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*⟩

lemma *variables-finite*:
  shows *finite* (*variables qs*)
⟨*proof*⟩

lemma *variables-list-prop*:
  shows *v* ∈ *set* (*variables-list qs*) ⟷ (∃ *q* ∈ *set qs*. *v* ∈ *vars q*)
  ⟨*proof*⟩

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*: $n \leq$ *Polynomial.degree* (*eval-mpoly-poly L p*) $\Longrightarrow$ *Polynomial.coeff* (*eval-mpoly-poly L p*) *n* = *eval-mpoly L* (*Polynomial.coeff p n*)
⟨*proof*⟩

lemma *eval-mpoly-poly-coeff2*: $\forall n >$ *Polynomial.degree* (*eval-mpoly-poly L p*). *Polynomial.coeff* (*eval-mpoly-poly L p*) *n* = *0*
⟨*proof*⟩

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* (λx. (*eval-mpoly-poly L x*)) *p-list*

interpretation *eval-mpoly-map-poly-comm-ring-hom*: *map-poly-comm-ring-hom eval-mpoly val*
⟨*proof*⟩

interpretation *eval-mpoly-map-poly-idom-hom*: *map-poly-idom-hom eval-mpoly val*⟨*proof*⟩

interpretation *eval-mpoly-poly-comm-ring-hom*: *comm-ring-hom eval-mpoly-poly val*
⟨*proof*⟩

interpretation *eval-mpoly-poly-map-poly-idom-hom*: *map-poly-idom-hom eval-mpoly-poly val*⟨*proof*⟩

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*⟩

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*
⟨*proof*⟩

lemma *one-less-degree-is-prefix*:
assumes *Polynomial.degree q* > *0*

shows *Sublist.prefix* (*Polynomial.coeffs* (*one-less-degree q*)) (*Polynomial.coeffs q*)
⟨*proof*⟩

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*⟩

lemma *coeff-one-less-degree-var*:
assumes *0 < Polynomial.degree p*
assumes *one-less-degree p ≠ 0*
shows *i ≤ Polynomial.degree* (*one-less-degree p*) ⟹
poly.coeff p i = poly.coeff (*one-less-degree p*) *i*
⟨*proof*⟩

lemma *coeff-one-less-degree*:
assumes *one-less-degree p ≠ 0*
shows *i ≤ Polynomial.degree* (*one-less-degree p*) ⟹
poly.coeff p i = poly.coeff (*one-less-degree p*) *i*
⟨*proof*⟩

lemma *coeff-one-less-degree-subset*:
assumes *one-less-degree q ≠ 0*
shows *set* (*Polynomial.coeffs* (*Multiv-Poly-Props.one-less-degree q*)) ⊆ *set* (*Polynomial.coeffs q*)
⟨*proof*⟩

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*
⟨*proof*⟩

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* = (∑ *i≤Polynomial.degree* (*one-less-degree p*). *Polynomial.coeff* (*one-less-degree p*) *i* ∗ *x* ^ *i*)
+ (*Polynomial.coeff p d*)∗(*x*^*d*)
⟨*proof*⟩

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*)
⟨*proof*⟩

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*⟩

lemma *isovar-sum-to-higher-degree*:
assumes *b* ≥ (*MPoly-Type.degree p x*)
shows (∑ *i* = *0..*(*MPoly-Type.degree p x*). *Polynomial.monom* (*isolate-variable-sparse p x i*) *i*) =
(∑ *i* = *0..b*. *Polynomial.monom* (*isolate-variable-sparse p x i*) *i*)
⟨*proof*⟩

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*⟩

lemma *multivar-as-univar*: *mpoly-to-mpoly-poly-alt x p* = *mpoly-to-mpoly-poly x p*
⟨*proof*⟩

5 Same mpoly eval means same polynomials

lemma *var-in-some-coeff*:
fixes *p*::*real mpoly Polynomial.poly*
fixes *w*::*real mpoly*
assumes *x* ∈ *vars* ((*poly p w*)::*real mpoly*)
shows *x* ∈ *vars w* ∨ (∃ *i*. *x* ∈ *vars* (*poly.coeff p i*))
⟨*proof*⟩

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*⟩

lemma *zero-list-member*:
  shows *m* < *n* ⟹ (*zero-list n*) ! *m* = *0*
⟨*proof*⟩

lemma *eval-mpoly-zero-is-zero*:
  assumes *all-same*: ⋀ *L*. *eval-mpoly L p* = *0*
  shows *p* = *0*
  ⟨*proof*⟩

6 Useful properties for decision proofs

lemma *eval-mpoly-same*:
  assumes *all-same*: (⋀ *L*. *eval-mpoly L p* = *eval-mpoly L q*)
  shows *p* = *q*
⟨*proof*⟩

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*⟩

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*⟩

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*

⟨*proof*⟩

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* (λ*p*. (*Polynomial.poly p x*)) (*map* (λ*q*. *eval-mpoly-poly xs q*) *reindexed-univ-qs*))
⟨*proof*⟩

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*⟩

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

⟨*proof*⟩

lemma *degree-valuation-le*:
shows *Polynomial.degree* (*eval-mpoly-poly val p*) ≤ *Polynomial.degree p*
⟨*proof*⟩

lemma *satisfies-evaluation-nonzero*:
assumes *satisfies-evaluation val p n*
assumes *n* ≠ *0*
shows *eval-mpoly val p* ≠ *0*
⟨*proof*⟩

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*)
⟨*proof*⟩

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*)
⟨*proof*⟩

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*⟩

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*⟩

lemma *eval-mpoly-poly-pseudo-mod*:
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-mod* (*eval-mpoly-poly val p*)
(*eval-mpoly-poly val q*) =

eval-mpoly-poly val (*pseudo-mod p q*)
⟨*proof*⟩

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*)
⟨*proof*⟩

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*. ∃(*val*::*real list*). *mpoly-sign val q* = *x*} =
{*x*. ∃(*val*::*real list*). ((∀ *v* ∈ *vars q*. *length val* > *v*) ∧ *mpoly-sign val q* = *x*)}
⟨*proof*⟩

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*
  ⟨*proof*⟩

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*
  ⟨*proof*⟩

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*⟩

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*⟩

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*
⟨*proof*⟩

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*)
⟨*proof*⟩

lemma *lookup-assump-means-inset*:
assumes *lookup-assump-aux p assumps* = *Some k*
shows (*p*, *k*) ∈ *set assumps*
⟨*proof*⟩

lemma *inset-means-lookup-assump-some*:
assumes (*p*, *k*) ∈ *set assumps*
shows ∃ *j*. *lookup-assump-aux p assumps* = *Some j*
⟨*proof*⟩

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*))
⟨*proof*⟩

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*: ⋀*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*⟩

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* ⟨*proof*⟩

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* ⟨*proof*⟩

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*⟩

lemma *sign-of-first-nonzero-coefficient-nonzer*:
assumes *inset*: *q* ∈ *set qs*
assumes *nonz*: (*eval-mpoly-poly val q*) ≠ *0*
assumes *sat-eval*: ⋀*p n*. (*p*,*n*) ∈ *set* (*map-mpoly-sign-data val* (*all-coeffs qs*)) ⟹ *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*))
⟨*proof*⟩

lemma *sign-of-first-nonzero-coefficient*:
assumes *inset*: *q* ∈ *set qs*
assumes *sat-eval*: ⋀*p n*. (*p*,*n*) ∈ *set* (*map-mpoly-sign-data val* (*all-coeffs qs*)) ⟹ *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*))
⟨*proof*⟩

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* ∈ *qs-signs*) ⟹ (∃ *val*. (*map* (*rat-of-int* ∘ *Sturm-Tarski.sign* ∘ (λ*p*. *eval-mpoly val p*)) *qs* = *sign-val*))
⟨*proof*⟩

lemma *in-list-lemma*:
assumes *n* = *length l*
shows *inlist*: (*v* ∈ *set l* ⟹ (∃ *k*≤*n*−*1*. *v* = *nth l k*))
⟨*proof*⟩

lemma *eval-list-longer-than-degree*:
assumes *gt-than*: ∀ *i* ∈ *vars q*. *length val* > *i*
assumes *length ell* ≥ *length val*
assumes ∀ *i* < *length val*. *ell* ! *i* = *val* ! *i*
shows *eval-mpoly ell q* = *eval-mpoly val q*
⟨*proof*⟩

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*⟩

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*⟩

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*⟩

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*))
⟨*proof*⟩

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*⟩

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*⟩

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*⟩

lemma *lookup-assump-aux-subset-not-none*:
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*
shows ∃ *i2*. *lookup-assump-aux* (*Polynomial.lead-coeff r*) *branch-assms* = *Some i2*
⟨*proof*⟩

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
)
)) ⟨proof⟩

termination
⟨proof⟩

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
)
))
⟨proof⟩

termination
⟨proof⟩

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 $n \neq 0$
assumes *satisfies-evaluation val* (*Polynomial.lead-coeff q*) *m*
assumes $m \neq 0$
shows *mul-pseudo-mod* (*eval-mpoly-poly val p*) (*eval-mpoly-poly val q*) =
eval-mpoly-poly val (*mul-pseudo-mod p q*)
⟨*proof*⟩

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*. $q = 0 \Longrightarrow$ *P p q assumps*
and *rec*: ⋀*p q assumps*.
⟦$q \neq 0$;
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*)⟧ $\Longrightarrow$
P p q assumps
and *lookup0*: ⋀*p q assumps*.
⟦$q \neq 0$;
lookup-assump-aux (*Polynomial.lead-coeff q*) *assumps* = *Some 0*;
P p (*one-less-degree q*) *assumps*⟧ $\Longrightarrow$ *P p q assumps*
and *lookupN0*: ⋀*p q assumps r*.
⟦$q \neq 0$;
lookup-assump-aux (*Polynomial.lead-coeff q*) *assumps* = *Some r*;
$r \neq 0$;
P q (*mul-pseudo-mod p q*) *assumps*⟧ $\Longrightarrow$ *P p q assumps*
shows *P p q assumps*
⟨*proof*⟩

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*. $p = 0 \Longrightarrow$ *P p q assumps*
and *rec*: ⋀*p q assumps*.
⟦$p \neq 0$;
lookup-assump-aux (*Polynomial.lead-coeff p*) *assumps* = *None*;
P (*one-less-degree p*) *q* ((*Polynomial.lead-coeff p*, *0*) # *assumps*)⟧ $\Longrightarrow$
P p q assumps
and *lookup0*: ⋀*p q assumps*.
⟦$p \neq 0$;
lookup-assump-aux (*Polynomial.lead-coeff p*) *assumps* = *Some 0*;
P (*one-less-degree p*) *q assumps*⟧ $\Longrightarrow$ *P p q assumps*
and *lookupN0*: ⋀*p q assumps r*.
⟦$p \neq 0$;
lookup-assump-aux (*Polynomial.lead-coeff p*) *assumps* = *Some r*;
$r \neq 0$⟧ $\Longrightarrow$ *P p q assumps*
shows *P p q assumps*

⟨*proof*⟩

lemma *spmods-multiv-aux-assum-acc*:
assumes (*acc′*,*seq′*) ∈ *set* (*spmods-multiv-aux p q acc*)
shows *set acc* ⊆ *set acc′*
⟨*proof*⟩

lemma *spmods-multiv-assum-acc*:
assumes (*acc′*,*seq′*) ∈ *set* (*spmods-multiv p q acc*)
shows *set acc* ⊆ *set acc′*
⟨*proof*⟩

lemma *lookup-assum-aux-mem*:
assumes *lookup-assump-aux x ls* = *Some i*
shows (*x*,*i*) ∈ *set ls*
⟨*proof*⟩

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*)
⟨*proof*⟩

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*
⟨*proof*⟩

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*
⟨*proof*⟩

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*
⟨*proof*⟩

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*
⟨*proof*⟩

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* ≠ *0* ⟹ ∃ *i*. *lookup-assump-aux* (*Polynomial.lead-coeff q*) *assumps* = *Some i*
⟨*proof*⟩

lemma *spmods-multiv-sound*:
assumes (*assumps*, *sturm-seq*) ∈ *set* (*spmods-multiv 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*)
⟨*proof*⟩

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 (λ*rhs*. (*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 (λ*lhs*. (*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*)]
)
  ))
  ⟨*proof*⟩
termination ⟨*proof*⟩

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 (λq. if q = 0 then 0 else (rat-of-int ∘ Sturm-Tarski.sign) (lookup-assump
(Polynomial.lead-coeff q) assumps)) qs,
     map (λq. 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 (λbranch.
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* $\Rightarrow$ (*real mpoly* $\times$ *rat*) *list* $\Rightarrow$ *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*) $\Rightarrow$ *Some* ($i \wedge j$)
| - $\Rightarrow$ *None*)
| *lookup-sem-M* (*Or l r*) *ls* = (*case* (*lookup-sem-M l ls*, *lookup-sem-M r ls*)
of (*Some i*, *Some j*) $\Rightarrow$ *Some* ($i \vee j$)
| - $\Rightarrow$ *None*)
| *lookup-sem-M* (*Neg l*) *ls* = (*case* (*lookup-sem-M l ls*)
of Some i $\Rightarrow$ *Some* (($\neg i$))
| - $\Rightarrow$ *None*)
| *lookup-sem-M* (*Atom* (*Less p*)) *ls* =
(*case* (*lookup-assump-aux p ls*) *of*
Some i $\Rightarrow$ *Some* ($i < 0$)
| - $\Rightarrow$ *None*
)
| *lookup-sem-M* (*Atom* (*Leq p*)) *ls* =
(*case* (*lookup-assump-aux p ls*) *of*
Some i $\Rightarrow$ *Some* ($i \leq 0$)
| - $\Rightarrow$ *None*
)
| *lookup-sem-M* (*Atom* (*Eq p*)) *ls* =
(*case* (*lookup-assump-aux p ls*) *of*
Some i $\Rightarrow$ *Some* ($i = 0$)
| - $\Rightarrow$ *None*
)
| *lookup-sem-M* (*Atom* (*Neq p*)) *ls* =
(*case* (*lookup-assump-aux p ls*) *of*
Some i $\Rightarrow$ *Some* ($i \neq 0$)
| - $\Rightarrow$ *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* $\times$ *rat*) $\Rightarrow$ *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* $\Rightarrow$ *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* (λ*csa*. *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*⟩

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*
  ⟨*proof*⟩

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 (λ*q*. *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*
⟨*proof*⟩

lemma *changes-cast-sgn-same-map*:
shows *changes* ((*map of-rat ell*)::*real list*) = *changes* (*ell*::*rat list*)
⟨*proof*⟩

lemma *spmods-multiv-lc-auxNone*:
assumes *inset*: (*assumps*, *sturm-seq*) ∈ *set* (*spmods-multiv p q acc*)
assumes *pnonz*: $p \neq 0$
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*))
∨ (∃ $k \neq 0$. (*assumps*, *sturm-seq*) ∈ *set* (*spmods-multiv-aux p q* ((*Polynomial.lead-coeff p*, *k*) # *acc*)))
⟨*proof*⟩

lemma *spmods-multiv-lc-auxSome1*:
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 0*
shows (((*Polynomial.lead-coeff p*), *0*) ∈ *set acc* ∧ (*assumps*, *sturm-seq*) ∈ *set* (*spmods-multiv* (*one-less-degree p*) *q acc*))
⟨*proof*⟩

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*)))
⟨*proof*⟩

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*⟩

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* ∧ $r \neq 0$
⟨*proof*⟩

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*]
⟨*proof*⟩

lemma *changes-eq*:
shows *changes q* = *changes* (*map real-of-int q*)
⟨*proof*⟩

lemma *eval-mpoly-commutes-helper*:
assumes *val-sat*: ⋀*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*⟩

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* (λ*x*. *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*⟩

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* (λ*x*. *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*⟩

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*)
⟨*proof*⟩

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*⟩

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*⟩

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*))
⟨*proof*⟩

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*)))
⟨*proof*⟩

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*
⟨*proof*⟩

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*⟩

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*⟩

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*
⟨*proof*⟩

lemma *construct-NofI-single-M-univariate-superset*:
assumes *new-p*: *new-p* = *sum-list* (*map* ($\lambda x.$ *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*⟩

lemma *construct-NofI-single-M-univariate*:
assumes *new-p*: *new-p* = *sum-list* (*map* ($\lambda x.$ *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 *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*⟩

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*⟩

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*⟩

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*⟩

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*)
  ⟨*proof*⟩

lemma *nonzcrb*:
  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*⟩

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*: $p \neq 0$
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*⟩

lemma *limit-pos-infinity-helper1*:
assumes *q-in*: *q* ∈ *set qs*
assumes *qnonz*: $q \neq 0$
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*)
⟨*proof*⟩

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*⟩

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*)))
⟨*proof*⟩

lemma *Sturm-Tarski-casting*:
shows ((*Sturm-Tarski.sign x*)) = *rat-of-int* (*Sturm-Tarski.sign x*)
⟨*proof*⟩

lemma *limit-pos-infinity*:
shows *consistent-sign-vec qs* (*crb* (*prod-list-var qs*)) = *sgn-pos-inf-rat-list qs*
⟨*proof*⟩

lemma *nonzcrb-helper-neg*:
assumes *q-in*: *q* ∈ *set qs*
assumes *qnonz*: $q \neq 0$
assumes *lengt*: *length* (*sorted-list-of-set* {(*x*::*real*). (∃ *q* ∈ *set*(*qs*). (*q* ≠ *0* ∧ *poly*

$q\ x = 0))\}$ :: *real list*) > *0*
shows $\neg(\exists\, x \leq (\textit{real-of-int}\ (-\textit{crb}\ (\textit{prod-list-var}\ qs))).\ \textit{poly}\ q\ x = 0)$
⟨*proof*⟩

lemma *nonzcrb-neg*:
assumes *q-in*: $q \in \textit{set}\ qs$
assumes *qnonz*: $q \neq 0$
shows $\neg(\exists\, x \leq (\textit{real-of-int}\ (-\textit{crb}\ (\textit{prod-list-var}\ qs))).\ \textit{poly}\ q\ x = 0)$
⟨*proof*⟩

lemma *root-lb-restate*:
fixes *p*:: *real poly*
assumes *pnonz*: $p \neq 0$
fixes *z*::*real*
assumes *zgt*: $\forall\, x.\ \textit{poly}\ p\ x{=}0 \longrightarrow x > z$
shows $x{\leq}z \Longrightarrow \textit{sgn}\ (\textit{poly}\ p\ x) = \textit{Sturm-Tarski.sign}\ (\textit{sgn-neg-inf}\ p)$
⟨*proof*⟩

lemma *limit-neg-infinity-helper1*:
assumes *q-in*: $q \in \textit{set}\ qs$
assumes *qnonz*: $q \neq 0$
assumes $x = -(\textit{crb}\ (\textit{prod-list-var}\ qs))$
shows $(\textit{if}\ (\textit{poly}\ q\ x > 0)\ \textit{then}\ 1\ \textit{else}\ (\textit{if}\ (\textit{poly}\ q\ x = 0)\ \textit{then}\ 0\ \textit{else}\ -1))$
$= ((\textit{Sturm-Tarski.sign}\ (\textit{sgn-neg-inf}\ q)))$
⟨*proof*⟩

lemma *limit-neg-infinity-helper2*:
assumes *q-in*: $q \in \textit{set}\ qs$
assumes *qnonz*: $q = 0$
assumes $x = (-\textit{crb}\ (\textit{prod-list-var}\ qs))$
shows $(\textit{if}\ (\textit{poly}\ q\ x > 0)\ \textit{then}\ 1\ \textit{else}\ (\textit{if}\ (\textit{poly}\ q\ x = 0)\ \textit{then}\ 0\ \textit{else}\ -1))$
$= \textit{Sturm-Tarski.sign}\ (\textit{sgn-neg-inf}\ q)$
⟨*proof*⟩

lemma *limit-neg-infinity-helper-var*:
assumes *q-in*: $q \in \textit{set}\ qs$
assumes $x = (-\textit{crb}\ (\textit{prod-list-var}\ qs))$
shows $(\textit{if}\ (\textit{poly}\ q\ x > 0)\ \textit{then}\ 1\ \textit{else}\ (\textit{if}\ (\textit{poly}\ q\ x = 0)\ \textit{then}\ 0\ \textit{else}\ -1))$
$= \textit{Sturm-Tarski.sign}\ (\textit{sgn-neg-inf}\ q)$
⟨*proof*⟩

lemma *limit-neg-infinity-helper*:
assumes *q-in*: $q \in \textit{set}\ qs$
assumes $x = (-\textit{crb}\ (\textit{prod-list-var}\ qs))$
shows $(\textit{if}\ (\textit{poly}\ q\ x > 0)\ \textit{then}\ 1\ \textit{else}\ (\textit{if}\ (\textit{poly}\ q\ x = 0)\ \textit{then}\ 0\ \textit{else}\ -1))$
$= (\textit{Sturm-Tarski.sign}\ (\textit{sgn-neg-inf}\ q))$
⟨*proof*⟩

lemma *limit-neg-infinity*:

shows *consistent-sign-vec qs* (−(*crb* (*prod-list-var qs*))) = *sgn-neg-inf-rat-list qs*
⟨*proof*⟩

lemma *csv-signs-at-same*:
shows *consistent-sign-vec qs x* = *signs-at qs x*
⟨*proof*⟩

lemma *complex-real-int-casting*:
fixes *z*:: *int*
shows (*complex-of-real* ∘ *real-of-int*) *z* = *complex-of-int z*
⟨*proof*⟩

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*⟩

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*⟩

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*}
⟨*proof*⟩

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*
⟨*proof*⟩

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*⟩

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*⟩

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*
⟨*proof*⟩

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*
⟨*proof*⟩

lemma *retrieve-polys-prop*:
assumes ⋀*x*. *x* ∈ *set ns* ⟹ *x* < *length qs*
shows (*eval-mpoly-poly-list val* (*retrieve-polys qs ns*)) = (*retrieve-polys* (*map* (*eval-mpoly-poly val*) *qs*) *ns*)
⟨*proof*⟩

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

assumes *rhs-vec-is*: (*assumps*, *rhs-vec*) ∈ *set*(*construct-rhs-vector-M p init-assumps qs Is*)
assumes ⋀*p n*. (*p*,*n*) ∈ *set assumps* ⟹ *satisfies-evaluation val p n*
assumes *well-def-subsets*: ⋀ *Is1 Is2 n*. (*Is1*, *Is2*) ∈ *set Is* ⟹

$(n \in set\ Is1 \vee n \in set\ Is2) \Longrightarrow n < length\ qs$
shows *rhs-vec = construct-rhs-vector-R (eval-mpoly-poly val p) (map (eval-mpoly-poly val) qs) Is*
⟨*proof*⟩

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* ⟹
$(n \in set\ Is1 \vee n \in set\ Is2) \Longrightarrow n < length\ qs$
shows *lhs-vec = solve-for-lhs-R (eval-mpoly-poly val p) (map (eval-mpoly-poly val) qs) subsets matr*
⟨*proof*⟩

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* ⟹
$(n \in set\ Is1 \vee n \in set\ Is2) \Longrightarrow n < length\ qs$
shows *mat-eq = reduce-system-R (eval-mpoly-poly val p) ((map (eval-mpoly-poly val) qs), init-mat-eq)*
⟨*proof*⟩

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* ⟹ $(n \in set\ Is1 \vee n \in set\ Is2) \Longrightarrow n < length\ qs$
obtains *acc mss* **where**
(*acc*,*mss*) ∈ *set* (*input-list*)
mat-eq = reduce-system-R (eval-mpoly-poly val p) (val-qs,mss)
⟨*proof*⟩

lemma *base-case-info-M-well-def*:
assumes (*init-assumps*, (*m*, (*subs*, *signs*))) ∈ *set base-case-info-M*
assumes (*Is1*, *Is2*) ∈ *set subs*
assumes $n \in set\ Is1 \vee n \in set\ Is2$
shows $n < 1$
⟨*proof*⟩

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*⟩

lemma *concat-map-in-set*:
assumes *x* ∈ *set* (*concat* (*map f ls*))
shows ∃ *i* < *length ls*. *x* ∈ *set* (*f* (*ls* ! *i*))
⟨*proof*⟩

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*⟩

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*⟩

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*⟩

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*⟩

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*
⟨*proof*⟩

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*⟩

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*⟩

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*
⟨*proof*⟩

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*
⟨*proof*⟩

lemma *calculate-data-gives-signs-at-roots*:
assumes (*assumps*, *signs*) ∈ *set* (*calculate-data-to-signs* (*calculate-data-M p qs*))
assumes ⋀*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*)
⟨*proof*⟩

lemma *calculate-data-gives-noncomp-signs-at-roots*:
assumes (*assumps*, *signs*) ∈ *set* (*calculate-data-to-signs* (*calculate-data-M p qs*))
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*))
⟨*proof*⟩

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 ⋀*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*)
⟨*proof*⟩

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*))
⟨*proof*⟩

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*
⟨*proof*⟩

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

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*
⟨*proof*⟩

lemma *prod-list-var-gen-nonzero*:
shows *prod-list-var-gen qs* ≠ *0*
⟨*proof*⟩

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*))
⟨*proof*⟩

lemma *lc-assump-generation-list-inv*:
assumes *val*: ⋀*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*)
shows *q* ∈ *set branch-poly-list* ⟹ *q* = *0* ∨ (∃ *i*. *lookup-assump-aux* (*Polynomial.lead-coeff q*) *branch-assms* = *Some i* ∧ *i* ≠ *0*)
⟨*proof*⟩

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* ⟨*proof*⟩

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*⟩

lemma *calc-data-to-signs-and-extract-signs*:
shows (*calculate-data-to-signs ell*) = *extract-signs ell*
⟨*proof*⟩

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*
⟨*proof*⟩

lemma *eval-prod-list-var-gen-match*:
assumes (*branch-assms*, *branch-poly-list*) ∈ *set* (*lc-assump-generation-list qs init-assumps*)
assumes $\bigwedge p\ n.\ (p,n) \in set\ branch\text{-}assms \Longrightarrow satisfies\text{-}evaluation\ val\ p\ n$
shows *eval-mpoly-poly val* (*prod-list-var-gen branch-poly-list*) =
prod-list-var-gen (*map* (*eval-mpoly-poly val*) *branch-poly-list*)
⟨*proof*⟩

lemma *map-branch-poly-list*:
assumes (*branch-assms*, *branch-poly-list*) ∈ *set* (*lc-assump-generation-list qs init-assumps*)
assumes $\bigwedge p\ n.\ (p,n) \in set\ branch\text{-}assms \Longrightarrow satisfies\text{-}evaluation\ val\ p\ n$
shows (*map* (*eval-mpoly-poly val*) *qs*) = (*map* (*eval-mpoly-poly val*) *branch-poly-list*)
⟨*proof*⟩

lemma *check-constant-degree-match*:
assumes (*a*, *q*) ∈ *set* (*lc-assump-generation init-q init-assumps*)
assumes $\bigwedge p\ n.\ (p,n) \in set\ a \Longrightarrow satisfies\text{-}evaluation\ val\ p\ n$
shows *Polynomial.degree q* = *Polynomial.degree* (*eval-mpoly-poly val init-q*)
⟨*proof*⟩

lemma *check-constant-degree-match-list*:
assumes (*branch-assms*, *branch-poly-list*) ∈ *set* (*lc-assump-generation-list qs init-assumps*)
assumes $\bigwedge p\ n.\ (p,n) \in set\ branch\text{-}assms \Longrightarrow satisfies\text{-}evaluation\ val\ p\ n$
shows (*check-all-const-deg-gen branch-poly-list*) = (*check-all-const-deg-gen* (*map* (*eval-mpoly-poly val*) *qs*))
⟨*proof*⟩

lemma *check-all-const-deg-match*:
shows *check-all-const-deg qs* = *check-all-const-deg-gen qs*
⟨*proof*⟩

lemma *prod-list-var-match*:
shows *prod-list-var-gen qs* = *prod-list-var qs*
⟨*proof*⟩

lemma *sign-lead-coeff-on-branch*:
assumes (*a*, *q*) ∈ *set* (*lc-assump-generation init-q init-assumps*)
assumes $q \neq 0$
assumes $\bigwedge p\ n.\ (p,n) \in set\ a \Longrightarrow satisfies\text{-}evaluation\ val\ p\ n$
shows ((*Sturm-Tarski.sign* (*lookup-assump* (*Polynomial.lead-coeff q*) *a*))) =
Sturm-Tarski.sign (*Polynomial.lead-coeff* (*eval-mpoly-poly val q*))
⟨*proof*⟩

lemma *sign-lead-coeff-on-branch-init*:
assumes (*a*, *q*) ∈ *set* (*lc-assump-generation init-q init-assumps*)
assumes $q \neq 0$
assumes $\bigwedge p\ n.\ (p,n) \in set\ a \Longrightarrow satisfies\text{-}evaluation\ val\ p\ n$

shows *Sturm-Tarski.sign* (*lookup-assump* (*Polynomial.lead-coeff q*) *a*) =
Sturm-Tarski.sign (*Polynomial.lead-coeff* (*eval-mpoly-poly val init-q*))
⟨*proof*⟩

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*))
⟨*proof*⟩

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*))
⟨*proof*⟩

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*
⟨*proof*⟩

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*⟩

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*
⟨*proof*⟩

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*
⟨*proof*⟩

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*⟩

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*⟩

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*⟩

17.10 Completeness

lemma *lc-assump-generation-valuation*:
assumes ⋀*p n*. (*p*,*n*) ∈ *set init-assumps* ⟹ *satisfies-evaluation val p n*
shows ∃ *branch* ∈ *set* (*lc-assump-generation q init-assumps*).
set (*fst branch*) ⊆ *set* (*init-assumps*) ∪ *set*
(*map* (λ*x*. (*x*, *mpoly-sign val x*)) (*Polynomial.coeffs q*))
⟨*proof*⟩

lemma *lc-assump-generation-valuation-satisfies-eval*:
fixes *q*:: *rmpoly*

assumes $(p,n) \in$ *set* (*map* (λx. (x, *mpoly-sign val x*)) *ell*)
shows *satisfies-evaluation val p n*
⟨*proof*⟩

lemma *lc-assump-generation-list-valuation*:
assumes $\bigwedge p\ n.\ (p,n) \in$ *set init-assumps* $\Longrightarrow$ *satisfies-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*))
⟨*proof*⟩

lemma *base-case-info-M-assumps-complete*:
assumes $\bigwedge p\ n.\ (p,n) \in$ *set init-assumps* $\Longrightarrow$ *satisfies-evaluation val p n*
shows $\exists$ (*assumps*, *mat-eq*) $\in$ *set* (*base-case-info-M-assumps init-assumps*).
($\forall$ $(p,n) \in$ *set assumps*. *satisfies-evaluation val p n*)
⟨*proof*⟩

lemma *matches-len-complete-spmods-ex*:
assumes $\bigwedge p'\ n'.\ (p',n') \in$ *set acc* $\Longrightarrow$ *satisfies-evaluation val* p' n'
shows $\exists$(*assumps*, *sturm-seq*) $\in$ *set* (*spmods-multiv p q acc*).
($\forall$ $(p,n) \in$ *set assumps*. *satisfies-evaluation val p n*)
⟨*proof*⟩

lemma *matches-len-complete-spmods*:
assumes $\bigwedge p\ n.\ (p,n) \in$ *set acc* $\Longrightarrow$ *satisfies-evaluation val p n*
obtains *assumps sturm-seq* **where**
(*assumps*, *sturm-seq*) $\in$ *set* (*spmods-multiv p q acc*)
$(f,n) \in$ *set assumps* $\Longrightarrow$ *satisfies-evaluation val f n*
⟨*proof*⟩

lemma *tarski-queries-complete-aux*:
assumes $\bigwedge p\ n.\ (p,n) \in$ *set init-assumps* $\Longrightarrow$ *satisfies-evaluation val p n*
shows $\exists$ (*assumps*, *tq*) $\in$ *set* (*construct-NofI-R-spmods p init-assumps I1 I2*).
($\forall$ $(p,n) \in$ *set assumps*. *satisfies-evaluation val p n*)
⟨*proof*⟩

lemma *tarski-queries-complete*:
assumes $\bigwedge p\ n.\ (p,n) \in$ *set init-assumps* $\Longrightarrow$ *satisfies-evaluation val p n*
shows $\exists$ (*assumps*, *tq*) $\in$ *set* (*construct-NofI-M p init-assumps I1 I2*).
($\forall$ $(p,n) \in$ *set assumps*. *satisfies-evaluation val p n*)
⟨*proof*⟩

lemma *solve-for-rhs-rec-M-complete*:
assumes $\bigwedge p\ n.\ (p,n) \in$ *set init-assumps* $\Longrightarrow$ *satisfies-evaluation val p n*
shows $\exists$ (*assumps*, *rhs-vec*) $\in$ *set* (*construct-rhs-vector-rec-M p init-assumps ell*).
($\forall$ $(p,n) \in$ *set assumps*. *satisfies-evaluation val p n*)
⟨*proof*⟩

lemma *solve-for-rhs-M-complete*:
assumes $\bigwedge p\ n.\ (p,n) \in set\ init\text{-}assumps \Longrightarrow satisfies\text{-}evaluation\ val\ p\ n$
shows $\exists\ (assumps,\ rhs\text{-}vec) \in set\ (construct\text{-}rhs\text{-}vector\text{-}M\ p\ init\text{-}assumps\ qs\ Is).$
$(\forall\ (p,n) \in set\ assumps.\ satisfies\text{-}evaluation\ val\ p\ n)$
⟨*proof*⟩

lemma *solve-for-lhs-M-complete*:
assumes $\bigwedge p\ n.\ (p,n) \in set\ init\text{-}assumps \Longrightarrow satisfies\text{-}evaluation\ val\ p\ n$
shows $\exists\ (assumps,\ lhs\text{-}vec) \in set\ (solve\text{-}for\text{-}lhs\text{-}M\ p\ init\text{-}assumps\ qs\ subsets\ matr).$
$(\forall\ (p,n) \in set\ assumps.\ satisfies\text{-}evaluation\ val\ p\ n)$
⟨*proof*⟩

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*⟩

lemma *reduce-system-M-concat-map-helper*:
fixes a:: *'a list*
fixes b:: *'a list list*
assumes $a \in set\ b$
shows $set\ a \subseteq\ set\ (concat\ b)$
⟨*proof*⟩

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*⟩

lemma *combine-systems-M-complete*:
assumes $(assumps1,\ mat\text{-}eq1) \in set\ list1$
assumes $(\forall\ (p,n) \in set\ assumps1.\ satisfies\text{-}evaluation\ val\ p\ n)$
assumes $(assumps2,\ mat\text{-}eq2) \in set\ list2$
assumes $(\forall\ (p,n) \in set\ assumps2.\ satisfies\text{-}evaluation\ val\ p\ n)$
shows $\exists\ (assumps,\ mat\text{-}eq) \in set\ (snd\ (combine\text{-}systems\text{-}M\ p\ q1\ list1\ q2\ list2)).$
$(\forall\ (p,n) \in set\ assumps.\ satisfies\text{-}evaluation\ val\ p\ n)$
⟨*proof*⟩

lemma *get-all-valuations-calculate-data-M*:
assumes $\bigwedge p\ n.\ (p,n) \in set\ init\text{-}assumps \Longrightarrow satisfies\text{-}evaluation\ val\ p\ n$
shows $\exists\ (assumps,\ mat\text{-}eq) \in set\ (calculate\text{-}data\text{-}assumps\text{-}M\ p\ qs\ init\text{-}assumps).$

(∀ (*p*,*n*) ∈ *set assumps*. *satisfies-evaluation val p n*)
⟨*proof*⟩

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*⟩

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*⟩

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*⟩

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*
⟨*proof*⟩

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*⟩

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*)
⟨*proof*⟩

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*)
⟨*proof*⟩

lemma *create-disjunction-eval*:
assumes *eval* (*create-disjunction data*) *xs*
shows ∃ *a* ∈ *set data*. (*eval* (*assump-to-atom-fm* (*fst a*)) *xs*)
⟨*proof*⟩

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

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*⟩

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*)
⟨*proof*⟩

lemma *create-disjunction-true*:
assumes (*assumps*, *signs*) ∈ *set data*
assumes (*eval* (*assump-to-atom-fm assumps*) *xs*)
shows *eval* (*create-disjunction data*) *xs*
⟨*proof*⟩

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*⟩

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

theorem *elim-forall-correct*:
fixes *F*:: *atom fm*
assumes *countQuantifiers F* = *0*

shows *eval* (*AllQ F*) *xs* = *eval* (*elim-forall F*) *xs*
⟨*proof*⟩

lemma *elim-exists-correct*:
fixes *F*:: *atom fm*
assumes *countQuantifiers F* = *0*
shows *eval* (*ExQ F*) *xs* = *eval* (*elim-exist F*) *xs*
⟨*proof*⟩

17.12 Correctness of QE

lemma *assump-to-atom-no-quantifiers*:
shows *countQuantifiers* (*assump-to-atom-fm a*) = *0*
⟨*proof*⟩

lemma *create-disjunction-no-quantifiers*:
shows *countQuantifiers* (*create-disjunction ell*) = *0*
⟨*proof*⟩

lemma *elim-forall-no-quantifiers*:
fixes *F*:: *atom fm*
shows *countQuantifiers* (*elim-forall F*) = *0*
⟨*proof*⟩

lemma *elim-exists-no-quantifiers*:
fixes *F*:: *atom fm*
shows *countQuantifiers* (*elim-exist F*) = *0*
⟨*proof*⟩

lemma *qe-removes-quantifiers*:
shows *countQuantifiers* (*qe F*) = *0*
⟨*proof*⟩

lemma *elim-exist-N-correct*:
assumes *countQuantifiers F* = *0*
shows *eval* (*ExN n F*) *xs* = *eval* ((*elim-exist* ^^ *n*) *F*) *xs*
⟨*proof*⟩

lemma *elim-all-N-correct*:
assumes *countQuantifiers F* = *0*
shows *eval* (*AllN n F*) *xs* = *eval* ((*elim-forall* ^^ *n*) *F*) *xs*
⟨*proof*⟩

theorem *qe-correct*:
fixes *F*:: *atom fm*
shows *eval F xs* = *eval* (*qe F*) *xs*
⟨*proof*⟩

theorem *qe-extended-correct*:

fixes *F*:: *atom fm*
shows *eval F xs* = *eval* (*qe-with-VS F*) *xs*
⟨*proof*⟩

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.