

The BKR Decision Procedure for Univariate Real Arithmetic

Katherine Cordwell, Yong Kiam Tan, and André Platzer

September 13, 2023

Abstract

We formalize the univariate case of Ben-Or, Kozen, and Reif's decision procedure for first-order real arithmetic [1] (the BKR algorithm). We also formalize the univariate case of Renegar's variation [2] of the BKR algorithm. The two formalizations differ mathematically in minor ways (that have significant impact on the multivariate case), but are quite similar in proof structure. Both rely on sign-determination (finding the set of consistent sign assignments for a set of polynomials). The method used for sign-determination is similar to Tarski's original quantifier elimination algorithm (it stores key information in a matrix equation), but with a reduction step to keep complexity low.

Remark

Note that theories `BKR_Decision` and `Renegar_Decision` inherit oracles `holds_by_evaluation` and `cancel_type_definition` from `Berlekamp_Zassenhaus`.

Contents

1	Kronecker Product	4
2	More DL Rank	5
3	Results on Invertibility	13
4	Setup	18
5	Base Case	18
6	Smashing	19
7	Reduction	20

8 Overall algorithm	21
9 Results with Sturm's Theorem	21
10 Setting up the construction: Definitions	22
11 Setting up the construction: Proofs	24
12 Base Case	30
13 Inductive Step	31
13.1 Lemmas on smashing subsets and signs	31
13.2 Well-defined subsets preserved when smashing	32
13.3 Well def signs preserved when smashing	32
13.4 Distinct signs preserved when smashing	32
13.5 Consistent sign assignments preserved when smashing	33
13.6 Main Results	33
14 Reduction Step Proofs	35
14.1 Showing sign conditions preserved when reducing	36
14.2 Showing matrix equation preserved when reducing	39
14.3 Showing matrix preserved	40
14.4 Showing invertibility preserved when reducing	41
14.5 Well def signs preserved when reducing	43
14.6 Distinct signs preserved when reducing	43
14.7 Well def subsets preserved when reducing	43
15 Overall Lemmas	44
15.1 Key properties preserved	44
15.1.1 Properties preserved when combining and reducing systems	44
15.1.2 For length 1 qs	45
15.1.3 For arbitrary qs	45
15.2 Some key results on consistent sign assignments	46
16 Algorithm	47
16.1 Parsing	47
16.2 Factoring	49
16.3 Auxiliary Polynomial	49
16.4 Setting Up the Procedure	50
16.5 Deciding Univariate Problems	50

17 Proofs	51
17.1 Parsing and Semantics	51
17.2 Factoring Lemmas	53
17.3 Auxiliary Polynomial Lemmas	56
17.4 Setting Up the Procedure: Lemmas	58
17.5 Decision Procedure: Main Lemmas	62
18 Base Case	63
19 Smashing	63
20 Reduction	64
21 Overall algorithm	64
22 Tarski Queries Changed	65
23 Matrix Equation	66
24 Base Case	72
25 Inductive Step	74
25.1 Lemmas on smashing subsets	74
25.2 Well-defined subsets preserved when smashing	75
25.3 Consistent Sign Assignments Preserved When Smashing	75
25.4 Main Results	75
26 Reduction Step Proofs	77
26.1 Showing sign conditions preserved when reducing	77
26.2 Showing matrix equation preserved when reducing	79
26.3 Showing matrix preserved	80
26.4 Showing invertibility preserved when reducing	81
26.5 Well def signs preserved when reducing	83
26.6 Distinct signs preserved when reducing	83
26.7 Well def subsets preserved when reducing	83
27 Overall Lemmas	84
27.1 Key properties preserved	84
27.1.1 Properties preserved when combining and reducing systems	84
27.1.2 For length 1 qs	85
27.1.3 For arbitrary qs	85
27.2 Some key results on consistent sign assignments	86

28 Algorithm	87
28.1 Proofs	88

```

theory More-Matrix
  imports Jordan-Normal-Form.Matrix
           Jordan-Normal-Form.DL-Rank
           Jordan-Normal-Form.VS-Connect
           Jordan-Normal-Form.Gauss-Jordan-Elimination
begin

```

1 Kronecker Product

definition *kronecker-product* :: 'a :: ring mat \Rightarrow 'a mat \Rightarrow 'a mat **where**

```

  kronecker-product A B =
    (let ra = dim-row A; ca = dim-col A;
      rb = dim-row B; cb = dim-col B
    in
      mat (ra*rb) (ca*cb)
        ( $\lambda(i,j).$ 
          A $$ (i div rb, j div cb) *
          B $$ (i mod rb, j mod cb)
        ))

```

lemma *arith*:
assumes $d < a$
assumes $c < b$
shows $b*d+c < a*(b::nat)$
<proof>

lemma *dim-kronecker[simp]*:
 $dim\text{-row } (kronecker\text{-product } A B) = dim\text{-row } A * dim\text{-row } B$
 $dim\text{-col } (kronecker\text{-product } A B) = dim\text{-col } A * dim\text{-col } B$
<proof>

lemma *kronecker-inverse-index*:
assumes $r < dim\text{-row } A$ $s < dim\text{-col } A$
assumes $v < dim\text{-row } B$ $w < dim\text{-col } B$
shows $kronecker\text{-product } A B$ \$\$ $(dim\text{-row } B*r+v, dim\text{-col } B*s+w) = A$ \$\$ (r,s)
 $* B$ \$\$ (v,w)
<proof>

lemma *kronecker-distr-left*:
assumes $dim\text{-row } B = dim\text{-row } C$ $dim\text{-col } B = dim\text{-col } C$
shows $kronecker\text{-product } A (B+C) = kronecker\text{-product } A B + kronecker\text{-product } A C$
<proof>

lemma *kronecker-distr-right*:
assumes $dim\text{-row } B = dim\text{-row } C$ $dim\text{-col } B = dim\text{-col } C$
shows $kronecker\text{-product } (B+C) A = kronecker\text{-product } B A + kronecker\text{-product } C A$

$C A$
 $\langle proof \rangle$

lemma *index-mat-mod[simp]*: $nr > 0 \ \& \ nc > 0 \implies mat \ nr \ nc \ f \ \S\S \ (i \ mod \ nr, j \ mod \ nc) = f \ (i \ mod \ nr, j \ mod \ nc)$
 $\langle proof \rangle$

lemma *kronecker-assoc*:
shows $kronecker-product \ A \ (kronecker-product \ B \ C) = kronecker-product \ (kronecker-product \ A \ B) \ C$
 $\langle proof \rangle$

lemma *sum-sum-mod-div*:
 $(\sum ia = 0::nat..<x. \sum ja = 0..<y. f \ ia \ ja) =$
 $(\sum ia = 0..<x*y. f \ (ia \ div \ y) \ (ia \ mod \ y))$
 $\langle proof \rangle$

lemma *kronecker-of-mult*:
assumes $dim-col \ (A :: 'a :: comm-ring \ mat) = dim-row \ C$
assumes $dim-col \ B = dim-row \ D$
shows $kronecker-product \ A \ B * kronecker-product \ C \ D = kronecker-product \ (A * C) \ (B * D)$
 $\langle proof \rangle$

lemma *inverts-mat-length*:
assumes $square-mat \ A \ invert-mat \ A \ B \ invert-mat \ B \ A$
shows $dim-row \ B = dim-row \ A \ dim-col \ B = dim-col \ A$
 $\langle proof \rangle$

lemma *less-mult-imp-mod-less*:
 $m \ mod \ i < i \ \mathbf{if} \ m < n * i \ \mathbf{for} \ m \ n \ i :: nat$
 $\langle proof \rangle$

lemma *kronecker-one*:
shows $kronecker-product \ ((1_m \ x)::'a :: ring-1 \ mat) \ (1_m \ y) = 1_m \ (x*y)$
 $\langle proof \rangle$

lemma *kronecker-invertible*:
assumes $invertible-mat \ (A :: 'a :: comm-ring-1 \ mat) \ invertible-mat \ B$
shows $invertible-mat \ (kronecker-product \ A \ B)$
 $\langle proof \rangle$

2 More DL Rank

instantiation $mat :: (conjugate) \ conjugate$
begin

definition $conjugate-mat :: 'a :: conjugate \ mat \Rightarrow 'a \ mat$

where $\text{conjugate } m = \text{mat } (\text{dim-row } m) (\text{dim-col } m) (\lambda(i,j). \text{conjugate } (m \text{ \$\$ } (i,j)))$

lemma $\text{dim-row-conjugate[simp]}$: $\text{dim-row } (\text{conjugate } m) = \text{dim-row } m$
 $\langle \text{proof} \rangle$

lemma $\text{dim-col-conjugate[simp]}$: $\text{dim-col } (\text{conjugate } m) = \text{dim-col } m$
 $\langle \text{proof} \rangle$

lemma $\text{carrier-vec-conjugate[simp]}$: $m \in \text{carrier-mat } nr \ nc \implies \text{conjugate } m \in \text{carrier-mat } nr \ nc$
 $\langle \text{proof} \rangle$

lemma $\text{mat-index-conjugate[simp]}$:
shows $i < \text{dim-row } m \implies j < \text{dim-col } m \implies \text{conjugate } m \text{ \$\$ } (i,j) = \text{conjugate } (m \text{ \$\$ } (i,j))$
 $\langle \text{proof} \rangle$

lemma $\text{row-conjugate[simp]}$: $i < \text{dim-row } m \implies \text{row } (\text{conjugate } m) \ i = \text{conjugate } (\text{row } m \ i)$
 $\langle \text{proof} \rangle$

lemma $\text{col-conjugate[simp]}$: $i < \text{dim-col } m \implies \text{col } (\text{conjugate } m) \ i = \text{conjugate } (\text{col } m \ i)$
 $\langle \text{proof} \rangle$

lemma rows-conjugate : $\text{rows } (\text{conjugate } m) = \text{map } \text{conjugate } (\text{rows } m)$
 $\langle \text{proof} \rangle$

lemma cols-conjugate : $\text{cols } (\text{conjugate } m) = \text{map } \text{conjugate } (\text{cols } m)$
 $\langle \text{proof} \rangle$

instance
 $\langle \text{proof} \rangle$

end

abbreviation $\text{conjugate-transpose} :: 'a::\text{conjugate mat} \Rightarrow 'a \text{ mat}$
where $\text{conjugate-transpose } A \equiv \text{conjugate } (A^T)$

notation $\text{conjugate-transpose } ((-^H) [1000])$

lemma $\text{transpose-conjugate}$:
shows $(\text{conjugate } A)^T = A^H$
 $\langle \text{proof} \rangle$

lemma $\text{vec-module-col-helper}$:
fixes $A :: ('a :: \text{field}) \text{ mat}$
shows $(0_v (\text{dim-row } A) \in \text{LinearCombinations.module.span class-ring } (\downarrow \text{carrier } =$

carrier-vec (*dim-row A*), *mult* = *undefined*, *one* = *undefined*, *zero* = 0_v (*dim-row A*), *add* = (+), *smult* = (\cdot_v) (*set (cols A)*)
 <proof>

lemma *vec-module-col-helper2*:

fixes *A*:: ('*a* :: field) *mat*

shows $\bigwedge a x. x \in \text{LinearCombinations.module.span class-ring}$

($\{ \text{carrier} = \text{carrier-vec } (\text{dim-row } A), \text{mult} = \text{undefined}, \text{one} = \text{undefined},$
 $\text{zero} = 0_v (\text{dim-row } A), \text{add} = (+), \text{smult} = (\cdot_v) \}$)

(*set (cols A)*) \implies

($\bigwedge a b v. (a + b) \cdot_v v = a \cdot_v v + b \cdot_v v \implies$

$a \cdot_v x$

$\in \text{LinearCombinations.module.span class-ring}$

($\{ \text{carrier} = \text{carrier-vec } (\text{dim-row } A), \text{mult} = \text{undefined}, \text{one} = \text{undefined},$
 $\text{zero} = 0_v (\text{dim-row } A), \text{add} = (+), \text{smult} = (\cdot_v) \}$)

(*set (cols A)*)

<proof>

lemma *vec-module-col*: *module* (*class-ring* :: '*a* :: field ring)

(*module-vec TYPE('a)*)

(*dim-row A*)

(*carrier* :=

LinearCombinations.module.span

class-ring (module-vec TYPE('a) (dim-row A) (set (cols A))))

<proof>

lemma *vec-vs-col*: *vectorspace* (*class-ring* :: '*a* :: field ring)

(*module-vec TYPE('a) (dim-row A)*)

(*carrier* :=

LinearCombinations.module.span

class-ring

(*module-vec TYPE('a)*)

(*dim-row A*)

(*set (cols A)*))

<proof>

lemma *cols-mat-mul-map*:

shows *cols* (*A * B*) = *map* ($(*_v)$ *A*) (*cols B*)

<proof>

lemma *cols-mat-mul*:

shows *set (cols (A * B))* = $(*_v)$ *A* ' *set (cols B)*

<proof>

lemma *set-obtain-sublist*:

assumes $S \subseteq \text{set } ls$

obtains *ss* **where** *distinct ss S = set ss*

<proof>

lemma *mul-mat-of-cols*:

assumes $A \in \text{carrier-mat } nr \ n$

assumes $\bigwedge j. j < \text{length } cs \implies cs \ ! \ j \in \text{carrier-vec } n$

shows $A * (\text{mat-of-cols } n \ cs) = \text{mat-of-cols } nr \ (\text{map } ((*_v) \ A) \ cs)$

<proof>

lemma *helper*:

fixes $x \ y \ z :: 'a :: \{\text{conjugatable-ring}, \text{comm-ring}\}$

shows $x * (y * z) = y * x * z$

<proof>

lemma *cscalar-prod-conjugate-transpose*:

fixes $x \ y :: 'a :: \{\text{conjugatable-ring}, \text{comm-ring}\} \ \text{vec}$

assumes $A \in \text{carrier-mat } nr \ nc$

assumes $x \in \text{carrier-vec } nr$

assumes $y \in \text{carrier-vec } nc$

shows $x \cdot c \ (A *_v \ y) = (A^H *_v \ x) \cdot c \ y$

<proof>

lemma *mat-mul-conjugate-transpose-vec-eq-0*:

fixes $v :: 'a :: \{\text{conjugatable-ordered-ring}, \text{semiring-no-zero-divisors}, \text{comm-ring}\}$

vec

assumes $A \in \text{carrier-mat } nr \ nc$

assumes $v \in \text{carrier-vec } nr$

assumes $A *_v \ (A^H *_v \ v) = 0_v \ nr$

shows $A^H *_v \ v = 0_v \ nc$

<proof>

lemma *row-mat-of-cols*:

assumes $i < nr$

shows $\text{row } (\text{mat-of-cols } nr \ ls) \ i = \text{vec } (\text{length } ls) \ (\lambda j. (ls \ ! \ j) \ \$i)$

<proof>

lemma *mat-of-cols-cons-mat-vec*:

fixes $v :: 'a :: \text{comm-ring} \ \text{vec}$

assumes $v \in \text{carrier-vec } (\text{length } ls)$

assumes $\text{dim-vec } a = nr$

shows

$\text{mat-of-cols } nr \ (a \ \# \ ls) *_v \ (v\text{Cons } m \ v) =$

$m \cdot_v \ a + \text{mat-of-cols } nr \ ls *_v \ v$

<proof>

lemma *smult-vec-zero*:

fixes $v :: 'a :: \text{ring} \ \text{vec}$

shows $0 \cdot_v \ v = 0_v \ (\text{dim-vec } v)$

<proof>

lemma *helper2*:

fixes $A :: 'a::comm-ring\ mat$
fixes $v :: 'a\ vec$
assumes $v \in carrier-vec\ (length\ ss)$
assumes $\bigwedge x. x \in set\ ls \implies dim-vec\ x = nr$
shows
 $mat-of-cols\ nr\ ss *_{v}\ v =$
 $mat-of-cols\ nr\ (ls\ @\ ss) *_{v}\ (0_{v}\ (length\ ls)\ @_{v}\ v)$
 $\langle proof \rangle$

lemma *mat-of-cols-mult-mat-vec-permute-list*:
fixes $v :: 'a::comm-ring\ list$
assumes $f\ permutes\ \{..<length\ ss\}$
assumes $length\ ss = length\ v$
shows
 $mat-of-cols\ nr\ (permute-list\ f\ ss) *_{v}\ vec-of-list\ (permute-list\ f\ v) =$
 $mat-of-cols\ nr\ ss *_{v}\ vec-of-list\ v$
 $\langle proof \rangle$

lemma *subindex-permutation*:
assumes $distinct\ ss\ set\ ss \subseteq \{..<length\ ls\}$
obtains $f\ where\ f\ permutes\ \{..<length\ ls\}$
 $permute-list\ f\ ls = map\ (!)\ ls\ (filter\ (\lambda i. i \notin set\ ss)\ [0..<length\ ls])\ @\ map\ (!)\$
 $ls)\ ss$
 $\langle proof \rangle$

lemma *subindex-permutation2*:
assumes $distinct\ ss\ set\ ss \subseteq \{..<length\ ls\}$
obtains $f\ where\ f\ permutes\ \{..<length\ ls\}$
 $ls = permute-list\ f\ (map\ (!)\ ls\ (filter\ (\lambda i. i \notin set\ ss)\ [0..<length\ ls])\ @\ map\$
 $(!)\ ls)\ ss)$
 $\langle proof \rangle$

lemma *distinct-list-subset-nths*:
assumes $distinct\ ss\ set\ ss \subseteq set\ ls$
obtains $ids\ where\ distinct\ ids\ set\ ids \subseteq \{..<length\ ls\}\ ss = map\ (!)\ ls\ ids$
 $\langle proof \rangle$

lemma *helper3*:
fixes $A :: 'a::comm-ring\ mat$
assumes $A: A \in carrier-mat\ nr\ nc$
assumes $ss: distinct\ ss\ set\ ss \subseteq set\ (cols\ A)$
assumes $v \in carrier-vec\ (length\ ss)$
obtains $c\ where\ mat-of-cols\ nr\ ss *_{v}\ v = A *_{v}\ c\ dim-vec\ c = nc$
 $\langle proof \rangle$

lemma *mat-mul-conjugate-transpose-sub-vec-eq-0*:
fixes $A :: 'a :: \{conjugatable-ordered-ring, semiring-no-zero-divisors, comm-ring\}$
 mat

assumes $A \in \text{carrier-mat } nr \ nc$
assumes $\text{distinct } ss \ \text{set } ss \subseteq \text{set } (\text{cols } (A^H))$
assumes $v \in \text{carrier-vec } (\text{length } ss)$
assumes $A *_{\mathbf{v}} (\text{mat-of-cols } nc \ ss \ *_{\mathbf{v}} \ v) = 0_{\mathbf{v}} \ nr$
shows $(\text{mat-of-cols } nc \ ss \ *_{\mathbf{v}} \ v) = 0_{\mathbf{v}} \ nc$
 <proof>

lemma *Units-invertible*:
fixes $A :: 'a :: \text{semiring-1 } \text{mat}$
assumes $A \in \text{Units } (\text{ring-mat } \text{TYPE}('a) \ n \ b)$
shows $\text{invertible-mat } A$
 <proof>

lemma *invertible-Units*:
fixes $A :: 'a :: \text{semiring-1 } \text{mat}$
assumes $\text{invertible-mat } A$
shows $A \in \text{Units } (\text{ring-mat } \text{TYPE}('a) \ (\text{dim-row } A) \ b)$
 <proof>

lemma *invertible-det*:
fixes $A :: 'a :: \text{field } \text{mat}$
assumes $A \in \text{carrier-mat } n \ n$
shows $\text{invertible-mat } A \longleftrightarrow \text{det } A \neq 0$
 <proof>

context *vec-space* **begin**

lemma *find-indices-distinct*:
assumes $\text{distinct } ss$
assumes $i < \text{length } ss$
shows $\text{find-indices } (ss \ ! \ i) \ ss = [i]$
 <proof>

lemma *lin-indpt-lin-comb-list*:
assumes $\text{distinct } ss$
assumes $\text{lin-indpt } (\text{set } ss)$
assumes $\text{set } ss \subseteq \text{carrier-vec } n$
assumes $\text{lincomb-list } f \ ss = 0_{\mathbf{v}} \ n$
assumes $i < \text{length } ss$
shows $f \ i = 0$
 <proof>

lemma *span-mat-mul-subset*:
assumes $A \in \text{carrier-mat } n \ d$
assumes $B \in \text{carrier-mat } d \ nc$
shows $\text{span } (\text{set } (\text{cols } (A * B))) \subseteq \text{span } (\text{set } (\text{cols } A))$
 <proof>

lemma *rank-mat-mul-right*:

assumes $A \in \text{carrier-mat } n \ d$
assumes $B \in \text{carrier-mat } d \ nc$
shows $\text{rank } (A * B) \leq \text{rank } A$

<proof>

lemma *sumlist-drop*:

assumes $\bigwedge v. v \in \text{set } ls \implies \text{dim-vec } v = n$
shows $\text{sumlist } ls = \text{sumlist } (\text{filter } (\lambda v. v \neq 0_v \ n) \ ls)$

<proof>

lemma *lincomb-list-alt*:

shows $\text{lincomb-list } c \ s =$
 $\text{sumlist } (\text{map2 } (\lambda i \ j. i \cdot_v \ s \ ! \ j) \ (\text{map } (\lambda i. \ c \ i) \ [0..\<\text{length } s]) \ [0..\<\text{length } s])$

<proof>

lemma *lincomb-list-alt2*:

assumes $\bigwedge v. v \in \text{set } s \implies \text{dim-vec } v = n$
assumes $\bigwedge i. i \in \text{set } ls \implies i < \text{length } s$

shows

$\text{sumlist } (\text{map2 } (\lambda i \ j. i \cdot_v \ s \ ! \ j) \ (\text{map } (\lambda i. \ c \ i) \ ls) \ ls) =$
 $\text{sumlist } (\text{map2 } (\lambda i \ j. i \cdot_v \ s \ ! \ j) \ (\text{map } (\lambda i. \ c \ i) \ (\text{filter } (\lambda i. \ c \ i \neq 0) \ ls)) \ (\text{filter } (\lambda i.$
 $c \ i \neq 0) \ ls))$

<proof>

lemma *two-set*:

assumes *distinct* ls
assumes $\text{set } ls = \text{set } [a,b]$
assumes $a \neq b$
shows $ls = [a,b] \vee ls = [b,a]$

<proof>

lemma *filter-disj-inds*:

assumes $i < \text{length } ls \ j < \text{length } ls \ i \neq j$
shows $\text{filter } (\lambda ia. ia \neq j \longrightarrow ia = i) \ [0..\<\text{length } ls] = [i, j] \vee$
 $\text{filter } (\lambda ia. ia \neq j \longrightarrow ia = i) \ [0..\<\text{length } ls] = [j, i]$

<proof>

lemma *lincomb-list-indpt-distinct*:

assumes $\bigwedge v. v \in \text{set } ls \implies \text{dim-vec } v = n$
assumes
 $\bigwedge c. \text{lincomb-list } c \ ls = 0_v \ n \implies (\forall i. i < (\text{length } ls) \longrightarrow c \ i = 0)$

shows *distinct* ls

<proof>

end

locale *conjugatable-vec-space* = *vec-space f-ty n for*

```

    f-ty::'a::conjugatable-ordered-field itself
  and n
begin

lemma transpose-rank-mul-conjugate-transpose:
  fixes A :: 'a mat
  assumes A ∈ carrier-mat n nc
  shows vec-space.rank nc AH ≤ rank (A * AH)
⟨proof⟩

lemma conjugate-transpose-rank-le:
  assumes A ∈ carrier-mat n nc
  shows vec-space.rank nc (AH) ≤ rank A
⟨proof⟩

lemma conjugate-finsum:
  assumes f: f : U → carrier-vec n
  shows conjugate (finsum V f U) = finsum V (conjugate ∘ f) U
⟨proof⟩

lemma rank-conjugate-le:
  assumes A:A ∈ carrier-mat n d
  shows rank (conjugate (A)) ≤ rank A
⟨proof⟩

lemma rank-conjugate:
  assumes A ∈ carrier-mat n d
  shows rank (conjugate A) = rank A
⟨proof⟩

end

lemma conjugate-transpose-rank:
  fixes A::'a::{conjugatable-ordered-field} mat
  shows vec-space.rank (dim-row A) A = vec-space.rank (dim-col A) (AH)
⟨proof⟩

lemma transpose-rank:
  fixes A::'a::{conjugatable-ordered-field} mat
  shows vec-space.rank (dim-row A) A = vec-space.rank (dim-col A) (AT)
⟨proof⟩

lemma rank-mat-mul-left:
  fixes A::'a::{conjugatable-ordered-field} mat
  assumes A ∈ carrier-mat n d
  assumes B ∈ carrier-mat d nc
  shows vec-space.rank n (A * B) ≤ vec-space.rank d B
⟨proof⟩

```

3 Results on Invertibility

definition *take-cols* :: 'a mat \Rightarrow nat list \Rightarrow 'a mat

where *take-cols* A inds = mat-of-cols (dim-row A) (map (!) (cols A)) (filter ((>) (dim-col A)) inds)

definition *take-cols-var* :: 'a mat \Rightarrow nat list \Rightarrow 'a mat

where *take-cols-var* A inds = mat-of-cols (dim-row A) (map (!) (cols A)) (inds)

definition *take-rows* :: 'a mat \Rightarrow nat list \Rightarrow 'a mat

where *take-rows* A inds = mat-of-rows (dim-col A) (map (!) (rows A)) (filter ((>) (dim-row A)) inds)

lemma *cong1*:

$x = y \implies \text{mat-of-cols } n \ x = \text{mat-of-cols } n \ y$
(proof)

lemma *nth-filter*:

assumes $j < \text{length } (\text{filter } P \text{ } l\text{s})$
shows $P \ ((\text{filter } P \text{ } l\text{s}) ! j)$
(proof)

lemma *take-cols-mat-mul*:

assumes $A \in \text{carrier-mat } nr \ n$
assumes $B \in \text{carrier-mat } n \ nc$
shows $A * \text{take-cols } B \text{ } inds = \text{take-cols } (A * B) \text{ } inds$
(proof)

lemma *take-cols-carrier-mat*:

assumes $A \in \text{carrier-mat } nr \ nc$
obtains n **where** $\text{take-cols } A \text{ } inds \in \text{carrier-mat } nr \ n$
(proof)

lemma *take-cols-carrier-mat-strict*:

assumes $A \in \text{carrier-mat } nr \ nc$
assumes $\bigwedge i. i \in \text{set } inds \implies i < nc$
shows $\text{take-cols } A \text{ } inds \in \text{carrier-mat } nr \ (\text{length } inds)$
(proof)

lemma *gauss-jordan-take-cols*:

assumes $\text{gauss-jordan } A \ (\text{take-cols } A \text{ } inds) = (C, D)$
shows $D = \text{take-cols } C \text{ } inds$
(proof)

lemma *dim-col-take-cols*:

assumes $\bigwedge j. j \in \text{set } inds \implies j < \text{dim-col } A$
shows $\text{dim-col } (\text{take-cols } A \text{ } inds) = \text{length } inds$
(proof)

lemma *dim-col-take-rows*[simp]:
shows $\dim\text{-col } (\text{take-rows } A \text{ inds}) = \dim\text{-col } A$
 $\langle \text{proof} \rangle$

lemma *cols-take-cols-subset*:
shows $\text{set } (\text{cols } (\text{take-cols } A \text{ inds})) \subseteq \text{set } (\text{cols } A)$
 $\langle \text{proof} \rangle$

lemma *dim-row-take-cols*[simp]:
shows $\dim\text{-row } (\text{take-cols } A \text{ ls}) = \dim\text{-row } A$
 $\langle \text{proof} \rangle$

lemma *dim-row-append-rows*[simp]:
shows $\dim\text{-row } (A @_r B) = \dim\text{-row } A + \dim\text{-row } B$
 $\langle \text{proof} \rangle$

lemma *rows-inj*:
assumes $\dim\text{-col } A = \dim\text{-col } B$
assumes $\text{rows } A = \text{rows } B$
shows $A = B$
 $\langle \text{proof} \rangle$

lemma *append-rows-index*:
assumes $\dim\text{-col } A = \dim\text{-col } B$
assumes $i < \dim\text{-row } A + \dim\text{-row } B$
assumes $j < \dim\text{-col } A$
shows $(A @_r B) \$\$ (i,j) = (\text{if } i < \dim\text{-row } A \text{ then } A \$\$ (i,j) \text{ else } B \$\$ (i - \dim\text{-row } A, j))$
 $\langle \text{proof} \rangle$

lemma *row-append-rows*:
assumes $\dim\text{-col } A = \dim\text{-col } B$
assumes $i < \dim\text{-row } A + \dim\text{-row } B$
shows $\text{row } (A @_r B) i = (\text{if } i < \dim\text{-row } A \text{ then } \text{row } A i \text{ else } \text{row } B (i - \dim\text{-row } A))$
 $\langle \text{proof} \rangle$

lemma *append-rows-mat-mul*:
assumes $\dim\text{-col } A = \dim\text{-col } B$
shows $(A @_r B) * C = A * C @_r B * C$
 $\langle \text{proof} \rangle$

lemma *cardlt*:
shows $\text{card } \{i. i < (n::\text{nat})\} \leq n$
 $\langle \text{proof} \rangle$

lemma *row-echelon-form-zero-rows*:
assumes *row-ech*: *row-echelon-form* A
assumes *dim-asm*: $\dim\text{-col } A \geq \dim\text{-row } A$

shows $\text{take-rows } A [0..<\text{length } (\text{pivot-positions } A)] @_r \ 0_m (\text{dim-row } A - \text{length } (\text{pivot-positions } A)) (\text{dim-col } A) = A$
 ⟨proof⟩

lemma *length-pivot-positions-dim-row*:
assumes *row-echelon-form* A
shows $\text{length } (\text{pivot-positions } A) \leq \text{dim-row } A$
 ⟨proof⟩

lemma *rref-pivot-positions*:
assumes *row-echelon-form* R
assumes $R: R \in \text{carrier-mat } nr \ nc$
shows $\bigwedge i \ j. (i,j) \in \text{set } (\text{pivot-positions } R) \implies i < nr \wedge j < nc$
 ⟨proof⟩

lemma *pivot-fun-monoton*:
assumes *pf*: *pivot-fun* $A \ f (\text{dim-col } A)$
assumes *dr*: $\text{dim-row } A = nr$
shows $\bigwedge i. i < nr \implies (\bigwedge k. ((k < nr \wedge i < k) \longrightarrow f \ i \leq f \ k))$
 ⟨proof⟩

lemma *pivot-positions-contains*:
assumes *row-ech*: *row-echelon-form* A
assumes *dim-h*: $\text{dim-col } A \geq \text{dim-row } A$
assumes *pivot-fun* $A \ f (\text{dim-col } A)$
shows $\forall i < (\text{length } (\text{pivot-positions } A)). (i, f \ i) \in \text{set } (\text{pivot-positions } A)$
 ⟨proof⟩

lemma *pivot-positions-form-helper-1*:
shows $(a, b) \in \text{set } (\text{pivot-positions-main-gen } z \ A \ nr \ nc \ i \ j) \implies i \leq a$
 ⟨proof⟩

lemma *pivot-positions-form-helper-2*:
shows *sorted-wrt* $(<) (\text{map fst } (\text{pivot-positions-main-gen } z \ A \ nr \ nc \ i \ j))$
 ⟨proof⟩

lemma *sorted-pivot-positions*:
shows *sorted-wrt* $(<) (\text{map fst } (\text{pivot-positions } A))$
 ⟨proof⟩

lemma *pivot-positions-form*:
assumes *row-ech*: *row-echelon-form* A
assumes *dim-h*: $\text{dim-col } A \geq \text{dim-row } A$
shows $\forall i < (\text{length } (\text{pivot-positions } A)). \text{fst } ((\text{pivot-positions } A) ! i) = i$
 ⟨proof⟩

lemma *take-cols-pivot-eq*:
assumes *row-ech*: *row-echelon-form* A
assumes *dim-h*: $\text{dim-col } A \geq \text{dim-row } A$

shows $\text{take-cols } A (\text{map snd } (\text{pivot-positions } A)) =$
 $1_m (\text{length } (\text{pivot-positions } A)) @_r$
 $0_m (\text{dim-row } A - \text{length } (\text{pivot-positions } A)) (\text{length } (\text{pivot-positions } A))$
 $\langle \text{proof} \rangle$

lemma *rref-right-mul*:

assumes *row-echelon-form* A

assumes $\text{dim-col } A \geq \text{dim-row } A$

shows

$\text{take-cols } A (\text{map snd } (\text{pivot-positions } A)) * \text{take-rows } A [0..<\text{length } (\text{pivot-positions } A)] = A$
 $\langle \text{proof} \rangle$

context *conjugatable-vec-space* **begin**

lemma *lin-indpt-id*:

shows $\text{lin-indpt } (\text{set } (\text{cols } (1_m \ n)::'a \ \text{vec list}))$
 $\langle \text{proof} \rangle$

lemma *lin-indpt-take-cols-id*:

shows $\text{lin-indpt } (\text{set } (\text{cols } (\text{take-cols } (1_m \ n) \ \text{inds})))$
 $\langle \text{proof} \rangle$

lemma *cols-id-unit-vecs*:

shows $\text{cols } (1_m \ d) = \text{unit-vecs } d$
 $\langle \text{proof} \rangle$

lemma *distinct-cols-id*:

shows $\text{distinct } (\text{cols } (1_m \ d)::'a \ \text{vec list})$
 $\langle \text{proof} \rangle$

lemma *distinct-map-nth*:

assumes *distinct* ls

assumes *distinct* $inds$

assumes $\bigwedge j. j \in \text{set } inds \implies j < \text{length } ls$

shows $\text{distinct } (\text{map } (!) \ ls) \ inds$

$\langle \text{proof} \rangle$

lemma *distinct-take-cols-id*:

assumes *distinct* $inds$

shows $\text{distinct } (\text{cols } (\text{take-cols } (1_m \ n) \ \text{inds}) :: 'a \ \text{vec list})$

$\langle \text{proof} \rangle$

lemma *rank-take-cols*:

assumes *distinct* $inds$

shows $\text{rank } (\text{take-cols } (1_m \ n) \ \text{inds}) = \text{length } (\text{filter } (>) \ n) \ \text{inds}$

$\langle \text{proof} \rangle$

lemma *rank-mul-left-invertible-mat*:

fixes $A::'a \text{ mat}$
assumes *invertible-mat* A
assumes $A \in \text{carrier-mat } n \ n$
assumes $B \in \text{carrier-mat } n \ nc$
shows $\text{rank } (A * B) = \text{rank } B$
 $\langle \text{proof} \rangle$

lemma *invertible-take-cols-rank*:
fixes $A::'a \text{ mat}$
assumes *invertible-mat* A
assumes $A \in \text{carrier-mat } n \ n$
assumes *distinct inds*
shows $\text{rank } (\text{take-cols } A \ \text{inds}) = \text{length } (\text{filter } ((>) \ n) \ \text{inds})$
 $\langle \text{proof} \rangle$

lemma *rank-take-cols-leq*:
assumes $R:R \in \text{carrier-mat } n \ nc$
shows $\text{rank } (\text{take-cols } R \ \text{ls}) \leq \text{rank } R$
 $\langle \text{proof} \rangle$

lemma *rank-take-cols-geq*:
assumes $R:R \in \text{carrier-mat } n \ nc$
assumes $t:\text{take-cols } R \ \text{ls} \in \text{carrier-mat } n \ r$
assumes $B:B \in \text{carrier-mat } r \ nc$
assumes $R = (\text{take-cols } R \ \text{ls}) * B$
shows $\text{rank } (\text{take-cols } R \ \text{ls}) \geq \text{rank } R$
 $\langle \text{proof} \rangle$

lemma *rref-drop-pivots*:
assumes *row-ech*: *row-echelon-form* R
assumes *dims*: $R \in \text{carrier-mat } n \ nc$
assumes *order*: $nc \geq n$
shows $\text{rank } (\text{take-cols } R \ (\text{map } \text{snd } (\text{pivot-positions } R))) = \text{rank } R$
 $\langle \text{proof} \rangle$

lemma *gjs-and-take-cols-var*:
fixes $A::'a \text{ mat}$
assumes $A:A \in \text{carrier-mat } n \ nc$
assumes *order*: $nc \geq n$
shows $(\text{take-cols } A \ (\text{map } \text{snd } (\text{pivot-positions } (\text{gauss-jordan-single } A)))) =$
 $(\text{take-cols-var } A \ (\text{map } \text{snd } (\text{pivot-positions } (\text{gauss-jordan-single } A))))$
 $\langle \text{proof} \rangle$

lemma *gauss-jordan-single-rank*:
fixes $A::'a \text{ mat}$
assumes $A:A \in \text{carrier-mat } n \ nc$
assumes *order*: $nc \geq n$
shows $\text{rank } (\text{take-cols } A \ (\text{map } \text{snd } (\text{pivot-positions } (\text{gauss-jordan-single } A)))) =$
 $\text{rank } A$

<proof>

lemma *lin-indpt-subset-cols*:
 fixes *A*:: 'a mat
 fixes *B*:: 'a vec set
 assumes *A* ∈ carrier-mat *n n*
 assumes *inv*: invertible-mat *A*
 assumes *B* ⊆ set (cols *A*)
 shows *lin-indpt B*
<proof>

lemma *rank-invertible-subset-cols*:
 fixes *A*:: 'a mat
 fixes *B*:: 'a vec list
 assumes *inv*: invertible-mat *A*
 assumes *A-square*: *A* ∈ carrier-mat *n n*
 assumes *set-sub*: set (*B*) ⊆ set (cols *A*)
 assumes *dist-B*: distinct *B*
 shows rank (mat-of-cols *n B*) = length *B*
<proof>

end

end

theory *BKR-Algorithm*
 imports
 Sturm-Tarski.Sturm-Tarski
 More-Matrix

begin

4 Setup

definition *retrieve-polys*:: 'a list ⇒ nat list ⇒ 'a list
 where *retrieve-polys qss index-list* = (map (nth *qss*) *index-list*)

definition *construct-NofI*:: real poly ⇒ real poly list ⇒ rat
 where *construct-NofI p I* = rat-of-int (changes-R-smods *p* ((pderiv *p*)*(prod-list *I*)))

definition *construct-rhs-vector*:: real poly ⇒ real poly list ⇒ nat list list ⇒ rat vec
 where *construct-rhs-vector p qs Is* = vec-of-list (map (λ *I*.(construct-NofI *p* (retrieve-polys *qs I*))) *Is*)

5 Base Case

definition *base-case-info*:: (rat mat × (nat list list × rat list list))
 where *base-case-info* =

((*mat-of-rows-list* 2 [[1,1], [1,-1]], ([[0],[0]], [[1],[-1]]))

definition *base-case-solve-for-lhs*:: *real poly* \Rightarrow *real poly* \Rightarrow *rat vec*

where *base-case-solve-for-lhs* *p q* = (*mult-mat-vec* (*mat-of-rows-list* 2 [[1/2, 1/2], [1/2, -1/2]]) (*construct-rhs-vector* *p* [*q*] [[], [0]]))

thm *gauss-jordan-compute-inverse*

primrec *matr-option*:: *nat* \Rightarrow 'a::{*one*, *zero*} *mat option* \Rightarrow 'a *mat*

where *matr-option* *dimen* *None* = *1_m* *dimen*
| *matr-option* *dimen* (*Some* *c*) = *c*

definition *mat-equal*:: 'a::*field mat* \Rightarrow 'a :: *field mat* \Rightarrow *bool*

where *mat-equal* *A B* = (*dim-row* *A* = *dim-row* *B* \wedge *dim-col* *A* = *dim-col* *B* \wedge (*mat-to-list* *A*) = (*mat-to-list* *B*))

definition *mat-inverse-var* :: 'a :: *field mat* \Rightarrow 'a *mat option* **where**

mat-inverse-var *A* = (*if* *dim-row* *A* = *dim-col* *A* *then*

let *one* = *1_m* (*dim-row* *A*) *in*

 (*case* *gauss-jordan* *A* *one* *of*

 (*B*, *C*) \Rightarrow *if* (*mat-equal* *B* *one*) *then* *Some* *C* *else* *None*) *else* *None*)

definition *solve-for-lhs*:: *real poly* \Rightarrow *real poly list* \Rightarrow *nat list list* \Rightarrow *rat mat* \Rightarrow *rat vec*

where *solve-for-lhs* *p qs subsets matr* =

mult-mat-vec (*matr-option* (*dim-row* *matr*) (*mat-inverse-var* *matr*)) (*construct-rhs-vector* *p qs subsets*)

6 Smashing

definition *subsets-smash*::*nat* \Rightarrow *nat list list* \Rightarrow *nat list list* \Rightarrow *nat list list*

where *subsets-smash* *n s1 s2* = *concat* (*map* (λ *l1*. *map* (λ *l2*. *l1* @ (*map* ((+) *n* *l2*))) *s2*) *s1*)

definition *signs-smash*::'a *list list* \Rightarrow 'a *list list* \Rightarrow 'a *list list*

where *signs-smash* *s1 s2* = *concat* (*map* (λ *l1*. *map* (λ *l2*. *l1* @ *l2*) *s2*) *s1*)

definition *smash-systems*:: *real poly* \Rightarrow *real poly list* \Rightarrow *real poly list* \Rightarrow *nat list list* \Rightarrow *nat list list* \Rightarrow

rat list list \Rightarrow *rat list list* \Rightarrow *rat mat* \Rightarrow *rat mat* \Rightarrow

real poly list \times (*rat mat* \times (*nat list list* \times *rat list list*))

where *smash-systems* *p qs1 qs2 subsets1 subsets2 signs1 signs2 mat1 mat2* =

 (*qs1*@*qs2*, (*kroncker-product* *mat1* *mat2*, (*subsets-smash* (*length* *qs1*) *subsets1* *subsets2*, *signs-smash* *signs1* *signs2*)))

fun *combine-systems*:: *real poly* \Rightarrow (*real poly list* \times (*rat mat* \times (*nat list list* \times *rat*

$list\ list))) \Rightarrow (real\ poly\ list \times (rat\ mat \times (nat\ list\ list \times rat\ list\ list)))$
 $\Rightarrow (real\ poly\ list \times (rat\ mat \times (nat\ list\ list \times rat\ list\ list)))$
where *combine-systems* $p\ (qs1, m1, sub1, sgn1)\ (qs2, m2, sub2, sgn2) =$
 $(smash-systems\ p\ qs1\ qs2\ sub1\ sub2\ sgn1\ sgn2\ m1\ m2)$

7 Reduction

definition *find-nonzeros-from-input-vec*:: $rat\ vec \Rightarrow nat\ list$

where *find-nonzeros-from-input-vec* $lhs-vec = filter\ (\lambda i. lhs-vec\ \$\ i \neq 0)\ [0..< dim-vec\ lhs-vec]$

definition *take-indices*:: $'a\ list \Rightarrow nat\ list \Rightarrow 'a\ list$

where *take-indices* $subsets\ indices = map\ (!)\ subsets)\ indices$

definition *take-cols-from-matrix*:: $'a\ mat \Rightarrow nat\ list \Rightarrow 'a\ mat$

where *take-cols-from-matrix* $matr\ indices-to-keep =$
 $mat-of-cols\ (dim-row\ matr)\ ((take-indices\ (cols\ matr)\ indices-to-keep):: 'a\ vec\ list)$

definition *take-rows-from-matrix*:: $'a\ mat \Rightarrow nat\ list \Rightarrow 'a\ mat$

where *take-rows-from-matrix* $matr\ indices-to-keep =$
 $mat-of-rows\ (dim-col\ matr)\ ((take-indices\ (rows\ matr)\ indices-to-keep):: 'a\ vec\ list)$

fun *reduce-mat-cols*:: $'a\ mat \Rightarrow rat\ vec \Rightarrow 'a\ mat$

where *reduce-mat-cols* $A\ lhs-vec = take-cols-from-matrix\ A\ (find-nonzeros-from-input-vec\ lhs-vec)$

definition *rows-to-keep*:: $('a::field)\ mat \Rightarrow nat\ list$ **where**

rows-to-keep $A = map\ snd\ (pivot-positions\ (gauss-jordan-single\ (A^T)))$

fun *reduction-step*:: $rat\ mat \Rightarrow rat\ list\ list \Rightarrow nat\ list\ list \Rightarrow rat\ vec \Rightarrow rat\ mat \times (nat\ list\ list \times rat\ list\ list)$

where *reduction-step* $A\ signs\ subsets\ lhs-vec =$
 $(let\ reduce-cols-A = (reduce-mat-cols\ A\ lhs-vec);$
 $rows-keep = rows-to-keep\ reduce-cols-A\ in$
 $(take-rows-from-matrix\ reduce-cols-A\ rows-keep,$
 $(take-indices\ subsets\ rows-keep,$
 $take-indices\ signs\ (find-nonzeros-from-input-vec\ lhs-vec))))$

fun *reduce-system*:: $real\ poly \Rightarrow (real\ poly\ list \times (rat\ mat \times (nat\ list\ list \times rat\ list\ list))) \Rightarrow (rat\ mat \times (nat\ list\ list \times rat\ list\ list))$

where *reduce-system* $p\ (qs,m,subs,signs) =$
 $reduction-step\ m\ signs\ subs\ (solve-for-lhs\ p\ qs\ subs\ m)$

8 Overall algorithm

```

fun calculate-data:: real poly  $\Rightarrow$  real poly list  $\Rightarrow$  (rat mat  $\times$  (nat list list  $\times$  rat list list))
  where
    calculate-data p qs =
    ( let len = length qs in
      if len = 0 then
        ( $\lambda(a,b,c).(a,b,\text{map } (\text{drop } 1) c)$ ) (reduce-system p ([1],base-case-info))
      else if len  $\leq$  1 then reduce-system p (qs,base-case-info)
      else
        (let q1 = take (len div 2) qs; left = calculate-data p q1;
          q2 = drop (len div 2) qs; right = calculate-data p q2;
          comb = combine-systems p (q1,left) (q2,right) in
          reduce-system p comb
        )
    )

```

```

definition find-consistent-signs-at-roots:: real poly  $\Rightarrow$  real poly list  $\Rightarrow$  rat list list
  where [code]:
    find-consistent-signs-at-roots p qs =
    ( let (M,S, $\Sigma$ ) = calculate-data p qs in  $\Sigma$  )

```

```

lemma find-consistent-signs-at-roots-thm:
  shows find-consistent-signs-at-roots p qs = snd (snd (calculate-data p qs))
  <proof>

```

```

end
theory Matrix-Equation-Construction

```

```

imports BKR-Algorithm
begin

```

9 Results with Sturm's Theorem

```

lemma relprime:
  fixes q::real poly
  assumes coprime p q
  assumes p  $\neq$  0
  assumes q  $\neq$  0
  shows changes-R-smods p (pderiv p) = card {x. poly p x = 0  $\wedge$  poly q x > 0} +
  card {x. poly p x = 0  $\wedge$  poly q x < 0}
  <proof>

```

```

lemma card-eq-const-sum:
  fixes k:: real
  assumes finite A

```

shows $k * \text{card } A = \text{sum } (\lambda x. k) A$
 ⟨proof⟩

lemma *restate-tarski*:

fixes $q::\text{real poly}$
assumes *coprime* $p q$
assumes $p \neq 0$
assumes $q \neq 0$
shows *changes-R-smods* $p ((pderiv p) * q) = \text{card } \{x. \text{poly } p x = 0 \wedge \text{poly } q x > 0\} - \text{int}(\text{card } \{x. \text{poly } p x = 0 \wedge \text{poly } q x < 0\})$
 ⟨proof⟩

lemma *restate-tarski2*:

fixes $q::\text{real poly}$
assumes $p \neq 0$
shows *changes-R-smods* $p ((pderiv p) * q) =$
 $\text{int}(\text{card } \{x. \text{poly } p x = 0 \wedge \text{poly } q x > 0\}) -$
 $\text{int}(\text{card } \{x. \text{poly } p x = 0 \wedge \text{poly } q x < 0\})$
 ⟨proof⟩

lemma *coprime-set-prod*:

fixes $I::\text{real poly set}$
shows *finite* $I \implies ((\forall q \in I. (\text{coprime } p q)) \longrightarrow (\text{coprime } p (\prod I)))$
 ⟨proof⟩

lemma *finite-nonzero-set-prod*:

fixes $I::\text{real poly set}$
shows *nonzero-hyp*: *finite* $I \implies ((\forall q \in I. q \neq 0) \longrightarrow \prod I \neq 0)$
 ⟨proof⟩

10 Setting up the construction: Definitions

definition *characterize-root-list-p*:: *real poly* \Rightarrow *real list*

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

lemma *construct-NoFI-prop*:

fixes $p::\text{real poly}$
fixes $I::\text{real poly list}$
assumes *nonzero*: $p \neq 0$
shows *construct-NoFI* $p I =$
 $\text{rat-of-int } (\text{int } (\text{card } \{x. \text{poly } p x = 0 \wedge \text{poly } (\text{prod-list } I) x > 0\}) -$
 $\text{int } (\text{card } \{x. \text{poly } p x = 0 \wedge \text{poly } (\text{prod-list } I) x < 0\}))$
 ⟨proof⟩

definition *construct-s-vector*:: *real poly* \Rightarrow *real poly list list* \Rightarrow *rat vec*

where *construct-s-vector* $p Is = \text{vec-of-list } (\text{map } (\lambda I. (\text{construct-NoFI } p I)) Is)$

definition *squash*::'a::linordered-field \Rightarrow rat

where *squash* $x =$ (if $x > 0$ then 1
else if $x < 0$ then -1
else 0)

definition *signs-at*::real poly list \Rightarrow real \Rightarrow rat list

where *signs-at* $qs\ x \equiv$
map (*squash* \circ ($\lambda q.$ poly $q\ x$)) qs

definition *characterize-consistent-signs-at-roots*:: real poly \Rightarrow real poly list \Rightarrow rat list list

where *characterize-consistent-signs-at-roots* $p\ qs =$
(remdups (map (*signs-at* qs) (*characterize-root-list-p* p))))

definition *consistent-sign-vec-copr*::real poly list \Rightarrow real \Rightarrow rat list

where *consistent-sign-vec-copr* $qs\ x \equiv$
map ($\lambda q.$ if (poly $q\ x > 0$) then (1::rat) else (-1::rat)) qs

definition *characterize-consistent-signs-at-roots-copr*:: real poly \Rightarrow real poly list \Rightarrow rat list list

where *characterize-consistent-signs-at-roots-copr* $p\ qss =$
(remdups (map (*consistent-sign-vec-copr* qss) (*characterize-root-list-p* p))))

lemma *csa-list-copr-rel*:

fixes p :: real poly

fixes qs :: real poly list

assumes nonzero: $p \neq 0$

assumes pairwise-rel-prime: $\forall q. ((List.member\ qs\ q) \longrightarrow (coprime\ p\ q))$

shows *characterize-consistent-signs-at-roots* $p\ qs =$ *characterize-consistent-signs-at-roots-copr* $p\ qs$
{proof}

definition *list-constr*:: nat list \Rightarrow nat \Rightarrow bool

where *list-constr* $L\ n \equiv list-all\ (\lambda x. x < n)\ L$

definition *all-list-constr*:: nat list list \Rightarrow nat \Rightarrow bool

where *all-list-constr* $L\ n \equiv (\forall x. List.member\ L\ x \longrightarrow list-constr\ x\ n)$

definition *z*:: nat list \Rightarrow rat list \Rightarrow rat

where z *index-list* *sign-asg* $\equiv (prod-list\ (map\ (nth\ sign-asg)\ index-list))$

definition *mtx-row*:: rat list list \Rightarrow nat list \Rightarrow rat list

where *mtx-row* *sign-list* *index-list* $\equiv (map\ (\lambda z. z\ index-list)\ sign-list)$

definition *matrix-A*:: $\text{rat list list} \Rightarrow \text{nat list list} \Rightarrow \text{rat mat}$

where *matrix-A sign-list subset-list* =
 $(\text{mat-of-rows-list } (\text{length sign-list}) (\text{map } (\lambda i . (\text{mtx-row sign-list } i)) \text{ subset-list}))$

definition *alt-matrix-A*:: $\text{rat list list} \Rightarrow \text{nat list list} \Rightarrow \text{rat mat}$

where *alt-matrix-A signs subsets* = $(\text{mat } (\text{length subsets}) (\text{length signs})$
 $(\lambda(i, j). z (\text{subsets } ! i) (\text{signs } ! j)))$

lemma *alt-matrix-char*: $\text{alt-matrix-A signs subsets} = \text{matrix-A signs subsets}$

<proof>

lemma *subsets-are-rows*: $\forall i < (\text{length subsets}). \text{row } (\text{alt-matrix-A signs subsets}) i$
 $= \text{vec } (\text{length signs}) (\lambda j. z (\text{subsets } ! i) (\text{signs } ! j))$

<proof>

lemma *signs-are-cols*: $\forall i < (\text{length signs}). \text{col } (\text{alt-matrix-A signs subsets}) i$
 $= \text{vec } (\text{length subsets}) (\lambda j. z (\text{subsets } ! j) (\text{signs } ! i))$

<proof>

definition *construct-lhs-vector*:: $\text{real poly} \Rightarrow \text{real poly list} \Rightarrow \text{rat list list} \Rightarrow \text{rat vec}$

where *construct-lhs-vector p qs signs* \equiv
 $\text{vec-of-list } (\text{map } (\lambda w. \text{rat-of-int } (\text{int } (\text{length } (\text{filter } (\lambda v. v = w) (\text{map } (\text{consistent-sign-vec-copr } qs) (\text{characterize-root-list-p } p)))))) \text{ signs})$

definition *satisfy-equation*:: $\text{real poly} \Rightarrow \text{real poly list} \Rightarrow \text{nat list list} \Rightarrow \text{rat list list}$
 $\Rightarrow \text{bool}$

where *satisfy-equation p qs subset-list sign-list* =
 $(\text{mult-mat-vec } (\text{matrix-A sign-list subset-list}) (\text{construct-lhs-vector } p \text{ qs sign-list}) = (\text{construct-rhs-vector } p \text{ qs subset-list}))$

11 Setting up the construction: Proofs

lemma *row-mat-of-rows-list*:

assumes *list-all* $(\lambda r. \text{length } r = nc)$ *rs*

assumes $i < \text{length } rs$

shows $\text{row } (\text{mat-of-rows-list } nc \text{ rs}) i = \text{vec-of-list } (nth \text{ rs } i)$

<proof>

lemma *mult-mat-vec-of-list*:

assumes $\text{length } ls = nc$

assumes *list-all* $(\lambda r. \text{length } r = nc)$ *rs*

shows $\text{mat-of-rows-list } nc \text{ rs } *_v \text{vec-of-list } ls =$

$\text{vec-of-list } (\text{map } (\lambda r. \text{vec-of-list } r \cdot \text{vec-of-list } ls) \text{ rs})$

<proof>

lemma *mtx-row-length*:

list-all ($\lambda r. \text{length } r = \text{length } \text{signs}$) (*map* (*mtx-row signs*) *ls*)
 <proof>

thm *construct-lhs-vector-def*

thm *poly-roots-finite*

lemma *construct-lhs-vector-clean:*

assumes $p \neq 0$

assumes $i < \text{length } \text{signs}$

shows (*construct-lhs-vector* p qs signs) \$ $i =$

$\text{card } \{x. \text{poly } p \ x = 0 \wedge ((\text{consistent-sign-vec-copr } qs \ x) = (\text{nth } \text{signs } i))\}$

<proof>

lemma *construct-lhs-vector-cleaner:*

assumes $p \neq 0$

shows (*construct-lhs-vector* p qs signs) =

vec-of-list (*map* ($\lambda s. \text{rat-of-int } (\text{card } \{x. \text{poly } p \ x = 0 \wedge ((\text{consistent-sign-vec-copr } qs \ x) = s)\})$) signs)

<proof>

lemma *z-signs:*

assumes *list-all* ($\lambda i. i < \text{length } \text{signs}$) I

assumes *list-all* ($\lambda s. s = 1 \vee s = -1$) signs

shows ($z \ I \ \text{signs} = 1$) \vee ($z \ I \ \text{signs} = -1$) <proof>

lemma *z-lemma:*

fixes $I:: \text{nat list}$

fixes $\text{sign}:: \text{rat list}$

assumes *consistent:* $\text{sign} \in \text{set } (\text{characterize-consistent-signs-at-roots-copr } p \ qs)$

assumes *welldefined:* *list-constr* I ($\text{length } qs$)

shows ($z \ I \ \text{sign} = 1$) \vee ($z \ I \ \text{sign} = -1$)

<proof>

lemma *in-set:*

fixes $p:: \text{real poly}$

assumes *nonzero:* $p \neq 0$

fixes $qs:: \text{real poly list}$

fixes $I:: \text{nat list}$

fixes $\text{sign}:: \text{rat list}$

fixes $x:: \text{real}$

assumes *root-p:* $x \in \{x. \text{poly } p \ x = 0\}$

assumes *sign-fix:* $\text{sign} = \text{consistent-sign-vec-copr } qs \ x$

assumes *welldefined:* *list-constr* I ($\text{length } qs$)

shows $\text{sign} \in \text{set } (\text{characterize-consistent-signs-at-roots-copr } p \ qs)$

<proof>

lemma nonzero-product:

fixes p :: *real poly*
assumes *nonzero*: $p \neq 0$
fixes qs :: *real poly list*
assumes *pairwise-rel-prime-1*: $\forall q. ((List.member\ qs\ q) \longrightarrow (coprime\ p\ q))$
fixes I :: *nat list*
fixes x :: *real*
assumes *root-p*: $x \in \{x. poly\ p\ x = 0\}$
assumes *welldefined*: *list-constr* I (*length* qs)
shows $(poly\ (prod-list\ (retrieve-polys\ qs\ I))\ x > 0) \vee (poly\ (prod-list\ (retrieve-polys\ qs\ I))\ x < 0)$
<proof>

lemma horiz-vector-helper-pos-ind:

fixes p :: *real poly*
assumes *nonzero*: $p \neq 0$
fixes qs :: *real poly list*
assumes *pairwise-rel-prime-1*: $\forall q. ((List.member\ qs\ q) \longrightarrow (coprime\ p\ q))$
fixes I :: *nat list*
fixes $sign$:: *rat list*
fixes x :: *real*
assumes *root-p*: $x \in \{x. poly\ p\ x = 0\}$
assumes *sign-fix*: $sign = consistent-sign-vec-copr\ qs\ x$
shows $(list-constr\ I\ (length\ qs)) \longrightarrow (poly\ (prod-list\ (retrieve-polys\ qs\ I))\ x > 0)$
 $\longleftrightarrow (z\ I\ sign = 1)$
<proof>

lemma horiz-vector-helper-pos:

fixes p :: *real poly*
assumes *nonzero*: $p \neq 0$
fixes qs :: *real poly list*
assumes *pairwise-rel-prime-1*: $\forall q. ((List.member\ qs\ q) \longrightarrow (coprime\ p\ q))$
fixes I :: *nat list*
fixes $sign$:: *rat list*
fixes x :: *real*
assumes *root-p*: $x \in \{x. poly\ p\ x = 0\}$
assumes *sign-fix*: $sign = consistent-sign-vec-copr\ qs\ x$
assumes *welldefined*: *list-constr* I (*length* qs)
shows $(poly\ (prod-list\ (retrieve-polys\ qs\ I))\ x > 0) \longleftrightarrow (z\ I\ sign = 1)$
<proof>

lemma horiz-vector-helper-neg:

fixes p :: *real poly*
assumes *nonzero*: $p \neq 0$
fixes qs :: *real poly list*
assumes *pairwise-rel-prime-1*: $\forall q. ((List.member\ qs\ q) \longrightarrow (coprime\ p\ q))$
fixes I :: *nat list*

fixes *sign*:: *rat list*
fixes *x*:: *real*
assumes *root-p*: $x \in \{x. \text{poly } p \ x = 0\}$
assumes *sign-fix*: $\text{sign} = \text{consistent-sign-vec-copr } qs \ x$
assumes *welldefined*: $\text{list-constr } I \ (\text{length } qs)$
shows $(\text{poly } (\text{prod-list } (\text{retrieve-polys } qs \ I)) \ x < 0) \longleftrightarrow (z \ I \ \text{sign} = -1)$
<proof>

lemma *vec-of-list-dot-rewrite*:
assumes $\text{length } xs = \text{length } ys$
shows $\text{vec-of-list } xs \cdot \text{vec-of-list } ys =$
 $\text{sum-list } (\text{map2 } (*) \ xs \ ys)$
<proof>

lemma *lhs-dot-rewrite*:
fixes *p*:: *real poly*
fixes *qs*:: *real poly list*
fixes *I*:: *nat list*
fixes *signs*:: *rat list list*
assumes *nonzero*: $p \neq 0$
shows
 $(\text{vec-of-list } (\text{mtx-row } signs \ I) \cdot (\text{construct-lhs-vector } p \ qs \ signs)) =$
 $\text{sum-list } (\text{map } (\lambda s. (z \ I \ s) \ * \ \text{rat-of-int } (\text{card } \{x. \text{poly } p \ x = 0 \wedge \text{consistent-sign-vec-copr } qs \ x = s\})) \ signs)$
<proof>

lemma *sum-list-distinct-filter*:
fixes *f*:: $'a \Rightarrow \text{int}$
assumes *distinct* $xs \ \text{distinct } ys$
assumes $\text{set } ys \subseteq \text{set } xs$
assumes $\bigwedge x. x \in \text{set } xs - \text{set } ys \implies f \ x = 0$
shows $\text{sum-list } (\text{map } f \ xs) = \text{sum-list } (\text{map } f \ ys)$
<proof>

lemma *construct-lhs-vector-drop-consistent*:
fixes *p*:: *real poly*
fixes *qs*:: *real poly list*
fixes *I*:: *nat list*
fixes *signs*:: *rat list list*
assumes *nonzero*: $p \neq 0$
assumes *distinct-signs*: $\text{distinct } signs$
assumes *all-info*: $\text{set } (\text{characterize-consistent-signs-at-roots-copr } p \ qs) \subseteq \text{set } (signs)$
assumes *welldefined*: $\text{list-constr } I \ (\text{length } qs)$
shows
 $(\text{vec-of-list } (\text{mtx-row } signs \ I) \cdot (\text{construct-lhs-vector } p \ qs \ signs)) =$
 $(\text{vec-of-list } (\text{mtx-row } (\text{characterize-consistent-signs-at-roots-copr } p \ qs) \ I) \cdot$
 $(\text{construct-lhs-vector } p \ qs \ (\text{characterize-consistent-signs-at-roots-copr } p \ qs)))$

<proof>

lemma *matrix-equation-helper-step:*

fixes *p*:: real poly

fixes *qs*:: real poly list

fixes *I*:: nat list

fixes *signs*:: rat list list

assumes *nonzero*: $p \neq 0$

assumes *distinct-signs*: distinct signs

assumes *all-info*: $\text{set}(\text{characterize-consistent-signs-at-roots-copr } p \text{ } qs) \subseteq \text{set}(\text{signs})$

assumes *welldefined*: list-constr *I* (length *qs*)

assumes *pairwise-rel-prime-1*: $\forall q. ((\text{List.member } qs \text{ } q) \longrightarrow (\text{coprime } p \text{ } q))$

shows $(\text{vec-of-list } (\text{mtx-row } signs \text{ } I) \cdot (\text{construct-lhs-vector } p \text{ } qs \text{ } signs)) =$

$\text{rat-of-int } (\text{card } \{x. \text{poly } p \text{ } x = 0 \wedge \text{poly } (\text{prod-list } (\text{retrieve-polys } qs \text{ } I)) \text{ } x > 0\})$

—

$\text{rat-of-int } (\text{card } \{x. \text{poly } p \text{ } x = 0 \wedge \text{poly } (\text{prod-list } (\text{retrieve-polys } qs \text{ } I)) \text{ } x < 0\})$

<proof>

lemma *matrix-equation-main-step:*

fixes *p*:: real poly

fixes *qs*:: real poly list

fixes *I*:: nat list

fixes *signs*:: rat list list

assumes *nonzero*: $p \neq 0$

assumes *distinct-signs*: distinct signs

assumes *all-info*: $\text{set}(\text{characterize-consistent-signs-at-roots-copr } p \text{ } qs) \subseteq \text{set}(\text{signs})$

assumes *welldefined*: list-constr *I* (length *qs*)

assumes *pairwise-rel-prime-1*: $\forall q. ((\text{List.member } qs \text{ } q) \longrightarrow (\text{coprime } p \text{ } q))$

shows $(\text{vec-of-list } (\text{mtx-row } signs \text{ } I) \cdot (\text{construct-lhs-vector } p \text{ } qs \text{ } signs)) =$

$\text{construct-NoFI } p \text{ } (\text{retrieve-polys } qs \text{ } I)$

<proof>

lemma *map-vec-vec-of-list-eq-intro:*

assumes $\text{map } f \text{ } xs = \text{map } g \text{ } ys$

shows $\text{map-vec } f \text{ } (\text{vec-of-list } xs) = \text{map-vec } g \text{ } (\text{vec-of-list } ys)$

<proof>

theorem *matrix-equation:*

fixes *p*:: real poly

fixes *qs*:: real poly list

fixes *subsets*:: nat list list

fixes *signs*:: rat list list

assumes *nonzero*: $p \neq 0$

assumes *distinct-signs*: distinct signs

assumes *all-info*: $\text{set}(\text{characterize-consistent-signs-at-roots-copr } p \text{ } qs) \subseteq \text{set}(\text{signs})$

assumes *pairwise-rel-prime*: $\forall q. ((\text{List.member } qs \text{ } q) \longrightarrow (\text{coprime } p \text{ } q))$

assumes *welldefined: all-list-constr (subsets) (length qs)*
shows *satisfy-equation p qs subsets signs*
<proof>

definition *roots:: real poly \Rightarrow real set*
where *roots p = {x. poly p x = 0}*

definition *sgn::'a::linordered-field \Rightarrow rat*
where *sgn x = (if x > 0 then 1*
else if x < 0 then -1
else 0)

definition *sgn-vec::real poly list \Rightarrow real \Rightarrow rat list*
where *sgn-vec qs x \equiv map (sgn \circ (λ q. poly q x)) qs*

definition *consistent-signs-at-roots:: real poly \Rightarrow real poly list \Rightarrow rat list set*
where *consistent-signs-at-roots p qs =*
(sgn-vec qs) ' (roots p)

lemma *consistent-signs-at-roots-eq:*
assumes *p \neq 0*
shows *consistent-signs-at-roots p qs =*
set (characterize-consistent-signs-at-roots p qs)
<proof>

abbreviation *w-vec:: real poly \Rightarrow real poly list \Rightarrow rat list list \Rightarrow rat vec*
where *w-vec \equiv construct-lhs-vector*

abbreviation *v-vec:: real poly \Rightarrow real poly list \Rightarrow nat list list \Rightarrow rat vec*
where *v-vec \equiv construct-rhs-vector*

abbreviation *M-mat:: rat list list \Rightarrow nat list list \Rightarrow rat mat*
where *M-mat \equiv matrix-A*

theorem *matrix-equation-pretty:*
assumes *p \neq 0*
assumes *\bigwedge q. q \in set qs \implies coprime p q*
assumes *distinct signs*
assumes *consistent-signs-at-roots p qs \subseteq set signs*
assumes *\bigwedge l i. l \in set subsets \implies i \in set l \implies i < length qs*
shows *M-mat signs subsets *_v w-vec p qs signs = v-vec p qs subsets*
<proof>

end
theory *BKR-Proofs*
imports *Matrix-Equation-Construction*

begin

definition *well-def-signs*:: $\text{nat} \Rightarrow \text{rat list list} \Rightarrow \text{bool}$
where *well-def-signs num-polys sign-conds* $\equiv \forall \text{ signs} \in \text{set}(\text{sign-conds}). (\text{length signs} = \text{num-polys})$

definition *satisfies-properties*:: $\text{real poly} \Rightarrow \text{real poly list} \Rightarrow \text{nat list list} \Rightarrow \text{rat list list} \Rightarrow \text{rat mat} \Rightarrow \text{bool}$

where *satisfies-properties p qs subsets signs matrix* =
 (*all-list-constr subsets (length qs) \wedge well-def-signs (length qs) signs \wedge distinct signs*
 \wedge *satisfy-equation p qs subsets signs \wedge invertible-mat matrix \wedge matrix = matrix-A signs subsets*
 \wedge *set (characterize-consistent-signs-at-roots-copr p qs) \subseteq set(signs)*
)

12 Base Case

lemma *mat-base-case*:

shows *matrix-A* $[[1], [-1]] \ [\ [], [0]] = (\text{mat-of-rows-list } 2 \ [[1, 1], [1, -1]])$
 $\langle \text{proof} \rangle$

lemma *base-case-sgas*:

fixes *q p*:: *real poly*
assumes *nonzero*: $p \neq 0$
assumes *rel-prime*: *coprime p q*
shows *set (characterize-consistent-signs-at-roots-copr p [q]) \subseteq $\{[1], [-1]\}$*
 $\langle \text{proof} \rangle$

lemma *base-case-sgas-alt*:

fixes *p*:: *real poly*
fixes *qs*:: *real poly list*
assumes *len1*: $\text{length qs} = 1$
assumes *nonzero*: $p \neq 0$
assumes *rel-prime*: $\forall q. (\text{List.member qs } q) \longrightarrow \text{coprime } p \ q$
shows *set (characterize-consistent-signs-at-roots-copr p qs) \subseteq $\{[1], [-1]\}$*
 $\langle \text{proof} \rangle$

lemma *base-case-satisfy-equation*:

fixes *q p*:: *real poly*
assumes *nonzero*: $p \neq 0$
assumes *rel-prime*: *coprime p q*
shows *satisfy-equation p [q] $[\ [], [0]] \ [[1], [-1]]$*
 $\langle \text{proof} \rangle$

lemma *base-case-satisfy-equation-alt*:

fixes *p*:: *real poly*
fixes *qs*:: *real poly list*
assumes *len1*: $\text{length qs} = 1$
assumes *nonzero*: $p \neq 0$

assumes *rel-prime*: $\forall q. (\text{List.member } qs \ q) \longrightarrow \text{coprime } p \ q$
shows *satisfy-equation* $p \ qs \ [\ [], [0]] \ [\ [1], [-1]]$
 $\langle \text{proof} \rangle$

lemma *base-case-matrix-eq*:
fixes $q \ p :: \text{real poly}$
assumes *nonzero*: $p \neq 0$
assumes *rel-prime*: *coprime* $p \ q$
shows $(\text{mult-mat-vec } (\text{mat-of-rows-list } 2 \ [\ [1, 1], [1, -1]]]) (\text{construct-lhs-vector } p \ [q] \ [\ [1], [-1]]]) =$
 $(\text{construct-rhs-vector } p \ [q] \ [\ [], [0]])$
 $\langle \text{proof} \rangle$

lemma *less-two*:
shows $j < \text{Suc } (\text{Suc } 0) \longleftrightarrow j = 0 \vee j = 1 \langle \text{proof} \rangle$

lemma *inverse-mat-base-case*:
shows *inverts-mat* $(\text{mat-of-rows-list } 2 \ [\ [1/2, 1/2], [1/2, -(1/2)]] :: \text{rat mat}) (\text{mat-of-rows-list } 2 \ [\ [1, 1], [1, -1]] :: \text{rat mat})$
 $\langle \text{proof} \rangle$

lemma *inverse-mat-base-case-2*:
shows *inverts-mat* $(\text{mat-of-rows-list } 2 \ [\ [1, 1], [1, -1]] :: \text{rat mat}) (\text{mat-of-rows-list } 2 \ [\ [1/2, 1/2], [1/2, -(1/2)]] :: \text{rat mat})$
 $\langle \text{proof} \rangle$

lemma *base-case-invertible-mat*:
shows *invertible-mat* $(\text{matrix-A } [\ [1], [-1]] \ [\ [], [0]])$
 $\langle \text{proof} \rangle$

13 Inductive Step

13.1 Lemmas on smashing subsets and signs

lemma *signs-smash-property*:
fixes $signs1 \ signs2 :: 'a \ \text{list list}$
fixes $a \ b :: \text{nat}$
shows $\forall (elem :: 'a \ \text{list}). (elem \in (\text{set } (\text{signs-smash } signs1 \ signs2))) \longrightarrow$
 $(\exists n \ m :: \text{nat}.$
 $elem = ((nth \ signs1 \ n) @ (nth \ signs2 \ m)))$
 $\langle \text{proof} \rangle$

lemma *signs-smash-property-set*:
fixes $signs1 \ signs2 :: 'a \ \text{list list}$
fixes $a \ b :: \text{nat}$
shows $\forall (elem :: 'a \ \text{list}). (elem \in (\text{set } (\text{signs-smash } signs1 \ signs2))) \longrightarrow$
 $(\exists (elem1 :: 'a \ \text{list}). \exists (elem2 :: 'a \ \text{list}).$
 $(elem1 \in \text{set}(signs1) \wedge elem2 \in \text{set}(signs2) \wedge elem = (elem1 @ elem2)))$
 $\langle \text{proof} \rangle$

lemma *subsets-smash-property*:
fixes *subsets1 subsets2* :: *nat list list*
fixes *n*:: *nat*
shows \forall (*elem* :: *nat list*). (*List.member* (*subsets-smash* *n subsets1 subsets2*)
elem) \longrightarrow
 $(\exists$ (*elem1*::*nat list*). \exists (*elem2*::*nat list*).
(*elem1* \in *set(subsets1)* \wedge *elem2* \in *set(subsets2)* \wedge *elem* = (*elem1* @ ((*map*
((+) *n*) *elem2*))))))
 \langle *proof* \rangle

13.2 Well-defined subsets preserved when smashing

lemma *list-constr-append*:
list-constr a n \wedge *list-constr b n* \longrightarrow *list-constr (a@b) n*
 \langle *proof* \rangle

lemma *well-def-step*:
fixes *qs1 qs2* :: *real poly list*
fixes *subsets1 subsets2* :: *nat list list*
assumes *well-def-subsets1*: *all-list-constr* (*subsets1*) (*length qs1*)
assumes *well-def-subsets2*: *all-list-constr* (*subsets2*) (*length qs2*)
shows *all-list-constr* ((*subsets-smash* (*length qs1*) *subsets1 subsets2*))
(*length (qs1 @ qs2)*)
 \langle *proof* \rangle

13.3 Well def signs preserved when smashing

lemma *well-def-signs-step*:
fixes *qs1 qs2* :: *real poly list*
fixes *signs1 signs2* :: *rat list list*
assumes *length qs1* > 0
assumes *length qs2* > 0
assumes *well-def1*: *well-def-signs* (*length qs1*) *signs1*
assumes *well-def2*: *well-def-signs* (*length qs2*) *signs2*
shows *well-def-signs* (*length (qs1@qs2)*) (*signs-smash signs1 signs2*)
 \langle *proof* \rangle

13.4 Distinct signs preserved when smashing

lemma *distinct-map-append*:
assumes *distinct ls*
shows *distinct* (*map* ((@) *xs*) *ls*)
 \langle *proof* \rangle

lemma *length-eq-append*:
assumes *length y* = *length b*
shows (*x @ y* = *a @ b*) \longleftrightarrow *x=a* \wedge *y = b*
 \langle *proof* \rangle

lemma *distinct-step*:
fixes *qs1 qs2* :: *real poly list*
fixes *signs1 signs2* :: *rat list list*
assumes *well-def1*: *well-def-signs n1 signs1*
assumes *well-def2*: *well-def-signs n2 signs2*
assumes *distinct1*: *distinct signs1*
assumes *distinct2*: *distinct signs2*
shows *distinct (signs-smash signs1 signs2)*
<proof>

13.5 Consistent sign assignments preserved when smashing

lemma *subset-smash-signs*:
fixes *a1 b1 a2 b2*:: *rat list list*
assumes *sub1*: *set a1* \subseteq *set a2*
assumes *sub2*: *set b1* \subseteq *set b2*
shows *set (signs-smash a1 b1)* \subseteq *set (signs-smash a2 b2)*
<proof>

lemma *subset-helper*:
fixes *p*:: *real poly*
fixes *qs1 qs2* :: *real poly list*
fixes *signs1 signs2* :: *rat list list*
shows $\forall x \in \text{set } (\text{characterize-consistent-signs-at-roots-copr } p \text{ } (qs1 @ qs2)).$
 $\exists x1 \in \text{set } (\text{characterize-consistent-signs-at-roots-copr } p \text{ } qs1).$
 $\exists x2 \in \text{set } (\text{characterize-consistent-signs-at-roots-copr } p \text{ } qs2).$
 $x = x1 @ x2$
<proof>

lemma *subset-step*:
fixes *p*:: *real poly*
fixes *qs1 qs2* :: *real poly list*
fixes *signs1 signs2* :: *rat list list*
assumes *csa1*: *set (characterize-consistent-signs-at-roots-copr p qs1)* \subseteq *set (signs1)*
assumes *csa2*: *set (characterize-consistent-signs-at-roots-copr p qs2)* \subseteq *set (signs2)*

shows *set (characterize-consistent-signs-at-roots-copr p*
 $(qs1 @ qs2))$
 \subseteq *set (signs-smash signs1 signs2)*
<proof>

13.6 Main Results

lemma *dim-row-mat-of-rows-list[simp]*:
shows *dim-row (mat-of-rows-list nr ls) = length ls*
<proof>

lemma *dim-col-mat-of-rows-list[simp]*:
shows *dim-col (mat-of-rows-list nr ls) = nr*
<proof>

lemma *dim-row-matrix-A[simp]*:
shows $\text{dim-row } (\text{matrix-A } \text{signs } \text{subsets}) = \text{length } \text{subsets}$
 $\langle \text{proof} \rangle$

lemma *dim-col-matrix-A[simp]*:
shows $\text{dim-col } (\text{matrix-A } \text{signs } \text{subsets}) = \text{length } \text{signs}$
 $\langle \text{proof} \rangle$

lemma *length-signs-smash*:
shows
 $\text{length } (\text{signs-smash } \text{signs1 } \text{signs2}) = \text{length } \text{signs1} * \text{length } \text{signs2}$
 $\langle \text{proof} \rangle$

lemma *length-subsets-smash*:
shows
 $\text{length } (\text{subsets-smash } n \text{ subs1 } \text{subs2}) = \text{length } \text{subs1} * \text{length } \text{subs2}$
 $\langle \text{proof} \rangle$

lemma *length-eq-concat*:
assumes $\bigwedge x. x \in \text{set } \text{ls} \implies \text{length } x = n$
assumes $i < n * \text{length } \text{ls}$
shows $\text{concat } \text{ls} ! i = \text{ls} ! (i \text{ div } n) ! (i \text{ mod } n)$
 $\langle \text{proof} \rangle$

lemma *z-append*:
assumes $\bigwedge i. i \in \text{set } \text{xs} \implies i < \text{length } \text{as}$
shows $z (\text{xs} @ (\text{map } ((+) (\text{length } \text{as})) \text{ys})) (\text{as} @ \text{bs}) = z \text{xs } \text{as} * z \text{ys } \text{bs}$
 $\langle \text{proof} \rangle$

lemma *matrix-construction-is-kronecker-product*:
fixes $qs1 :: \text{real poly list}$
fixes $\text{subs1 } \text{subs2} :: \text{nat list list}$
fixes $\text{signs1 } \text{signs2} :: \text{rat list list}$

assumes $\bigwedge l i. l \in \text{set } \text{subs1} \implies i \in \text{set } l \implies i < n1$
assumes $\bigwedge j. j \in \text{set } \text{signs1} \implies \text{length } j = n1$
shows
 $(\text{matrix-A } (\text{signs-smash } \text{signs1 } \text{signs2}) (\text{subsets-smash } n1 \text{ subs1 } \text{subs2})) =$
 $\text{kronecker-product } (\text{matrix-A } \text{signs1 } \text{subs1}) (\text{matrix-A } \text{signs2 } \text{subs2})$
 $\langle \text{proof} \rangle$

lemma *inductive-step*:
fixes $p :: \text{real poly}$
fixes $qs1 \text{ } qs2 :: \text{real poly list}$
fixes $\text{subsets1 } \text{subsets2} :: \text{nat list list}$
fixes $\text{signs1 } \text{signs2} :: \text{rat list list}$

assumes nonzero: $p \neq 0$
assumes nontriv1: $\text{length } qs1 > 0$
assumes nontriv2: $\text{length } qs2 > 0$
assumes pairwise-rel-prime1: $\forall q. ((\text{List.member } qs1 \ q) \longrightarrow (\text{coprime } p \ q))$
assumes pairwise-rel-prime2: $\forall q. ((\text{List.member } qs2 \ q) \longrightarrow (\text{coprime } p \ q))$
assumes welldefined-signs1: $\text{well-def-signs } (\text{length } qs1) \ \text{signs1}$
assumes welldefined-signs2: $\text{well-def-signs } (\text{length } qs2) \ \text{signs2}$
assumes distinct-signs1: $\text{distinct } \text{signs1}$
assumes distinct-signs2: $\text{distinct } \text{signs2}$
assumes all-info1: $\text{set } (\text{characterize-consistent-signs-at-roots-copr } p \ qs1) \subseteq \text{set}(\text{signs1})$
assumes all-info2: $\text{set } (\text{characterize-consistent-signs-at-roots-copr } p \ qs2) \subseteq \text{set}(\text{signs2})$
assumes welldefined-subsets1: $\text{all-list-constr } (\text{subsets1}) \ (\text{length } qs1)$
assumes welldefined-subsets2: $\text{all-list-constr } (\text{subsets2}) \ (\text{length } qs2)$
assumes invertibleMtx1: $\text{invertible-mat } (\text{matrix-A } \text{signs1 } \text{subsets1})$
assumes invertibleMtx2: $\text{invertible-mat } (\text{matrix-A } \text{signs2 } \text{subsets2})$
shows satisfy-equation $p \ (qs1@qs2) \ (\text{subsets-smash } (\text{length } qs1) \ \text{subsets1} \ \text{subsets2})$
 $(\text{signs-smash } \text{signs1} \ \text{signs2})$
 $\wedge \text{invertible-mat } (\text{matrix-A } (\text{signs-smash } \text{signs1} \ \text{signs2}) \ (\text{subsets-smash } (\text{length } qs1) \ \text{subsets1} \ \text{subsets2}))$
 $\langle \text{proof} \rangle$

14 Reduction Step Proofs

definition get-matrix:: $(\text{rat mat} \times (\text{nat list list} \times \text{rat list list})) \Rightarrow \text{rat mat}$
where $\text{get-matrix } \text{data} = \text{fst}(\text{data})$

definition get-subsets:: $(\text{rat mat} \times (\text{nat list list} \times \text{rat list list})) \Rightarrow \text{nat list list}$
where $\text{get-subsets } \text{data} = \text{fst}(\text{snd}(\text{data}))$

definition get-signs:: $(\text{rat mat} \times (\text{nat list list} \times \text{rat list list})) \Rightarrow \text{rat list list}$
where $\text{get-signs } \text{data} = \text{snd}(\text{snd}(\text{data}))$

definition reduction-signs:: $\text{real poly} \Rightarrow \text{real poly list} \Rightarrow \text{rat list list} \Rightarrow \text{nat list list}$
 $\Rightarrow \text{rat mat} \Rightarrow \text{rat list list}$
where $\text{reduction-signs } p \ qs \ \text{signs} \ \text{subsets} \ \text{matr} =$
 $(\text{take-indices } \text{signs} \ (\text{find-nonzeros-from-input-vec } (\text{solve-for-lhs } p \ qs \ \text{subsets} \ \text{matr})))$

definition reduction-subsets:: $\text{real poly} \Rightarrow \text{real poly list} \Rightarrow \text{rat list list} \Rightarrow \text{nat list list}$
 $\Rightarrow \text{rat mat} \Rightarrow \text{nat list list}$
where $\text{reduction-subsets } p \ qs \ \text{signs} \ \text{subsets} \ \text{matr} =$
 $(\text{take-indices } \text{subsets} \ (\text{rows-to-keep } (\text{reduce-mat-cols } \text{matr} \ (\text{solve-for-lhs } p \ qs \ \text{subsets} \ \text{matr}))))$

lemma reduction-signs-is-get-signs: $\text{reduction-signs } p \ qs \ \text{signs} \ \text{subsets} \ m = \text{get-signs}$
 $(\text{reduce-system } p \ (qs, (m, (\text{subsets}, \text{signs}))))$
 $\langle \text{proof} \rangle$

lemma *reduction-subsets-is-get-subsets*: *reduction-subsets p qs signs subsets m = get-subsets (reduce-system p (qs, (m, (subsets, signs))))*
 ⟨proof⟩

lemma *find-zeros-from-vec-prop*:
fixes *input-vec* :: *rat vec*
shows $\forall n < (\text{dim-vec } \text{input-vec}). ((\text{input-vec } \$ n \neq 0) \longleftrightarrow \text{List.member } (\text{find-nonzeros-from-input-vec } \text{input-vec}) n)$
 ⟨proof⟩

14.1 Showing sign conditions preserved when reducing

lemma *take-indices-lem*:
fixes *p* :: *real poly*
fixes *qs* :: *real poly list*
fixes *arb-list* :: '*a list list*
fixes *index-list* :: *nat list*
fixes *n* :: *nat*
assumes *lest*: $n < \text{length } (\text{take-indices } \text{arb-list } \text{index-list})$
assumes *well-def*: $\forall q. (\text{List.member } \text{index-list } q \longrightarrow q < \text{length } \text{arb-list})$
shows $\exists k < \text{length } \text{arb-list}. (\text{take-indices } \text{arb-list } \text{index-list}) ! n = \text{arb-list} ! k$
 ⟨proof⟩

lemma *invertible-means-mult-id*:
fixes *A* :: '*a*::*field mat*
assumes *asm*: *invertible-mat A*
shows *matr-option (dim-row A)*
 $(\text{mat-inverse } (A)) * A = 1_m (\text{dim-row } A)$
 ⟨proof⟩

lemma *dim-invertible*:
fixes *A* :: '*a*::*field mat*
fixes *n* :: *nat*
assumes *asm*: *invertible-mat A*
assumes *dim*: $A \in \text{carrier-mat } n n$
shows *matr-option (dim-row A)*
 $(\text{mat-inverse } (A)) \in \text{carrier-mat } n n$
 ⟨proof⟩

lemma *mult-assoc*:
fixes *A B* :: *rat mat*
fixes *v* :: *rat vec*
fixes *n* :: *nat*
assumes $A \in \text{carrier-mat } n n$
assumes $B \in \text{carrier-mat } n n$
assumes $\text{dim-vec } v = n$
shows $A *_v (\text{mult-mat-vec } B v) = (A * B) *_v v$
 ⟨proof⟩

lemma *size-of-mat*:
fixes *subsets* :: *nat list list*
fixes *signs* :: *rat list list*
shows (*matrix-A signs subsets*) \in *carrier-mat* (*length subsets*) (*length signs*)
 \langle *proof* \rangle

lemma *size-of-lhs*:
fixes *p*:: *real poly*
fixes *qs* :: *real poly list*
fixes *signs* :: *rat list list*
shows *dim-vec* (*construct-lhs-vector p qs signs*) = *length signs*
 \langle *proof* \rangle

lemma *size-of-rhs*:
fixes *p*:: *real poly*
fixes *qs* :: *real poly list*
fixes *subsets* :: *nat list list*
shows *dim-vec* (*construct-rhs-vector p qs subsets*) = *length subsets*
 \langle *proof* \rangle

lemma *same-size*:
fixes *p*:: *real poly*
fixes *qs* :: *real poly list*
fixes *subsets* :: *nat list list*
fixes *signs* :: *rat list list*
assumes *invertible-mat*: *invertible-mat* (*matrix-A signs subsets*)
shows *length subsets* = *length signs*
 \langle *proof* \rangle

lemma *mat-equal-list-lem*:
fixes *A*:: '*a*::*field mat*
fixes *B*:: '*a*::*field mat*
shows (*dim-row A* = *dim-row B* \wedge *dim-col A* = *dim-col B* \wedge *mat-to-list A* =
mat-to-list B)
 \implies *A* = *B*
 \langle *proof* \rangle

lemma *mat-inverse-same*: *mat-inverse-var A* = *mat-inverse A*
 \langle *proof* \rangle

lemma *construct-lhs-matches-solve-for-lhs*:
fixes *p*:: *real poly*
fixes *qs* :: *real poly list*
fixes *subsets* :: *nat list list*
fixes *signs* :: *rat list list*
assumes *match*: *satisfy-equation p qs subsets signs*
assumes *invertible-mat*: *invertible-mat* (*matrix-A signs subsets*)
shows (*construct-lhs-vector p qs signs*) = *solve-for-lhs p qs subsets* (*matrix-A*)

signs subsets)
(*proof*)

lemma *reduction-signs-set-helper-lemma:*

fixes *A*:: *rat list set*
fixes *C*:: *rat vec*
fixes *B*:: *rat list list*
assumes *dim-vec C = length B*
assumes *sub: A ⊆ set(B)*
assumes *not-in-hyp: ∀ n < dim-vec C. C \$ n = 0 → (nth B n) ∉ A*
shows *A ⊆ set (take-indices B*
(find-nonzeros-from-input-vec C))

(*proof*)

lemma *reduction-signs-set-helper-lemma2:*

fixes *A*:: *rat list set*
fixes *C*:: *rat vec*
fixes *B*:: *rat list list*
assumes *dist: distinct B*
assumes *eq-len: dim-vec C = length B*
assumes *sub: A ⊆ set(B)*
assumes *not-in-hyp: ∀ n < dim-vec C. C \$ n ≠ 0 → (nth B n) ∈ A*
shows *set (take-indices B*
(find-nonzeros-from-input-vec C)) ⊆ A

(*proof*)

lemma *reduction-doesnt-break-things-signs:*

fixes *p*:: *real poly*
fixes *qs*:: *real poly list*
fixes *subsets*:: *nat list list*
fixes *signs*:: *rat list list*
assumes *nonzero: p ≠ 0*
assumes *welldefined-signs1: well-def-signs (length qs) signs*
assumes *distinct-signs: distinct signs*
assumes *all-info: set (characterize-consistent-signs-at-roots-copr p qs) ⊆ set(signs)*
assumes *match: satisfy-equation p qs subsets signs*
assumes *invertible-mat: invertible-mat (matrix-A signs subsets)*
shows *set (characterize-consistent-signs-at-roots-copr p qs) ⊆ set(reduction-signs*
p qs signs subsets (matrix-A signs subsets))
(*proof*)

lemma *reduction-deletes-bad-sign-conds:*

fixes *p*:: *real poly*
fixes *qs*:: *real poly list*
fixes *subsets*:: *nat list list*
fixes *signs*:: *rat list list*
assumes *nonzero: p ≠ 0*

assumes *welldefined-signs1*: *well-def-signs (length qs) signs*
assumes *distinct-signs*: *distinct signs*
assumes *all-info*: *set (characterize-consistent-signs-at-roots-copr p qs) \subseteq set(signs)*
assumes *match*: *satisfy-equation p qs subsets signs*
assumes *invertible-mat*: *invertible-mat (matrix-A signs subsets)*
shows *set (characterize-consistent-signs-at-roots-copr p qs) = set(reduction-signs p qs signs subsets (matrix-A signs subsets))*
<proof>

theorem *reduce-system-sign-conditions*:

fixes *p*:: *real poly*
fixes *qs* :: *real poly list*
fixes *subsets* :: *nat list list*
fixes *signs* :: *rat list list*
assumes *nonzero*: *p \neq 0*
assumes *welldefined-signs1*: *well-def-signs (length qs) signs*
assumes *distinct-signs*: *distinct signs*
assumes *all-info*: *set (characterize-consistent-signs-at-roots-copr p qs) \subseteq set(signs)*
assumes *match*: *satisfy-equation p qs subsets signs*
assumes *invertible-mat*: *invertible-mat (matrix-A signs subsets)*
shows *set (get-signs (reduce-system p (qs, ((matrix-A signs subsets), (subsets, signs)))))) = set (characterize-consistent-signs-at-roots-copr p qs)*
<proof>

14.2 Showing matrix equation preserved when reducing

lemma *rows-to-keep-lem*:

fixes *A*:: (*a*::*field*) *mat*
shows $\bigwedge ell. ell \in set (rows-to-keep A) \implies ell < dim-row A$
<proof>

lemma *reduce-system-matrix-equation-preserved*:

fixes *p*:: *real poly*
fixes *qs* :: *real poly list*
fixes *subsets* :: *nat list list*
fixes *signs* :: *rat list list*
assumes *nonzero*: *p \neq 0*
assumes *welldefined-signs*: *well-def-signs (length qs) signs*
assumes *welldefined-subsets*: *all-list-constr (subsets) (length qs)*
assumes *distinct-signs*: *distinct signs*
assumes *all-info*: *set (characterize-consistent-signs-at-roots-copr p qs) \subseteq set(signs)*
assumes *match*: *satisfy-equation p qs subsets signs*
assumes *invertible-mat*: *invertible-mat (matrix-A signs subsets)*
assumes *pairwise-rel-prime*: $\forall q. ((List.member qs q) \longrightarrow (coprime p q))$
shows *satisfy-equation p qs (get-subsets (reduce-system p (qs, ((matrix-A signs subsets), (subsets, signs))))))*
(get-signs (reduce-system p (qs, ((matrix-A signs subsets), (subsets, signs))))))
<proof>

14.3 Showing matrix preserved

lemma *reduce-system-matrix-signs-helper-aux:*

fixes p :: *real poly*
fixes qs :: *real poly list*
fixes $subsets$:: *nat list list*
fixes $signs$:: *rat list list*
fixes S :: *nat list*
assumes *well-def-h*: $\forall x. List.member\ S\ x \longrightarrow x < length\ signs$
assumes *nonzero*: $p \neq 0$
shows *alt-matrix-A* (*take-indices signs S*) *subsets* = *take-cols-from-matrix* (*alt-matrix-A signs subsets*) S
(*proof*)

lemma *reduce-system-matrix-signs-helper:*

fixes p :: *real poly*
fixes qs :: *real poly list*
fixes $subsets$:: *nat list list*
fixes $signs$:: *rat list list*
fixes S :: *nat list*
assumes *well-def-h*: $\forall x. List.member\ S\ x \longrightarrow x < length\ signs$
assumes *nonzero*: $p \neq 0$
shows *matrix-A* (*take-indices signs S*) *subsets* = *take-cols-from-matrix* (*matrix-A signs subsets*) S
(*proof*)

lemma *reduce-system-matrix-subsets-helper-aux:*

fixes p :: *real poly*
fixes qs :: *real poly list*
fixes $subsets$:: *nat list list*
fixes $signs$:: *rat list list*
fixes S :: *nat list*
assumes *inv*: $length\ subsets \geq length\ signs$
assumes *well-def-h*: $\forall x. List.member\ S\ x \longrightarrow x < length\ subsets$
assumes *nonzero*: $p \neq 0$
shows *alt-matrix-A signs* (*take-indices subsets S*) = *take-rows-from-matrix* (*alt-matrix-A signs subsets*) S
(*proof*)

lemma *reduce-system-matrix-subsets-helper:*

fixes p :: *real poly*
fixes qs :: *real poly list*
fixes $subsets$:: *nat list list*
fixes $signs$:: *rat list list*
fixes S :: *nat list*
assumes *nonzero*: $p \neq 0$
assumes *inv*: $length\ subsets \geq length\ signs$
assumes *well-def-h*: $\forall x. List.member\ S\ x \longrightarrow x < length\ subsets$

shows $\text{matrix-}A \text{ signs } (\text{take-indices subsets } S) = \text{take-rows-from-matrix } (\text{matrix-}A \text{ signs subsets}) S$
 ⟨proof⟩

lemma *reduce-system-matrix-match*:

fixes p :: real poly
fixes qs :: real poly list
fixes $subsets$:: nat list list
fixes $signs$:: rat list list
assumes nonzero: $p \neq 0$
assumes welldefined-signs1: well-def-signs (length qs) $signs$
assumes distinct-signs: distinct $signs$
assumes all-info: $\text{set } (\text{characterize-consistent-signs-at-roots-copr } p \text{ } qs) \subseteq \text{set}(signs)$
assumes match: satisfy-equation $p \text{ } qs \text{ } subsets \text{ } signs$
assumes inv: invertible-mat (matrix- $A \text{ signs subsets}$)
shows $\text{matrix-}A \text{ (get-signs (reduce-system } p \text{ (} qs, ((\text{matrix-}A \text{ signs subsets}), (subsets, signs))))$
 $\text{(get-subsets (reduce-system } p \text{ (} qs, ((\text{matrix-}A \text{ signs subsets}), (subsets, signs))))$
 $=$
 $\text{(get-matrix (reduce-system } p \text{ (} qs, ((\text{matrix-}A \text{ signs subsets}), (subsets, signs))))$
 ⟨proof⟩

14.4 Showing invertibility preserved when reducing

lemma *well-def-find-zeros-from-lhs-vec*:

fixes p :: real poly
fixes qs :: real poly list
fixes $subsets$:: nat list list
fixes $signs$:: rat list list
assumes len-eq: length $subsets = \text{length } signs$
assumes inv: invertible-mat (matrix- $A \text{ signs subsets}$)
assumes nonzero: $p \neq 0$
assumes welldefined-signs1: well-def-signs (length qs) $signs$
assumes distinct-signs: distinct $signs$
assumes all-info: $\text{set } (\text{characterize-consistent-signs-at-roots-copr } p \text{ } qs) \subseteq \text{set}(signs)$
assumes match: satisfy-equation $p \text{ } qs \text{ } subsets \text{ } signs$
shows $(\bigwedge j. j \in \text{set } (\text{find-nonzeros-from-input-vec } (\text{solve-for-lhs } p \text{ } qs \text{ } subsets \text{ } (\text{matrix-}A \text{ signs subsets}))) \implies$
 $j < \text{length } (\text{cols } (\text{matrix-}A \text{ signs subsets})))$
 ⟨proof⟩

lemma *take-cols-subsets-og-cols*:

fixes p :: real poly
fixes qs :: real poly list
fixes $subsets$:: nat list list
fixes $signs$:: rat list list
assumes len-eq: length $subsets = \text{length } signs$
assumes inv: invertible-mat (matrix- $A \text{ signs subsets}$)
assumes nonzero: $p \neq 0$

assumes *welldefined-signs1*: *well-def-signs (length qs) signs*
assumes *distinct-signs*: *distinct signs*
assumes *all-info*: *set (characterize-consistent-signs-at-roots-copr p qs) \subseteq set(signs)*
assumes *match*: *satisfy-equation p qs subsets signs*
shows *set (take-indices (cols (matrix-A signs subsets))*
(find-nonzeros-from-input-vec (solve-for-lhs p qs subsets (matrix-A signs
subsets))))
 \subseteq *set (cols (matrix-A signs subsets))*
<proof>

lemma *reduction-doesnt-break-things-invertibility-step1*:
fixes *p*:: *real poly*
fixes *qs* :: *real poly list*
fixes *subsets* :: *nat list list*
fixes *signs* :: *rat list list*
assumes *len-eq*: *length subsets = length signs*
assumes *inv*: *invertible-mat (matrix-A signs subsets)*
assumes *nonzero*: *p \neq 0*
assumes *welldefined-signs1*: *well-def-signs (length qs) signs*
assumes *distinct-signs*: *distinct signs*
assumes *all-info*: *set (characterize-consistent-signs-at-roots-copr p qs) \subseteq set(signs)*
assumes *match*: *satisfy-equation p qs subsets signs*
shows *vec-space.rank (length signs) (reduce-mat-cols (matrix-A signs subsets)*
(solve-for-lhs p qs subsets (matrix-A signs subsets))) =
(length (find-nonzeros-from-input-vec (solve-for-lhs p qs subsets (matrix-A signs
subsets))))
<proof>

lemma *rechar-take-cols*: *take-cols-var A B = take-cols-from-matrix A B*
<proof>

lemma *rows-and-cols-transpose*: *rows M = cols M^T*
<proof>

lemma *take-rows-and-take-cols*: *(take-rows-from-matrix M r) = (take-cols-from-matrix*
M^T r)^T
<proof>

lemma *reduction-doesnt-break-things-invertibility*:
fixes *p*:: *real poly*
fixes *qs* :: *real poly list*
fixes *subsets* :: *nat list list*
fixes *signs* :: *rat list list*
assumes *len-eq*: *length subsets = length signs*
assumes *inv*: *invertible-mat (matrix-A signs subsets)*
assumes *nonzero*: *p \neq 0*
assumes *welldefined-signs1*: *well-def-signs (length qs) signs*
assumes *distinct-signs*: *distinct signs*
assumes *all-info*: *set (characterize-consistent-signs-at-roots-copr p qs) \subseteq set(signs)*

assumes *match*: *satisfy-equation p qs subsets signs*
shows *invertible-mat (get-matrix (reduce-system p (qs, ((matrix-A signs subsets), (subsets, signs)))))*
 ⟨*proof*⟩

14.5 Well def signs preserved when reducing

lemma *reduction-doesnt-break-length-signs*:
fixes *p* :: *real poly*
fixes *qs* :: *real poly list*
fixes *subsets* :: *nat list list*
fixes *signs* :: *rat list list*
assumes *length-init* : $\forall x \in \text{set}(\text{signs}). \text{length } x = \text{length } qs$
assumes *sat-eq*: *satisfy-equation p qs subsets signs*
assumes *inv-mat*: *invertible-mat (matrix-A signs subsets)*
shows $\forall x \in \text{set}(\text{reduction-signs } p \text{ } qs \text{ } signs \text{ } subsets \text{ } (\text{matrix-A signs subsets})).$
 $\text{length } x = \text{length } qs$
 ⟨*proof*⟩

14.6 Distinct signs preserved when reducing

lemma *reduction-signs-are-distinct*:
fixes *p* :: *real poly*
fixes *qs* :: *real poly list*
fixes *subsets* :: *nat list list*
fixes *signs* :: *rat list list*
assumes *sat-eq*: *satisfy-equation p qs subsets signs*
assumes *inv-mat*: *invertible-mat (matrix-A signs subsets)*
assumes *distinct-init*: *distinct signs*
shows *distinct (reduction-signs p qs signs subsets (matrix-A signs subsets))*
 ⟨*proof*⟩

14.7 Well def subsets preserved when reducing

lemma *reduction-doesnt-break-subsets*:
fixes *p* :: *real poly*
fixes *qs* :: *real poly list*
fixes *subsets* :: *nat list list*
fixes *signs* :: *rat list list*
assumes *nonzero*: $p \neq 0$
assumes *length-init* : *all-list-constr subsets (length qs)*
assumes *sat-eq*: *satisfy-equation p qs subsets signs*
assumes *inv-mat*: *invertible-mat (matrix-A signs subsets)*
shows *all-list-constr (reduction-subsets p qs signs subsets (matrix-A signs subsets)) (length qs)*
 ⟨*proof*⟩

15 Overall Lemmas

lemma *combining-to-smash*: $\text{combine-systems } p \text{ (qs1, m1, (sub1, sgn1)) (qs2, m2, (sub2, sgn2))}$
 $= \text{smash-systems } p \text{ qs1 qs2 sub1 sub2 sgn1 sgn2 m1 m2}$
 ⟨proof⟩

lemma *getter-functions*: $\text{calculate-data } p \text{ qs} = (\text{get-matrix } (\text{calculate-data } p \text{ qs}),$
 $(\text{get-subsets } (\text{calculate-data } p \text{ qs}), \text{get-signs } (\text{calculate-data } p \text{ qs}))$
 ⟨proof⟩

15.1 Key properties preserved

15.1.1 Properties preserved when combining and reducing systems

lemma *combining-sys-satisfies-properties-helper*:
fixes $p :: \text{real poly}$
fixes $qs1 :: \text{real poly list}$
fixes $qs2 :: \text{real poly list}$
fixes $subsets1 \text{ subsets2} :: \text{nat list list}$
fixes $signs1 \text{ signs2} :: \text{rat list list}$
fixes $matrix1 \text{ matrix2} :: \text{rat mat}$
assumes $\text{nonzero: } p \neq 0$
assumes $\text{nontriv1: } \text{length } qs1 > 0$
assumes $\text{pairwise-rel-prime1: } \forall q. ((\text{List.member } qs1 \text{ } q) \longrightarrow (\text{coprime } p \text{ } q))$
assumes $\text{nontriv2: } \text{length } qs2 > 0$
assumes $\text{pairwise-rel-prime2: } \forall q. ((\text{List.member } qs2 \text{ } q) \longrightarrow (\text{coprime } p \text{ } q))$
assumes $\text{satisfies-properties-sys1: } \text{satisfies-properties } p \text{ qs1 subsets1 signs1 matrix1}$
assumes $\text{satisfies-properties-sys2: } \text{satisfies-properties } p \text{ qs2 subsets2 signs2 matrix2}$
shows $\text{satisfies-properties } p \text{ (qs1@qs2) (get-subsets (snd ((combine-systems } p$
 $(qs1,(\text{matrix1, (subsets1, signs1))) (qs2,(\text{matrix2, (subsets2, signs2))))))$
 $(\text{get-signs (snd ((combine-systems } p \text{ (qs1,(\text{matrix1, (subsets1, signs1))) (qs2,(\text{matrix2,}$
 $(\text{subsets2, signs2))))))$
 $(\text{get-matrix (snd ((combine-systems } p \text{ (qs1,(\text{matrix1, (subsets1, signs1))) (qs2,(\text{matrix2,}$
 $(\text{subsets2, signs2))))))$
 ⟨proof⟩

lemma *combining-sys-satisfies-properties*:
fixes $p :: \text{real poly}$
fixes $qs1 :: \text{real poly list}$
fixes $qs2 :: \text{real poly list}$
assumes $\text{nonzero: } p \neq 0$
assumes $\text{nontriv1: } \text{length } qs1 > 0$
assumes $\text{pairwise-rel-prime1: } \forall q. ((\text{List.member } qs1 \text{ } q) \longrightarrow (\text{coprime } p \text{ } q))$
assumes $\text{nontriv2: } \text{length } qs2 > 0$
assumes $\text{pairwise-rel-prime2: } \forall q. ((\text{List.member } qs2 \text{ } q) \longrightarrow (\text{coprime } p \text{ } q))$
assumes $\text{satisfies-properties-sys1: } \text{satisfies-properties } p \text{ qs1 (get-subsets (calculate-data$

```

p qs1)) (get-signs (calculate-data p qs1)) (get-matrix (calculate-data p qs1))
  assumes satisfies-properties-sys2: satisfies-properties p qs2 (get-subsets (calculate-data
p qs2)) (get-signs (calculate-data p qs2)) (get-matrix (calculate-data p qs2))
  shows satisfies-properties p (qs1@qs2) (get-subsets (snd ((combine-systems p
(qs1,calculate-data p qs1) (qs2,calculate-data p qs2))))))
  (get-signs (snd ((combine-systems p (qs1,calculate-data p qs1) (qs2,calculate-data
p qs2))))))
  (get-matrix (snd ((combine-systems p (qs1,calculate-data p qs1) (qs2,calculate-data
p qs2))))))
  <proof>

```

lemma *reducing-sys-satisfies-properties:*

```

fixes p:: real poly
fixes qs :: real poly list
fixes subsets :: nat list list
fixes signs :: rat list list
fixes matrix:: rat mat
assumes nonzero: p ≠ 0
assumes nontriv: length qs > 0
assumes pairwise-rel-prime: ∀ q. ((List.member qs q) → (coprime p q))
assumes satisfies-properties-sys: satisfies-properties p qs subsets signs matrix
shows satisfies-properties p qs (get-subsets (reduce-system p (qs,matrix,subsets,signs)))
  (get-signs (reduce-system p (qs,matrix,subsets,signs)))
  (get-matrix (reduce-system p (qs,matrix,subsets,signs)))
  <proof>

```

15.1.2 For length 1 qs

lemma *length-1-calculate-data-satisfies-properties:*

```

fixes p:: real poly
fixes qs :: real poly list
fixes subsets :: nat list list
fixes signs :: rat list list
assumes nonzero: p ≠ 0
assumes len1: length qs = 1
assumes pairwise-rel-prime: ∀ q. ((List.member qs q) → (coprime p q))
shows satisfies-properties p qs (get-subsets (calculate-data p qs)) (get-signs (calculate-data
p qs)) (get-matrix (calculate-data p qs))
  <proof>

```

15.1.3 For arbitrary qs

lemma *append-not-distinct-helper:* (List.member l1 m ∧ List.member l2 m) →
(distinct (l1@l2) = False)
<proof>

lemma *calculate-data-satisfies-properties:*

```

fixes p:: real poly
fixes qs :: real poly list
fixes subsets :: nat list list

```

```

fixes signs :: rat list list
shows (p ≠ 0 ∧ (length qs > 0) ∧ (∀ q. ((List.member qs q) → (coprime p q)))
)
  → satisfies-properties p qs (get-subsets (calculate-data p qs)) (get-signs (calculate-data
p qs)) (get-matrix (calculate-data p qs))
⟨proof⟩

```

15.2 Some key results on consistent sign assignments

lemma *find-consistent-signs-at-roots-len1*:

```

fixes p :: real poly
fixes qs :: real poly list
fixes subsets :: nat list list
fixes signs :: rat list list
assumes nonzero: p ≠ 0
assumes len1: length qs = 1
assumes pairwise-rel-prime: ∀ q. ((List.member qs q) → (coprime p q))
shows set (find-consistent-signs-at-roots p qs) = set (characterize-consistent-signs-at-roots-copr
p qs)
⟨proof⟩

```

lemma *smaller-sys-are-good*:

```

fixes p :: real poly
fixes qs1 :: real poly list
fixes qs2 :: real poly list
fixes subsets :: nat list list
fixes signs :: rat list list
assumes nonzero: p ≠ 0
assumes nontriv1: length qs1 > 0
assumes pairwise-rel-prime1: ∀ q. ((List.member qs1 q) → (coprime p q))
assumes nontriv2: length qs2 > 0
assumes pairwise-rel-prime2: ∀ q. ((List.member qs2 q) → (coprime p q))
assumes set (find-consistent-signs-at-roots p qs1) = set (characterize-consistent-signs-at-roots-copr
p qs1)
assumes set (find-consistent-signs-at-roots p qs2) = set (characterize-consistent-signs-at-roots-copr
p qs2)
shows set (snd (snd (reduce-system p (combine-systems p (qs1, calculate-data p qs1)
(qs2, calculate-data p qs2))))))
  = set (characterize-consistent-signs-at-roots-copr p (qs1@qs2))
⟨proof⟩

```

lemma *find-consistent-signs-at-roots-1*:

```

fixes p :: real poly
fixes qs :: real poly list
shows (p ≠ 0 ∧ length qs > 0 ∧
  (∀ q. ((List.member qs q) → (coprime p q)))) →
  set (find-consistent-signs-at-roots p qs) = set (characterize-consistent-signs-at-roots-copr
p qs)

```

<proof>

lemma *find-consistent-signs-at-roots-0:*

fixes *p:: real poly*

assumes $p \neq 0$

shows $set(find-consistent-signs-at-roots\ p\ []) =$

$set(characterize-consistent-signs-at-roots-copr\ p\ [])$

<proof>

lemma *find-consistent-signs-at-roots-copr:*

fixes *p:: real poly*

fixes *qs :: real poly list*

assumes $p \neq 0$

assumes $\bigwedge q. q \in set\ qs \implies coprime\ p\ q$

shows $set(find-consistent-signs-at-roots\ p\ qs) = set(characterize-consistent-signs-at-roots-copr\ p\ qs)$

<proof>

lemma *find-consistent-signs-at-roots:*

fixes *p:: real poly*

fixes *qs :: real poly list*

assumes $p \neq 0$

assumes $\bigwedge q. q \in set\ qs \implies coprime\ p\ q$

shows $set(find-consistent-signs-at-roots\ p\ qs) = set(characterize-consistent-signs-at-roots\ p\ qs)$

<proof>

theorem *find-consistent-signs-at-roots-alt:*

assumes $p \neq 0$

assumes $\bigwedge q. q \in set\ qs \implies coprime\ p\ q$

shows $set(find-consistent-signs-at-roots\ p\ qs) = consistent-signs-at-roots\ p\ qs$

<proof>

end

theory *BKR-Decision*

imports *BKR-Algorithm*

Berlekamp-Zassenhaus.Factorize-Rat-Poly

Algebraic-Numbers.Real-Roots

BKR-Proofs

HOL.Deriv

begin

16 Algorithm

16.1 Parsing

datatype *'a fml =*

```

And 'a fml 'a fml
| Or 'a fml 'a fml
| Gt 'a
| Geq 'a
| Lt 'a
| Leq 'a
| Eq 'a
| Neq 'a

```

primrec *lookup-sem* :: *nat fml* \Rightarrow (*'a::linordered-field list*) \Rightarrow *bool*
where

```

lookup-sem (And l r) ls = (lookup-sem l ls  $\wedge$  lookup-sem r ls)
| lookup-sem (Or l r) ls = (lookup-sem l ls  $\vee$  lookup-sem r ls)
| lookup-sem (Gt p) ls = (ls ! p > 0)
| lookup-sem (Geq p) ls = (ls ! p  $\geq$  0)
| lookup-sem (Lt p) ls = (ls ! p < 0)
| lookup-sem (Leq p) ls = (ls ! p  $\leq$  0)
| lookup-sem (Eq p) ls = (ls ! p = 0)
| lookup-sem (Neq p) ls = (ls ! p  $\neq$  0)

```

primrec *poly-list* :: *'a fml* \Rightarrow *'a list*

where

```

poly-list (And l r) = poly-list l @ poly-list r
| poly-list (Or l r) = poly-list l @ poly-list r
| poly-list (Gt p) = [p]
| poly-list (Geq p) = [p]
| poly-list (Lt p) = [p]
| poly-list (Leq p) = [p]
| poly-list (Eq p) = [p]
| poly-list (Neq p) = [p]

```

primrec *index-of-aux* :: *'a list* \Rightarrow *'a* \Rightarrow *nat* \Rightarrow *nat* **where**

```

index-of-aux [] y n = n
| index-of-aux (x#xs) y n =
  (if x = y then n else index-of-aux xs y (n+1))

```

definition *index-of* :: *'a list* \Rightarrow *'a* \Rightarrow *nat* **where**

```

index-of xs y = index-of-aux xs y 0

```

definition *convert* :: *'a fml* \Rightarrow (*nat fml* \times *'a list*)

where

```

convert fml = (
  let ps = remdups (poly-list fml)
  in
  (map-fml (index-of ps) fml, ps)
)

```


16.2 Factoring

definition *factorize-rat-poly-monic* :: *rat poly* \Rightarrow (*rat* \times (*rat poly* \times *nat*) *list*)

where

```

factorize-rat-poly-monic p = (
  let (c,fs) = factorize-rat-poly p ;
      lcs = prod-list (map (\(f,i). (lead-coeff f) ^ Suc i) fs) ;
      fs = map (\(f,i). (normalize f, i)) fs
  in
  (c * lcs,fs)
)

```

definition *factorize-polys* :: *rat poly list* \Rightarrow (*rat poly list* \times (*rat* \times (*nat* \times *nat*) *list*) *list*)

where

```

factorize-polys ps = (
  let fact-ps = map factorize-rat-poly-monic ps;
      factors = remdups (map fst (concat (map snd fact-ps))) ;
      data = map (\(c,fs). (c, map (\(f,pow). (index-of factors f, pow) ) fs))
  in
  (factors,data)
)

```

definition *undo-factorize* :: (*rat* \times (*nat* \times *nat*) *list*) \Rightarrow *rat list* \Rightarrow *rat*

where

```

undo-factorize cfs signs =
  squash
  (case cfs of (c,fs)  $\Rightarrow$ 
   (c * prod-list (map (\(f,pow). (signs ! f) ^ Suc pow) fs)))

```

definition *undo-factorize-polys* :: (*rat* \times (*nat* \times *nat*) *list*) *list* \Rightarrow *rat list* \Rightarrow *rat list*

where

```

undo-factorize-polys ls signs = map (\l. undo-factorize l signs) ls

```

16.3 Auxiliary Polynomial

definition *crb*:: *real poly* \Rightarrow *int* **where**

```

crb p = ceiling (2 + max-list-non-empty (map (\ i. norm (coeff p i)) [0 ..< degree
p]))
/ norm (lead-coeff p)

```

definition *coprime-r* :: *real poly list* \Rightarrow *real poly*

where

```

coprime-r ps = pderiv (prod-list ps) * ([:-(crb (prod-list ps)),1:]) * ([:(crb
(prod-list ps)),1:])

```

16.4 Setting Up the Procedure

definition *insertAt* :: *nat* \Rightarrow '*a* \Rightarrow '*a* list \Rightarrow '*a* list **where**
insertAt *n x ls* = *take n ls @ x # (drop n ls)*

definition *removeAt* :: *nat* \Rightarrow '*a* list \Rightarrow '*a* list **where**
removeAt *n ls* = *take n ls @ (drop (n+1) ls)*

definition *find-sgas-aux*:: *real poly list* \Rightarrow *rat list list*
where *find-sgas-aux in-list* =
*concat (map (λi .
map (λv . *insertAt* *i 0 v*) (*find-consistent-signs-at-roots* (*in-list* ! *i*) (*removeAt* *i*
in-list))*
) [*0..<length in-list*])

definition *find-sgas* :: *real poly list* \Rightarrow *rat list list*
where
find-sgas ps = (
let *r = coprime-r ps* in
find-consistent-signs-at-roots r ps @ find-sgas-aux ps
)

definition *find-consistent-signs* :: *rat poly list* \Rightarrow *rat list list*
where
find-consistent-signs ps = (
let (*fs,data*) = *factorize-polys ps*;
sgas = *find-sgas (map (map-poly of-rat) fs)*;
rsgas = *map (undo-factorize-polys data) sgas*
in
(*if fs* = [] *then [(map (λx . if poly *x 0* < 0 *then* -1 *else if* poly *x 0* = 0 *then* 0*
else 1) *ps)] else rsgas*)
)

16.5 Deciding Univariate Problems

definition *decide-universal* :: *rat poly fml* \Rightarrow *bool*
where [*code*]:
decide-universal fml = (
let (*fml-struct,polys*) = *convert fml*;
conds = *find-consistent-signs polys*
in
list-all (lookup-sem fml-struct) conds
)

definition *decide-existential* :: *rat poly fml* \Rightarrow *bool*
where [*code*]:
decide-existential fml = (

```

let (fml-struct, polys) = convert fml;
    conds = find-consistent-signs polys
in
  find (lookup-sem fml-struct) conds ≠ None
)

```

17 Proofs

17.1 Parsing and Semantics

primrec *real-sem* :: *real poly fml* ⇒ *real* ⇒ *bool*

where

```

  real-sem (And l r) x = (real-sem l x ∧ real-sem r x)
| real-sem (Or l r) x = (real-sem l x ∨ real-sem r x)
| real-sem (Gt p) x = (poly p x > 0)
| real-sem (Geq p) x = (poly p x ≥ 0)
| real-sem (Lt p) x = (poly p x < 0)
| real-sem (Leq p) x = (poly p x ≤ 0)
| real-sem (Eq p) x = (poly p x = 0)
| real-sem (Neq p) x = (poly p x ≠ 0)

```

primrec *fml-sem* :: *rat poly fml* ⇒ *real* ⇒ *bool*

where

```

  fml-sem (And l r) x = (fml-sem l x ∧ fml-sem r x)
| fml-sem (Or l r) x = (fml-sem l x ∨ fml-sem r x)
| fml-sem (Gt p) x = (rpoly p x > 0)
| fml-sem (Geq p) x = (rpoly p x ≥ 0)
| fml-sem (Lt p) x = (rpoly p x < 0)
| fml-sem (Leq p) x = (rpoly p x ≤ 0)
| fml-sem (Eq p) x = (rpoly p x = 0)
| fml-sem (Neq p) x = (rpoly p x ≠ 0)

```

lemma *poly-list-set-fml*:

shows *set (poly-list fml) = set-fml fml*

⟨*proof*⟩

lemma *convert-semantics-lem*:

assumes $\bigwedge p. p \in \text{set } (\text{poly-list } fml) \implies$

ls ! (*index-of ps p*) = *rpoly p x*

shows *fml-sem fml x = lookup-sem (map-fml (index-of ps) fml) ls*

⟨*proof*⟩

lemma *index-of-aux-more*:

shows *index-of-aux ls p n* ≥ *n*

⟨*proof*⟩

lemma *index-of-aux-lookup*:

assumes *p* ∈ *set ls*

shows $(\text{index-of-aux } ls \ p \ n) - n < \text{length } ls$
 $ls ! ((\text{index-of-aux } ls \ p \ n) - n) = p$
 $\langle \text{proof} \rangle$

lemma *index-of-lookup*:
assumes $p \in \text{set } ls$
shows $\text{index-of } ls \ p < \text{length } ls$
 $ls ! (\text{index-of } ls \ p) = p$
 $\langle \text{proof} \rangle$

lemma *convert-semantics*:
shows $\text{fml-sem } fml \ x = \text{lookup-sem } (\text{fst } (\text{convert } fml)) (\text{map } (\lambda p. \text{rpoly } p \ x) (\text{snd } (\text{convert } fml)))$
 $\langle \text{proof} \rangle$

lemma *convert-closed*:
shows $\bigwedge i. i \in \text{set-fml } (\text{fst } (\text{convert } fml)) \implies i < \text{length } (\text{snd } (\text{convert } fml))$
 $\langle \text{proof} \rangle$

definition *sign-vec::rat poly list \Rightarrow real \Rightarrow rat list*
where $\text{sign-vec } qs \ x \equiv$
 $\text{map } (\text{squash } \circ (\lambda p. \text{rpoly } p \ x)) \ qs$

definition *consistent-sign-vectors::rat poly list \Rightarrow real set \Rightarrow rat list set*
where $\text{consistent-sign-vectors } qs \ S = (\text{sign-vec } qs) \ ' S$

lemma *sign-vec-semantics*:
assumes $\bigwedge i. i \in \text{set-fml } fml \implies i < \text{length } ls$
shows $\text{lookup-sem } fml (\text{map } (\lambda p. \text{rpoly } p \ x) \ ls) = \text{lookup-sem } fml (\text{sign-vec } ls \ x)$
 $\langle \text{proof} \rangle$

lemma *universal-lookup-sem*:
assumes $\bigwedge i. i \in \text{set-fml } fml \implies i < \text{length } qs$
assumes $\text{set } \text{signs} = \text{consistent-sign-vectors } qs \ UNIV$
shows $(\forall x::\text{real}. \text{lookup-sem } fml (\text{map } (\lambda p. \text{rpoly } p \ x) \ qs)) \longleftrightarrow$
 $\text{list-all } (\text{lookup-sem } fml) \ \text{signs}$
 $\langle \text{proof} \rangle$

lemma *existential-lookup-sem*:
assumes $\bigwedge i. i \in \text{set-fml } fml \implies i < \text{length } qs$
assumes $\text{set } \text{signs} = \text{consistent-sign-vectors } qs \ UNIV$
shows $(\exists x::\text{real}. \text{lookup-sem } fml (\text{map } (\lambda p. \text{rpoly } p \ x) \ qs)) \longleftrightarrow$
 $\text{find } (\text{lookup-sem } fml) \ \text{signs} \neq \text{None}$
 $\langle \text{proof} \rangle$

17.2 Factoring Lemmas

interpretation *of-rat-poly-hom*: *map-poly-comm-semiring-hom of-rat* \langle *proof* \rangle

interpretation *of-rat-poly-hom*: *map-poly-comm-ring-hom of-rat* \langle *proof* \rangle

interpretation *of-rat-poly-hom*: *map-poly-idom-hom of-rat* \langle *proof* \rangle

lemma *finite-prod-map-of-rat-poly-hom*:

shows *poly (real-of-rat-poly ($\prod (a,b)\in s. f a b$)) y = ($\prod (a,b)\in s. poly (real-of-rat-poly (f a b)) y$)*
 \langle *proof* \rangle

lemma *sign-vec-index-of*:

assumes *f \in set ftrs*

shows *sign-vec ftrs x ! (index-of ftrs f) = squash (rpoly f x)*

\langle *proof* \rangle

lemma *squash-idem*:

shows *squash (squash x) = squash x*

\langle *proof* \rangle

lemma *squash-mult*:

shows *squash ((a::real) * b) = squash a * squash b*

\langle *proof* \rangle

lemma *squash-prod-list*:

shows *squash (prod-list (ls::real list)) = prod-list (map squash ls)*

\langle *proof* \rangle

lemma *squash-pow*:

shows *squash ((x::real) ^ (y::nat)) = (squash x) ^ y*

\langle *proof* \rangle

lemma *squash-real-of-rat[simp]*:

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

\langle *proof* \rangle

lemma *factorize-rat-poly-monic-irreducible-monic*:

assumes *factorize-rat-poly-monic f = (c,fs)*

assumes *(fi,i) \in set fs*

shows *irreducible fi \wedge monic fi*

\langle *proof* \rangle

lemma *square-free-normalize*:

assumes *square-free p*

shows *square-free (normalize p)*

\langle *proof* \rangle

lemma *coprime-normalize*:

assumes *coprime a b*

shows *coprime (normalize a) b*

<proof>

lemma *undo-normalize:*

shows $a = \text{Polynomial.smult } (\text{unit-factor } (\text{lead-coeff } a)) (\text{normalize } a)$
<proof>

lemma *finite-smult-distr:*

assumes *distinct fs*
shows $(\prod_{(x,y) \in \text{set } fs} \text{Polynomial.smult } ((f \ x \ y)::\text{rat}) (g \ x \ y)) =$
 $\text{Polynomial.smult } (\prod_{(x,y) \in \text{set } fs} f \ x \ y) (\prod_{(x,y) \in \text{set } fs} g \ x \ y)$
<proof>

lemma *normalize-coprime-degree:*

assumes $\text{normalize } (f::\text{rat poly}) = \text{normalize } g$
assumes *coprime f g*
shows $\text{degree } f = 0$
<proof>

lemma *factorize-rat-poly-monic-square-free-factorization:*

assumes *res: factorize-rat-poly-monic f = (c,fs)*
shows *square-free-factorization f (c,fs)*
<proof>

lemma *undo-factorize-correct:*

assumes $\text{factorize-rat-poly-monic } p = (c,fs)$
assumes $\bigwedge f \ p. (f,p) \in \text{set } fs \implies f \in \text{set } \text{ftrs}$
shows $\text{undo-factorize } (c, \text{map } (\lambda(f,pow). (\text{index-of } \text{ftrs } f, \text{pow})) fs) (\text{sign-vec } \text{ftrs } x) = \text{squash } (\text{rpolys } p \ x)$
<proof>

lemma *length-sign-vec[simp]:*

shows $\text{length } (\text{sign-vec } ps \ x) = \text{length } ps$ *<proof>*

lemma *factorize-polys-has-factors:*

assumes $\text{factorize-polys } ps = (\text{ftrs}, \text{data})$
assumes $p \in \text{set } ps$
assumes $\text{factorize-rat-poly-monic } p = (c,fs)$
shows $\text{set } (\text{map } \text{fst } fs) \subseteq \text{set } \text{ftrs}$
<proof>

lemma *factorize-polys-undo-factorize-polys:*

assumes $\text{factorize-polys } ps = (\text{ftrs}, \text{data})$
shows $\text{undo-factorize-polys } \text{data } (\text{sign-vec } \text{ftrs } x) = \text{sign-vec } ps \ x$
<proof>

lemma *factorize-polys-irreducible-monic:*

assumes $\text{factorize-polys } ps = (fs, \text{data})$
shows $\text{distinct } fs \ \bigwedge f. f \in \text{set } fs \implies \text{irreducible } f \ \wedge \ \text{monic } f$
<proof>

lemma *factorize-polys-square-free*:

assumes *factorize-polys ps = (fs,data)*

shows $\bigwedge f. f \in \text{set } fs \implies \text{square-free } f$

<proof>

lemma *irreducible-monic-coprime*:

assumes *f: monic f irreducible (f::rat poly)*

assumes *g: monic g irreducible (g::rat poly)*

assumes $f \neq g$

shows *coprime f g*

<proof>

lemma *factorize-polys-coprime*:

assumes *factorize-polys ps = (fs,data)*

shows $\bigwedge f g. f \in \text{set } fs \implies g \in \text{set } fs \implies f \neq g \implies \text{coprime } f g$

<proof>

lemma *coprime-rat-poly-real-poly*:

assumes *coprime p (q::rat poly)*

shows *coprime (real-of-rat-poly p) ((real-of-rat-poly q)::real poly)*

<proof>

lemma *coprime-rat-poly-iff-coprimerreal-poly*:

shows *coprime p (q::rat poly) \longleftrightarrow coprime (real-of-rat-poly p) ((real-of-rat-poly q)::real poly)*

<proof>

lemma *factorize-polys-map-distinct*:

assumes *factorize-polys ps = (fs,data)*

assumes *fss = map real-of-rat-poly fs*

shows *distinct fss*

<proof>

lemma *factorize-polys-map-square-free*:

assumes *factorize-polys ps = (fs,data)*

assumes *fss = map real-of-rat-poly fs*

shows $\bigwedge f. f \in \text{set } fss \implies \text{square-free } f$

<proof>

lemma *factorize-polys-map-coprime*:

assumes *factorize-polys ps = (fs,data)*

assumes *fss = map real-of-rat-poly fs*

shows $\bigwedge f g. f \in \text{set } fss \implies g \in \text{set } fss \implies f \neq g \implies \text{coprime } f g$

<proof>

lemma *coprime-prod-list*:

assumes $\bigwedge p. p \in \text{set } ps \implies p \neq 0$

assumes *coprime (prod-list ps) (q::real poly)*

shows $\bigwedge p. p \in \text{set } ps \implies \text{coprime } p \ q$
 $\langle \text{proof} \rangle$

lemma *factorize-polys-square-free-prod-list*:
assumes *factorize-polys* $ps = (fs, data)$
shows *square-free* (*prod-list* fs)
 $\langle \text{proof} \rangle$

lemma *factorize-polys-map-square-free-prod-list*:
assumes *factorize-polys* $ps = (fs, data)$
assumes $fss = \text{map } \text{real-of-rat-poly } fs$
shows *square-free* (*prod-list* fss)
 $\langle \text{proof} \rangle$

lemma *factorize-polys-map-coprime-pderiv*:
assumes *factorize-polys* $ps = (fs, data)$
assumes $fss = \text{map } \text{real-of-rat-poly } fs$
shows $\bigwedge f. f \in \text{set } fss \implies \text{coprime } f \ (\text{pderiv } (\text{prod-list } fss))$
 $\langle \text{proof} \rangle$

definition *pairwise-coprime-list*:: *rat poly list* \Rightarrow *bool*
where *pairwise-coprime-list* $qs =$
 $(\forall m < \text{length } qs. \forall n < \text{length } qs.$
 $m \neq n \longrightarrow \text{coprime } (qs ! n) (qs ! m))$

lemma *coprime-factorize*:
fixes $qs:: \text{rat poly list}$
shows *pairwise-coprime-list* (*fst*(*factorize-polys* qs))
 $\langle \text{proof} \rangle$

lemma *squarefree-factorization-degree*:
assumes *square-free-factorization* $p (c, fs)$
shows $\text{degree } p = \text{sum-list } (\text{map } (\lambda(f, c). (c+1) * \text{degree } f) fs)$
 $\langle \text{proof} \rangle$

17.3 Auxiliary Polynomial Lemmas

definition *roots-of-coprime-r*:: *real poly list* \Rightarrow *real set*
where *roots-of-coprime-r* $qs = \{x. \text{poly } (\text{coprime-r } qs) x = 0\}$

lemma *crb-lem-pos*:
fixes $x:: \text{real}$
fixes $p:: \text{real poly}$
assumes $x. \text{poly } p x = 0$
assumes $p: p \neq 0$
shows $x < \text{crb } p$
 $\langle \text{proof} \rangle$

lemma *crb-lem-neg*:

fixes $x:: \text{real}$
fixes $p:: \text{real poly}$
assumes $x: \text{poly } p \ x = 0$
assumes $p: p \neq 0$
shows $x > -\text{crb } p$
<proof>

lemma *prod-zero*:

shows $\forall x . \text{poly } (\text{prod-list } (qs:: \text{rat poly list})) \ x = 0 \longleftrightarrow (\exists q \in \text{set } (qs). \text{poly } q \ x = 0)$
<proof>

lemma *coprime-r-zero1*: $\text{poly } (\text{coprime-r } qs) (\text{crb } (\text{prod-list } qs)) = 0$

<proof>

lemma *coprime-r-zero2*: $\text{poly } (\text{coprime-r } qs) (-\text{crb } (\text{prod-list } qs)) = 0$

<proof>

lemma *coprime-mult*:

fixes $a:: \text{real poly}$
fixes $b:: \text{real poly}$
fixes $c:: \text{real poly}$
assumes *algebraic-semidom-class.coprime* $a \ b$
assumes *algebraic-semidom-class.coprime* $a \ c$
shows *algebraic-semidom-class.coprime* $a \ (b*c)$
<proof>

lemma *coprime-r-coprime-prop*:

fixes $ps:: \text{rat poly list}$
assumes *factorize-polys* $ps = (fs, data)$
assumes $fss = \text{map } \text{real-of-rat-poly } fs$
shows $\bigwedge f. f \in \text{set } fss \implies \text{coprime } f \ (\text{coprime-r } fss)$
<proof>

lemma *coprime-r-nonzero*:

fixes $ps:: \text{rat poly list}$
assumes *factorize-polys* $ps = (fs, data)$
assumes *nonempty-fs*: $fs \neq []$
assumes *fss-is*: $fss = \text{map } \text{real-of-rat-poly } fs$
shows $(\text{coprime-r } fss) \neq 0$
<proof>

lemma *Rolle-pderiv*:

fixes $q:: \text{real poly}$
fixes $x1 \ x2:: \text{real}$

shows $(x1 < x2 \wedge \text{poly } q \ x1 = 0 \wedge \text{poly } q \ x2 = 0) \longrightarrow (\exists w. x1 < w \wedge w < x2 \wedge \text{poly } (\text{pderiv } q) \ w = 0)$
 ⟨proof⟩

lemma *coprime-r-roots-prop*:

fixes $qs:: \text{real poly list}$

assumes *pairwise-rel-prime*: $\forall q1 \ q2. (q1 \neq q2 \wedge (\text{List.member } qs \ q1) \wedge (\text{List.member } qs \ q2)) \longrightarrow \text{coprime } q1 \ q2$

shows $\forall x1. \forall x2. ((x1 < x2 \wedge (\exists q1 \in \text{set } (qs). (\text{poly } q1 \ x1) = 0) \wedge (\exists q2 \in \text{set } (qs). (\text{poly } q2 \ x2) = 0)) \longrightarrow (\exists q. x1 < q \wedge q < x2 \wedge \text{poly } (\text{coprime-r } qs) \ q = 0))$
 ⟨proof⟩

17.4 Setting Up the Procedure: Lemmas

definition *has-no-zeros*:: $\text{rat list} \Rightarrow \text{bool}$

where *has-no-zeros* $l = (0 \notin \text{set } l)$

lemma *hnz-prop*: $\text{has-no-zeros } l \longleftrightarrow \neg(\exists k < \text{length } l. l ! k = 0)$

⟨proof⟩

definition *cast-rat-list*:: $\text{rat poly list} \Rightarrow \text{real poly list}$

where *cast-rat-list* $qs = \text{map } \text{real-of-rat-poly } qs$

definition *consistent-sign-vectors-r*:: $\text{real poly list} \Rightarrow \text{real set} \Rightarrow \text{rat list set}$

where *consistent-sign-vectors-r* $qs \ S = (\text{signs-at } qs) \ ' S$

lemma *consistent-sign-vectors-consistent-sign-vectors-r*:

shows *consistent-sign-vectors-r* $(\text{cast-rat-list } qs) \ S = \text{consistent-sign-vectors } qs \ S$
 ⟨proof⟩

lemma *coprime-over-reals-coprime-over-rats*:

fixes $qs:: \text{rat poly list}$

assumes *csa-in*: $csa \in (\text{consistent-sign-vectors } qs \ \text{UNIV})$

assumes *p1p2*: $p1 \neq p2 \wedge p1 < \text{length } csa \wedge p2 < \text{length } csa \wedge csa ! p1 = 0 \wedge csa ! p2 = 0$

shows $\neg \text{algebraic-semidom-class.coprime } (\text{nth } qs \ p1) \ (\text{nth } qs \ p2)$
 ⟨proof⟩

lemma *zero-above*:

fixes $qs:: \text{rat poly list}$

fixes $x1:: \text{real}$

assumes *hnz*: $\text{has-no-zeros } (\text{sign-vec } qs \ x1)$

shows $(\forall x2 > x1. ((\text{sign-vec } qs \ x1) \neq (\text{sign-vec } qs \ x2)) \longrightarrow (\exists (r::\text{real}) > x1. (r \leq x2 \wedge (\exists q \in \text{set } (qs). \text{rpoly } q \ r = 0))))$
 ⟨proof⟩

lemma *zero-below*:

fixes $qs:: \text{rat poly list}$
fixes $x1:: \text{real}$
assumes $hnz: \text{has-no-zeros } (\text{sign-vec } qs \ x1)$
shows $\forall x2 < x1. ((\text{sign-vec } qs \ x1) \neq (\text{sign-vec } qs \ x2)) \longrightarrow$
 $(\exists (r::\text{real}) < x1. (r \geq x2 \wedge (\exists q \in \text{set}(qs). \text{rpoly } q \ r = 0)))$
 $\langle \text{proof} \rangle$

lemma *sorted-list-lemma*:
fixes $l:: \text{real list}$
fixes $a \ b:: \text{real}$
fixes $n:: \text{nat}$
assumes $a < b$
assumes $(n + 1) < \text{length } l$
assumes $\text{strict-sort: sorted-wrt } (<) \ l$
assumes $lt\text{-}a: (l \ ! \ n) < a$
assumes $b\text{-}lt: b < (l \ ! \ (n+1))$
shows $\neg(\exists (x::\text{real}). (\text{List.member } l \ x \wedge a \leq x \wedge x \leq b))$
 $\langle \text{proof} \rangle$

lemma *roots-of-coprime-r-location-property*:
fixes $qs:: \text{rat poly list}$
fixes $sga:: \text{rat list}$
fixes zer-list
assumes $\text{pairwise-rel-prime: pairwise-coprime-list } qs$
assumes $\text{all-squarefree: } \bigwedge q. q \in \text{set } qs \implies \text{rsquarefree } q$
assumes $x1\text{-prop: } sga = \text{sign-vec } qs \ x1$
assumes $hnz: \text{has-no-zeros } sga$
assumes $\text{zer-list-prop: } \text{zer-list} = \text{sorted-list-of-set } \{(x::\text{real}). (\exists q \in \text{set}(qs). (\text{rpoly } q \ x = 0))\}$
shows $\text{zer-list} \neq [] \longrightarrow ((x1 < (\text{zer-list} \ ! \ 0)) \vee (x1 > (\text{zer-list} \ ! \ (\text{length } \text{zer-list} - 1))) \vee$
 $(\exists n < (\text{length } \text{zer-list} - 1). x1 > (\text{zer-list} \ ! \ n) \wedge x1 < (\text{zer-list} \ ! \ (n+1))))$
 $\langle \text{proof} \rangle$

lemma *roots-of-coprime-r-capture-sgas-without-zeros*:
fixes $qs:: \text{rat poly list}$
fixes $sga:: \text{rat list}$
assumes $\text{pairwise-rel-prime: pairwise-coprime-list } qs$
assumes $\text{all-squarefree: } \bigwedge q. q \in \text{set } qs \implies \text{rsquarefree } q$
assumes $ex\text{-}x1: sga = \text{sign-vec } qs \ x1$
assumes $hnz: \text{has-no-zeros } sga$
shows $(\exists w \in (\text{roots-of-coprime-r } (\text{cast-rat-list } qs)). sga = (\text{sign-vec } qs \ w))$
 $\langle \text{proof} \rangle$

lemma *find-csas-lemma-nozeros*:
fixes $qs:: \text{rat poly list}$

assumes fs : *factorize-polys* $qs = (fs, data)$
assumes $fs \neq []$
shows $(csa \in (\text{consistent-sign-vectors } fs \text{ UNIV}) \wedge \text{has-no-zeros } csa) \longleftrightarrow$
 $csa \in \text{set } (\text{find-consistent-signs-at-roots } (\text{coprime-r } (\text{cast-rat-list } fs)) (\text{cast-rat-list } fs))$
<proof>

lemma *range-eq*:
fixes a
fixes b
shows $a = b \longrightarrow \text{range } a = \text{range } b$
<proof>

lemma *removeAt-distinct*:
shows $\text{distinct } fss \implies \text{distinct } (\text{removeAt } i \ fss)$
<proof>

lemma *consistent-signs-atw*:
assumes $\bigwedge p. p \in \text{set } fs \implies \text{poly } p \ x \neq 0$
shows $\text{consistent-sign-vec-copr } fs \ x = \text{signs-at } fs \ x$
<proof>

lemma *characterize-consistent-signs-at-roots-rw*:
assumes $p \neq 0$
assumes *copr*: $\bigwedge q. q \in \text{set } fs \implies \text{coprime } p \ q$
shows $\text{set } (\text{characterize-consistent-signs-at-roots } p \ fs) =$
 $\text{consistent-sign-vectors-r } fs \ (\{x. \text{poly } p \ x = 0\})$
<proof>

lemma *signs-at-insert*:
assumes $i \leq \text{length } qs$
shows $\text{insertAt } i \ (\text{squash } (\text{poly } p \ x)) \ (\text{signs-at } qs \ x) =$
 $\text{signs-at } (\text{insertAt } i \ p \ qs) \ x$
<proof>

lemma *length-removeAt*:
assumes $i < \text{length } ls$
shows $\text{length } (\text{removeAt } i \ ls) = \text{length } ls - 1$
<proof>

lemma *insertAt-removeAt*:
assumes $i < \text{length } ls$
shows $\text{insertAt } i \ (ls!i) \ (\text{removeAt } i \ ls) = ls$
<proof>

lemma *insertAt-nth*:
assumes $i \leq \text{length } ls$
shows $\text{insertAt } i \ n \ ls ! i = n$

<proof>

lemma *length-signs-at[simp]*:
shows $\text{length} (\text{signs-at } qs \ x) = \text{length } qs$
<proof>

lemma *find-csa-at-index*:
assumes $i < \text{length } fss$
assumes $\bigwedge p \ q. p \in \text{set } fss \implies q \in \text{set } fss \implies p \neq q \implies \text{coprime } p \ q$
assumes $\bigwedge p. p \in \text{set } fss \implies p \neq 0$
assumes *distinct fss*
shows
 $\text{set} (\text{map } (\lambda v. \text{insertAt } i \ 0 \ v) (\text{find-consistent-signs-at-roots } (fss \ ! \ i) (\text{removeAt } i \ fss))) =$
 $\{v \in \text{consistent-sign-vectors-r } fss \ UNIV. v \ ! \ i = 0\}$
<proof>

lemma *in-set-concat-map*:
assumes $i < \text{length } ls$
assumes $x \in \text{set} (f \ (ls \ ! \ i))$
shows $x \in \text{set} (\text{concat } (\text{map } f \ ls))$
<proof>

lemma *find-csas-lemma-onezero-gen*:
fixes $qs:: \text{rat poly list}$
assumes $fs: \text{factorize-polys } qs = (fs, data)$
assumes $fss: fss = \text{cast-rat-list } fs$
shows $(csa \in (\text{consistent-sign-vectors-r } fss \ UNIV) \wedge \neg(\text{has-no-zeros } csa))$
 $\longleftrightarrow csa \in \text{set} (\text{find-sgas-aux } fss)$
<proof>

lemma *mem-append*: $\text{List.member } (l1@l2) \ m \longleftrightarrow (\text{List.member } l1 \ m \vee \text{List.member } l2 \ m)$
<proof>

lemma *same-sign-cond-rationals-reals*:
fixes $qs:: \text{rat poly list}$
assumes $lenh: \text{length} (\text{fst}(\text{factorize-polys } qs)) > 0$
shows $\text{set}(\text{find-sgas } (\text{map} (\text{map-poly of-rat}) (\text{fst}(\text{factorize-polys } qs)))) = \text{consistent-sign-vectors } (\text{fst}(\text{factorize-polys } qs)) \ UNIV$
<proof>

lemma *factorize-polys-undo-factorize-polys-set*:
assumes $\text{factorize-polys } ps = (ftrs, data)$
assumes $sgas = \text{find-sgas } (\text{map} (\text{map-poly of-rat}) \ ftrs)$
assumes $sgas\text{-set}: \text{set} (sgas) = \text{consistent-sign-vectors } ftrs \ UNIV$
shows $\text{set} (\text{map} (\text{undo-factorize-polys } data) \ sgas) = \text{consistent-sign-vectors } ps \ UNIV$
<proof>

lemma *main-step-aux1*:
fixes *qs*:: *rat poly list*
assumes *empty*: $(fst(factorize-polys\ qs)) = []$
shows $set\ (find-consistent-signs\ qs) = consistent-sign-vectors\ qs\ UNIV$
 $\langle proof \rangle$

lemma *main-step-aux2*:
fixes *qs*:: *rat poly list*
assumes *lenh*: $length\ (fst(factorize-polys\ qs)) > 0$
shows $set\ (find-consistent-signs\ qs) = consistent-sign-vectors\ qs\ UNIV$
 $\langle proof \rangle$

lemma *main-step*:
fixes *qs*:: *rat poly list*
shows $set\ (find-consistent-signs\ qs) = consistent-sign-vectors\ qs\ UNIV$
 $\langle proof \rangle$

17.5 Decision Procedure: Main Lemmas

lemma *decide-univ-lem-helper*:
assumes $(fml-struct, polys) = convert\ fml$
shows $(\forall x::real. lookup-sem\ fml-struct\ (map\ (\lambda p. rpoly\ p\ x)\ polys)) \longleftrightarrow$
 $(decide-universal\ fml)$
 $\langle proof \rangle$

lemma *decide-exis-lem-helper*:
assumes $(fml-struct, polys) = convert\ fml$
shows $(\exists x::real. lookup-sem\ fml-struct\ (map\ (\lambda p. rpoly\ p\ x)\ polys)) \longleftrightarrow$
 $(decide-existential\ fml)$
 $\langle proof \rangle$

theorem *decision-procedure*:
shows $(\forall x::real. fml-sem\ fml\ x) \longleftrightarrow (decide-universal\ fml)$
 $\exists x::real. fml-sem\ fml\ x \longleftrightarrow (decide-existential\ fml)$
 $\langle proof \rangle$

end
theory *Renegar-Algorithm*
imports *BKR-Algorithm*
begin

definition *construct-NoFI-R*:: *real poly* \Rightarrow *real poly list* \Rightarrow *real poly list* \Rightarrow *rat*
where $construct-NoFI-R\ p\ I1\ I2 = ($
 $let\ new-p = sum-list\ (map\ (\lambda x. x^{\wedge}2)\ (p\ \# I1))\ in$
 $rat-of-int\ (changes-R-smods\ new-p\ ((pderiv\ new-p)*(prod-list\ I2))))$

definition *construct-rhs-vector-R*:: $real\ poly \Rightarrow real\ poly\ list \Rightarrow (nat\ list * nat\ list)\ list \Rightarrow rat\ vec$
where *construct-rhs-vector-R* $p\ qs\ Is =$
 $vec\ of\ list\ (map\ (\lambda(I1,I2).$
 $(construct\ NoFI\ R\ p\ (retrieve\ polys\ qs\ I1)\ (retrieve\ polys\ qs\ I2)))\ Is)$

18 Base Case

definition *base-case-info-R*:: $(rat\ mat \times ((nat\ list * nat\ list)\ list \times rat\ list\ list))$
where *base-case-info-R* $=$
 $((mat\ of\ rows\ list\ 3\ [[1,1,1], [0,1,0], [1,0,-1]], (([]), ([0], []), ([], [0])), [[1],[0],[-1]]))$

definition *base-case-solve-for-lhs*:: $real\ poly \Rightarrow real\ poly \Rightarrow rat\ vec$
where *base-case-solve-for-lhs* $p\ q = (mult\ mat\ vec\ (mat\ of\ rows\ list\ 3\ [[1/2, -1/2,$
 $1/2], [0, 1, 0], [1/2, -1/2, -1/2]])\ (construct\ rhs\ vector\ R\ p\ [q]\ [([], []), ([0], []), ([], [0]), [0]]))$

definition *solve-for-lhs-R*:: $real\ poly \Rightarrow real\ poly\ list \Rightarrow (nat\ list * nat\ list)\ list \Rightarrow rat\ mat \Rightarrow rat\ vec$
where *solve-for-lhs-R* $p\ qs\ subsets\ matr =$
 $mult\ mat\ vec\ (matr\ option\ (dim\ row\ matr)\ (mat\ inverse\ var\ matr))\ (construct\ rhs\ vector\ R\ p\ qs\ subsets)$

19 Smashing

definition *subsets-smash-R*:: $nat \Rightarrow (nat\ list * nat\ list)\ list \Rightarrow (nat\ list * nat\ list)\ list \Rightarrow (nat\ list * nat\ list)\ list$
where *subsets-smash-R* $n\ s1\ s2 = concat\ (map\ (\lambda l1. map\ (\lambda l2. (((fst\ l1)\ @\ (map\ ((+)\ n)\ (fst\ l2))), (snd\ l1)\ @\ (map\ ((+)\ n)\ (snd\ l2))))\ s2)\ s1)$

definition *smash-systems-R*:: $('a::zero)\ poly \Rightarrow ('a::zero)\ poly\ list \Rightarrow ('a::zero)\ poly\ list \Rightarrow (nat\ list * nat\ list)\ list \Rightarrow (nat\ list * nat\ list)\ list \Rightarrow rat\ list\ list \Rightarrow rat\ list\ list \Rightarrow rat\ mat \Rightarrow rat\ mat \Rightarrow ('a::zero)\ poly\ list \times (rat\ mat \times ((nat\ list * nat\ list)\ list \times rat\ list\ list))$
where *smash-systems-R* $p\ qs1\ qs2\ subsets1\ subsets2\ signs1\ signs2\ mat1\ mat2 =$
 $(qs1@qs2, (kronecker\ product\ mat1\ mat2, (subsets\ smash\ R\ (length\ qs1)\ subsets1\ subsets2, signs\ smash\ signs1\ signs2)))$

fun *combine-systems-R*:: $('a::zero)\ poly \Rightarrow (('a::zero)\ poly\ list \times (rat\ mat \times ((nat\ list * nat\ list)\ list \times rat\ list\ list))) \Rightarrow (('a::zero)\ poly\ list \times (rat\ mat \times ((nat\ list * nat\ list)\ list \times rat\ list\ list)))$
 $\Rightarrow (('a::zero)\ poly\ list \times (rat\ mat \times ((nat\ list * nat\ list)\ list \times rat\ list\ list)))$
where *combine-systems-R* $p\ (qs1, m1, sub1, sgn1)\ (qs2, m2, sub2, sgn2) =$
 $(smash\ systems\ R\ p\ qs1\ qs2\ sub1\ sub2\ sgn1\ sgn2\ m1\ m2)$

20 Reduction

```

fun reduction-step-R:: rat mat  $\Rightarrow$  rat list list  $\Rightarrow$  (nat list*nat list) list  $\Rightarrow$  rat vec
 $\Rightarrow$  rat mat  $\times$  ((nat list*nat list) list  $\times$  rat list list)
  where reduction-step-R A signs subsets lhs-vec =
    (let reduce-cols-A = (reduce-mat-cols A lhs-vec);
      rows-keep = rows-to-keep reduce-cols-A in
    (take-rows-from-matrix reduce-cols-A rows-keep,
    (take-indices subsets rows-keep,
    take-indices signs (find-nonzeros-from-input-vec lhs-vec))))

fun reduce-system-R:: real poly  $\Rightarrow$  (real poly list  $\times$  (rat mat  $\times$  ((nat list*nat list)
list  $\times$  rat list list)))  $\Rightarrow$  (rat mat  $\times$  ((nat list*nat list) list  $\times$  rat list list))
  where reduce-system-R p (qs,m,subs,signs) =
    reduction-step-R m signs subs (solve-for-lhs-R p qs subs m)

```

21 Overall algorithm

```

fun calculate-data-R:: real poly  $\Rightarrow$  real poly list  $\Rightarrow$  (rat mat  $\times$  ((nat list*nat list)
list  $\times$  rat list list))
  where
    calculate-data-R p qs =
    ( let len = length qs in
      if len = 0 then
        ( $\lambda(a,b,c).(a,b, \text{map } (\text{drop } 1) c)$ ) (reduce-system-R p ([1],base-case-info-R))
      else if len  $\leq$  1 then reduce-system-R p (qs,base-case-info-R)
      else
        (let q1 = take (len div 2) qs; left = calculate-data-R p q1;
          q2 = drop (len div 2) qs; right = calculate-data-R p q2;
          comb = combine-systems-R p (q1,left) (q2,right) in
        reduce-system-R p comb
    )
  )

```

```

definition find-consistent-signs-at-roots-R:: real poly  $\Rightarrow$  real poly list  $\Rightarrow$  rat list list
  where [code]:
    find-consistent-signs-at-roots-R p qs =
    ( let (M,S, $\Sigma$ ) = calculate-data-R p qs in  $\Sigma$  )

```

```

lemma find-consistent-signs-at-roots-thm-R:
  shows find-consistent-signs-at-roots-R p qs = snd (snd (calculate-data-R p qs))
  <proof>

```

```

end
theory Renegar-Proofs
  imports Renegar-Algorithm
    BKR-Proofs

```


begin

22 Tarski Queries Changed

lemma *construct-NofI-R-relation:*

fixes $p:: \text{real poly}$

fixes $I1:: \text{real poly list}$

fixes $I2:: \text{real poly list}$

shows $\text{construct-NofI-R } p \ I1 \ I2 =$

$\text{construct-NofI } (\text{sum-list } (\text{map } \text{power2 } (p \ \# \ I1))) \ I2$

$\langle \text{proof} \rangle$

lemma *sum-list-map-power2:*

shows $\text{sum-list } (\text{map } \text{power2 } \text{ls}) \geq (0::\text{real poly})$

$\langle \text{proof} \rangle$

lemma *sum-list-map-power2-poly:*

shows $\text{poly } (\text{sum-list } (\text{map } \text{power2 } (\text{ls}::\text{real poly list}))) \ x \geq (0::\text{real})$

$\langle \text{proof} \rangle$

lemma *construct-NofI-R-prop-helper:*

fixes $p:: \text{real poly}$

fixes $I1:: \text{real poly list}$

fixes $I2:: \text{real poly list}$

assumes $\text{nonzero: } p \neq 0$

shows $\text{construct-NofI-R } p \ I1 \ I2 =$

$\text{rat-of-int } (\text{int } (\text{card } \{x. \text{poly } (\text{sum-list } (\text{map } (\lambda x. x^2) (p \ \# \ I1))) \ x = 0 \wedge \text{poly } (\text{prod-list } I2) \ x > 0\}) -$

$\text{int } (\text{card } \{x. \text{poly } (\text{sum-list } (\text{map } (\lambda x. x^2) (p \ \# \ I1))) \ x = 0 \wedge \text{poly } (\text{prod-list } I2) \ x < 0\}))$

$\langle \text{proof} \rangle$

lemma *zer-iff:*

fixes $p:: \text{real poly}$

fixes $\text{ls}:: \text{real poly list}$

shows $\text{poly } (\text{sum-list } (\text{map } (\lambda x. x^2) \text{ls})) \ x = 0 \longleftrightarrow (\forall i \in \text{set } \text{ls}. \text{poly } i \ x = 0)$

$\langle \text{proof} \rangle$

lemma *construct-NofI-prop-R:*

fixes $p:: \text{real poly}$

fixes $I1:: \text{real poly list}$

fixes $I2:: \text{real poly list}$

assumes $\text{nonzero: } p \neq 0$

shows $\text{construct-NofI-R } p \ I1 \ I2 =$

$\text{rat-of-int } (\text{int } (\text{card } \{x. \text{poly } p \ x = 0 \wedge (\forall q \in \text{set } I1. \text{poly } q \ x = 0) \wedge \text{poly } (\text{prod-list } I2) \ x > 0\}) -$

$\text{int } (\text{card } \{x. \text{poly } p \ x = 0 \wedge (\forall q \in \text{set } I1. \text{poly } q \ x = 0) \wedge \text{poly } (\text{prod-list } I2) \ x < 0\}))$

$\langle \text{proof} \rangle$

23 Matrix Equation

definition *map-sgas*:: $\text{rat list} \Rightarrow \text{rat list}$
where *map-sgas* $l = \text{map } (\lambda r. (1 - r^2)) l$

definition *z-R*:: $(\text{nat list} * \text{nat list}) \Rightarrow \text{rat list} \Rightarrow \text{rat}$
where *z-R* $\text{index-list sign-asg} \equiv (\text{prod-list } (\text{map } (\text{nth } (\text{map-sgas } \text{sign-asg})) (\text{fst}(\text{index-list})))) * (\text{prod-list } (\text{map } (\text{nth } \text{sign-asg}) (\text{snd}(\text{index-list}))))$

definition *mtx-row-R*:: $\text{rat list list} \Rightarrow (\text{nat list} * \text{nat list}) \Rightarrow \text{rat list}$
where *mtx-row-R* $\text{sign-list index-list} \equiv (\text{map } ((\text{z-R } \text{index-list}) \text{sign-list}))$

definition *matrix-A-R*:: $\text{rat list list} \Rightarrow (\text{nat list} * \text{nat list}) \text{ list} \Rightarrow \text{rat mat}$
where *matrix-A-R* $\text{sign-list subset-list} = (\text{mat-of-rows-list } (\text{length } \text{sign-list}) (\text{map } (\lambda i. (\text{mtx-row-R } \text{sign-list } i)) \text{subset-list}))$

definition *all-list-constr-R*:: $(\text{nat list} * \text{nat list}) \text{ list} \Rightarrow \text{nat} \Rightarrow \text{bool}$
where *all-list-constr-R* $L n \equiv (\forall x. \text{List.member } L x \longrightarrow (\text{list-constr } (\text{fst } x) n \wedge \text{list-constr } (\text{snd } x) n))$

definition *alt-matrix-A-R*:: $\text{rat list list} \Rightarrow (\text{nat list} * \text{nat list}) \text{ list} \Rightarrow \text{rat mat}$
where *alt-matrix-A-R* $\text{signs subsets} = (\text{mat } (\text{length } \text{subsets}) (\text{length } \text{signs}) (\lambda(i, j). \text{z-R } (\text{subsets } ! i) (\text{signs } ! j)))$

lemma *alt-matrix-char-R*: $\text{alt-matrix-A-R } \text{signs subsets} = \text{matrix-A-R } \text{signs subsets}$
<proof>

lemma *subsets-are-rows-R*: $\forall i < (\text{length } \text{subsets}). \text{row } (\text{alt-matrix-A-R } \text{signs subsets}) i = \text{vec } (\text{length } \text{signs}) (\lambda j. \text{z-R } (\text{subsets } ! i) (\text{signs } ! j))$
<proof>

lemma *signs-are-cols-R*: $\forall i < (\text{length } \text{signs}). \text{col } (\text{alt-matrix-A-R } \text{signs subsets}) i = \text{vec } (\text{length } \text{subsets}) (\lambda j. \text{z-R } (\text{subsets } ! j) (\text{signs } ! i))$
<proof>

definition *consistent-sign-vec*:: $\text{real poly list} \Rightarrow \text{real} \Rightarrow \text{rat list}$
where *consistent-sign-vec* $qs x \equiv \text{map } (\lambda q. \text{if } (\text{poly } q x > 0) \text{ then } (1::\text{rat}) \text{ else } (\text{if } (\text{poly } q x = 0) \text{ then } (0::\text{rat}) \text{ else } (-1::\text{rat}))) qs$

definition *construct-lhs-vector-R*:: $\text{real poly} \Rightarrow \text{real poly list} \Rightarrow \text{rat list list} \Rightarrow \text{rat vec}$
where *construct-lhs-vector-R* $p qs \text{ signs} \equiv \text{vec-of-list } (\text{map } (\lambda w. \text{rat-of-int } (\text{int } (\text{length } (\text{filter } (\lambda v. v = w) (\text{map } (\text{consistent-sign-vec } qs) (\text{characterize-root-list-p } p)))))) \text{signs})$

definition *satisfy-equation-R*:: $real\ poly \Rightarrow real\ poly\ list \Rightarrow (nat\ list * nat\ list)\ list \Rightarrow rat\ list\ list \Rightarrow bool$

where *satisfy-equation-R* $p\ qs\ subset-list\ sign-list =$
 $(mult-mat-vec\ (matrix-A-R\ sign-list\ subset-list)\ (construct-lhs-vector-R\ p\ qs\ sign-list)) = (construct-rhs-vector-R\ p\ qs\ subset-list)$

lemma *construct-lhs-vector-clean-R*:

assumes $p \neq 0$
assumes $i < length\ signs$
shows $(construct-lhs-vector-R\ p\ qs\ signs)\ \$\ i =$
 $card\ \{x.\ poly\ p\ x = 0 \wedge ((consistent-sign-vec\ qs\ x) = (nth\ signs\ i))\}$
 $\langle proof \rangle$

lemma *construct-lhs-vector-cleaner-R*:

assumes $p \neq 0$
shows $(construct-lhs-vector-R\ p\ qs\ signs) =$
 $vec-of-list\ (map\ (\lambda s.\ rat-of-int\ (card\ \{x.\ poly\ p\ x = 0 \wedge ((consistent-sign-vec\ qs\ x) = s)\}))\ signs)$
 $\langle proof \rangle$

lemma *z-signs-R2*:

fixes $I:: nat\ list$
fixes $signs:: rat\ list$
assumes $lf: list-all\ (\lambda i.\ i < length\ signs)\ I$
assumes $la: list-all\ (\lambda s.\ s = 1 \vee s = 0 \vee s = -1)\ signs$
shows $(prod-list\ (map\ (nth\ signs)\ I)) = 1 \vee$
 $(prod-list\ (map\ (nth\ signs)\ I)) = 0 \vee$
 $(prod-list\ (map\ (nth\ signs)\ I)) = -1 \langle proof \rangle$

lemma *z-signs-R1*:

fixes $I:: nat\ list$
fixes $signs:: rat\ list$
assumes $lf: list-all\ (\lambda i.\ i < length\ signs)\ I$
assumes $la: list-all\ (\lambda s.\ s = 1 \vee s = 0 \vee s = -1)\ signs$
shows $(prod-list\ (map\ (nth\ (map-sgas\ signs))\ I)) = 1 \vee$
 $(prod-list\ (map\ (nth\ (map-sgas\ signs))\ I)) = 0 \langle proof \rangle$

lemma *z-signs-R*:

fixes $I:: (nat\ list * nat\ list)$
fixes $signs:: rat\ list$
assumes $lf: list-all\ (\lambda i.\ i < length\ signs)\ (fst\ I)$
assumes $ls: list-all\ (\lambda i.\ i < length\ signs)\ (snd\ I)$
assumes $la: list-all\ (\lambda s.\ s = 1 \vee s = 0 \vee s = -1)\ signs$
shows $(z-R\ I\ signs = 1) \vee (z-R\ I\ signs = 0) \vee (z-R\ I\ signs = -1)$
 $\langle proof \rangle$

lemma *z-lemma-R*:

fixes $I:: \text{nat list} * \text{nat list}$
fixes $\text{sign}:: \text{rat list}$
assumes $\text{consistent}: \text{sign} \in \text{set} (\text{characterize-consistent-signs-at-roots } p \text{ } qs)$
assumes $\text{welldefined1}: \text{list-constr } (\text{fst } I) (\text{length } qs)$
assumes $\text{welldefined2}: \text{list-constr } (\text{snd } I) (\text{length } qs)$
shows $(z\text{-R } I \text{ sign} = 1) \vee (z\text{-R } I \text{ sign} = 0) \vee (z\text{-R } I \text{ sign} = -1)$
 $\langle \text{proof} \rangle$

lemma in-set-R :
fixes $p:: \text{real poly}$
assumes $\text{nonzero}: p \neq 0$
fixes $qs:: \text{real poly list}$
fixes $\text{sign}:: \text{rat list}$
fixes $x:: \text{real}$
assumes $\text{root-p}: x \in \{x. \text{poly } p \ x = 0\}$
assumes $\text{sign-fix}: \text{sign} = \text{consistent-sign-vec } qs \ x$
shows $\text{sign} \in \text{set} (\text{characterize-consistent-signs-at-roots } p \text{ } qs)$
 $\langle \text{proof} \rangle$

lemma $\text{consistent-signs-prop-R}$:
fixes $p:: \text{real poly}$
assumes $\text{nonzero}: p \neq 0$
fixes $qs:: \text{real poly list}$
fixes $\text{sign}:: \text{rat list}$
fixes $x:: \text{real}$
assumes $\text{root-p}: x \in \{x. \text{poly } p \ x = 0\}$
assumes $\text{sign-fix}: \text{sign} = \text{consistent-sign-vec } qs \ x$
shows $\text{list-all } (\lambda s. s = 1 \vee s = 0 \vee s = -1) \ \text{sign}$
 $\langle \text{proof} \rangle$

lemma $\text{horiz-vector-helper-pos-ind-R1}$:
fixes $p:: \text{real poly}$
assumes $\text{nonzero}: p \neq 0$
fixes $qs:: \text{real poly list}$
fixes $I:: \text{nat list}$
fixes $\text{sign}:: \text{rat list}$
fixes $x:: \text{real}$
assumes $\text{root-p}: x \in \{x. \text{poly } p \ x = 0\}$
assumes $\text{sign-fix}: \text{sign} = \text{consistent-sign-vec } qs \ x$
assumes $\text{asm}: \text{list-constr } I (\text{length } qs)$
shows $(\text{prod-list } (\text{map } (\text{nth } (\text{map-sgas } \text{sign})) \ I)) = 1 \iff$
 $(\forall p \in \text{set } (\text{retrieve-polys } qs \ I). \text{poly } p \ x = 0)$
 $\langle \text{proof} \rangle$

lemma $\text{csv-length-same-as-qlist}$:
fixes $p:: \text{real poly}$
assumes $\text{nonzero}: p \neq 0$

fixes qs :: *real poly list*
fixes $sign$:: *rat list*
fixes x :: *real*
assumes $root-p$: $x \in \{x. poly\ p\ x = 0\}$
assumes $sign-fix$: $sign = consistent-sign-vec\ qs\ x$
shows $length\ sign = length\ qs$
 $\langle proof \rangle$

lemma *horiz-vector-helper-zer-ind-R2*:

fixes p :: *real poly*
assumes $nonzero$: $p \neq 0$
fixes qs :: *real poly list*
fixes I :: *nat list*
fixes $sign$:: *rat list*
fixes x :: *real*
assumes $root-p$: $x \in \{x. poly\ p\ x = 0\}$
assumes $sign-fix$: $sign = consistent-sign-vec\ qs\ x$
assumes asm : $list-constr\ I\ (length\ qs)$
shows $(prod-list\ (map\ (nth\ sign)\ I)) = 0 \iff$
 $(poly\ (prod-list\ (retrieve-polys\ qs\ I))\ x = 0)$
 $\langle proof \rangle$

lemma *horiz-vector-helper-pos-ind-R2*:

fixes p :: *real poly*
assumes $nonzero$: $p \neq 0$
fixes qs :: *real poly list*
fixes I :: *nat list*
fixes $sign$:: *rat list*
fixes x :: *real*
assumes $root-p$: $x \in \{x. poly\ p\ x = 0\}$
assumes $sign-fix$: $sign = consistent-sign-vec\ qs\ x$
assumes asm : $list-constr\ I\ (length\ qs)$
shows $(prod-list\ (map\ (nth\ sign)\ I)) = 1 \iff$
 $(poly\ (prod-list\ (retrieve-polys\ qs\ I))\ x > 0)$
 $\langle proof \rangle$

lemma *horiz-vector-helper-pos-ind-R*:

fixes p :: *real poly*
assumes $nonzero$: $p \neq 0$
fixes qs :: *real poly list*
fixes I :: $nat\ list * nat\ list$
fixes $sign$:: *rat list*
fixes x :: *real*
assumes $root-p$: $x \in \{x. poly\ p\ x = 0\}$
assumes $sign-fix$: $sign = consistent-sign-vec\ qs\ x$
assumes $asm1$: $list-constr\ (fst\ I)\ (length\ qs)$
assumes $asm2$: $list-constr\ (snd\ I)\ (length\ qs)$
shows $((\forall p \in set\ (retrieve-polys\ qs\ (fst\ I)). poly\ p\ x = 0) \wedge (poly\ (prod-list\ (retrieve-polys\ qs\ (snd\ I)))\ x > 0)) \iff (z-R\ I\ sign = 1)$

<proof>

lemma *horiz-vector-helper-pos-R:*

fixes *p*:: real poly
assumes *nonzero*: $p \neq 0$
fixes *qs*:: real poly list
fixes *I*:: nat list*nat list
fixes *sign*:: rat list
fixes *x*:: real
assumes *root-p*: $x \in \{x. \text{poly } p \ x = 0\}$
assumes *sign-fix*: $\text{sign} = \text{consistent-sign-vec } qs \ x$
assumes *welldefined1*: list-constr (fst *I*) (length *qs*)
assumes *welldefined2*: list-constr (snd *I*) (length *qs*)
shows $((\forall p \in \text{set } (\text{retrieve-polys } qs \ (\text{fst } I)). \text{poly } p \ x = 0) \wedge (\text{poly } (\text{prod-list } (\text{retrieve-polys } qs \ (\text{snd } I))) \ x > 0)) \longleftrightarrow (z\text{-R } I \ \text{sign} = 1)$
<proof>

lemma *horiz-vector-helper-neg-R:*

fixes *p*:: real poly
assumes *nonzero*: $p \neq 0$
fixes *qs*:: real poly list
fixes *I*:: nat list*nat list
fixes *sign*:: rat list
fixes *x*:: real
assumes *root-p*: $x \in \{x. \text{poly } p \ x = 0\}$
assumes *sign-fix*: $\text{sign} = \text{consistent-sign-vec } qs \ x$
assumes *welldefined1*: list-constr (fst *I*) (length *qs*)
assumes *welldefined2*: list-constr (snd *I*) (length *qs*)
shows $((\forall p \in \text{set } (\text{retrieve-polys } qs \ (\text{fst } I)). \text{poly } p \ x = 0) \wedge (\text{poly } (\text{prod-list } (\text{retrieve-polys } qs \ (\text{snd } I))) \ x < 0)) \longleftrightarrow (z\text{-R } I \ \text{sign} = -1)$
<proof>

lemma *lhs-dot-rewrite:*

fixes *p*:: real poly
fixes *qs*:: real poly list
fixes *I*:: nat list*nat list
fixes *signs*:: rat list list
assumes *nonzero*: $p \neq 0$
shows
 $(\text{vec-of-list } (\text{mtx-row-R } signs \ I) \cdot (\text{construct-lhs-vector-R } p \ qs \ signs)) =$
 $\text{sum-list } (\text{map } (\lambda s. (z\text{-R } I \ s) * \text{rat-of-int } (\text{card } \{x. \text{poly } p \ x = 0 \wedge \text{consistent-sign-vec } qs \ x = s\})) \ signs)$
<proof>

lemma *construct-lhs-vector-drop-consistent-R:*

fixes *p*:: real poly
fixes *qs*:: real poly list
fixes *I*:: nat list*nat list

fixes *signs*:: rat list list
assumes *nonzero*: $p \neq 0$
assumes *distinct-signs*: distinct signs
assumes *all-info*: set (characterize-consistent-signs-at-roots p qs) \subseteq set(*signs*)
assumes *welldefined1*: list-constr (fst I) (length qs)
assumes *welldefined2*: list-constr (snd I) (length qs)
shows
 $(\text{vec-of-list } (\text{mtx-row-R } \textit{signs } I) \cdot (\text{construct-lhs-vector-R } p \textit{qs } \textit{signs})) =$
 $(\text{vec-of-list } (\text{mtx-row-R } (\text{characterize-consistent-signs-at-roots } p \textit{qs}) I) \cdot$
 $(\text{construct-lhs-vector-R } p \textit{qs } (\text{characterize-consistent-signs-at-roots } p \textit{qs})))$
 <proof>

lemma *matrix-equation-helper-step-R*:

fixes p :: real poly
fixes qs :: real poly list
fixes I :: nat list*nat list
fixes *signs*:: rat list list
assumes *nonzero*: $p \neq 0$
assumes *distinct-signs*: distinct signs
assumes *all-info*: set (characterize-consistent-signs-at-roots p qs) \subseteq set(*signs*)
assumes *welldefined1*: list-constr (fst I) (length qs)
assumes *welldefined2*: list-constr (snd I) (length qs)
shows $(\text{vec-of-list } (\text{mtx-row-R } \textit{signs } I) \cdot (\text{construct-lhs-vector-R } p \textit{qs } \textit{signs})) =$
 $\text{rat-of-int } (\text{card } \{x. \text{poly } p \ x = 0 \wedge (\forall p \in \text{set } (\text{retrieve-polys } \textit{qs } (\text{fst } I)). \text{poly } p \ x$
 $= 0) \wedge \text{poly } (\text{prod-list } (\text{retrieve-polys } \textit{qs } (\text{snd } I))) \ x > 0\}) -$
 $\text{rat-of-int } (\text{card } \{x. \text{poly } p \ x = 0 \wedge (\forall p \in \text{set } (\text{retrieve-polys } \textit{qs } (\text{fst } I)). \text{poly } p \ x$
 $= 0) \wedge \text{poly } (\text{prod-list } (\text{retrieve-polys } \textit{qs } (\text{snd } I))) \ x < 0\})$
 <proof>

lemma *matrix-equation-main-step-R*:

fixes p :: real poly
fixes qs :: real poly list
fixes I :: nat list*nat list
fixes *signs*:: rat list list
assumes *nonzero*: $p \neq 0$
assumes *distinct-signs*: distinct signs
assumes *all-info*: set (characterize-consistent-signs-at-roots p qs) \subseteq set(*signs*)
assumes *welldefined1*: list-constr (fst I) (length qs)
assumes *welldefined2*: list-constr (snd I) (length qs)
shows $(\text{vec-of-list } (\text{mtx-row-R } \textit{signs } I) \cdot$
 $(\text{construct-lhs-vector-R } p \textit{qs } \textit{signs})) =$
 $\text{construct-NofI-R } p (\text{retrieve-polys } \textit{qs } (\text{fst } I)) (\text{retrieve-polys } \textit{qs } (\text{snd } I))$
 <proof>

lemma *mtx-row-length-R*:

$\text{list-all } (\lambda r. \text{length } r = \text{length } \textit{signs}) (\text{map } (\text{mtx-row-R } \textit{signs}) \textit{ls})$
 <proof>

theorem *matrix-equation-R*:

fixes *p*:: *real poly*

fixes *qs*:: *real poly list*

fixes *subsets*:: *(nat list*nat list) list*

fixes *signs*:: *rat list list*

assumes *nonzero*: $p \neq 0$

assumes *distinct-signs*: *distinct signs*

assumes *all-info*: $\text{set}(\text{characterize-consistent-signs-at-roots } p \text{ } qs) \subseteq \text{set}(\text{signs})$

assumes *welldefined*: *all-list-constr-R* (*subsets*) (*length qs*)

shows *satisfy-equation-R* *p qs subsets signs*

<proof>

lemma *consistent-signs-at-roots-eq*:

assumes $p \neq 0$

shows *consistent-signs-at-roots* *p qs* =

$\text{set}(\text{characterize-consistent-signs-at-roots } p \text{ } qs)$

<proof>

abbreviation *w-vec-R*:: *real poly* \Rightarrow *real poly list* \Rightarrow *rat list list* \Rightarrow *rat vec*

where *w-vec-R* \equiv *construct-lhs-vector-R*

abbreviation *v-vec-R*:: *real poly* \Rightarrow *real poly list* \Rightarrow *(nat list*nat list) list* \Rightarrow *rat vec*

where *v-vec-R* \equiv *construct-rhs-vector-R*

abbreviation *M-mat-R*:: *rat list list* \Rightarrow *(nat list*nat list) list* \Rightarrow *rat mat*

where *M-mat-R* \equiv *matrix-A-R*

theorem *matrix-equation-pretty*:

fixes *subsets*:: *(nat list*nat list) list*

assumes $p \neq 0$

assumes *distinct signs*

assumes *consistent-signs-at-roots* $p \text{ } qs \subseteq \text{set } \text{signs}$

assumes $\bigwedge a \ b \ i. (a, b) \in \text{set}(\text{subsets}) \implies (i \in \text{set } a \vee i \in \text{set } b) \implies i < \text{length } qs$

shows $M\text{-mat-R } \text{signs } \text{subsets} *_v \text{ } w\text{-vec-R } p \text{ } qs \text{ } \text{signs} = v\text{-vec-R } p \text{ } qs \text{ } \text{subsets}$

<proof>

24 Base Case

definition *satisfies-properties-R*:: *real poly* \Rightarrow *real poly list* \Rightarrow *(nat list*nat list) list* \Rightarrow *rat list list* \Rightarrow *rat mat* \Rightarrow *bool*

where *satisfies-properties-R* $p \text{ } qs \text{ } \text{subsets } \text{signs } \text{matrix} =$

$(\text{all-list-constr-R } \text{subsets} (\text{length } qs) \wedge \text{well-def-signs } (\text{length } qs) \text{ } \text{signs} \wedge \text{distinct signs}$

$\wedge \text{satisfy-equation-R } p \text{ } qs \text{ } \text{subsets } \text{signs} \wedge \text{invertible-mat } \text{matrix} \wedge \text{matrix} = \text{matrix-A-R } \text{signs } \text{subsets}$

$\wedge \text{set}(\text{characterize-consistent-signs-at-roots } p \text{ } qs) \subseteq \text{set}(\text{signs})$
 $)$

lemma *mat-base-case-R:*

shows *matrix-A-R* $[[1],[0],[-1]] \ [(\square, \square), ([0], \square), (\square, [0])] = (\text{mat-of-rows-list } 3$
 $[[1,1,1], [0,1,0], [1,0,-1]])$
 $\langle \text{proof} \rangle$

lemma *base-case-sgas-R:*

fixes $q \ p:: \text{real poly}$
assumes *nonzero:* $p \neq 0$
shows $\text{set}(\text{characterize-consistent-signs-at-roots } p \ [q]) \subseteq \{[1],[0], [-1]\}$
 $\langle \text{proof} \rangle$

lemma *base-case-sgas-alt-R:*

fixes $p:: \text{real poly}$
fixes $qs:: \text{real poly list}$
assumes *len1:* $\text{length } qs = 1$
assumes *nonzero:* $p \neq 0$
shows $\text{set}(\text{characterize-consistent-signs-at-roots } p \ qs) \subseteq \{[1], [0], [-1]\}$
 $\langle \text{proof} \rangle$

lemma *base-case-satisfy-equation-R:*

fixes $q \ p:: \text{real poly}$
assumes *nonzero:* $p \neq 0$
shows *satisfy-equation-R* $p \ [q] \ [(\square, \square), ([0], \square), (\square, [0])] \ [[1],[0],[-1]]$
 $\langle \text{proof} \rangle$

lemma *base-case-satisfy-equation-alt-R:*

fixes $p:: \text{real poly}$
fixes $qs:: \text{real poly list}$
assumes *len1:* $\text{length } qs = 1$
assumes *nonzero:* $p \neq 0$
shows *satisfy-equation-R* $p \ qs \ [(\square, \square), ([0], \square), (\square, [0])] \ [[1],[0],[-1]]$
 $\langle \text{proof} \rangle$

lemma *base-case-matrix-eq:*

fixes $q \ p:: \text{real poly}$
assumes *nonzero:* $p \neq 0$
shows $(\text{mult-mat-vec}(\text{mat-of-rows-list } 3 \ [[1,1,1], [0,1,0], [1,0,-1]]) (\text{construct-lhs-vector-R}$
 $p \ [q] \ [[1],[0],[-1]]) =$
 $(\text{construct-rhs-vector-R } p \ [q] \ [(\square, \square), ([0], \square), (\square, [0])])$
 $\langle \text{proof} \rangle$

lemma *less-three:* $(n::\text{nat}) < \text{Suc}(\text{Suc}(\text{Suc } 0)) \longleftrightarrow n = 0 \vee n = 1 \vee n = 2$

$\langle \text{proof} \rangle$

lemma *inverse-mat-base-case-R:*

shows *inverts-mat* $(\text{mat-of-rows-list } 3 \ [[1/2, -1/2, 1/2], [0, 1, 0], [1/2, -1/2,$

$-1/2]$::rat mat) (mat-of-rows-list 3 [[1,1,1], [0,1,0], [1,0,-1]]::rat mat)
 ⟨proof⟩

lemma inverse-mat-base-case-2-R:

shows inverts-mat (mat-of-rows-list 3 [[1,1,1], [0,1,0], [1,0,-1]]::rat mat) (mat-of-rows-list 3 [[1/2, -1/2, 1/2], [0, 1, 0], [1/2, -1/2, -1/2]]:: rat mat)
 ⟨proof⟩

lemma base-case-invertible-mat-R:

shows invertible-mat (matrix-A-R [[1],[0], [- 1]] [([]), ([]),([0], [])],([], [0]))
 ⟨proof⟩

25 Inductive Step

25.1 Lemmas on smashing subsets

definition subsets-first-component-list::(nat list*nat list) list \Rightarrow nat list list
where subsets-first-component-list I = map ($\lambda I.$ (fst I)) I

definition subsets-second-component-list::(nat list*nat list) list \Rightarrow nat list list
where subsets-second-component-list I = map ($\lambda I.$ (snd I)) I

definition smash-list-list::('a list*'a list) list \Rightarrow ('a list*'a list) list \Rightarrow ('a list*'a list) list
where smash-list-list s1 s2 = concat (map ($\lambda l1.$ map ($\lambda l2.$ (fst l1 @ fst l2, snd l1 @ snd l2)) s2) s1)

lemma smash-list-list-property-set:

fixes l1 l2 :: ('a list*'a list) list
fixes a b:: nat
shows \forall (elem :: ('a list*'a list)). (elem \in (set (smash-list-list l1 l2))) \longrightarrow
 (\exists (elem1:: ('a list*'a list)). \exists (elem2:: ('a list*'a list)).
 (elem1 \in set(l1) \wedge elem2 \in set(l2) \wedge elem = (fst elem1@ fst elem2, snd elem1 @ snd elem2))))
 ⟨proof⟩

lemma subsets-smash-property-R:

fixes subsets1 subsets2 :: (nat list*nat list) list
fixes n:: nat
shows \forall (elem :: nat list*nat list). (List.member (subsets-smash-R n subsets1 subsets2) elem) \longrightarrow
 (\exists (elem1::nat list*nat list). \exists (elem2::nat list*nat list).
 (elem1 \in set(subsets1) \wedge elem2 \in set(subsets2) \wedge elem = ((fst elem1) @ (map ((+) n) (fst elem2)), (snd elem1) @ (map ((+) n) (snd elem2)))))
 ⟨proof⟩

25.2 Well-defined subsets preserved when smashing

lemma *well-def-step-R*:
fixes $qs1\ qs2 :: \text{real poly list}$
fixes $subsets1\ subsets2 :: (\text{nat list} * \text{nat list})\ \text{list}$
assumes $\text{well-def-subsets1}: \text{all-list-constr-R}\ (subsets1)\ (\text{length}\ qs1)$
assumes $\text{well-def-subsets2}: \text{all-list-constr-R}\ (subsets2)\ (\text{length}\ qs2)$
shows $\text{all-list-constr-R}\ ((\text{subsets-smash-R}\ (\text{length}\ qs1)\ subsets1\ subsets2))$
 $(\text{length}\ (qs1\ @\ qs2))$
 $\langle \text{proof} \rangle$

25.3 Consistent Sign Assignments Preserved When Smashing

lemma *subset-helper-R*:
fixes $p :: \text{real poly}$
fixes $qs1\ qs2 :: \text{real poly list}$
fixes $signs1\ signs2 :: \text{rat list list}$
shows $\forall x \in \text{set}\ (\text{characterize-consistent-signs-at-roots}\ p\ (qs1\ @\ qs2)).$
 $\exists x1 \in \text{set}\ (\text{characterize-consistent-signs-at-roots}\ p\ qs1).$
 $\exists x2 \in \text{set}\ (\text{characterize-consistent-signs-at-roots}\ p\ qs2).$
 $x = x1 @ x2$
 $\langle \text{proof} \rangle$

lemma *subset-step-R*:
fixes $p :: \text{real poly}$
fixes $qs1\ qs2 :: \text{real poly list}$
fixes $signs1\ signs2 :: \text{rat list list}$
assumes $\text{csa1}: \text{set}\ (\text{characterize-consistent-signs-at-roots}\ p\ qs1) \subseteq \text{set}\ (signs1)$
assumes $\text{csa2}: \text{set}\ (\text{characterize-consistent-signs-at-roots}\ p\ qs2) \subseteq \text{set}\ (signs2)$
shows $\text{set}\ (\text{characterize-consistent-signs-at-roots}\ p$
 $(qs1\ @\ qs2))$
 $\subseteq \text{set}\ (\text{signs-smash}\ signs1\ signs2)$
 $\langle \text{proof} \rangle$

25.4 Main Results

lemma *dim-row-matrix-A-R[simp]*:
shows $\text{dim-row}\ (\text{matrix-A-R}\ signs\ subsets) = \text{length}\ subsets$
 $\langle \text{proof} \rangle$

lemma *dim-col-matrix-A-R[simp]*:
shows $\text{dim-col}\ (\text{matrix-A-R}\ signs\ subsets) = \text{length}\ signs$
 $\langle \text{proof} \rangle$

lemma *length-subsets-smash-R*:
shows
 $\text{length}\ (\text{subsets-smash-R}\ n\ subs1\ subs2) = \text{length}\ subs1 * \text{length}\ subs2$
 $\langle \text{proof} \rangle$

lemma *z-append-R*:

fixes *xs*:: (nat list * nat list)
assumes $\bigwedge i. i \in \text{set } (\text{fst } xs) \implies i < \text{length } as$
assumes $\bigwedge i. i \in \text{set } (\text{snd } xs) \implies i < \text{length } as$
shows $z\text{-R } ((\text{fst } xs) @ (\text{map } ((+) (\text{length } as)) (\text{fst } ys)), (\text{snd } xs) @ (\text{map } ((+) (\text{length } as)) (\text{snd } ys))) (as @ bs) = z\text{-R } xs \text{ as } * z\text{-R } ys \text{ bs}$
 $\langle \text{proof} \rangle$

lemma *matrix-construction-is-kronecker-product-R*:

fixes *qs1* :: real poly list
fixes *subs1 subs2* :: (nat list*nat list) list
fixes *signs1 signs2* :: rat list list

assumes $\bigwedge l. l \in \text{set } \text{subs1} \implies (i \in \text{set } (\text{fst } l) \vee i \in \text{set } (\text{snd } l)) \implies i < n1$
assumes $\bigwedge j. j \in \text{set } \text{signs1} \implies \text{length } j = n1$
shows $(\text{matrix-A-R } (\text{signs-smash } \text{signs1 } \text{signs2}) (\text{subsets-smash-R } n1 \text{ subs1 } \text{subs2}))$
 $=$
 $\text{kronecker-product } (\text{matrix-A-R } \text{signs1 } \text{subs1}) (\text{matrix-A-R } \text{signs2 } \text{subs2})$
 $\langle \text{proof} \rangle$

lemma *inductive-step-R*:

fixes *p*:: real poly
fixes *qs1 qs2* :: real poly list
fixes *subsets1 subsets2* :: (nat list*nat list) list
fixes *signs1 signs2* :: rat list list
assumes *nonzero*: $p \neq 0$
assumes *nontriv1*: $\text{length } qs1 > 0$
assumes *nontriv2*: $\text{length } qs2 > 0$
assumes *welldefined-signs1*: $\text{well-def-signs } (\text{length } qs1) \text{ signs1}$
assumes *welldefined-signs2*: $\text{well-def-signs } (\text{length } qs2) \text{ signs2}$
assumes *distinct-signs1*: $\text{distinct } \text{signs1}$
assumes *distinct-signs2*: $\text{distinct } \text{signs2}$
assumes *all-info1*: $\text{set } (\text{characterize-consistent-signs-at-roots } p \text{ qs1}) \subseteq \text{set}(\text{signs1})$
assumes *all-info2*: $\text{set } (\text{characterize-consistent-signs-at-roots } p \text{ qs2}) \subseteq \text{set}(\text{signs2})$
assumes *welldefined-subsets1*: $\text{all-list-constr-R } (\text{subsets1}) (\text{length } qs1)$
assumes *welldefined-subsets2*: $\text{all-list-constr-R } (\text{subsets2}) (\text{length } qs2)$
assumes *invertibleMtx1*: $\text{invertible-mat } (\text{matrix-A-R } \text{signs1 } \text{subsets1})$
assumes *invertibleMtx2*: $\text{invertible-mat } (\text{matrix-A-R } \text{signs2 } \text{subsets2})$
shows $\text{satisfy-equation-R } p (qs1@qs2) (\text{subsets-smash-R } (\text{length } qs1) \text{ subsets1 } \text{subsets2}) (\text{signs-smash } \text{signs1 } \text{signs2})$
 $\wedge \text{invertible-mat } (\text{matrix-A-R } (\text{signs-smash } \text{signs1 } \text{signs2}) (\text{subsets-smash-R } (\text{length } qs1) \text{ subsets1 } \text{subsets2}))$
 $\langle \text{proof} \rangle$

26 Reduction Step Proofs

definition *get-matrix-R*:: (rat mat × ((nat list*nat list) list × rat list list)) ⇒ rat mat
where *get-matrix-R data* = fst(data)

definition *get-subsets-R*:: (rat mat × ((nat list*nat list) list × rat list list)) ⇒ (nat list*nat list) list
where *get-subsets-R data* = fst(snd(data))

definition *get-signs-R*:: (rat mat × ((nat list*nat list) list × rat list list)) ⇒ rat list list
where *get-signs-R data* = snd(snd(data))

definition *reduction-signs-R*:: real poly ⇒ real poly list ⇒ rat list list ⇒ (nat list*nat list) list ⇒ rat mat ⇒ rat list list
where *reduction-signs-R p qs signs subsets matr* =
 (take-indices signs (find-nonzeros-from-input-vec (solve-for-lhs-R p qs subsets matr)))

definition *reduction-subsets-R*:: real poly ⇒ real poly list ⇒ rat list list ⇒ (nat list*nat list) list ⇒ rat mat ⇒ (nat list*nat list) list
where *reduction-subsets-R p qs signs subsets matr* =
 (take-indices subsets (rows-to-keep (reduce-mat-cols matr (solve-for-lhs-R p qs subsets matr))))

lemma *reduction-signs-is-get-signs-R*: reduction-signs-R p qs signs subsets m =
 get-signs-R (reduce-system-R p (qs, (m, (subsets, signs))))
 ⟨proof⟩

lemma *reduction-subsets-is-get-subsets-R*: reduction-subsets-R p qs signs subsets m
 = get-subsets-R (reduce-system-R p (qs, (m, (subsets, signs))))
 ⟨proof⟩

26.1 Showing sign conditions preserved when reducing

lemma *take-indices-lem-R*:
fixes *p*:: real poly
fixes *qs* :: real poly list
fixes *arb-list* :: ('a list*'a list) list
fixes *index-list* :: nat list
fixes *n*:: nat
assumes *lest*: n < length (take-indices arb-list index-list)
assumes *well-def*: ∀ q.(List.member index-list q → q < length arb-list)
shows ∃ k<length arb-list.
 (take-indices arb-list index-list) ! n = arb-list ! k
 ⟨proof⟩

lemma *size-of-mat-R*:

fixes *subsets* :: (nat list*nat list) list
fixes *signs* :: rat list list
shows (matrix-A-R signs subsets) ∈ carrier-mat (length subsets) (length signs)
 ⟨proof⟩

lemma *size-of-lhs-R*:
fixes *p*:: real poly
fixes *qs* :: real poly list
fixes *signs* :: rat list list
shows dim-vec (construct-lhs-vector-R p qs signs) = length signs
 ⟨proof⟩

lemma *size-of-rhs-R*:
fixes *p*:: real poly
fixes *qs* :: real poly list
fixes *subsets* :: (nat list*nat list) list
shows dim-vec (construct-rhs-vector-R p qs subsets) = length subsets
 ⟨proof⟩

lemma *same-size-R*:
fixes *p*:: real poly
fixes *qs* :: real poly list
fixes *subsets* :: (nat list*nat list) list
fixes *signs* :: rat list list
assumes invertible-mat: invertible-mat (matrix-A-R signs subsets)
shows length subsets = length signs
 ⟨proof⟩

lemma *construct-lhs-matches-solve-for-lhs-R*:
fixes *p*:: real poly
fixes *qs* :: real poly list
fixes *subsets* :: (nat list*nat list) list
fixes *signs* :: rat list list
assumes match: satisfy-equation-R p qs subsets signs
assumes invertible-mat: invertible-mat (matrix-A-R signs subsets)
shows (construct-lhs-vector-R p qs signs) = solve-for-lhs-R p qs subsets (matrix-A-R signs subsets)
 ⟨proof⟩

lemma *reduction-doesnt-break-things-signs-R*:
fixes *p*:: real poly
fixes *qs* :: real poly list
fixes *subsets* :: (nat list*nat list) list
fixes *signs* :: rat list list
assumes nonzero: $p \neq 0$
assumes welldefined-signs1: well-def-signs (length qs) signs
assumes distinct-signs: distinct signs
assumes all-info: set (characterize-consistent-signs-at-roots p qs) ⊆ set(signs)

assumes *match: satisfy-equation-R* p qs *subsets signs*
assumes *invertible-mat: invertible-mat* (*matrix-A-R signs subsets*)
shows set (*characterize-consistent-signs-at-roots* p qs) \subseteq set (*reduction-signs-R* p qs *signs subsets* (*matrix-A-R signs subsets*))
<proof>

lemma *reduction-deletes-bad-sign-conds-R:*

fixes p :: *real poly*
fixes qs :: *real poly list*
fixes $subsets$:: (*nat list*nat list*) *list*
fixes $signs$:: *rat list list*
assumes *nonzero: p ≠ 0*
assumes *welldefined-signs1: well-def-signs* (*length* qs) *signs*
assumes *distinct-signs: distinct signs*
assumes *all-info: set* (*characterize-consistent-signs-at-roots* p qs) \subseteq set (*signs*)
assumes *match: satisfy-equation-R* p qs *subsets signs*
assumes *invertible-mat: invertible-mat* (*matrix-A-R signs subsets*)
shows set (*characterize-consistent-signs-at-roots* p qs) = set (*reduction-signs-R* p qs *signs subsets* (*matrix-A-R signs subsets*))
<proof>

theorem *reduce-system-sign-conditions-R:*

fixes p :: *real poly*
fixes qs :: *real poly list*
fixes $subsets$:: (*nat list*nat list*) *list*
fixes $signs$:: *rat list list*
assumes *nonzero: p ≠ 0*
assumes *welldefined-signs1: well-def-signs* (*length* qs) *signs*
assumes *distinct-signs: distinct signs*
assumes *all-info: set* (*characterize-consistent-signs-at-roots* p qs) \subseteq set (*signs*)
assumes *match: satisfy-equation-R* p qs *subsets signs*
assumes *invertible-mat: invertible-mat* (*matrix-A-R signs subsets*)
shows set (*get-signs-R* (*reduce-system-R* p (qs , ((*matrix-A-R signs subsets*), (*subsets, signs*)))))) = set (*characterize-consistent-signs-at-roots* p qs)
<proof>

26.2 Showing matrix equation preserved when reducing

lemma *reduce-system-matrix-equation-preserved-R:*

fixes p :: *real poly*
fixes qs :: *real poly list*
fixes $subsets$:: (*nat list*nat list*) *list*
fixes $signs$:: *rat list list*
assumes *nonzero: p ≠ 0*
assumes *welldefined-signs: well-def-signs* (*length* qs) *signs*
assumes *welldefined-subsets: all-list-constr-R* (*subsets*) (*length* qs)
assumes *distinct-signs: distinct signs*
assumes *all-info: set* (*characterize-consistent-signs-at-roots* p qs) \subseteq set (*signs*)

assumes *match*: *satisfy-equation-R* *p* *qs* *subsets* *signs*
assumes *invertible-mat*: *invertible-mat* (*matrix-A-R* *signs* *subsets*)
shows *satisfy-equation-R* *p* *qs* (*get-subsets-R* (*reduce-system-R* *p* (*qs*, ((*matrix-A-R* *signs* *subsets*), (*subsets*, *signs*))))))
(*get-signs-R* (*reduce-system-R* *p* (*qs*, ((*matrix-A-R* *signs* *subsets*), (*subsets*, *signs*))))))
⟨*proof*⟩

26.3 Showing matrix preserved

lemma *reduce-system-matrix-signs-helper-aux-R*:

fixes *p*:: *real poly*
fixes *qs*:: *real poly list*
fixes *subsets*:: (*nat list***nat list*) *list*
fixes *signs*:: *rat list list*
fixes *S*:: *nat list*
assumes *well-def-h*: $\forall x. \text{List.member } S \ x \longrightarrow x < \text{length } \text{signs}$
assumes *nonzero*: $p \neq 0$
shows *alt-matrix-A-R* (*take-indices* *signs* *S*) *subsets* = *take-cols-from-matrix* (*alt-matrix-A-R* *signs* *subsets*) *S*
⟨*proof*⟩

lemma *reduce-system-matrix-signs-helper-R*:

fixes *p*:: *real poly*
fixes *qs*:: *real poly list*
fixes *subsets*:: (*nat list***nat list*) *list*
fixes *signs*:: *rat list list*
fixes *S*:: *nat list*
assumes *well-def-h*: $\forall x. \text{List.member } S \ x \longrightarrow x < \text{length } \text{signs}$
assumes *nonzero*: $p \neq 0$
shows *matrix-A-R* (*take-indices* *signs* *S*) *subsets* = *take-cols-from-matrix* (*matrix-A-R* *signs* *subsets*) *S*
⟨*proof*⟩

lemma *reduce-system-matrix-subsets-helper-aux-R*:

fixes *p*:: *real poly*
fixes *qs*:: *real poly list*
fixes *subsets*:: (*nat list** *nat list*) *list*
fixes *signs*:: *rat list list*
fixes *S*:: *nat list*
assumes *inv*: $\text{length } \text{subsets} \geq \text{length } \text{signs}$
assumes *well-def-h*: $\forall x. \text{List.member } S \ x \longrightarrow x < \text{length } \text{subsets}$
assumes *nonzero*: $p \neq 0$
shows *alt-matrix-A-R* *signs* (*take-indices* *subsets* *S*) = *take-rows-from-matrix* (*alt-matrix-A-R* *signs* *subsets*) *S*
⟨*proof*⟩

lemma *reduce-system-matrix-subsets-helper-R*:

fixes p :: *real poly*
fixes qs :: *real poly list*
fixes $subsets$:: *(nat list*nat list) list*
fixes $signs$:: *rat list list*
fixes S :: *nat list*
assumes *nonzero*: $p \neq 0$
assumes *inv*: $\text{length } subsets \geq \text{length } signs$
assumes *well-def-h*: $\forall x. \text{List.member } S \ x \longrightarrow x < \text{length } subsets$
shows *matrix-A-R signs (take-indices subsets S) = take-rows-from-matrix (matrix-A-R signs subsets) S*
<proof>

lemma *reduce-system-matrix-match-R*:

fixes p :: *real poly*
fixes qs :: *real poly list*
fixes $subsets$:: *(nat list*nat list) list*
fixes $signs$:: *rat list list*
assumes *nonzero*: $p \neq 0$
assumes *welldefined-signs1*: *well-def-signs (length qs) signs*
assumes *distinct-signs*: *distinct signs*
assumes *all-info*: $\text{set } (\text{characterize-consistent-signs-at-roots } p \ qs) \subseteq \text{set}(signs)$
assumes *match*: *satisfy-equation-R p qs subsets signs*
assumes *inv*: *invertible-mat (matrix-A-R signs subsets)*
shows *matrix-A-R (get-signs-R (reduce-system-R p (qs, ((matrix-A-R signs subsets), (subsets, signs)))))) = (get-subsets-R (reduce-system-R p (qs, ((matrix-A-R signs subsets), (subsets, signs)))))) = (get-matrix-R (reduce-system-R p (qs, ((matrix-A-R signs subsets), (subsets, signs))))))*
<proof>

26.4 Showing invertibility preserved when reducing

thm *conjugatable-vec-space.gauss-jordan-single-rank*

thm *vec-space.full-rank-lin-indpt*

lemma *well-def-find-zeros-from-lhs-vec-R*:

fixes p :: *real poly*
fixes qs :: *real poly list*
fixes $subsets$:: *(nat list*nat list) list*
fixes $signs$:: *rat list list*
assumes *len-eq*: $\text{length } subsets = \text{length } signs$
assumes *inv*: *invertible-mat (matrix-A-R signs subsets)*
assumes *nonzero*: $p \neq 0$
assumes *welldefined-signs1*: *well-def-signs (length qs) signs*
assumes *distinct-signs*: *distinct signs*
assumes *all-info*: $\text{set } (\text{characterize-consistent-signs-at-roots } p \ qs) \subseteq \text{set}(signs)$

assumes *match: satisfy-equation-R p qs subsets signs*
shows $(\bigwedge j. j \in \text{set} (\text{find-nonzeros-from-input-vec}$
 $\quad (\text{solve-for-lhs-R } p \text{ } qs \text{ } \text{subsets} (\text{matrix-A-R } \text{signs } \text{subsets}))) \implies$
 $\quad j < \text{length} (\text{cols} (\text{matrix-A-R } \text{signs } \text{subsets})))$
 <proof>

lemma *take-cols-subsets-og-cols-R:*

fixes *p:: real poly*
fixes *qs :: real poly list*
fixes *subsets :: (nat list*nat list) list*
fixes *signs :: rat list list*
assumes *len-eq: length subsets = length signs*
assumes *inv: invertible-mat (matrix-A-R signs subsets)*
assumes *nonzero: p ≠ 0*
assumes *welldefined-signs1: well-def-signs (length qs) signs*
assumes *distinct-signs: distinct signs*
assumes *all-info: set (characterize-consistent-signs-at-roots p qs) ⊆ set(signs)*
assumes *match: satisfy-equation-R p qs subsets signs*
shows *set (take-indices (cols (matrix-A-R signs subsets))*
 $\quad (\text{find-nonzeros-from-input-vec} (\text{solve-for-lhs-R } p \text{ } qs \text{ } \text{subsets} (\text{matrix-A-R}$
 $\quad \text{signs } \text{subsets}))))$
 $\subseteq \text{set} (\text{cols} (\text{matrix-A-R } \text{signs } \text{subsets}))$
 <proof>

lemma *reduction-doesnt-break-things-invertibility-step1-R:*

fixes *p:: real poly*
fixes *qs :: real poly list*
fixes *subsets :: (nat list*nat list) list*
fixes *signs :: rat list list*
assumes *len-eq: length subsets = length signs*
assumes *inv: invertible-mat (matrix-A-R signs subsets)*
assumes *nonzero: p ≠ 0*
assumes *welldefined-signs1: well-def-signs (length qs) signs*
assumes *distinct-signs: distinct signs*
assumes *all-info: set (characterize-consistent-signs-at-roots p qs) ⊆ set(signs)*
assumes *match: satisfy-equation-R p qs subsets signs*
shows *vec-space.rank (length signs) (reduce-mat-cols (matrix-A-R signs subsets)*
 $\quad (\text{solve-for-lhs-R } p \text{ } qs \text{ } \text{subsets} (\text{matrix-A-R } \text{signs } \text{subsets}))) =$
 $\quad (\text{length} (\text{find-nonzeros-from-input-vec} (\text{solve-for-lhs-R } p \text{ } qs \text{ } \text{subsets} (\text{matrix-A-R}$
 $\quad \text{signs } \text{subsets}))))$
 <proof>

lemma *reduction-doesnt-break-things-invertibility-R:*

fixes *p:: real poly*
fixes *qs :: real poly list*
fixes *subsets :: (nat list*nat list) list*

fixes *signs* :: *rat list list*
assumes *len-eq*: *length subsets = length signs*
assumes *inv*: *invertible-mat (matrix-A-R signs subsets)*
assumes *nonzero*: *p ≠ 0*
assumes *welldefined-signs1*: *well-def-signs (length qs) signs*
assumes *distinct-signs*: *distinct signs*
assumes *all-info*: *set (characterize-consistent-signs-at-roots p qs) ⊆ set(signs)*
assumes *match*: *satisfy-equation-R p qs subsets signs*
shows *invertible-mat (get-matrix-R (reduce-system-R p (qs, ((matrix-A-R signs subsets), (subsets, signs))))))*
<proof>

26.5 Well def signs preserved when reducing

lemma *reduction-doesnt-break-length-signs-R*:

fixes *p*:: *real poly*
fixes *qs* :: *real poly list*
fixes *subsets* :: *(nat list*nat list) list*
fixes *signs* :: *rat list list*
assumes *length-init* : $\forall x \in \text{set}(\text{signs}). \text{length } x = \text{length } qs$
assumes *sat-eq*: *satisfy-equation-R p qs subsets signs*
assumes *inv-mat*: *invertible-mat (matrix-A-R signs subsets)*
shows $\forall x \in \text{set}(\text{reduction-signs-R } p \text{ } qs \text{ } signs \text{ } subsets \text{ } (\text{matrix-A-R } signs \text{ } subsets))$.

length x = length qs
<proof>

26.6 Distinct signs preserved when reducing

lemma *reduction-signs-are-distinct-R*:

fixes *p*:: *real poly*
fixes *qs* :: *real poly list*
fixes *subsets* :: *(nat list*nat list) list*
fixes *signs* :: *rat list list*
assumes *sat-eq*: *satisfy-equation-R p qs subsets signs*
assumes *inv-mat*: *invertible-mat (matrix-A-R signs subsets)*
assumes *distinct-init*: *distinct signs*
shows *distinct (reduction-signs-R p qs signs subsets (matrix-A-R signs subsets))*
<proof>

26.7 Well def subsets preserved when reducing

lemma *reduction-doesnt-break-subsets-R*:

fixes *p*:: *real poly*
fixes *qs* :: *real poly list*
fixes *subsets* :: *(nat list* nat list) list*
fixes *signs* :: *rat list list*
assumes *nonzero*: *p ≠ 0*
assumes *length-init* : *all-list-constr-R subsets (length qs)*
assumes *sat-eq*: *satisfy-equation-R p qs subsets signs*

assumes *inv-mat*: *invertible-mat* (*matrix-A-R* *signs* *subsets*)
shows *all-list-constr-R* (*reduction-subsets-R* *p* *qs* *signs* *subsets* (*matrix-A-R* *signs* *subsets*)) (*length* *qs*)
 ⟨*proof*⟩

27 Overall Lemmas

lemma *combining-to-smash-R*: *combine-systems-R* *p* (*qs1*, *m1*, (*sub1*, *sgn1*)) (*qs2*, *m2*, (*sub2*, *sgn2*))
 = *smash-systems-R* *p* *qs1* *qs2* *sub1* *sub2* *sgn1* *sgn2* *m1* *m2*
 ⟨*proof*⟩

lemma *getter-functions-R*: *calculate-data-R* *p* *qs* = (*get-matrix-R* (*calculate-data-R* *p* *qs*), (*get-subsets-R* (*calculate-data-R* *p* *qs*), *get-signs-R* (*calculate-data-R* *p* *qs*)))
 ⟨*proof*⟩

27.1 Key properties preserved

27.1.1 Properties preserved when combining and reducing systems

lemma *combining-sys-satisfies-properties-helper-R*:
fixes *p*:: *real poly*
fixes *qs1*:: *real poly list*
fixes *qs2*:: *real poly list*
fixes *subsets1* *subsets2*:: (*nat list* * *nat list*) *list*
fixes *signs1* *signs2*:: *rat list list*
fixes *matrix1* *matrix2*:: *rat mat*
assumes *nonzero*: $p \neq 0$
assumes *nontriv1*: $\text{length } qs1 > 0$
assumes *nontriv2*: $\text{length } qs2 > 0$
assumes *satisfies-properties-sys1*: *satisfies-properties-R* *p* *qs1* *subsets1* *signs1* *matrix1*
assumes *satisfies-properties-sys2*: *satisfies-properties-R* *p* *qs2* *subsets2* *signs2* *matrix2*
shows *satisfies-properties-R* *p* (*qs1*@*qs2*) (*get-subsets-R* (*snd* ((*combine-systems-R* *p* (*qs1*,(*matrix1*, (*subsets1*, *signs1*))) (*qs2*,(*matrix2*, (*subsets2*, *signs2*))))))
 (*get-signs-R* (*snd* ((*combine-systems-R* *p* (*qs1*,(*matrix1*, (*subsets1*, *signs1*))) (*qs2*,(*matrix2*, (*subsets2*, *signs2*))))))
 (*get-matrix-R* (*snd* ((*combine-systems-R* *p* (*qs1*,(*matrix1*, (*subsets1*, *signs1*))) (*qs2*,(*matrix2*, (*subsets2*, *signs2*))))))
 ⟨*proof*⟩

lemma *combining-sys-satisfies-properties-R*:
fixes *p*:: *real poly*
fixes *qs1*:: *real poly list*
fixes *qs2*:: *real poly list*
assumes *nonzero*: $p \neq 0$
assumes *nontriv1*: $\text{length } qs1 > 0$

assumes *nontriv2*: $\text{length } qs2 > 0$
assumes *satisfies-properties-sys1*: *satisfies-properties-R* p $qs1$ (*get-subsets-R* (*calculate-data-R* p $qs1$)) (*get-signs-R* (*calculate-data-R* p $qs1$)) (*get-matrix-R* (*calculate-data-R* p $qs1$))
assumes *satisfies-properties-sys2*: *satisfies-properties-R* p $qs2$ (*get-subsets-R* (*calculate-data-R* p $qs2$)) (*get-signs-R* (*calculate-data-R* p $qs2$)) (*get-matrix-R* (*calculate-data-R* p $qs2$))
shows *satisfies-properties-R* p ($qs1@qs2$) (*get-subsets-R* (*snd* ((*combine-systems-R* p ($qs1$,*calculate-data-R* p $qs1$) ($qs2$,*calculate-data-R* p $qs2$))))))
(*get-signs-R* (*snd* ((*combine-systems-R* p ($qs1$,*calculate-data-R* p $qs1$) ($qs2$,*calculate-data-R* p $qs2$))))))
(*get-matrix-R* (*snd* ((*combine-systems-R* p ($qs1$,*calculate-data-R* p $qs1$) ($qs2$,*calculate-data-R* p $qs2$))))))
<proof>

lemma *reducing-sys-satisfies-properties-R*:

fixes p :: *real poly*
fixes qs :: *real poly list*
fixes $subsets$:: (*nat list***nat list*) *list*
fixes $signs$:: *rat list list*
fixes $matrix$:: *rat mat*
assumes *nonzero*: $p \neq 0$
assumes *nontriv*: $\text{length } qs > 0$
assumes *satisfies-properties-sys*: *satisfies-properties-R* p qs $subsets$ $signs$ $matrix$
shows *satisfies-properties-R* p qs (*get-subsets-R* (*reduce-system-R* p (qs , $matrix$, $subsets$, $signs$)))
(*get-signs-R* (*reduce-system-R* p (qs , $matrix$, $subsets$, $signs$)))
(*get-matrix-R* (*reduce-system-R* p (qs , $matrix$, $subsets$, $signs$)))
<proof>

27.1.2 For length 1 qs

lemma *length-1-calculate-data-satisfies-properties-R*:

fixes p :: *real poly*
fixes qs :: *real poly list*
fixes $subsets$:: (*nat list***nat list*) *list*
fixes $signs$:: *rat list list*
assumes *nonzero*: $p \neq 0$
assumes *len1*: $\text{length } qs = 1$
shows *satisfies-properties-R* p qs (*get-subsets-R* (*calculate-data-R* p qs)) (*get-signs-R* (*calculate-data-R* p qs)) (*get-matrix-R* (*calculate-data-R* p qs))
<proof>

27.1.3 For arbitrary qs

lemma *calculate-data-satisfies-properties-R*:

fixes p :: *real poly*
fixes qs :: *real poly list*
fixes $subsets$:: (*nat list***nat list*) *list*
fixes $signs$:: *rat list list*
shows ($p \neq 0 \wedge (\text{length } qs > 0)$)

\longrightarrow *satisfies-properties-R* p qs (*get-subsets-R* (*calculate-data-R* p qs)) (*get-signs-R* (*calculate-data-R* p qs)) (*get-matrix-R* (*calculate-data-R* p qs))
 <proof>

27.2 Some key results on consistent sign assignments

lemma *find-consistent-signs-at-roots-len1-R*:

fixes p :: *real poly*
fixes qs :: *real poly list*
fixes $subsets$:: (*nat list*nat list*) *list*
fixes $signs$:: *rat list list*
assumes *nonzero*: $p \neq 0$
assumes *len1*: $\text{length } qs = 1$
shows $\text{set}(\text{find-consistent-signs-at-roots-R } p \text{ } qs) = \text{set}(\text{characterize-consistent-signs-at-roots } p \text{ } qs)$
 <proof>

lemma *smaller-sys-are-good-R*:

fixes p :: *real poly*
fixes $qs1$:: *real poly list*
fixes $qs2$:: *real poly list*
fixes $subsets$:: (*nat list*nat list*) *list*
fixes $signs$:: *rat list list*
assumes *nonzero*: $p \neq 0$
assumes *nontriv1*: $\text{length } qs1 > 0$
assumes *nontriv2*: $\text{length } qs2 > 0$
assumes $\text{set}(\text{find-consistent-signs-at-roots-R } p \text{ } qs1) = \text{set}(\text{characterize-consistent-signs-at-roots } p \text{ } qs1)$
assumes $\text{set}(\text{find-consistent-signs-at-roots-R } p \text{ } qs2) = \text{set}(\text{characterize-consistent-signs-at-roots } p \text{ } qs2)$
shows $\text{set}(\text{snd}(\text{snd}(\text{reduce-system-R } p \text{ } (\text{combine-systems-R } p \text{ } (qs1, \text{calculate-data-R } p \text{ } qs1) \text{ } (qs2, \text{calculate-data-R } p \text{ } qs2)))))) = \text{set}(\text{characterize-consistent-signs-at-roots } p \text{ } (qs1@qs2))$
 <proof>

lemma *find-consistent-signs-at-roots-1-R*:

fixes p :: *real poly*
fixes qs :: *real poly list*
shows $(p \neq 0 \wedge \text{length } qs > 0) \longrightarrow \text{set}(\text{find-consistent-signs-at-roots-R } p \text{ } qs) = \text{set}(\text{characterize-consistent-signs-at-roots } p \text{ } qs)$
 <proof>

lemma *find-consistent-signs-at-roots-0-R*:

fixes p :: *real poly*
assumes $p \neq 0$
shows $\text{set}(\text{find-consistent-signs-at-roots-R } p \text{ } []) = \text{set}(\text{characterize-consistent-signs-at-roots } p \text{ } [])$
 <proof>

```

lemma find-consistent-signs-at-roots-R:
  fixes p:: real poly
  fixes qs :: real poly list
  assumes  $p \neq 0$ 
  shows  $\text{set}(\text{find-consistent-signs-at-roots-R } p \text{ } qs) = \text{set}(\text{characterize-consistent-signs-at-roots } p \text{ } qs)$ 
   $\langle \text{proof} \rangle$ 

end
theory Renegar-Decision
  imports Renegar-Proofs
           BKR-Decision
begin

```

28 Algorithm

```

definition consistent-sign-vectors-R::real poly list  $\Rightarrow$  real set  $\Rightarrow$  rat list set
  where  $\text{consistent-sign-vectors-R } qs \text{ } S = (\text{consistent-sign-vec } qs) \text{ ' } S$ 

```

```

primrec prod-list-var:: ('a::idom) list  $\Rightarrow$  ('a::idom)
  where  $\text{prod-list-var } [] = 1$ 
  |  $\text{prod-list-var } (h\#T) = (\text{if } h = 0 \text{ then } (\text{prod-list-var } T) \text{ else } (h * \text{prod-list-var } T))$ 

```

```

primrec check-all-const-deg:: real poly list  $\Rightarrow$  bool
  where  $\text{check-all-const-deg } [] = \text{True}$ 
  |  $\text{check-all-const-deg } (h\#T) = (\text{if degree } h = 0 \text{ then } (\text{check-all-const-deg } T) \text{ else } \text{False})$ 

```

```

definition poly-f :: real poly list  $\Rightarrow$  real poly
  where
     $\text{poly-f } ps =$ 
     $(\text{if } (\text{check-all-const-deg } ps) = \text{True} \text{ then } [:0, 1:] \text{ else}$ 
     $(\text{pderiv } (\text{prod-list-var } ps)) * (\text{prod-list-var } ps) * ([:-(\text{crb } (\text{prod-list-var } ps)), 1:] *$ 
     $([:(\text{crb } (\text{prod-list-var } ps)), 1:])))$ 

```

```

definition find-consistent-signs-R :: real poly list  $\Rightarrow$  rat list list
  where
     $\text{find-consistent-signs-R } ps = \text{find-consistent-signs-at-roots-R } (\text{poly-f } ps) \text{ } ps$ 

```

```

definition decide-universal-R :: real poly fml  $\Rightarrow$  bool
  where [code]:
     $\text{decide-universal-R } fml = ($ 
     $\text{let } (\text{fml-struct}, \text{polys}) = \text{convert } fml;$ 
     $\text{conds} = \text{find-consistent-signs-R } \text{polys}$ 
    in
     $\text{list-all } (\text{lookup-sem } \text{fml-struct}) \text{ } \text{conds}$ 
     $)$ 

```

definition *decide-existential-R* :: real poly fml \Rightarrow bool
where [code]:
decide-existential-R fml = (
 let (fml-struct, polys) = convert fml;
 conds = find-consistent-signs-R polys
 in
 find (lookup-sem fml-struct) conds \neq None
)

28.1 Proofs

definition *roots-of-poly-f*:: real poly list \Rightarrow real set
where *roots-of-poly-f* qs = {x. poly (poly-f qs) x = 0}

lemma *prod-list-var-nonzero*:
shows prod-list-var qs \neq 0
 <proof>

lemma *q-dvd-prod-list-var-prop*:
assumes q \in set qs
assumes q \neq 0
shows q dvd prod-list-var qs <proof>

lemma *check-all-const-deg-prop*:
shows check-all-const-deg l = True \longleftrightarrow ($\forall p \in$ set(l). degree p = 0)
 <proof>

lemma *poly-f-nonzero*:
fixes qs :: real poly list
shows (poly-f qs) \neq 0
 <proof>

lemma *poly-f-roots-prop-1*:
fixes qs:: real poly list
assumes non-const: check-all-const-deg qs = False
shows $\forall x1. \forall x2. ((x1 < x2 \wedge (\exists q1 \in$ set (qs). q1 \neq 0 \wedge (poly q1 x1) = 0) \wedge
 ($\exists q2 \in$ set(qs). q2 \neq 0 \wedge (poly q2 x2) = 0)) \longrightarrow ($\exists q. x1 < q \wedge q < x2 \wedge$ poly
 (poly-f qs) q = 0))
 <proof>

lemma *main-step-aux1-R*:
fixes qs:: real poly list
assumes non-const: check-all-const-deg qs = True
shows set (find-consistent-signs-R qs) = consistent-sign-vectors-R qs UNIV
 <proof>

lemma *sorted-list-lemma-var*:

fixes l :: *real list*
fixes x :: *real*
assumes $\text{length } l > 1$
assumes $\text{strict-sort: sorted-wrt } (<) l$
assumes $x\text{-not-in: } \neg (\text{List.member } l x)$
assumes $lt\text{-a: } x > (l ! 0)$
assumes $b\text{-lt: } x < (l ! (\text{length } l - 1))$
shows $(\exists n. n < \text{length } l - 1 \wedge x > l ! n \wedge x < l !(n+1))$ *<proof>*

lemma *all-sample-points-prop*:
assumes $\text{is-not-const: check-all-const-deg } qs = \text{False}$
assumes $s\text{-is: } S = (\text{characterize-root-list-p } (pderiv (\text{prod-list-var } qs) * (\text{prod-list-var } qs)) * ([:-(\text{crb } (\text{prod-list-var } qs)),1:] * ([:(\text{crb } (\text{prod-list-var } qs)),1:])))$
shows $\text{consistent-sign-vectors-R } qs \text{ UNIV} = \text{consistent-sign-vectors-R } qs (\text{set } S)$
<proof>

lemma *main-step-aux2-R*:
fixes qs :: *real poly list*
assumes $\text{is-not-const: check-all-const-deg } qs = \text{False}$
shows $\text{set } (\text{find-consistent-signs-R } qs) = \text{consistent-sign-vectors-R } qs \text{ UNIV}$
<proof>

lemma *main-step-R*:
fixes qs :: *real poly list*
shows $\text{set } (\text{find-consistent-signs-R } qs) = \text{consistent-sign-vectors-R } qs \text{ UNIV}$
<proof>

lemma *consistent-sign-vec-semantics-R*:
assumes $\bigwedge i. i \in \text{set-fml } fml \implies i < \text{length } ls$
shows $\text{lookup-sem } fml (\text{map } (\lambda p. \text{poly } p x) ls) = \text{lookup-sem } fml (\text{consistent-sign-vec } ls x)$
<proof>

lemma *universal-lookup-sem-R*:
assumes $\bigwedge i. i \in \text{set-fml } fml \implies i < \text{length } qs$
assumes $\text{set signs} = \text{consistent-sign-vectors-R } qs \text{ UNIV}$
shows $(\forall x::\text{real}. \text{lookup-sem } fml (\text{map } (\lambda p. \text{poly } p x) qs)) \longleftrightarrow \text{list-all } (\text{lookup-sem } fml) \text{ signs}$
<proof>

lemma *existential-lookup-sem-R*:
assumes $\bigwedge i. i \in \text{set-fml } fml \implies i < \text{length } qs$
assumes $\text{set signs} = \text{consistent-sign-vectors-R } qs \text{ UNIV}$
shows $(\exists x::\text{real}. \text{lookup-sem } fml (\text{map } (\lambda p. \text{poly } p x) qs)) \longleftrightarrow \text{find } (\text{lookup-sem } fml) \text{ signs} \neq \text{None}$
<proof>

lemma *decide-univ-lem-helper-R:*

fixes *fml:: real poly fml*

assumes $(fml\text{-struct}, polys) = \text{convert } fml$

shows $(\forall x::real. \text{lookup-sem } fml\text{-struct } (\text{map } (\lambda p. \text{poly } p \ x) \ polys)) \longleftrightarrow (\text{decide-universal-R } fml)$

<proof>

lemma *decide-exis-lem-helper-R:*

fixes *fml:: real poly fml*

assumes $(fml\text{-struct}, polys) = \text{convert } fml$

shows $(\exists x::real. \text{lookup-sem } fml\text{-struct } (\text{map } (\lambda p. \text{poly } p \ x) \ polys)) \longleftrightarrow (\text{decide-existential-R } fml)$

<proof>

lemma *convert-semantics-lem-R:*

assumes $\bigwedge p. p \in \text{set } (\text{poly-list } fml) \implies$

$ls \ ! \ (\text{index-of } ps \ p) = \text{poly } p \ x$

shows $\text{real-sem } fml \ x = \text{lookup-sem } (\text{map-fml } (\text{index-of } ps) \ fml) \ ls$

<proof>

lemma *convert-semantics-R:*

shows $\text{real-sem } fml \ x = \text{lookup-sem } (\text{fst } (\text{convert } fml)) (\text{map } (\lambda p. \text{poly } p \ x) (\text{snd } (\text{convert } fml)))$

<proof>

theorem *decision-procedure-R:*

shows $(\forall x::real. \text{real-sem } fml \ x) \longleftrightarrow (\text{decide-universal-R } fml)$

$\exists x::real. \text{real-sem } fml \ x \longleftrightarrow (\text{decide-existential-R } fml)$

<proof>

end

Acknowledgements

This material is based upon work supported by the National Science Foundation Graduate Research Fellowship under Grants Nos. DGE1252522 and DGE1745016. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the National Science Foundation. This research was also sponsored by the National Science Foundation under Grant No. CNS-1739629, the AFOSR under grant number FA9550-16-1-0288, and A*STAR, Singapore. The views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of any sponsoring institution, the U.S. government or any other entity.

References

- [1] 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.
- [2] 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.