is*) = *length Is*
by *simp*
then have *length rhs-list* = *length Is*
using *rhs-list-char*
by *auto*
have *len-is2*: *length ?rhs-list-var* = *length Is*
by *auto*
have ⋀*n*. *n* < *length Is* ⟹ *rhs-list* ! *n* = *?rhs-list-var* ! *n*
proof −
fix *n*
assume ∗: *n* < *length Is*
then obtain *Is1 Is2* **where** *Is-prop*: *Is* ! *n* = (*Is1*, *Is2*)
by *fastforce*
then have (*pull-out-pairs qs Is*) ! *n* = ((*retrieve-polys qs Is1*), (*retrieve-polys qs Is2*))
using ∗ **by** *force*
then have *nth-1*: *rhs-list* ! *n* = *construct-NofI-R* (*eval-mpoly-poly val p*) (*eval-mpoly-poly-list val* (*retrieve-polys qs Is1*)) (*eval-mpoly-poly-list val* (*retrieve-polys qs Is2*))
using *rhs-list-char*
by (*simp add*: ∗ *len-is1*)
have *nth-2*: *map* (λ(*I1*, *I2*).
construct-NofI-R (*eval-mpoly-poly val p*) (*retrieve-polys* (*map* (*eval-mpoly-poly val*) *qs*) *I1*)
(*retrieve-polys* (*map* (*eval-mpoly-poly val*) *qs*) *I2*)) *Is* ! *n*
= *construct-NofI-R* (*eval-mpoly-poly val p*) (*retrieve-polys* (*map* (*eval-mpoly-poly val*) *qs*) *Is1*)
(*retrieve-polys* (*map* (*eval-mpoly-poly val*) *qs*) *Is2*)
using *Is-prop*
by (*simp add*: ∗)
have *ret-poly1*: (*eval-mpoly-poly-list val* (*retrieve-polys qs Is1*)) = (*retrieve-polys* (*map* (*eval-mpoly-poly val*) *qs*) *Is1*)
unfolding *retrieve-polys-def eval-mpoly-poly-list-def*
using *well-def-subsets retrieve-polys-prop Is-prop*
by (*metis* ∗ *eval-mpoly-poly-list-def in-set-conv-nth retrieve-polys-def*)
have *ret-poly2*: (*eval-mpoly-poly-list val* (*retrieve-polys qs Is2*)) = (*retrieve-polys* (*map* (*eval-mpoly-poly val*) *qs*) *Is2*)
unfolding *retrieve-polys-def eval-mpoly-poly-list-def*
using *well-def-subsets retrieve-polys-prop Is-prop*

by (*metis* ∗ *eval-mpoly-poly-list-def in-set-conv-nth retrieve-polys-def*)
have *construct-NofI-R* (*eval-mpoly-poly val p*) (*eval-mpoly-poly-list val* (*retrieve-polys qs Is1*)) (*eval-mpoly-poly-list val* (*retrieve-polys qs Is2*))
= *construct-NofI-R* (*eval-mpoly-poly val p*) (*retrieve-polys* (*map* (*eval-mpoly-poly val*) *qs*) *Is1*)
(*retrieve-polys* (*map* (*eval-mpoly-poly val*) *qs*) *Is2*)
using *ret-poly1 ret-poly2*
by *auto*
then show *rhs-list* ! *n* = *?rhs-list-var* ! *n*
using *nth-1 nth-2* **by** *auto*
qed
then show *?thesis*
using *len-is1 len-is2*
by (*metis* ‹*length rhs-list* = *length Is*› *nth-equalityI*)
qed
then show *?thesis*
using *rhs-list-is lov-2 lov-1*
unfolding *construct-rhs-vector-R-def*
using *rhs-list-prop* **by** *force*
qed

17.3 Connect multivariate LHS vector to univariate

lemma *solve-for-lhs-vector-M-univariate*:
assumes *lhs-in*: (*assumps*, *lhs-vec*) ∈ *set* (*solve-for-lhs-M p init-assumps qs subsets matr*)
assumes *val*: ⋀*p n*. (*p*,*n*) ∈ *set assumps* ⟹ *satisfies-evaluation val p n*
assumes *well-def-subsets*: ⋀ *Is1 Is2 n*. (*Is1*, *Is2*) ∈ *set subsets* ⟹
(*n* ∈ *set Is1* ∨ *n* ∈ *set Is2*) ⟹ *n* < *length qs*
shows *lhs-vec* = *solve-for-lhs-R* (*eval-mpoly-poly val p*) (*map* (*eval-mpoly-poly val*) *qs*) *subsets matr*
proof −
let *?lhs-univ* = *solve-for-lhs-R* (*eval-mpoly-poly val p*) (*map* (*eval-mpoly-poly val*) *qs*) *subsets matr*
have (*assumps*, *lhs-vec*) ∈ *set*(*map* (*λrhs*. (*fst rhs*, *solve-for-lhs-single-M p qs subsets matr* (*snd rhs*))) (*construct-rhs-vector-M p init-assumps qs subsets*))
using *lhs-in*
using *solve-for-lhs-M-def* **by** *auto*
then obtain *rhs* **where** *rhs-prop*: *rhs* ∈ *set*(*construct-rhs-vector-M p init-assumps qs subsets*)
(*assumps*, *lhs-vec*) = (*fst rhs*, *solve-for-lhs-single-M p qs subsets matr* (*snd rhs*))
by *auto*
then have *snd-is*: *snd rhs* = *construct-rhs-vector-R* (*eval-mpoly-poly val p*) (*map* (*eval-mpoly-poly val*) *qs*) *subsets*
using *construct-rhs-vector-M-univariate*
by (*metis assms*(*2*) *fst-conv prod.collapse well-def-subsets*)
have *fst rhs* = *assumps* **using** *rhs-prop*
by *force*
have *?lhs-univ* = *mult-mat-vec* (*matr-option* (*dim-row matr*) (*mat-inverse-var*

matr)) (*snd rhs*)
using *snd-is*
by (*simp add*: *solve-for-lhs-R-def*)
then show *?thesis*
using *rhs-prop*(*2*) **unfolding** *solve-for-lhs-single-M-def*
by *auto*
qed

17.4 Connect multivariate reduction step to univariate

lemma *reduce-system-single-M-univariate*:
assumes *inset*: (*assumps*, *mat-eq*) ∈ *set*(*reduce-system-single-M p qs* (*init-assumps*, *init-mat-eq*))
assumes *val*: ⋀*p n*. (*p*,*n*) ∈ *set assumps* ⟹ *satisfies-evaluation val p n*
assumes *init*: *init-mat-eq* = (*m*, (*subs*, *signs*))
assumes *well-def-subsets*: ⋀ *Is1 Is2 n*. (*Is1*, *Is2*) ∈ *set subs* ⟹
(*n* ∈ *set Is1* ∨ *n* ∈ *set Is2*) ⟹ *n* < *length qs*
shows *mat-eq* = *reduce-system-R* (*eval-mpoly-poly val p*) ((*map* (*eval-mpoly-poly val*) *qs*), *init-mat-eq*)
proof −
have (*assumps*, *mat-eq*) ∈ *set* (*map* (λ*lhs*. (*fst lhs*, *reduction-step-R m signs subs* (*snd lhs*))) (*solve-for-lhs-M p init-assumps qs subs m*))
using *inset*
using *assms*(*3*)
by *force*
then obtain *lhs* **where** *lhs-prop*: *lhs* ∈ *set* (*solve-for-lhs-M p init-assumps qs subs m*)
(*assumps*, *mat-eq*) = (*fst lhs*, *reduction-step-R m signs subs* (*snd lhs*))
by *auto*
then have *snd lhs* = *solve-for-lhs-R* (*eval-mpoly-poly val p*) (*map* (*eval-mpoly-poly val*) *qs*) *subs m*
using *solve-for-lhs-vector-M-univariate assms*
by (*smt* (*verit*, *best*) *prod.exhaust-sel prod.simps*(*1*))
then have *mat-eq* = *reduction-step-R m signs subs* (*solve-for-lhs-R* (*eval-mpoly-poly val p*) (*map* (*eval-mpoly-poly val*) *qs*) *subs m*)
using *lhs-prop*
by *force*
then show *?thesis*
using *init*
using *reduce-system-R.simps* **by** *presburger*
qed

lemma *reduce-system-M-univariate*:
assumes (*assumps*, *mat-eq*) ∈ *set*(*reduce-system-M p qs input-list*)
assumes *val*: ⋀*p n*. (*p*,*n*) ∈ *set assumps* ⟹ *satisfies-evaluation val p n*
assumes *val-qs*: *val-qs* = (*map* (*eval-mpoly-poly val*) *qs*)
assumes *all-subsets-well-def*: ⋀ *init-assumps init-mat-eq Is1 Is2 n subs m signs*.

(*init-assumps*, (*m*, (*subs*, *signs*))) ∈ *set input-list* ⟹

(*Is1*, *Is2*) ∈ *set subs* ⟹ (*n* ∈ *set Is1* ∨ *n* ∈ *set Is2*) ⟹ *n* < *length qs*
obtains *acc mss* **where**
(*acc*,*mss*) ∈ *set* (*input-list*)
mat-eq = *reduce-system-R* (*eval-mpoly-poly val p*) (*val-qs*,*mss*)
proof −
have (*assumps*, *mat-eq*) ∈ *set*(*concat* (*map* (*reduce-system-single-M p qs*) *input-list*))
by (*metis assms*(*1*) *reduce-system-M.simps*)
then obtain *init-assumps init-m init-subs init-signs* **where**
mat-eq-prop:
(*init-assumps*, (*init-m*, (*init-subs*, *init-signs*))) ∈ *set input-list*
(*assumps*, *mat-eq*) ∈ *set* (*reduce-system-single-M p qs* (*init-assumps*, (*init-m*, (*init-subs*, *init-signs*))))
by *auto*
then have *well-def-subsets*: ⋀ *Is1 Is2 n*. (*Is1*, *Is2*) ∈ *set init-subs* ⟹
(*n* ∈ *set Is1* ∨ *n* ∈ *set Is2*) ⟹ *n* < *length qs*
using *all-subsets-well-def*
by *blast*
then have *mat-eq-is*: *mat-eq* = *reduce-system-R* (*eval-mpoly-poly val p*) ((*map* (*eval-mpoly-poly val*) *qs*), (*init-m*, (*init-subs*, *init-signs*)))
using *reduce-system-single-M-univariate mat-eq-prop assms*(*2*) **by** *blast*
then show *?thesis* **using** *mat-eq-prop*
using *assms*(*3*) *that* **by** *blast*
qed

lemma *base-case-info-M-well-def*:
assumes (*init-assumps*, (*m*, (*subs*, *signs*))) ∈ *set base-case-info-M*
assumes (*Is1*, *Is2*) ∈ *set subs*
assumes *n* ∈ *set Is1* ∨ *n* ∈ *set Is2*
shows *n* < *1*
proof −
have (*m*, (*subs*, *signs*)) = *base-case-info-R* **using** *assms*(*1*)
unfolding *base-case-info-M-def* **using** *Renegar-Algorithm.base-case-info-R-def Renegar-Algorithm.base-case-info-R-def*
by *simp*
then have *s*: *subs* = [([], []),([*0*], []),([], [*0*])] **unfolding** *base-case-info-R-def*
by *auto*
have (*n* ∈ *set Is1* ∨ *n* ∈ *set Is2*) **using** *assms*(*3*)
by (*simp add*: *in-set-member*)
thus *?thesis* **using** *assms*(*2*) **unfolding** *s* **by** *auto*
qed

17.5 Connect multivariate combining systems to univariate

lemma *base-case-with-assumps-info-M-well-def*:
assumes (*init-assumps*, (*m*, (*subs*, *signs*))) ∈ *set* (*base-case-info-M-assumps a*)
assumes (*Is1*, *Is2*) ∈ *set subs*
assumes *n* ∈ *set Is1* ∨ *n* ∈ *set Is2*
shows *n* < *1*

proof −
  have (*m*, (*subs*, *signs*)) = *base-case-info-R* **using** *assms*(*1*)
    unfolding *base-case-info-M-assumps-def*
   using *Renegar-Algorithm.base-case-info-R-def Renegar-Algorithm.base-case-info-R-def*
    by *auto*
  then have *s*: *subs* = [([], []),([*0*], []),([], [*0*])] **unfolding** *base-case-info-R-def*
    by *auto*
  have (*n* ∈ *set Is1* ∨ *n* ∈ *set Is2*) **using** *assms*(*3*)
    by (*simp add*: *in-set-member*)
  thus *?thesis* **using** *assms*(*2*) **unfolding** *s* **by** *auto*
qed

lemma *concat-map-in-set*:
  assumes *x* ∈ *set* (*concat* (*map f ls*))
  shows ∃ *i* < *length ls*. *x* ∈ *set* (*f* (*ls* ! *i*))
  using *assms*
 by (*smt* (*verit*, *best*) *in-set-conv-nth length-map map-nth-eq-conv nth-concat-split*)

lemma *combine-systems-R-snd*:
  assumes *length qs1* = *length new-qs1*
  shows *snd* (*combine-systems-R p* (*qs1*, *sys1*) (*qs2*, *sys2*)) =
    snd (*combine-systems-R new-p* (*new-qs1*, *sys1*) (*new-qs2*, *sys2*))
proof −
  obtain *m1 sub1 sgn1* **where** *sys1*: *sys1* = (*m1*, *sub1*, *sgn1*)
    using *prod-cases3* **by** *blast*
  obtain *m2 sub2 sgn2* **where** *sys2*: *sys2* = (*m2*, *sub2*, *sgn2*)
    using *prod-cases3* **by** *blast*
  have *h1*: *snd* (*combine-systems-R p* (*qs1*, *sys1*) (*qs2*, *sys2*)) =
    snd (*smash-systems-R p qs1 qs2 sub1 sub2 sgn1 sgn2 m1 m2*)
    using *sys1 sys2* **by** *auto*
  have *h2*: *snd* (*combine-systems-R new-p* (*new-qs1*, *sys1*) (*new-qs2*, *sys2*)) =
    snd (*smash-systems-R new-p new-qs1 new-qs2 sub1 sub2 sgn1 sgn2 m1 m2*)
    using *sys1 sys2* **by** *auto*
  show *?thesis*
    using *h1 h2 assms* **unfolding** *smash-systems-R-def* **by** *auto*
qed

17.6 Subset Properties

lemma *construct-rhs-vector-M-subset-prop*:
 assumes (*assumps*, *rhs-vec*) ∈ *set* (*construct-rhs-vector-M p init-assumps qs subsets*)
  shows *set init-assumps* ⊆ *set assumps*
proof −
  obtain *rhs-list* **where** (*assumps*, *rhs-list*) ∈ *set* (*construct-rhs-vector-rec-M p init-assumps* (*pull-out-pairs qs subsets*))
    rhs-vec = *vec-of-list rhs-list*
    using *assms* **unfolding** *construct-rhs-vector-M-def* **by** *auto*

then show *?thesis*
using *construct-rhs-vector-rec-M-subset-prop* **by** *auto*
qed

lemma *construct-lhs-vector-rec-M-subset-prop*:
assumes (*assumps*, *lhs-list*) ∈ *set* (*solve-for-lhs-M p init-assumps qs subsets matr*)
shows *set init-assumps* ⊆ *set assumps*
proof −
obtain *rhs-vec* **where** (*assumps*, *rhs-vec*) ∈ *set* (*construct-rhs-vector-M p init-assumps qs subsets*)
lhs-list = *matr-option* (*dim-row matr*) (*mat-inverse-var matr*) $*_v$ *rhs-vec*
using *assms* **unfolding** *solve-for-lhs-M-def solve-for-lhs-single-M-def*
by *auto*
then show *?thesis*
using *construct-rhs-vector-M-subset-prop*[*of assumps*] **by** *auto*
qed

lemma *reduce-system-single-M-subset-prop*:
assumes (*assumps*, *mat-eq*) ∈ *set* (*reduce-system-single-M p qs* (*init-assumps*, (*m*,*subs*,*signs*)))
shows *set init-assumps* ⊆ *set assumps*
proof −
obtain *lhs-vec* **where** (*assumps*, *lhs-vec*) ∈ *set* (*solve-for-lhs-M p init-assumps qs subs m*)
mat-eq = *reduction-step-R m signs subs lhs-vec*
using *assms*
by (*auto*)
then show *?thesis*
using *construct-lhs-vector-rec-M-subset-prop*[*of assumps*]
by *auto*
qed

lemma *calculate-data-assumps-M-subset*:
assumes (*assumps*, *mat-eq*) ∈ *set* (*calculate-data-assumps-M p qs init-assumps*)
shows *set init-assumps* ⊆ *set assumps*
using *assms*
proof (*induction length qs arbitrary*: *qs assumps mat-eq rule*: *less-induct*)
case *less*
{**assume** ∗: *length qs* = *0*
then have (*assumps*, *mat-eq*) ∈ *set* (*map* (λ(*assumps*,(*a*,(*b*,*c*))). (*assumps*, (*a*,*b*,*map* (*drop 1*) *c*))) (*reduce-system-M p* [*1*] (*base-case-info-M-assumps init-assumps*)))
using *less.prems* **by** *auto*
then obtain *a b c* **where** (*assumps*, (*a*, (*b*, *c*))) ∈ *set* (*reduce-system-M p* [*1*] (*base-case-info-M-assumps init-assumps*))
by *auto*
then have (*assumps*, (*a*, (*b*, *c*))) ∈ *set* (*concat* (*map* (*reduce-system-single-M p* [*1*]) [(*init-assumps*, *base-case-info-R*)]))
unfolding *base-case-info-M-assumps-def*

using *Renegar-Algorithm.base-case-info-R-def Renegar-Algorithm.base-case-info-R-def*

by (*auto*)
then have (*assumps*, (*a*, (*b*, *c*))) ∈ *set*(*reduce-system-single-M p* [*1*] (*init-assumps*, *base-case-info-R*))
by *auto*
then have *set init-assumps* ⊆ *set assumps*
unfolding *base-case-info-R-def*
using *reduce-system-single-M-subset-prop*[*of assumps* (*a*, (*b*, *c*)) *p* [*1*] *init-assumps*]
by *auto*
}
moreover **{assume** ∗: *length qs* = *1*
then have (*assumps*, *mat-eq*) ∈ *set* (*reduce-system-M p qs* (*base-case-info-M-assumps init-assumps*))
using *less.prems* **by** *auto*
then obtain *a b c* **where** (*assumps*, (*a*, (*b*, *c*))) ∈ *set* (*reduce-system-M p qs* (*base-case-info-M-assumps init-assumps*))
by (*smt* (*verit*) *prod.sel*(*2*) *prod-cases4*)
then have (*assumps*, (*a*, (*b*, *c*))) ∈ *set* (*concat* (*map* (*reduce-system-single-M p qs*) [(*init-assumps*, *base-case-info-R*)]))
unfolding *base-case-info-M-assumps-def*
using *Renegar-Algorithm.base-case-info-R-def Renegar-Algorithm.base-case-info-R-def*

by (*auto*)
then have (*assumps*, (*a*, (*b*, *c*))) ∈ *set*(*reduce-system-single-M p qs* (*init-assumps*, *base-case-info-R*))
by *auto*
then have *set init-assumps* ⊆ *set assumps*
unfolding *base-case-info-R-def*
using *reduce-system-single-M-subset-prop*[*of assumps* (*a*, (*b*, *c*)) *p qs init-assumps*]
by *auto*
}
moreover **{assume** ∗: *length qs* > *1*
let *?len* = *length qs*
let *?q1* = *take* (*?len div 2*) *qs*
let *?left* = *calculate-data-assumps-M p ?q1 init-assumps*
let *?q2* = *drop* (*?len div 2*) *qs*
let *?right* = *calculate-data-assumps-M p ?q2 init-assumps*
let *?comb* = *combine-systems-M p ?q1 ?left ?q2 ?right*
have *len-q1-less*: *length ?q1* < *length qs*
using ∗ **by** *auto*
have *inset-red*: (*assumps*, *mat-eq*) ∈ *set*(*reduce-system-M p* (*fst ?comb*) (*snd ?comb*))
using ∗ *less.prems*
by (*smt* (*verit*, *best*) *calculate-data-assumps-M.simps less-one nat-less-le not-one-less-zero*)
have *fst ?comb* = *qs*
by *auto*
then have (*assumps*, *mat-eq*) ∈ *set*(*reduce-system-M p qs* (*snd ?comb*))

using *inset-red*
by *auto*
then obtain *assm-pre m-pre subs-pre signs-pre* **where** *assumps-reduce*:
(*assm-pre*, (*m-pre*, *subs-pre*, *signs-pre*)) ∈ *set* (*snd ?comb*)
(*assumps*, *mat-eq*) ∈ *set*(*reduce-system-single-M p qs* (*assm-pre*, (*m-pre*, *subs-pre*, *signs-pre*)))
by (*metis concat-map-in-set find-consistent-signs-at-roots-single-M.cases nth-mem reduce-system-M.simps*)
then obtain *meq1 meq2 assm1 assm2* **where** *subsystems*:
(*assm1*, *meq1*) ∈ *set* (*calculate-data-assumps-M p ?q1 init-assumps*)
(*assm2*, *meq2*) ∈ *set* (*calculate-data-assumps-M p ?q2 init-assumps*)
(*assm-pre*, (*m-pre*, *subs-pre*, *signs-pre*)) = *combine-systems-single-M p ?q1* (*assm1*, *meq1*) *?q2* (*assm2*, *meq2*)
by *auto*
then have *assm-pre*: *assm-pre* = *assm1*@*assm2*
by *auto*
have *set init-assumps* ⊆ *set assm1*
using *less.hyps*[*of ?q1*] *less.prems subsystems*(*1*) *len-q1-less*
by *auto*
then have *set init-assumps* ⊆ *set assm-pre*
using *assm-pre* **by** *auto*
then have *set init-assumps* ⊆ *set assumps*
using *assumps-reduce*(*2*) *reduce-system-single-M-subset-prop*[*of assumps mat-eq p qs assm-pre m-pre subs-pre signs-pre*]
by *auto*
}
ultimately show *?case*
by (*meson less-one linorder-neqE-nat*)
qed

lemma *extract-signs-M-subset*:
assumes (*assumps*, *signs*) ∈ *set* (*extract-signs* (*calculate-data-assumps-M p qs init-assumps*))
shows *set init-assumps* ⊆ *set assumps*
proof −
obtain *mat-eq* **where**
(*assumps*, *mat-eq*) ∈ *set* (*calculate-data-assumps-M p qs init-assumps*)
signs = *snd* (*snd mat-eq*)
using *assms* **by** *auto*
then show *?thesis*
using *calculate-data-assumps-M-subset*[*of assumps mat-eq p qs init-assumps*]
by *auto*
qed

17.7 Top-level Results: Connect calculate data methods to univariate

lemma *all-list-constr-R-matches-well-def*:
assumes *welldef*: *all-list-constr-R subs* (*length q*)

shows (*Is1*, *Is2*) ∈ *set* (*subs*) ⟹ *n* ∈ *set Is1* ∨ *n* ∈ *set Is2* ⟹ *n* < *length q*
proof –
assume *inset*: (*Is1*, *Is2*) ∈ *set* (*subs*)
assume *inlist*: *n* ∈ *set Is1* ∨ *n* ∈ *set Is2*
have *welldef-var*: ∀ *x*. *x* ∈ *set subs* ⟶
list-constr (*fst x*) (*length q*) ∧ *list-constr* (*snd x*) (*length q*)
using *welldef* **unfolding** *all-list-constr-R-def*
by (*simp add*: *in-set-member*)
have (*Is1*, *Is2*) ∈ *set subs*
using *inset* **by** *auto*
then have (∀ *x*∈*set Is1*. *x* < *length q*) ∧ (∀ *x*∈*set Is2*. *x* < *length q*)
using *welldef-var*
by (*simp add*: *Ball-set list-constr-def*)
then show *n* < *length q*
using *inlist*
by *metis*
qed

lemma *calculate-data-M-univariate*:
assumes *mat-eq*: (*assumps*, *mat-eq*) ∈ *set* (*calculate-data-M p qs*)
assumes ⋀*p n*. (*p*,*n*) ∈ *set assumps* ⟹ *satisfies-evaluation val p n*
assumes *p-nonzero*: *eval-mpoly-poly val p* ≠ *0*
shows *calculate-data-R* (*eval-mpoly-poly val p*) (*map* (*eval-mpoly-poly val*) *qs*) = *mat-eq*
using *assms*
proof (*induct length qs arbitrary*: *val p mat-eq assumps qs rule*: *less-induct*)
case (*less qs val p mat-eq assumps*)
have *length qs* = *0* ∨ *length qs* = *1* ∨ *length qs* > *1*
by (*meson less-one nat-neq-iff*)
moreover {**assume** ∗: *length qs* = *0*
let *?m* = (*mat-of-rows-list 3* [[*1*,*1*,*1*], [*0*,*1*,*0*], [*1*,*0*,−*1*]])
let *?subs* = [([], []),([*0*], []),([], [*0*])]
let *?signs* = [[*1*],[*0*],[−*1*]]
let *?eval-p* = *eval-mpoly-poly val p*
have *mat-eq-in*: (*assumps*, *mat-eq*) ∈ *set* (*calculate-data-M p* [])
using ∗ *less.prems*(*1*) **by** *auto*
let *?map-base* = *map* (λ(*assumps*,(*a*,(*b*,*c*))). (*assumps*, (*a*,*b*,*map* (*drop 1*) *c*))) (*reduce-system-M p* [*1*] *base-case-info-M*)
have (*assumps*, *mat-eq*) ∈ *set ?map-base*
using *mat-eq-in*
by *auto*
then obtain *a1 b1 c1* **where** *a1b1c1-prop*:
(*assumps*, (*a1*, (*b1*, *c1*))) ∈ *set* (*reduce-system-M p* [*1*] *base-case-info-M*)
mat-eq = (*a1*, (*b1*, *map* (*drop 1*) *c1*))
by *auto*
have *base-case-well-def*: ⋀*init-assumps init-mat-eq Is1 Is2 n subs m signs*.
(*init-assumps*, *m*, *subs*, *signs*) ∈ *set base-case-info-M* ⟹
(*Is1*, *Is2*) ∈ *set subs* ⟹ *n* ∈ *set Is1* ∨ *n* ∈ *set Is2* ⟹ *n* < *length* [*1*]
using *base-case-info-M-well-def* **by** *auto*

have *map-is*: [*1*] = *map* (*eval-mpoly-poly val*) [*1*]
unfolding *eval-mpoly-poly-def eval-mpoly-def*
by *auto*
then have ∃ *acc mss*. (*acc*,*mss*) ∈ *set* (*base-case-info-M*) ∧ (*a1*, (*b1*, *c1*)) = *reduce-system-R* (*eval-mpoly-poly val p*) ([*1*],*mss*)
using *reduce-system-M-univariate*[*of assumps* (*a1*, *b1*, *c1*) *p* [*1*] *base-case-info-M val* [*1*]]
a1b1c1-prop less(*3*) *base-case-well-def*
apply (*auto*)
by *metis*
then obtain *acc mss* **where** *a1b1c1-connect*:
(*acc*,*mss*) ∈ *set* (*base-case-info-M*)
(*a1*, (*b1*, *c1*)) = *reduce-system-R* (*eval-mpoly-poly val p*) ([*1*],*mss*)
by *auto*
then have *mss-is*: *mss* = *base-case-info-R*
unfolding *base-case-info-M-def*
by *auto*
obtain *a b c* **where** *abc-prop*: (*a*, *b*, *c*) = *reduction-step-R ?m ?signs ?subs* (*solve-for-lhs-R ?eval-p* [*1*] *?subs ?m*)
by (*metis reduction-step-R.simps*)
then have (*a*, *b*, *c*) = (*a1*, *b1*, *c1*)
using *abc-prop a1b1c1-prop*
by (*metis a1b1c1-connect*(*2*) *base-case-info-R-def mss-is reduce-system-R.simps*)

then have *mat-eq-is*: (*a*, *b*, *map* (*drop* (*Suc 0*)) *c*) = *mat-eq*
using *a1b1c1-prop*(*2*) **by** (*auto*)
have *qs* = [] ⟹ *mat-eq* = (*a*, *b*, *map* (*drop* (*Suc 0*)) *c*) ⟹
(*case reduce-system-R* (*eval-mpoly-poly val p*) ([*1*], *base-case-info-R*) *of*
(*a*, *b*, *c*) ⇒ (*a*, *b*, *map* (*drop* (*Suc 0*)) *c*)) = (*a*, *b*, *map* (*drop* (*Suc 0*)) *c*)
using *abc-prop* **unfolding** *base-case-info-R-def*
using *old.prod.case reduce-system-R.simps*
by (*smt* (*verit*, *ccfv-SIG*))
then have *calculate-data-R ?eval-p* (*map* (*eval-mpoly-poly val*) *qs*)= *mat-eq*
using ∗ *mat-eq-is*
by *simp*
}
moreover **{assume** ∗: *length qs* = *1*
have *meq*: (*assumps*, *mat-eq*) ∈ *set* (*reduce-system-M p qs base-case-info-M*)
using ∗ *le-eq-less-or-eq less*(*2*) *one-neq-zero* **by** *auto*
have ∗∗: ⋀*init-assumps init-mat-eq Is1 Is2 n subs m signs*.
(*init-assumps*, *m*, *subs*, *signs*) ∈ *set base-case-info-M* ⟹
(*Is1*, *Is2*) ∈ *set subs* ⟹
n ∈ *set Is1* ∨ *n* ∈ *set Is2* ⟹
n < *length qs* **unfolding** ∗
using *base-case-info-M-well-def*
by *meson*
from *reduce-system-M-univariate*[*OF meq less*(*3*), *of map* (*eval-mpoly-poly val*) *qs*]
obtain *acc mss* **where** *ams*: (*acc*,*mss*) ∈ *set base-case-info-M*

mat-eq = *reduce-system-R* (*eval-mpoly-poly val p*)
(*map* (*eval-mpoly-poly val*) *qs*, *mss*)
using ∗∗ **by** *blast*
then have *mss* = *base-case-info-R* **unfolding** *base-case-info-M-def*
by *auto*
then have *calculate-data-R* (*eval-mpoly-poly val p*) (*map* (*eval-mpoly-poly val*)
qs) = *mat-eq*
using *ams* ∗ **by** *simp*
}

moreover **{assume** ∗: *length qs* > *1*
have *inset*: (*assumps*, *mat-eq*) ∈ *set* (*calculate-data-M p qs*)
using *less.prems*(*1*) **by** *auto*
let *?len* = *length qs*
let *?q1* = *take* (*?len div 2*) *qs*
let *?q2* = *drop* (*?len div 2*) *qs*
let *?left* = *calculate-data-M p ?q1*
let *?right* = *calculate-data-M p ?q2*
let *?comb* = *combine-systems-M p ?q1 ?left ?q2 ?right*
let *?eval-p* = (*eval-mpoly-poly val p*)
let *?eval-q1* = (*map* (*eval-mpoly-poly val*) *?q1*)
let *?eval-q2* = (*map* (*eval-mpoly-poly val*) *?q2*)
have *map-q1*: *map* (*eval-mpoly-poly val*) *?q1* =
(*take* (*length* (*map* (*eval-mpoly-poly val*) *qs*) *div 2*) (*map* (*eval-mpoly-poly val*)
qs))
by (*auto simp add*: *take-map*)
have *map-q2*: *map* (*eval-mpoly-poly val*) *?q2* =
(*drop* (*length* (*map* (*eval-mpoly-poly val*) *qs*) *div 2*) (*map* (*eval-mpoly-poly val*)
qs))
by (*auto simp add*: *drop-map*)
have *fst ?comb* = *qs*
by *auto*
then have (*assumps*, *mat-eq*) ∈ *set* (*reduce-system-M p qs* (*snd ?comb*))
using *inset* ∗
by (*smt* (*verit*) *calculate-data-M.simps less-numeral-extra*(*2*) *less-one nat-less-le*)
then have (*assumps*, *mat-eq*) ∈ *set* (*concat* (*map* (*reduce-system-single-M p*
qs) (*snd ?comb*)))
by (*metis reduce-system-M.simps*)
then have ∃ *sys*. *sys* ∈ *set* (*snd ?comb*) ∧ (*assumps*, *mat-eq*) ∈ *set*(*reduce-system-single-M*
p qs sys)
using *concat-map-in-set in-set-member*
by (*metis nth-mem*)
then obtain *a-pre me-pre* **where** *reduce-prop*: (*a-pre*, *me-pre*) ∈ *set* (*snd*
?comb)
(*assumps*, *mat-eq*) ∈ *set*(*reduce-system-single-M p qs* (*a-pre*, *me-pre*))
by *fastforce*
then obtain *a1 me1 a2 me2* **where** *mes-prop*: (*a1*, *me1*) ∈ *set ?left*
(*a2*, *me2*) ∈ *set ?right*
(*a-pre*, *me-pre*) = *combine-systems-single-M p ?q1* (*a1*, *me1*) *?q2* (*a2*, *me2*)

by *auto*
then have *a-pre*: *a-pre* = *a1*@*a2*
by *auto*
have *lengt*: *length qs div 2* ≥ *1*
using ∗ **by** *auto*
then have *len-q1*: *length ?q1* < *length qs*
by *auto*
have *len-q2*: *length ?q2* < *length qs*
using *lengt* **by** *auto*
obtain *mat-pre subs-pre signs-pre* **where** *me-decomp*:
me-pre = (*mat-pre*,*subs-pre*,*signs-pre*)
using *mes-prop*
using *prod-cases3* **by** *blast*
then have *assumps* ∈ *set* (*map fst* (*solve-for-lhs-M p a-pre qs subs-pre mat-pre*))
using *reduce-prop*(*2*) **by** *auto*
then have *assumps* ∈ *set* (*map fst* (*construct-rhs-vector-M p a-pre qs subs-pre*))
unfolding *solve-for-lhs-M-def* **by** *auto*
then obtain *a-rhs-list* **where** (*assumps*, *a-rhs-list*)
∈ *set* (*construct-rhs-vector-rec-M p a-pre* (*pull-out-pairs qs subs-pre*))
unfolding *construct-rhs-vector-M-def* **by** *auto*
then have *a-pre-subset*: *set a-pre* ⊆ *set assumps*
using *construct-rhs-vector-rec-M-subset-prop*[*of assumps - p a-pre* (*pull-out-pairs qs subs-pre*)]
by *auto*
have *set-a1*: *set a1* ⊆ *set assumps*
using *a-pre-subset a-pre* **by** *auto*
then have *a1-satisfies*: (⋀*p n*. (*p*, *n*) ∈ *set a1* ⟹ *satisfies-evaluation val p n*)
using *less*(*3*) **by** *blast*
from *less.hyps*[*of ?q1 a1 me1*, *OF len-q1 mes-prop*(*1*) *a1-satisfies*]
have *me1-ind*: *calculate-data-R* (*eval-mpoly-poly val p*) (*map* (*eval-mpoly-poly val*) *?q1*) = *me1*
using *less*(*4*) **by** *blast*
have *set-a2*: *set a2* ⊆ *set assumps*
using *a-pre-subset a-pre* **by** *auto*
then have *a2-satisfies*: (⋀*p n*. (*p*, *n*) ∈ *set a2* ⟹ *satisfies-evaluation val p n*)
using *less*(*3*) **by** *blast*
from *less.hyps*[*of ?q2 a2 me2*, *OF len-q2 mes-prop*(*2*) *a2-satisfies*]
have *me2-ind*: *calculate-data-R ?eval-p* (*map* (*eval-mpoly-poly val*) *?q2*) = *me2*
using *less*(*4*) **by** *blast*
have *a-pre* = *a1* @ *a2* ⟹ *me-pre* =
snd (*combine-systems-R p* (*take* (*length qs div 2*) *qs*, *me1*)
(*drop* (*length qs div 2*) *qs*, *me2*)) ⟹
snd (*combine-systems-R p* (*take* (*length qs div 2*) *qs*, *me1*)
(*drop* (*length qs div 2*) *qs*, *me2*)) =
snd (*combine-systems-R* (*eval-mpoly-poly val p*)
(*map* (*eval-mpoly-poly val*) (*take* (*length qs div 2*) *qs*), *me1*)
(*map* (*eval-mpoly-poly val*) (*drop* (*length qs div 2*) *qs*), *me2*))
using *combine-systems-R-snd*
by (*metis length-map*)

then have *me-pre*: *me-pre* = *snd* (*combine-systems-R* (*eval-mpoly-poly val p*) ((*map* (*eval-mpoly-poly val*) *?q1*), *me1*) ((*map* (*eval-mpoly-poly val*) *?q2*), *me2*))
using *mes-prop*(*3*)
by (*auto*)
obtain *mat-pre1 subs-pre1 signs-pre1* **where** *me1-decomp*:
me1 = (*mat-pre1*,*subs-pre1*,*signs-pre1*)
using *prod-cases3* **by** *blast*
obtain *mat-pre2 subs-pre2 signs-pre2* **where** *me2-decomp*:
me2 = (*mat-pre2*,*subs-pre2*,*signs-pre2*)
using *prod-cases3* **by** *blast*
have *fst-comb*: *fst* (*combine-systems-R ?eval-p* ((*map* (*eval-mpoly-poly val*) *?q1*), *me1*) ((*map* (*eval-mpoly-poly val*) *?q2*), *me2*)) = (*map* (*eval-mpoly-poly val*) *?q1*) @ (*map* (*eval-mpoly-poly val*) *?q2*)
proof −
have *fst* (*combine-systems-R* (*eval-mpoly-poly val p*) ((*map* (*eval-mpoly-poly val*) *?q1*), *me1*) ((*map* (*eval-mpoly-poly val*) *?q2*), *me2*)) =
fst (*smash-systems-R* (*eval-mpoly-poly val p*) (*map* (*eval-mpoly-poly val*) *?q1*) (*map* (*eval-mpoly-poly val*) *?q2*) *subs-pre1 subs-pre2 signs-pre1 signs-pre2 mat-pre1 mat-pre2*)
using *combining-to-smash-R me1-decomp me2-decomp* **by** *force*
then show *?thesis*
unfolding *smash-systems-R-def* **by** *auto*
qed
then have *map* (*eval-mpoly-poly val*) *qs* = *fst* (*combine-systems-R* (*eval-mpoly-poly val p*) ((*map* (*eval-mpoly-poly val*) *?q1*), *me1*) ((*map* (*eval-mpoly-poly val*) *?q2*), *me2*))
by (*metis append-take-drop-id map-append*)
then have *me-pre-var*: (*map* (*eval-mpoly-poly val*) *qs*, *me-pre*) = (*combine-systems-R* (*eval-mpoly-poly val p*) ((*map* (*eval-mpoly-poly val*) *?q1*), *me1*) ((*map* (*eval-mpoly-poly val*) *?q2*), *me2*))
using *me-pre* **by** *auto*
have *len-hyp*: *length* (*map* (*eval-mpoly-poly val*) *qs*) > *1*
using ∗ **by** *auto*
have *len-eq*: *length* (*map* (*eval-mpoly-poly val*) *qs*) = *length qs*
by *simp*
have *len-q1-gt0*: *length* (*map* (*eval-mpoly-poly val*) *?q1*) > *0*
using *len-hyp len-eq* **by** *auto*
have *len-q2-gt0*: *length* (*map* (*eval-mpoly-poly val*) *?q2*) > *0*
using *len-hyp len-eq* **by** *auto*
let *?uni-sys-q1* = *calculate-data-R* (*eval-mpoly-poly val p*) (*map* (*eval-mpoly-poly val*) *?q1*)
have *sat-props-q1-univ*: *satisfies-properties-R ?eval-p* (*map* (*eval-mpoly-poly val*) *?q1*) (*get-subsets-R ?uni-sys-q1*) (*get-signs-R ?uni-sys-q1*) (*get-matrix-R ?uni-sys-q1*)
using *calculate-data-satisfies-properties-R*[*of ?eval-p* (*map* (*eval-mpoly-poly val*) (*take* (*length qs div 2*) *qs*))]
len-q1-gt0 **using** *less.prems*(*3*) **by** *auto*
let *?uni-sys-q2* = *calculate-data-R* (*eval-mpoly-poly val p*) (*map* (*eval-mpoly-poly val*) *?q2*)
have *sat-props-q2-univ*: *satisfies-properties-R* (*eval-mpoly-poly val p*) (*map*

(*eval-mpoly-poly val*) *?q2*) (*get-subsets-R ?uni-sys-q2*) (*get-signs-R ?uni-sys-q2*) (*get-matrix-R ?uni-sys-q2*)
using *calculate-data-satisfies-properties-R*[*of* (*eval-mpoly-poly val p*) (*map* (*eval-mpoly-poly val*) (*drop* (*length qs div 2*) *qs*))]
len-q2-gt0 **using** *less.prems*(*3*) **by** *auto*
have *comb-satisfies*: *satisfies-properties-R ?eval-p* (*?eval-q1*@(*map* (*eval-mpoly-poly val*) *?q2*))
(*get-subsets-R* (*snd* ((*combine-systems-R ?eval-p* (*?eval-q1*,*?uni-sys-q1*) (*?eval-q2*,*?uni-sys-q2*)))))

(*get-signs-R* (*snd* ((*combine-systems-R ?eval-p* (*?eval-q1*,*?uni-sys-q1*) (*?eval-q2*,*?uni-sys-q2*)))))

(*get-matrix-R* (*snd* ((*combine-systems-R ?eval-p* (*?eval-q1*,*?uni-sys-q1*) (*?eval-q2*,*?uni-sys-q2*)))))
using *combining-sys-satisfies-properties-R*[*of ?eval-p ?eval-q1 ?eval-q2*] *len-q1-gt0 len-q2-gt0 less.prems*(*3*)
sat-props-q2-univ sat-props-q1-univ
by *auto*
then have *well-def-subs-pre*: *all-list-constr-R* (*get-subsets-R* (*snd* ((*combine-systems-R ?eval-p* (*?eval-q1*,*?uni-sys-q1*) (*?eval-q2*,*?uni-sys-q2*))))) (*length* (*?eval-q1*@*?eval-q2*))
unfolding *satisfies-properties-R-def* **by** *auto*
have *get-subs-pre*: (*get-subsets-R* (*snd* ((*combine-systems-R ?eval-p* (*?eval-q1*,*?uni-sys-q1*) (*?eval-q2*,*?uni-sys-q2*))))) = *subs-pre*
using *me-decomp* **unfolding** *get-subsets-R-def*
by (*metis fst-conv me1-ind me2-ind me-pre-var snd-conv*)
have *well-def*: ($\bigwedge$*Is1 Is2 n*. (*Is1*, *Is2*) ∈ *set* (*fst* (*snd me-pre*)) ⟹ *n* ∈ *set Is1* ∨ *n* ∈ *set Is2* ⟹ *n* < *length qs*)
using *well-def-subs-pre get-subs-pre*
all-list-constr-R-matches-well-def[*of subs-pre*]
by (*smt* (*verit*, *ccfv-SIG*) *fst-comb fst-conv get-subsets-R-def length-map me1-ind me2-ind me-pre-var snd-conv*)
then have *reduce-mat-eq*: *mat-eq* = *reduce-system-R* (*eval-mpoly-poly val p*) ((*map* (*eval-mpoly-poly val*) *qs*), *me-pre*)
using *reduce-system-single-M-univariate*[*OF reduce-prop*(*2*) *less.prems*(*2*)]
by (*metis prod.exhaust-sel*)
let *?eval-qs* = (*map* (*eval-mpoly-poly val*) *qs*)
have *calculate-data-R* (*eval-mpoly-poly val p*) (*map* (*eval-mpoly-poly val*) *qs*) =

(*let q1* = *take* ((*length ?eval-qs*) *div 2*) *?eval-qs*; *left* = *calculate-data-R ?eval-p q1*;
q2 = *drop* ((*length ?eval-qs*) *div 2*) *?eval-qs*; *right* = *calculate-data-R ?eval-p q2*;
comb = *combine-systems-R ?eval-p* (*q1*,*left*) (*q2*,*right*) *in*
reduce-system-R ?eval-p comb)
using *len-hyp*
by (*smt* (*z3*) *calculate-data-R.simps less-one nat-less-le semiring-norm*(*136*))

moreover have ... = (*let q1* =(*map* (*eval-mpoly-poly val*) *?q1*);
left = *calculate-data-R* (*eval-mpoly-poly val p*) *q1*;
q2 = (*map* (*eval-mpoly-poly val*) *?q2*);
right = *calculate-data-R* (*eval-mpoly-poly val p*) *q2*

in Let (*combine-systems-R* (*eval-mpoly-poly val p*) (*q1*, *left*) (*q2*, *right*))
(*reduce-system-R* (*eval-mpoly-poly val p*)))
using *map-q1 map-q2* **by** *auto*
moreover have ... = (*let q1* =(*map* (*eval-mpoly-poly val*) *?q1*); *left* = *me1*;
q2 = (*map* (*eval-mpoly-poly val*) *?q2*); *right* = *me2 in*
Let (*combine-systems-R* (*eval-mpoly-poly val p*) (*q1*, *left*) (*q2*, *right*))
(*reduce-system-R* (*eval-mpoly-poly val p*)))
unfolding *Let-def me1-ind me2-ind* **by** *auto*
moreover have ... = *mat-eq*
unfolding *Let-def reduce-mat-eq me-pre-var* **by** *auto*
ultimately have *calculate-data-R* (*eval-mpoly-poly val p*) (*map* (*eval-mpoly-poly val*) *qs*) = *mat-eq*
by *auto*
}
ultimately show *?case* **by** *blast*
qed

lemma *calculate-data-M-assumps-univariate*:
assumes *mat-eq*: (*assumps*, *mat-eq*) ∈ *set* (*calculate-data-assumps-M p qs init-assumps*)
assumes ⋀*p n*. (*p*,*n*) ∈ *set assumps* ⟹ *satisfies-evaluation val p n*
assumes *p-nonzero*: *eval-mpoly-poly val p* ≠ *0*
shows *calculate-data-R* (*eval-mpoly-poly val p*) (*map* (*eval-mpoly-poly val*) *qs*) = *mat-eq*
using *assms*
proof (*induct length qs arbitrary*: *val p mat-eq assumps qs rule*: *less-induct*)
case (*less qs val p mat-eq assumps*)
have *length qs* = *0* ∨ *length qs* = *1* ∨ *length qs* > *1*
by (*meson less-one nat-neq-iff*)
moreover **{assume** ∗: *length qs* = *0*
let *?m* = (*mat-of-rows-list 3* [[*1*,*1*,*1*], [*0*,*1*,*0*], [*1*,*0*,−*1*]])
let *?subs* = [([], []),([*0*], []),([], [*0*])]
let *?signs* = [[*1*],[*0*],[−*1*]]
let *?eval-p* = *eval-mpoly-poly val p*
have *mat-eq-in*: (*assumps*, *mat-eq*) ∈ *set* (*calculate-data-assumps-M p* [] *init-assumps*)
using ∗ *less.prems*(*1*) **by** *auto*
let *?map-base* = *map* (λ(*assumps*,(*a*,(*b*,*c*))). (*assumps*, (*a*,*b*,*map* (*drop 1*) *c*))) (*reduce-system-M p* [*1*] (*base-case-info-M-assumps init-assumps*))
have (*assumps*, *mat-eq*) ∈ *set ?map-base*
using *mat-eq-in*
by *auto*
then obtain *a1 b1 c1* **where** *a1b1c1-prop*:
(*assumps*, (*a1*, (*b1*, *c1*))) ∈ *set* (*reduce-system-M p* [*1*] (*base-case-info-M-assumps init-assumps*))
mat-eq = (*a1*, (*b1*, *map* (*drop 1*) *c1*))
by *auto*
have *base-case-well-def*: ⋀*in-a init-mat-eq Is1 Is2 n subs m signs*.
(*in-a*, *m*, *subs*, *signs*) ∈ *set* (*base-case-info-M-assumps init-assumps*) ⟹
(*Is1*, *Is2*) ∈ *set subs* ⟹ *n* ∈ *set Is1* ∨ *n* ∈ *set Is2* ⟹ *n* < *length* [*1*]
using *base-case-with-assumps-info-M-well-def*

by *auto*
have [*1*] = *map* (*eval-mpoly-poly val*) [*1*]
unfolding *eval-mpoly-poly-def eval-mpoly-def*
by *auto*
then have ∃ *acc mss*. (*acc*,*mss*) ∈ *set* ((*base-case-info-M-assumps init-assumps*))
∧ (*a1*, (*b1*, *c1*)) = *reduce-system-R* (*eval-mpoly-poly val p*) ([*1*],*mss*)
using *reduce-system-M-univariate*[*of assumps* (*a1*, *b1*, *c1*) *p* [*1*] (*base-case-info-M-assumps init-assumps*) *val* [*1*]]
a1b1c1-prop less(*3*) *base-case-well-def* **apply** (*auto*)
by *metis*
then obtain *acc mss* **where** *a1b1c1-connect*:
(*acc*,*mss*) ∈ *set* ((*base-case-info-M-assumps init-assumps*))
(*a1*, (*b1*, *c1*)) = *reduce-system-R* (*eval-mpoly-poly val p*) ([*1*],*mss*)
by *auto*
then have *mss-is*: *mss* = *base-case-info-R*
unfolding *base-case-info-M-assumps-def*
using *Renegar-Algorithm.base-case-info-R-def Renegar-Algorithm.base-case-info-R-def*
by *auto*
obtain *a b c* **where** *abc-prop*: (*a*, *b*, *c*) = *reduction-step-R ?m ?signs ?subs* (*solve-for-lhs-R ?eval-p* [*1*] *?subs ?m*)
by (*metis reduction-step-R.simps*)
then have (*a*, *b*, *c*) = (*a1*, *b1*, *c1*)
using *abc-prop a1b1c1-prop*
by (*metis a1b1c1-connect*(*2*) *base-case-info-R-def mss-is reduce-system-R.simps*)

then have *mat-eq-is*: (*a*, *b*, *map* (*drop* (*Suc 0*)) *c*) = *mat-eq*
using *a1b1c1-prop*(*2*) **by** (*auto*)
have (*a*, *b*, *c*) =
reduction-step-R (*mat-of-rows-list 3* [[*1*, *1*, *1*], [*0*, *1*, *0*], [*1*, *0*, − *1*]])
[[*1*], [*0*], [− *1*]] [([], []), ([*0*], []), ([], [*0*])]
(*solve-for-lhs-R* (*eval-mpoly-poly val p*) [*1*]
[([], []), ([*0*], []), ([], [*0*])]
(*mat-of-rows-list 3* [[*1*, *1*, *1*], [*0*, *1*, *0*], [*1*, *0*, − *1*]]))
using *abc-prop*
unfolding *base-case-info-R-def*
by (*smt* (*verit*) *old.prod.case reduce-system-R.simps*)
then have *calculate-data-R ?eval-p* (*map* (*eval-mpoly-poly val*) *qs*)= *mat-eq*
using *mat-eq-is* ∗
by (*smt* (*z3*) *a1b1c1-connect*(*2*) *a1b1c1-prop*(*2*) *calculate-data-R.simps length-map mss-is split-conv*)
}
moreover {assume ∗: *length qs* = *1*
have *meq*: (*assumps*, *mat-eq*) ∈ *set* (*reduce-system-M p qs* (*base-case-info-M-assumps init-assumps*))
using ∗ *le-eq-less-or-eq less*(*2*) *one-neq-zero* **by** *auto*
have ∗∗: ⋀*init-assumps init-mat-eq Is1 Is2 n subs m signs*.
(*init-assumps*, *m*, *subs*, *signs*) ∈ *set* (*base-case-info-M-assumps init-assumps*) ⟹
(*Is1*, *Is2*) ∈ *set subs* ⟹

n ∈ *set Is1* ∨ *n* ∈ *set Is2* ⟹
n < *length qs* **unfolding** ∗
using *base-case-with-assumps-info-M-well-def*
by *meson*
from *reduce-system-M-univariate*[*OF meq less*(*3*), *of map* (*eval-mpoly-poly val*)
qs]
obtain *acc mss* **where** *ams*: (*acc*,*mss*) ∈ *set* (*base-case-info-M-assumps init-assumps*)

mat-eq = *reduce-system-R* (*eval-mpoly-poly val p*)
(*map* (*eval-mpoly-poly val*) *qs*, *mss*)
using ∗∗
apply (*auto*)
by (*smt* (*z3*) ∗ *base-case-with-assumps-info-M-well-def*)
then have *mss* = *base-case-info-R* **unfolding** *base-case-info-M-assumps-def*
using *Renegar-Algorithm.base-case-info-R-def Renegar-Algorithm.base-case-info-R-def*
by *auto*
then have *calculate-data-R* (*eval-mpoly-poly val p*) (*map* (*eval-mpoly-poly val*)
qs) = *mat-eq*
using *ams* ∗ **by** *simp*
}

moreover **{assume** ∗: *length qs* > *1*
have *inset*: (*assumps*, *mat-eq*) ∈ *set* (*calculate-data-assumps-M p qs init-assumps*)
using *less.prems*(*1*)
by *auto*
let *?len* = *length qs*
let *?q1* = *take* (*?len div 2*) *qs*
let *?q2* = *drop* (*?len div 2*) *qs*
let *?left* = *calculate-data-assumps-M p ?q1 init-assumps*
let *?right* = *calculate-data-assumps-M p ?q2 init-assumps*
let *?comb* = *combine-systems-M p ?q1 ?left ?q2 ?right*
let *?eval-p* = (*eval-mpoly-poly val p*)
let *?eval-q1* = (*map* (*eval-mpoly-poly val*) *?q1*)
let *?eval-q2* = (*map* (*eval-mpoly-poly val*) *?q2*)
have *map-q1*: *map* (*eval-mpoly-poly val*) *?q1* =
(*take* (*length* (*map* (*eval-mpoly-poly val*) *qs*) *div 2*) (*map* (*eval-mpoly-poly val*)
qs))
by (*auto simp add*: *take-map*)
have *map-q2*: *map* (*eval-mpoly-poly val*) *?q2* =
(*drop* (*length* (*map* (*eval-mpoly-poly val*) *qs*) *div 2*) (*map* (*eval-mpoly-poly val*)
qs))
by (*auto simp add*: *drop-map*)
have *fst ?comb* = *qs*
by *auto*
then have (*assumps*, *mat-eq*) ∈ *set* (*reduce-system-M p qs* (*snd ?comb*))
using *inset* ∗
by (*smt* (*z3*) *calculate-data-assumps-M.simps gr-implies-not0 less-one nat-less-le*)

then have (*assumps*, *mat-eq*) ∈ *set* (*concat* (*map* (*reduce-system-single-M p*

qs) (*snd ?comb*)))
by (*metis reduce-system-M.simps*)
then have ∃ *sys*. *sys* ∈ *set* (*snd ?comb*) ∧ (*assumps*, *mat-eq*) ∈ *set*(*reduce-system-single-M p qs sys*)
using *concat-map-in-set in-set-member*
by (*metis nth-mem*)
then obtain *a-pre me-pre* **where** *reduce-prop*: (*a-pre*, *me-pre*) ∈ *set* (*snd ?comb*)
(*assumps*, *mat-eq*) ∈ *set*(*reduce-system-single-M p qs* (*a-pre*, *me-pre*))
by *fastforce*
then obtain *a1 me1 a2 me2* **where** *mes-prop*: (*a1*, *me1*) ∈ *set ?left*
(*a2*, *me2*) ∈ *set ?right*
(*a-pre*, *me-pre*) = *combine-systems-single-M p ?q1* (*a1*, *me1*) *?q2* (*a2*, *me2*)
by *auto*
then have *a-pre*: *a-pre* = *a1@a2*
by *auto*
have *lengt*: *length qs div 2* ≥ *1*
using ∗ **by** *auto*
then have *len-q1*: *length ?q1* < *length qs*
by *auto*
have *len-q2*: *length ?q2* < *length qs*
using *lengt* **by** *auto*
obtain *mat-pre subs-pre signs-pre* **where** *me-decomp*:
me-pre = (*mat-pre*,*subs-pre*,*signs-pre*)
using *mes-prop*
using *prod-cases3* **by** *blast*
then have *assumps* ∈ *set* (*map fst* (*solve-for-lhs-M p a-pre qs subs-pre mat-pre*))
using *reduce-prop*(*2*) **by** *auto*
then have *assumps* ∈ *set* (*map fst* (*construct-rhs-vector-M p a-pre qs subs-pre*))
unfolding *solve-for-lhs-M-def* **by** *auto*
then obtain *a-rhs-list* **where** (*assumps*, *a-rhs-list*)
∈ *set* (*construct-rhs-vector-rec-M p a-pre* (*pull-out-pairs qs subs-pre*))
unfolding *construct-rhs-vector-M-def* **by** *auto*
then have *a-pre-subset*: *set a-pre* ⊆ *set assumps*
using *construct-rhs-vector-rec-M-subset-prop*[*of assumps - p a-pre* (*pull-out-pairs qs subs-pre*)]
by *auto*
have *set-a1*: *set a1* ⊆ *set assumps*
using *a-pre-subset a-pre* **by** *auto*
then have *a1-satisfies*: (⋀*p n*. (*p*, *n*) ∈ *set a1* ⟹ *satisfies-evaluation val p n*)
using *less*(*3*) **by** *blast*
from *less.hyps*[*of ?q1 a1 me1*, *OF len-q1 mes-prop*(*1*) *a1-satisfies*]
have *me1-ind*: *calculate-data-R* (*eval-mpoly-poly val p*) (*map* (*eval-mpoly-poly val*) *?q1*) = *me1*
using *less*(*4*) **by** *blast*
have *set-a2*: *set a2* ⊆ *set assumps*
using *a-pre-subset a-pre* **by** *auto*
then have *a2-satisfies*: (⋀*p n*. (*p*, *n*) ∈ *set a2* ⟹ *satisfies-evaluation val p n*)
using *less*(*3*) **by** *blast*

from *less.hyps*[*of ?q2 a2 me2*, *OF len-q2 mes-prop*(*2*) *a2-satisfies*]
have *me2-ind*: *calculate-data-R ?eval-p* (*map* (*eval-mpoly-poly val*) *?q2*) = *me2*
using *less*(*4*) **by** *blast*
have *me-pre*: *me-pre* = *snd* (*combine-systems-R* (*eval-mpoly-poly val p*) ((*map* (*eval-mpoly-poly val*) *?q1*), *me1*) ((*map* (*eval-mpoly-poly val*) *?q2*), *me2*))
using *mes-prop*(*3*) *combine-systems-R-snd length-map*
by (*metis combine-systems-single-M.simps snd-conv*)
obtain *mat-pre1 subs-pre1 signs-pre1* **where** *me1-decomp*:
me1 = (*mat-pre1*,*subs-pre1*,*signs-pre1*)
using *prod-cases3* **by** *blast*
obtain *mat-pre2 subs-pre2 signs-pre2* **where** *me2-decomp*:
me2 = (*mat-pre2*,*subs-pre2*,*signs-pre2*)
using *prod-cases3* **by** *blast*
have *fst-comb*: *fst* (*combine-systems-R ?eval-p* ((*map* (*eval-mpoly-poly val*) *?q1*), *me1*) ((*map* (*eval-mpoly-poly val*) *?q2*), *me2*)) = (*map* (*eval-mpoly-poly val*) *?q1*) @ (*map* (*eval-mpoly-poly val*) *?q2*)
proof −
have *fst* (*combine-systems-R* (*eval-mpoly-poly val p*) ((*map* (*eval-mpoly-poly val*) *?q1*), *me1*) ((*map* (*eval-mpoly-poly val*) *?q2*), *me2*)) =
fst (*smash-systems-R* (*eval-mpoly-poly val p*) (*map* (*eval-mpoly-poly val*) *?q1*) (*map* (*eval-mpoly-poly val*) *?q2*) *subs-pre1 subs-pre2 signs-pre1 signs-pre2 mat-pre1 mat-pre2*)
using *combining-to-smash-R me1-decomp me2-decomp* **by** *force*
then show *?thesis*
unfolding *smash-systems-R-def* **by** *auto*
qed
then have *map* (*eval-mpoly-poly val*) *qs* = *fst* (*combine-systems-R* (*eval-mpoly-poly val p*) ((*map* (*eval-mpoly-poly val*) *?q1*), *me1*) ((*map* (*eval-mpoly-poly val*) *?q2*), *me2*))
by (*metis append-take-drop-id map-append*)
then have *me-pre-var*: (*map* (*eval-mpoly-poly val*) *qs*, *me-pre*) = (*combine-systems-R* (*eval-mpoly-poly val p*) ((*map* (*eval-mpoly-poly val*) *?q1*), *me1*) ((*map* (*eval-mpoly-poly val*) *?q2*), *me2*))
using *me-pre* **by** *auto*
have *len-hyp*: *length* (*map* (*eval-mpoly-poly val*) *qs*) > *1*
using ∗ **by** *auto*
have *len-eq*: *length* (*map* (*eval-mpoly-poly val*) *qs*) = *length qs*
by *simp*
have *len-q1-gt0*: *length* (*map* (*eval-mpoly-poly val*) *?q1*) > *0*
using *len-hyp len-eq* **by** *auto*
have *len-q2-gt0*: *length* (*map* (*eval-mpoly-poly val*) *?q2*) > *0*
using *len-hyp len-eq* **by** *auto*
let *?uni-sys-q1* = *calculate-data-R* (*eval-mpoly-poly val p*) (*map* (*eval-mpoly-poly val*) *?q1*)
have *sat-props-q1-univ*: *satisfies-properties-R ?eval-p* (*map* (*eval-mpoly-poly val*) *?q1*) (*get-subsets-R ?uni-sys-q1*) (*get-signs-R ?uni-sys-q1*) (*get-matrix-R ?uni-sys-q1*)
using *calculate-data-satisfies-properties-R*[*of ?eval-p* (*map* (*eval-mpoly-poly val*) (*take* (*length qs div 2*) *qs*))]
len-q1-gt0 **using** *less.prems*(*3*) **by** *auto*

let *?uni-sys-q2* = *calculate-data-R* (*eval-mpoly-poly val p*) (*map* (*eval-mpoly-poly val*) *?q2*)
have *sat-props-q2-univ*: *satisfies-properties-R* (*eval-mpoly-poly val p*) (*map* (*eval-mpoly-poly val*) *?q2*) (*get-subsets-R ?uni-sys-q2*) (*get-signs-R ?uni-sys-q2*) (*get-matrix-R ?uni-sys-q2*)
using *calculate-data-satisfies-properties-R*[*of* (*eval-mpoly-poly val p*) (*map* (*eval-mpoly-poly val*) (*drop* (*length qs div 2*) *qs*))]
len-q2-gt0 **using** *less.prems*(*3*) **by** *auto*
have *comb-satisfies*: *satisfies-properties-R ?eval-p* (*?eval-q1*@(*map* (*eval-mpoly-poly val*) *?q2*))
(*get-subsets-R* (*snd* ((*combine-systems-R ?eval-p* (*?eval-q1*,*?uni-sys-q1*) (*?eval-q2*,*?uni-sys-q2*)))))

(*get-signs-R* (*snd* ((*combine-systems-R ?eval-p* (*?eval-q1*,*?uni-sys-q1*) (*?eval-q2*,*?uni-sys-q2*)))))

(*get-matrix-R* (*snd* ((*combine-systems-R ?eval-p* (*?eval-q1*,*?uni-sys-q1*) (*?eval-q2*,*?uni-sys-q2*)))))
using *combining-sys-satisfies-properties-R*[*of ?eval-p ?eval-q1 ?eval-q2*] *len-q1-gt0 len-q2-gt0 less.prems*(*3*)
sat-props-q2-univ sat-props-q1-univ
by *auto*
then have *well-def-subs-pre*: *all-list-constr-R* (*get-subsets-R* (*snd* ((*combine-systems-R ?eval-p* (*?eval-q1*,*?uni-sys-q1*) (*?eval-q2*,*?uni-sys-q2*))))) (*length* (*?eval-q1*@*?eval-q2*))
unfolding *satisfies-properties-R-def* **by** *auto*
have *get-subs-pre*: (*get-subsets-R* (*snd* ((*combine-systems-R ?eval-p* (*?eval-q1*,*?uni-sys-q1*) (*?eval-q2*,*?uni-sys-q2*))))) = *subs-pre*
using *me-decomp* **unfolding** *get-subsets-R-def*
by (*metis fst-conv me1-ind me2-ind me-pre-var snd-conv*)
have *well-def*: ($\bigwedge$*Is1 Is2 n*. (*Is1*, *Is2*) $\in$ *set* (*fst* (*snd me-pre*)) $\Longrightarrow$ *n* $\in$ *set Is1* $\vee$ *n* $\in$ *set Is2* $\Longrightarrow$ *n* $<$ *length qs*)
using *well-def-subs-pre get-subs-pre*
all-list-constr-R-matches-well-def[*of subs-pre*]
by (*smt* (*verit*, *ccfv-SIG*) *fst-comb fst-conv get-subsets-R-def length-map me1-ind me2-ind me-pre-var snd-conv*)
then have *reduce-mat-eq*: *mat-eq* = *reduce-system-R* (*eval-mpoly-poly val p*) ((*map* (*eval-mpoly-poly val*) *qs*), *me-pre*)
using *reduce-system-single-M-univariate*[*OF reduce-prop*(*2*) *less.prems*(*2*)]
by (*metis prod.exhaust-sel*)
let *?eval-qs* = (*map* (*eval-mpoly-poly val*) *qs*)
have *calculate-data-R* (*eval-mpoly-poly val p*) (*map* (*eval-mpoly-poly val*) *qs*) =

(*let q1* = *take* ((*length ?eval-qs*) *div 2*) *?eval-qs*; *left* = *calculate-data-R ?eval-p q1*;
q2 = *drop* ((*length ?eval-qs*) *div 2*) *?eval-qs*; *right* = *calculate-data-R ?eval-p q2*;
comb = *combine-systems-R ?eval-p* (*q1*,*left*) (*q2*,*right*) *in*
reduce-system-R ?eval-p comb)
using *len-hyp*
by (*smt* (*z3*) *calculate-data-R.simps less-one nat-less-le semiring-norm*(*136*))

moreover have ... = (*let q1* =(*map* (*eval-mpoly-poly val*) *?q1*);

left = *calculate-data-R* (*eval-mpoly-poly val p*) *q1*;
q2 = (*map* (*eval-mpoly-poly val*) *?q2*);
right = *calculate-data-R* (*eval-mpoly-poly val p*) *q2*
in Let (*combine-systems-R* (*eval-mpoly-poly val p*) (*q1*, *left*) (*q2*, *right*))
(*reduce-system-R* (*eval-mpoly-poly val p*)))
using *map-q1 map-q2* **by** *auto*
moreover have ... = (*let q1* =(*map* (*eval-mpoly-poly val*) *?q1*); *left* = *me1*;
q2 = (*map* (*eval-mpoly-poly val*) *?q2*); *right* = *me2 in*
Let (*combine-systems-R* (*eval-mpoly-poly val p*) (*q1*, *left*) (*q2*, *right*))
(*reduce-system-R* (*eval-mpoly-poly val p*)))
unfolding *Let-def me1-ind me2-ind* **by** *auto*
moreover have ... = *mat-eq*
unfolding *Let-def reduce-mat-eq me-pre-var* **by** *auto*
ultimately have *calculate-data-R* (*eval-mpoly-poly val p*) (*map* (*eval-mpoly-poly val*) *qs*) = *mat-eq*
by *auto*
}
ultimately show *?case* **by** *blast*
qed

lemma *calculate-data-gives-signs-at-roots*:
assumes (*assumps*, *signs*) ∈ *set* (*calculate-data-to-signs* (*calculate-data-M p qs*))
assumes $\bigwedge$*p n*. (*p*,*n*) ∈ *set assumps* ⟹ *satisfies-evaluation val p n*
assumes *eval-mpoly-poly val p* ≠ *0*
shows *signs* = *find-consistent-signs-at-roots-R* (*eval-mpoly-poly val p*) (*map* (*eval-mpoly-poly val*) *qs*)
using *assms calculate-data-M-univariate*
unfolding *find-consistent-signs-at-roots-R-def* **by** *auto*

lemma *calculate-data-gives-noncomp-signs-at-roots*:
assumes (*assumps*, *signs*) ∈ *set* (*calculate-data-to-signs* (*calculate-data-M p qs*))
assumes $\bigwedge$*p n*. (*p*,*n*) ∈ *set assumps* ⟹ *satisfies-evaluation val p n*
assumes *eval-mpoly-poly val p* ≠ *0*
shows *set signs* = *set* (*characterize-consistent-signs-at-roots* (*eval-mpoly-poly val p*) (*map* (*eval-mpoly-poly val*) *qs*))
using *assms find-consistent-signs-at-roots-R calculate-data-gives-signs-at-roots*
by *metis*

lemma *calculate-data-assumps-gives-signs-at-roots*:
assumes (*assumps*, *signs*) ∈ *set* (*calculate-data-to-signs* (*calculate-data-assumps-M p qs init-assumps*))
assumes $\bigwedge$*p n*. (*p*,*n*) ∈ *set assumps* ⟹ *satisfies-evaluation val p n*
assumes *eval-mpoly-poly val p* ≠ *0*
shows *signs* = *find-consistent-signs-at-roots-R* (*eval-mpoly-poly val p*) (*map* (*eval-mpoly-poly val*) *qs*)
using *assms calculate-data-M-assumps-univariate*
unfolding *find-consistent-signs-at-roots-R-def*
by *auto*

lemma *calculate-data-assumps-gives-noncomp-signs-at-roots*:
assumes (*assumps*, *signs*) ∈ *set* (*calculate-data-to-signs* (*calculate-data-assumps-M p qs init-assumps*))
assumes ⋀*p n*. (*p*,*n*) ∈ *set assumps* ⟹ *satisfies-evaluation val p n*
assumes *eval-mpoly-poly val p* ≠ *0*
shows *set signs* = *set* (*characterize-consistent-signs-at-roots* (*eval-mpoly-poly val p*) (*map* (*eval-mpoly-poly val*) *qs*))
using *assms find-consistent-signs-at-roots-R calculate-data-assumps-gives-signs-at-roots*
by *metis*

end

theory *Hybrid-Multiv-Algorithm-Proofs*

imports *Hybrid-Multiv-Algorithm*
Hybrid-Multiv-Matrix-Proofs
Virtual-Substitution.ExportProofs

begin

17.8 Lemmas about branching (lc assump generation)

lemma *lc-assump-generation-induct*[*case-names Base Rec Lookup0 LookupN0*]:
fixes *q* :: *real mpoly Polynomial.poly*
fixes *assumps* ::(*real mpoly* × *rat*) *list*
assumes *base*: ⋀*q assumps*. *q* = *0* ⟹ *P q assumps*
and *rec*: ⋀*q assumps*.
⟦*q* ≠ *0*;
lookup-assump-aux (*Polynomial.lead-coeff q*) *assumps* = *None*;
P (*one-less-degree q*) ((*Polynomial.lead-coeff q*, *0*) # *assumps*)⟧ ⟹
P q assumps
and *lookup0*: ⋀*q assumps*.
⟦*q* ≠ *0*;
lookup-assump-aux (*Polynomial.lead-coeff q*) *assumps* = *Some 0*;
P (*one-less-degree q*) *assumps*⟧ ⟹ *P q assumps*
and *lookupN0*: ⋀*q assumps r*.
⟦*q* ≠ *0*;
lookup-assump-aux (*Polynomial.lead-coeff q*) *assumps* = *Some r*;
r ≠ *0*⟧ ⟹ *P q assumps*
shows *P q assumps*
apply(*induct q assumps rule*: *lc-assump-generation.induct*)
by (*metis base rec lookup0 lookupN0 not-None-eq*)

lemma *lc-assump-generation-subset*:
assumes (*branch-assms*, *branch-poly-list*) ∈ *set*(*lc-assump-generation q assumps*)
shows *set assumps* ⊆ *set branch-assms*

using *assms*
proof (*induct q assumps rule*: *lc-assump-generation-induct*)
case (*Base q assumps*)
then show *?case*
by (*auto simp add*: *lc-assump-generation.simps*)
next
case (*Rec q assumps*)
let *?zero* = *lc-assump-generation* (*one-less-degree q*) ((*Polynomial.lead-coeff q*, (*0*::*rat*)) # *assumps*)
let *?one* = ((*Polynomial.lead-coeff q*, (*1*::*rat*)) # *assumps*, *q*)
let *?minus-one* = ((*Polynomial.lead-coeff q*, (−*1*::*rat*)) # *assumps*, *q*)
have (*branch-assms*, *branch-poly-list*) ∈ *set* (*?one*#*?minus-one*#*?zero*)
using *Rec.hyps Rec*(*4*) *lc-assump-generation.simps* **by** *auto*
then show *?case*
using *Rec*(*3*) **by** *force*
next
case (*Lookup0 q assumps*)
then show *?case*
using *lc-assump-generation.simps*
by *simp*
next
case (*LookupN0 q assumps r*)
then show *?case*
by (*auto simp add*: *lc-assump-generation.simps*)
qed

lemma *branch-init-assms-subset*:
assumes (*branch-assms*, *branch-poly-list*) ∈ *set* (*lc-assump-generation-list qs init-assumps*)
shows *set init-assumps* ⊆ *set branch-assms*
using *assms*
proof (*induct qs arbitrary*: *branch-assms branch-poly-list init-assumps*)
case *Nil*
then show *?case*
by (*simp add*: *lc-assump-generation-list.simps*(*1*))
next
case (*Cons a bpl*)
then show *?case*
using *lc-assump-generation-subset*
apply (*auto simp add*: *lc-assump-generation-list.simps*)
by *blast*
qed

lemma *prod-list-var-gen-nonzero*:
shows *prod-list-var-gen qs* ≠ *0*
proof (*induct qs*)
case *Nil*
then show *?case* **by** *auto*
next

case (*Cons a qs*)
then show *?case* **by** *auto*
qed

lemma *lc-assump-generation-inv*:
assumes (*a*, *q*) ∈ *set* (*lc-assump-generation init-q assumps*)
shows *q* = (*0*::*rmpoly*) ∨ (∃ *i*. (*lookup-assump-aux* (*Polynomial.lead-coeff q*) *a* = *Some i* ∧ *i* ≠ *0*))
using *assms*
proof (*induct init-q assumps arbitrary*: *q a rule*: *lc-assump-generation-induct*)
case (*Base init-q assumps*)
then show *?case*
using *lc-assump-generation.simps* **by** *auto*
next
case (*Rec init-q assumps*)
let *?zero* = *lc-assump-generation* (*one-less-degree init-q*) ((*Polynomial.lead-coeff init-q*, (*0*::*rat*)) # *assumps*)
let *?one* = ((*Polynomial.lead-coeff init-q*, (*1*::*rat*)) # *assumps*, *init-q*)
let *?minus-one* = ((*Polynomial.lead-coeff init-q*, (−*1*::*rat*)) # *assumps*, *init-q*)
have (*a*, *q*) ∈ *set* (*?one* # *?minus-one* # *?zero*)
using *Rec.prems Rec.hyps*(*1*) *Rec.hyps*(*2*) *lc-assump-generation.simps*
by *auto*
then have *eo*: (*a*, *q*) = *?one* ∨ (*a*, *q*) = *?minus-one* ∨ (*a*, *q*) ∈ *set*(*?zero*)
by *auto*
{assume ∗: (*a*, *q*) = *?one*
then have *q* = (*0*::*rmpoly*) ∨ (∃ *i*. (*lookup-assump-aux* (*Polynomial.lead-coeff q*) *a* = *Some i* ∧ *i* ≠ *0*))
by *auto*
}
moreover {assume ∗: (*a*, *q*) = *?minus-one*
then have *q* = (*0*::*rmpoly*) ∨ (∃ *i*. (*lookup-assump-aux* (*Polynomial.lead-coeff q*) *a* = *Some i* ∧ *i* ≠ *0*))
by *auto*
}
moreover **{assume** ∗: (*a*, *q*) ∈ *set*(*?zero*)
then have *q* = (*0*::*rmpoly*) ∨ (∃ *i*. (*lookup-assump-aux* (*Polynomial.lead-coeff q*) *a* = *Some i* ∧ *i* ≠ *0*))
using *Rec.hyps Rec.prems* **by** *auto*
}
ultimately show *?case*
using *lc-assump-generation.simps*
using *eo* **by** *fastforce*
next
case (*Lookup0 init-q assumps*)
then show *?case* **using** *lc-assump-generation.simps* **by** *auto*
next
case (*LookupN0 init-q assumps r*)
then show *?case* **using** *lc-assump-generation.simps* **by** *auto*

qed

lemma *lc-assump-generation-list-inv*:
assumes *val*: $\bigwedge p\ n.\ (p,n) \in set\ branch\text{-}assms \Longrightarrow satisfies\text{-}evaluation\ val\ p\ n$
assumes $(branch\text{-}assms,\ branch\text{-}poly\text{-}list) \in set\ (lc\text{-}assump\text{-}generation\text{-}list\ qs\ init\text{-}assumps)$
shows $q \in set\ branch\text{-}poly\text{-}list \Longrightarrow q = 0 \lor (\exists\, i.\ lookup\text{-}assump\text{-}aux\ (Polynomial.lead\text{-}coeff\ q)\ branch\text{-}assms = Some\ i \land i \neq 0)$
using *assms*
proof (*induct qs arbitrary*: *q init-assumps branch-poly-list branch-assms*)
case *Nil*
then have (*branch-assms*, *branch-poly-list*) ∈ *set* [(*init-assumps*, [])]
using *lc-assump-generation-list.simps* **by** *auto*
then have *branch-poly-list* = []
using *in-set-member*
by *simp*
then show *?case*
using *Nil.prems*(*1*)
by *simp*
next
case (*Cons a qs*)
let *?rec* = *lc-assump-generation a init-assumps*
have *inset*: (*branch-assms*,*branch-poly-list*) ∈ *set* (
concat (*map* (
λ(*new-assumps*, *r*). (*let list-rec* = *lc-assump-generation-list qs new-assumps in*
map (λ*elem*. (*fst elem*, *r*#(*snd elem*))) *list-rec*)) *?rec*))
using *Cons.prems lc-assump-generation-list.simps*
by *auto*
then obtain *new-assumps r* **where** *deconstruct-prop*:
(*new-assumps*, *r*) ∈ *set ?rec*
(*branch-assms*,*branch-poly-list*) ∈ *set* (*let list-rec* = *lc-assump-generation-list qs new-assumps in*
map (λ*elem*. (*fst elem*, *r*#(*snd elem*))) *list-rec*)
using *inset*
by (*metis* (*no-types*, *lifting*) *concat-map-in-set nth-mem prod.collapse split-def*)
then obtain *elem list-rec* **where** *list-rec-prop*:
list-rec = *lc-assump-generation-list qs new-assumps*
elem ∈ *set list-rec*
(*branch-assms*,*branch-poly-list*) = (*fst elem*, *r*#(*snd elem*))
by *auto*
then have *pair-in-set*: (*branch-assms*, *snd elem*) ∈ *set* (*lc-assump-generation-list qs new-assumps*)
using *deconstruct-prop* **by** *auto*
then have *snd-elem-is*: $\bigwedge q.\ q \in set\ (snd\ elem) \Longrightarrow$
$q = 0 \lor (\exists\, i.\ lookup\text{-}assump\text{-}aux\ (Polynomial.lead\text{-}coeff\ q)\ branch\text{-}assms = Some\ i \land i \neq 0)$
using *Cons.hyps*
by (*simp add*: *local.Cons*(*3*))
have *ris-var*: $r = 0 \lor (\exists\, i.\ lookup\text{-}assump\text{-}aux\ (Polynomial.lead\text{-}coeff\ r) new\text{-}assumps$

= *Some i* ∧ *i* ≠ *0*)
using *lc-assump-generation-inv*[*of new-assumps r a init-assumps*] *deconstruct-prop*(*1*)

by *auto*
have *set new-assumps* ⊆ *set branch-assms*
using *deconstruct-prop list-rec-prop*
using *pair-in-set branch-init-assms-subset* **by** *presburger*
then have (∃ *i*. *lookup-assump-aux* (*Polynomial.lead-coeff r*) *new-assumps* = *Some i* ∧ *i* ≠ *0*) ⟹
(∃ *i*. *lookup-assump-aux* (*Polynomial.lead-coeff r*) *branch-assms* = *Some i* ∧ *i* ≠ *0*)
using *val lookup-assump-aux-subset-consistency*
using *local.Cons*(*3*) **by** *blast*
then have *ris*: *r* = *0* ∨ (∃ *i*. *lookup-assump-aux* (*Polynomial.lead-coeff r*) *branch-assms* = *Some i* ∧ *i* ≠ *0*)
using *ris-var* **by** *auto*
then show *?case*
using *ris snd-elem-is list-rec-prop*(*3*)
using *Cons.prems*(*1*) **by** *auto*
qed

17.9 Correctness of sign determination inner

lemma *q-dvd-prod-list-var-prop*:
assumes *q* ∈ *set qs*
assumes *q* ≠ *0*
shows *q dvd prod-list-var-gen qs* **using** *assms*
proof (*induct qs*)
case *Nil*
then show *?case* **by** *auto*
next
case (*Cons a qs*)
then have *eo*: *q* = *a* ∨*q* ∈ *set qs* **by** *auto*
have *c1*: *q* = *a* ⟹ *q dvd prod-list-var-gen* (*a*#*qs*)
proof −
assume *q* = *a*
then have *prod-list-var-gen* (*a*#*qs*) = *q*∗(*prod-list-var-gen qs*) **using** *Cons.prems*
unfolding *prod-list-var-gen-def* **by** *auto*
then show *?thesis* **using** *prod-list-var-gen-nonzero*[*of qs*] **by** *auto*
qed
have *c2*: *q* ∈ *set qs* ⟶ *q dvd prod-list-var-gen qs*
using *Cons.prems Cons.hyps* **unfolding** *prod-list-var-gen-def* **by** *auto*
show *?case* **using** *eo c1 c2* **by** *auto*
qed

lemma *poly-p-nonzero-on-branch*:
assumes *assms*: ⋀*p n*. (*p*,*n*) ∈ *set branch-assms* ⟹ *satisfies-evaluation val p n*
assumes (*branch-assms*, *branch-poly-list*) ∈ *set* (*lc-assump-generation-list qs init-assumps*)

assumes *p* = *poly-p-in-branch* (*branch-assms*, *branch-poly-list*)
shows *eval-mpoly-poly val p* ≠ *0*
proof −

{**assume** ∗: (*check-all-const-deg-gen branch-poly-list* = *True*)
then have *p-is*: *p* = [:*0*, *1*:]
using *assms*
using *poly-p-in-branch.simps* **by** *presburger*
then have *eval-mpoly-poly val p* ≠ *0*
by (*metis Polynomial.lead-coeff-monom degree-0 dvd-refl eval-commutes eval-mpoly-map-poly-comm-ring-ho not-is-unit-0 poly-dvd-1 x-as-monom*)
}
moreover {**assume** ∗: (*check-all-const-deg-gen branch-poly-list* = *False*)
then have *p-is*: *p* = (*pderiv* (*prod-list-var-gen branch-poly-list*)) ∗ (*prod-list-var-gen branch-poly-list*)
using *assms*
using *poly-p-in-branch.simps* **by** *presburger*
then have *assms-inv*: ⋀*q*. *q* ∈ *set branch-poly-list* ⟹ *q* = *0* ∨ (∃ *i*. *lookup-assump-aux* (*Polynomial.lead-coeff q*) *branch-assms* = *Some i* ∧ *i* ≠ *0*)
using *lc-assump-generation-list-inv assms*
by (*meson assms*(*2*))
have *q-inv*: ⋀*q*. *q* ∈ *set branch-poly-list* ⟹ *q* ≠ *0* ⟹ *eval-mpoly-poly val q* ≠ *0*
using *assms-inv assms*
by (*metis eval-commutes leading-coeff-0-iff lookup-assum-aux-mem satisfies-evaluation-nonzero*)
then have ⋀*q*. *q* ∈ *set branch-poly-list* ⟹ (*q* ≠ *0* ⟷ *eval-mpoly-poly val q* ≠ *0*)
by *auto*
then have *prod-list-eval*: (*eval-mpoly-poly val* (*prod-list-var-gen branch-poly-list*)) = (*prod-list-var-gen* (*map* (*eval-mpoly-poly val*) *branch-poly-list*))
proof (*induct branch-poly-list*)
case *Nil*
then show *?case* **by** *auto*
next
case (*Cons a branch-poly-list*)
{ **assume** ∗: *a* = *0*
then have *h1*: *eval-mpoly-poly val* (*prod-list-var-gen* (*a* # *branch-poly-list*)) =
eval-mpoly-poly val (*prod-list-var-gen branch-poly-list*)
by *simp*
have *eval-mpoly-poly val a* = *0*
using ∗ **by** *auto*
then have *h2*: *prod-list-var-gen* (*map* (*eval-mpoly-poly val*) (*a* # *branch-poly-list*)) = *prod-list-var-gen* (*map* (*eval-mpoly-poly val*) *branch-poly-list*)
by *auto*
then have *eval-mpoly-poly val* (*prod-list-var-gen* (*a* # *branch-poly-list*)) = *prod-list-var-gen* (*map* (*eval-mpoly-poly val*) (*a* # *branch-poly-list*))
using *Cons.hyps Cons.prems h1 h2*

by (*simp add*: *member-rec*(*1*))
}
moreover { **assume** ∗: $a \neq 0$
then have *h1*: *eval-mpoly-poly val* (*prod-list-var-gen* (*a* # *branch-poly-list*)) =
(*eval-mpoly-poly val a*)∗(*eval-mpoly-poly val* (*prod-list-var-gen branch-poly-list*))
by (*simp add*: *eval-mpoly-poly-comm-ring-hom.hom-mult*)
have *eval-mpoly-poly val a* $\neq 0$
using ∗ *assms Cons.prems*
by (*meson list.set-intros*(*1*))
then have *h2*: *prod-list-var-gen* (*map* (*eval-mpoly-poly val*) (*a* # *branch-poly-list*))
= (*eval-mpoly-poly val a*)∗ *prod-list-var-gen* (*map* (*eval-mpoly-poly val*) *branch-poly-list*)
by *auto*
have *eval-mpoly-poly val* (*prod-list-var-gen* (*a* # *branch-poly-list*)) =
prod-list-var-gen (*map* (*eval-mpoly-poly val*) (*a* # *branch-poly-list*))
using *Cons.hyps Cons.prems h1 h2*
by (*simp add*: *member-rec*(*1*))
}
ultimately show *?case*
by *auto*
qed
have *degree-q-inv*: $\bigwedge q.\ q \in$ *set branch-poly-list* $\Longrightarrow q \neq 0 \Longrightarrow$ *Polynomial.degree* (*eval-mpoly-poly val q*) = *Polynomial.degree q*
using *assms-inv assms q-inv*
by (*metis degree-valuation lookup-assum-aux-mem*)
have *prod-nonz*: *eval-mpoly-poly val* (*prod-list-var-gen branch-poly-list*) $\neq 0$
using *q-inv prod-list-eval*
by (*simp add*: *prod-list-var-gen-nonzero*)
have *ex-q*: $\exists\, q \in$ *set branch-poly-list*. ($q \neq 0 \wedge$ *Polynomial.degree* $q > 0$)
using ∗ **proof** (*induct branch-poly-list*)
case *Nil*
then show *?case* **by** *auto*
next
case (*Cons a branch-poly-list*)
then show *?case*
by (*metis bot-nat-0.not-eq-extremum check-all-const-deg-gen.simps*(*2*) *degree-0 list.set-intros*(*1*) *list.set-intros*(*2*))
qed
obtain *pos-deg-poly* **where** *pos-deg-poly*: *pos-deg-poly*∈*set branch-poly-list* ∧ *pos-deg-poly* $\neq 0 \wedge 0 <$ *Polynomial.degree pos-deg-poly*
using *ex-q* **by** *blast*
have *pos-deg-poly dvd* (*prod-list-var-gen branch-poly-list*)
by (*simp add*: *Hybrid-Multiv-Algorithm-Proofs.q-dvd-prod-list-var-prop pos-deg-poly*)
then have *nonc-dvd*: (*eval-mpoly-poly val pos-deg-poly*) *dvd* (*eval-mpoly-poly val* (*prod-list-var-gen branch-poly-list*))
by *blast*
have *Polynomial.degree* (*eval-mpoly-poly val pos-deg-poly*) > 0
using *pos-deg-poly degree-q-inv*

by *metis*
then have *prod-nonc*:*Polynomial.degree* (*eval-mpoly-poly val* (*prod-list-var-gen branch-poly-list*)) ≠ *0*
using *nonc-dvd prod-nonz*
by (*metis bot-nat-0.extremum-strict dvd-const*)
then have *eval-mpoly-poly val p* ≠ *0*
using *prod-nonz prod-nonc*
by (*metis eval-mpoly-poly-comm-ring-hom.hom-mult no-zero-divisors p-is pderiv-commutes pderiv-eq-0-iff*)
}
ultimately show *?thesis* **by** *auto*
qed

lemma *calc-data-to-signs-and-extract-signs*:
shows (*calculate-data-to-signs ell*) = *extract-signs ell*
by *auto*

lemma *branch-poly-eval*:
assumes (*a*, *q*) ∈ *set* (*lc-assump-generation init-q init-assumps*)
assumes ⋀*p n*. (*p*,*n*) ∈ *set a* ⟹ *satisfies-evaluation val p n*
shows (*eval-mpoly-poly val*) *q* = (*eval-mpoly-poly val*) *init-q*
using *assms*
proof (*induct init-q init-assumps arbitrary*: *q a rule*: *lc-assump-generation-induct*)
case (*Base init-q init-assumps*)
then show *?case*
by (*simp add*: *lc-assump-generation.simps*)
next
case (*Rec init-q init-assumps*)
then show *?case* **using** *lc-assump-generation.simps*
basic-trans-rules(*31*) *eval-mpoly-poly-one-less-degree in-set-member lc-assump-generation-subset member-rec*(*1*) *option.case*(*1*) *prod.inject*
by (*smt* (*verit*, *best*))
next
case (*Lookup0 init-q init-assumps*)
then show *?case*
using *lc-assump-generation.simps*
by (*smt* (*z3*) *eval-mpoly-poly-one-less-degree lc-assump-generation-subset lookup-assum-aux-mem option.simps*(*5*) *subset-eq*)

next
case (*LookupN0 init-q init-assumps r*)
then show *?case*
by (*simp add*: *lc-assump-generation.simps*)
qed

lemma *eval-prod-list-var-gen-match*:
assumes (*branch-assms*, *branch-poly-list*) ∈ *set* (*lc-assump-generation-list qs init-assumps*)

```
  assumes ⋀p n. (p,n) ∈ set branch-assms ⟹ satisfies-evaluation val p n
  shows eval-mpoly-poly val (prod-list-var-gen branch-poly-list) =
    prod-list-var-gen (map (eval-mpoly-poly val) branch-poly-list)
  using assms
proof (induct qs arbitrary: branch-assms branch-poly-list init-assumps val)
  case Nil
  then have (branch-assms, branch-poly-list) ∈ set [(init-assumps, [])]
    using lc-assump-generation-list.simps by auto
  then have branch-poly-list = []
    using in-set-member
    by simp
  then show ?case
    by simp
next
  case (Cons a qs)
  let ?rec = lc-assump-generation a init-assumps
  have inset: (branch-assms,branch-poly-list) ∈ set (
    concat (map (
     λ(new-assumps, r). (let list-rec = lc-assump-generation-list qs new-assumps in
      map (λelem. (fst elem, r#(snd elem))) list-rec) ) ?rec ))
    using Cons.prems lc-assump-generation-list.simps
    by auto
  then obtain new-assumps r where deconstruct-prop:
    (new-assumps, r) ∈ set ?rec
    (branch-assms,branch-poly-list) ∈ set (let list-rec = lc-assump-generation-list qs
new-assumps in
      map (λelem. (fst elem, r#(snd elem))) list-rec)
    using inset
   by (metis (no-types, lifting) concat-map-in-set nth-mem prod.collapse split-def)
  then obtain elem list-rec where list-rec-prop:
    list-rec = lc-assump-generation-list qs new-assumps
    elem ∈ set list-rec
    (branch-assms,branch-poly-list) = (fst elem, r#(snd elem))
    by auto
 then have branch-assms-inset: (branch-assms, snd elem) ∈ set (lc-assump-generation-list
qs new-assumps)
    using deconstruct-prop by auto
  then have set new-assumps ⊆ set branch-assms
    using deconstruct-prop list-rec-prop branch-init-assms-subset
    by presburger
 have ris-var: r = 0 ∨ (∃ i. lookup-assump-aux (Polynomial.lead-coeff r)new-assumps
= Some i ∧ i ≠ 0)
   using lc-assump-generation-inv[of new-assumps r a init-assumps] deconstruct-prop(1)

    by auto
  then have r-prop: r = 0 ⟷ eval-mpoly-poly val r = 0
   by (metis ‹set new-assumps ⊆ set branch-assms› basic-trans-rules(31) eval-commutes
eval-mpoly-poly-comm-ring-hom.hom-zero leading-coeff-0-iff local.Cons(3) lookup-assum-aux-mem
satisfies-evaluation-nonzero)
```

have *eval-mpoly-poly val* (*prod-list-var-gen* (*snd elem*)) =
prod-list-var-gen (*map* (*eval-mpoly-poly val*) (*snd elem*))
using *Cons.hyps list-rec-prop*
using *branch-assms-inset local.Cons*(*3*) **by** *blast*
then show *?case* **using** *r-prop list-rec-prop*(*3*)
by (*simp add*: *eval-mpoly-poly-comm-ring-hom.hom-mult*)
qed

lemma *map-branch-poly-list*:
assumes (*branch-assms*, *branch-poly-list*) ∈ *set* (*lc-assump-generation-list qs init-assumps*)
assumes ⋀*p n*. (*p*,*n*) ∈ *set branch-assms* ⟹ *satisfies-evaluation val p n*
shows (*map* (*eval-mpoly-poly val*) *qs*) = (*map* (*eval-mpoly-poly val*) *branch-poly-list*)
using *assms*
proof (*induct qs arbitrary*: *branch-assms branch-poly-list init-assumps*)
case *Nil*
then have (*branch-assms*, *branch-poly-list*) ∈ *set* [(*init-assumps*, [])]
using *lc-assump-generation-list.simps* **by** *auto*
then have *branch-poly-list* = []
using *in-set-member*
by *simp*
then show *?case*
using *Nil.prems*(*1*)
by *meson*
next
case (*Cons a qs*)
let *?rec* = *lc-assump-generation a init-assumps*
have *inset*: (*branch-assms*,*branch-poly-list*) ∈ *set* (
concat (*map* (
λ(*new-assumps*, *r*). (*let list-rec* = *lc-assump-generation-list qs new-assumps in*
map (λ*elem*. (*fst elem*, *r*#(*snd elem*))) *list-rec*)) *?rec*))
using *Cons.prems lc-assump-generation-list.simps*
by *auto*
then obtain *new-assumps r* **where** *deconstruct-prop*:
(*new-assumps*, *r*) ∈ *set ?rec*
(*branch-assms*,*branch-poly-list*) ∈ *set* (*let list-rec* = *lc-assump-generation-list qs new-assumps in*
map (λ*elem*. (*fst elem*, *r*#(*snd elem*))) *list-rec*)
using *inset*
by (*metis* (*no-types*, *lifting*) *concat-map-in-set nth-mem prod.collapse split-def*)
then obtain *elem list-rec* **where** *list-rec-prop*:
list-rec = *lc-assump-generation-list qs new-assumps*
elem ∈ *set list-rec*
(*branch-assms*,*branch-poly-list*) = (*fst elem*, *r*#(*snd elem*))
by *auto*
then have *branch-assms-inset*: (*branch-assms*, *snd elem*) ∈ *set* (*lc-assump-generation-list qs new-assumps*)
using *deconstruct-prop* **by** *auto*
then have *map-prop*: *map* (*eval-mpoly-poly val*) (*qs*) =

map (*eval-mpoly-poly val*) (*snd elem*) **using** *Cons.hyps Cons.prems*
by *blast*
have *set new-assumps* ⊆ *set branch-assms*
using *deconstruct-prop list-rec-prop branch-assms-inset branch-init-assms-subset*

by *presburger*
then have *eval-mpoly-poly val r* = *eval-mpoly-poly val a*
using *branch-poly-eval*
by (*meson basic-trans-rules*(*31*) *deconstruct-prop*(*1*) *local.Cons*(*3*))
then show *?case*
using *map-prop list-rec-prop*(*3*)
by *simp*
qed

lemma *check-constant-degree-match*:
assumes (*a*, *q*) ∈ *set* (*lc-assump-generation init-q init-assumps*)
assumes ⋀*p n*. (*p*,*n*) ∈ *set a* ⟹ *satisfies-evaluation val p n*
shows *Polynomial.degree q* = *Polynomial.degree* (*eval-mpoly-poly val init-q*)
using *assms*
proof (*induct init-q init-assumps arbitrary*: *q a rule*: *lc-assump-generation-induct*
)
case (*Base init-q init-assumps*)
then show *?case*
using *lc-assump-generation.simps*
by *simp*
next
case (*Rec init-q init-assumps*)
then show *?case* **using** *lc-assump-generation.simps*
by (*metis branch-poly-eval degree-0 degree-valuation eval-mpoly-poly-comm-ring-hom.hom-zero lc-assump-generation-inv lookup-assum-aux-mem*)
next
case (*Lookup0 init-q init-assumps*)
then show *?case* **using** *lc-assump-generation.simps*
by (*metis branch-poly-eval degree-0 degree-valuation eval-mpoly-poly-comm-ring-hom.hom-zero lc-assump-generation-inv lookup-assum-aux-mem*)
next
case (*LookupN0 init-q init-assumps r*)
then show *?case*
using *lc-assump-generation.simps*
by (*metis branch-poly-eval degree-0 degree-valuation eval-mpoly-poly-comm-ring-hom.hom-zero lc-assump-generation-inv lookup-assum-aux-mem*)
qed

lemma *check-constant-degree-match-list*:
assumes (*branch-assms*, *branch-poly-list*) ∈ *set* (*lc-assump-generation-list qs init-assumps*)
assumes ⋀*p n*. (*p*,*n*) ∈ *set branch-assms* ⟹ *satisfies-evaluation val p n*
shows (*check-all-const-deg-gen branch-poly-list*) = (*check-all-const-deg-gen* (*map* (*eval-mpoly-poly val*) *qs*))

using *assms*
proof (*induct qs arbitrary*: *branch-assms branch-poly-list init-assumps*)
case *Nil*
then have (*branch-assms*, *branch-poly-list*) ∈ *set* [(*init-assumps*, [])]
using *lc-assump-generation-list.simps*
by *auto*
then have *branch-poly-list* = []
using *in-set-member*
by *simp*
then show *?case*
by *simp*
next
case (*Cons a qs*)
let *?rec* = *lc-assump-generation a init-assumps*
have *inset*: (*branch-assms*,*branch-poly-list*) ∈ *set* (
concat (*map* (
λ(*new-assumps*, *r*). (*let list-rec* = *lc-assump-generation-list qs new-assumps in*
map (λ*elem*. (*fst elem*, *r*#(*snd elem*))) *list-rec*)) *?rec*))
using *Cons.prems lc-assump-generation-list.simps*
by *auto*
then obtain *new-assumps r* **where** *deconstruct-prop*:
(*new-assumps*, *r*) ∈ *set ?rec*
(*branch-assms*,*branch-poly-list*) ∈ *set* (*let list-rec* = *lc-assump-generation-list qs new-assumps in*
map (λ*elem*. (*fst elem*, *r*#(*snd elem*))) *list-rec*)
using *inset*
by (*metis* (*no-types*, *lifting*) *concat-map-in-set nth-mem prod.collapse split-def*)
then obtain *elem list-rec* **where** *list-rec-prop*:
list-rec = *lc-assump-generation-list qs new-assumps*
elem ∈ *set list-rec*
(*branch-assms*,*branch-poly-list*) = (*fst elem*, *r*#(*snd elem*))
by *auto*
then have *branch-assms-inset*: (*branch-assms*, *snd elem*) ∈ *set* (*lc-assump-generation-list qs new-assumps*)
using *deconstruct-prop* **by** *auto*
then have *ind-prop*: (*check-all-const-deg-gen* (*snd elem*)) = (*check-all-const-deg-gen* (*map* (*eval-mpoly-poly val*) *qs*)) **using** *Cons.hyps Cons.prems*
by *blast*
have *set new-assumps* ⊆ *set branch-assms*
using *deconstruct-prop list-rec-prop branch-assms-inset branch-init-assms-subset*

by *presburger*
then have *Polynomial.degree r* = *Polynomial.degree* (*eval-mpoly-poly val a*)
using *check-constant-degree-match*
by (*meson basic-trans-rules*(*31*) *deconstruct-prop*(*1*) *local.Cons*(*3*))
then show *?case*
using *ind-prop list-rec-prop*(*3*)
by *simp*
qed

lemma *check-all-const-deg-match*:
  shows *check-all-const-deg qs* = *check-all-const-deg-gen qs*
proof (*induct qs*)
  case *Nil*
  then show *?case* **by** *auto*
next
  case (*Cons a qs*)
  then show *?case* **by** *auto*
qed

lemma *prod-list-var-match*:
  shows *prod-list-var-gen qs* = *prod-list-var qs*
proof (*induct qs*)
  case *Nil*
  then show *?case* **by** *auto*
next
  case (*Cons a qs*)
  then show *?case* **by** *auto*
qed

lemma *sign-lead-coeff-on-branch*:
  assumes (*a*, *q*) ∈ *set* (*lc-assump-generation init-q init-assumps*)
  assumes *q* ≠ *0*
  assumes ⋀*p n*. (*p*,*n*) ∈ *set a* ⟹ *satisfies-evaluation val p n*
  shows ((*Sturm-Tarski.sign* (*lookup-assump* (*Polynomial.lead-coeff q*) *a*))) =
   Sturm-Tarski.sign (*Polynomial.lead-coeff* (*eval-mpoly-poly val q*))
  using *assms*
proof (*induct init-q init-assumps arbitrary*: *q a rule*: *lc-assump-generation-induct*
)
  case (*Base init-q init-assumps*)
  then show *?case* **using** *lc-assump-generation.simps*
    by *simp*
next
  case (*Rec init-q init-assumps*)
  let *?zero* = *lc-assump-generation* (*one-less-degree init-q*) ((*Polynomial.lead-coeff*
init-q, (*0*::*rat*)) # *init-assumps*)
  let *?one* = ((*Polynomial.lead-coeff init-q*, (*1*::*rat*)) # *init-assumps*, *init-q*)
  let *?minus-one* = ((*Polynomial.lead-coeff init-q*, (−*1*::*rat*)) # *init-assumps*,
init-q)
  have *inset*: (*a*, *q*) ∈ *set* (*?one*#*?minus-one*#*?zero*)
    using *Rec.hyps lc-assump-generation.simps Rec*(*4*)
    by *auto*
  { **assume** ∗ : (*a*, *q*) = *?one*
    then have *h1*: *Sturm-Tarski.sign* (*lookup-assump* (*Polynomial.lead-coeff q*) *a*)
= (*1*::*rat*)
      by *auto*
    have *h2*: *satisfies-evaluation val* (*Polynomial.lead-coeff q*) (*1*)
      using *Rec.hyps*

by (*metis* ∗ *Pair-inject Rec.prems*(*3*) *list.set-intros*(*1*))
then have *h3*: *Sturm-Tarski.sign* (*Polynomial.lead-coeff* (*eval-mpoly-poly val q*)) = (*1*::*int*)
unfolding *satisfies-evaluation-def eval-mpoly-def*
by (*metis Sturm-Tarski.sign-def h2 lead-coeff-valuation of-int-hom.injectivity one-neq-zero satisfies-evaluation-def verit-comp-simplify*(*28*))

have *Sturm-Tarski.sign* (*lookup-assump* (*Polynomial.lead-coeff q*) *a*) =
((*Sturm-Tarski.sign* (*Polynomial.lead-coeff* (*eval-mpoly-poly val q*))))
using *h1 h3*
by *auto*
} moreover {
assume ∗ : (*a*, *q*) = *?minus-one*
then have *h1*: *Sturm-Tarski.sign* (*lookup-assump* (*Polynomial.lead-coeff q*) *a*) = (−*1*::*rat*)
by *auto*
have *h2*: *satisfies-evaluation val* (*Polynomial.lead-coeff q*) (−*1*)
using *Rec.hyps*
by (*metis* ∗ *Pair-inject Rec.prems*(*3*) *list.set-intros*(*1*))
then have *h3*: *rat-of-int* (*Sturm-Tarski.sign* (*Polynomial.lead-coeff* (*eval-mpoly-poly val q*))) = (−*1*::*rat*)
unfolding *satisfies-evaluation-def eval-mpoly-def*
by (*metis Sturm-Tarski.sign-def degree-valuation eval-mpoly-def eval-mpoly-map-poly-comm-ring-hom.base.eval-mpoly-poly-def h2 of-int-hom.hom-one of-int-hom.injectivity of-int-minus rel-simps*(*88*) *sign-uminus verit-comp-simplify*(*28*))

have *Sturm-Tarski.sign* (*lookup-assump* (*Polynomial.lead-coeff q*) *a*) =
((*Sturm-Tarski.sign* (*Polynomial.lead-coeff* (*eval-mpoly-poly val q*))))
using *h1 h3*
by (*metis of-int-eq-iff of-rat-of-int-eq*)
}
moreover {
assume ∗ : (*a*, *q*) ∈ *set ?zero*
then have (*a*, *q*)
∈ *set* (*lc-assump-generation* (*Multiv-Poly-Props.one-less-degree init-q*)
((*Polynomial.lead-coeff init-q*, *0*) # *init-assumps*))
by *auto*
then have *set* ((*Polynomial.lead-coeff init-q*, *0*) # *init-assumps*) ⊆ *set a*
using *lc-assump-generation-subset* **by** *presburger*
then have ⋀*p n*. (*p*, *n*) ∈ *set* ((*Polynomial.lead-coeff init-q*, *0*) # *init-assumps*) ⟹ *satisfies-evaluation val p n*
using *Rec.prems* **by** *auto*
then have *Sturm-Tarski.sign* (*lookup-assump* (*Polynomial.lead-coeff q*) *a*) =
Sturm-Tarski.sign (*Polynomial.lead-coeff* (*eval-mpoly-poly val q*))
using *Rec.hyps Rec.prems*
by (*smt* (*verit*, *ccfv-SIG*) ∗ *Sturm-Tarski.sign-cases more-arith-simps*(*10*) *neg-one-neq-one sign-uminus*)

}

ultimately show *?case* **using** *lc-assump-generation.simps*
inset
by (*smt* (*verit*, *del-insts*) *set-ConsD*)
next
case (*Lookup0 init-q init-assumps*)
then show *?case* **using** *lc-assump-generation.simps*
by *auto*

next
case (*LookupN0 init-q init-assumps r*)
then have *match*: (*a*, *q*) = (*init-assumps*, *init-q*)
using *lc-assump-generation.simps in-set-member* **by** *auto*
then obtain *i* **where** *i-prop*: *i* ≠ *0*
lookup-assump-aux (*Polynomial.lead-coeff init-q*) *init-assumps* = *Some i*
(*Polynomial.lead-coeff init-q*, *i*) ∈ *set init-assumps*
using *LookupN0.prems LookupN0.hyps*
by (*meson lookup-assum-aux-mem*)
then have (*Polynomial.lead-coeff init-q*, *i*) ∈ *set a*
using *match* **by** *auto*
then have *sat-eval*: *satisfies-evaluation val* (*Polynomial.lead-coeff init-q*) *i*
using *LookupN0*(*6*) **by** *blast*
have *Sturm-Tarski.sign* (*lookup-assump* (*Polynomial.lead-coeff q*) *a*)
= *Sturm-Tarski.sign i*
using *i-prop match lookup-assump-aux-subset-consistent-sign*
by *simp*
then show *?case*
by (*smt* (*verit*, *del-insts*) *LookupN0*(*6*) *LookupN0.prems*(*1*) *sat-eval branch-poly-eval i-prop*(*1*) *lead-coeff-valuation of-int-hom.injectivity satisfies-evaluation-def*)
qed

lemma *sign-lead-coeff-on-branch-init*:
assumes (*a*, *q*) ∈ *set* (*lc-assump-generation init-q init-assumps*)
assumes *q* ≠ *0*
assumes ⋀*p n*. (*p*,*n*) ∈ *set a* ⟹ *satisfies-evaluation val p n*
shows *Sturm-Tarski.sign* (*lookup-assump* (*Polynomial.lead-coeff q*) *a*) =
Sturm-Tarski.sign (*Polynomial.lead-coeff* (*eval-mpoly-poly val init-q*))
using *sign-lead-coeff-on-branch branch-poly-eval*
by (*metis assms*(*1*) *assms*(*2*) *assms*(*3*))

lemma *pos-limit-point-on-branch*:
assumes (*a*, *q*) ∈ *set* (*lc-assump-generation init-q init-assumps*)
assumes ⋀*p n*. (*p*,*n*) ∈ *set a* ⟹ *satisfies-evaluation val p n*
shows *rat-of-int* (*Sturm-Tarski.sign* (*sgn-pos-inf* (*eval-mpoly-poly val q*))) =
(*if q* = *0 then 0 else Sturm-Tarski.sign* (*lookup-assump* (*Polynomial.lead-coeff q*) *a*))
using *assms*
proof −
have *rat-of-int* (*Sturm-Tarski.sign* (*sgn-pos-inf* (*eval-mpoly-poly val q*))) =
Sturm-Tarski.sign (*Polynomial.lead-coeff* (*eval-mpoly-poly val q*))

unfolding *sgn-pos-inf-def*
by *auto*
then show *?thesis* **using** *sign-lead-coeff-on-branch assms*
by (*smt* (*verit*, *del-insts*) *eval-mpoly-poly-comm-ring-hom.hom-zero leading-coeff-0-iff sign-simps*(*2*))

qed

lemma *pos-limit-point-on-branch-init*:
assumes (*a*, *q*) ∈ *set* (*lc-assump-generation init-q init-assumps*)
assumes ⋀*p n*. (*p*,*n*) ∈ *set a* ⟹ *satisfies-evaluation val p n*
shows *rat-of-int* (*Sturm-Tarski.sign* (*sgn-pos-inf* (*eval-mpoly-poly val init-q*))) =

(*if q* = *0 then 0 else Sturm-Tarski.sign* (*lookup-assump* (*Polynomial.lead-coeff q*) *a*))
using *assms pos-limit-point-on-branch sign-lead-coeff-on-branch-init*
using *branch-poly-eval* **by** *force*

lemma *pos-limit-point-on-branch-list*:
assumes (*branch-assms*, *branch-poly-list*) ∈ *set* (*lc-assump-generation-list qs init-assumps*)
assumes ⋀*p n*. (*p*,*n*) ∈ *set branch-assms* ⟹ *satisfies-evaluation val p n*
assumes (*pos-limit-branch*, *neg-limit-branch*) = *limit-points-on-branch* (*branch-assms*, *branch-poly-list*)
shows *map rat-of-int* (*sgn-pos-inf-rat-list* (*map* (*eval-mpoly-poly val*) *qs*)) = *pos-limit-branch*
using *assms*
proof (*induct qs arbitrary*: *pos-limit-branch neg-limit-branch branch-assms branch-poly-list init-assumps*)
case *Nil*
then have (*branch-assms*, *branch-poly-list*) ∈ *set* [(*init-assumps*, [])]
using *lc-assump-generation-list.simps*
by *auto*
then have *branch-poly-list* = []
using *in-set-member*
by *simp*
then show *?case* **using** *Nil.prems*
unfolding *eval-mpoly-poly-def sgn-pos-inf-rat-list-def*
by *auto*
next
case (*Cons a qs*)
let *?rec* = *lc-assump-generation a init-assumps*
have *inset*: (*branch-assms*,*branch-poly-list*) ∈ *set* (
concat (*map* (
λ(*new-assumps*, *r*). (*let list-rec* = *lc-assump-generation-list qs new-assumps in*
map (λ*elem*. (*fst elem*, *r*#(*snd elem*))) *list-rec*)) *?rec*))
using *Cons.prems lc-assump-generation-list.simps*
by *auto*
then obtain *new-assumps r* **where** *deconstruct-prop*:

(*new-assumps*, *r*) ∈ *set ?rec*
(*branch-assms*,*branch-poly-list*) ∈ *set* (*let list-rec* = *lc-assump-generation-list qs new-assumps in*
map (λ*elem*. (*fst elem*, *r*#(*snd elem*))) *list-rec*)
using *inset*
by (*metis* (*no-types*, *lifting*) *concat-map-in-set nth-mem prod.collapse split-def*)
then obtain *elem list-rec* **where** *list-rec-prop*:
list-rec = *lc-assump-generation-list qs new-assumps*
elem ∈ *set list-rec*
(*branch-assms*,*branch-poly-list*) = (*fst elem*, *r*#(*snd elem*))
by *auto*
then have *snd-elem-inset*: (*branch-assms*, *snd elem*) ∈ *set* (*lc-assump-generation-list qs new-assumps*)
using *deconstruct-prop* **by** *auto*
obtain *pos-limit-sublist neg-limit-sublist* **where** *sublist-prop*:
(*pos-limit-sublist*, *neg-limit-sublist*) = *limit-points-on-branch* (*branch-assms*, *snd elem*)
by *auto*
then have *ind-prop*: *pos-limit-sublist* = *map rat-of-int* (*sgn-pos-inf-rat-list* (*map* (*eval-mpoly-poly val*) *qs*))
using *snd-elem-inset Cons.hyps Cons.prems*
by *blast*
have *sublist-connection*: *pos-limit-branch* = (*if r* = *0 then 0 else Sturm-Tarski.sign* (*lookup-assump* (*Polynomial.lead-coeff r*) *branch-assms*))#*pos-limit-sublist*
using *Cons.prems*(*3*) *sublist-prop list-rec-prop*(*3*)
by *simp*
have *list-prop*: *map* (λ*x*. *rat-of-int* (*Sturm-Tarski.sign* (*sgn-pos-inf x*)))
(*map* (*eval-mpoly-poly val*) (*a*#*qs*)) =
(*rat-of-int* (*Sturm-Tarski.sign* (*sgn-pos-inf* (*eval-mpoly-poly val a*)))) # *map* (λ*x*. *rat-of-int* (*Sturm-Tarski.sign* (*sgn-pos-inf x*)))
(*map* (*eval-mpoly-poly val*) *qs*)
by *simp*
have *tail-prop*: *pos-limit-branch* =
(*if r* = *0 then 0*
else Sturm-Tarski.sign
(*lookup-assump* (*Polynomial.lead-coeff r*) *branch-assms*)) #
map (λ*x*. *rat-of-int* (*Sturm-Tarski.sign* (*sgn-pos-inf x*)))
(*map* (*eval-mpoly-poly val*) *qs*)
using *sublist-connection ind-prop* **unfolding** *sgn-pos-inf-rat-list-def*
by *auto*
have *subs*: *set new-assumps* ⊆ *set branch-assms*
using *deconstruct-prop list-rec-prop snd-elem-inset branch-init-assms-subset*
by *presburger*
then have *r-sign*: *rat-of-int* (*Sturm-Tarski.sign* (*sgn-pos-inf* (*eval-mpoly-poly val a*))) =
(*if r* = *0 then 0 else Sturm-Tarski.sign* (*lookup-assump* (*Polynomial.lead-coeff r*) *new-assumps*))
using *pos-limit-point-on-branch-init deconstruct-prop*(*1*) *local.Cons*(*3*) *subsetD*
by (*smt* (*verit*, *ccfv-threshold*))

have *r-inv*: *r* = (*0*::*rmpoly*) ∨ (∃ *i*. (*lookup-assump-aux* (*Polynomial.lead-coeff r*) *new-assumps* = *Some i* ∧ *i* ≠ *0*))
  using *lc-assump-generation-inv*
  by (*meson deconstruct-prop*(*1*))
have *r-match*: (*if r* = *0 then 0 else Sturm-Tarski.sign* (*lookup-assump* (*Polynomial.lead-coeff r*) *new-assumps*))
= (*if r* = *0 then 0 else Sturm-Tarski.sign* (*lookup-assump* (*Polynomial.lead-coeff r*) *branch-assms*))
proof −
  {**assume** ∗: *r* = *0*
  then have (*if r* = *0 then 0 else Sturm-Tarski.sign* (*lookup-assump* (*Polynomial.lead-coeff r*) *new-assumps*))
= (*if r* = *0 then 0 else Sturm-Tarski.sign* (*lookup-assump* (*Polynomial.lead-coeff r*) *branch-assms*))
    by *auto*
  } **moreover** {**assume** ∗: *r* ≠ *0*
   then obtain *i1* **where** *i1-prop*: *lookup-assump-aux* (*Polynomial.lead-coeff r*) *new-assumps* = *Some i1* ∧ *i1* ≠ *0*
    using *r-inv* **by** *auto*
   then obtain *i2* **where** *i2-prop*: *lookup-assump-aux* (*Polynomial.lead-coeff r*) *branch-assms* = *Some i2* ∧ *i2* ≠ *0*
    using *lookup-assump-aux-subset-consistency subs*
    using *Cons.prems*(*2*) **by** *blast*
    then have *Sturm-Tarski.sign* (*lookup-assump* (*Polynomial.lead-coeff r*) *new-assumps*) =
   Sturm-Tarski.sign (*lookup-assump* (*Polynomial.lead-coeff r*) *branch-assms*)
    using *lookup-assump-aux-subset-consistent-sign subs*
     i1-prop i2-prop Cons.prems(*2*)
   proof −
    obtain *mm* :: *real list* ⇒ (*real mpoly* × *rat*) *list* ⇒ *real mpoly* **and** *rr* :: *real list* ⇒ (*real mpoly* × *rat*) *list* ⇒ *rat* **where**
     ∀ *x4 x5*. (∃ *v6 v7*. (*v6*, *v7*) ∈ *set x5* ∧ ¬ *satisfies-evaluation x4 v6 v7*) = ((*mm x4 x5*, *rr x4 x5*) ∈ *set x5* ∧ ¬ *satisfies-evaluation x4* (*mm x4 x5*) (*rr x4 x5*))
     by *moura*
     then have ∀ *ps rs psa p r ra*. ((*mm rs ps*, *rr rs ps*) ∈ *set ps* ∧ ¬ *satisfies-evaluation rs* (*mm rs ps*) (*rr rs ps*) ∨ ¬ *set psa* ⊆ *set ps* ∨ *lookup-assump-aux* (*Polynomial.lead-coeff p*) *psa* ≠ *Some r* ∨ *lookup-assump-aux* (*Polynomial.lead-coeff p*) *ps* ≠ *Some ra*) ∨ (*Sturm-Tarski.sign r*) = *Sturm-Tarski.sign ra*
     by (*meson lookup-assump-aux-subset-consistent-sign*)
    then have (*Sturm-Tarski.sign i1*) = *Sturm-Tarski.sign i2*
     using *i1-prop i2-prop local.Cons*(*3*) *subs* **by** *blast*
    then show *?thesis*
     by (*simp add*: *i1-prop i2-prop*)
   qed

  then have (*if r* = *0 then 0 else Sturm-Tarski.sign* (*lookup-assump* (*Polynomial.lead-coeff r*) *new-assumps*))
= (*if r* = *0 then 0 else Sturm-Tarski.sign* (*lookup-assump* (*Polynomial.lead-coeff r*) *branch-assms*))

by *auto*
}
ultimately show *?thesis*
by *auto*
qed
then have *rat-of-int* (*Sturm-Tarski.sign* (*sgn-pos-inf* (*eval-mpoly-poly val a*))) =

(*if r* = *0 then 0 else Sturm-Tarski.sign* (*lookup-assump* (*Polynomial.lead-coeff r*) *branch-assms*))
using *r-sign r-match*
by *fastforce*
then have *pos-limit-branch* = (*rat-of-int* (*Sturm-Tarski.sign* (*sgn-pos-inf* (*eval-mpoly-poly val a*)))) #
(*map* (*λx. rat-of-int* (*Sturm-Tarski.sign* (*sgn-pos-inf x*)))
(*map* (*eval-mpoly-poly val*) *qs*))
using *tail-prop*
by (*smt* (*verit*, *ccfv-threshold*) *list.inj-map-strong list.map*(*2*) *of-rat-hom.eq-iff*)

then show *?case* **using** *list-prop*
using *sgn-pos-inf-rat-list-def*
by (*auto*)
qed

lemma *neg-limit-point-on-branch-init*:
assumes (*a*, *q*) ∈ *set* (*lc-assump-generation init-q init-assumps*)
assumes ⋀*p n*. (*p*,*n*) ∈ *set a* ⟹ *satisfies-evaluation val p n*
shows *rat-of-int* (*Sturm-Tarski.sign* (*sgn-neg-inf* (*eval-mpoly-poly val init-q*))) =

(*if q* = *0 then 0 else* (*Sturm-Tarski.sign* (*lookup-assump* (*Polynomial.lead-coeff q*) *a*))∗(−*1*)^(*Polynomial.degree q*))
proof −
have *at-inf*: *rat-of-int*
(*Sturm-Tarski.sign* (*sgn-class.sgn* (*Polynomial.lead-coeff* (*eval-mpoly-poly val init-q*)))) =
(*if q* = *0 then 0*
else Sturm-Tarski.sign (*lookup-assump* (*Polynomial.lead-coeff q*) *a*))
using *assms pos-limit-point-on-branch-init*
unfolding *sgn-pos-inf-def*
by *auto*
have *Polynomial.degree q* = *Polynomial.degree* (*eval-mpoly-poly val init-q*)
using *check-constant-degree-match assms* **by** *auto*
then show *?thesis* **using** *at-inf*
unfolding *sgn-neg-inf-def*
using *Sturm-Tarski-casting degree-0 even-zero more-arith-simps*(*6*) *more-arith-simps*(*8*) *ring-1-class.minus-one-power-iff* **by** *auto*

qed

lemma *neg-limit-point-on-branch-list*:

assumes (*branch-assms*, *branch-poly-list*) ∈ *set* (*lc-assump-generation-list qs init-assumps*)
assumes ⋀*p n*. (*p*,*n*) ∈ *set branch-assms* ⟹ *satisfies-evaluation val p n*
assumes (*pos-limit-branch*, *neg-limit-branch*) = *limit-points-on-branch* (*branch-assms*, *branch-poly-list*)
shows *map rat-of-int* (*sgn-neg-inf-rat-list* (*map* (*eval-mpoly-poly val*) *qs*)) = *neg-limit-branch*
using *assms*
proof (*induct qs arbitrary*: *pos-limit-branch neg-limit-branch branch-assms branch-poly-list init-assumps*)
case *Nil*
then have (*branch-assms*, *branch-poly-list*) ∈ *set* [(*init-assumps*, [])]
using *lc-assump-generation-list.simps*
by *auto*
then have *branch-poly-list* = []
using *in-set-member*
by *simp*
then show *?case* **using** *Nil.prems*
unfolding *eval-mpoly-poly-def sgn-neg-inf-rat-list-def*
by *auto*
next
case (*Cons a qs*)
let *?rec* = *lc-assump-generation a init-assumps*
have *inset*: (*branch-assms*,*branch-poly-list*) ∈ *set* (
concat (*map* (
λ(*new-assumps*, *r*). (*let list-rec* = *lc-assump-generation-list qs new-assumps in*
map (λ*elem*. (*fst elem*, *r*#(*snd elem*))) *list-rec*)) *?rec*))
using *Cons.prems lc-assump-generation-list.simps*
by *auto*
then obtain *new-assumps r* **where** *deconstruct-prop*:
(*new-assumps*, *r*) ∈ *set ?rec*
(*branch-assms*,*branch-poly-list*) ∈ *set* (*let list-rec* = *lc-assump-generation-list qs new-assumps in*
map (λ*elem*. (*fst elem*, *r*#(*snd elem*))) *list-rec*)
using *inset*
by (*metis* (*no-types*, *lifting*) *concat-map-in-set nth-mem prod.collapse split-def*)
then obtain *elem list-rec* **where** *list-rec-prop*:
list-rec = *lc-assump-generation-list qs new-assumps*
elem ∈ *set list-rec*
(*branch-assms*,*branch-poly-list*) = (*fst elem*, *r*#(*snd elem*))
by *auto*
then have *snd-elem-inset*: (*branch-assms*, *snd elem*) ∈ *set* (*lc-assump-generation-list qs new-assumps*)
using *deconstruct-prop* **by** *auto*
obtain *pos-limit-sublist neg-limit-sublist* **where** *sublist-prop*:
(*pos-limit-sublist*, *neg-limit-sublist*) = *limit-points-on-branch* (*branch-assms*, *snd elem*)
by *auto*
then have *ind-prop-var*: *neg-limit-sublist* = *map rat-of-int* (*sgn-neg-inf-rat-list*

(*map* (*eval-mpoly-poly val*) *qs*))
  using *snd-elem-inset Cons.hyps Cons.prems* **by** *blast*
then have *ind-prop*: *neg-limit-sublist* = *sgn-neg-inf-rat-list* (*map* (*eval-mpoly-poly val*) *qs*)
  by *auto*
have *sublist-connection*: *pos-limit-branch* = (*if r* = *0 then 0 else Sturm-Tarski.sign* (*lookup-assump* (*Polynomial.lead-coeff r*) *branch-assms*))#*pos-limit-sublist*
  using *Cons.prems*(*3*) *sublist-prop list-rec-prop*(*3*)
  by *simp*
then have *sublist-connection-var*: *pos-limit-branch* = (*if r* = *0 then 0 else* (*rat-of-int* ∘ *Sturm-Tarski.sign*) (*lookup-assump* (*Polynomial.lead-coeff r*) *branch-assms*))#*pos-limit-sublist*
  using *Cons.prems*(*3*) *sublist-prop list-rec-prop*(*3*)
  by *simp*
have *list-prop*: *map* (λ*x*. *rat-of-int* (*Sturm-Tarski.sign* (*sgn-neg-inf x*)))
  (*map* (*eval-mpoly-poly val*) (*a*#*qs*)) =
(*rat-of-int* (*Sturm-Tarski.sign* (*sgn-neg-inf* (*eval-mpoly-poly val a*)))) # *map* (λ*x*. *rat-of-int* (*Sturm-Tarski.sign* (*sgn-neg-inf x*)))
  (*map* (*eval-mpoly-poly val*) *qs*)
  by *simp*
have *ind-prop-var2*: *neg-limit-sublist* =
  (*map* (λ*x*. (*rat-of-int* ∘ *Sturm-Tarski.sign*) (*sgn-neg-inf x*))
  (*map* (*eval-mpoly-poly val*) *qs*))
  using *ind-prop-var* **unfolding** *sgn-neg-inf-rat-list-def*
  by *auto*
have *neg-limit-branch* =
  (*if r* = *0 then* (*0*::*rat*)
  else ((*rat-of-int* ∘ *Sturm-Tarski.sign*)
  (*lookup-assump* (*Polynomial.lead-coeff r*) *branch-assms*)∗(−*1*)^(*Polynomial.degree r*))) #
  map (λ*x*. ((*rat-of-int* ∘ *Sturm-Tarski.sign*) (*sgn-neg-inf x*)))
  (*map* (*eval-mpoly-poly val*) *qs*)
  using *sublist-connection-var ind-prop-var2 local.Cons*(*4*) **unfolding** *sgn-neg-inf-rat-list-def*

  unfolding *limit-points-on-branch.simps*
  by (*metis* (*no-types*, *lifting*) *limit-points-on-branch.simps list.simps*(*9*) *list-rec-prop*(*3*) *prod.simps*(*1*) *sublist-prop*)
then have *tail-prop-helper*: *neg-limit-branch* =
  (*if r* = *0 then 0*
  else rat-of-int (*Sturm-Tarski.sign*
  (*lookup-assump* (*Polynomial.lead-coeff r*) *branch-assms*)∗(−*1*)^(*Polynomial.degree r*))) #
  map (λ*x*. *rat-of-int* (*Sturm-Tarski.sign* (*sgn-neg-inf x*)))
  (*map* (*eval-mpoly-poly val*) *qs*)
  by *auto*
then have *tail-prop*: *neg-limit-branch* =
  (*if r* = *0 then 0*
  else Sturm-Tarski.sign
  (*lookup-assump* (*Polynomial.lead-coeff r*) *branch-assms*)∗(−*1*)^(*Polynomial.degree r*)) #

map (λ*x*. *rat-of-int* (*Sturm-Tarski.sign* (*sgn-neg-inf x*)))
(*map* (*eval-mpoly-poly val*) *qs*)
by (*smt* (*verit*, *ccfv-threshold*) *list.map*(*2*) *of-int-hom.hom-zero of-rat-of-int-eq*)
have *subs*: *set new-assumps* ⊆ *set branch-assms*
using *deconstruct-prop list-rec-prop snd-elem-inset branch-init-assms-subset*
by *presburger*
then have *r-sign*: *rat-of-int* (*Sturm-Tarski.sign* (*sgn-neg-inf* (*eval-mpoly-poly val a*))) =
(*if r* = *0 then 0 else Sturm-Tarski.sign* (*lookup-assump* (*Polynomial.lead-coeff r*) *new-assumps*)∗(−*1*)^(*Polynomial.degree r*))
using *neg-limit-point-on-branch-init deconstruct-prop*(*1*) *local.Cons*(*3*) *subsetD*
by (*smt* (*verit*, *ccfv-threshold*))
have *r-inv*: *r* = (*0*::*rmpoly*) ∨ (∃ *i*. (*lookup-assump-aux* (*Polynomial.lead-coeff r*) *new-assumps* = *Some i* ∧ *i* ≠ *0*))
using *lc-assump-generation-inv*
by (*meson deconstruct-prop*(*1*))
have *r-match-pre*: (*if r* = *0 then 0 else Sturm-Tarski.sign* (*lookup-assump* (*Polynomial.lead-coeff r*) *new-assumps*))
= (*if r* = *0 then 0 else Sturm-Tarski.sign* (*lookup-assump* (*Polynomial.lead-coeff r*) *branch-assms*))
proof −
{**assume** ∗: *r* = *0*
then have (*if r* = *0 then 0 else Sturm-Tarski.sign* (*lookup-assump* (*Polynomial.lead-coeff r*) *new-assumps*))
= (*if r* = *0 then 0 else Sturm-Tarski.sign* (*lookup-assump* (*Polynomial.lead-coeff r*) *branch-assms*))
by *auto*
} **moreover** {**assume** ∗: *r* ≠ *0*
then obtain *i1* **where** *i1-prop*: *lookup-assump-aux* (*Polynomial.lead-coeff r*) *new-assumps* = *Some i1* ∧ *i1* ≠ *0*
using *r-inv* **by** *auto*
then obtain *i2* **where** *i2-prop*: *lookup-assump-aux* (*Polynomial.lead-coeff r*) *branch-assms* = *Some i2* ∧ *i2* ≠ *0*
using *lookup-assump-aux-subset-consistency subs*
using *Cons.prems*(*2*) **by** *blast*
then have *Sturm-Tarski.sign* (*lookup-assump* (*Polynomial.lead-coeff r*) *new-assumps*) =
Sturm-Tarski.sign (*lookup-assump* (*Polynomial.lead-coeff r*) *branch-assms*)
using *lookup-assump-aux-subset-consistent-sign subs*
i1-prop i2-prop Cons.prems(*2*)
proof −
obtain *mm* :: *real list* ⇒ (*real mpoly* × *rat*) *list* ⇒ *real mpoly* **and** *rr* :: *real list* ⇒ (*real mpoly* × *rat*) *list* ⇒ *rat* **where**
∀ *x4 x5*. (∃ *v6 v7*. (*v6*, *v7*) ∈ *set x5* ∧ ¬ *satisfies-evaluation x4 v6 v7*) = ((*mm x4 x5*, *rr x4 x5*) ∈ *set x5* ∧ ¬ *satisfies-evaluation x4* (*mm x4 x5*) (*rr x4 x5*))
by *moura*
then have ∀ *ps rs psa p r ra*. ((*mm rs ps*, *rr rs ps*) ∈ *set ps* ∧ ¬ *satisfies-evaluation rs* (*mm rs ps*) (*rr rs ps*) ∨ ¬ *set psa* ⊆ *set ps* ∨ *lookup-assump-aux* (*Polynomial.lead-coeff p*) *psa* ≠ *Some r* ∨ *lookup-assump-aux* (*Polynomial.lead-coeff*

p) *ps* ≠ *Some ra*) ∨ (*Sturm-Tarski.sign r*) = *Sturm-Tarski.sign ra*
by (*meson lookup-assump-aux-subset-consistent-sign*)
then have (*Sturm-Tarski.sign i1*) = *Sturm-Tarski.sign i2*
using *i1-prop i2-prop local.Cons*(*3*) *subs* **by** *blast*
then show *?thesis*
by (*simp add*: *i1-prop i2-prop*)
qed

then have (*if r* = *0 then 0 else Sturm-Tarski.sign* (*lookup-assump* (*Polynomial.lead-coeff r*) *new-assumps*))
= (*if r* = *0 then 0 else Sturm-Tarski.sign* (*lookup-assump* (*Polynomial.lead-coeff r*) *branch-assms*))
by *auto*
}
ultimately show *?thesis*
by *auto*
qed
then have *r-match*: (*if r* = *0 then 0 else Sturm-Tarski.sign* (*lookup-assump* (*Polynomial.lead-coeff r*) *new-assumps*)∗(−*1*) ^(*Polynomial.degree r*))
= (*if r* = *0 then 0 else Sturm-Tarski.sign* (*lookup-assump* (*Polynomial.lead-coeff r*) *branch-assms*)∗(−*1*) ^(*Polynomial.degree r*))
by *metis*
then have *rat-of-int* (*Sturm-Tarski.sign* (*sgn-neg-inf* (*eval-mpoly-poly val a*)))
=
(*if r* = *0 then 0 else Sturm-Tarski.sign* (*lookup-assump* (*Polynomial.lead-coeff r*) *branch-assms*)∗(−*1*) ^(*Polynomial.degree r*))
using *r-sign r-match*
by *fastforce*
then have *neg-limit-branch* = (*rat-of-int* (*Sturm-Tarski.sign* (*sgn-neg-inf* (*eval-mpoly-poly val a*)))) #
(*map* (λ*x*. *rat-of-int* (*Sturm-Tarski.sign* (*sgn-neg-inf x*)))
(*map* (*eval-mpoly-poly val*) *qs*))
using *tail-prop tail-prop-helper*
by (*simp add*: *of-rat-hom.injectivity*)
then show *?case* **using** *list-prop*
using *sgn-neg-inf-rat-list-def*
by *auto*
qed

lemma *complex-rat-casting-lemma*:
fixes *a*:: *int list*
fixes *b*:: *rat list*
shows *map complex-of-int a* = *map of-rat b* ⟹ *map rat-of-int a* = *b*
by (*metis* (*no-types*, *lifting*) *list.inj-map-strong list.map-comp of-rat-hom.injectivity of-rat-of-int*)

lemma *complex-rat-casting-lemma-sets*:
fixes *a*:: *rat list list*
fixes *b1*:: *int list*

fixes *b2*:: *int list*
fixes *c*:: *rat list list*
assumes *set* (*map* (*map of-rat*) *a*) ∪ {*map complex-of-int b1*, *map complex-of-int b2*}
= *set* (*map* (*map of-rat*) *c*)
shows *set a* ∪ {*map rat-of-int b1*, *map rat-of-int b2*} = *set c*
proof −
have *map complex-of-int b1* ∈ *set* (*map* (*map of-rat*) *c*)
using *assms* **by** *auto*
then have *h1*: *map rat-of-int b1* ∈ *set c*
using *complex-rat-casting-lemma* **by** *auto*
have *map complex-of-int b2* ∈ *set* (*map* (*map of-rat*) *c*)
using *assms* **by** *auto*
then have *h2*: *map rat-of-int b2* ∈ *set c*
using *complex-rat-casting-lemma* **by** *auto*
have *set-lem*: ⋀ *a b c* . *a* ∪ *b* = *c* ⟹ *a* ⊆ *c*
by *blast*
have *set* (*map* (*map of-rat*) *a*) ⊆ *set* (*map* (*map of-rat*) *c*)
using *assms set-lem*[*of set* (*map* (*map of-rat*) *a*) {*map complex-of-int b1*, *map complex-of-int b2*} *set* (*map* (*map of-rat*) *c*)]
by *auto*
then have ⋀*x*. *x* ∈ *set* (*map* (*map of-rat*) *a*) ⟹ *x* ∈ *set* (*map* (*map of-rat*) *c*)
by *auto*
then have *h3*: ⋀*x*. *x* ∈ *set a* ⟹ *x* ∈ *set c*
using *complex-rat-casting-lemma*
by *fastforce*
have *h4-a*: ⋀*x*. *x* ∈ *set* (*map* (*map of-rat*) *c*) ⟹
x ≠ *map complex-of-int b1* ⟹ *x* ∉ *set* (*map* (*map of-rat*) *a*) ⟹ *x* = *map complex-of-int b2*
using *assms*
by *blast*
have *h4*: ⋀*x*. *x* ∈ *set c* ⟹
x ≠ *map rat-of-int b1* ⟹ *x* ∉ *set a* ⟹ *x* = *map rat-of-int b2*
proof −
fix *x*
assume *a1*: *x* ∈ *set c*
assume *a2*: *x* ≠ *map rat-of-int b1*
assume *a3*: *x* ∉ *set a*
have ((*map of-rat x*)::*complex list*) ∈ *set* (*map* (*map of-rat*) *c*)
using *a1* **by** *auto*
then have *impl*: ((*map of-rat x*)::*complex list*) ≠ *map complex-of-int b1* ⟹ (*map of-rat x*) ∉ *set* (*map* (*map of-rat*) *a*) ⟹ (*map of-rat x*) = *map complex-of-int b2*
using *h4-a*[*of* ((*map of-rat x*)::*complex list*)]
using *a3 imageE inj-map-eq-map list.set-map of-rat-hom.inj-f* **by** *fastforce*
have *impl-h1*: ((*map of-rat x*)::*complex list*) ≠ *map complex-of-int b1*
using *a2 complex-rat-casting-lemma*
by *metis*
have *impl-h2*: (*map of-rat x*) ∉ *set* (*map* (*map of-rat*) *a*)

using *a3 complex-rat-casting-lemma* **by** (*auto*)
then have (*map of-rat x*) = *map complex-of-int b2*
using *impl impl-h1 impl-h2* **by** *auto*
then show *x* = *map rat-of-int b2*
using *complex-rat-casting-lemma* **by** *metis*
qed
show *?thesis* **using** *h1 h2 h3 h4* **by** *auto*
qed

lemma *complex-rat-casting-lemma-sets2*:
shows {*map rat-of-int*
(*map* (λ*x*. *Sturm-Tarski.sign* (*sgn-neg-inf x*))
(*map* (*eval-mpoly-poly val*) *qs*)),
map rat-of-int
(*map* (λ*x*. *Sturm-Tarski.sign* (*sgn-pos-inf x*))
(*map* (*eval-mpoly-poly val*) *qs*))} = {*map* (λ*x*. (*rat-of-int* ∘ *Sturm-Tarski.sign*) (*sgn-neg-inf x*))
(*map* (*eval-mpoly-poly val*) *qs*),
map (λ*x*. (*rat-of-int* ∘ *Sturm-Tarski.sign*) (*sgn-pos-inf x*))
(*map* (*eval-mpoly-poly val*) *qs*)}
proof −
have *h1*: *map rat-of-int*
(*map* (λ*x*. *Sturm-Tarski.sign* (*sgn-neg-inf x*))
(*map* (*eval-mpoly-poly val*) *qs*)) = *map* (λ*x*. (*rat-of-int* ∘ *Sturm-Tarski.sign*) (*sgn-neg-inf x*))
(*map* (*eval-mpoly-poly val*) *qs*)
by *auto*
have *h2*: *map rat-of-int*
(*map* (λ*x*. *Sturm-Tarski.sign* (*sgn-pos-inf x*))
(*map* (*eval-mpoly-poly val*) *qs*)) = *map* (λ*x*. (*rat-of-int* ∘ *Sturm-Tarski.sign*) (*sgn-pos-inf x*))
(*map* (*eval-mpoly-poly val*) *qs*)
by *auto*
show *?thesis* **using** *h1 h2*
by *auto*
qed

lemma *sign-determination-inner-gives-noncomp-signs-at-roots*:
assumes (*assumps*, *signs*) ∈ *set* (*sign-determination-inner qs init-assumps*)
assumes ⋀*p n*. (*p*,*n*) ∈ *set assumps* ⟹ *satisfies-evaluation val p n*
shows *set signs* = (*consistent-sign-vectors-R* (*map* (*eval-mpoly-poly val*) *qs*) *UNIV*)
proof −
have *elem-in-set*: (*assumps*, *signs*) ∈ *set* (*let branches* = *lc-assump-generation-list qs init-assumps in*
concat (*map* (λ*branch*.
let poly-p-branch = *poly-p-in-branch branch*;
(*pos-limit-branch*, *neg-limit-branch*) = *limit-points-on-branch branch*;
calculate-data-branch = *extract-signs* (*calculate-data-assumps-M poly-p-branch*

(*snd branch*) (*fst branch*))
in map (λ(*a*, *signs*). (*a*, *pos-limit-branch*#*neg-limit-branch*#*signs*)) *calculate-data-branch*
) *branches*))
using *assms*
by *auto*
obtain *branch* **where** *branch-prop*: *branch* ∈ *set* (*lc-assump-generation-list qs init-assumps*)
(*assumps*, *signs*) ∈ *set* (
let poly-p-branch = *poly-p-in-branch branch*;
(*pos-limit-branch*, *neg-limit-branch*) = *limit-points-on-branch branch*;
calculate-data-branch = *extract-signs* (*calculate-data-assumps-M poly-p-branch* (*snd branch*) (*fst branch*))
in map (λ(*a*, *signs*). (*a*, *pos-limit-branch*#*neg-limit-branch*#*signs*)) *calculate-data-branch*)
using *elem-in-set*
by *auto*
then obtain *branch-assms branch-poly-list* **where** *branch-is*:
branch = (*branch-assms*, *branch-poly-list*)
using *poly-p-in-branch.cases* **by** *blast*
then obtain *calculate-data-branch poly-p-branch pos-limit-branch neg-limit-branch*
where *branch-prop-expanded*:
poly-p-branch = *poly-p-in-branch branch*
(*pos-limit-branch*, *neg-limit-branch*) = *limit-points-on-branch branch*
calculate-data-branch = *extract-signs* (*calculate-data-assumps-M poly-p-branch branch-poly-list branch-assms*)
branch ∈ *set* (*lc-assump-generation-list qs init-assumps*)
(*assumps*, *signs*) ∈ *set* (
map (λ(*a*, *signs*). (*a*, *pos-limit-branch*#*neg-limit-branch*#*signs*)) *calculate-data-branch*)
using *branch-prop*
by (*metis* (*no-types*, *lifting*) *case-prod-beta fst-conv prod.exhaust-sel snd-conv*)
obtain *calc-a calc-signs* **where** *calc-prop*:
(*calc-a*, *calc-signs*) ∈ *set calculate-data-branch*
(*assumps*, *signs*) = (*calc-a*, *pos-limit-branch*#*neg-limit-branch*#*calc-signs*)
using *in-set-member branch-prop-expanded*(*5*)
by *auto*
then have *branch-assms-subset*: *set branch-assms* ⊆ *set assumps*
using *branch-prop-expanded*(*3*) *extract-signs-M-subset*
by *blast*
have *set init-assumps* ⊆ *set branch-assms*
using *branch-is branch-prop*(*1*) *branch-init-assms-subset*
by *blast*
have *assumps-calc-a*: *assumps* = *calc-a*
using *calc-prop* **by** *auto*
let *?poly-p-branch* = *poly-p-in-branch* (*branch-assms*, *branch-poly-list*)
have *nonz-poly-p*: *eval-mpoly-poly val ?poly-p-branch* ≠ *0*
using *poly-p-nonzero-on-branch*
using ‹*set branch-assms* ⊆ *set assumps*› *assms*(*2*) *branch-is branch-prop*(*1*) **by** *blast*

have *map-branch-poly-list*: (*map* (*eval-mpoly-poly val*) *qs*) = (*map* (*eval-mpoly-poly val*) *branch-poly-list*)
using *map-branch-poly-list*
using *assms*(*2*) *branch-assms-subset branch-is branch-prop*(*1*) **by** *blast*
have (*calc-a*, *calc-signs*)
∈ *set* (*calculate-data-to-signs* (*calculate-data-assumps-M poly-p-branch branch-poly-list branch-assms*))
using *calc-prop branch-prop-expanded calc-data-to-signs-and-extract-signs*
by *metis*
then have (*assumps*, *calc-signs*)
∈ *set* (*calculate-data-to-signs* (*calculate-data-assumps-M poly-p-branch branch-poly-list branch-assms*))
using *assumps-calc-a* **by** *auto*
then have *set calc-signs* = *set*(*characterize-consistent-signs-at-roots* (*eval-mpoly-poly val ?poly-p-branch*) (*map* (*eval-mpoly-poly val*) *branch-poly-list*))
using *nonz-poly-p calculate-data-assumps-gives-noncomp-signs-at-roots*[*of assumps signs poly-p-branch branch-poly-list branch-assms val*]
using *assms*(*2*) *branch-is branch-prop-expanded*(*1*) *calculate-data-assumps-gives-noncomp-signs-at-roots* **by** *blast*
then have *calc-signs-is*: *set calc-signs* = *set*(*characterize-consistent-signs-at-roots* (*eval-mpoly-poly val ?poly-p-branch*) (*map* (*eval-mpoly-poly val*) *qs*))
using *map-branch-poly-list* **by** *auto*
have *poly-p-is*: *poly-p-in-branch* (*branch-assms*, *branch-poly-list*) = (*if* (*check-all-const-deg-gen branch-poly-list* = *True*) *then* [:*0*, *1*:] *else*
(*pderiv* (*prod-list-var-gen branch-poly-list*)) ∗ (*prod-list-var-gen branch-poly-list*))
by *auto*
have *const-match*: (*check-all-const-deg-gen branch-poly-list*) = (*check-all-const-deg-gen* (*map* (*eval-mpoly-poly val*) *qs*))
using *check-constant-degree-match-list*
using *assms*(*2*) *branch-assms-subset branch-is branch-prop*(*1*) **by** *blast*
have *eval-mpoly-poly val* (*prod-list-var-gen branch-poly-list*) =
prod-list-var-gen (*map* (*eval-mpoly-poly val*) *branch-poly-list*)
using *eval-prod-list-var-gen-match*
using *assms*(*2*) *branch-assms-subset branch-is branch-prop*(*1*) **by** *blast*
then have *same-prod*: *eval-mpoly-poly val* ((*pderiv* (*prod-list-var-gen branch-poly-list*)) ∗ (*prod-list-var-gen branch-poly-list*))
= *pderiv* (*prod-list-var-gen* (*map* (*eval-mpoly-poly val*) *branch-poly-list*)) ∗ (*prod-list-var-gen* (*map* (*eval-mpoly-poly val*) *branch-poly-list*))
by (*metis eval-mpoly-poly-comm-ring-hom.hom-mult pderiv-commutes*)
{assume ∗: *check-all-const-deg-gen branch-poly-list* = *True*
then have *?poly-p-branch* = [: *0*, *1* :]
using *poly-p-is* **by** *presburger*
then have (*eval-mpoly-poly val ?poly-p-branch*) = [:*0*, *1*:]
unfolding *eval-mpoly-poly-def*
by *auto*
then have (*eval-mpoly-poly val ?poly-p-branch*) = (*poly-f-nocrb* (*map* (*eval-mpoly-poly val*) *qs*))
unfolding *poly-f-nocrb-def* **using** *const-match check-all-const-deg-match* ∗
by *presburger*

}
moreover {**assume** ∗: *check-all-const-deg-gen branch-poly-list* = *False*
then have (*eval-mpoly-poly val ?poly-p-branch*) = *pderiv* (*prod-list-var-gen* (*map* (*eval-mpoly-poly val*) *branch-poly-list*)) ∗ (*prod-list-var-gen* (*map* (*eval-mpoly-poly val*) *branch-poly-list*))
using *poly-p-is same-prod* **by** *auto*
then have *h1*: (*eval-mpoly-poly val ?poly-p-branch*) = *pderiv* (*prod-list-var-gen* (*map* (*eval-mpoly-poly val*) *qs*)) ∗ (*prod-list-var-gen* (*map* (*eval-mpoly-poly val*) *qs*))
using *map-branch-poly-list*
by *presburger*
have (*check-all-const-deg* (*map* (*eval-mpoly-poly val*) *qs*) = *False*)
using ∗ *const-match check-all-const-deg-match* **by** *auto*
then have (*eval-mpoly-poly val ?poly-p-branch*) = (*poly-f-nocrb* (*map* (*eval-mpoly-poly val*) *qs*))
using *h1 prod-list-var-match*
unfolding *poly-f-nocrb-def*
by *metis*
}
ultimately have *poly-branch-no-crb-connect*: (*eval-mpoly-poly val ?poly-p-branch*) = (*poly-f-nocrb* (*map* (*eval-mpoly-poly val*) *qs*))
by *auto*
let *?eval-qs* = (*map* (*eval-mpoly-poly val*) *qs*)
have *set calc-signs* =
set (*characterize-consistent-signs-at-roots* (*poly-f-nocrb ?eval-qs*) *?eval-qs*)
using *poly-branch-no-crb-connect calc-signs-is*
by *presburger*

then have *calc-signs-relation*: (*set calc-signs*) ∪ {*sgn-neg-inf-rat-list ?eval-qs*, *sgn-pos-inf-rat-list ?eval-qs*} =
set (*characterize-consistent-signs-at-roots* (*poly-f ?eval-qs*) *?eval-qs*)
using *poly-f-ncrb-connection*[*of ?eval-qs*]
by *auto*

then have *set calc-signs* ∪
{*map rat-of-int*
(*sgn-neg-inf-rat-list* (*map* (*eval-mpoly-poly val*) *qs*)),
map rat-of-int
(*sgn-pos-inf-rat-list* (*map* (*eval-mpoly-poly val*) *qs*))} =
set (*characterize-consistent-signs-at-roots*
(*poly-f* (*map* (*eval-mpoly-poly val*) *qs*))
(*map* (*eval-mpoly-poly val*) *qs*))
unfolding *sgn-neg-inf-rat-list2-def sgn-pos-inf-rat-list2-def*
using *complex-rat-casting-lemma-sets*[*of calc-signs* (*sgn-neg-inf-rat-list* (*map* (*eval-mpoly-poly val*) *qs*)) (*sgn-pos-inf-rat-list* (*map* (*eval-mpoly-poly val*) *qs*))
(*characterize-consistent-signs-at-roots* (*poly-f* (*map* (*eval-mpoly-poly val*) *qs*)) (*map* (*eval-mpoly-poly val*) *qs*))]
by (*auto*)
then have *calc-signs-relation-var*: (*set calc-signs*) ∪ ({*sgn-neg-inf-rat-list2 ?eval-qs*, *sgn-pos-inf-rat-list2 ?eval-qs*}::*rat list set*) =

set (*characterize-consistent-signs-at-roots* (*poly-f ?eval-qs*) *?eval-qs*)
unfolding *sgn-neg-inf-rat-list2-def sgn-pos-inf-rat-list2-def sgn-neg-inf-rat-list-def sgn-pos-inf-rat-list-def*
using *complex-rat-casting-lemma-sets2*
by *auto*
have *pos-inf*: *sgn-pos-inf-rat-list ?eval-qs* = *pos-limit-branch*
using *branch-prop-expanded*(*2*) *pos-limit-point-on-branch-list*
using *assms*(*2*) *branch-assms-subset branch-is branch-prop*(*1*)
by (*smt* (*verit*, *ccfv-threshold*) *basic-trans-rules*(*31*) *list.map-comp of-rat-of-int*)
then have *pos-inf-var*: *sgn-pos-inf-rat-list2 ?eval-qs* = *pos-limit-branch*
unfolding *sgn-pos-inf-rat-list2-def*
using *complex-rat-casting-lemma*[*of* (*sgn-pos-inf-rat-list* (*map* (*eval-mpoly-poly val*) *qs*)) *pos-limit-branch*]
unfolding *sgn-pos-inf-rat-list-def* **by** *auto*
have *neg-inf*: *sgn-neg-inf-rat-list ?eval-qs* = *neg-limit-branch*
using *branch-prop-expanded*(*2*) *neg-limit-point-on-branch-init*
using *assms*(*2*) *branch-assms-subset branch-is branch-prop*(*1*)
by (*smt* (*verit*, *ccfv-SIG*) *basic-trans-rules*(*31*) *list.map-comp neg-limit-point-on-branch-list of-rat-of-int*)
then have *neg-inf-var*: *sgn-neg-inf-rat-list2 ?eval-qs* = *neg-limit-branch*
unfolding *sgn-neg-inf-rat-list2-def*
using *complex-rat-casting-lemma*[*of* (*sgn-neg-inf-rat-list* (*map* (*eval-mpoly-poly val*) *qs*)) *neg-limit-branch*]
unfolding *sgn-neg-inf-rat-list-def*
by *auto*
have *set signs* = *set* (*characterize-consistent-signs-at-roots* (*poly-f ?eval-qs*) *?eval-qs*)
using *calc-signs-relation-var pos-inf-var neg-inf-var calc-prop*(*2*)
unfolding *sgn-neg-inf-rat-list2-def sgn-pos-inf-rat-list2-def*
by *auto*
then show *set signs* = (*consistent-sign-vectors-R* (*map* (*eval-mpoly-poly val*) *qs*) *UNIV*)
using *find-consistent-signs-at-roots-R poly-f-ncrb-connection calc-signs-is main-step-R*
by (*metis find-consistent-signs-R-def poly-f-nonzero*)
qed

17.10 Completeness

lemma *lc-assump-generation-valuation*:
assumes $\bigwedge p\ n.\ (p,n) \in set\ init\text{-}assumps \Longrightarrow satisfies\text{-}evaluation\ val\ p\ n$
shows $\exists\, branch \in set\ (lc\text{-}assump\text{-}generation\ q\ init\text{-}assumps).$
$set\ (fst\ branch) \subseteq set\ (init\text{-}assumps) \cup set$
$(map\ (\lambda x.\ (x,\ mpoly\text{-}sign\ val\ x))\ (Polynomial.coeffs\ q))$
using *assms*
proof (*induct q init-assumps rule*: *lc-assump-generation-induct*)
case (*Base q assumps*)
then show *?case*
using *lc-assump-generation.simps* **by** *auto*
next

case (*Rec q assumps*)
let *?zero* = *lc-assump-generation* (*one-less-degree q*) ((*Polynomial.lead-coeff q*, (*0*::*rat*)) # *assumps*)
let *?one* = ((*Polynomial.lead-coeff q*, (*1*::*rat*)) # *assumps*, *q*)
let *?minus-one* = ((*Polynomial.lead-coeff q*, (−*1*::*rat*)) # *assumps*, *q*)
have *set-is*: *set* (*lc-assump-generation q assumps*) = *set* (*?one*#*?minus-one*#*?zero*)
using *Rec.hyps Rec*(*4*) *lc-assump-generation.simps* **by** *auto*
let *?lc* = (*Polynomial.lead-coeff q*)
have *eo*: *satisfies-evaluation val ?lc* (*0*::*rat*) ∨ *satisfies-evaluation val ?lc* (*1*::*rat*) ∨ *satisfies-evaluation val ?lc* (−*1*::*rat*)
unfolding *satisfies-evaluation-def eval-mpoly-def Sturm-Tarski.sign-def* **by** *auto*
{**assume** ∗: *satisfies-evaluation val ?lc* (*1*::*rat*)
then have *1* = *mpoly-sign val* (*Polynomial.lead-coeff q*)
unfolding *satisfies-evaluation-def eval-mpoly-def mpoly-sign-def Sturm-Tarski.sign-def*
by *simp*
then have (*Polynomial.lead-coeff q*, *1*) ∈ *set* (*map* (*λx*. (*x*, *mpoly-sign val x*)) (*Polynomial.coeffs q*))
by (*simp add*: *Rec*(*1*) *coeff-in-coeffs*)
then have *set* (*fst ?one*) ⊆ *set assumps* ∪ *set* (*map* (*λx*. (*x*, *mpoly-sign val x*)) (*Polynomial.coeffs q*))
by *auto*
then have ∃ *branch*∈*set* (*lc-assump-generation q assumps*).
set (*fst branch*) ⊆ *set assumps* ∪ *set* (*map* (*λx*. (*x*, *mpoly-sign val x*)) (*Polynomial.coeffs q*))
using *set-is* **by** *auto*
} **moreover**
{**assume** ∗: *satisfies-evaluation val ?lc* (−*1*::*rat*)
then have −*1* = *mpoly-sign val* (*Polynomial.lead-coeff q*)
unfolding *satisfies-evaluation-def eval-mpoly-def mpoly-sign-def Sturm-Tarski.sign-def*
by (*smt* (*z3*) *of-int-1 of-int-minus rel-simps*(*66*) *zero-neq-neg-one*)
then have (*Polynomial.lead-coeff q*, −*1*) ∈ *set* (*map* (*λx*. (*x*, *mpoly-sign val x*)) (*Polynomial.coeffs q*))
by (*simp add*: *Rec*(*1*) *coeff-in-coeffs*)
then have *set* (*fst ?minus-one*) ⊆ *set assumps* ∪ *set* (*map* (*λx*. (*x*, *mpoly-sign val x*)) (*Polynomial.coeffs q*))
by *auto*
then have ∃ *branch*∈*set* (*lc-assump-generation q assumps*).
set (*fst branch*) ⊆ *set assumps* ∪ *set* (*map* (*λx*. (*x*, *mpoly-sign val x*)) (*Polynomial.coeffs q*))
using *set-is* **by** *auto*
} **moreover** {**assume** ∗: *satisfies-evaluation val ?lc* (*0*::*rat*)
then have ∃ *branch*∈*set* (*lc-assump-generation* (*Multiv-Poly-Props.one-less-degree q*) ((*Polynomial.lead-coeff q*, *0*) # *assumps*)).
set (*fst branch*)
⊆ *set* ((*Polynomial.lead-coeff q*, *0*) # *assumps*) ∪
set (*map* (*λx*. (*x*, *mpoly-sign val x*)) (*Polynomial.coeffs* (*Multiv-Poly-Props.one-less-degree q*)))
using *lc-assump-generation.simps*
using *Rec*(*3*) *Rec*(*4*) **by** *fastforce*

then obtain *branch* **where** *branch-prop*: *branch*∈*set* (*lc-assump-generation* (*Multiv-Poly-Props.one-less-degree q*) ((*Polynomial.lead-coeff q*, *0*) # *assumps*))
set (*fst branch*)
⊆ *set* ((*Polynomial.lead-coeff q*, *0*) # *assumps*) ∪
set (*map* (λ*x*. (*x*, *mpoly-sign val x*)) (*Polynomial.coeffs* (*Multiv-Poly-Props.one-less-degree q*)))
by *auto*
then have *branch-prop-2*: *branch*∈*set ?zero*
by *auto*
have *0* = *mpoly-sign val* (*Polynomial.lead-coeff q*) **using** ∗
unfolding *satisfies-evaluation-def eval-mpoly-def mpoly-sign-def Sturm-Tarski.sign-def*
by *simp*
then have *lc-sign*: (*Polynomial.lead-coeff q*, *0*) ∈ *set* (*map* (λ*x*. (*x*, *mpoly-sign val x*)) (*Polynomial.coeffs q*))
by (*simp add*: *Rec*(*1*) *coeff-in-coeffs*)
{**assume** ∗∗: *Multiv-Poly-Props.one-less-degree q* ≠ *0*
then have *set* (*Polynomial.coeffs* (*Multiv-Poly-Props.one-less-degree q*)) ⊆ *set* (*Polynomial.coeffs q*)
using *coeff-one-less-degree-subset* **unfolding** *Polynomial.coeffs-def*
by *blast*
then have *set* (*fst branch*) ⊆ *set assumps* ∪
set (*map* (λ*x*. (*x*, *mpoly-sign val x*)) (*Polynomial.coeffs q*))
using *branch-prop*
by (*smt* (*verit*, *del-insts*) *UnCI UnE lc-sign imageE image-eqI list.set-map set-ConsD subset-code*(*1*))
then have ∃ *branch*∈*set* (*lc-assump-generation q assumps*).
set (*fst branch*)
⊆ *set assumps* ∪
set (*map* (λ*x*. (*x*, *mpoly-sign val x*)) (*Polynomial.coeffs q*))
using *branch-prop-2*
using *set-is* **by** *force*
}
moreover {**assume** ∗∗: *Multiv-Poly-Props.one-less-degree q* = *0*
then have ∃ *branch*∈*set* (*lc-assump-generation q assumps*).
set (*fst branch*) ⊆ *set assumps* ∪
set (*map* (λ*x*. (*x*, *mpoly-sign val x*)) (*Polynomial.coeffs q*))
using *Rec.hyps*(*2*) *UnCI lc-sign insert-subset lc-assump-generation.elims list.set*(*2*) *option.case*(*1*) *prod.sel*(*1*) *subsetI*
by (*metis* (*no-types*, *lifting*))
}
ultimately have ∃ *branch*∈*set* (*lc-assump-generation q assumps*).
set (*fst branch*) ⊆ *set assumps* ∪ *set* (*map* (λ*x*. (*x*, *mpoly-sign val x*)) (*Polynomial.coeffs q*))
by *auto*
}
ultimately show *?case* **using** *eo* **by** *auto*
next
case (*Lookup0 q assumps*)
then obtain *branch* **where** *branch-prop*:*branch*∈*set* (*lc-assump-generation* (*Multiv-Poly-Props.one-less-degre*

q) *assumps*)
set (*fst branch*)
⊆ *set assumps* ∪ *set* (*map* (*λx.* (*x*, *mpoly-sign val x*)) (*Polynomial.coeffs* (*Multiv-Poly-Props.one-less-degree q*)))
by *auto*
have *same-gen*: *lc-assump-generation* (*one-less-degree q*) *assumps* = *lc-assump-generation q assumps*
using *Lookup0.hyps lc-assump-generation.simps* **by** *auto*
then have *branch-prop-2* :*branch*∈*set* (*lc-assump-generation q assumps*)
using *branch-prop*(*1*) **by** *auto*
{assume ∗: *Multiv-Poly-Props.one-less-degree q* ≠ *0*
then have *set* (*Polynomial.coeffs* (*Multiv-Poly-Props.one-less-degree q*)) ⊆ *set* (*Polynomial.coeffs q*)
using *coeff-one-less-degree-subset* **unfolding** *Polynomial.coeffs-def*
by *blast*
then have *set* (*fst branch*)
⊆ *set assumps* ∪
set (*map* (*λx.* (*x*, *mpoly-sign val x*)) (*Polynomial.coeffs q*))
using *branch-prop-2 branch-prop*(*2*) **by** *auto*
then have ∃ *branch*∈*set* (*lc-assump-generation q assumps*).
set (*fst branch*)
⊆ *set assumps* ∪
set (*map* (*λx.* (*x*, *mpoly-sign val x*)) (*Polynomial.coeffs q*))
using *branch-prop-2*
by *auto*
}
moreover {assume ∗: *Multiv-Poly-Props.one-less-degree q* = *0*
then have ∃ *branch*∈*set* (*lc-assump-generation q assumps*).
set (*fst branch*)
⊆ *set assumps* ∪
set (*map* (*λx.* (*x*, *mpoly-sign val x*)) (*Polynomial.coeffs q*))
by (*simp add*: *Lookup0*(*2*) *lc-assump-generation.simps*)
}
ultimately show *?case*
by *force*
next
case (*LookupN0 q assumps r*)
then show *?case* **using** *lc-assump-generation.simps*
by *simp*
qed

lemma *lc-assump-generation-valuation-satisfies-eval*:
fixes *q*:: *rmpoly*
assumes (*p*,*n*) ∈ *set* (*map* (*λx.* (*x*, *mpoly-sign val x*)) *ell*)
shows *satisfies-evaluation val p n*
proof −
obtain *c* **where** *c-prop*: *c* ∈ *set ell*
(*p*, *n*) = (*c*, *mpoly-sign val c*)
using *assms* **by** *auto*

have *satisfies-evaluation val c* (*mpoly-sign val c*)
unfolding *satisfies-evaluation-def mpoly-sign-def eval-mpoly-def Sturm-Tarski.sign-def* **by** *auto*
then show *?thesis*
using *c-prop* **by** *auto*
qed

lemma *lc-assump-generation-list-valuation*:
assumes $\bigwedge p\ n.\ (p,n) \in set\ init\text{-}assumps \Longrightarrow satisfies\text{-}evaluation\ val\ p\ n$
shows $\exists$ *branch* $\in$ *set* (*lc-assump-generation-list qs init-assumps*).
set (*fst branch*) $\subseteq$ *set* (*init-assumps*) $\cup$ *set*
(*map* (λx. (*x*, *mpoly-sign val x*)) (*coeffs-list qs*))
using *assms*
proof (*induct qs arbitrary*: *init-assumps*)
case *Nil*
then show *?case*
using *lc-assump-generation-list.simps*
by *simp*
next
case (*Cons a qs*)
then obtain *branch-a* **where** *branch-a-prop*: *branch-a* $\in$ *set* (*lc-assump-generation a init-assumps*)
set (*fst branch-a*) $\subseteq$ *set* (*init-assumps*) $\cup$ *set*
(*map* (λx. (*x*, *mpoly-sign val x*)) (*Polynomial.coeffs a*))
using *lc-assump-generation-valuation*
by *blast*
then obtain *branch-a-assms branch-a-poly* **where** *branch-a-is*: *branch-a* = (*branch-a-assms*, *branch-a-poly*)
set (*branch-a-assms*) $\subseteq$ *set* (*init-assumps*) $\cup$ *set*
(*map* (λx. (*x*, *mpoly-sign val x*)) (*Polynomial.coeffs a*))
by (*meson prod.exhaust-sel*)
have *inset*: *set* (*map* (λ*elem*. (*fst elem*, *branch-a-poly*#(*snd elem*))) (*lc-assump-generation-list qs branch-a-assms*))
$\subseteq$ *set* (*lc-assump-generation-list* (*a*#*qs*) *init-assumps*)
using *lc-assump-generation-list.simps branch-a-is*(*1*) *branch-a-prop*(*1*)
by *auto*
have $\bigwedge p\ n.\ (p,n) \in set\ branch\text{-}a\text{-}assms \Longrightarrow satisfies\text{-}evaluation\ val\ p\ n$
using *lc-assump-generation-valuation-satisfies-eval branch-a-is*(*1*) *fst-conv local.Cons*(*2*) *Set.basic-monos*(*7*) *UnE*
proof −
fix *p* :: *real mpoly* **and** *n* :: *rat*
assume (*p*, *n*) $\in$ *set branch-a-assms*
then show *satisfies-evaluation val p n*
by (*meson UnE lc-assump-generation-valuation-satisfies-eval Set.basic-monos*(*7*) *branch-a-is*(*2*) *local.Cons*(*2*))
qed
then obtain *branch* **where** *branch-props*:
branch$\in$*set* (*lc-assump-generation-list qs branch-a-assms*)

set (*fst branch*)
  ⊆ *set branch-a-assms* ∪
    set (*map* (λ*x*. (*x*, *mpoly-sign val x*)) (*coeffs-list qs*))
using *Cons.hyps*[*of branch-a-assms*] **by** *auto*
then have *branch-inset*:
(*fst branch*, *branch-a-poly*#(*snd branch*))
∈ *set* (*lc-assump-generation-list* (*a*#*qs*) *init-assumps*)
using *inset*
by *auto*
then have *subset-h*: *set* (*fst branch*) ⊆ *set* (*init-assumps*) ∪ (*set*
  (*map* (λ*x*. (*x*, *mpoly-sign val x*)) (*Polynomial.coeffs a*))
∪ *set* (*map* (λ*x*. (*x*, *mpoly-sign val x*)) (*coeffs-list qs*)))
using *branch-props*(*2*) *branch-a-is*(*2*) **by** *auto*
have *coeffs-list* (*a* # *qs*) = (*Polynomial.coeffs a*) @ (*coeffs-list qs*)
unfolding *coeffs-list-def* **by** *auto*
then have *same-set*:
set (*map* (λ*x*. (*x*, *mpoly-sign val x*)) (*coeffs-list* (*a*#*qs*))) = (*set*
  (*map* (λ*x*. (*x*, *mpoly-sign val x*)) (*Polynomial.coeffs a*))
∪ *set* (*map* (λ*x*. (*x*, *mpoly-sign val x*)) (*coeffs-list qs*)))
by *auto*
then have *set* (*fst branch*) ⊆ *set* (*init-assumps*) ∪ *set*
  (*map* (λ*x*. (*x*, *mpoly-sign val x*)) (*coeffs-list* (*a*#*qs*)))
using *subset-h* **by** *auto*
then show *?case*
using *branch-inset*
by (*metis* (*no-types*, *lifting*) *prod.sel*(*1*))
qed

lemma *base-case-info-M-assumps-complete*:
assumes $\bigwedge$*p n*. (*p*,*n*) ∈ *set init-assumps* ⟹ *satisfies-evaluation val p n*
shows ∃ (*assumps*, *mat-eq*) ∈ *set* (*base-case-info-M-assumps init-assumps*).
(∀ (*p*,*n*) ∈ *set assumps*. *satisfies-evaluation val p n*)
using *assms* **unfolding** *base-case-info-M-assumps-def* **by** *auto*

lemma *matches-len-complete-spmods-ex*:
assumes $\bigwedge$*p' n'*. (*p'*,*n'*) ∈ *set acc* ⟹ *satisfies-evaluation val p' n'*
shows ∃(*assumps*, *sturm-seq*) ∈ *set* (*spmods-multiv p q acc*).
(∀ (*p*,*n*) ∈ *set assumps*. *satisfies-evaluation val p n*)
using *assms*
proof (*induct p q acc rule*: *spmods-multiv-induct*)
case (*Base p q acc*)
then show *?case*
by (*auto simp add*: *spmods-multiv.simps*)
next
case (*Rec p q acc*)
let *?left* = *spmods-multiv* (*one-less-degree p*) *q* ((*Polynomial.lead-coeff p*, (*0*::*rat*))
\# *acc*)
let *?res-one* = *spmods-multiv-aux p q* ((*Polynomial.lead-coeff p*, (*1*::*rat*)) # *acc*)
let *?res-minus-one* = *spmods-multiv-aux p q* ((*Polynomial.lead-coeff p*, (−*1*::*rat*))

acc)
  have *spmods-is*: *spmods-multiv p q acc* = *?left* @ (*?res-one* @ *?res-minus-one*)
    using *spmods-multiv.simps Rec*(*1*−*2*) **by** *auto*
  have *satisfies-evaluation val* (*Polynomial.lead-coeff p*) *0* ∨
    satisfies-evaluation val (*Polynomial.lead-coeff p*) *1* ∨
    satisfies-evaluation val (*Polynomial.lead-coeff p*) (−*1*)
    unfolding *satisfies-evaluation-def*
    apply *auto*
    using *Sturm-Tarski.sign-cases*
    by (*metis of-int-1 of-int-minus*)
  then have *q*:
  (∀ *p′ n′*. (*p′*,*n′*) ∈ *set* ((*Polynomial.lead-coeff p*, *0*) # *acc*) ⟶ *satisfies-evaluation val p′ n′*) ∨
  (∀ *p′ n′*. (*p′*,*n′*) ∈ *set* ((*Polynomial.lead-coeff p*, *1*) # *acc*) ⟶ *satisfies-evaluation val p′ n′*) ∨
  (∀ *p′ n′*. (*p′*,*n′*) ∈ *set* ((*Polynomial.lead-coeff p*, −*1*) # *acc*) ⟶ *satisfies-evaluation val p′ n′*)
    using *Rec*
    by *simp*
  moreover {
    assume ∗:(∀ *p′ n′*. (*p′*,*n′*) ∈ *set* ((*Polynomial.lead-coeff p*, *0*) # *acc*) ⟶ *satisfies-evaluation val p′ n′*)
    then have ∃ *a*∈*set* (*spmods-multiv* (*Multiv-Poly-Props.one-less-degree p*) *q*
        ((*Polynomial.lead-coeff p*, *0*) # *acc*)).
      case a of
      (*a*, *ss*) ⇒
        ∀ *a*∈*set a*. *case a of* (*a*, *b*) ⇒ *satisfies-evaluation val a b*
      using *Rec* **by** *auto*
    then have *?case* **using** *spmods-is* **by** *auto*
  }
  moreover {
    assume ∗:(∀ *p′ n′*. (*p′*,*n′*) ∈ *set* ((*Polynomial.lead-coeff p*, *1*) # *acc*) ⟶ *satisfies-evaluation val p′ n′*)
    then obtain *assumps sturm-seq* **where**
      (*assumps*, *sturm-seq*) ∈ *set* (*spmods-multiv-aux p q* ((*Polynomial.lead-coeff p*, (*1*::*rat*)) # *acc*))
      ⋀ *p n*. (*p*,*n*) ∈ *set assumps* ⟹ *satisfies-evaluation val p n*
      using *matches-len-complete*[*of* ((*Polynomial.lead-coeff p*, (*1*::*rat*)) # *acc*) *val p q*]
      by *blast*
    then have *?case* **using** *spmods-is* **by** *auto*
  }
  moreover {
    assume ∗:(∀ *p′ n′*. (*p′*,*n′*) ∈ *set* ((*Polynomial.lead-coeff p*, −*1*) # *acc*) ⟶ *satisfies-evaluation val p′ n′*)
    then obtain *assumps sturm-seq* **where**
      (*assumps*, *sturm-seq*) ∈ *set* (*spmods-multiv-aux p q* ((*Polynomial.lead-coeff p*, (−*1*::*rat*)) # *acc*))
      ⋀ *p n*. (*p*,*n*) ∈ *set assumps* ⟹ *satisfies-evaluation val p n*

using *matches-len-complete*[*of* ((*Polynomial.lead-coeff p*, (−*1*::*rat*)) # *acc*) *val p q*]
by *blast*
then have *?case* **using** *spmods-is* **by** *auto*
}
ultimately show *?case*
by *fastforce*
next
case (*Lookup0 p q acc*)
then show *?case* **by** (*auto simp add*: *spmods-multiv.simps*)
next
case (*LookupN0 p q acc r*)
then have *spmods-is*: *spmods-multiv p q acc* = *spmods-multiv-aux p q acc*
using *spmods-multiv.simps* **by** *auto*
obtain *assumps sturm-seq* **where**
(*assumps*, *sturm-seq*) ∈ *set* (*spmods-multiv-aux p q acc*)
⋀ *p n*. (*p*,*n*) ∈ *set assumps* ⟹ *satisfies-evaluation val p n*
using *LookupN0*(*4*) *matches-len-complete*[*of acc val p q*]
by *blast*
then show *?case*
using *spmods-is* **by** *auto*
qed

lemma *matches-len-complete-spmods*:
assumes ⋀*p n*. (*p*,*n*) ∈ *set acc* ⟹ *satisfies-evaluation val p n*
obtains *assumps sturm-seq* **where**
(*assumps*, *sturm-seq*) ∈ *set* (*spmods-multiv p q acc*)
(*f*,*n*) ∈ *set assumps* ⟹ *satisfies-evaluation val f n*
using *assms matches-len-complete-spmods-ex*
by (*smt* (*verit*, *ccfv-threshold*) *case-prodD case-prodE*)

lemma *tarski-queries-complete-aux*:
assumes ⋀*p n*. (*p*,*n*) ∈ *set init-assumps* ⟹ *satisfies-evaluation val p n*
shows ∃ (*assumps*, *tq*) ∈ *set* (*construct-NofI-R-spmods p init-assumps I1 I2*).
(∀ (*p*,*n*) ∈ *set assumps*. *satisfies-evaluation val p n*)
unfolding *construct-NofI-R-spmods-def*
using *assms matches-len-complete-spmods-ex*[*of init-assumps val monoid-add-class.sum-list* (*map power2* (*p* # *I1*))]
by *meson*

lemma *tarski-queries-complete*:
assumes ⋀*p n*. (*p*,*n*) ∈ *set init-assumps* ⟹ *satisfies-evaluation val p n*
shows ∃ (*assumps*, *tq*) ∈ *set* (*construct-NofI-M p init-assumps I1 I2*).
(∀ (*p*,*n*) ∈ *set assumps*. *satisfies-evaluation val p n*)
proof −
have *cnoi-is*: *construct-NofI-M p init-assumps I1 I2* =
(*let ss-list* = *construct-NofI-R-spmods p init-assumps I1 I2 in*
map construct-NofI-single-M ss-list)
by *auto*

obtain *assumps tq* **where** *assumps-tq*: (*assumps*, *tq*) ∈ *set* (*construct-NofI-R-spmods p init-assumps I1 I2*)
(∀ (*p*,*n*) ∈ *set assumps*. *satisfies-evaluation val p n*)
using *tarski-queries-complete-aux assms*
by (*smt* (*verit*, *ccfv-threshold*) *case-prodE*)
then have *inset*: *construct-NofI-single-M* (*assumps*, *tq*) ∈ *set* (*construct-NofI-M p init-assumps I1 I2*)
by (*metis* (*no-types*, *lifting*) ‹*construct-NofI-M p init-assumps I1 I2* = *Let* (*construct-NofI-R-spmods p init-assumps I1 I2*) (*map construct-NofI-single-M*)› *in-set-conv-nth length-map nth-map*)

obtain *r* **where** (*assumps*, *r*) = *construct-NofI-single-M* (*assumps*, *tq*)
using *construct-NofI-single-M.simps* **by** *auto*
then show *?thesis*
using *inset assumps-tq*(*2*)
by (*smt* (*verit*) *case-prodI*)
qed

lemma *solve-for-rhs-rec-M-complete*:
assumes ⋀*p n*. (*p*,*n*) ∈ *set init-assumps* ⟹ *satisfies-evaluation val p n*
shows ∃ (*assumps*, *rhs-vec*) ∈ *set* (*construct-rhs-vector-rec-M p init-assumps ell*).
(∀ (*p*,*n*) ∈ *set assumps*. *satisfies-evaluation val p n*)
using *assms* **proof** (*induct ell arbitrary*: *init-assumps*)
case *Nil*
then show *?case* **by** *auto*
next
case (*Cons a ell*)
then obtain *qs1 qs2* **where** *qs-prop*: *a* = (*qs1*, *qs2*)
by (*meson prod.exhaust*)
have ∃ (*assumps*, *tq*) ∈ *set* (*construct-NofI-M p init-assumps qs1 qs2*).
(∀ (*p*,*n*) ∈ *set assumps*. *satisfies-evaluation val p n*)
using *tarski-queries-complete assms*
using *local.Cons*(*2*) **by** *presburger*
then obtain *assumps tq* **where** *assumps-tq*: (*assumps*, *tq*) ∈ *set* (*construct-NofI-M p init-assumps qs1 qs2*)
(∀ (*p*,*n*) ∈ *set assumps*. *satisfies-evaluation val p n*)
by *blast*
then have *ind-h*: ∃ *a*∈*set* (*construct-rhs-vector-rec-M p assumps ell*).
case a of
(*assumps1*, *rhs-vec1*) ⇒
∀ *a*∈*set assumps1*. *case a of* (*a*, *b*) ⇒ *satisfies-evaluation val a b*
using *Cons.hyps*
by (*metis* (*no-types*, *lifting*) *case-prodD*)
{**assume** ∗: *ell* = []
then have *construct-rhs-vector-rec-M p init-assumps* ((*qs1*, *qs2*)#[])
= *construct-rhs-vector-rec-M p init-assumps* (*a*#*ell*)
using *qs-prop*
by *blast*

have *rhs-is*: *construct-rhs-vector-rec-M p init-assumps* ((*qs1*, *qs2*)#[]) =
(*let TQ-list* = *construct-NofI-M p init-assumps qs1 qs2 in*
map (λ(*new-assumps*, *tq*). (*new-assumps*, [*tq*])) *TQ-list*)
using ∗ *construct-rhs-vector-rec-M.simps*(*2*) **by** *blast*
have (*assumps*, [*tq*]) ∈ *set* (*construct-rhs-vector-rec-M p init-assumps* ((*qs1*, *qs2*)#[]))
using *assumps-tq rhs-is*
by (*smt* (*verit*, *best*) *image-eqI list.set-map old.prod.case*)
then have *?case* **using** *assumps-tq*(*1*)
using ∗ *assumps-tq*(*2*) *qs-prop* **by** *blast*
} moreover {assume ∗: *length ell* > *0*
then obtain *v va* **where** *ell* = *v*#*va*
by (*meson Suc-le-length-iff Suc-less-eq le-simps*(*2*))
then have *construct-rhs-vector-rec-M p init-assumps* ((*qs1*, *qs2*)#*ell*) =
concat (*let TQ-list* = *construct-NofI-M p init-assumps qs1 qs2 in*
(*map* (λ(*new-assumps*, *tq*). (*let rec* = *construct-rhs-vector-rec-M p new-assumps ell in*
map (λ*r*. (*fst r*, *tq*#*snd r*)) *rec*)) *TQ-list*))
using ∗ *construct-rhs-vector-rec-M.simps*(*3*) **by** *auto*
then have *subset-prop*: *set* (*let rec* = *construct-rhs-vector-rec-M p assumps ell in*
map (λ*r*. (*fst r*, *tq*#*snd r*)) *rec*) ⊆ *set* (*construct-rhs-vector-rec-M p init-assumps* ((*qs1*, *qs2*)#*ell*))
using *assumps-tq*
by *auto*
then obtain *assumps1 rhs-vec1* **where** *assumps1-rhs1*:
(*assumps1*, *rhs-vec1*) ∈ *set* (*construct-rhs-vector-rec-M p assumps ell*)
(∀ (*p*,*n*) ∈ *set assumps1*. *satisfies-evaluation val p n*)
using *ind-h*
by *blast*
then have (*assumps1*, *tq*#*rhs-vec1*) ∈ *set* (*let rec* = *construct-rhs-vector-rec-M p assumps ell in*
map (λ*r*. (*fst r*, *tq*#*snd r*)) *rec*)
by (*metis* (*no-types*, *lifting*) *fst-eqD in-set-conv-nth length-map nth-map snd-conv*)
then have (*assumps1*, *tq*#*rhs-vec1*) ∈ *set* (*construct-rhs-vector-rec-M p init-assumps* ((*qs1*, *qs2*)#*ell*))
using *subset-prop* **by** *auto*
then have *?case*
using *assumps1-rhs1*(*2*) *qs-prop*
by *auto*
}
ultimately show *?case* **by** *auto*
qed

lemma *solve-for-rhs-M-complete*:
assumes $\bigwedge$*p n*. (*p*,*n*) ∈ *set init-assumps* ⟹ *satisfies-evaluation val p n*
shows ∃ (*assumps*, *rhs-vec*) ∈ *set* (*construct-rhs-vector-M p init-assumps qs Is*).

(∀ (p,n) ∈ set assumps. satisfies-evaluation val p n)
proof −
obtain *assumps rhs-rec-vec* **where** *assumps-rec*:
(*assumps*, *rhs-rec-vec*) ∈ *set* (*construct-rhs-vector-rec-M p init-assumps* (*pull-out-pairs qs Is*))
(∀ (*p*,*n*) ∈ *set assumps. satisfies-evaluation val p n*)
using *assms solve-for-rhs-rec-M-complete*
by (*smt* (*verit*, *ccfv-threshold*) *case-prodE*)
then have (*assumps*, *rhs-rec-vec*)
∈ *set* (*construct-rhs-vector-rec-M p init-assumps*
(*map* (λ(*I1*, *I2*). (*retrieve-polys qs I1*, *retrieve-polys qs I2*)) *Is*)) ⟹
(⋀*z f A*. (*z* ∈ *f* ‘ *A*) = (∃ *x*∈*A*. *z* = *f x*)) ⟹
∀ *x*∈*set assumps. case x of* (*x*, *xa*) ⇒ *satisfies-evaluation val x xa* ⟹
∃ *x*∈*set* (*construct-rhs-vector-rec-M p init-assumps*
(*map* (λ(*I1*, *I2*). (*retrieve-polys qs I1*, *retrieve-polys qs I2*))
Is)).
∀ *x*∈*set* (*fst x*). *case x of* (*x*, *xa*) ⇒ *satisfies-evaluation val x xa*
by *fastforce*
then show *?thesis*
using *image-iff assumps-rec* **unfolding** *construct-rhs-vector-M-def* **by** *auto*
qed

lemma *solve-for-lhs-M-complete*:
assumes ⋀*p n*. (*p*,*n*) ∈ *set init-assumps* ⟹ *satisfies-evaluation val p n*
shows ∃ (*assumps*, *lhs-vec*) ∈ *set* (*solve-for-lhs-M p init-assumps qs subsets matr*).
(∀ (*p*,*n*) ∈ *set assumps. satisfies-evaluation val p n*)
proof −
obtain *assumps rhs-vec* **where** *assumps-prop*:
(*assumps*, *rhs-vec*) ∈ *set* (*construct-rhs-vector-M p init-assumps qs subsets*)
(∀ (*p*,*n*) ∈ *set assumps. satisfies-evaluation val p n*)
using *solve-for-rhs-M-complete assms*
by (*metis* (*mono-tags*, *lifting*) *prod.simps*(*2*) *surj-pair*)
let *?rhs* = (*assumps*, *rhs-vec*)
have (*assumps*, *rhs-vec*)
∈ *set* (*construct-rhs-vector-M p init-assumps qs subsets*) ⟹
(*assumps*, *solve-for-lhs-single-M p qs subsets matr rhs-vec*)
∈ (λ*rhs*. (*fst rhs*, *solve-for-lhs-single-M p qs subsets matr* (*snd rhs*))) ‘
set (*construct-rhs-vector-M p init-assumps qs subsets*)
using *image-iff* **by** *fastforce*
then have (*fst ?rhs*, *solve-for-lhs-single-M p qs subsets matr* (*snd ?rhs*))
∈ *set* (*solve-for-lhs-M p init-assumps qs subsets matr*)
using *assumps-prop*(*1*) **unfolding** *solve-for-lhs-M-def*
by *auto*
then show *?thesis* **using** *assumps-prop*(*2*)
by *auto*
qed

lemma *reduce-system-single-M-complete*:

assumes $\bigwedge p\ n.\ (p,n) \in set\ init\text{-}assumps \Longrightarrow satisfies\text{-}evaluation\ val\ p\ n$
shows $\exists\ (assumps,\ mat\text{-}eq) \in set\ (reduce\text{-}system\text{-}single\text{-}M\ p\ qs\ (init\text{-}assumps,\ (m,subs,signs))).$
$(\forall\ (p,n) \in set\ assumps.\ satisfies\text{-}evaluation\ val\ p\ n)$
proof −
obtain *assumps lhs-vec* **where** *assumps-lhs*:
$(assumps,\ lhs\text{-}vec) \in set\ (solve\text{-}for\text{-}lhs\text{-}M\ p\ init\text{-}assumps\ qs\ subs\ m)$
$(\forall\ (p,n) \in set\ assumps.\ satisfies\text{-}evaluation\ val\ p\ n)$
using *assms solve-for-lhs-M-complete*
by (*metis* (*mono-tags*, *lifting*) *split-conv surj-pair*)
then have $(assumps,\ lhs\text{-}vec) \in set\ (solve\text{-}for\text{-}lhs\text{-}M\ p\ init\text{-}assumps\ qs\ subs\ m) \Longrightarrow$
$\forall x \in set\ assumps.\ case\ x\ of\ (p,\ n) \Rightarrow satisfies\text{-}evaluation\ val\ p\ n \Longrightarrow$
$(\bigwedge z\ f\ A.\ (z \in f\ `\ A) = (\exists x \in A.\ z = f\ x)) \Longrightarrow$
$\exists x \in set\ (solve\text{-}for\text{-}lhs\text{-}M\ p\ init\text{-}assumps\ qs\ subs\ m).$
$\forall x \in set\ (fst\ x).\ case\ x\ of\ (x,\ xa) \Rightarrow satisfies\text{-}evaluation\ val\ x\ xa$
by (*metis fst-conv*)
then show *?thesis*
using *assumps-lhs image-iff reduce-system-single-M.simps* **by** *auto*
qed

lemma *reduce-system-M-concat-map-helper*:
fixes *a*:: *'a list*
fixes *b*:: *'a list list*
assumes $a \in set\ b$
shows $set\ a \subseteq\ set\ (concat\ b)$
using *assms* **by** *auto*

lemma *reduce-system-M-complete*:
assumes $\bigwedge p\ n.\ (p,n) \in set\ init\text{-}assumps \Longrightarrow satisfies\text{-}evaluation\ val\ p\ n$
assumes $(init\text{-}assumps,\ mat\text{-}eq) \in set\ input\text{-}list$
shows $\exists\ (assumps,\ mat\text{-}eq) \in set\ (reduce\text{-}system\text{-}M\ p\ qs\ input\text{-}list).$
$(\forall\ (p,n) \in set\ assumps.\ satisfies\text{-}evaluation\ val\ p\ n)$
proof −
obtain *m subs signs* **where** *mat-eq*: $(init\text{-}assumps,\ (m,subs,signs)) \in set\ input\text{-}list$
using *assms*(*2*)
by (*metis prod-cases3*)
have *reduce*: $reduce\text{-}system\text{-}M\ p\ qs\ input\text{-}list = concat\ (map\ (reduce\text{-}system\text{-}single\text{-}M\ p\ qs)\ input\text{-}list)$
using *reduce-system-M.simps* **by** *auto*
have *elem*: $(reduce\text{-}system\text{-}single\text{-}M\ p\ qs\ (init\text{-}assumps,\ (m,subs,signs))) \in set\ (map\ (reduce\text{-}system\text{-}single\text{-}M\ p\ qs)\ input\text{-}list)$
using *mat-eq*
by (*metis* (*no-types*, *opaque-lifting*) *image-eqI image-set*)
then have $set\ (reduce\text{-}system\text{-}single\text{-}M\ p\ qs\ (init\text{-}assumps,\ (m,subs,signs))) \subseteq set\ (concat\ (map\ (reduce\text{-}system\text{-}single\text{-}M\ p\ qs)\ input\text{-}list))$
using *reduce-system-M-concat-map-helper*[*of* (*reduce-system-single-M p qs* (*init-assumps*, (*m*,*subs*,*signs*))) (*map* (*reduce-system-single-M p qs*) *input-list*)]

by *auto*
then have *subset*: *set* (*reduce-system-single-M p qs* (*init-assumps*, (*m*,*subs*,*signs*))) ⊆ *set*
(*reduce-system-M p qs input-list*)
using *mat-eq reduce* **by** *auto*
have ∃ (*assumps*, *mat-eq*)∈ *set* (*reduce-system-single-M p qs* (*init-assumps*, (*m*,*subs*,*signs*))).
(∀ (*p*,*n*) ∈ *set assumps*. *satisfies-evaluation val p n*)
using *reduce-system-single-M-complete assms*(*1*)
by *presburger*
then show *?thesis* **using** *subset*
using *basic-trans-rules*(*31*) **by** *blast*
qed

lemma *combine-systems-M-complete*:
assumes (*assumps1*, *mat-eq1*) ∈ *set list1*
assumes (∀ (*p*,*n*) ∈ *set assumps1*. *satisfies-evaluation val p n*)
assumes (*assumps2*, *mat-eq2*) ∈ *set list2*
assumes (∀ (*p*,*n*) ∈ *set assumps2*. *satisfies-evaluation val p n*)
shows ∃ (*assumps*, *mat-eq*) ∈ *set* (*snd* (*combine-systems-M p q1 list1 q2 list2*)).
(∀ (*p*,*n*) ∈ *set assumps*. *satisfies-evaluation val p n*)
proof −
have *snd-is*: *snd* (*combine-systems-M p q1 list1 q2 list2*) = *concat* (*map* (λ*l1*. (*map* (λ*l2*. *combine-systems-single-M p q1 l1 q2 l2*) *list2*)) *list1*)
by *auto*
have (*assumps1*, *mat-eq1*) ∈ *set list1* ⟹
(*assumps2*, *mat-eq2*) ∈ *set list2* ⟹
∃ *x*∈*set list1*.
(*assumps1* @ *assumps2*,
snd (*combine-systems-R p* (*q1*, *mat-eq1*) (*q2*, *mat-eq2*)))
∈ *combine-systems-single-M p q1 x q2* ‘ *set list2*
by (*metis combine-systems-single-M.simps rev-image-eqI*)
then have *inset*: *combine-systems-single-M p q1* (*assumps1*, *mat-eq1*) *q2* (*assumps2*, *mat-eq2*)
∈ *set* (*concat* (*map* (λ*l1*. (*map* (λ*l2*. *combine-systems-single-M p q1 l1 q2 l2*) *list2*)) *list1*))
using *snd-is assms*(*1*) *assms*(*3*) **by** *auto*
obtain *mat-eq* **where** *mat-eq*: (*append assumps1 assumps2*, *mat-eq*) = *combine-systems-single-M p q1* (*assumps1*, *mat-eq1*) *q2* (*assumps2*, *mat-eq2*)
using *combine-systems-single-M.simps* **by** *auto*
then have (*append assumps1 assumps2*, *mat-eq*) ∈
set (*concat* (*map* (λ*l1*. (*map* (λ*l2*. *combine-systems-single-M p q1 l1 q2 l2*) *list2*)) *list1*))
using *inset* **by** *auto*
then have *inset2*: (*append assumps1 assumps2*, *mat-eq*) ∈ *set* (*snd* (*combine-systems-M p q1 list1 q2 list2*))
using *snd-is* **by** *auto*
have ⋀*p n*. (*p*,*n*) ∈ *set* (*append assumps1 assumps2*) ⟹ *satisfies-evaluation val p n*

using *assms*(*2*) *assms*(*4*) **by** *auto*
then show *?thesis*
using *inset2*
by *blast*
qed

lemma *get-all-valuations-calculate-data-M*:
assumes $\bigwedge$*p n*. (*p*,*n*) ∈ *set init-assumps* ⟹ *satisfies-evaluation val p n*
shows ∃ (*assumps*, *mat-eq*) ∈ *set* (*calculate-data-assumps-M p qs init-assumps*).
(∀ (*p*,*n*) ∈ *set assumps*. *satisfies-evaluation val p n*)
using *assms*
proof (*induct length qs arbitrary*: *val p init-assumps qs rule*: *less-induct*)
case (*less qs val p init-assumps*)
have *length qs* = *0* ∨ *length qs* = *1* ∨ *length qs* > *1*
by (*meson less-one nat-neq-iff*)
{assume ∗: *length qs* = *0*
then have *calc-data-is*: *calculate-data-assumps-M p* [] *init-assumps*
= *map* (λ(*assumps*,(*a*,(*b*,*c*))). (*assumps*, (*a*,*b*,*map* (*drop 1*) *c*))) (*reduce-system-M p* [*1*] (*base-case-info-M-assumps init-assumps*))
using *calculate-data-assumps-M.simps*
by *simp*
obtain *assumps-inbtw mat-eq-inbtw* **where** *inbtw-props*: (*assumps-inbtw*, *mat-eq-inbtw*) ∈ *set* (*base-case-info-M-assumps init-assumps*)
(∀ (*p*,*n*) ∈ *set assumps-inbtw*. *satisfies-evaluation val p n*)
using *base-case-info-M-assumps-complete less*(*2*)
by (*metis* (*no-types*, *lifting*) *case-prodE*)
then have ($\bigwedge$*p n*. (*p*, *n*) ∈ *set assumps-inbtw* ⟹ *satisfies-evaluation val p n*)
by *auto*
then have ∃ (*assumps*, *mat-eq*) ∈ *set* (*reduce-system-M p* [*1*] (*base-case-info-M-assumps init-assumps*)).
(∀ (*p*,*n*) ∈ *set assumps*. *satisfies-evaluation val p n*)
using *reduce-system-M-complete*[*of assumps-inbtw val mat-eq-inbtw* (*base-case-info-M-assumps init-assumps*) *p* [*1*]] *inbtw-props*(*1*)
by *fastforce*
then obtain *assumps mat-eq* **where** *assumps-mat*: (*assumps*, *mat-eq*) ∈ *set* (*reduce-system-M p* [*1*] (*base-case-info-M-assumps init-assumps*))
(∀ (*p*,*n*) ∈ *set assumps*. *satisfies-evaluation val p n*)
by *blast*
then obtain *a b c* **where** *mat-eq* = (*a*, (*b*, *c*))
by (*metis prod.exhaust-sel*)
then have (*assumps*, (*a*, (*b*, *c*))) ∈ *set* (*reduce-system-M p* [*1*] (*base-case-info-M-assumps init-assumps*))
using *assumps-mat*(*1*) **by** *auto*
then have (*assumps*, (*a*,*b*,*map* (*drop 1*) *c*)) ∈ *set* (*calculate-data-assumps-M p* [] *init-assumps*)
using *calc-data-is image-iff*
by (*smt* (*z3*) *image-set split-conv*)
then have ∃ (*assumps*, *mat-eq*) ∈ *set* (*calculate-data-assumps-M p qs init-assumps*).
(∀ (*p*,*n*) ∈ *set assumps*. *satisfies-evaluation val p n*)

using *assumps-mat*(*2*) ∗ **by** *blast*
} **moreover**
{assume ∗: *length qs* = *1*
then have *calc-data-is*: *calculate-data-assumps-M p qs init-assumps* = *reduce-system-M p qs* (*base-case-info-M-assumps init-assumps*)
by *auto*
obtain *assumps-inbtw mat-eq-inbtw* **where** *inbtw-props*: (*assumps-inbtw*, *mat-eq-inbtw*) ∈ *set* (*base-case-info-M-assumps init-assumps*)
(∀ (*p*,*n*) ∈ *set assumps-inbtw*. *satisfies-evaluation val p n*)
using *base-case-info-M-assumps-complete less*(*2*)
by (*metis* (*no-types*, *lifting*) *case-prodE*)
then have (⋀*p n*. (*p*, *n*) ∈ *set assumps-inbtw* ⟹ *satisfies-evaluation val p n*)
by *auto*
then have ∃ (*assumps*, *mat-eq*) ∈ *set* (*reduce-system-M p qs* (*base-case-info-M-assumps init-assumps*)).
(∀ (*p*,*n*) ∈ *set assumps*. *satisfies-evaluation val p n*)
using *reduce-system-M-complete*[*of assumps-inbtw val mat-eq-inbtw* (*base-case-info-M-assumps init-assumps*) *p qs*] *inbtw-props*(*1*)
by *fastforce*
then have ∃ (*assumps*, *mat-eq*) ∈ *set* (*calculate-data-assumps-M p qs init-assumps*).
(∀ (*p*,*n*) ∈ *set assumps*. *satisfies-evaluation val p n*)
using *calc-data-is* **by** *auto*
} **moreover**
{assume ∗: *length qs* > *1*
let *?len* = *length qs*
have *calc-data-is*: *calculate-data-assumps-M p qs init-assumps*
= (*let q1* = *take* (*?len div 2*) *qs*; *left* = *calculate-data-assumps-M p q1 init-assumps*;
q2 = *drop* (*?len div 2*) *qs*; *right* = *calculate-data-assumps-M p q2 init-assumps*;
comb = *combine-systems-M p q1 left q2 right in*
reduce-system-M p (*fst comb*) (*snd comb*)
) **using** ∗ *calculate-data-assumps-M.simps*
by (*smt* (*z3*) *div-eq-0-iff bot-nat-0.not-eq-extremum div-greater-zero-iff rel-simps*(*69*))

let *?q1* = *take* (*?len div 2*) *qs*
let *?left* = *calculate-data-assumps-M p ?q1 init-assumps*
let *?q2* = *drop* (*?len div 2*) *qs*
let *?right* = *calculate-data-assumps-M p ?q2 init-assumps*
let *?comb* = *combine-systems-M p ?q1 ?left ?q2 ?right*
have *calc-data-is2*: *calculate-data-assumps-M p qs init-assumps*
= *reduce-system-M p* (*fst ?comb*) (*snd ?comb*) **using** *calc-data-is*
unfolding *Let-def*
by *auto*
have *length ?q1* < *?len* **using** ∗
by *auto*
then have ∃ (*assumps*, *mat-eq*) ∈ *set ?left*.
(∀ (*p*,*n*) ∈ *set assumps*. *satisfies-evaluation val p n*)
using *less.hyps*[*of ?q1 init-assumps val p*]
less.prems **by** *auto*

```
    then obtain assumps1 mat-eq1 where assumps1:
      (assumps1, mat-eq1) ∈ set ?left
      (∀ (p,n) ∈ set assumps1. satisfies-evaluation val p n)
      by blast
    have length ?q2 < ?len using ∗
      by auto
    then have ∃ (assumps, mat-eq) ∈ set ?right.
   (∀ (p,n) ∈ set assumps. satisfies-evaluation val p n)
      using less.hyps[of ?q2 init-assumps val p]
        less.prems by auto
    then obtain assumps2 mat-eq2 where assumps2:
      (assumps2, mat-eq2) ∈ set ?right
      (∀ (p,n) ∈ set assumps2. satisfies-evaluation val p n)
      by blast
    have ∃ (assumps, mat-eq) ∈ set (snd ?comb).
   (∀ (p,n) ∈ set assumps. satisfies-evaluation val p n)
      using combine-systems-M-complete assumps1 assumps2
      by auto
    then obtain assumps-inbtw mat-eq-inbtw where
      (assumps-inbtw, mat-eq-inbtw) ∈ set (snd ?comb)
      (∀ (p,n) ∈ set assumps-inbtw. satisfies-evaluation val p n)
      by blast
     then have ∃ (assumps, mat-eq) ∈ set (reduce-system-M p (fst ?comb) (snd
?comb)).
    (∀ (p,n) ∈ set assumps. satisfies-evaluation val p n)
     using reduce-system-M-complete[of assumps-inbtw val mat-eq-inbtw snd ?comb
p fst ?comb]
      by fastforce
   then have ∃ (assumps, mat-eq) ∈ set (calculate-data-assumps-M p qs init-assumps).
   (∀ (p,n) ∈ set assumps. satisfies-evaluation val p n)
      using calc-data-is2
      by presburger
  }
  ultimately show ?case
    using ‹length qs = 0 ∨ length qs = 1 ∨ 1 < length qs› by fastforce
qed

fun extract-signs-single:: assumps × matrix-equation ⇒ (assumps × rat list list)
  where extract-signs-single (assumps, mat-eq) = (assumps, snd (snd mat-eq))

lemma extract-signs-alt-char:
  shows extract-signs qs = map extract-signs-single qs
proof (induct qs)
  case Nil
  then show ?case by auto
next
  case (Cons a qs)
  then show ?case
    using extract-signs.simps extract-signs-single.simps
```

by (*smt* (*verit*, *ccfv-threshold*) *list.map*(*2*) *snd-conv split-conv surj-pair*)
qed

lemma *get-all-valuations-helper*:
assumes (*assumps*, *mat-eq*) ∈ *set ell*
assumes *extract-signs-single* (*assumps*, *mat-eq*) = (*assumps*, *signs*)
shows (*assumps*, *signs*) ∈ *set* (*extract-signs ell*)
proof −
have *extract-signs ell* = *map extract-signs-single ell*
using *extract-signs-alt-char*
by *metis*
then show *?thesis* **using** *assms image-eqI*
by (*metis list.set-map*)
qed

lemma *get-all-valuations-alt*:
assumes ⋀*p n*. (*p*,*n*) ∈ *set init-assumps* ⟹ *satisfies-evaluation val p n*
shows ∃(*assumps*, *signs*) ∈ *set* (*sign-determination-inner qs init-assumps*).
(∀ *p n*. (*p*,*n*) ∈ *set assumps* ⟶ *satisfies-evaluation val p n*)
proof − **obtain** *branch-gen* **where** *branch-gen-prop*:
branch-gen ∈ *set* (*lc-assump-generation-list qs init-assumps*)
set (*fst branch-gen*) ⊆ *set* (*init-assumps*) ∪ *set*
(*map* (λ*x*. (*x*, *mpoly-sign val x*)) (*coeffs-list qs*))
using *assms lc-assump-generation-list-valuation*
by *blast*
then have *branch-gen*: ⋀*p n*. (*p*,*n*) ∈ *set* (*fst branch-gen*) ⟹ *satisfies-evaluation val p n*
using *lc-assump-generation-valuation-satisfies-eval*
by (*meson UnE assms in-mono*)
let *?poly-p-branch* = *poly-p-in-branch branch-gen*
let *?pos-limit-branch* = *fst* (*limit-points-on-branch branch-gen*)
let *?neg-limit-branch* = *snd* (*limit-points-on-branch branch-gen*)
let *?calculate-data-branch* = *extract-signs* (*calculate-data-assumps-M ?poly-p-branch* (*snd branch-gen*) (*fst branch-gen*))
have *lim-points*: *limit-points-on-branch branch-gen* = (*?pos-limit-branch*, *?neg-limit-branch*)
by *auto*
have *set* (*let poly-p-branch* = *poly-p-in-branch branch-gen*;
(*pos-limit-branch*, *neg-limit-branch*) = *limit-points-on-branch branch-gen*;
calculate-data-branch = *extract-signs* (*calculate-data-assumps-M poly-p-branch* (*snd branch-gen*) (*fst branch-gen*))
in map (λ(*a*, *signs*). (*a*, *pos-limit-branch*#*neg-limit-branch*#*signs*)) *calculate-data-branch*)
⊆ *set* (*sign-determination-inner qs init-assumps*)
using *branch-gen-prop*(*1*) *sign-determination-inner.simps* **by** *auto*
then have *in-signdet*: *set* (*map* (λ(*a*, *signs*). (*a*, *?pos-limit-branch*#*?neg-limit-branch*#*signs*)) *?calculate-data-branch*)
⊆ *set* (*sign-determination-inner qs init-assumps*)
using *lim-points* **unfolding** *Let-def*
proof −

assume *set* (*case limit-points-on-branch branch-gen of* (*pos-limit-branch*, *neg-limit-branch*) ⇒ *map* (λ(*a*, *signs*). (*a*, *pos-limit-branch* # *neg-limit-branch* # *signs*)) (*extract-signs* (*calculate-data-assumps-M* (*poly-p-in-branch branch-gen*) (*snd branch-gen*) (*fst branch-gen*)))) ⊆ *set* (*sign-determination-inner qs init-assumps*)
  then show *?thesis*
    by (*simp add*: *split-def*)
qed
obtain *assumps mat-eq* **where** *assumps-mat-prop*:
  (*assumps*, *mat-eq*) ∈ *set* (*calculate-data-assumps-M ?poly-p-branch* (*snd branch-gen*) (*fst branch-gen*))
  (∀ (*p*,*n*) ∈ *set assumps*. *satisfies-evaluation val p n*)
  using *get-all-valuations-calculate-data-M branch-gen*
  by (*smt* (*verit*, *del-insts*) *prod.exhaust-sel split-def*)
then obtain *m subsets signs* **where** *mat-eq-is*: *mat-eq* = (*m*, (*subsets*, *signs*))
  using *prod-cases3* **by** *blast*
then have *pull-out*: *extract-signs-single* (*assumps*, *mat-eq*) = (*assumps*, *signs*)
  using *extract-signs-single.simps* **by** *auto*
have *?calculate-data-branch* = (*map extract-signs-single*
    (*calculate-data-assumps-M* (*poly-p-in-branch branch-gen*)
    (*snd branch-gen*) (*fst branch-gen*)))
  using *extract-signs-alt-char* **by** *auto*
then have (*assumps*, *signs*) ∈ *set ?calculate-data-branch*
  using *assumps-mat-prop*(*1*) *pull-out get-all-valuations-helper*
  by *blast*
then have (*assumps*, *?pos-limit-branch*#*?neg-limit-branch*#*signs*) ∈
  set (*sign-determination-inner qs init-assumps*)
  using *in-signdet* **by** *auto*
then show *?thesis*
  using *assumps-mat-prop*(*2*)
  by *blast*
qed

lemma *get-all-valuations*:
assumes ⋀*p n*. (*p*,*n*) ∈ *set init-assumps* ⟹ *satisfies-evaluation val p n*
obtains *assumps signs* **where** (*assumps*, *signs*) ∈ *set* (*sign-determination-inner qs init-assumps*)
  ⋀*p n*. (*p*,*n*) ∈ *set assumps* ⟹ *satisfies-evaluation val p n*
using *assms get-all-valuations-alt*
by (*metis* (*no-types*, *lifting*) *case-prodE*)

17.11 Correctness of elim forall and elim exist

lemma *subset-zip-is-subset*:
assumes *set qs1* ⊆ *set qs*
assumes *signs* = *map* (*mpoly-sign val*) *qs*
assumes *signs1* = *map* (*mpoly-sign val*) *qs1*
shows *subset*: *set* (*zip qs1 signs1*) ⊆ *set* (*zip qs signs*)
proof *clarsimp*
fix *a b*

assume ∗: (*a*, *b*) ∈ *set* (*zip qs1 signs1*)
then have *a* ∈ *set qs1*
by (*meson set-zip-leftD*)
then have *a-in-qs*: *a* ∈ *set qs*
using *assms* **by** *auto*
have ⋀*ba*. *set qs1* ⊆ *set qs* ⟹
signs = *map* (*mpoly-sign val*) *qs* ⟹
signs1 = *map* (*mpoly-sign val*) *qs1* ⟹
zip qs1 (*map* (*mpoly-sign val*) *qs1*) =
map2 (λ*x y*. (*x*, *mpoly-sign val y*)) *qs1 qs1* ⟹
(*a*, *ba*) ∈ *set* (*zip qs1 qs1*) ⟹
b = *mpoly-sign val ba* ⟹ *mpoly-sign val ba* = *mpoly-sign val a*
by (*simp add*: *zip-same*)
then have *b-is*: *b* = *mpoly-sign val a*
using ∗ *assms zip-map2*[*of qs1 mpoly-sign val qs1*]
by (*auto*)
have (*a*, *mpoly-sign val a*) ∈ *set* (*map* (λ(*x*,*y*).(*x*, *mpoly-sign val y*)) (*zip qs qs*))
using *a-in-qs zip-same*[*of a a qs*] **by** *auto*
then have (*a*, *mpoly-sign val a*) ∈ *set* (*map2* (λ*x y*. (*x*, *mpoly-sign val y*)) *qs qs*)
by *auto*
then show (*a*, *b*) ∈ *set* (*zip qs signs*)
using *b-is a-in-qs assms zip-map2*[*of qs mpoly-sign val qs*]
using *assms*(*1*) **by** *presburger*
qed

lemma *extract-polys-subset*:
assumes *signs* = *map* (*mpoly-sign val*) *qs*
assumes *signs1* = *map* (*mpoly-sign val*) *qs1*
assumes *set qs1* ⊆ *set qs*
assumes *Some w* = *lookup-sem-M F* (*zip qs1 signs1*)
shows *lookup-sem-M F* (*zip qs signs*) = *lookup-sem-M F* (*zip qs1 signs1*)
using *assms*
proof (*induct F arbitrary*: *w*)
case *TrueF*
then show *?case* **by** *auto*
next
case *FalseF*
then show *?case* **by** *auto*
next
case (*Atom x*)
have *subset*: *set* (*zip qs1 signs1*) ⊆ *set* (*zip qs signs*)
using *subset-zip-is-subset Atom.prems* **by** *auto*
show *?case*
proof (*cases x*)
case (*Less x1*)
then obtain *i* **where** *i-prop*: *lookup-assump-aux x1* (*zip qs1 signs1*) = *Some i*
using *Atom.prems lookup-sem-M.simps*
by (*metis lookup-assump-aux-eo option.simps*(*3*) *option.simps*(*4*))
have *i-is*: (*x1*, *i*) ∈ *set* (*zip qs1 signs1*)

```
      using i-prop lookup-assump-means-inset[of x1 (zip qs1 signs1) i]
        in-set-member[of (x1, i) (zip qs1 signs1) ] by auto
    then have (x1, i) ∈ set (zip qs signs)
      using subset by auto
    then obtain i1 where lookup-assump-aux x1 (zip qs signs) = Some i1
      using inset-means-lookup-assump-some[of x1 i (zip qs signs)]
      by meson
    then have i1-is: (x1, i1) ∈ set (zip qs signs)
      using lookup-assump-means-inset[of x1 (zip qs signs) i1]
        in-set-member[of (x1, i1) (zip qs signs)] by auto
    have ⋀b. signs1 = map (mpoly-sign val) qs1 ⟹
          zip qs1 (map (mpoly-sign val) qs1) =
          map2 (λx y. (x, mpoly-sign val y)) qs1 qs1 ⟹
          (x1, b) ∈ set (zip qs1 qs1) ⟹
          i = mpoly-sign val b ⟹ mpoly-sign val b = mpoly-sign val x1
      by (simp add: zip-same)
    then have i-sign :i = mpoly-sign val x1
      using i-is  Atom.prems(2) zip-map2[of qs1 mpoly-sign val qs1]
      by (auto)
    have ⋀b. signs = map (mpoly-sign val) qs ⟹
          zip qs (map (mpoly-sign val) qs) =
          map2 (λx y. (x, mpoly-sign val y)) qs qs ⟹
          (x1, b) ∈ set (zip qs qs) ⟹
          i1 = mpoly-sign val b ⟹ mpoly-sign val b = mpoly-sign val x1
      by (simp add: zip-same)
    then have i1-sign: i1 = mpoly-sign val x1
      using i1-is  Atom.prems(1) zip-map2[of qs mpoly-sign val qs]
      by (auto)
    have Sturm-Tarski.sign i1 = Sturm-Tarski.sign i
      using i-sign i1-sign
      by blast
    then show ?thesis using Atom.prems i-prop lookup-sem-M.simps
       using Less ‹lookup-assump-aux x1 (zip qs signs) = Some i1› i1-sign i-sign
by presburger
  next
    case (Eq x1)
    then obtain i where i-prop: lookup-assump-aux x1 (zip qs1 signs1) = Some i
      using Atom.prems lookup-sem-M.simps
      by (metis lookup-assump-aux-eo option.simps(3) option.simps(4))
    have i-is: (x1, i) ∈ set (zip qs1 signs1)
      using i-prop lookup-assump-means-inset[of x1 (zip qs1 signs1) i]
        in-set-member[of (x1, i) (zip qs1 signs1) ] by auto
    then have (x1, i) ∈ set (zip qs signs)
      using subset by auto
    then obtain i1 where lookup-assump-aux x1 (zip qs signs) = Some i1
      by (meson inset-means-lookup-assump-some)
    then have i1-is: (x1, i1) ∈ set (zip qs signs)
      using lookup-assump-means-inset[of x1 (zip qs signs) i1]
        in-set-member[of (x1, i1) (zip qs signs)] by auto
```

have $\bigwedge b$. *signs1* = *map* (*mpoly-sign val*) *qs1* $\Longrightarrow$
zip qs1 (*map* (*mpoly-sign val*) *qs1*) =
map2 (λx *y*. (*x*, *mpoly-sign val y*)) *qs1 qs1* $\Longrightarrow$
(*x1*, *b*) $\in$ *set* (*zip qs1 qs1*) $\Longrightarrow$
i = *mpoly-sign val b* $\Longrightarrow$ *mpoly-sign val b* = *mpoly-sign val x1*
by (*simp add*: *zip-same*)
then have *i-sign* :*i* = *mpoly-sign val x1*
using *i-is* *Atom.prems*(*2*) *zip-map2*[*of qs1 mpoly-sign val qs1*]
by (*auto*)
have $\bigwedge b$. *signs* = *map* (*mpoly-sign val*) *qs* $\Longrightarrow$
zip qs (*map* (*mpoly-sign val*) *qs*) =
map2 (λx *y*. (*x*, *mpoly-sign val y*)) *qs qs* $\Longrightarrow$
(*x1*, *b*) $\in$ *set* (*zip qs qs*) $\Longrightarrow$
i1 = *mpoly-sign val b* $\Longrightarrow$ *mpoly-sign val b* = *mpoly-sign val x1*
by (*simp add*: *zip-same*)
then have *i1-sign*: *i1* = *mpoly-sign val x1*
using *i1-is* *Atom.prems*(*1*) *zip-map2*[*of qs mpoly-sign val qs*]
by (*auto*)
have *Sturm-Tarski.sign i1* = *Sturm-Tarski.sign i*
using *i-sign i1-sign*
by *blast*
then show *?thesis* **using** *Atom.prems i-prop lookup-sem-M.simps*
using *Eq* ‹*lookup-assump-aux x1* (*zip qs signs*) = *Some i1*› *i1-sign i-sign* **by** *presburger*
next
case (*Leq x1*)
then obtain *i* **where** *i-prop*: *lookup-assump-aux x1* (*zip qs1 signs1*) = *Some i*
using *Atom.prems lookup-sem-M.simps*
by (*metis lookup-assump-aux-eo option.simps*(*3*) *option.simps*(*4*))
have *i-is*: (*x1*, *i*) $\in$ *set* (*zip qs1 signs1*)
using *i-prop lookup-assump-means-inset*[*of x1* (*zip qs1 signs1*) *i*]
in-set-member[*of* (*x1*, *i*) (*zip qs1 signs1*)] **by** *auto*
then have (*x1*, *i*) $\in$ *set* (*zip qs signs*)
using *subset* **by** *auto*
then obtain *i1* **where** *lookup-assump-aux x1* (*zip qs signs*) = *Some i1*
by (*meson inset-means-lookup-assump-some*)
then have *i1-is*: (*x1*, *i1*) $\in$ *set* (*zip qs signs*)
using *lookup-assump-means-inset*[*of x1* (*zip qs signs*) *i1*]
in-set-member[*of* (*x1*, *i1*) (*zip qs signs*)] **by** *auto*
have $\bigwedge b$. *signs1* = *map* (*mpoly-sign val*) *qs1* $\Longrightarrow$
zip qs1 (*map* (*mpoly-sign val*) *qs1*) =
map2 (λx *y*. (*x*, *mpoly-sign val y*)) *qs1 qs1* $\Longrightarrow$
(*x1*, *b*) $\in$ *set* (*zip qs1 qs1*) $\Longrightarrow$
i = *mpoly-sign val b* $\Longrightarrow$ *mpoly-sign val b* = *mpoly-sign val x1*
by (*simp add*: *zip-same*)
then have *i-sign* :*i* = *mpoly-sign val x1*
using *i-is* *Atom.prems*(*2*) *zip-map2*[*of qs1 mpoly-sign val qs1*]
by (*auto*)
have $\bigwedge b$. *signs* = *map* (*mpoly-sign val*) *qs* $\Longrightarrow$

zip qs (*map* (*mpoly-sign val*) *qs*) =
map2 (λ*x y*. (*x*, *mpoly-sign val y*)) *qs qs* ⟹
(*x1*, *b*) ∈ *set* (*zip qs qs*) ⟹
i1 = *mpoly-sign val b* ⟹ *mpoly-sign val b* = *mpoly-sign val x1*
by (*simp add*: *zip-same*)
then have *i1-sign*: *i1* = *mpoly-sign val x1*
using *i1-is Atom.prems*(*1*) *zip-map2*[*of qs mpoly-sign val qs*]
by (*auto*)
have *Sturm-Tarski.sign i1* = *Sturm-Tarski.sign i*
using *i-sign i1-sign*
by *blast*
then show *?thesis* **using** *Atom.prems i-prop lookup-sem-M.simps*
using *Leq* ‹*lookup-assump-aux x1* (*zip qs signs*) = *Some i1*› *i1-sign i-sign* **by**
presburger
next
case (*Neq x1*)
then obtain *i* **where** *i-prop*: *lookup-assump-aux x1* (*zip qs1 signs1*) = *Some i*
using *Atom.prems lookup-sem-M.simps*
by (*metis lookup-assump-aux-eo option.simps*(*3*) *option.simps*(*4*))
have *i-is*: (*x1*, *i*) ∈ *set* (*zip qs1 signs1*)
using *i-prop lookup-assump-means-inset*[*of x1* (*zip qs1 signs1*) *i*]
in-set-member[*of* (*x1*, *i*) (*zip qs1 signs1*)] **by** *auto*
then have (*x1*, *i*) ∈ *set* (*zip qs signs*)
using *subset* **by** *auto*
then obtain *i1* **where** *lookup-assump-aux x1* (*zip qs signs*) = *Some i1*
by (*meson inset-means-lookup-assump-some*)
then have *i1-is*: (*x1*, *i1*) ∈ *set* (*zip qs signs*)
using *lookup-assump-means-inset*[*of x1* (*zip qs signs*) *i1*]
in-set-member[*of* (*x1*, *i1*) (*zip qs signs*)] **by** *auto*
have ⋀*b*. *signs1* = *map* (*mpoly-sign val*) *qs1* ⟹
zip qs1 (*map* (*mpoly-sign val*) *qs1*) =
map2 (λ*x y*. (*x*, *mpoly-sign val y*)) *qs1 qs1* ⟹
(*x1*, *b*) ∈ *set* (*zip qs1 qs1*) ⟹
i = *mpoly-sign val b* ⟹ *mpoly-sign val b* = *mpoly-sign val x1*
by (*simp add*: *zip-same*)
then have *i-sign* :*i* = *mpoly-sign val x1*
using *i-is Atom.prems*(*2*) *zip-map2*[*of qs1 mpoly-sign val qs1*]
by (*auto*)
have ⋀*b*. *signs* = *map* (*mpoly-sign val*) *qs* ⟹
zip qs (*map* (*mpoly-sign val*) *qs*) =
map2 (λ*x y*. (*x*, *mpoly-sign val y*)) *qs qs* ⟹
(*x1*, *b*) ∈ *set* (*zip qs qs*) ⟹
i1 = *mpoly-sign val b* ⟹ *mpoly-sign val b* = *mpoly-sign val x1*
by (*simp add*: *zip-same*)
then have *i1-sign*: *i1* = *mpoly-sign val x1*
using *i1-is Atom.prems*(*1*) *zip-map2*[*of qs mpoly-sign val qs*]
by (*auto*)
have *Sturm-Tarski.sign i1* = *Sturm-Tarski.sign i*
using *i-sign i1-sign*

by *blast*
then show *?thesis* **using** *Atom.prems i-prop lookup-sem-M.simps*
using *Neq* ‹*lookup-assump-aux x1* (*zip qs signs*) = *Some i1*› *i1-sign i-sign* **by** *presburger*
qed
next
case (*And F1 F2*)
let *?sub-ell* = (*zip qs1 signs1*)
have *case-some*: *lookup-sem-M* (*fm.And F1 F2*) *?sub-ell* = (*case* (*lookup-sem-M F1 ?sub-ell*, *lookup-sem-M F2 ?sub-ell*)
of (*Some i*, *Some j*) ⇒ *Some* (*i* ∧ *j*)
| - ⇒ *None*)
using *lookup-sem-M.simps* **by** *simp*
have *e1*: ∃ *w1*. *lookup-sem-M F1 ?sub-ell* = *Some w1*
using *case-some And.prems*(*4*)
using *proper-interval-bool.elims*(*1*) **by** *fastforce*
have *e2*: ∃ *w2*. *lookup-sem-M F2 ?sub-ell* = *Some w2*
using *case-some And.prems*(*4*)
using *proper-interval-bool.elims*(*1*)
by (*smt* (*z3*) *option.case*(*1*) *option.simps*(*5*) *split-conv*)
then obtain *w1 w2* **where** *lookup-sem-M F1 ?sub-ell* = *Some w1*
lookup-sem-M F2 ?sub-ell = *Some w2*
using *e1 e2* **by** *auto*
have *ind1*: *lookup-sem-M F1* (*zip qs signs*) = *lookup-sem-M F1* (*zip qs1 signs1*)
using *e1 And.hyps*(*1*) *And.prems*(*1*−*3*) **by** *auto*
have *ind2*: *lookup-sem-M F2* (*zip qs signs*) = *lookup-sem-M F2* (*zip qs1 signs1*)
using *e2 And.hyps*(*2*) *And.prems*(*1*−*3*) **by** *auto*
then show *?case* **using** *lookup-sem-M.simps*
ind1 ind2
by *presburger*
next
case (*Or F1 F2*)
let *?sub-ell* = (*zip qs1 signs1*)
have *case-some*: *lookup-sem-M* (*fm.Or F1 F2*) *?sub-ell* = (*case* (*lookup-sem-M F1 ?sub-ell*, *lookup-sem-M F2 ?sub-ell*)
of (*Some i*, *Some j*) ⇒ *Some* (*i* ∨ *j*)
| - ⇒ *None*)
using *lookup-sem-M.simps* **by** *simp*
have *e1*: ∃ *w1*. *lookup-sem-M F1 ?sub-ell* = *Some w1*
using *case-some Or.prems*(*4*)
using *proper-interval-bool.elims*(*1*) **by** *fastforce*
have *e2*: ∃ *w2*. *lookup-sem-M F2 ?sub-ell* = *Some w2*
using *case-some Or.prems*(*4*)
using *proper-interval-bool.elims*(*1*)
by (*smt* (*z3*) *option.case*(*1*) *option.simps*(*5*) *split-conv*)
then obtain *w1 w2* **where** *lookup-sem-M F1 ?sub-ell* = *Some w1*
lookup-sem-M F2 ?sub-ell = *Some w2*
using *e1 e2* **by** *auto*
have *ind1*: *lookup-sem-M F1* (*zip qs signs*) = *lookup-sem-M F1* (*zip qs1 signs1*)

```
    using e1 Or.hyps(1) Or.prems(1−3) by auto
  have ind2: lookup-sem-M F2 (zip qs signs) = lookup-sem-M F2 (zip qs1 signs1)
    using e2 Or.hyps(2) Or.prems(1−3) by auto
  then show ?case using lookup-sem-M.simps
      ind1 ind2
    by presburger
next
  case (Neg F)
  then show ?case
    by (metis lookup-sem-M.simps(5) not-Some-eq option.case(1))
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 (metis add-Suc-right le-Suc-eq' le-add-same-cancel1 lookup-sem-M.simps(10)
lookup-sem-M.simps(14))
next
  case (AllN x1 F)
  then show ?case
    by (metis lookup-sem-M.simps(11) lookup-sem-M.simps(15) zero-list.cases)
qed

lemma extract-polys-semantics:
  assumes qs = extract-polys F
  assumes signs = map (mpoly-sign val) qs
  assumes countQuantifiers F = 0
  shows Some (eval F val) = lookup-sem-M F (zip qs signs)
  using assms
proof (induct F arbitrary: qs signs)
  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 x1)
    then have extract-polys (fm.Atom x) = [x1]
      using extract-polys.simps by auto
    then have qs-is: qs = [x1] using Atom.prems
      by auto
```

then have *signs-is*: *signs* = [*mpoly-sign val x1*]
using *Atom.prems* **by** *auto*
then have *zip-is*: *zip qs signs* = [(*x1*, *mpoly-sign val x1*)]
using *qs-is* **by** *auto*
then have *lookup-sem-is*: *lookup-sem-M* (*fm.Atom x*) (*zip qs signs*) =
(*case* (*lookup-assump-aux x1* (*zip qs signs*)) *of*
Some i ⇒ *Some* (*i* < *0*)
| - ⇒ *None*)**using** *Less*
by *auto*
have *lookup-assump-aux x1* (*zip qs signs*) = *Some* (*mpoly-sign val x1*)
using *zip-is* **by** *auto*
then have *lookup-is*: *lookup-sem-M* (*fm.Atom x*) (*zip qs signs*) = *Some* ((*mpoly-sign val x1*) < *0*)
using *lookup-sem-is* **by** *auto*
have *eval-is*: *eval* (*fm.Atom x*) *val* = (*insertion* (*nth-default 0 val*) *x1* < *0*)
using *Less* **by** *auto*
have (*mpoly-sign val x1*) < *0* = (*insertion* (*nth-default 0 val*) *x1* < *0*)
unfolding *mpoly-sign-def eval-mpoly-def Sturm-Tarski.sign-def*
by *auto*
then show *?thesis* **using** *eval-is lookup-is*
by *auto*
next
case (*Eq x1*)
then have *extract-polys* (*fm.Atom x*) = [*x1*]
using *extract-polys.simps* **by** *auto*
then have *qs-is*: *qs* = [*x1*] **using** *Atom.prems*
by *auto*
then have *signs-is*: *signs* = [*mpoly-sign val x1*]
using *Atom.prems* **by** *auto*
then have *zip-is*: *zip qs signs* = [(*x1*, *mpoly-sign val x1*)]
using *qs-is* **by** *auto*
then have *lookup-sem-is*: *lookup-sem-M* (*fm.Atom x*) (*zip qs signs*) =
(*case* (*lookup-assump-aux x1* (*zip qs signs*)) *of*
Some i ⇒ *Some* (*i* = *0*)
| - ⇒ *None*)**using** *Eq*
by *auto*
have *lookup-assump-aux x1* (*zip qs signs*) = *Some* (*mpoly-sign val x1*)
using *zip-is* **by** *auto*
then have *lookup-is*: *lookup-sem-M* (*fm.Atom x*) (*zip qs signs*) = *Some* ((*mpoly-sign val x1*) = *0*)
using *lookup-sem-is* **by** *auto*
have *eval-is*: *eval* (*fm.Atom x*) *val* = (*insertion* (*nth-default 0 val*) *x1* = *0*)
using *Eq* **by** *auto*
have (*mpoly-sign val x1*) = *0* = (*insertion* (*nth-default 0 val*) *x1* = *0*)
unfolding *mpoly-sign-def eval-mpoly-def Sturm-Tarski.sign-def*
by *auto*
then show *?thesis* **using** *eval-is lookup-is*
by *auto*
next

case (*Leq x1*)
then have *extract-polys* (*fm.Atom x*) = [*x1*]
using *extract-polys.simps* **by** *auto*
then have *qs-is*: *qs* = [*x1*] **using** *Atom.prems*
by *auto*
then have *signs-is*: *signs* = [*mpoly-sign val x1*]
using *Atom.prems* **by** *auto*
then have *zip-is*: *zip qs signs* = [(*x1*, *mpoly-sign val x1*)]
using *qs-is* **by** *auto*
then have *lookup-sem-is*: *lookup-sem-M* (*fm.Atom x*) (*zip qs signs*) =
(*case* (*lookup-assump-aux x1* (*zip qs signs*)) *of*
Some i ⇒ *Some* (*i* ≤ *0*)
| - ⇒ *None*)**using** *Leq*
by *auto*
have *lookup-assump-aux x1* (*zip qs signs*) = *Some* (*mpoly-sign val x1*)
using *zip-is* **by** *auto*
then have *lookup-is*: *lookup-sem-M* (*fm.Atom x*) (*zip qs signs*) = *Some* ((*mpoly-sign val x1*) ≤ *0*)
using *lookup-sem-is* **by** *auto*
have *eval-is*: *eval* (*fm.Atom x*) *val* = (*insertion* (*nth-default 0 val*) *x1* ≤ *0*)
using *Leq* **by** *auto*
have (*mpoly-sign val x1*) ≤ *0* = (*insertion* (*nth-default 0 val*) *x1* ≤ *0*)
unfolding *mpoly-sign-def eval-mpoly-def Sturm-Tarski.sign-def*
by *auto*
then show *?thesis* **using** *eval-is lookup-is*
by *auto*
next
case (*Neq x1*)
then have *extract-polys* (*fm.Atom x*) = [*x1*]
using *extract-polys.simps* **by** *auto*
then have *qs-is*: *qs* = [*x1*] **using** *Atom.prems*
by *auto*
then have *signs-is*: *signs* = [*mpoly-sign val x1*]
using *Atom.prems* **by** *auto*
then have *zip-is*: *zip qs signs* = [(*x1*, *mpoly-sign val x1*)]
using *qs-is* **by** *auto*
then have *lookup-sem-is*: *lookup-sem-M* (*fm.Atom x*) (*zip qs signs*) =
(*case* (*lookup-assump-aux x1* (*zip qs signs*)) *of*
Some i ⇒ *Some* (*i* ≠ *0*)
| - ⇒ *None*)**using** *Neq*
by *auto*
have *lookup-assump-aux x1* (*zip qs signs*) = *Some* (*mpoly-sign val x1*)
using *zip-is* **by** *auto*
then have *lookup-is*: *lookup-sem-M* (*fm.Atom x*) (*zip qs signs*) = *Some* ((*mpoly-sign val x1*) ≠ *0*)
using *lookup-sem-is* **by** *auto*
have *eval-is*: *eval* (*fm.Atom x*) *val* = (*insertion* (*nth-default 0 val*) *x1* ≠ *0*)
using *Neq* **by** *auto*
have (*mpoly-sign val x1*) ≠ *0* = (*insertion* (*nth-default 0 val*) *x1* ≠ *0*)

```
      unfolding mpoly-sign-def eval-mpoly-def Sturm-Tarski.sign-def
      by auto
    then show ?thesis using eval-is lookup-is
      by auto
  qed
next
  case (And F1 F2)
  have qs-is: qs = (extract-polys F1)@(extract-polys F2)
    using And.prems(1) extract-polys.simps
    by force
  then obtain signs1 signs2 where signs-prop:
    signs1 = map (mpoly-sign val) (extract-polys F1)
    signs2 = map (mpoly-sign val) (extract-polys F2)
    signs = signs1 @ signs2
    using And.prems(2) by auto
  have subset-f1: set (extract-polys F1) ⊆ set qs
    using qs-is by auto
  have subset-f2: set (extract-polys F2) ⊆ set qs
    using qs-is by auto
  have f1-free: countQuantifiers F1 = 0
    using And.prems(3)
    by auto
  have f2-free: countQuantifiers F2 = 0
    using And.prems(3)
    by auto
  have zip-is: (zip qs signs) = (zip (extract-polys F1) signs1)@(zip (extract-polys
F2) signs2)
    using qs-is signs-prop(3)
    by (simp add: signs-prop(1))
  have ind1: Some (eval F1 val) = lookup-sem-M F1 (zip (extract-polys F1) signs1)
    using And.hyps(1) signs-prop(1) f1-free by auto
  have subset: set (zip (extract-polys F1) signs1) ⊆ set (zip qs signs)
    using subset-zip-is-subset And.prems signs-prop(1) by auto
  then have lookup1: lookup-sem-M F1 (zip (extract-polys F1) signs1) = lookup-sem-M
F1 (zip qs signs)
    using extract-polys-subset[of signs val qs signs1 extract-polys F1] ind1 subset-f1
signs-prop(1) qs-is And.prems(2)
    by auto
  then have restate-ind1: lookup-sem-M F1 (zip qs signs) = Some (eval F1 val)
    using ind1 lookup1
    by auto
  have ind2: Some (eval F2 val) = lookup-sem-M F2 (zip (extract-polys F2) signs2)
    using And.hyps(2) signs-prop(2) f2-free by auto
  then have lookup2: lookup-sem-M F2 (zip (extract-polys F2) signs2) = lookup-sem-M
F2 (zip qs signs)
    using extract-polys-subset[of signs val qs signs2 extract-polys F2] ind2 subset-f2
signs-prop(2) And.prems(2)
    by auto
  then have restate-ind2: lookup-sem-M F2 (zip qs signs) = Some (eval F2 val)
```

using *ind2 lookup2*
by *auto*
show *?case* **using** *signs-prop*(*3*)
restate-ind1 restate-ind2
qs-is zip-is lookup-sem-M.simps
by *simp*
next
case (*Or F1 F2*)
have *qs-is*: *qs* = (*extract-polys F1*)@(*extract-polys F2*)
using *Or.prems*(*1*) *extract-polys.simps*
by *force*
then obtain *signs1 signs2* **where** *signs-prop*:
signs1 = *map* (*mpoly-sign val*) (*extract-polys F1*)
signs2 = *map* (*mpoly-sign val*) (*extract-polys F2*)
signs = *signs1* @ *signs2*
using *Or.prems*(*2*) **by** *auto*
have *subset-f1*: *set* (*extract-polys F1*) ⊆ *set qs*
using *qs-is* **by** *auto*
have *subset-f2*: *set* (*extract-polys F2*) ⊆ *set qs*
using *qs-is* **by** *auto*
have *f1-free*: *countQuantifiers F1* = *0*
using *Or.prems*(*3*)
by *auto*
have *f2-free*: *countQuantifiers F2* = *0*
using *Or.prems*(*3*)
by *auto*
have *zip-is*: (*zip qs signs*) = (*zip* (*extract-polys F1*) *signs1*)@(*zip* (*extract-polys F2*) *signs2*)
using *qs-is signs-prop*(*3*)
by (*simp add*: *signs-prop*(*1*))
have *ind1*: *Some* (*eval F1 val*) = *lookup-sem-M F1* (*zip* (*extract-polys F1*) *signs1*)
using *Or.hyps*(*1*) *signs-prop*(*1*) *f1-free* **by** *auto*
have *subset*: *set* (*zip* (*extract-polys F1*) *signs1*) ⊆ *set* (*zip qs signs*)
using *subset-zip-is-subset Or.prems signs-prop*(*1*) **by** *auto*
then have *lookup1*: *lookup-sem-M F1* (*zip* (*extract-polys F1*) *signs1*) = *lookup-sem-M F1* (*zip qs signs*)
using *extract-polys-subset*[*of signs val qs signs1 extract-polys F1*] *ind1 subset-f1 signs-prop*(*1*) *qs-is Or.prems*(*2*)
by *auto*
then have *restate-ind1*: *lookup-sem-M F1* (*zip qs signs*) = *Some* (*eval F1 val*)
using *ind1 lookup1*
by *auto*
have *ind2*: *Some* (*eval F2 val*) = *lookup-sem-M F2* (*zip* (*extract-polys F2*) *signs2*)
using *Or.hyps*(*2*) *signs-prop*(*2*) *f2-free* **by** *auto*
then have *lookup2*: *lookup-sem-M F2* (*zip* (*extract-polys F2*) *signs2*) = *lookup-sem-M F2* (*zip qs signs*)
using *extract-polys-subset*[*of signs val qs signs2 extract-polys F2*] *ind2 subset-f2 signs-prop*(*2*) *Or.prems*(*2*)
by *auto*

```
  then have restate-ind2: lookup-sem-M F2 (zip qs signs) = Some (eval F2 val)
    using ind2 lookup2
    by auto
  show ?case using signs-prop(3)
      restate-ind1 restate-ind2
      qs-is zip-is lookup-sem-M.simps
    by simp
next
  case (Neg F)
  have qs-is: qs = (extract-polys F)
    using Neg.prems(1) extract-polys.simps
    by force
  have cq-zer: countQuantifiers F = 0
    using Neg.prems(3) by auto
  have Some (eval F val) = lookup-sem-M F (zip qs signs)
    using qs-is cq-zer Neg.hyps[of qs signs] Neg.prems(2)
    by auto
  then show ?case using lookup-sem-M.simps
    by (smt (verit, best) eval.simps(6) option.simps(5))
next
  case (ExQ F)
  have countQuantifiers (ExQ F) > 0
    by auto
  then show ?case using ExQ.prems(3) by auto
next
  case (AllQ F)
  have countQuantifiers (AllQ F) > 0
    by auto
  then show ?case using AllQ.prems(3) by auto
next
  case (ExN x1 F)
  {assume *: x1 = 0
    then have h1: eval (ExN x1 F) val = eval F val
      by auto
    have h2: lookup-sem-M (ExN x1 F) (zip qs signs)
        = lookup-sem-M F (zip qs signs)
      using * by auto
    have qs-is: qs = extract-polys F using * ExN.prems(1)
        extract-polys.simps by auto
    have countQuantifiers F = 0
      using ExN.prems(3) by auto
    then have Some (eval (ExN x1 F) val) =
       lookup-sem-M (ExN x1 F) (zip qs signs)
      using qs-is h1 h2 ExN.hyps[of qs signs] ExN.prems(2) by auto
  }
  moreover {assume *: x1 > 0
    then have countQuantifiers (ExN x1 F) > 0
      by auto
    then have Some (eval (ExN x1 F) val) =
```

```
      lookup-sem-M (ExN x1 F) (zip qs signs)
     using ExN.prems(3) by auto
  }
  ultimately show ?case
    by fastforce
next
  case (AllN x1 F)
  {assume *: x1 = 0
    then have h1: eval (AllN x1 F) val = eval F val
      by auto
    have h2: lookup-sem-M (AllN x1 F) (zip qs signs)
        = lookup-sem-M F (zip qs signs)
      using * by auto
    have qs-is: qs = extract-polys F using * AllN.prems(1)
        extract-polys.simps by auto
    have countQuantifiers F = 0
      using AllN.prems(3) by auto
    then have Some (eval (AllN x1 F) val) =
       lookup-sem-M (AllN x1 F) (zip qs signs)
      using qs-is h1 h2 AllN.hyps[of qs signs] AllN.prems(2) by auto
  }
  moreover {assume *: x1 > 0
    then have countQuantifiers (AllN x1 F) > 0
      by auto
    then have Some (eval (AllN x1 F) val) =
       lookup-sem-M (AllN x1 F) (zip qs signs)
      using AllN.prems(3) by auto
  }
  ultimately show ?case
    by fastforce
qed

lemma create-disjunction-eval:
  assumes eval (create-disjunction data) xs
  shows ∃ a ∈ set data. (eval (assump-to-atom-fm (fst a)) xs)
  using assms
proof (induct data)
  case Nil
  then show ?case by auto
next
  case (Cons a data)
  then show ?case
   by (metis create-disjunction.elims eval-Or fst-conv list.set-intros(1) list.set-intros(2)
list.simps(1) list.simps(3))
qed

lemma assump-to-atom-fm-conjunction:
  assumes eval (assump-to-atom-fm assumps) xs
  shows (p, n) ∈ set assumps ⟹ satisfies-evaluation xs p n
```

```
  using assms
proof (induct assumps)
  case Nil
  then show ?case by auto
next
  case (Cons a assumps)
  then have eval-prop: eval (assump-to-atom-fm assumps) xs
    eval (Atom (assump-to-atom ((fst a), (snd a)))) xs
     apply (metis assump-to-atom-fm.simps(2) eval.simps(4) old.prod.exhaust)
   by (metis Cons.prems(2) assump-to-atom-fm.simps(2) eval.simps(4) prod.exhaust-sel)
  {assume ∗ : (p, n) = a
   have aEval (if n = 0 then atom.Eq p else if n < 0 then Less p else Less (− p))
    xs ⟹ Sturm-Tarski.sign (eval-mpoly xs p) = Sturm-Tarski.sign n
    proof −
      assume eval-h: aEval (if n = 0 then atom.Eq p else if n < 0 then Less p else
Less (− p))
    xs
      {assume ∗∗: n = 0
        then have Sturm-Tarski.sign (eval-mpoly xs p) = Sturm-Tarski.sign n
          using eval-h unfolding eval-mpoly-def by auto
      } moreover
      {assume ∗∗: n > 0
        then have Sturm-Tarski.sign (eval-mpoly xs p) = Sturm-Tarski.sign n
          using eval-h unfolding eval-mpoly-def by auto
      } moreover
      {assume ∗∗: n < 0
        then have Sturm-Tarski.sign (eval-mpoly xs p) = Sturm-Tarski.sign n
          using eval-h unfolding eval-mpoly-def by auto
      }
      ultimately show Sturm-Tarski.sign (eval-mpoly xs p) = Sturm-Tarski.sign
n
        using linorder-cases by blast
    qed
    then have satisfies-evaluation xs p n
      using eval-prop(2) ∗ unfolding satisfies-evaluation-def
      by auto
  }
  moreover {assume ∗: (p, n) ∈ set assumps
    then have satisfies-evaluation xs p n
      using eval-prop(1) Cons.hyps by auto
  }
  ultimately show ?case using Cons.prems(1) by auto
qed

lemma eval-elim-forall-correct1:
  fixes F:: atom fm
  assumes ∗: countQuantifiers F = 0
  assumes eval (elim-forall F) xs
  shows (eval F (x#xs))
```

proof −
  let *?qs* = *extract-polys F*
  let *?univ-qs* = *univariate-in ?qs 0*
  let *?reindexed-univ-qs* = *map* (*map-poly* (*lowerPoly 0 1*)) *?univ-qs*
  let *?data* = *sign-determination-inner ?reindexed-univ-qs* []
  let *?new-data* = *filter* (λ(*assumps*, *signs-list*).
      list-all (λ *signs*.
      let paired-signs = *zip ?qs signs in*
        lookup-sem-M F paired-signs = (*Some True*))
      signs-list
) *?data*
  have *elim-forall F* = *create-disjunction ?new-data*
    unfolding *elim-forall.simps Let-def*
    by *fastforce*
  then have *eval* (*create-disjunction ?new-data*) *xs*
    using *assms*
    by *presburger*
  then obtain *a* **where** *a-prop1*: *a* ∈ *set ?new-data*
    (*eval* (*assump-to-atom-fm* (*fst a*)) *xs*)
    using *create-disjunction-eval*
    by *blast*
  then have *a-prop2*: *a* ∈ *set ?data*
    (*list-all* (λ *signs*.
      let paired-signs = *zip ?qs signs in*
        lookup-sem-M F paired-signs = (*Some True*))
      (*snd a*))
     apply (*smt* (*verit*, *best*) *mem-Collect-eq set-filter*)
    by (*smt* (*verit*, *del-insts*) *a-prop1*(*1*) *mem-Collect-eq set-filter split-def*)
  have *a-in*: (*fst a*, *snd a*) ∈ *set* (*sign-determination-inner ?reindexed-univ-qs* [])
    using *a-prop2*(*1*) **by** *auto*
  have ⋀*p n*. (*p*,*n*) ∈ *set* (*fst a*) ⟹ *satisfies-evaluation xs p n*
    using *a-prop1*(*2*) *assump-to-atom-fm-conjunction*
    by *blast*
  from *sign-determination-inner-gives-noncomp-signs-at-roots*[*OF a-in this*]
  have *set* (*snd a*) =
    consistent-sign-vectors-R (*map* (*eval-mpoly-poly xs*) *?reindexed-univ-qs*) *UNIV*
    by *auto*
 then have *csv*: (*consistent-sign-vec* (*map* (*eval-mpoly-poly xs*) *?reindexed-univ-qs*))
x ∈ *set* (*snd a*)
    unfolding *consistent-sign-vectors-R-def* **by** *auto*

 have *map-rel1*: *map* (*Sturm-Tarski.sign*) (*map* (λ*p*. *poly p x*) (*map* (*eval-mpoly-poly*
xs) (*map* (*map-poly* (*lowerPoly 0 1*)) (*univariate-in* (*extract-polys F*) *0*))))
= (*consistent-sign-vec* (*map* (*eval-mpoly-poly xs*) *?reindexed-univ-qs*)) *x*
    unfolding *consistent-sign-vec-def Sturm-Tarski.sign-def*
    by *auto*
  then have *map-rel2*: *map* (*eval-mpoly* (*x* # *xs*)) (*extract-polys F*) =
    map (λ*p*. *poly p x*) (*map* (*eval-mpoly-poly xs*) (*map* (*map-poly* (*lowerPoly 0 1*))
(*univariate-in* (*extract-polys F*) *0*)))

using *reindexed-univ-qs-eval*[*of* - (*extract-polys F*)] **by** *auto*
then have *map-rel3*: *map Sturm-Tarski.sign* (*map* (*eval-mpoly* (*x* # *xs*)) (*extract-polys F*)) =
map (*Sturm-Tarski.sign*) (*map* (λ*p. poly p x*) (*map* (*eval-mpoly-poly xs*) (*map* (*map-poly* (*lowerPoly 0 1*)) (*univariate-in* (*extract-polys F*) *0*))))
by *auto*
have *map-rel4*: *map Sturm-Tarski.sign* (*map* (*eval-mpoly* (*x* # *xs*)) (*extract-polys F*)) = *map* (*mpoly-sign* (*x*#*xs*)) *?qs*
unfolding *mpoly-sign-def* **by** *auto*
have *map* (*mpoly-sign* (*x*#*xs*)) *?qs* =(*consistent-sign-vec* (*map* (*eval-mpoly-poly xs*) *?reindexed-univ-qs*)) *x*
using *map-rel1 map-rel2 map-rel3 map-rel4*
by (*metis list.inj-map-strong of-rat-hom.injectivity*)

then obtain *signs* **where** *signs-prop*: *signs* ∈ *set* (*snd a*)
signs = *map* (*mpoly-sign* (*x*#*xs*)) *?qs*
using *csv* **by** *auto*
then have *lookup-sem-match*: *Some* (*eval F* (*x*#*xs*)) = *lookup-sem-M F* (*zip ?qs signs*)
using ∗ *extract-polys-semantics* **by** *auto*
have *lookup-sem-M F* (*zip ?qs signs*) = *Some* (*True*)
using *a-prop2*(*2*) *signs-prop*(*1*) **unfolding** *Let-def*
by (*smt* (*verit*, *ccfv-SIG*) *lookup-sem-match a-prop2*(*2*) *list.pred-mono-strong mem-Collect-eq option.simps*(*1*) *set-filter subset-code*(*1*) *subset-set-code*)
then show *?thesis*
using *lookup-sem-match* **by** *auto*
qed

lemma *assump-to-atom-fm-eval*:
assumes ⋀*p n*. (*p*,*n*) ∈ *set assumps* ⟹ *satisfies-evaluation xs p n*
shows (*eval* (*assump-to-atom-fm assumps*) *xs*)
using *assms*
proof (*induct assumps*)
case *Nil*
then show *?case* **by** *auto*
next
case (*Cons a assumps*)
then have *h1*: *eval* (*assump-to-atom-fm assumps*) *xs*
by *simp*
have *sat-eval*: *satisfies-evaluation xs* (*fst a*) (*snd a*)
using *Cons.prems* **by** *auto*
have *h2*: *eval* (*Atom* (*assump-to-atom* ((*fst a*), (*snd a*)))) *xs*
proof −
{**assume** ∗: *snd a* = *0*
then have *eval* (*Atom* (*assump-to-atom* ((*fst a*), (*snd a*)))) *xs*
using *sat-eval* **unfolding** *satisfies-evaluation-def eval-mpoly-def Sturm-Tarski.sign-def*
by (*smt* (*verit*, *del-insts*) *aEval.simps*(*1*) *assump-to-atom.simps eval.simps*(*1*) *less-numeral-extra*(*3*) *of-int-hom.eq-iff*)

```
    } moreover
    {assume *: 0 < snd a
      then have (eval-mpoly xs (fst a)) > 0
        using sat-eval unfolding satisfies-evaluation-def Sturm-Tarski.sign-def
        by (smt (verit, del-insts) of-int-hom.eq-iff)

      then have aEval (Less (−(fst a))) xs
        using * unfolding eval-mpoly-def by auto
      then have  aEval (assump-to-atom a) xs
        using * assump-to-atom.simps[of fst a snd a] by auto
      then have eval (Atom (assump-to-atom ((fst a), (snd a)))) xs
        by auto
    } moreover
    {assume *: 0 > snd a
      then have (eval-mpoly xs (fst a)) < 0
        using sat-eval unfolding satisfies-evaluation-def Sturm-Tarski.sign-def
        by (smt (z3) of-int-hom.eq-iff sign-simps(1) sign-simps(3))

      then have aEval (Less ((fst a))) xs
        using * unfolding eval-mpoly-def by auto
      then have  aEval (assump-to-atom a) xs
        using * assump-to-atom.simps[of fst a snd a] by auto
      then have eval (Atom (assump-to-atom ((fst a), (snd a)))) xs
        by auto
    }
    ultimately show ?thesis
      by linarith
  qed
  then show ?case
    using h1 h2
    by (metis assump-to-atom-fm.simps(2) eval.simps(4) prod.exhaust-sel)
qed

lemma create-disjunction-true:
  assumes (assumps, signs) ∈ set data
  assumes  (eval (assump-to-atom-fm assumps) xs)
  shows eval (create-disjunction data) xs
  using assms
proof (induct data)
  case Nil
  then show ?case by auto
next
  case (Cons a data)
  then show ?case
   by (metis create-disjunction.simps(2) eval.simps(5) prod.exhaust-sel set-ConsD)

qed
```

lemma *eval-elim-forall-correct2*:
  fixes *F*:: *atom fm*
  assumes *countQuantifiers F* = *0*
  assumes (∀ *x*. (*eval F* (*x*#*xs*)))
  shows *eval* (*elim-forall F*) *xs*
proof −
  let *?qs* = *extract-polys F*
  let *?univ-qs* = *univariate-in ?qs 0*
  let *?reindexed-univ-qs* = *map* (*map-poly* (*lowerPoly 0 1*)) *?univ-qs*
  let *?data* = *sign-determination-inner ?reindexed-univ-qs* []
  let *?new-data* = *filter* (λ(*assumps*, *signs-list*).
    list-all (λ *signs*.
    let paired-signs = *zip ?qs signs in*
      lookup-sem-M F paired-signs = (*Some True*))
    signs-list
) *?data*
  have *h1*: *elim-forall F* = *create-disjunction ?new-data*
    unfolding *elim-forall.simps Let-def*
    by *fastforce*
  have ⋀*p n*. (*p*,*n*) ∈ *set* [] ⟹ *satisfies-evaluation xs p n*
    by *auto*

  then obtain *assumps signs* **where** *assumps-signs*: (*assumps*, *signs*) ∈ *set ?data*
    ⋀*p n*. (*p*,*n*) ∈ *set assumps* ⟹ *satisfies-evaluation xs p n*
    using *get-all-valuations*[*of* [] *xs*]
    by *blast*
  then have *assump-to-atom-eval*: (*eval* (*assump-to-atom-fm assumps*) *xs*)
    using *assump-to-atom-fm-eval*
    by *blast*
  then have *a-in*: (*assumps*, *signs*) ∈ *set* (*sign-determination-inner ?reindexed-univ-qs*
[])
    using *assumps-signs*(*1*)
    by *auto*
  from *sign-determination-inner-gives-noncomp-signs-at-roots*[*OF a-in assumps-signs*(*2*)]
  have *signs-is-csvs*: *set signs* =
    consistent-sign-vectors-R (*map* (*eval-mpoly-poly xs*) *?reindexed-univ-qs*) *UNIV*
    by *auto*

  have ⋀ *sign*. *sign* ∈ *set signs* ⟹
      lookup-sem-M F (*zip ?qs sign*) = (*Some True*)
  proof −
    fix *sign*
    assume *sign* ∈ *set signs*
    have ∃ (*x*::*real*). *sign* = *consistent-sign-vec* (*map* (*eval-mpoly-poly xs*) *?reindexed-univ-qs*)
x
      using *signs-is-csvs* **unfolding** *consistent-sign-vectors-R-def*
      using ‹*sign* ∈ *set signs*› **by** *auto*
    then obtain *x* **where** *x-prop*: *sign* = *consistent-sign-vec* (*map* (*eval-mpoly-poly*
xs) *?reindexed-univ-qs*) *x*

by *auto*
have *map-rel1*: *map* (*Sturm-Tarski.sign*) (*map* (*λp. poly p x*) (*map* (*eval-mpoly-poly xs*) (*map* (*map-poly* (*lowerPoly 0 1*)) (*univariate-in* (*extract-polys F*) *0*))))
= (*consistent-sign-vec* (*map* (*eval-mpoly-poly xs*) *?reindexed-univ-qs*)) *x*
unfolding *consistent-sign-vec-def Sturm-Tarski.sign-def*
by *auto*
then have *map-rel2*: *map* (*eval-mpoly* (*x* # *xs*)) (*extract-polys F*) =
map (*λp. poly p x*) (*map* (*eval-mpoly-poly xs*) (*map* (*map-poly* (*lowerPoly 0 1*)) (*univariate-in* (*extract-polys F*) *0*)))
using *reindexed-univ-qs-eval*[*of* - (*extract-polys F*)] **by** *auto*
then have *map-rel3*: *map Sturm-Tarski.sign* (*map* (*eval-mpoly* (*x* # *xs*)) (*extract-polys F*)) =
map (*Sturm-Tarski.sign*) (*map* (*λp. poly p x*) (*map* (*eval-mpoly-poly xs*) (*map* (*map-poly* (*lowerPoly 0 1*)) (*univariate-in* (*extract-polys F*) *0*))))
by *auto*
have *map-rel4*: *map Sturm-Tarski.sign* (*map* (*eval-mpoly* (*x* # *xs*)) (*extract-polys F*)) = *map* (*mpoly-sign* (*x#xs*)) *?qs*
unfolding *mpoly-sign-def* **by** *auto*
have *map* (*mpoly-sign* (*x#xs*)) *?qs* =(*consistent-sign-vec* (*map* (*eval-mpoly-poly xs*) *?reindexed-univ-qs*)) *x*
using *map-rel1 map-rel2 map-rel3 map-rel4*
by (*metis* (*no-types*, *lifting*) *list.inj-map-strong of-rat-hom.injectivity*)

then have *sign* = *map* (*mpoly-sign* (*x#xs*)) *?qs*
using *x-prop* **by** *auto*
then have *Some* (*eval F* (*x#xs*)) = *lookup-sem-M F* (*zip ?qs sign*)
using *assms*(*1*) *extract-polys-semantics* **by** *auto*
then show *lookup-sem-M F* (*zip ?qs sign*) = (*Some True*)
using *assms*(*2*) **by** *auto*
qed
then have *list-all* (*λ sign*.
let paired-signs = *zip ?qs sign in*
lookup-sem-M F paired-signs = (*Some True*))
signs
by (*meson Ball-set-list-all*)
then have *h2*: (*assumps*, *signs*) ∈ *set ?new-data*
using *assumps-signs*(*1*)
by (*smt* (*verit*, *del-insts*) *case-prod-conv mem-Collect-eq set-filter*)

show *?thesis*
using *h1 h2 assump-to-atom-eval create-disjunction-true*
by *presburger*
qed

lemma *eval-elim-forall-correct*:
fixes *F*:: *atom fm*
assumes *countQuantifiers F* = *0*
shows (∀ *x*. (*eval F* (*x#xs*))) = *eval* (*elim-forall F*) *xs*
using *eval-elim-forall-correct2 eval-elim-forall-correct1*

using *assms* **by** *blast*

theorem *elim-forall-correct*:
fixes *F*:: *atom fm*
assumes *countQuantifiers F* = *0*
shows *eval* (*AllQ F*) *xs* = *eval* (*elim-forall F*) *xs*
using *eval-elim-forall-correct*
using *assms eval.simps*(*8*) **by** *blast*

lemma *elim-exists-correct*:
fixes *F*:: *atom fm*
assumes *countQuantifiers F* = *0*
shows *eval* (*ExQ F*) *xs* = *eval* (*elim-exist F*) *xs*
using *elim-forall-correct*
by (*metis* (*no-types*, *opaque-lifting*) *assms countQuantifiers.simps*(*6*) *elim-exist-def eval.simps*(*6*) *eval.simps*(*7*) *eval.simps*(*8*))

17.12 Correctness of QE

lemma *assump-to-atom-no-quantifiers*:
shows *countQuantifiers* (*assump-to-atom-fm a*) = *0*
proof (*induct a*)
case *Nil*
then show *?case* **by** *auto*
next
case (*Cons a1 a2*)
then show *?case*
using *countQuantifiers.simps*
by (*smt* (*verit*, *best*) *add.right-neutral assump-to-atom-fm.elims list.simps*(*1*))
qed

lemma *create-disjunction-no-quantifiers*:
shows *countQuantifiers* (*create-disjunction ell*) = *0*
proof (*induct ell*)
case *Nil*
then show *?case* **by** *auto*
next
case (*Cons a ell*)
then show *?case*
using *assump-to-atom-no-quantifiers*
by (*metis add.right-neutral countQuantifiers.simps*(*5*) *create-disjunction.elims list.simps*(*1*) *list.simps*(*3*))
qed

lemma *elim-forall-no-quantifiers*:
fixes *F*:: *atom fm*
shows *countQuantifiers* (*elim-forall F*) = *0*
using *create-disjunction-no-quantifiers*

by (*metis elim-forall.simps*)

lemma *elim-exists-no-quantifiers*:
fixes *F*:: *atom fm*
shows *countQuantifiers* (*elim-exist F*) = *0*
using *elim-forall-no-quantifiers*
using *countQuantifiers.simps*(*6*) *elim-exist-def* **by** *presburger*

lemma *qe-removes-quantifiers*:
shows *countQuantifiers* (*qe F*) = *0*
proof (*induct F*)
case *TrueF*
then show *?case* **by** *auto*
next
case *FalseF*
then show *?case* **by** *auto*
next
case (*Atom x*)
then show *?case* **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*)
then show *?case*
using *elim-exists-no-quantifiers qe.simps*(*7*) **by** *presburger*
next
case (*AllQ F*)
then show *?case*
using *elim-forall-no-quantifiers*
using *qe.simps*(*8*) **by** *presburger*
next
case (*ExN n F*)
then show *?case*
proof (*induct n*)
case *0*
then show *?case*
by *auto*
next
case (*Suc n*)
then show *?case*
using *elim-exists-no-quantifiers*
by *simp*

qed
next
case (*AllN n F*)
then show *?case*
proof (*induct n*)
case *0*
then show *?case*
by *auto*
next
case (*Suc n*)
then show *?case*
using *elim-forall-no-quantifiers*
by (*metis funpow.simps*(*2*) *o-apply qe.simps*(*9*))
qed
qed

lemma *elim-exist-N-correct*:
assumes *countQuantifiers F* = *0*
shows *eval* (*ExN n F*) *xs* = *eval* ((*elim-exist* ^^ *n*) *F*) *xs*
using *assms*
proof (*induct n arbitrary*: *xs F*)
case *0*
then show *?case*
by *auto*
next
case (*Suc n*)
have *eval* (*ExN* (*Suc n*) *F*) *xs* = *eval* (*ExN n* (*ExQ F*)) *xs*
using *unwrapExist′* **by** *auto*
moreover have ... = *eval* (*ExN n* (*elim-exist F*)) *xs*
using *elim-exists-correct Suc.prems*
by *auto*
moreover have ... = *eval* ((*elim-exist* ^^ *n*) (*elim-exist F*)) *xs*
using *Suc.hyps*[*of elim-exist F*] *elim-exists-no-quantifiers*
by *auto*
moreover have ... = *eval* ((*elim-exist* ^^ (*n*+*1*)) *F*) *xs*
using *funpow-Suc-right* **unfolding** *o-def*
by (*smt* (*verit*, *best*) *o-apply semiring-norm*(*174*))
ultimately show *?case*
by (*metis semiring-norm*(*174*))
qed

lemma *elim-all-N-correct*:
assumes *countQuantifiers F* = *0*
shows *eval* (*AllN n F*) *xs* = *eval* ((*elim-forall* ^^ *n*) *F*) *xs*
using *assms*
proof (*induct n arbitrary*: *xs F*)
case *0*
then show *?case*
by (*metis clearQuantifiers.simps*(*11*) *funpow-0 opt′*)

```
next
  case (Suc n)
  have eval (AllN (Suc n) F) xs = eval (AllN n (AllQ F)) xs
    using  unwrapForall' by auto
  moreover have ... =  eval (AllN n (elim-forall F)) xs
    using elim-forall-correct Suc.prems
    using eval.simps(9) by presburger
  moreover have ... =  eval ((elim-forall ^^ n) (elim-forall F)) xs
    using Suc.hyps[of  elim-forall F] elim-forall-no-quantifiers
    by presburger
  moreover have ... =  eval ((elim-forall ^^ (n+1)) F) xs
    using funpow-Suc-right unfolding o-def
    by (smt (verit, best) o-apply semiring-norm(174))
  ultimately show ?case
    by (metis semiring-norm(174))
qed

theorem qe-correct:
  fixes F:: atom fm
  shows eval F xs = eval (qe F) xs
proof (induct F arbitrary: xs)
  case TrueF
  then show ?case by auto
next
  case FalseF
  then show ?case by auto
next
  case (Atom x)
  then show ?case by auto
next
  case (And F1 F2)
  then show ?case
    using eval.simps
    by auto
next
  case (Or F1 F2)
  then show ?case
    using eval.simps
    by auto
next
  case (Neg F)
  then show ?case
    using eval.simps
    by auto
next
  case (ExQ F)
  have countQuantifiers (qe F) = 0
    using qe-removes-quantifiers by auto
  from elim-exists-correct[OF this]
```

have *h*: *eval* (*ExQ* (*qe F*)) *xs* = *eval* (*elim-exist* (*qe F*)) *xs*
.
have *eval* (*ExQ F*) *xs* = *eval* (*ExQ* (*qe F*)) *xs*
using *ExQ.hyps*
by *simp*
then show *?case*
using *h*
by *auto*
next
case (*AllQ F*)
have *countQuantifiers* (*qe F*) = *0*
using *qe-removes-quantifiers* **by** *auto*
from *elim-forall-correct*[*OF this*]
have *h*: *eval* (*AllQ* (*qe F*)) *xs* = *eval* (*elim-forall* (*qe F*)) *xs*
.
have *eval* (*AllQ F*) *xs* = *eval* (*AllQ* (*qe F*)) *xs*
using *AllQ.hyps*
by *simp*
then show *?case*
using *h*
using *qe.simps*(*8*) **by** *presburger*
next
case (*ExN n F*)
have *cq*: *countQuantifiers* (*qe F*) = *0*
using *qe-removes-quantifiers* **by** *auto*
have *h*: *eval* (*ExN n F*) *xs* = *eval* (*ExN n* (*qe F*)) *xs*
using *ExN.hyps*
by *simp*
have *eval* (*ExN n* (*qe F*)) *xs* = *eval* ((*elim-exist* ^^ *n*) (*qe F*)) *xs*
using *elim-exist-N-correct*[*OF cq*] **.**
then show *?case*
using *h* **by** *auto*
next
case (*AllN n F*)
have *cq*: *countQuantifiers* (*qe F*) = *0*
using *qe-removes-quantifiers* **by** *auto*
have *h*: *eval* (*AllN n F*) *xs* = *eval* (*AllN n* (*qe F*)) *xs*
using *AllN.hyps*
by *simp*
have *eval* (*AllN n* (*qe F*)) *xs* = *eval* ((*elim-forall* ^^ *n*) (*qe F*)) *xs*
using *elim-all-N-correct*[*OF cq*] **.**
then show *?case*
using *h* **by** *auto*
qed

theorem *qe-extended-correct*:
fixes *F*:: *atom fm*
shows *eval F xs* = *eval* (*qe-with-VS F*) *xs*
using *qe-correct VSLEG*

by (*simp add*: *qe-with-VS-def*)

end

Acknowledgements

This material is based upon work supported by the National Science Foundation under Grant No. CNS-1739629, a National Science Foundation Graduate Research Fellowship under Grants Nos. DGE1252522 and DGE1745016, by the AFOSR under grant number FA9550-16-1-0288, by A*STAR, Singapore, and the Alexander von Humboldt Foundation. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the National Science Foundation, AFOSR, or A*STAR.

References

[1] S. Basu, R. Pollack, and M.-F. Roy. *Algorithms in Real Algebraic Geometry.* Springer, Berlin, Heidelberg, second edition, 2006.

[2] M. Ben-Or, D. Kozen, and J. H. Reif. The complexity of elementary algebra and geometry. *J. Comput. Syst. Sci.*, 32(2):251–264, 1986.

[3] C. Cohen. *Formalized algebraic numbers: construction and first-order theory.* PhD thesis, École polytechnique, Nov 2012.

[4] C. Cohen. Formalization of a sign determination algorithm in real algebraic geometry. Preprint on webpage at https://hal.inria.fr/hal-03274013/document, 2021.

[5] C. Cohen and A. Mahboubi. Formal proofs in real algebraic geometry: from ordered fields to quantifier elimination. *Log. Methods Comput. Sci.*, 8(1), 2012.

[6] K. Kosaian, Y. K. Tan, and A. Platzer. A first complete algorithm for real quantifier elimination in isabelle/hol. In B. Pientka and S. Zdancewic, editors, *CPP*, New York, 2023. ACM.

[7] J. Renegar. On the computational complexity and geometry of the first-order theory of the reals, part III: Quantifier elimination. *J. Symb. Comput.*, 13(3):329–352, 1992.

[8] A. Tarski. *A Decision Method for Elementary Algebra and Geometry.* RAND Corporation, Santa Monica, CA, 1951.