

The BKR Decision Procedure for Univariate Real Arithmetic

Katherine Cordwell, Yong Kiam Tan, and André Platzer

February 6, 2026

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	8
3	Results on Invertibility	26
4	Setup	49
5	Base Case	49
6	Smashing	50
7	Reduction	51

8 Overall algorithm	51
9 Results with Sturm's Theorem	52
10 Setting up the construction: Definitions	55
11 Setting up the construction: Proofs	58
12 Base Case	73
13 Inductive Step	75
13.1 Lemmas on smashing subsets and signs	75
13.2 Well-defined subsets preserved when smashing	76
13.3 Well def signs preserved when smashing	77
13.4 Distinct signs preserved when smashing	77
13.5 Consistent sign assignments preserved when smashing	78
13.6 Main Results	80
14 Reduction Step Proofs	84
14.1 Showing sign conditions preserved when reducing	85
14.2 Showing matrix equation preserved when reducing	94
14.3 Showing matrix preserved	96
14.4 Showing invertibility preserved when reducing	102
14.5 Well def signs preserved when reducing	108
14.6 Distinct signs preserved when reducing	109
14.7 Well def subsets preserved when reducing	110
15 Overall Lemmas	112
15.1 Key properties preserved	112
15.1.1 Properties preserved when combining and reducing systems	112
15.1.2 For length 1 qs	115
15.1.3 For arbitrary qs	116
15.2 Some key results on consistent sign assignments	119
16 Algorithm	126
16.1 Parsing	126
16.2 Factoring	127
16.3 Auxiliary Polynomial	128
16.4 Setting Up the Procedure	128
16.5 Deciding Univariate Problems	129

17 Proofs	129
17.1 Parsing and Semantics	129
17.2 Factoring Lemmas	131
17.3 Auxiliary Polynomial Lemmas	141
17.4 Setting Up the Procedure: Lemmas	146
17.5 Decision Procedure: Main Lemmas	170
18 Base Case	171
19 Smashing	171
20 Reduction	172
21 Overall algorithm	172
22 Tarski Queries Changed	173
23 Matrix Equation	174
24 Base Case	195
25 Inductive Step	197
25.1 Lemmas on smashing subsets	197
25.2 Well-defined subsets preserved when smashing	198
25.3 Consistent Sign Assignments Preserved When Smashing	199
25.4 Main Results	201
26 Reduction Step Proofs	204
26.1 Showing sign conditions preserved when reducing	205
26.2 Showing matrix equation preserved when reducing	212
26.3 Showing matrix preserved	214
26.4 Showing invertibility preserved when reducing	220
26.5 Well def signs preserved when reducing	227
26.6 Distinct signs preserved when reducing	227
26.7 Well def subsets preserved when reducing	228
27 Overall Lemmas	229
27.1 Key properties preserved	230
27.1.1 Properties preserved when combining and reducing systems	230
27.1.2 For length 1 qs	233
27.1.3 For arbitrary qs	233
27.2 Some key results on consistent sign assignments	235

28 Algorithm	241
28.1 Proofs	242

```

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 *kroncker-product* :: 'a :: ring mat \Rightarrow 'a mat \Rightarrow 'a mat **where**

```

  kroncker-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 –

```

have b*d+c < b*(d+1)
  by (simp add: assms(2))
thus ?thesis

```

by (metis One-nat-def Suc-leI add.right-neutral add-Suc-right assms(1) less-le-trans mult.commute mult-le-cancel2)

qed

lemma *dim-kroncker[simp]*:

```

  dim-row (kroncker-product A B) = dim-row A * dim-row B
  dim-col (kroncker-product A B) = dim-col A * dim-col B
unfolding kroncker-product-def Let-def by auto

```

lemma *kroncker-inverse-index*:

```

assumes r < dim-row A s < dim-col A
assumes v < dim-row B w < dim-col B
shows kroncker-product A B $$ (dim-row B*r+v, dim-col B*s+w) = A $$ (r,s)
  * B $$ (v,w)

```

proof –

```

from arith[OF assms(1) assms(3)]
have dim-row B*r+v < dim-row A * dim-row B .
moreover from arith[OF assms(2) assms(4)]
have dim-col B * s + w < dim-col A * dim-col B .

```

ultimately show *?thesis*
unfolding *kronecker-product-def Let-def*
using *assms* **by** *auto*
qed

lemma *kronecker-distr-left*:
assumes *dim-row B = dim-row C dim-col B = dim-col C*
shows *kronecker-product A (B+C) = kronecker-product A B + kronecker-product A C*
unfolding *kronecker-product-def Let-def*
using *assms* **apply** (*auto simp add: mat-eq-iff*)
by (*metis (no-types, lifting) distrib-left index-add-mat(1) mod-less-divisor mult-eq-0-iff neq0-conv not-less-zero*)

lemma *kronecker-distr-right*:
assumes *dim-row B = dim-row C dim-col B = dim-col C*
shows *kronecker-product (B+C) A = kronecker-product B A + kronecker-product C A*
unfolding *kronecker-product-def Let-def*
using *assms* **by** (*auto simp add: mat-eq-iff less-mult-imp-div-less distrib-right*)

lemma *index-mat-mod[simp]*: $nr > 0 \ \& \ nc > 0 \implies \text{mat } nr \ nc \ f \ \S\S \ (i \bmod nr, j \bmod nc) = f \ (i \bmod nr, j \bmod nc)$
by *auto*

lemma *kronecker-assoc*:
shows *kronecker-product A (kronecker-product B C) = kronecker-product (kronecker-product A B) C*
unfolding *kronecker-product-def Let-def*
apply (*case-tac dim-row B * dim-row C > 0 & dim-col B * dim-col C > 0*)
apply (*auto simp add: mat-eq-iff less-mult-imp-div-less*)
using *Rings.ordered-semiring-strict-class.mult-pos-pos*
by (*smt (verit, best) div-less-iff-less-mult div-mult2-eq kronecker-inverse-index mod-less-divisor mod-mult2-eq mult.assoc mult commute mult-div-mod-eq*)

lemma *sum-sum-mod-div*:
 $(\sum ia = 0..nat..<x. \sum ja = 0..<y. f ia ja) =$
 $(\sum ia = 0..<x*y. f (ia \text{ div } y) (ia \bmod y))$
proof –
have *1: inj-on ($\lambda ia. (ia \text{ div } y, ia \bmod y)$) $\{0..<x * y\}$*
by (*smt (verit, best) Pair-inject div-mod-decomp inj-onI*)
have *21: $\{0..<x\} \times \{0..<y\} \subseteq (\lambda ia. (ia \text{ div } y, ia \bmod y)) \text{ ` } \{0..<x * y\}$*
proof *clarsimp*
fix *a b*
assume **: a < x b < y*
have *a * y + b $\in \{0..<x*y\}$*
by (*metis arith * atLeastLessThan-iff le0 mult commute*)
thus *(a, b) $\in (\lambda ia. (ia \text{ div } y, ia \bmod y)) \text{ ` } \{0..<x * y\}$*
using *** **by** (*auto simp add: image-iff*)

(metis ‹ $a * y + b \in \{0..<x * y\}$ › add.commute add.right-neutral div-less
div-mult-self1 less-zeroE mod-eq-self-iff-div-eq-0 mod-mult-self1)

qed

have 22:($\lambda ia. (ia \text{ div } y, ia \text{ mod } y)$) ‘ $\{0..<x * y\} \subseteq \{0..<x\} \times \{0..<y\}$ ’

using less-mult-imp-div-less **apply** auto

by (metis mod-less-divisor mult.commute neq0-conv not-less-zero)

have 2: $\{0..<x\} \times \{0..<y\} = (\lambda ia. (ia \text{ div } y, ia \text{ mod } y))$ ‘ $\{0..<x * y\}$ ’

using 21 22 **by** auto

have *: $(\sum ia = 0::\text{nat}..<x. \sum ja = 0..<y. f ia ja) =$

$(\sum (x, y) \in \{0..<x\} \times \{0..<y\}. f x y)$

by (auto simp add: sum.cartesian-product)

show ?thesis **unfolding** *

apply (intro sum.reindex-cong[of $\lambda ia. (ia \text{ div } y, ia \text{ mod } y)$])

using 1 2 **by** auto

qed

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)

unfolding kronecker-product-def Let-def mat-eq-iff

proof clarsimp

fix i j

assume ij: $i < \text{dim-row } A * \text{dim-row } B$ $j < \text{dim-col } C * \text{dim-col } D$

have 1: $(A * C) \$\$ (i \text{ div } \text{dim-row } B, j \text{ div } \text{dim-col } D) =$

$\text{row } A (i \text{ div } \text{dim-row } B) \cdot \text{col } C (j \text{ div } \text{dim-col } D)$

using ij less-mult-imp-div-less **by** (auto intro!: index-mult-mat)

have 2: $(B * D) \$\$ (i \text{ mod } \text{dim-row } B, j \text{ mod } \text{dim-col } D) =$

$\text{row } B (i \text{ mod } \text{dim-row } B) \cdot \text{col } D (j \text{ mod } \text{dim-col } D)$

using ij **apply** (auto intro!: index-mult-mat)

using gr-implies-not0 **apply** fastforce

using gr-implies-not0 **by** fastforce

have 3: $\bigwedge x. x < \text{dim-row } C * \text{dim-row } D \implies$

$A \$\$ (i \text{ div } \text{dim-row } B, x \text{ div } \text{dim-row } D) *$

$B \$\$ (i \text{ mod } \text{dim-row } B, x \text{ mod } \text{dim-row } D) *$

$(C \$\$ (x \text{ div } \text{dim-row } D, j \text{ div } \text{dim-col } D) *$

$D \$\$ (x \text{ mod } \text{dim-row } D, j \text{ mod } \text{dim-col } D)) =$

$\text{row } A (i \text{ div } \text{dim-row } B) \$ (x \text{ div } \text{dim-row } D) *$

$\text{col } C (j \text{ div } \text{dim-col } D) \$ (x \text{ div } \text{dim-row } D) *$

$(\text{row } B (i \text{ mod } \text{dim-row } B) \$ (x \text{ mod } \text{dim-row } D) *$

$\text{col } D (j \text{ mod } \text{dim-col } D) \$ (x \text{ mod } \text{dim-row } D))$

proof –

fix x

assume *: $x < \text{dim-row } C * \text{dim-row } D$

have 1: $\text{row } A (i \text{ div } \text{dim-row } B) \$ (x \text{ div } \text{dim-row } D) = A \$\$ (i \text{ div } \text{dim-row } B, x \text{ div } \text{dim-row } D)$

by (simp add: * assms(1) less-mult-imp-div-less row-def)

have 2: row B ($i \bmod \dim\text{-row } B$) $\$$ ($x \bmod \dim\text{-row } D$) = B $\$\$$ ($i \bmod \dim\text{-row } B$, $x \bmod \dim\text{-row } D$)
by (*metis* * *assms*(2) *ij*(1) *index-row*(1) *mod-less-divisor* *nat-0-less-mult-iff* *neq0-conv* *not-less-zero*)
have 3: col C ($j \bmod \dim\text{-col } D$) $\$$ ($x \bmod \dim\text{-row } D$) = C $\$\$$ ($x \bmod \dim\text{-row } D$, $j \bmod \dim\text{-col } D$)
by (*simp* *add*: * *ij*(2) *less-mult-imp-div-less*)
have 4: col D ($j \bmod \dim\text{-col } D$) $\$$ ($x \bmod \dim\text{-row } D$) = D $\$\$$ ($x \bmod \dim\text{-row } D$, $j \bmod \dim\text{-col } D$)
by (*metis* * *bot-nat-0*.*not-eq-extremum* *ij*(2) *index-col* *mod-less-divisor* *mult-zero-right* *not-less-zero*)
show A $\$\$$ ($i \bmod \dim\text{-row } B$, $x \bmod \dim\text{-row } D$) *
 B $\$\$$ ($i \bmod \dim\text{-row } B$, $x \bmod \dim\text{-row } D$) *
 $(C$ $\$\$$ ($x \bmod \dim\text{-row } D$, $j \bmod \dim\text{-col } D$) *
 D $\$\$$ ($x \bmod \dim\text{-row } D$, $j \bmod \dim\text{-col } D$)) =
row A ($i \bmod \dim\text{-row } B$) $\$$ ($x \bmod \dim\text{-row } D$) *
col C ($j \bmod \dim\text{-col } D$) $\$$ ($x \bmod \dim\text{-row } D$) *
(row B ($i \bmod \dim\text{-row } B$) $\$$ ($x \bmod \dim\text{-row } D$) *
col D ($j \bmod \dim\text{-col } D$) $\$$ ($x \bmod \dim\text{-row } D$)) **unfolding** 1 2 3 4
by (*simp* *add*: *mult.assoc* *mult.left-commute*)
qed
have *: $(A * C)$ $\$\$$ ($i \bmod \dim\text{-row } B$, $j \bmod \dim\text{-col } D$) *
 $(B * D)$ $\$\$$ ($i \bmod \dim\text{-row } B$, $j \bmod \dim\text{-col } D$) =
 $(\sum ia = 0..<\dim\text{-row } C * \dim\text{-row } D.$
 A $\$\$$ ($i \bmod \dim\text{-row } B$, $ia \bmod \dim\text{-row } D$) *
 B $\$\$$ ($i \bmod \dim\text{-row } B$, $ia \bmod \dim\text{-row } D$) *
 $(C$ $\$\$$ ($ia \bmod \dim\text{-row } D$, $j \bmod \dim\text{-col } D$) *
 D $\$\$$ ($ia \bmod \dim\text{-row } D$, $j \bmod \dim\text{-col } D$)))
unfolding 1 2 *scalar-prod-def* *sum-product* *sum-sum-mod-div*
apply (*auto* *simp* *add*: *sum-product* *sum-sum-mod-div* *intro!*: *sum.cong*)
using 3 **by** *presburger*
show *vec* ($\dim\text{-col } A * \dim\text{-col } B$)
 $(\lambda j. A$ $\$\$$ ($i \bmod \dim\text{-row } B$, $j \bmod \dim\text{-col } B$) *
 B $\$\$$ ($i \bmod \dim\text{-row } B$, $j \bmod \dim\text{-col } B$)) *
 $\text{vec}(\dim\text{-row } C * \dim\text{-row } D)$
 $(\lambda i. C$ $\$\$$ ($i \bmod \dim\text{-row } D$, $j \bmod \dim\text{-col } D$) *
 D $\$\$$ ($i \bmod \dim\text{-row } D$, $j \bmod \dim\text{-col } D$)) =
 $(A * C)$ $\$\$$ ($i \bmod \dim\text{-row } B$, $j \bmod \dim\text{-col } D$) *
 $(B * D)$ $\$\$$ ($i \bmod \dim\text{-row } B$, $j \bmod \dim\text{-col } D$)
unfolding * *scalar-prod-def*
by (*auto* *simp* *add*: *assms* *sum-product* *sum-sum-mod-div* *intro!*: *sum.cong*)
qed

lemma *inverts-mat-length*:

assumes *square-mat* A *inverts-mat* A B *inverts-mat* B A
shows $\dim\text{-row } B = \dim\text{-row } A$ $\dim\text{-col } B = \dim\text{-col } A$
apply (*metis* *assms*(1) *assms*(3) *index-mult-mat*(3) *index-one-mat*(3) *inverts-mat-def* *square-mat.simps*)
by (*metis* *assms*(1) *assms*(2) *index-mult-mat*(3) *index-one-mat*(3) *inverts-mat-def*)

square-mat.simps)

lemma *less-mult-imp-mod-less*:
*m mod i < i if m < n * i for m n i :: nat*
using *gr-implies-not-zero* **that by** *fastforce*

lemma *kronecker-one*:
shows *kronecker-product* ((1_m x)::'a :: *ring-1 mat*) (1_m y) = 1_m ($x*y$)
unfolding *kronecker-product-def* *Let-def*
apply (*auto simp add:mat-eq-iff less-mult-imp-div-less less-mult-imp-mod-less*)
by (*metis div-mult-mod-eq*)

lemma *kronecker-invertible*:
assumes *invertible-mat* (A :: 'a :: *comm-ring-1 mat*) *invertible-mat* B
shows *invertible-mat* (*kronecker-product* A B)
proof –
obtain A_i **where** A_i : *inverts-mat* A A_i *inverts-mat* A_i A **using** *assms invertible-mat-def* **by** *blast*
obtain B_i **where** B_i : *inverts-mat* B B_i *inverts-mat* B_i B **using** *assms invertible-mat-def* **by** *blast*
have *square-mat* (*kronecker-product* A B)
by (*metis (no-types, lifting) assms(1) assms(2) dim-col-mat(1) dim-row-mat(1) invertible-mat-def kronecker-product-def square-mat.simps*)
moreover have *inverts-mat* (*kronecker-product* A B) (*kronecker-product* A_i B_i)
using A_i B_i **unfolding** *inverts-mat-def*
by (*metis (no-types, lifting) dim-kronecker(1) index-mult-mat(3) index-one-mat(3) kronecker-of-mult kronecker-one*)
moreover have *inverts-mat* (*kronecker-product* A_i B_i) (*kronecker-product* A B)
using A_i B_i **unfolding** *inverts-mat-def*
by (*metis (no-types, lifting) dim-kronecker(1) index-mult-mat(3) index-one-mat(3) kronecker-of-mult kronecker-one*)
ultimately show *?thesis* **unfolding** *invertible-mat-def* **by** *blast*
qed

2 More DL Rank

instantiation *mat* :: (*conjugate*) *conjugate*
begin

definition *conjugate-mat* :: 'a :: *conjugate mat* \Rightarrow 'a *mat*
where *conjugate* m = *mat* (*dim-row* m) (*dim-col* m) ($\lambda(i,j). \text{conjugate } (m \text{ } \$\$ (i,j))$)

lemma *dim-row-conjugate[simp]*: *dim-row* (*conjugate* m) = *dim-row* m
unfolding *conjugate-mat-def* **by** *auto*

lemma *dim-col-conjugate[simp]*: *dim-col* (*conjugate* m) = *dim-col* m
unfolding *conjugate-mat-def* **by** *auto*

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

by (*auto*)

lemma *mat-index-conjugate[simp]*:

shows $i < \text{dim-row } m \implies j < \text{dim-col } m \implies \text{conjugate } m \ \$\$ (i,j) = \text{conjugate } (m \ \$\$ (i,j))$

unfolding *conjugate-mat-def* **by** *auto*

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

by (*auto*)

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

by (*auto*)

lemma *rows-conjugate*: $\text{rows } (\text{conjugate } m) = \text{map } \text{conjugate } (\text{rows } m)$

by (*simp add: list-eq-iff-nth-eq*)

lemma *cols-conjugate*: $\text{cols } (\text{conjugate } m) = \text{map } \text{conjugate } (\text{cols } m)$

by (*simp add: list-eq-iff-nth-eq*)

instance

proof

fix $a \ b :: 'a \ \text{mat}$

show $\text{conjugate } (\text{conjugate } a) = a$

unfolding *mat-eq-iff* **by** *auto*

let $?a = \text{conjugate } a$

let $?b = \text{conjugate } b$

show $\text{conjugate } a = \text{conjugate } b \longleftrightarrow a = b$

by (*metis dim-col-conjugate dim-row-conjugate mat-index-conjugate conjugate-cancel-iff mat-eq-iff*)

qed

end

abbreviation *conjugate-transpose* :: $'a :: \text{conjugate } \text{mat} \Rightarrow 'a \ \text{mat}$

where *conjugate-transpose* $A \equiv \text{conjugate } (A^T)$

notation *conjugate-transpose* ($\langle (-^H) \rangle$ [1000])

lemma *transpose-conjugate*:

shows $(\text{conjugate } A)^T = A^H$

unfolding *conjugate-mat-def*

by *auto*

lemma *vec-module-col-helper*:

fixes $A :: ('a :: \text{field}) \ \text{mat}$

shows $(0_v \text{ (dim-row } A) \in \text{LinearCombinations.module.span class-ring } (\!| \text{carrier} = \text{carrier-vec (dim-row } A), \text{mult} = \text{undefined, one} = \text{undefined, zero} = 0_v \text{ (dim-row } A), \text{add} = (+), \text{smult} = (\cdot_v)\!|) \text{ (set (cols } A)))$
proof –
have $\forall v. (0::'a) \cdot_v v + v = v$
by *auto*
then show $0_v \text{ (dim-row } A) \in \text{LinearCombinations.module.span class-ring } (\!| \text{carrier} = \text{carrier-vec (dim-row } A), \text{mult} = \text{undefined, one} = \text{undefined, zero} = 0_v \text{ (dim-row } A), \text{add} = (+), \text{smult} = (\cdot_v)\!|) \text{ (set (cols } A))$
by (*metis cols-dim module-vec-def right-zero-vec smult-carrier-vec vec-space.prod-in-span zero-carrier-vec*)
qed

lemma *vec-module-col-helper2*:

fixes $A:: ('a :: \text{field}) \text{ mat}$
shows $\bigwedge a x. x \in \text{LinearCombinations.module.span class-ring } (\!| \text{carrier} = \text{carrier-vec (dim-row } A), \text{mult} = \text{undefined, one} = \text{undefined, zero} = 0_v \text{ (dim-row } A), \text{add} = (+), \text{smult} = (\cdot_v)\!|) \text{ (set (cols } A)) \implies$
 $(\bigwedge a b v. (a + b) \cdot_v v = a \cdot_v v + b \cdot_v v) \implies$
 $a \cdot_v x$
 $\in \text{LinearCombinations.module.span class-ring } (\!| \text{carrier} = \text{carrier-vec (dim-row } A), \text{mult} = \text{undefined, one} = \text{undefined, zero} = 0_v \text{ (dim-row } A), \text{add} = (+), \text{smult} = (\cdot_v)\!|) \text{ (set (cols } A))$

proof –
fix $a :: 'a$ **and** $x :: 'a \text{ vec}$
assume $x \in \text{LinearCombinations.module.span class-ring } (\!| \text{carrier} = \text{carrier-vec (dim-row } A), \text{mult} = \text{undefined, one} = \text{undefined, zero} = 0_v \text{ (dim-row } A), \text{add} = (+), \text{smult} = (\cdot_v)\!|) \text{ (set (cols } A))$
then show $a \cdot_v x \in \text{LinearCombinations.module.span class-ring } (\!| \text{carrier} = \text{carrier-vec (dim-row } A), \text{mult} = \text{undefined, one} = \text{undefined, zero} = 0_v \text{ (dim-row } A), \text{add} = (+), \text{smult} = (\cdot_v)\!|) \text{ (set (cols } A))$
by (*metis (full-types) cols-dim idom-vec.smult-in-span module-vec-def*)
qed

lemma *vec-module-col*: $\text{module (class-ring } :: 'a :: \text{field ring})$

$(\text{module-vec TYPE('a)}$
 $(\text{dim-row } A)$
 $(\!| \text{carrier} :=$
 $\text{LinearCombinations.module.span}$
 $\text{class-ring (module-vec TYPE('a) (dim-row } A)) \text{ (set (cols } A))\!|)$

proof –
interpret *abelian-group module-vec TYPE('a) (dim-row A)*
 $(\!| \text{carrier} :=$
 $\text{LinearCombinations.module.span}$
 $\text{class-ring (module-vec TYPE('a) (dim-row } A)) \text{ (set (cols } A))\!|)$
apply (*unfold-locales*)
apply (*auto simp add:module-vec-def*)

```

apply (metis cols-dim module-vec-def partial-object.select-convs(1) ring.simps(2)
vec-vs vectorspace.span-add1)
apply (metis assoc-add-vec cols-dim module-vec-def vec-space.cV vec-vs
vectorspace.span-closed)
using vec-module-col-helper[of A] apply (auto)
apply (metis cols-dim left-zero-vec module-vec-def partial-object.select-convs(1)
vec-vs vectorspace.span-closed)
apply (metis cols-dim module-vec-def partial-object.select-convs(1) right-zero-vec
vec-vs vectorspace.span-closed)
apply (metis cols-dim comm-add-vec module-vec-def vec-space.cV vec-vs vec-
torspace.span-closed)
unfolding Units-def apply auto
by (metis (no-types, opaque-lifting) cols-dim comm-add-vec module-vec-def par-
tial-object.select-convs(1) uminus-l-inv-vec vec-space.vec-neg vec-vs vectorspace.span-closed
vectorspace.span-neg)
show ?thesis
apply (unfold-locales)
unfolding class-ring-simps apply auto
unfolding module-vec-simps using add-smult-distrib-vec apply auto
apply (auto simp add:module-vec-def)
using vec-module-col-helper2
apply blast
by (smt (verit) cols-dim module-vec-def smult-add-distrib-vec vec-space.cV vec-vs
vectorspace.span-closed)
qed

```

```

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))))
unfolding vectorspace-def
using vec-module-col class-field
by (auto simp: class-field-def)

```

```

lemma cols-mat-mul-map:
shows cols (A * B) = map ((*v) A) (cols B)
unfolding list-eq-iff-nth-eq
by auto

```

```

lemma cols-mat-mul:
shows set (cols (A * B)) = (*v) A ‘ set (cols B)
by (simp add: cols-mat-mul-map)

```

```

lemma set-obtain-sublist:

```

assumes $S \subseteq \text{set } ls$
obtains ss **where** $\text{distinct } ss \ S = \text{set } ss$
using *assms finite-distinct-list infinite-super* **by** *blast*

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 $A * (\text{mat-of-cols } n \ cs) = \text{mat-of-cols } nr \ (\text{map } ((*_v) \ A) \ cs)$
unfolding *mat-eq-iff*
using *assms apply auto*
apply (*subst mat-of-cols-index*)
by *auto*

lemma *helper*:

fixes $x \ y \ z :: 'a :: \{\text{conjugatable-ring}, \text{comm-ring}\}$
shows $x * (y * z) = y * x * z$
by (*simp add: mult.assoc mult.left-commute*)

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 $x \cdot c \ (A *_v \ y) = (A^H *_v \ x) \cdot c \ y$
unfolding *mult-mat-vec-def scalar-prod-def*
using *assms apply (auto simp add: sum-distrib-left sum-distrib-right sum-conjugate conjugate-dist-mul)*
apply (*subst sum.swap*)
by (*meson helper mult.assoc mult.left-commute sum.cong*)

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

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

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 –

have $(A^H *_v \ v) \cdot c \ (A^H *_v \ v) = (A *_v \ (A^H *_v \ v)) \cdot c \ v$

by (*metis (mono-tags, lifting) Matrix.carrier-vec-conjugate assms(1) assms(2) assms(3) carrier-matD(2) conjugate-zero-vec cscalar-prod-conjugate-transpose dim-row-conjugate index-transpose-mat(2) mult-mat-vec-def scalar-prod-left-zero scalar-prod-right-zero vec-carrier*)

also have $\dots = 0$

by (*simp add: assms(2) assms(3)*)

ultimately have $(A^H *_v \ v) \cdot c \ (A^H *_v \ v) = 0$ **by** *auto*

thus *?thesis*

apply (*subst conjugate-square-eq-0-vec[symmetric]*)

```

    using assms(1) carrier-dim-vec apply fastforce
    by auto
qed

lemma row-mat-of-cols:
  assumes  $i < nr$ 
  shows  $row (mat-of-cols\ nr\ ls)\ i = vec (length\ ls)\ (\lambda j. (ls\ !\ j)\ \$i)$ 
  by (simp add: assms mat-of-cols-index vec-eq-iff)

lemma mat-of-cols-cons-mat-vec:
  fixes  $v :: 'a::comm-ring\ vec$ 
  assumes  $v \in carrier-vec (length\ ls)$ 
  assumes  $dim-vec\ a = nr$ 
  shows
     $mat-of-cols\ nr\ (a\ \#\ ls)\ *_v\ (vCons\ m\ v) =$ 
     $m \cdot_v\ a + mat-of-cols\ nr\ ls\ *_v\ v$ 
  unfolding mult-mat-vec-def vec-eq-iff
  using assms by
    (auto simp add: row-mat-of-cols vec-Suc o-def mult.commute)

lemma smult-vec-zero:
  fixes  $v :: 'a::ring\ vec$ 
  shows  $0 \cdot_v\ v = 0_v (dim-vec\ v)$ 
  unfolding smult-vec-def vec-eq-iff
  by (auto)

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
     $mat-of-cols\ nr\ ss\ *_v\ v =$ 
     $mat-of-cols\ nr\ (ls\ @\ ss)\ *_v\ (0_v (length\ ls)\ @_v\ v)$ 
  using assms(2)
proof (induction ls)
  case Nil
  then show ?case by auto
next
  case (Cons  $a\ ls$ )
  then show ?case apply (auto simp add: zero-vec-Suc)
    apply (subst mat-of-cols-cons-mat-vec)
    by (auto simp add: assms smult-vec-zero)
qed

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
 $mat\text{-of}\text{-cols } nr \ (permute\text{-list } f \ ss) \ *_v \ vec\text{-of}\text{-list } (permute\text{-list } f \ v) =$
 $mat\text{-of}\text{-cols } nr \ ss \ *_v \ vec\text{-of}\text{-list } v$
unfolding $mat\text{-of}\text{-cols}\text{-def } mult\text{-mat}\text{-vec}\text{-def } vec\text{-eq}\text{-iff } scalar\text{-prod}\text{-def}$
proof $clarsimp$
fix i
assume $i < nr$
from $sum.permute[OF \ assms(1)]$
have $(\sum ia < length \ ss. \ ss \ ! \ f \ ia \ \$ \ i \ * \ v \ ! \ f \ ia) =$
 $sum \ ((\lambda ia. \ ss \ ! \ f \ ia \ \$ \ i \ * \ v \ ! \ f \ ia) \circ f) \ \{..\ < length \ ss\} .$
also have $... = (\sum ia = 0.. < length \ v. \ ss \ ! \ f \ ia \ \$ \ i \ * \ v \ ! \ f \ ia)$
using $assms(2) \ calculation \ lessThan\text{-atLeast}0$ **by** $auto$
ultimately have $*$: $(\sum ia = 0.. < length \ v.$
 $ss \ ! \ f \ ia \ \$ \ i \ * \ v \ ! \ f \ ia) =$
 $(\sum ia = 0.. < length \ v.$
 $ss \ ! \ ia \ \$ \ i \ * \ v \ ! \ ia)$
by $(metis \ (mono\text{-tags}, \ lifting) \ \langle \bigwedge g. \ sum \ g \ \{..\ < length \ ss\} = sum \ (g \circ f)$
 $\{..\ < length \ ss\} \rangle \ assms(2) \ comp\text{-apply} \ lessThan\text{-atLeast}0 \ sum.cong)$
show $(\sum ia = 0.. < length \ v.$
 $vec \ (length \ ss) \ (\lambda j. \ permute\text{-list } f \ ss \ ! \ j \ \$ \ i) \ \$ \ ia \ *$
 $vec\text{-of}\text{-list } (permute\text{-list } f \ v) \ \$ \ ia) =$
 $(\sum ia = 0.. < length \ v. \ vec \ (length \ ss) \ (\lambda j. \ ss \ ! \ j \ \$ \ i) \ \$ \ ia \ * \ vec\text{-of}\text{-list } v \ \$ \ ia)$
using $assms \ *$ **by** $(auto \ simp \ add: \ permute\text{-list}\text{-nth} \ vec\text{-of}\text{-list}\text{-index})$
qed

lemma $subindex\text{-permutation}$:
assumes $distinct \ ss \ set \ ss \subseteq \{..\ < length \ ls\}$
obtains f **where** f $permutes \ \{..\ < length \ ls\}$
 $permute\text{-list } f \ ls = map \ (!) \ ls \ (filter \ (\lambda i. \ i \notin \ set \ ss) \ [0.. < length \ ls]) \ @ \ map$
 $(!) \ ls \ ss$
proof $-$
have $set \ [0.. < length \ ls] = set \ (filter \ (\lambda i. \ i \notin \ set \ ss) \ [0.. < length \ ls] \ @ \ ss)$
using $assms$ **unfolding** $multiset\text{-eq}\text{-iff}$ **by** $auto$
then have $mset \ [0.. < length \ ls] = mset \ (filter \ (\lambda i. \ i \notin \ set \ ss) \ [0.. < length \ ls] \ @$
 $ss)$
apply $(subst \ set\text{-eq}\text{-iff}\text{-mset}\text{-eq}\text{-distinct}[symmetric])$
using $assms$ **by** $auto$
then have $mset \ ls = mset \ (map \ (!) \ ls$
 $(filter \ (\lambda i. \ i \notin \ set \ ss)$
 $[0.. < length \ ls]) \ @ \ map \ (!) \ ls \ ss)$
by $(metis \ map\text{-append} \ map\text{-nth} \ mset\text{-map})$
thus $?thesis$
by $(metis \ mset\text{-eq}\text{-permutation} \ that)$
qed

lemma $subindex\text{-permutation}2$:
assumes $distinct \ ss \ set \ ss \subseteq \{..\ < length \ ls\}$
obtains f **where** f $permutes \ \{..\ < length \ ls\}$

$ls = \text{permute-list } f \text{ (map (!) } ls \text{ (filter } (\lambda i. i \notin \text{set } ss) [0..<\text{length } ls]) \text{ @ map$
 $(!) \text{ } ls \text{) } ss)$
using *subindex-permutation*
by (*metis* *assms*(1) *assms*(2) *length-permute-list* *mset-eq-permutation* *mset-permute-list*)

lemma *distinct-list-subset-nths*:

assumes *distinct* *ss* *set* *ss* \subseteq *set* *ls*

obtains *ids* **where** *distinct* *ids* *set* *ids* \subseteq $\{..<\text{length } ls\}$ *ss* = *map* (!) *ls* *ids*

proof –

let *?ids* = *map* ($\lambda i. \text{@}j. j < \text{length } ls \wedge ls!j = i$) *ss*

have 1: *distinct* *?ids* **unfolding** *distinct-map*

using *assms* **apply** (*auto simp add: inj-on-def*)

by (*smt* (*verit*) *in-set-conv-nth* *someI* *subset-eq*)

have 2: *set* *?ids* \subseteq $\{..<\text{length } ls\}$

using *assms* **apply** (*auto*)

by (*metis* (*mono-tags*, *lifting*) *in-mono* *in-set-conv-nth* *tfl-some*)

have 3: *ss* = *map* (!) *ls* *?ids*

using *assms* **apply** (*auto simp add: list-eq-iff-nth-eq*)

by (*smt* (*verit*, *best*) *in-set-conv-nth* *someI* *subsetD*)

show (\bigwedge *ids*. *distinct* *ids* \implies

$\text{set } ids \subseteq \{..<\text{length } ls\} \implies$

$ss = \text{map } (!) \text{ } ls \text{ } ids \implies \textit{thesis} \implies$

thesis **using** 1 2 3 **by** *blast*

qed

lemma *helper3*:

fixes *A* $:: 'a::\text{comm-ring}$ *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*

proof –

from *distinct-list-subset-nths*[*OF* *ss*]

obtain *ids* **where** *ids*: *distinct* *ids* *set* *ids* \subseteq $\{..<\text{length } (\text{cols } A)\}$

and *ss*: *ss* = *map* (!) (*cols* *A*) *ids* **by** *blast*

let *?ls* = *map* (!) (*cols* *A*) (*filter* ($\lambda i. i \notin \text{set } ids$) $[0..<\text{length } (\text{cols } A)]$)

from *subindex-permutation2*[*OF* *ids*] **obtain** *f* **where**

f: *f* *permutes* $\{..<\text{length } (\text{cols } A)\}$

cols *A* = *permute-list* *f* (*?ls* @ *ss*) **using** *ss* **by** *blast*

have *: $\bigwedge x. x \in \text{set } ?ls \implies \text{dim-vec } x = nr$

using *A* **by** *auto*

let *?cs1* = (*list-of-vec* (0_v (*length* *?ls*) @_{*v*} *v*))

from *helper2*[*OF* *assms*(4)]

have *mat-of-cols* *nr* *ss* $*_v$ *v* = *mat-of-cols* *nr* (*?ls* @ *ss*) $*_v$ *vec-of-list* (*?cs1*)

using *

by (*metis* *vec-list*)

also have ... = *mat-of-cols* *nr* (*permute-list* *f* (*?ls* @ *ss*)) $*_v$ *vec-of-list* (*permute-list* *f* *?cs1*)

apply (*auto intro!*: *mat-of-cols-mult-mat-vec-permute-list*[*symmetric*])

apply (*metis cols-length* $f(1)$ $f(2)$ *length-append length-map length-permute-list*)
using *assms(4)* **by** *auto*
also have ... = $A *_v \text{vec-of-list (permute-list } f \text{ ?cs1)}$ **using** $f(2)$ *assms* **by** *auto*
ultimately show
 $(\bigwedge c. \text{mat-of-cols } nr \text{ } ss *_v v = A *_v c \implies \text{dim-vec } c = nc \implies \text{thesis}) \implies \text{thesis}$
by (*metis* A *assms(4)* *carrier-matD(2)* *carrier-vecD* *cols-length* *dim-vec-of-list*
 $f(2)$ *index-append-vec(2)* *index-zero-vec(2)* *length-append* *length-list-of-vec* *length-permute-list*)
qed

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

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

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

proof –

have $A^H \in \text{carrier-mat } nc \text{ } nr$ **using** *assms(1)* **by** *auto*
from *helper3[OF this assms(2–4)]*
obtain c **where** $c: \text{mat-of-cols } nc \text{ } ss *_v v = A^H *_v c \text{ } \text{dim-vec } c = nr$ **by** *blast*
have $1: c \in \text{carrier-vec } nr$
using c *carrier-vec-dim-vec* **by** *blast*
have $2: A *_v (A^H *_v c) = 0_v \text{ } nr$ **using** c *assms(5)* **by** *auto*
from *mat-mul-conjugate-transpose-vec-eq-0[OF assms(1) 1 2]*
have $A^H *_v c = 0_v \text{ } nc$.
thus *?thesis* **unfolding** $c(1)[\text{symmetric}]$.

qed

lemma *Units-invertible*:

fixes $A :: 'a :: \text{semiring-1}$ *mat*
assumes $A \in \text{Units (ring-mat TYPE('a) } n \text{ } b)$
shows *invertible-mat* A
using *assms* **unfolding** *Units-def invertible-mat-def*
apply (*auto simp add: ring-mat-def*)
using *inverts-mat-def* **by** *blast*

lemma *invertible-Units*:

fixes $A :: 'a :: \text{semiring-1}$ *mat*
assumes *invertible-mat* A
shows $A \in \text{Units (ring-mat TYPE('a) (dim-row } A) \text{ } b)$
using *assms* **unfolding** *Units-def invertible-mat-def*
apply (*auto simp add: ring-mat-def*)
by (*metis* *assms* *carrier-mat-triv* *invertible-mat-def* *inverts-mat-def* *inverts-mat-length(1)*
inverts-mat-length(2))

lemma *invertible-det*:

fixes $A :: 'a :: \text{field}$ *mat*
assumes $A \in \text{carrier-mat } n \text{ } n$

```

shows invertible-mat  $A \longleftrightarrow \det A \neq 0$ 
apply auto
using invertible-Units unit-imp-det-non-zero apply fastforce
using assms by (auto intro!: Units-invertible det-non-zero-imp-unit)

context vec-space begin

lemma find-indices-distinct:
  assumes distinct ss
  assumes  $i < \text{length } ss$ 
  shows find-indices (ss ! i) ss = [i]
proof -
  have set (find-indices (ss ! i) ss) = {i}
    using assms apply auto by (simp add: assms(1) assms(2) nth-eq-iff-index-eq)
  thus ?thesis
    by (metis distinct.simps(2) distinct-find-indices empty-iff empty-set insert-iff
list.exhaust list.simps(15))
qed

lemma lin-indpt-lin-comb-list:
  assumes distinct ss
  assumes lin-indpt (set ss)
  assumes set ss  $\subseteq$  carrier-vec n
  assumes lincomb-list f ss =  $0_v$  n
  assumes  $i < \text{length } ss$ 
  shows f i = 0
proof -
  from lincomb-list-as-lincomb[OF assms(3)]
  have lincomb-list f ss = lincomb (mk-coeff ss f) (set ss) .
  also have ... = lincomb ( $\lambda v. \text{sum } f (\text{set } (\text{find-indices } v \text{ ss}))$ ) (set ss)
    unfolding mk-coeff-def
    apply (subst R.sumlist-map-as-finsum)
    by (auto simp add: distinct-find-indices)
  ultimately have lincomb-list f ss = lincomb ( $\lambda v. \text{sum } f (\text{set } (\text{find-indices } v \text{ ss}))$ )
(set ss) by auto
  then have *:lincomb ( $\lambda v. \text{sum } f (\text{set } (\text{find-indices } v \text{ ss}))$ ) (set ss) =  $0_v$  n
    using assms(4) by auto
  have finite (set ss) by simp
  from not-lindepD[OF assms(2) this - - *]
  have ( $\lambda v. \text{sum } f (\text{set } (\text{find-indices } v \text{ ss}))$ )  $\in$  set ss  $\rightarrow$  {0}
    by auto
  from funcset-mem[OF this]
  have sum f (set (find-indices (nth ss i) ss))  $\in$  {0}
    using assms(5) by auto
  thus ?thesis unfolding find-indices-distinct[OF assms(1) assms(5)]
    by auto
qed

```

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 –
have *: $\bigwedge v. \exists ca. \text{lincomb-list } v \ (\text{cols } (A * B)) = \text{lincomb-list } ca \ (\text{cols } A)$
proof –
fix v
have $\text{lincomb-list } v \ (\text{cols } (A * B)) = (A * B) *_v \text{vec } nc \ v$
apply (*subst lincomb-list-as-mat-mult*)
apply (*metis assms(1) carrier-dim-vec carrier-matD(1) cols-dim index-mult-mat(2) subset-code(1)*)
by (*metis assms(1) assms(2) carrier-matD(1) carrier-matD(2) cols-length index-mult-mat(2) index-mult-mat(3) mat-of-cols-cols*)
also have $\dots = A *_v (B *_v \text{vec } nc \ v)$
using *assms(1) assms(2)* **by** *auto*
also have $\dots = \text{lincomb-list } (\lambda i. (B *_v \text{vec } nc \ v) \$ i) \ (\text{cols } A)$
apply (*subst lincomb-list-as-mat-mult*)
using *assms(1) carrier-dim-vec cols-dim* **apply** *blast*
by (*metis assms(1) assms(2) carrier-matD(1) carrier-matD(2) cols-length dim-mult-mat-vec dim-vec eq-vecI index-vec mat-of-cols-cols*)
ultimately have $\text{lincomb-list } v \ (\text{cols } (A * B)) = \text{lincomb-list } (\lambda i. (B *_v \text{vec } nc \ v) \$ i) \ (\text{cols } A)$ **by** *auto*
thus $\exists ca. \text{lincomb-list } v \ (\text{cols } (A * B)) = \text{lincomb-list } ca \ (\text{cols } A)$ **by** *auto*
qed
show *?thesis*
apply (*subst span-list-as-span[symmetric]*)
apply (*metis assms(1) carrier-matD(1) cols-dim index-mult-mat(2)*)
apply (*subst span-list-as-span[symmetric]*)
using *assms(1) cols-dim* **apply** *blast*
by (*auto simp add:span-list-def **)
qed

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 –
have $\text{subspace class-ring } (\text{local.span } (\text{set } (\text{cols } (A*B)))) \ (\text{vs } (\text{local.span } (\text{set } (\text{cols } A))))$
unfolding *subspace-def*
by (*metis assms(1) assms(2) carrier-matD(1) cols-dim index-mult-mat(2) nested-submodules span-is-submodule vec-space.span-mat-mul-subset vec-vs-col*)
from *vectorspace.subspace-dim[OF - this]*
have $\text{vectorspace.dim class-ring } (\text{vs } (\text{local.span } (\text{set } (\text{cols } A)))) \ (\text{vs } (\text{local.span } (\text{set } (\text{cols } (A * B)))) \leq$

```

vectorspace.dim class-ring
  (vs (local.span (set (cols A)))))
  apply auto
  by (metis (no-types) assms(1) carrier-matD(1) fin-dim-span-cols index-mult-mat(2)
mat-of-cols-carrier(1) mat-of-cols-cols vec-vs-col)
  thus ?thesis unfolding rank-def
  by auto
qed

```

```

lemma sumlist-drop:
  assumes  $\bigwedge v. v \in \text{set } ls \implies \text{dim-vec } v = n$ 
  shows sumlist ls = sumlist (filter ( $\lambda v. v \neq 0_v$  n) ls)
  using assms
proof (induction ls)
  case Nil
  then show ?case by auto
next
  case (Cons a ls)
  then show ?case using dim-sumlist by auto
qed

```

```

lemma lincomb-list-alt:
  shows lincomb-list c s =
    sumlist (map2 ( $\lambda i j. i \cdot_v s ! j$ ) (map ( $\lambda i. c i$ ) [0..length s]) [0..length s])
  unfolding lincomb-list-def
  by (smt (verit, ccfv-SIG) length-map map2-map-map map-nth nth-equalityI nth-map)

```

```

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
    sumlist (map2 ( $\lambda i j. i \cdot_v s ! j$ ) (map ( $\lambda i. c i$ ) ls) ls) =
    sumlist (map2 ( $\lambda i j. i \cdot_v s ! j$ ) (map ( $\lambda i. c i$ ) (filter ( $\lambda i. c i \neq 0$ ) ls)) (filter
    ( $\lambda i. c i \neq 0$ ) ls)))
  using assms(2)
proof (induction ls)
  case Nil
  then show ?case by auto
next
  case (Cons a s)
  then show ?case
    apply auto
    apply (subst smult-l-null)
    apply (simp add: assms(1) carrier-vecI)
    apply (subst left-zero-vec)
    apply (subst sumlist-carrier)
    apply auto
  by (metis (no-types, lifting) assms(1) carrier-dim-vec mem-Collect-eq nth-mem
set-filter set-zip-rightD)

```

qed

lemma *two-set*:

assumes *distinct ls*
assumes *set ls = set [a,b]*
assumes $a \neq b$
shows $ls = [a,b] \vee ls = [b,a]$
apply (*cases ls*)
using *assms(2)* apply *auto[1]*

proof –

fix $x\ xs$
assume $ls:ls = x \# xs$
obtain $y\ ys$ where $xs:xs = y \# ys$
by (*metis (no-types) <ls = x # xs> assms(2) assms(3) list.set-cases list.set-intros(1) list.set-intros(2) set-ConsD*)
have $1:x = a \vee x = b$
using $\langle ls = x \# xs \rangle$ *assms(2)* by *auto*
have $2:y = a \vee y = b$
using $\langle ls = x \# xs \rangle \langle xs = y \# ys \rangle$ *assms(2)* by *auto*
have $3:ys = []$
using $\langle ls = x \# xs \rangle \langle xs = y \# ys \rangle$ *assms*
by (*cases ys; clarsimp*) (*metis insertE insertI1 insertI2 singletonD*)
show $ls = [a, b] \vee ls = [b, a]$ using $ls\ xs\ 1\ 2\ 3$ *assms*
by *auto*

qed

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 –

have $1: \text{distinct } (\text{filter } (\lambda ia. ia = i \vee ia = j) [0..<\text{length } ls])$
using *distinct-filter distinct-upt* by *blast*
have $2:\text{set } (\text{filter } (\lambda ia. ia = i \vee ia = j) [0..<\text{length } ls]) = \{i, j\}$
using *assms* by *auto*
show *?thesis* using *two-set[OF 1]*
using *assms(3) empty-set filter-cong list.simps(15)*
by (*smt(verit, ccfv-SIG) 2 assms(3) empty-set filter-cong list.simps(15)*)

qed

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*
unfolding *distinct-conv-nth*

proof *clarsimp*

fix $i\ j$
assume $ij: i < \text{length } ls\ j < \text{length } ls\ i \neq j$

```

assume lsij:  $ls ! i = ls ! j$ 
have lincomb-list ( $\lambda k. \text{if } k = i \text{ then } 1 \text{ else if } k = j \text{ then } -1 \text{ else } 0$ )  $ls =$ 
  ( $ls ! i$ ) - ( $ls ! j$ )
unfolding lincomb-list-alt
apply (subst lincomb-list-alt2[OF assms(1)])
apply auto
using filter-disj-inds[OF ij]
apply auto
using ij(3) apply force
using assms(1) ij(2) apply auto[1]
using ij(3) apply blast
using assms(1) ij(2) by auto
also have ... =  $0_v$  n unfolding lsij
apply (rule minus-cancel-vec)
using  $\langle j < \text{length } ls \rangle$  assms(1)
using carrier-vec-dim-vec nth-mem by blast
ultimately have lincomb-list ( $\lambda k. \text{if } k = i \text{ then } 1 \text{ else if } k = j \text{ then } -1 \text{ else } 0$ )
 $ls = 0_v$  n by auto
from assms(2)[OF this]
show False
using  $\langle i < \text{length } ls \rangle$  by auto
qed

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  $\in$  carrier-mat n nc
  shows vec-space.rank nc  $A^H \leq \text{rank } (A * A^H)$ 
proof -
  have 1:  $A^H \in \text{carrier-mat } nc \ n$  using assms by auto
  have 2:  $A * A^H \in \text{carrier-mat } n \ n$  using assms by auto

  let ?P = ( $\lambda T. T \subseteq \text{set } (\text{cols } A^H) \wedge \text{module.lin-indpt class-ring } (\text{module-vec}$ 
  TYPE('a) nc) T)
  have  $*$ :  $\bigwedge A. ?P A \implies \text{finite } A \wedge \text{card } A \leq n$ 
  proof clarsimp
  fix S
  assume S:  $S \subseteq \text{set } (\text{cols } A^H)$ 
  have  $\text{card } S \leq \text{card } (\text{set } (\text{cols } A^H))$  using S
  using card-mono by blast
  also have ...  $\leq \text{length } (\text{cols } A^H)$  using card-length by blast
  also have ...  $\leq n$  using assms by auto
  ultimately show  $\text{finite } S \wedge \text{card } S \leq n$ 

```

```

    by (meson List.finite-set S dual-order.trans finite-subset)
qed
have **: ?P {}
  apply (subst module.lin-dep-def)
  by (auto simp add: vec-module)
from maximal-exists[OF *]
obtain S where S: maximal S ?P using **
  by (metis (no-types, lifting))

from vec-space.rank-card-indpt[OF 1 S]
have rankeq: vec-space.rank nc AH = card S .

have s-hyp: S ⊆ set (cols AH)
  using S unfolding maximal-def by simp
have modhyp: module.lin-indpt class-ring (module-vec TYPE('a) nc) S
  using S unfolding maximal-def by simp

obtain ss where ss: set ss = S distinct ss
  by (metis (mono-tags) S maximal-def set-obtain-sublist)
have ss2: set (map ((*v) A) ss) = (*v) A ' S
  by (simp add: ss(1))
have rw-hyp: cols (mat-of-cols n (map ((*v) A) ss)) = cols (A * mat-of-cols nc
ss)
  unfolding cols-def apply (auto)
  using mat-vec-as-mat-mat-mult[of A n nc]
  by (metis (no-types, lifting) 1 assms carrier-matD(1) cols-dim mul-mat-of-cols
nth-mem s-hyp ss(1) subset-code(1))
  then have rw: mat-of-cols n (map ((*v) A) ss) = A * mat-of-cols nc ss
    by (metis assms carrier-matD(1) index-mult-mat(2) mat-of-cols-carrier(2)
mat-of-cols-cols)
  have indpt: ∧c. lincomb-list c (map ((*v) A) ss) = 0v n ⇒
    ∀i. (i < (length ss) → c i = 0)
  proof clarsimp
    fix c i
    assume *: lincomb-list c (map ((*v) A) ss) = 0v n
    assume i: i < length ss
    have ∀w∈set (map ((*v) A) ss). dim-vec w = n
      using assms by auto
    from lincomb-list-as-mat-mult[OF this]
    have A * mat-of-cols nc ss *v vec (length ss) c = 0v n
      using * rw by auto
    then have hq: A *v (mat-of-cols nc ss *v vec (length ss) c) = 0v n
      by (metis assms assoc-mult-mat-vec mat-of-cols-carrier(1) vec-carrier)

  then have eq1: (mat-of-cols nc ss *v vec (length ss) c) = 0v nc
    apply (intro mat-mul-conjugate-transpose-sub-vec-eq-0)
    using assms ss s-hyp by auto

```

```

have dim-hyp2:  $\forall w \in \text{set } ss. \text{dim-vec } w = nc$ 
  using ss(1) s-hyp
  by (metis 1 carrier-matD(1) carrier-vecD cols-dim subsetD)
from vec-module.lincomb-list-as-mat-mult[OF this, symmetric]
have mat-of-cols nc ss  $\ast_v$  vec (length ss) c = module.lincomb-list (module-vec
TYPE('a) nc) c ss .
  then have module.lincomb-list (module-vec TYPE('a) nc) c ss =  $0_v$  nc using
eq1 by auto
  from vec-space.lin-indpt-lin-comb-list[OF ss(2) - - this i]
  show c i = 0 using modhyp ss s-hyp
  using 1 cols-dim by blast
qed
have distinct: distinct (map (( $\ast_v$ ) A) ss)
  by (metis (no-types, lifting) assms carrier-matD(1) dim-mult-mat-vec imageE
indpt length-map lincomb-list-indpt-distinct ss2)
then have 3: card S = card (( $\ast_v$ ) A ' S)
  by (metis ss distinct-card image-set length-map)
then have 4: ( $\ast_v$ ) A ' S  $\subseteq$  set (cols (A * AH))
  using cols-mat-mul 'S  $\subseteq$  set (cols AH) by blast
have 5: lin-indpt (( $\ast_v$ ) A ' S)
proof clarsimp
  assume ld:lin-dep (( $\ast_v$ ) A ' S)
  have *: finite (( $\ast_v$ ) A ' S)
  by (metis List.finite-set ss2)
  have **: ( $\ast_v$ ) A ' S  $\subseteq$  carrier-vec n
  using 2 4 cols-dim by blast
  from finite-lin-dep[OF * ld **]
  obtain a v where
    a: lincomb a (( $\ast_v$ ) A ' S) =  $0_v$  n and
    v: v  $\in$  ( $\ast_v$ ) A ' S a v  $\neq$  0 by blast
  obtain i where i:v = map (( $\ast_v$ ) A) ss ! i i < length ss
  using v unfolding ss2[symmetric]
  using find-first-le nth-find-first by force
  from ss2[symmetric]
  have set (map (( $\ast_v$ ) A) ss)  $\subseteq$  carrier-vec n using ** ss2 by auto
  from lincomb-as-lincomb-list-distinct[OF this distinct] have
    lincomb-list
    ( $\lambda i. a$  (map (( $\ast_v$ ) A) ss ! i)) (map (( $\ast_v$ ) A) ss) =  $0_v$  n
  using a ss2 by auto
  from indpt[OF this]
  show False using v i by simp
qed
from rank-ge-card-indpt[OF 2 4 5]
have card (( $\ast_v$ ) A ' S)  $\leq$  rank (A * AH) .
thus ?thesis using rankeq 3 by linarith
qed

```

lemma conjugate-transpose-rank-le:

assumes $A \in \text{carrier-mat } n \text{ nc}$
shows $\text{vec-space.rank nc } (A^H) \leq \text{rank } A$
by (*metis* *assms carrier-matD(2) carrier-mat-triv dim-row-conjugate dual-order.trans index-transpose-mat(2) rank-mat-mul-right transpose-rank-mul-conjugate-transpose*)

lemma *conjugate-finsum*:

assumes $f: f : U \rightarrow \text{carrier-vec } n$
shows $\text{conjugate } (\text{finsum } V f U) = \text{finsum } V (\text{conjugate } \circ f) U$
using f
proof (*induct U rule: infinite-finite-induct*)
case (*infinite A*)
then show *?case by auto*
next
case *empty*
then show *?case by auto*
next
case (*insert u U*)
hence $f: f : U \rightarrow \text{carrier-vec } n \text{ f } u : \text{carrier-vec } n$ **by** *auto*
then have $cf: \text{conjugate } \circ f : U \rightarrow \text{carrier-vec } n$
 $(\text{conjugate } \circ f) u : \text{carrier-vec } n$
apply (*simp add: Pi-iff*)
by (*simp add: f(2)*)
then show *?case*
unfolding *finsum-insert[OF insert(1) insert(2) f]*
unfolding *finsum-insert[OF insert(1) insert(2) cf]*
apply (*subst conjugate-add-vec[of - n]*)
using $f(2)$ **apply** *blast*
using $M.\text{finsum-closed } f(1)$ **apply** *blast*
by (*simp add: comp-def f(1) insert.hyps(3)*)

qed

lemma *rank-conjugate-le*:

assumes $A: A \in \text{carrier-mat } n \text{ d}$
shows $\text{rank } (\text{conjugate } (A)) \leq \text{rank } A$
proof –

let $?P = (\lambda T. T \subseteq \text{set } (\text{cols } (\text{conjugate } A)) \wedge \text{lin-indpt } T)$
have $*: \bigwedge A. ?P A \implies \text{finite } A \wedge \text{card } A \leq d$
by (*metis List.finite-set assms card-length card-mono carrier-matD(2) cols-length dim-col-conjugate dual-order.trans rev-finite-subset*)
have $** : ?P \{\}$
by (*simp add: finite-lin-indpt2*)
from *maximal-exists[OF *]*
obtain S **where** $S: \text{maximal } S \text{ ?P}$ **using** $**$
by (*metis (no-types, lifting)*)
have $s\text{-hyp}: S \subseteq \text{set } (\text{cols } (\text{conjugate } A)) \text{ lin-indpt } S$
using S **unfolding** *maximal-def*
apply *blast*
by (*metis (no-types, lifting) S maximal-def*)

```

from rank-card-indpt[OF - S, of d]
have rankeq: rank (conjugate A) = card S using assms by auto
have 1:conjugate ' S  $\subseteq$  set (cols A)
using S apply auto
by (metis (no-types, lifting) cols-conjugate conjugate-id image-eqI in-mono
list.set-map s-hyp(1))
have 2: lin-indpt (conjugate ' S)
apply (rule ccontr)
apply (auto simp add: lin-dep-def)
proof -
fix T c v
assume T: T  $\subseteq$  conjugate ' S finite T and
lc:lincomb c T = 0_v n and v  $\in$  T c v  $\neq$  0
let ?T = conjugate ' T
let ?c = conjugate  $\circ$  c  $\circ$  conjugate
have 1: finite ?T using T by auto
have 2: ?T  $\subseteq$  S using T by auto
have 3: ?c  $\in$  ?T  $\rightarrow$  UNIV by auto
have lincomb ?c ?T = ( $\bigoplus_{v \in T.}$  conjugate (c x)  $\cdot_v$  conjugate x)
unfolding lincomb-def
apply (subst finsum-reindex)
apply auto
apply (metis 2 carrier-vec-conjugate assms carrier-matD(1) cols-dim im-
age-eqI s-hyp(1) subsetD)
by (meson conjugate-cancel-iff inj-onI)
also have ... = ( $\bigoplus_{v \in T.}$  conjugate (c x  $\cdot_v$  x))
by (simp add: conjugate-smult-vec)
also have ... = conjugate ( $\bigoplus_{v \in T.}$  (c x  $\cdot_v$  x))
apply(subst conjugate-finsum[of  $\lambda x.(c x \cdot_v x)$  T])
apply (auto simp add:o-def)
by (smt (verit, ccfv-SIG) Matrix.carrier-vec-conjugate Pi-I' T(1) assms car-
rier-matD(1) cols-dim dim-row-conjugate imageE s-hyp(1) smult-carrier-vec sub-
set-eq)
also have ... = conjugate (lincomb c T)
using lincomb-def by presburger
ultimately have lincomb ?c ?T = conjugate (lincomb c T) by auto
then have 4:lincomb ?c ?T = 0_v n using lc by auto
from not-lindepD[OF s-hyp(2) 1 2 3 4]
have conjugate  $\circ$  c  $\circ$  conjugate  $\in$  conjugate ' T  $\rightarrow$  {0} .
then have c v = 0
by (simp add: Pi-iff  $\langle v \in T \rangle$ )
thus False using  $\langle c v \neq 0 \rangle$  by auto
qed
from rank-ge-card-indpt[OF A 1 2]
have 3:card (conjugate ' S)  $\leq$  rank A .
have 4: card (conjugate ' S) = card S
apply (auto intro!: card-image)
by (meson conjugate-cancel-iff inj-onI)
show ?thesis using rankeq 3 4 by auto

```

qed

lemma *rank-conjugate*:

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

shows $\text{rank } (\text{conjugate } A) = \text{rank } A$

using *rank-conjugate-le*

by (*metis carrier-vec-conjugate assms conjugate-id dual-order.antisym*)

end

lemma *conjugate-transpose-rank*:

fixes $A::'a::\{\text{conjugatable-ordered-field}\} \text{ mat}$

shows $\text{vec-space.rank } (\text{dim-row } A) \ A = \text{vec-space.rank } (\text{dim-col } A) \ (A^H)$

using *conjugatable-vec-space.conjugate-transpose-rank-le*

by (*metis (no-types, lifting) Matrix.transpose-transpose carrier-matI conjugate-id dim-col-conjugate dual-order.antisym index-transpose-mat(2) transpose-conjugate*)

lemma *transpose-rank*:

fixes $A::'a::\{\text{conjugatable-ordered-field}\} \text{ mat}$

shows $\text{vec-space.rank } (\text{dim-row } A) \ A = \text{vec-space.rank } (\text{dim-col } A) \ (A^T)$

by (*metis carrier-mat-triv conjugatable-vec-space.rank-conjugate conjugate-transpose-rank index-transpose-mat(2)*)

lemma *rank-mat-mul-left*:

fixes $A::'a::\{\text{conjugatable-ordered-field}\} \text{ mat}$

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

assumes $B \in \text{carrier-mat } d \ nc$

shows $\text{vec-space.rank } n \ (A * B) \leq \text{vec-space.rank } d \ B$

by (*metis (no-types, lifting) Matrix.transpose-transpose assms(1) assms(2) carrier-matD(1) carrier-matD(2) carrier-mat-triv conjugatable-vec-space.rank-conjugate conjugate-transpose-rank index-mult-mat(3) index-transpose-mat(3) transpose-mult vec-space.rank-mat-mul-right*)

3 Results on Invertibility

definition *take-cols* :: $'a \text{ mat} \Rightarrow \text{nat list} \Rightarrow 'a \text{ mat}$

where $\text{take-cols } A \ \text{inds} = \text{mat-of-cols } (\text{dim-row } A) \ (\text{map } (!) \ (\text{cols } A)) \ (\text{filter } (>) \ (\text{dim-col } A)) \ \text{inds})$

definition *take-cols-var* :: $'a \text{ mat} \Rightarrow \text{nat list} \Rightarrow 'a \text{ mat}$

where $\text{take-cols-var } A \ \text{inds} = \text{mat-of-cols } (\text{dim-row } A) \ (\text{map } (!) \ (\text{cols } A)) \ (\text{inds})$

definition *take-rows* :: $'a \text{ mat} \Rightarrow \text{nat list} \Rightarrow 'a \text{ mat}$

where $\text{take-rows } A \ \text{inds} = \text{mat-of-rows } (\text{dim-col } A) \ (\text{map } (!) \ (\text{rows } A)) \ (\text{filter } (>) \ (\text{dim-row } A)) \ \text{inds})$

lemma *cong1*:

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

by *auto*

lemma *nth-filter*:
assumes $j < \text{length} (\text{filter } P \text{ } ls)$
shows $P ((\text{filter } P \text{ } ls) ! j)$
by (*simp add: assms list-ball-nth*)

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 –
have $\bigwedge j. j < \text{length} (\text{map } (!) (\text{cols } B)) (\text{filter } ((>) \ nc) \ inds) \implies$
 $(\text{map } (!) (\text{cols } B)) (\text{filter } ((>) \ nc) \ inds) ! j \in \text{carrier-vec } n$
using *assms apply auto*
apply (*subst cols-nth*)
using *nth-filter by auto*
from *mul-mat-of-cols[OF assms(1) this]*
have $A * \text{take-cols } B \text{ } inds = \text{mat-of-cols } nr (\text{map } (\lambda x. A *_v \text{cols } B ! x) (\text{filter}$
 $((>) (\text{dim-col } B)) \ inds))$
unfolding *take-cols-def* **using** *assms* **by** (*auto simp add: o-def*)
also have $\dots = \text{take-cols } (A * B) \text{ } inds$
unfolding *take-cols-def* **using** *assms* **by** (*auto intro!: cong1*)
ultimately show *?thesis* **by** *auto*
qed

lemma *take-cols-carrier-mat*:
assumes $A \in \text{carrier-mat } nr \ nc$
obtains n **where** $\text{take-cols } A \text{ } inds \in \text{carrier-mat } nr \ n$
unfolding *take-cols-def*
using *assms*
by *fastforce*

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 $\text{take-cols } A \text{ } inds \in \text{carrier-mat } nr (\text{length } inds)$
unfolding *take-cols-def*
using *assms* **by** *auto*

lemma *gauss-jordan-take-cols*:
assumes *gauss-jordan* $A (\text{take-cols } A \text{ } inds) = (C,D)$
shows $D = \text{take-cols } C \text{ } inds$
proof –
obtain $nr \ nc$ **where** $A: A \in \text{carrier-mat } nr \ nc$ **by** *auto*
from *take-cols-carrier-mat[OF this]*
obtain n **where** $B: \text{take-cols } A \text{ } inds \in \text{carrier-mat } nr \ n$ **by** *auto*
from *gauss-jordan-transform[OF A B assms, of undefined]*
obtain P **where** $PP: P \in \text{Units} (\text{ring-mat } TYPE('a) \ nr \ \text{undefined})$ **and**
 $CD: C = P * A \ D = P * \text{take-cols } A \text{ } inds$ **by** *blast*

have $P: P \in \text{carrier-mat } nr \ nr$
by (*metis* (*no-types*, *lifting*) *Units-def* *PP mem-Collect-eq* *partial-object.select-convs(1)*)
ring-mat-def)
from *take-cols-mat-mul*[*OF P A*]
have $P * \text{take-cols } A \ \text{inds} = \text{take-cols } (P * A) \ \text{inds} .$
thus *?thesis* **using** *CD* **by** *blast*
qed

lemma *dim-col-take-cols*:
assumes $\bigwedge j. j \in \text{set inds} \implies j < \text{dim-col } A$
shows $\text{dim-col } (\text{take-cols } A \ \text{inds}) = \text{length inds}$
unfolding *take-cols-def*
using *assms* **by** *auto*

lemma *dim-col-take-rows*[*simp*]:
shows $\text{dim-col } (\text{take-rows } A \ \text{inds}) = \text{dim-col } A$
unfolding *take-rows-def* **by** *auto*

lemma *cols-take-cols-subset*:
shows $\text{set } (\text{cols } (\text{take-cols } A \ \text{inds})) \subseteq \text{set } (\text{cols } A)$
unfolding *take-cols-def*
apply (*subst cols-mat-of-cols*)
apply *auto*
using *in-set-conv-nth* **by** *fastforce*

lemma *dim-row-take-cols*[*simp*]:
shows $\text{dim-row } (\text{take-cols } A \ \text{ls}) = \text{dim-row } A$
by (*simp* *add: take-cols-def*)

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

lemma *rows-inj*:
assumes $\text{dim-col } A = \text{dim-col } B$
assumes $\text{rows } A = \text{rows } B$
shows $A = B$
unfolding *mat-eq-iff*
apply *auto*
apply (*metis* *assms(2)* *length-rows*)
using *assms(1)* **apply** *blast*
by (*metis* *assms(1)* *assms(2)* *mat-of-rows-rows*)

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

unfolding *append-rows-def*
apply (*subst index-mat-four-block*)
using *assms* **by** *auto*

lemma *row-append-rows*:
assumes $\dim\text{-col } A = \dim\text{-col } B$
assumes $i < \dim\text{-row } A + \dim\text{-row } B$
shows $\text{row } (A @_r B) i = (\text{if } i < \dim\text{-row } A \text{ then row } A i \text{ else row } B (i - \dim\text{-row } A))$
unfolding *vec-eq-iff*
using *assms* **by** (*auto simp add: append-rows-def*)

lemma *append-rows-mat-mul*:
assumes $\dim\text{-col } A = \dim\text{-col } B$
shows $(A @_r B) * C = A * C @_r B * C$
unfolding *mat-eq-iff*
apply *auto*
apply (*simp add: append-rows-def*)
apply (*subst index-mult-mat*)
apply *auto*
apply (*simp add: append-rows-def*)
apply (*subst append-rows-index*)
apply *auto*
apply (*simp add: append-rows-def*)
apply (*metis add.right-neutral append-rows-def assms index-mat-four-block(3) index-mult-mat(1) index-mult-mat(3) index-zero-mat(3) row-append-rows trans-less-add1*)
by (*metis add-cancel-right-right add-diff-inverse-nat append-rows-def assms index-mat-four-block(3) index-mult-mat(1) index-mult-mat(3) index-zero-mat(3) nat-add-left-cancel-less row-append-rows*)

lemma *cardlt*:
shows $\text{card } \{i. i < (n::\text{nat})\} \leq n$
by *simp*

lemma *row-echelon-form-zero-rows*:
assumes *row-ech: row-echelon-form A*
assumes *dim-asm: dim-col A ≥ dim-row A*
shows $\text{take-rows } A [0..<\text{length } (\text{pivot-positions } A)] @_r 0_m (\dim\text{-row } A - \text{length } (\text{pivot-positions } A)) (\dim\text{-col } A) = A$
proof –
have *ex-pivot-fun: ∃ f. pivot-fun A f (dim-col A)* **using** *row-ech* **unfolding** *row-echelon-form-def* **by** *auto*
have *len-help: length (pivot-positions A) = card {i. i < dim-row A ∧ row A i ≠ 0_v (dim-col A)}*
using *ex-pivot-fun pivot-positions* [**where** $A = A$, **where** $nr = \dim\text{-row } A$, **where** $nc = \dim\text{-col } A$]
by *auto*
then have *len-help2: length (pivot-positions A) ≤ dim-row A*
by (*metis (no-types, lifting) card-mono cardlt finite-Collect-less-nat le-trans*)

```

mem-Collect-eq subsetI)
  have fileq: filter (λy. y < dim-row A) [0..<length (pivot-positions A)] = [0..<length
(pivot-positions A)]
    apply (rule filter-True)
    using len-help2 by auto
  have ∀ n. card {i. i < n ∧ row A i ≠ 0_v (dim-col A)} ≤ n
  proof clarsimp
    fix n
    have h: ∀ x. x ∈ {i. i < n ∧ row A i ≠ 0_v (dim-col A)} → x ∈ {..<n}
      by simp
    then have h1: {i. i < n ∧ row A i ≠ 0_v (dim-col A)} ⊆ {..<n}
      by blast
    then have h2: (card {i. i < n ∧ row A i ≠ 0_v (dim-col A)}::nat) ≤ (card
{..<n}::nat)
      using card-mono by blast
    then show (card {i. i < n ∧ row A i ≠ 0_v (dim-col A)}::nat) ≤ (n::nat) using
h2 card-lessThan[of n]
      by auto
    qed
  then have pivot-len: length (pivot-positions A) ≤ dim-row A using len-help
    by simp
  have alt-char: mat-of-rows (dim-col A)
    (map (!) (rows A)) (filter (λy. y < dim-col A) [0..<length (pivot-positions
A)])) =
    mat-of-rows (dim-col A) (map (!) (rows A)) [0..<length (pivot-positions A)])
    using pivot-len dim-asm
    by auto
  have h1: ∧ i j. i < dim-row A ⇒
    j < dim-col A ⇒
    i < dim-row (take-rows A [0..<length (pivot-positions A)]) ⇒
    take-rows A [0..<length (pivot-positions A)] $$ (i, j) = A $$ (i, j)
  proof -
    fix i
    fix j
    assume i < dim-row A
    assume j-lt: j < dim-col A
    assume i-lt: i < dim-row (take-rows A [0..<length (pivot-positions A)])
    have lt: length (pivot-positions A) ≤ dim-row A using pivot-len by auto
    have h1: take-rows A [0..<length (pivot-positions A)] $$ (i, j) = (row (take-rows
A [0..<length (pivot-positions A)]) i)$j
      by (simp add: i-lt j-lt)
    then have h2: (row (take-rows A [0..<length (pivot-positions A)]) i)$j = (row
A i)$j
      using lt alt-char i-lt unfolding take-rows-def by auto
    show take-rows A [0..<length (pivot-positions A)] $$ (i, j) = A $$ (i, j)
      using h1 h2
      by (simp add: ⟨i < dim-row A⟩ j-lt)
    qed
  let ?nc = dim-col A

```

```

let ?nr = dim-row A
have h2:  $\bigwedge i j. i < \text{dim-row } A \implies$ 
   $j < \text{dim-col } A \implies$ 
   $\neg i < \text{dim-row } (\text{take-rows } A [0..<\text{length } (\text{pivot-positions } A)]) \implies$ 
   $0_m (\text{dim-row } A - \text{length } (\text{pivot-positions } A)) (\text{dim-col } A) \text{ \$\$}$ 
   $(i - \text{dim-row } (\text{take-rows } A [0..<\text{length } (\text{pivot-positions } A)]), j) =$ 
   $A \text{ \$\$ } (i, j)$ 
proof -
  fix i
  fix j
  assume lt-i:  $i < \text{dim-row } A$ 
  assume lt-j:  $j < \text{dim-col } A$ 
  assume not-lt:  $\neg i < \text{dim-row } (\text{take-rows } A [0..<\text{length } (\text{pivot-positions } A)])$ 
  let ?ip = i+1
  have h0:  $\exists f. \text{pivot-fun } A f (\text{dim-col } A) \wedge f i = ?nc$ 
  proof -
    have half1:  $\exists f. \text{pivot-fun } A f (\text{dim-col } A)$  using assms unfolding row-echelon-form-def
      by blast
    have half2:  $\forall f. \text{pivot-fun } A f (\text{dim-col } A) \longrightarrow f i = ?nc$ 
    proof clarsimp
      fix f
      assume is-piv:  $\text{pivot-fun } A f (\text{dim-col } A)$ 
      have len-pp:  $\text{length } (\text{pivot-positions } A) = \text{card } \{i. i < ?nr \wedge \text{row } A i \neq 0_v\}$ 
      ?nc} using is-piv pivot-positions[of A ?nr ?nc f]
        by auto
      have  $\forall i. (i < ?nr \wedge \text{row } A i \neq 0_v \text{ ?nc}) \longleftrightarrow (i < ?nr \wedge f i \neq ?nc)$ 
        using is-piv pivot-fun-zero-row-iff[of A f ?nc ?nr]
        by blast
      then have len-pp-var:  $\text{length } (\text{pivot-positions } A) = \text{card } \{i. i < ?nr \wedge f i$ 
 $\neq ?nc\}$ 
        using len-pp by auto
      have allj-hyp:  $\forall j < ?nr. f j = ?nc \longrightarrow ((\text{Suc } j) < ?nr \longrightarrow f (\text{Suc } j) = ?nc)$ 
        using is-piv unfolding pivot-fun-def
        using lt-i
        by (metis le-antisym le-less)
      have if-then-bad:  $f i \neq ?nc \longrightarrow (\forall j. j \leq i \longrightarrow f j \neq ?nc)$ 
    proof clarsimp
      fix j
      assume not-i:  $f i \neq ?nc$ 
      assume j-leq:  $j \leq i$ 
      assume bad-asm:  $f j = ?nc$ 
      have  $\bigwedge k. k \geq j \implies k < ?nr \implies f k = ?nc$ 
    proof -
      fix k :: nat
      assume a1:  $j \leq k$ 
      assume a2:  $k < \text{dim-row } A$ 
      have f3:  $\bigwedge n. \neg n < \text{dim-row } A \vee f n \neq f j \vee \neg \text{Suc } n < \text{dim-row } A \vee f$ 
 $(\text{Suc } n) = f j$ 

```

```

using allj-hyp bad-asm by presburger
obtain nn :: nat => nat => (nat => bool) => nat where
  f4:  $\bigwedge n na p nb nc. (\neg n \leq na \vee \text{Suc } n \leq \text{Suc } na) \wedge (\neg p nb \vee \neg nc \leq nb \vee \neg p (nn nc nb p) \vee p nc) \wedge (\neg p nb \vee \neg nc \leq nb \vee p nc \vee p (\text{Suc } (nn nc nb p)))$ 
    using inc-induct by (metis Suc-le-mono)
then have f5:  $\bigwedge p. \neg p k \vee p j \vee p (\text{Suc } (nn j k p))$ 
    using a1 by presburger
have f6:  $\bigwedge p. \neg p k \vee \neg p (nn j k p) \vee p j$ 
    using f4 a1 by meson
    { assume nn j k ( $\lambda n. n < \text{dim-row } A \wedge f n \neq \text{dim-col } A$ ) < dim-row A
       $\wedge f (nn j k (\lambda n. n < \text{dim-row } A \wedge f n \neq \text{dim-col } A)) \neq \text{dim-col } A$ 
      moreover
        { assume (nn j k ( $\lambda n. n < \text{dim-row } A \wedge f n \neq \text{dim-col } A$ ) < dim-row
          A  $\wedge f (nn j k (\lambda n. n < \text{dim-row } A \wedge f n \neq \text{dim-col } A)) \neq \text{dim-col } A$ )  $\wedge (\neg j <$ 
            dim-row A  $\vee f j = \text{dim-col } A$ )
          then have  $\neg k < \text{dim-row } A \vee f k = \text{dim-col } A$ 
          using f6
          by (metis (mono-tags, lifting)) }
        ultimately have  $(\neg j < \text{dim-row } A \vee f j = \text{dim-col } A) \wedge (\neg \text{Suc } (nn$ 
          j k ( $\lambda n. n < \text{dim-row } A \wedge f n \neq \text{dim-col } A$ )) < dim-row A  $\vee f (\text{Suc } (nn j k (\lambda n.$ 
            n < dim-row A  $\wedge f n \neq \text{dim-col } A))) = \text{dim-col } A) \vee \neg k < \text{dim-row } A \vee f k =$ 
            dim-col A
          using bad-asm
          by blast }
        moreover
          { assume  $(\neg j < \text{dim-row } A \vee f j = \text{dim-col } A) \wedge (\neg \text{Suc } (nn j k (\lambda n. n$ 
            < dim-row A  $\wedge f n \neq \text{dim-col } A$ )) < dim-row A  $\vee f (\text{Suc } (nn j k (\lambda n. n < \text{dim-row}$ 
              A  $\wedge f n \neq \text{dim-col } A))) = \text{dim-col } A$ )
            then have  $\neg k < \text{dim-row } A \vee f k = \text{dim-col } A$ 
            using f5
            proof -
              have  $\neg (\text{Suc } (nn j k (\lambda n. n < \text{dim-row } A \wedge f n \neq \text{dim-col } A)) <$ 
                dim-row A  $\wedge f (\text{Suc } (nn j k (\lambda n. n < \text{dim-row } A \wedge f n \neq \text{dim-col } A))) \neq \text{dim-col}$ 
                A)  $\wedge \neg (j < \text{dim-row } A \wedge f j \neq \text{dim-col } A)$ 
              using  $\langle (\neg j < \text{dim-row } A \vee f j = \text{dim-col } A) \wedge (\neg \text{Suc } (nn j k$ 
                ( $\lambda n. n < \text{dim-row } A \wedge f n \neq \text{dim-col } A$ )) < dim-row A  $\vee f (\text{Suc } (nn j k (\lambda n. n <$ 
                  dim-row A  $\wedge f n \neq \text{dim-col } A))) = \text{dim-col } A) \rangle$  by linarith
              then have  $\neg (k < \text{dim-row } A \wedge f k \neq \text{dim-col } A)$ 
              by (metis (mono-tags, lifting) a2 bad-asm f5 le-less)
              then show ?thesis
              by meson
            qed }
          ultimately show f k = dim-col A
          using f3 a2 by (metis (lifting) Suc-lessD bad-asm)
        qed
      then show False using lt-i not-i
      using j-leq by blast
    qed

```

```

have  $f\ i \neq ?nc \longrightarrow (\{0..<?ip\} \subseteq \{y. y < ?nr \wedge f\ y \neq \text{dim-col } A\})$ 
proof -
  have  $h1: f\ i \neq \text{dim-col } A \longrightarrow (\forall j \leq i. j < ?nr \wedge f\ j \neq \text{dim-col } A)$ 
    using if-then-bad lt-i by auto
  then show ?thesis by auto
qed
then have  $gteq: f\ i \neq ?nc \longrightarrow (\text{card } \{i. i < ?nr \wedge f\ i \neq \text{dim-col } A\} \geq$ 
( $i+1$ ))
  using card-lessThan[of ?ip] card-mono[where  $B = \{i. i < ?nr \wedge f\ i \neq \text{dim-col } A\}$ , where  $A = \{0..<?ip\}$ ]
  by auto
then have  $clear: \text{dim-row } (\text{take-rows } A\ [0..<\text{length } (\text{pivot-positions } A)]) =$ 
 $\text{length } (\text{pivot-positions } A)$ 
  unfolding take-rows-def using dim-asm fileq by (auto)
have  $i + 1 > \text{length } (\text{pivot-positions } A)$  using not-lt clear by auto
then show  $f\ i = ?nc$  using gteq len-pp-var by auto
qed
show ?thesis using half1 half2
  by blast
qed
then have  $h1a: \text{row } A\ i = 0_v\ (\text{dim-col } A)$ 
  using pivot-fun-zero-row-iff[of  $A - ?nc\ ?nr$ ]
  using lt-i by blast
then have  $h1: A\ \$\$ (i, j) = 0$ 
  using index-row(1) lt-i lt-j by fastforce
have  $h2a: i - \text{dim-row } (\text{take-rows } A\ [0..<\text{length } (\text{pivot-positions } A)]) < \text{dim-row}$ 
 $A - \text{length } (\text{pivot-positions } A)$ 
  using pivot-len lt-i not-lt
  by (simp add: take-rows-def)
then have  $h2: 0_m\ (\text{dim-row } A - \text{length } (\text{pivot-positions } A))\ (\text{dim-col } A)\ \$\$$ 
 $(i - \text{dim-row } (\text{take-rows } A\ [0..<\text{length } (\text{pivot-positions } A)]), j) = 0$ 
  unfolding zero-mat-def using pivot-len lt-i lt-j
  using index-mat(1) by blast
then show  $0_m\ (\text{dim-row } A - \text{length } (\text{pivot-positions } A))\ (\text{dim-col } A)\ \$\$$ 
 $(i - \text{dim-row } (\text{take-rows } A\ [0..<\text{length } (\text{pivot-positions } A)]), j) =$ 
 $A\ \$\$ (i, j)$  using h1 h2
  by simp
qed
have  $h3: (\text{dim-row } (\text{take-rows } A\ [0..<\text{length } (\text{pivot-positions } A)]))::\text{nat} + ((\text{dim-row}$ 
 $A::\text{nat}) - \text{length } (\text{pivot-positions } A)::\text{nat})) =$ 
 $\text{dim-row } A$ 
proof -
  have  $h0: \text{dim-row } (\text{take-rows } A\ [0..<\text{length } (\text{pivot-positions } A)]) = (\text{length}$ 
 $(\text{pivot-positions } A)::\text{nat})$ 
  by (simp add: take-rows-def fileq)
  then show ?thesis using add-diff-inverse-nat pivot-len
  by linarith
qed
have  $h4: \bigwedge i\ j. i < \text{dim-row } A \implies$ 

```

```

       $j < \text{dim-col } A \implies$ 
       $i < \text{dim-row } (\text{take-rows } A [0..<\text{length } (\text{pivot-positions } A)]) +$ 
       $(\text{dim-row } A - \text{length } (\text{pivot-positions } A))$ 
      using pivot-len
      by (simp add: h3)
    then show ?thesis apply (subst mat-eq-iff)
      using h1 h2 h3 h4 by (auto simp add: append-rows-def)
  qed

lemma length-pivot-positions-dim-row:
  assumes row-echelon-form A
  shows length (pivot-positions A)  $\leq$  dim-row A
proof -
  have 1:  $A \in \text{carrier-mat } (\text{dim-row } A) (\text{dim-col } A)$  by auto
  obtain f where 2: pivot-fun A f (dim-col A)
    using assms row-echelon-form-def by blast
  from pivot-positions(4)[OF 1 2] have
    length (pivot-positions A) = card { $i. i < \text{dim-row } A \wedge \text{row } A \ i \neq 0_v (\text{dim-col } A)$ } .
  also have ...  $\leq$  card { $i. i < \text{dim-row } A$ }
    apply (rule card-mono)
    by auto
  ultimately show ?thesis by auto
qed

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 -
  obtain f where f: pivot-fun R f nc
    using assms(1) assms(2) row-echelon-form-def by blast
  have *:  $\bigwedge i. i < nr \implies f \ i \leq nc$  using f
    using R pivot-funD(1) by blast
  from pivot-positions[OF R f]
  have set (pivot-positions R) = {( $i, f \ i$ ) |  $i. i < nr \wedge f \ i \neq nc$ } by auto
  then have **: set (pivot-positions R) = {( $i, f \ i$ ) |  $i. i < nr \wedge f \ i < nc$ }
    using *
    by fastforce
  fix i j
  assume ( $i, j$ )  $\in$  set (pivot-positions R)
  thus  $i < nr \wedge j < nc$  using **
    by simp
qed

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

```

```

proof –
  fix i
  assume  $i < nr$ 
  show  $(\bigwedge k. ((k < nr \wedge i < k) \longrightarrow f\ i \leq f\ k))$ 
  proof –
    fix k
    show  $((k < nr \wedge i < k) \longrightarrow f\ i \leq f\ k)$ 
    proof (induct k)
      case 0
      then show ?case
      by blast
    next
      case (Suc k)
      then show ?case
      by (smt (verit, ccfv-SIG) dr le-less-trans less-Suc-eq less-imp-le-nat pf
pivot-funD(1) pivot-funD(3))
    qed
  qed
qed

```

lemma *pivot-positions-contains*:

```

assumes row-ech: row-echelon-form A
assumes dim-h:  $\dim\text{-col } A \geq \dim\text{-row } A$ 
assumes pivot-fun A f (dim-col A)
shows  $\forall i < (\text{length } (\text{pivot-positions } A)). (i, f\ i) \in \text{set } (\text{pivot-positions } A)$ 
proof –
  let ?nr = dim-row A
  let ?nc = dim-col A
  let ?pp = pivot-positions A
  have i-nr:  $\forall i < (\text{length } ?pp). i < ?nr$  using rref-pivot-positions assms
    using length-pivot-positions-dim-row less-le-trans by blast
  have i-nc:  $\forall i < (\text{length } ?pp). f\ i < ?nc$ 
  proof clarsimp
    fix i
    assume i-lt:  $i < \text{length } ?pp$ 
    have fis-nc:  $f\ i = ?nc \implies (\forall k > i. k < ?nr \longrightarrow f\ k = ?nc)$ 
    proof –
      assume is-nc:  $f\ i = ?nc$ 
      show  $(\forall k > i. k < ?nr \longrightarrow f\ k = ?nc)$ 
      proof clarsimp
        fix k
        assume k-gt:  $k > i$ 
        assume k-lt:  $k < ?nr$ 
        have fk-lt:  $f\ k \leq ?nc$  using pivot-funD(1)[of A ?nr f ?nc k] k-lt apply
(auto)
        using  $\langle \text{pivot-fun } A\ f\ (\text{dim-col } A) \rangle$  by blast
      show  $f\ k = ?nc$ 
      using fk-lt is-nc k-gt k-lt assms pivot-fun-monoton[of A f ?nr i k]
      using  $\langle \text{pivot-fun } A\ f\ (\text{dim-col } A) \rangle$  by auto
    qed
  qed

```

```

    qed
  qed
  have ncimp: f i = ?nc  $\implies$  ( $\forall k \geq i. k \notin \{i. i < ?nr \wedge \text{row } A \ i \neq 0_v \ ?nc\}$ )
  proof -
    assume nchyp: f i = ?nc
    show ( $\forall k \geq i. k \notin \{i. i < ?nr \wedge \text{row } A \ i \neq 0_v \ ?nc\}$ )
    proof clarsimp
      fix k
      assume i-lt: i  $\leq$  k
      assume k-lt: k < dim-row A
      show row A k = 0_v (dim-col A)
        using i-lt k-lt fis-nc
        using pivot-fun-zero-row-iff[of A f ?nc ?nr]
        using  $\langle$ pivot-fun A f (dim-col A) $\rangle$  le-neq-implies-less nchyp by blast
    qed
  qed
  then have f i = ?nc  $\implies$  card { i. i < ?nr  $\wedge$  row A i  $\neq$  0_v ?nc }  $\leq$  i
  proof -
    assume nchyp: f i = ?nc
    have h: { i. i < ?nr  $\wedge$  row A i  $\neq$  0_v ?nc }  $\subseteq$  { 0..i }
      using atLeast0LessThan le-less-linear nchyp ncimp by blast
    then show card { i. i < ?nr  $\wedge$  row A i  $\neq$  0_v ?nc }  $\leq$  i
      using card-lessThan
      using subset-eq-atLeast0-lessThan-card by blast
  qed
  then show f i < ?nc using i-lt pivot-positions(4)[of A ?nr ?nc f]
    apply (auto)
  by (metis  $\langle$ pivot-fun A f (dim-col A) $\rangle$  i-nr le-neq-implies-less not-less pivot-funD(1))

  qed
  then show ?thesis
    using pivot-positions(1)
  by (smt (verit, ccfv-SIG)  $\langle$ pivot-fun A f (dim-col A) $\rangle$  carrier-matI i-nr less-not-refl
  mem-Collect-eq)
  qed

  lemma pivot-positions-form-helper-1:
    shows (a, b)  $\in$  set (pivot-positions-main-gen z A nr nc i j)  $\implies$  i  $\leq$  a
  proof (induct i j rule: pivot-positions-main-gen.induct[of nr nc A z])
    case (1 i j)
    then show ?case using pivot-positions-main-gen.simps[of z A nr nc i j]
      by (metis Pair-inject Suc-leD emptyE list.set(1) nle-le set-ConsD)
  qed

  lemma pivot-positions-form-helper-2:
    shows sorted-wrt (<) (map fst (pivot-positions-main-gen z A nr nc i j))
  proof (induct i j rule: pivot-positions-main-gen.induct[of nr nc A z])
    case (1 i j)
    then show ?case using pivot-positions-main-gen.simps[of z A nr nc i j]

```

by (auto simp: pivot-positions-form-helper-1 Suc-le-lessD)
qed

lemma sorted-pivot-positions:

shows sorted-wrt ($<$) (map fst (pivot-positions A))

using pivot-positions-form-helper-2

by (simp add: pivot-positions-form-helper-2 pivot-positions-gen-def)

lemma pivot-positions-form:

assumes row-ech: row-echelon-form A

assumes dim-h: dim-col A \geq dim-row A

shows $\forall i < (\text{length } (\text{pivot-positions } A)). \text{fst } ((\text{pivot-positions } A) ! i) = i$

proof clarsimp

let ?nr = dim-row A

let ?nc = dim-col A

let ?pp = pivot-positions A :: (nat \times nat) list

fix i

assume i-lt: $i < \text{length } (\text{pivot-positions } A)$

have $\exists f. \text{pivot-fun } A f ?nc$ using row-ech unfolding row-echelon-form-def

by blast

then obtain f where pf: pivot-fun A f ?nc

by blast

have all-f-in: $\forall i < (\text{length } ?pp). (i, f i) \in \text{set } ?pp$

using pivot-positions-contains pf

assms

by blast

have sorted-hyp: $\bigwedge (p::\text{nat}) (q::\text{nat}). p < (\text{length } ?pp) \implies q < (\text{length } ?pp) \implies p < q \implies (\text{fst } (?pp ! p) < \text{fst } (?pp ! q))$

proof -

fix p::nat

fix q::nat

assume p-lt: $p < q$

assume p-welldef: $p < (\text{length } ?pp)$

assume q-welldef: $q < (\text{length } ?pp)$

show $\text{fst } (?pp ! p) < \text{fst } (?pp ! q)$

using sorted-pivot-positions p-lt p-welldef q-welldef sorted-wrt-nth-less by

fastforce

qed

have h: $i < (\text{length } ?pp) \longrightarrow \text{fst } (\text{pivot-positions } A ! i) = i$

proof (induct i)

case 0

have $\exists j. \text{fst } (\text{pivot-positions } A ! j) = 0$

by (metis all-f-in fst-conv i-lt in-set-conv-nth length-greater-0-conv list.size(3)

not-less0)

then obtain j where jth: $\text{fst } (\text{pivot-positions } A ! j) = 0$

by blast

have $j \neq 0 \longrightarrow (\text{fst } (\text{pivot-positions } A ! 0) > 0 \longrightarrow j \leq 0)$

by (smt (verit, ccfv-SIG) all-f-in fst-conv i-lt in-set-conv-nth less-nat-zero-code not-gr-zero sorted-hyp)

```

then show ?case
  using jth neq0-conv by blast
next
  case (Suc i)
  have ind-h:  $i < \text{length } (\text{pivot-positions } A) \longrightarrow \text{fst } (\text{pivot-positions } A ! i) = i$ 
    using Suc.hyps by blast
  have thesis-h:  $(\text{Suc } i) < \text{length } (\text{pivot-positions } A) \implies \text{fst } (\text{pivot-positions } A ! (\text{Suc } i)) = (\text{Suc } i)$ 
  proof –
    assume suc-i-lt:  $(\text{Suc } i) < \text{length } (\text{pivot-positions } A)$ 
    have fst-i-is:  $\text{fst } (\text{pivot-positions } A ! i) = i$  using suc-i-lt ind-h
      using Suc-lessD by blast
    have  $(\exists j < (\text{length } ?pp). \text{fst } (\text{pivot-positions } A ! j) = (\text{Suc } i))$ 
      by (metis suc-i-lt all-f-in fst-conv in-set-conv-nth)
    then obtain j where jth:  $j < (\text{length } ?pp) \wedge \text{fst } (\text{pivot-positions } A ! j) = (\text{Suc } i)$ 
      by blast
    have  $j > i$ 
      using sorted-hyp
      by (metis Suc-lessD fst-i-is jth n-not-Suc-n nat-neq-iff suc-i-lt)
    have  $j > (\text{Suc } i) \implies \text{False}$ 
    proof –
      assume j-gt:  $j > (\text{Suc } i)$ 
      then have h1:  $\text{fst } (\text{pivot-positions } A ! (\text{Suc } i)) > i$ 
        using fst-i-is sorted-pivot-positions
        using sorted-hyp suc-i-lt by force
      have  $\text{fst } (\text{pivot-positions } A ! j) > \text{fst } (\text{pivot-positions } A ! (\text{Suc } i))$ 
        using jth j-gt sorted-hyp apply (auto)
        by fastforce
      then have h2:  $\text{fst } (\text{pivot-positions } A ! (\text{Suc } i)) < (\text{Suc } i)$ 
        using jth
        by simp
      show False using h1 h2
        using not-less-eq by blast
    qed
    show  $\text{fst } (\text{pivot-positions } A ! (\text{Suc } i)) = (\text{Suc } i)$ 
      using Suc-lessI  $\langle \text{Suc } i < j \implies \text{False} \rangle \langle i < j \rangle$  jth by blast
  qed
then show ?case
  by blast
qed
then show  $\text{fst } (\text{pivot-positions } A ! i) = i$ 
  using i-lt by auto
qed

```

```

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  $\text{take-cols } A (\text{map } \text{snd } (\text{pivot-positions } A)) =$ 

```

$1_m (\text{length } (\text{pivot-positions } A)) @_r$
 $0_m (\text{dim-row } A - \text{length } (\text{pivot-positions } A)) (\text{length } (\text{pivot-positions } A))$

proof –

let $?nr = \text{dim-row } A$
let $?nc = \text{dim-col } A$
have $h1: \text{dim-col}$
 $(1_m (\text{length } (\text{pivot-positions } A)) @_r$
 $0_m (\text{dim-row } A - \text{length } (\text{pivot-positions } A)) (\text{length } (\text{pivot-positions } A))) =$
 $(\text{length } (\text{pivot-positions } A))$
by (*simp add: append-rows-def*)
have $\text{len-pivot}: \text{length } (\text{pivot-positions } A) = \text{card } \{i. i < ?nr \wedge \text{row } A \ i \neq 0_v$
 $?nc\}$
using *row-ech pivot-positions(4) row-echelon-form-def* **by** *blast*
have $\text{pp-leq-nc}: \forall f. \text{pivot-fun } A \ f \ ?nc \longrightarrow (\forall i < ?nr. f \ i \leq ?nc)$ **unfolding**
pivot-fun-def
by *meson*
have $\text{pivot-set}: \exists f. \text{pivot-fun } A \ f \ ?nc \wedge \text{set } (\text{pivot-positions } A) = \{(i, f \ i) \mid i. i$
 $< ?nr \wedge f \ i \neq ?nc\}$
using *row-ech row-echelon-form-def pivot-positions(1)*
by (*smt (verit) Collect-cong carrier-matI*)
then have $\text{pivot-set-alt}: \exists f. \text{pivot-fun } A \ f \ ?nc \wedge \text{set } (\text{pivot-positions } A) = \{(i, f$
 $i) \mid i. i < ?nr \wedge \text{row } A \ i \neq 0_v \ ?nc\}$
using *pivot-positions pivot-fun-zero-row-iff Collect-cong carrier-mat-triv*
by (*smt (verit, best)*)
have $\exists f. \text{pivot-fun } A \ f \ ?nc \wedge \text{set } (\text{pivot-positions } A) = \{(i, f \ i) \mid i. f \ i \leq ?nc \wedge$
 $i < ?nr \wedge f \ i \neq ?nc\}$
using *pivot-set pp-leq-nc* **by** *auto*
then have $\text{pivot-set-var}: \exists f. \text{pivot-fun } A \ f \ ?nc \wedge \text{set } (\text{pivot-positions } A) = \{(i,$
 $f \ i) \mid i. i < ?nr \wedge f \ i < ?nc\}$
by *auto*
have $\text{length } (\text{map } \text{snd } (\text{pivot-positions } A)) = \text{card } (\text{set } (\text{map } \text{snd } (\text{pivot-positions}$
 $A)))$
using *row-ech row-echelon-form-def pivot-positions(3) distinct-card[where xs*
 $= \text{map } \text{snd } (\text{pivot-positions } A)]$
by (*metis carrier-mat-triv*)
then have $\text{length } (\text{map } \text{snd } (\text{pivot-positions } A)) = \text{card } (\text{set } (\text{pivot-positions } A))$
by (*metis card-distinct distinct-card distinct-map length-map*)
then have $\text{length } (\text{map } \text{snd } (\text{pivot-positions } A)) = \text{card } \{i. i < ?nr \wedge \text{row } A \ i$
 $\neq 0_v \ ?nc\}$
using *pivot-set-alt*
by (*simp add: len-pivot*)
then have $\text{length-asm}: \text{length } (\text{map } \text{snd } (\text{pivot-positions } A)) = \text{length } (\text{pivot-positions}$
 $A)$
using *len-pivot* **by** *linarith*
then have $\forall a. \text{List.member } (\text{map } \text{snd } (\text{pivot-positions } A)) \ a \longrightarrow a < \text{dim-col } A$
using *row-ech rref-pivot-positions*
by *auto blast*
then have $h2b: (\text{filter } (\lambda y. y < \text{dim-col } A) (\text{map } \text{snd } (\text{pivot-positions } A))) =$
 $(\text{map } \text{snd } (\text{pivot-positions } A))$

```

    by simp
  then have h2a: length (map (!) (cols A)) (filter (λy. y < dim-col A) (map snd
    (pivot-positions A)))) = length (pivot-positions A)
    using length-asm
    by (simp add: h2b)
  then have h2: length (pivot-positions A) ≤ dim-row A ⇒
    dim-col (take-cols A (map snd (pivot-positions A))) = (length (pivot-positions
    A))
    unfolding take-cols-def using mat-of-cols-carrier by auto
  have h-len: length (pivot-positions A) ≤ dim-row A ⇒
    dim-col (take-cols A (map snd (pivot-positions A))) =
    dim-col
      (1m (length (pivot-positions A)) @r
        0m (dim-row A - length (pivot-positions A)) (length (pivot-positions A)))
    using h1 h2
  by (simp add: h1 assms length-pivot-positions-dim-row)
  have h2: ∧i j. length (pivot-positions A) ≤ dim-row A ⇒
    i < dim-row A ⇒
    j < dim-col
      (1m (length (pivot-positions A)) @r
        0m (dim-row A - length (pivot-positions A)) (length (pivot-positions
    A)))) ⇒
    take-cols A (map snd (pivot-positions A)) $$ (i, j) =
    (1m (length (pivot-positions A)) @r
      0m (dim-row A - length (pivot-positions A)) (length (pivot-positions A)))
    $$
      (i, j)
  proof -
    fix i
    fix j
    let ?pp = (pivot-positions A)
    assume len-lt: length (pivot-positions A) ≤ dim-row A
    assume i-lt: i < dim-row A
    assume j-lt: j < dim-col
      (1m (length (pivot-positions A)) @r
        0m (dim-row A - length (pivot-positions A)) (length (pivot-positions
    A))))
    let ?w = ((map snd (pivot-positions A)) ! j)
    have breaking-it-down: mat-of-cols (dim-row A)
      (map (!) (cols A)) (map snd (pivot-positions A))) $$ (i, j)
      = ((cols A) ! ?w) $ i
    apply (auto)
    by (metis comp-apply h1 i-lt j-lt length-map mat-of-cols-index nth-map)
    have h1a: i < (length ?pp) ⇒ (mat-of-cols (dim-row A) (map (!) (cols A))
    (map snd (pivot-positions A))) $$ (i, j)
      = (1m (length (pivot-positions A))) $$ (i, j)
  proof -
    assume i < (length ?pp)

```

```

have  $\exists f$ . pivot-fun  $A f$  ?nc using row-ech unfolding row-echelon-form-def
  by blast
then obtain  $f$  where pivot-fun  $A f$  ?nc
  by blast
have  $j$ -nc:  $j < (\text{length } ?pp)$  using  $j$ -lt
  by (simp add: h1)
then have  $j$ -lt-nr:  $j < ?nr$  using  $dim$ -h
  using  $len$ -lt by linarith
then have is-this-true: (pivot-positions  $A$ ) !  $j = (j, f j)$ 
  using pivot-positions-form pivot-positions(1)[of A ?nr ?nc f]
proof –
  have pivot-positions  $A$  !  $j \in \text{set } (\text{pivot-positions } A)$ 
    using  $j$ -nc nth-mem by blast
  then have  $\exists n$ . pivot-positions  $A$  !  $j = (n, f n) \wedge n < \text{dim-row } A \wedge f n \neq$ 
dim-col  $A$ 
    using  $\langle \llbracket A \in \text{carrier-mat } (\text{dim-row } A) (\text{dim-col } A); \text{pivot-fun } A f (\text{dim-col } A) \rrbracket \implies \text{set } (\text{pivot-positions } A) = \{(i, f i) \mid i < \text{dim-row } A \wedge f i \neq \text{dim-col } A\} \rangle$ 
pivot-fun  $A f (\text{dim-col } A) \rangle$  by blast
  then show ?thesis
    by (metis (no-types)  $\langle \wedge A. \llbracket \text{row-echelon-form } A; \text{dim-row } A \leq \text{dim-col } A \rrbracket \implies \forall i < \text{length } (\text{pivot-positions } A). \text{fst } (\text{pivot-positions } A ! i) = i \rangle \text{dim-h fst-conv } j$ -nc row-ech)
  qed
then have  $w$ -is: ? $w = f j$ 
  by (metis h1 j-lt nth-map snd-conv)
have  $h0$ :  $i = j \longrightarrow ((\text{cols } A) ! ?w) \$ i = 1$ 
  using  $w$ -is pivot-funD(4)[of A ?nr f ?nc i]  $\langle \forall a. \text{List.member } (\text{map snd } (\text{pivot-positions } A)) a \longrightarrow a < \text{dim-col } A \rangle \langle i < \text{length } (\text{pivot-positions } A) \rangle \langle \text{pivot-fun } A f (\text{dim-col } A) \rangle$ 
cols-length i-lt length-asm
  by (auto simp add: image-iff) (metis cols-nth index-col nth-mem)
have  $h1$ :  $i \neq j \longrightarrow ((\text{cols } A) ! ?w) \$ i = 0$ 
  using  $\langle \forall a. \text{List.member } (\text{map snd } (\text{pivot-positions } A)) a \longrightarrow a < \text{dim-col } A \rangle \langle \text{pivot-fun } A f (\text{dim-col } A) \rangle$ 
h1 i-lt w-is
  by auto (smt (verit, best) cols-nth h1 index-col j-lt j-lt-nr length-asm list.set-map nth-mem pivot-fun-def)
show (mat-of-cols ( $dim$ -row  $A$ ) (map (!) (cols  $A$ )) (map snd (pivot-positions  $A$ ))) $$ ( $i, j$ )
  = ( $1_m$  (length (pivot-positions  $A$ ))) $$ ( $i, j$ ) using  $h0$   $h1$  breaking-it-down
  by (metis  $\langle i < \text{length } (\text{pivot-positions } A) \rangle h2 h-len index-one-mat(1) j-lt len-lt$ )
qed
have  $h1b$ :  $i \geq (\text{length } ?pp) \implies (\text{mat-of-cols } (\text{dim-row } A) (\text{map } (!) (\text{cols } A)) (\text{map } \text{snd } (\text{pivot-positions } A))) \$ (i, j) = 0$ 
proof –
  assume  $i$ -gt:  $i \geq (\text{length } ?pp)$ 
  have  $h0a$ :  $((\text{cols } A) ! ((\text{map } \text{snd } (\text{pivot-positions } A)) ! j)) \$ i = (\text{row } A i) \$$ 

```

?w

using $\langle \forall a. \text{List.member (map snd (pivot-positions A)) } a \longrightarrow a < \text{dim-col } A \rangle$ *h1 i-lt j-lt length-asm*

by (*auto simp add: image-iff*)
(smt (verit) cols-length h1 index-row(1) mat-of-cols-cols mat-of-cols-index nth-map nth-mem)

have *h0b*:
take-rows A [0..<length (pivot-positions A)] @_r 0_m (dim-row A - length (pivot-positions A)) (dim-col A) = A

using *assms row-echelon-form-zero-rows[of A]*

by *blast*

then have *h0c*: *(row A i) = 0_v (dim-col A)* **using** *i-gt*

by (*smt (verit, best) add-diff-cancel-left' add-diff-cancel-right' add-less-cancel-left dim-col-take-rows*
dim-row-append-rows i-lt index-zero-mat(2) index-zero-mat(3) le-Suc-ex len-lt nat-less-le nle-le row-append-rows row-zero)

then show ?thesis

using $\langle \forall a. \text{List.member (map snd (pivot-positions A)) } a \longrightarrow a < \text{dim-col } A \rangle$ *h0a breaking-it-down*
h1 j-lt length-asm

by (*auto simp add: image-iff*)
(smt (verit, del-insts) h1 index-zero-vec(1) nth-map nth-mem)

qed

have *h1*: *mat-of-cols (dim-row A)*
(map (!) (cols A) (map snd (pivot-positions A))) $\S\S$ *(i, j) =*
(1_m (length (pivot-positions A)) @_r
0_m (dim-row A - length (pivot-positions A)) (length (pivot-positions A)))
 $\S\S$

(i, j) **using** *h1a h1b*

by (*smt (verit) add-diff-inverse-nat append-rows-index diff-less-mono h1 i-lt index-one-mat(2) index-one-mat(3) index-zero-mat(1) index-zero-mat(2) index-zero-mat(3) j-lt leD len-lt not-le-imp-less*)

then show *take-cols A (map snd (pivot-positions A))* $\S\S$ *(i, j) =*
(1_m (length (pivot-positions A)) @_r
0_m (dim-row A - length (pivot-positions A)) (length (pivot-positions A)))
 $\S\S$

(i, j)

unfolding *take-cols-def*

by (*simp add: h2b*)

qed

show ?thesis

unfolding *mat-eq-iff*

using *length-pivot-positions-dim-row[OF assms(1)] h-len h2* **by** *auto*

qed

lemma *rref-right-mul*:

assumes *row-echelon-form A*

assumes *dim-col A ≥ dim-row A*

shows
 $take\text{-}cols\ A\ (map\ snd\ (pivot\text{-}positions\ A))\ * take\text{-}rows\ A\ [0..<length\ (pivot\text{-}positions\ A)] = A$
proof –
from $take\text{-}cols\text{-}pivot\text{-}eq[OF\ assms]$ **have**
 $1: take\text{-}cols\ A\ (map\ snd\ (pivot\text{-}positions\ A)) =$
 $1_m\ (length\ (pivot\text{-}positions\ A))\ @_r$
 $0_m\ (dim\text{-}row\ A - length\ (pivot\text{-}positions\ A))\ (length\ (pivot\text{-}positions\ A)) .$
have $2: take\text{-}cols\ A\ (map\ snd\ (pivot\text{-}positions\ A))\ * take\text{-}rows\ A\ [0..<length\ (pivot\text{-}positions\ A)] =$
 $take\text{-}rows\ A\ [0..<length\ (pivot\text{-}positions\ A)]\ @_r\ 0_m\ (dim\text{-}row\ A - length\ (pivot\text{-}positions\ A))\ (dim\text{-}col\ A)$
unfolding 1
apply $(simp\ add: append\text{-}rows\text{-}mat\text{-}mul)$
by $(metis\ (no\text{-}types,\ lifting)\ 1\ add\text{-}right\text{-}imp\text{-}eq\ assms\ dim\text{-}col\text{-}take\text{-}rows\ dim\text{-}row\text{-}append\text{-}rows\ dim\text{-}row\text{-}take\text{-}cols\ index\text{-}one\text{-}mat(2)\ index\text{-}zero\text{-}mat(2)\ left\text{-}mult\text{-}one\text{-}mat'\ left\text{-}mult\text{-}zero\text{-}mat'\ row\text{-}echelon\text{-}form\text{-}zero\text{-}rows)$
from $row\text{-}echelon\text{-}form\text{-}zero\text{-}rows[OF\ assms]$ **have** $... = A .$
thus $?thesis$
by $(simp\ add: 2)$
qed

context $conjugatable\text{-}vec\text{-}space$ **begin**

lemma $lin\text{-}indpt\text{-}id:$
shows $lin\text{-}indpt\ (set\ (cols\ (1_m\ n)::'a\ vec\ list))$
proof –
have $*: set\ (cols\ (1_m\ n)) = set\ (rows\ (1_m\ n))$
by $(metis\ cols\text{-}transpose\ transpose\text{-}one)$
have $det\ (1_m\ n) \neq 0$ **using** $det\text{-}one$ **by** $auto$
from $det\text{-}not\text{-}0\text{-}imp\text{-}lin\text{-}indpt\text{-}rows[OF\ \textit{this}]$
have $lin\text{-}indpt\ (set\ (rows\ (1_m\ n)))$
using $one\text{-}carrier\text{-}mat$ **by** $blast$
thus $?thesis$
by $(simp\ add: *)$
qed

lemma $lin\text{-}indpt\text{-}take\text{-}cols\text{-}id:$
shows $lin\text{-}indpt\ (set\ (cols\ (take\text{-}cols\ (1_m\ n)\ inds)))$
proof –
have $subset\text{-}h: set\ (cols\ (take\text{-}cols\ (1_m\ n)\ inds)) \subseteq set\ (cols\ (1_m\ n)::'a\ vec\ list)$
using $cols\text{-}take\text{-}cols\text{-}subset$ **by** $blast$
then show $?thesis$ **using** $lin\text{-}indpt\text{-}id\ subset\text{-}li\text{-}is\text{-}li$ **by** $auto$
qed

lemma $cols\text{-}id\text{-}unit\text{-}vecs:$
shows $cols\ (1_m\ d) = unit\text{-}vecs\ d$
unfolding $unit\text{-}vecs\text{-}def\ list\text{-}eq\text{-}iff\text{-}nth\text{-}eq$
by $auto$

lemma *distinct-cols-id*:
shows *distinct (cols (1_m d)::'a vec list)*
by (*simp add: conjugatable-vec-space.cols-id-unit-vecs vec-space.unit-vecs-distinct*)

lemma *distinct-map-nth*:
assumes *distinct ls*
assumes *distinct inds*
assumes $\bigwedge j. j \in \text{set } inds \implies j < \text{length } ls$
shows *distinct (map (!) ls) inds*
by (*simp add: assms(1) assms(2) assms(3) distinct-map inj-on-nth*)

lemma *distinct-take-cols-id*:
assumes *distinct inds*
shows *distinct (cols (take-cols (1_m n) inds) :: 'a vec list)*
unfolding *take-cols-def*
apply (*subst cols-mat-of-cols*)
apply (*auto intro!: distinct-map-nth simp add: distinct-cols-id*)
using *assms distinct-filter* **by** *blast*

lemma *rank-take-cols*:
assumes *distinct inds*
shows *rank (take-cols (1_m n) inds) = length (filter ((>) n) inds)*
apply (*subst lin-indpt-full-rank[of - length (filter ((>) n) inds)]*)
apply (*auto simp add: lin-indpt-take-cols-id*)
apply (*metis (full-types) index-one-mat(2) index-one-mat(3) length-map mat-of-cols-carrier(1) take-cols-def*)
by (*simp add: assms distinct-take-cols-id*)

lemma *rank-mul-left-invertible-mat*:
fixes *A::'a mat*
assumes *invertible-mat A*
assumes *A ∈ carrier-mat n n*
assumes *B ∈ carrier-mat n nc*
shows *rank (A * B) = rank B*
proof –
obtain *C* **where** *C: inverts-mat A C inverts-mat C A*
using *assms invertible-mat-def* **by** *blast*
from *C* **have** *ceq: C * A = 1_m n*
by (*metis assms(2) carrier-matD(2) index-mult-mat(3) index-one-mat(3) inverts-mat-def*)
then have **:B = C*A*B*
using *assms(3)* **by** *auto*
from *rank-mat-mul-left[OF assms(2–3)]*
have ***:* *rank (A*B) ≤ rank B .*
have *1:* *C ∈ carrier-mat n n* **using** *C ceq*
by (*metis assms(2) carrier-matD(1) carrier-matI index-mult-mat(3) index-one-mat(3) inverts-mat-def*)
have *2:* *A * B ∈ carrier-mat n nc* **using** *assms* **by** *auto*

have $\text{rank } B = \text{rank } (C * A * B)$ **using** * **by** *auto*
also have $\dots \leq \text{rank } (A * B)$ **using** *rank-mat-mul-left[OF 1 2]*
using 1 *assms(2) assms(3)* **by** *auto*
ultimately show *?thesis* **using** ** **by** *auto*
qed

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

proof –

have $A = A * 1_m \ n$ **using** *assms(2)* **by** *auto*
then have $\text{take-cols } A \ \text{inds} = A * \text{take-cols } (1_m \ n) \ \text{inds}$
by (*metis assms(2) one-carrier-mat take-cols-mat-mul*)
then have $\text{rank } (\text{take-cols } A \ \text{inds}) = \text{rank } (\text{take-cols } (1_m \ n) \ \text{inds})$
by (*metis assms(1) assms(2) conjugatable-vec-space.rank-mul-left-invertible-mat one-carrier-mat take-cols-carrier-mat*)
thus *?thesis*
by (*simp add: assms(3) conjugatable-vec-space.rank-take-cols*)
qed

lemma *rank-take-cols-leq*:

assumes $R:R \in \text{carrier-mat } n \ nc$
shows $\text{rank } (\text{take-cols } R \ \text{ls}) \leq \text{rank } R$
proof –
from *take-cols-mat-mul[OF R]*
have $\text{take-cols } R \ \text{ls} = R * \text{take-cols } (1_m \ nc) \ \text{ls}$
by (*metis assms one-carrier-mat right-mult-one-mat*)
thus *?thesis*
by (*metis assms one-carrier-mat take-cols-carrier-mat vec-space.rank-mat-mul-right*)
qed

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 $\text{rank } (\text{take-cols } R \ \text{ls}) \geq \text{rank } R$
by (*metis B assms(4) t vec-space.rank-mat-mul-right*)

lemma *rref-drop-pivots*:

assumes *row-ech: row-echelon-form R*
assumes *dims: R ∈ carrier-mat n nc*
assumes *order: nc ≥ n*
shows $\text{rank } (\text{take-cols } R \ (\text{map } \text{snd } (\text{pivot-positions } R))) = \text{rank } R$
proof –
let $?B = \text{take-rows } R \ [0..<\text{length } (\text{pivot-positions } R)]$

```

have equa:  $R = \text{take-cols } R (\text{map snd } (\text{pivot-positions } R)) * ?B$  using assms
rref-right-mul
by (metis carrier-matD(1) carrier-matD(2))
have ex-r:  $\exists r. \text{take-cols } R (\text{map snd } (\text{pivot-positions } R)) \in \text{carrier-mat } n \ r \wedge ?B$ 
 $\in \text{carrier-mat } r \ nc$ 
proof –
have h1:
 $\text{take-cols } R (\text{map snd } (\text{pivot-positions } R)) \in \text{carrier-mat } n \ (\text{length } (\text{pivot-positions}$ 
 $R))$ 
using assms
by (metis in-set-impl-in-set-zip2 length-map rref-pivot-positions take-cols-carrier-mat-strict
zip-map-fst-snd)
have  $\exists f. \text{pivot-fun } R \ f \ nc$  using row-ech unfolding row-echelon-form-def
using dims
by blast
then have  $\text{length } (\text{pivot-positions } R) = \text{card } \{i. i < n \wedge \text{row } R \ i \neq 0_v \ nc\}$ 
using pivot-positions[of R n nc]
using dims by auto
then have  $nc \geq \text{length } (\text{pivot-positions } R)$  using order
using carrier-matD(1) dims dual-order.trans length-pivot-positions-dim-row
row-ech by blast
then have  $\text{dim-col } R \geq \text{length } (\text{pivot-positions } R)$  using dims by auto
then have h2:  $?B \in \text{carrier-mat } (\text{length } (\text{pivot-positions } R)) \ nc$  unfolding
take-rows-def
using dims
by (smt (verit) atLeastLessThan-iff carrier-matD(2) filter-True le-eq-less-or-eq
length-map
 $\text{length-pivot-positions-dim-row less-trans map-nth mat-of-cols-carrier(1)}$ 
row-ech set-upt transpose-carrier-mat transpose-mat-of-rows)
show ?thesis using h1 h2
by blast
qed

have  $\text{rank } R \leq \text{rank } (\text{take-cols } R (\text{map snd } (\text{pivot-positions } R)))$ 
using dims ex-r rank-take-cols-geq [where  $R = R$ , where  $B = ?B$ , where  $ls$ 
 $= (\text{map snd } (\text{pivot-positions } R))$ , where  $nc = nc$ ]
using equa by blast
thus ?thesis
using assms(2) conjugatable-vec-space.rank-take-cols-leq le-antisym by blast
qed

```

lemma *gjs-and-take-cols-var*:

```

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

```

have $\forall x. \text{List.member (map snd (pivot-positions (gauss-jordan-single A))) } x \longrightarrow$
 $x \leq \text{dim-col } A$
using $A \text{ rref-pivot-positions [of } \langle \text{gauss-jordan-single } A \rangle n \text{ nc]}$
by $\text{auto (metis gauss-jordan-single(2,3) order-le-less)}$
then have $(\text{filter } (\lambda y. y < \text{dim-col } A) (\text{map snd (pivot-positions (gauss-jordan-single } A)))) =$
 $(\text{map snd (pivot-positions (gauss-jordan-single } A)))$
by $(\text{metis (no-types, lifting) } A \text{ carrier-matD(2) filter-True gauss-jordan-single(2) gauss-jordan-single(3) in-set-impl-in-set-zip2 length-map rref-pivot-positions zip-map-fst-snd})$
then show $?thesis \text{ unfolding take-cols-def take-cols-var-def}$
by simp
qed

lemma $\text{gauss-jordan-single-rank:}$

fixes $A::'a \text{ mat}$
assumes $A:A \in \text{carrier-mat } n \text{ nc}$
assumes $\text{order: } nc \geq n$
shows $\text{rank (take-cols } A (\text{map snd (pivot-positions (gauss-jordan-single } A)))) =$
 $\text{rank } A$
proof –
let $?R = \text{gauss-jordan-single } A$
obtain $P \text{ where } P:P \in \text{Units (ring-mat TYPE('a) } n \text{ undefined) and}$
 $i: ?R = P * A \text{ using gauss-jordan-transform[OF } A]$
by $(\text{metis } A \text{ carrier-matD(1) fst-eqD gauss-jordan-single-def surj-pair zero-carrier-mat})$
have $p\text{carrier: } P \in \text{carrier-mat } n \text{ n using } P \text{ unfolding Units-def}$
by $(\text{auto simp add: ring-mat-def})$
have $\text{invertible-mat } P \text{ using } P \text{ unfolding invertible-mat-def Units-def inverts-mat-def}$
apply auto
apply $(\text{simp add: ring-mat-simps(5)})$
by $(\text{metis index-mult-mat(2) index-one-mat(2) ring-mat-simps(1) ring-mat-simps(3)})$
then
obtain $P_i \text{ where } P_i: \text{invertible-mat } P_i \text{ } P_i * P = 1_m \text{ } n$
proof –
assume $a1: \bigwedge P_i. \llbracket \text{invertible-mat } P_i; P_i * P = 1_m \text{ } n \rrbracket \implies \text{thesis}$
have $\text{dim-row } P = n$
by $(\text{metis (no-types) } A \text{ assms(1) carrier-matD(1) gauss-jordan-single(2) } i \text{ index-mult-mat(2)})$
then show $?thesis$
using $a1 \text{ by (metis (no-types) } \langle \text{invertible-mat } P \rangle \text{ index-mult-mat(3) index-one-mat(3) invertible-mat-def inverts-mat-def square-mat.simps)}$
qed
then have $p_i\text{carrier: } P_i \in \text{carrier-mat } n \text{ n}$
by $(\text{metis carrier-mat-triv index-mult-mat(2) index-one-mat(2) invertible-mat-def square-mat.simps})$
have $R1:\text{row-echelon-form } ?R$
using $\text{assms(2) gauss-jordan-single(3) by blast}$
have $R2: ?R \in \text{carrier-mat } n \text{ nc}$
using $A \text{ assms(2) gauss-jordan-single(2) by auto}$
have $Rcm: \text{take-cols } ?R (\text{map snd (pivot-positions } ?R))$

```

    ∈ carrier-mat n (length (map snd (pivot-positions ?R)))
  apply (rule take-cols-carrier-mat-strict[OF R2])
  using rref-pivot-positions[OF R1 R2] by auto
  have Pi * ?R = A using i Pi
    by (smt (verit, best) A assoc-mult-mat left-mult-one-mat pcarrier pi-carrier)
  then have rank (take-cols A (map snd (pivot-positions ?R))) = rank (take-cols
(Pi * ?R) (map snd (pivot-positions ?R)))
    by auto
  also have ... = rank ( Pi * take-cols ?R (map snd (pivot-positions ?R)))
    by (metis A gauss-jordan-single(2) pi-carrier take-cols-mat-mul)
  also have ... = rank (take-cols ?R (map snd (pivot-positions ?R)))
    by (intro rank-mul-left-invertible-mat[OF Pi(1) pi-carrier Rcm])
  also have ... = rank ?R
    using assms(2) conjugatable-vec-space.rref-drop-pivots gauss-jordan-single(3)
    using R1 R2 by blast
  ultimately show ?thesis
    using A ⟨P ∈ carrier-mat n n⟩ ⟨invertible-mat P⟩ conjugatable-vec-space.rank-mul-left-invertible-mat
i
    by auto
qed

```

```

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 -
  have det A ≠ 0
    using assms(1) inv invertible-det by blast
  then have lin-indpt (set (rows AT))
    using assms(1) idom-vec.lin-dep-cols-imp-det-0 by auto
  thus ?thesis using subset-li-is-li assms(3)
    by auto
qed

```

```

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 -
  let ?B-mat = (mat-of-cols n B)
  have h1: lin-indpt (set(B))
    using assms lin-indpt-subset-cols[of A] by auto

```

```

have set B  $\subseteq$  carrier-vec n
  using set-sub A-square cols-dim[of A] by auto
then have cols-B: cols (mat-of-cols n B) = B using cols-mat-of-cols by auto
then have maximal (set B) ( $\lambda T. T \subseteq \text{set } (B) \wedge \text{lin-indpt } T$ ) using h1
  by (simp add: maximal-def subset-antisym)
then have h2: maximal (set B) ( $\lambda T. T \subseteq \text{set } (\text{cols } (\text{mat-of-cols } n \ B)) \wedge \text{lin-indpt } T$ )
  using cols-B by auto
have h3: rank (mat-of-cols n B) = card (set B)
  using h1 h2 rank-card-indpt[of ?B-mat]
  using mat-of-cols-carrier(1) by blast
then show ?thesis using assms distinct-card by auto
qed

end

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

```

begin

4 Setup

```

definition retrieve-polys:: 'a list  $\Rightarrow$  nat list  $\Rightarrow$  'a list
  where retrieve-polys qss index-list = (map (nth qss) index-list)

```

```

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

```

```

definition construct-rhs-vector:: real poly  $\Rightarrow$  real poly list  $\Rightarrow$  nat list list  $\Rightarrow$  rat vec
  where construct-rhs-vector p qs Is = vec-of-list (map ( $\lambda I. (\text{construct-NofI } p (\text{retrieve-polys } qs \ I))) \ Is$ )

```

5 Base Case

```

definition base-case-info:: (rat mat  $\times$  (nat list list  $\times$  rat list list))
  where base-case-info =
    ((mat-of-rows-list 2 [[1,1], [1,-1]], ([[],[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*:: $\text{nat} \Rightarrow 'a::\{\text{one}, \text{zero}\} \text{mat option} \Rightarrow 'a \text{ mat}$
where *matr-option* *dimen* *None* = 1_m dimen
| *matr-option* *dimen* (*Some* *c*) = *c*

definition *mat-equal*:: $'a:: \text{field mat} \Rightarrow 'a :: \text{field mat} \Rightarrow \text{bool}$
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* :: $'a :: \text{field mat} \Rightarrow 'a \text{ mat option}$ **where**
mat-inverse-var *A* = (if *dim-row* *A* = *dim-col* *A* then
let *one* = $1_m (\text{dim-row } A)$ 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*:: $\text{real poly} \Rightarrow \text{real poly list} \Rightarrow \text{nat list list} \Rightarrow \text{rat mat} \Rightarrow \text{rat}$
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*:: $\text{nat} \Rightarrow \text{nat list list} \Rightarrow \text{nat list list} \Rightarrow \text{nat list list}$
where *subsets-smash* *n s1 s2* = *concat* (*map* ($\lambda l1. \text{map} (\lambda l2. l1 @ (\text{map} ((+)$
n) *l2*)) *s2*) *s1*)

definition *signs-smash*:: $'a \text{ list list} \Rightarrow 'a \text{ list list} \Rightarrow 'a \text{ list list}$
where *signs-smash* *s1 s2* = *concat* (*map* ($\lambda l1. \text{map} (\lambda l2. l1 @ l2)$ *s2*) *s1*)

definition *smash-systems*:: $\text{real poly} \Rightarrow \text{real poly list} \Rightarrow \text{real poly list} \Rightarrow \text{nat list list}$
 $\Rightarrow \text{nat list list} \Rightarrow$
 $\text{rat list list} \Rightarrow \text{rat list list} \Rightarrow \text{rat mat} \Rightarrow \text{rat mat} \Rightarrow$
 $\text{real poly list} \times (\text{rat mat} \times (\text{nat list list} \times \text{rat list list}))$
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*:: $\text{real poly} \Rightarrow (\text{real poly list} \times (\text{rat mat} \times (\text{nat list list} \times \text{rat}$
list list))) $\Rightarrow (\text{real poly list} \times (\text{rat mat} \times (\text{nat list list} \times \text{rat list list})))$
 $\Rightarrow (\text{real poly list} \times (\text{rat mat} \times (\text{nat list list} \times \text{rat list list})))$
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*:: $\text{rat vec} \Rightarrow \text{nat list}$

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

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

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

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

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

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

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

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

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

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

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

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

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*:: $\text{real poly} \Rightarrow (\text{real poly list} \times (\text{rat mat} \times (\text{nat list list} \times \text{rat list list}))) \Rightarrow (\text{rat mat} \times (\text{nat list list} \times \text{rat list list}))$

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*:: $\text{real poly} \Rightarrow \text{real poly list} \Rightarrow (\text{rat mat} \times (\text{nat list list} \times \text{rat list list}))$

where
calculate-data *p* *qs* =
(*let* *len* = *length* *qs* *in*
if *len* = 0 *then*

```

      (λ(a,b,c).(a,b,map (drop 1) c)) (reduce-system p ([1],base-case-info))
    else if len ≤ 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* ⇒ *real poly list* ⇒ *rat list list*
where [code]:

```

  find-consistent-signs-at-roots p qs =
  ( let (M,S,Σ) = calculate-data p qs in Σ )

```

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

```

  shows find-consistent-signs-at-roots p qs = snd (snd (calculate-data p qs))
  by (simp add: case-prod-beta find-consistent-signs-at-roots-def)

```

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 ≠ 0*

assumes *q ≠ 0*

shows *changes-R-smods p (pderiv p) = card {x. poly p x = 0 ∧ poly q x > 0}*
+ *card {x. poly p x = 0 ∧ poly q x < 0}*

proof –

have 1: *{x. poly p x = 0 ∧ poly q x = 0} = {}*

using *assms(1) coprime-poly-0* **by** *auto*

have 2: *changes-R-smods p (pderiv p) = int (card {x . poly p x = 0})* **using**
sturm-R **by** *auto*

have 3: *{x. poly p x = 0 ∧ poly q x > 0} ∩ {x. poly p x = 0 ∧ poly q x < 0}*
= *{}* **by** *auto*

have *{x . poly p x = 0} = {x. poly p x = 0 ∧ poly q x > 0} ∪ {x. poly p x = 0*
∧ *poly q x < 0} ∪ {x. poly p x = 0 ∧ poly q x = 0}* **by** *force*

then have *{x . poly p x = 0} = {x. poly p x = 0 ∧ poly q x > 0} ∪ {x. poly p*
x = 0 ∧ poly q x < 0} **using** 1 **by** *auto*

then have *(card {x . poly p x = 0}) = (card ({x. poly p x = 0 ∧ poly q x > 0}*
∪ *{x. poly p x = 0 ∧ poly q x < 0}))* **by** *presburger*

then have 4: *(card {x . poly p x = 0}) = card {x. poly p x = 0 ∧ poly q x > 0}*

+ $\text{card } \{x. \text{poly } p \ x = 0 \wedge \text{poly } q \ x < 0\}$ **using** \mathcal{B} **by** (*simp add: card-Un-disjoint*
assms(2) poly-roots-finite)
show *?thesis* **by** (*simp add: 2 4*)
qed

lemma *card-eq-const-sum*:

fixes $k:: \text{real}$
assumes *finite A*
shows $k * \text{card } A = \text{sum } (\lambda x. k) \ A$
proof –
have $\text{plus} \circ (\lambda -. \text{Suc } 0) = (\lambda -. \text{Suc})$
by (*simp add: fun-eq-iff*)
then have $\text{Finite-Set.fold } (\text{plus} \circ (\lambda -. \text{Suc } 0)) = \text{Finite-Set.fold } (\lambda -. \text{Suc})$
by (*rule arg-cong*)
then have $\text{Finite-Set.fold } (\text{plus} \circ (\lambda -. \text{Suc } 0)) \ 0 \ A = \text{Finite-Set.fold } (\lambda -. \text{Suc}) \ 0 \ A$
by (*blast intro: fun-cong*)
then show *?thesis*
by (*simp add: card.eq-fold sum.eq-fold*)
qed

lemma *restate-tarski*:

fixes $q:: \text{real poly}$
assumes *coprime p q*
assumes $p \neq 0$
assumes $q \neq 0$
shows $\text{changes-R-smods } p \ ((\text{pderiv } p) * q) = \text{card } \{x. \text{poly } p \ x = 0 \wedge \text{poly } q \ x > 0\} - \text{int}(\text{card } \{x. \text{poly } p \ x = 0 \wedge \text{poly } q \ x < 0\})$
proof –
have \mathcal{B} : $\text{taq } \{x. \text{poly } p \ x = 0\} \ q \equiv \sum y \in \{x. \text{poly } p \ x = 0\}. \text{ Sturm-Tarski.sign } (\text{poly } q \ y)$ **by** (*simp add: taq-def*)
have $\mathcal{4}$: $\{x. \text{poly } p \ x = 0\} = \{x. \text{poly } p \ x = 0 \wedge \text{poly } q \ x > 0\} \cup \{x. \text{poly } p \ x = 0 \wedge \text{poly } q \ x < 0\} \cup \{x. \text{poly } p \ x = 0 \wedge \text{poly } q \ x = 0\}$ **by** *force*
then have $\mathcal{5}$: $\{x. \text{poly } p \ x = 0\} = \{x. \text{poly } p \ x = 0 \wedge \text{poly } q \ x > 0\} \cup \{x. \text{poly } p \ x = 0 \wedge \text{poly } q \ x < 0\}$ **using** *assms(1) coprime-poly-0* **by** *auto*
then have $\mathcal{6}$: $\sum y \in \{x. \text{poly } p \ x = 0\}. \text{ Sturm-Tarski.sign } (\text{poly } q \ y) \equiv \sum y \in \{x. \text{poly } p \ x = 0 \wedge \text{poly } q \ x > 0\} \cup \{x. \text{poly } p \ x = 0 \wedge \text{poly } q \ x < 0\}. \text{ Sturm-Tarski.sign } (\text{poly } q \ y)$ **by** *presburger*
then have $\mathcal{12}$: $\text{taq } \{x. \text{poly } p \ x = 0\} \ q \equiv \sum y \in \{x. \text{poly } p \ x = 0 \wedge \text{poly } q \ x > 0\} \cup \{x. \text{poly } p \ x = 0 \wedge \text{poly } q \ x < 0\}. \text{ Sturm-Tarski.sign } (\text{poly } q \ y)$
by (*simp add: 3*)
have $\mathcal{7}$: $\{x. \text{poly } p \ x = 0 \wedge \text{poly } q \ x > 0\} \cap \{x. \text{poly } p \ x = 0 \wedge \text{poly } q \ x < 0\} = \{\}$ **by** *auto*
then have $\mathcal{8}$: $\sum y \in \{x. \text{poly } p \ x = 0 \wedge \text{poly } q \ x > 0\} \cup \{x. \text{poly } p \ x = 0 \wedge \text{poly } q \ x < 0\}. \text{ Sturm-Tarski.sign } (\text{poly } q \ y) \equiv (\sum y \in \{x. \text{poly } p \ x = 0 \wedge \text{poly } q \ x > 0\}. \text{ Sturm-Tarski.sign } (\text{poly } q \ y)) + (\sum y \in \{x. \text{poly } p \ x = 0 \wedge \text{poly } q \ x < 0\}. \text{ Sturm-Tarski.sign } (\text{poly } q \ y))$ **by** (*simp add: assms(2) poly-roots-finite sum.union-disjoint*)

then have 13: $\text{taq } \{x. \text{poly } p \ x=0\} \ q \equiv (\sum y \in \{x. \text{poly } p \ x = 0 \wedge \text{poly } q \ x > 0\}. \text{Sturm-Tarski.sign } (\text{poly } q \ y)) + (\sum y \in \{x. \text{poly } p \ x = 0 \wedge \text{poly } q \ x < 0\}. \text{Sturm-Tarski.sign}(\text{poly } q \ y))$
by (*simp add: 12*)
then have 9: $\text{taq } \{x. \text{poly } p \ x = 0\} \ q \equiv (\sum y \in \{x. \text{poly } p \ x = 0 \wedge \text{poly } q \ x > 0\}. 1) + (\sum y \in \{x. \text{poly } p \ x = 0 \wedge \text{poly } q \ x < 0\}. (-1))$ **by** *simp*
have 10: $(\sum y \in \{x. \text{poly } p \ x = 0 \wedge \text{poly } q \ x > 0\}. 1) = \text{card } \{x. \text{poly } p \ x = 0 \wedge \text{poly } q \ x > 0\}$ **using** *card-eq-sum* **by** *auto*
have 11: $(\sum y \in \{x. \text{poly } p \ x = 0 \wedge \text{poly } q \ x < 0\}. (-1)) = -1 * \text{card } \{x. \text{poly } p \ x = 0 \wedge \text{poly } q \ x < 0\}$ **using** *card-eq-const-sum* **by** *simp*
have 14: $\text{taq } \{x. \text{poly } p \ x = 0\} \ q \equiv \text{card } \{x. \text{poly } p \ x = 0 \wedge \text{poly } q \ x > 0\} + -1 * \text{card } \{x. \text{poly } p \ x = 0 \wedge \text{poly } q \ x < 0\}$ **using** 9 10 11 **by** *simp*
have 1: $\text{changes-R-smods } p \ (\text{pderiv } p * q) = \text{taq } \{x. \text{poly } p \ x=0\} \ q$ **using** *sturm-tarski-R* **by** *simp*
then have 15: $\text{changes-R-smods } p \ (\text{pderiv } p * q) = \text{card } \{x. \text{poly } p \ x = 0 \wedge \text{poly } q \ x > 0\} + (-1 * \text{card } \{x. \text{poly } p \ x = 0 \wedge \text{poly } q \ x < 0\})$ **using** 14 **by** *linarith*
have 16: $(-1 * \text{card } \{x. \text{poly } p \ x = 0 \wedge \text{poly } q \ x < 0\}) = - \text{card } \{x. \text{poly } p \ x = 0 \wedge \text{poly } q \ x < 0\}$ **by** *auto*
then show *?thesis* **using** 15 **by** *linarith*
qed

lemma *restate-tarski2:*

fixes $q::\text{real poly}$
assumes $p \neq 0$
shows $\text{changes-R-smods } p \ ((\text{pderiv } p) * q) = \text{int}(\text{card } \{x. \text{poly } p \ x = 0 \wedge \text{poly } q \ x > 0\}) - \text{int}(\text{card } \{x. \text{poly } p \ x = 0 \wedge \text{poly } q \ x < 0\})$
unfolding *sturm-tarski-R[symmetric]* *taq-def*
proof –
let $?all = \{x. \text{poly } p \ x=0\}$
let $?lt = \{x. \text{poly } p \ x=0 \wedge \text{poly } q \ x < 0\}$
let $?gt = \{x. \text{poly } p \ x=0 \wedge \text{poly } q \ x > 0\}$
let $?eq = \{x. \text{poly } p \ x=0 \wedge \text{poly } q \ x = 0\}$
have $\text{eq: } ?all = ?lt \cup ?gt \cup ?eq$ **by** *force*
from *poly-roots-finite[OF assms]* **have** $\text{fin: finite } ?all$.
show $(\sum x \mid \text{poly } p \ x = 0. \text{Sturm-Tarski.sign } (\text{poly } q \ x)) = \text{int}(\text{card } ?gt) - \text{int}(\text{card } ?lt)$
unfolding *eq*
apply (*subst sum-Un*)
apply (*auto simp add:fin*)
apply (*subst sum-Un*)
by (*auto simp add:fin*)
qed

lemma *coprime-set-prod:*

fixes $I::\text{real poly set}$
shows $\text{finite } I \implies ((\forall q \in I. (\text{coprime } p \ q)) \longrightarrow (\text{coprime } p \ (\prod I)))$
proof (*induct rule: finite-induct*)
case empty

```

    then show ?case
      by simp
next
  case (insert x F)
  then show ?case using coprime-mult-right-iff
    by simp
qed

lemma finite-nonzero-set-prod:
  fixes I:: real poly set
  shows nonzero-hyp: finite I  $\implies$   $((\forall q \in I. q \neq 0) \longrightarrow \prod I \neq 0)$ 
proof (induct rule: finite-induct)
  case empty
  then show ?case
    by simp
next
  case (insert x F)
  have h:  $\prod (insert\ x\ F) = x * (\prod F)$ 
    by (simp add: insert.hyps(1) insert.hyps(2))
  have h-xin:  $x \in insert\ x\ F$ 
    by simp
  have hq:  $(\forall q \in (insert\ x\ F). q \neq 0) \longrightarrow x \neq 0$  using h-xin
    by blast
  show ?case using h hq
    using insert.hyps(3) by auto
qed

```

10 Setting up the construction: Definitions

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

lemma *construct-NoFI-prop*:
 fixes *p*:: real poly
 fixes *I*:: real poly list
 assumes nonzero: $p \neq 0$
 shows *construct-NoFI* $p\ I =$
 $rat-of-int (int (card \{x. poly\ p\ x = 0 \wedge poly (prod-list\ I)\ x > 0\}) -$
 $int (card \{x. poly\ p\ x = 0 \wedge poly (prod-list\ I)\ x < 0\}))$
unfolding *construct-NoFI-def*
using *assms restate-tarski2 nonzero rsquarefree-def*
by (*simp add: rsquarefree-def*)

definition *construct-s-vector*:: real poly \Rightarrow real poly list list \Rightarrow rat vec
 where *construct-s-vector* $p\ Is = vec-of-list (map (\lambda I.(construct-NoFI\ p\ I))\ Is)$

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 –

have $\forall q \in set(qs). \forall x \in set(\text{characterize-root-list-p } p). poly\ q\ x \neq 0$

using pairwise-rel-prime

using coprime-poly-0 nonzero poly-roots-finite *characterize-root-list-p-def* **by** *fastforce*

then have $h: \forall q \in set(qs). \forall x \in set(\text{characterize-root-list-p } p). \text{squash } (poly\ q\ x) =$ (if (poly $q\ x > 0$) then (1::rat) else (-1::rat))

by (*simp add: squash-def*)

have map ($\lambda r. map (\lambda p. \text{if } 0 < poly\ p\ r \text{ then } 1 \text{ else } -1) qs$) (*characterize-root-list-p* p) = map ($\lambda r. map (\text{squash} \circ (\lambda p. poly\ p\ r)) qs$) (*characterize-root-list-p* p)

by (*simp add: h*)

thus *thesis* **unfolding** *characterize-consistent-signs-at-roots-def* *characterize-consistent-signs-at-roots-copr-def* *signs-at-def* *consistent-sign-vec-copr-def*

by *presburger*

qed

definition *list-constr*:: $\text{nat list} \Rightarrow \text{nat} \Rightarrow \text{bool}$
where *list-constr* $L n \equiv \text{list-all } (\lambda x. x < n) L$

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

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

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

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

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

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

proof –

have *h0*: $(\bigwedge i j. i < \text{length } \text{subsets} \implies$
 $j < \text{length } \text{signs} \implies$
 $\text{map } (\lambda \text{index-list}. \text{map } (z \text{ index-list}) \text{signs}) \text{subsets} ! i ! j = z (\text{subsets} !$
 $i) (\text{signs} ! j))$

proof –

fix *i*

fix *j*

assume *i-lt*: $i < \text{length } \text{subsets}$

assume *j-lt*: $j < \text{length } \text{signs}$

show $((\text{map } (\lambda \text{index-list}. \text{map } (z \text{ index-list}) \text{signs}) \text{subsets}) ! i) ! j = z (\text{subsets} !$
 $i) (\text{signs} ! j)$

proof –

have *h0*: $(\text{map } (\lambda \text{index-list}. \text{map } (z \text{ index-list}) \text{signs}) \text{subsets}) ! i = \text{map } (z$
 $(\text{subsets} ! i)) \text{signs}$

using *nth-map i-lt*

by *blast*

then show *?thesis* **using** *nth-map j-lt*

by *simp*

qed

qed

have *h*: $\text{mat } (\text{length } \text{subsets}) (\text{length } \text{signs}) (\lambda(i, j). z (\text{subsets} ! i) (\text{signs} ! j)) =$
 $\text{mat } (\text{length } \text{subsets}) (\text{length } \text{signs}) (\lambda(i, y). \text{map } (\lambda \text{index-list}. \text{map } (z \text{ index-list})$
 $\text{signs}) \text{subsets} ! i ! y)$

using *h0 eq-matI* **where** $A = \text{mat } (\text{length } \text{subsets}) (\text{length } \text{signs}) (\lambda(i, j). z$
 $(\text{subsets} ! i) (\text{signs} ! j)),$

where $B = \text{mat } (\text{length } \text{subsets}) (\text{length } \text{signs}) (\lambda(i, y). \text{map } (\lambda \text{index-list}.$

$\text{map } (z \text{ index-list } \text{signs } \text{subsets } ! i ! y)]$
by *auto*
show *?thesis unfolding alt-matrix-A-def matrix-A-def mat-of-rows-list-def* **ap-**
ply (*auto*) **unfolding** *mtx-row-def*
using *h* **by** *blast*
qed

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

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

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

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

11 Setting up the construction: Proofs

lemma *row-mat-of-rows-list*:
assumes *list-all* $(\lambda r. \text{length } r = \text{nc}) \text{rs}$
assumes $i < \text{length } \text{rs}$
shows $\text{row } (\text{mat-of-rows-list } \text{nc } \text{rs}) i = \text{vec-of-list } (\text{nth } \text{rs } i)$
by (*smt assms(1) assms(2) dim-col-mat(1) dim-vec-of-list eq-vecI index-row(2) index-vec list-all-length mat-of-rows-list-def row-mat split-conv vec-of-list-index*)

lemma *mult-mat-vec-of-list*:
assumes $\text{length } \text{ls} = \text{nc}$
assumes *list-all* $(\lambda r. \text{length } r = \text{nc}) \text{rs}$
shows $\text{mat-of-rows-list } \text{nc } \text{rs} *_v \text{vec-of-list } \text{ls} = \text{vec-of-list } (\text{map } (\lambda r. \text{vec-of-list } r \cdot \text{vec-of-list } \text{ls}) \text{rs})$
unfolding *mult-mat-vec-def*
using *row-mat-of-rows-list assms*
apply *auto*
by (*smt dim-row-mat(1) dim-vec dim-vec-of-list eq-vecI index-map-vec(1) index-map-vec(2) index-vec list-all-length mat-of-rows-list-def row-mat-of-rows-list vec-of-list-index*)

lemma *mtx-row-length*:
list-all ($\lambda r. \text{length } r = \text{length } \text{signs}$) (*map* (*mtx-row signs*) *ls*)
apply (*induction ls*)
by (*auto simp add: mtx-row-def*)

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 –
from *poly-roots-finite[OF assms(1)]* **have** *finite* $\{x. \text{poly } p \ x = 0\}$.
then have *eq*: (*Collect*
 $((\lambda v. v = \text{signs } ! i) \circ$
 $\text{consistent-sign-vec-copr } qs) \cap$
 $\text{set } (\text{sorted-list-of-set}$
 $\{x. \text{poly } p \ x = 0\})$) =
 $\{x. \text{poly } p \ x = 0 \wedge \text{consistent-sign-vec-copr } qs \ x = \text{signs } ! i\}$
by *auto*
show *?thesis*
unfolding *construct-lhs-vector-def vec-of-list-index characterize-root-list-p-def*
apply *auto*
apply (*subst nth-map[OF assms(2)]*)
apply *auto*
apply (*subst distinct-length-filter*)
using *eq* **by** *auto*

qed

lemma *construct-lhs-vector-cleaner*:
assumes $p \neq 0$
shows (*construct-lhs-vector p qs signs*) =
 $\text{vec-of-list } (\text{map } (\lambda s. \text{rat-of-int } (\text{card } \{x. \text{poly } p \ x = 0 \wedge ((\text{consistent-sign-vec-copr } qs \ x) = s)\})) \ \text{signs})$
apply (*rule eq-vecI*)
apply (*auto simp add: construct-lhs-vector-clean[OF assms]*)
apply (*simp add: vec-of-list-index*)
unfolding *construct-lhs-vector-def*
using *assms construct-lhs-vector-clean construct-lhs-vector-def* **apply** *auto[1]*
by *simp*

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 signs = 1) ∨ (z I signs = -1) using assms
proof (induction I)
  case Nil
  then show ?case
    by (auto simp add:z-def)
next
  case (Cons a I)
  moreover have signs ! a = 1 ∨ signs ! a = -1
    by (metis (mono-tags, lifting) add-Suc-right calculation(2) calculation(3) gr0-conv-Suc
list.size(4) list-all-length nth-Cons-0)
  ultimately show ?case
    by (auto simp add:z-def)
qed

```

lemma *z-lemma*:

```

fixes I:: nat list
fixes sign:: rat list
assumes consistent: sign ∈ set (characterize-consistent-signs-at-roots-copr p qs)
assumes welldefined: list-constr I (length qs)
shows (z I sign = 1) ∨ (z I sign = -1)
proof (rule z-signs)
  have length sign = length qs using consistent
  by (auto simp add: characterize-consistent-signs-at-roots-copr-def consistent-sign-vec-copr-def)
  thus list-all (λi. i < length sign) I
    using welldefined
  by (auto simp add: list-constr-def characterize-consistent-signs-at-roots-copr-def
consistent-sign-vec-copr-def)
  show list-all (λs. s = 1 ∨ s = - 1) sign using consistent
  apply (auto simp add: list.pred-map characterize-consistent-signs-at-roots-copr-def
consistent-sign-vec-copr-def)
  using Ball-set
  by force
qed

```

lemma *in-set*:

```

fixes p:: real poly
assumes nonzero: p ≠ 0
fixes qs:: real poly list
fixes I:: nat list
fixes sign:: rat list
fixes x:: real
assumes root-p: x ∈ {x. poly p x = 0}
assumes sign-fix: sign = consistent-sign-vec-copr qs x
assumes welldefined: list-constr I (length qs)
shows sign ∈ set (characterize-consistent-signs-at-roots-copr p qs)
proof –
  have h1: consistent-sign-vec-copr qs x ∈
    set (remdups (map (consistent-sign-vec-copr qs) (sorted-list-of-set {x. poly p x

```

```

= 0})))
  using root-p apply auto apply (subst set-sorted-list-of-set)
  using nonzero poly-roots-finite rsquarefree-def apply blast by auto
  thus ?thesis unfolding characterize-consistent-signs-at-roots-copr-def character-
ize-root-list-p-def using sign-fix
    by blast
qed

```

```

lemma nonzero-product:
  fixes p:: real poly
  assumes nonzero: p≠0
  fixes qs:: real poly list
  assumes pairwise-rel-prime-1:  $\bigwedge q. q \in \text{set } qs \implies \text{coprime } p \ q$ 
  fixes I:: nat list
  fixes x:: real
  assumes root-p:  $x \in \{x. \text{poly } p \ x = 0\}$ 
  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 -
  have  $\bigwedge x. x \in \text{set } (\text{retrieve-polys } qs \ I) \implies \text{coprime } p \ x$ 
    using welldefined
  by (auto simp add: retrieve-polys-def list-constr-def list-all-iff intro: pairwise-rel-prime-1)
  then have coprimeh: coprime p (prod-list (retrieve-polys qs I))
    using prod-list-coprime-right by auto
  thus ?thesis using root-p
    using coprime-poly-0 linorder-neqE-linordered-idom by blast
qed

```

```

lemma horiz-vector-helper-pos-ind:
  fixes p:: real poly
  assumes nonzero: p≠0
  fixes qs:: real poly list
  assumes pairwise-rel-prime-1:  $\forall q. ((\text{List.member } qs \ q) \longrightarrow (\text{coprime } p \ q))$ 
  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: 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 (induct I)
  case Nil
  then show ?case
    by (simp add: retrieve-polys-def z-def)
next
  case (Cons a I)

```

```

have welldef: list-constr (a#I) (length qs) → (list-constr I (length qs))
unfolding list-constr-def list-all-def by auto
have set-hyp: list-constr I (length qs) → sign ∈ set (characterize-consistent-signs-at-roots-copr
p qs)
using in-set using nonzero root-p sign-fix by blast
have z-hyp: list-constr I (length qs) → ((z I sign = 1) ∨ (z I sign = -1))
using set-hyp z-lemma[where sign=sign, where I = I, where p=p, where
qs=qs] by blast
have sign-hyp: sign = map (λ q. if (poly q x > 0) then 1 else -1) qs
using sign-fix unfolding consistent-sign-vec-copr-def by blast
have ind-hyp-1: list-constr (a#I) (length qs) →
((0 < poly (prod-list (retrieve-polys qs I)) x) = (z I sign = 1))
using welldef Cons.hyps by auto
have ind-hyp-2: list-constr (a#I) (length qs) →
((0 > poly (prod-list (retrieve-polys qs I)) x) = (z I sign = -1))
using welldef z-hyp Cons.hyps nonzero-product
using pairwise-rel-prime-1 nonzero root-p by auto
have h1: prod-list (retrieve-polys qs (a # I)) = (nth qs a)*(prod-list (retrieve-polys
qs I))
by (simp add: retrieve-polys-def)
have h2: (z (a # I) sign) = (nth sign a)*(z I sign)
by (metis (mono-tags, opaque-lifting) list.simps(9) prod-list.Cons z-def)
have h3help: list-constr (a#I) (length qs) → a < length qs unfolding list-constr-def
by simp
then have h3: list-constr (a#I) (length qs) →
((nth sign a) = (if (poly (nth qs a) x > 0) then 1 else -1))
using nth-map sign-hyp by auto
have h2: (0 < poly ((nth qs a)*(prod-list (retrieve-polys qs I))) x) ↔
((0 < poly (nth qs a) x ∧ (0 < poly (prod-list (retrieve-polys qs I)) x)) ∨
(0 > poly (nth qs a) x ∧ (0 > poly (prod-list (retrieve-polys qs I)) x)))
by (simp add: zero-less-mult-iff)
have final-hyp-a: list-constr (a#I) (length qs) → (((0 < poly (nth qs a) x ∧ (0
< poly (prod-list (retrieve-polys qs I)) x))
∨ (0 > poly (nth qs a) x ∧ (0 > poly (prod-list (retrieve-polys qs I)) x))) =
((nth sign a)*(z I sign) = 1))
proof -
have extra-hyp-a: list-constr (a#I) (length qs) → (0 < poly (nth qs a) x =
((nth sign a) = 1)) using h3
by simp
have extra-hyp-b: list-constr (a#I) (length qs) → (0 > poly (nth qs a) x =
((nth sign a) = -1))
using h3 apply (auto) using coprime-poly-0 h3help nth-mem pairwise-rel-prime-1
root-p by fastforce
have ind-hyp-1: list-constr (a#I) (length qs) → (((0 < poly (nth qs a) x ∧
(z I sign = 1)) ∨
(0 > poly (nth qs a) x ∧ (z I sign = -1)))
= ((nth sign a)*(z I sign) = 1)) using extra-hyp-a extra-hyp-b
using zmult-eq-1-iff
by (simp add: h3)

```

```

    then show ?thesis
      using ind-hyp-1 ind-hyp-2 by (simp add: Cons.hyps welldef)
    qed
    then show ?case
      using h1 z-def by (simp add: zero-less-mult-iff)
    qed

```

```

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))  $x > 0$ )  $\longleftrightarrow (z\ I\ sign = 1)$ 
  using horiz-vector-helper-pos-ind
  using pairwise-rel-prime-1 nonzero root-p sign-fix welldefined by blast

```

```

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:  $x \in \{x. poly\ p\ x = 0\}$ 
  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 -
  have set-hyp: list-constr I (length qs)  $\longrightarrow sign \in set (characterize-consistent-signs-at-roots-copr\ p\ qs)$ 
  using in-set using nonzero root-p sign-fix by blast
  have z-hyp: list-constr I (length qs)  $\longrightarrow ((z\ I\ sign = 1) \vee (z\ I\ sign = -1))$ 
  using set-hyp z-lemma[where sign=sign, where I = I, where p=p, where qs=qs] by blast
  have poly-hyp: (poly (prod-list (retrieve-polys qs I))  $x > 0$ )  $\vee$  (poly (prod-list (retrieve-polys qs I))  $x < 0$ )
  using nonzero-product
  using pairwise-rel-prime-1 nonzero root-p
  using welldefined by auto
  have pos-hyp: (poly (prod-list (retrieve-polys qs I))  $x > 0$ )  $\longleftrightarrow (z\ I\ sign = 1)$ 
using horiz-vector-helper-pos
  using pairwise-rel-prime-1 nonzero root-p sign-fix welldefined by blast

```

```

show ?thesis using z-hyp poly-hyp pos-hyp apply (auto)
using welldefined by blast
qed

```

```

lemma vec-of-list-dot-rewrite:
assumes length xs = length ys
shows vec-of-list xs • vec-of-list ys =
  sum-list (map2 (*) xs ys)
using assms
proof (induction xs arbitrary:ys)
  case Nil
  then show ?case by auto
next
  case (Cons a xs)
  then show ?case apply auto
  by (smt (verit, best) Suc-length-conv list.simps(9) old.prod.case scalar-prod-vCons
sum-list.Cons vec-of-list-Cons zip-Cons-Cons)
qed

```

```

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≠0
shows
  (vec-of-list (mtx-row signs I) • (construct-lhs-vector p qs signs)) =
  sum-list (map (λs. (z I s) * rat-of-int (card {x. poly p x = 0 ∧ consis-
tent-sign-vec-copr qs x = s})) signs)
proof –
  have p ≠ 0 using nonzero by auto
  from construct-lhs-vector-cleaner[OF this]
  have rhseq: construct-lhs-vector p qs signs =
    vec-of-list
      (map (λs. rat-of-int (card {x. poly p x = 0 ∧ consistent-sign-vec-copr qs x =
s})) signs) by auto
  have (vec-of-list (mtx-row signs I) • (construct-lhs-vector p qs signs)) =
    sum-list (map2 (*) (mtx-row signs I) (map (λs. rat-of-int (card {x. poly p x =
0 ∧ consistent-sign-vec-copr qs x = s})) signs))
  unfolding rhseq
  apply (intro vec-of-list-dot-rewrite)
  by (auto simp add: mtx-row-def)
  thus ?thesis unfolding mtx-row-def
  using map2-map-map
  by (auto simp add: map2-map-map)
qed

```

```

lemma sum-list-distinct-filter:

```

fixes $f:: 'a \Rightarrow int$
assumes $distinct\ xs\ distinct\ ys$
assumes $set\ ys \subseteq set\ xs$
assumes $\bigwedge x. x \in set\ xs - set\ ys \implies f\ x = 0$
shows $sum\text{-}list\ (map\ f\ xs) = sum\text{-}list\ (map\ f\ ys)$
by ($metis\ List.\text{finite-set}\ assms(1)\ assms(2)\ assms(3)\ assms(4)\ sum.\text{mono-neutral-cong-left}\ sum\text{-}list\text{-}distinct\text{-}conv\text{-}sum\text{-}set$)

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 $distinct\text{-}signs: distinct\ signs$
assumes $all\text{-}info: set\ (characterize\text{-}consistent\text{-}signs\text{-}at\text{-}roots\text{-}copr\ p\ qs) \subseteq set\ (signs)$
assumes $welldefined: list\text{-}constr\ I\ (length\ qs)$
shows
 $(vec\text{-}of\text{-}list\ (mtx\text{-}row\ signs\ I) \cdot (construct\text{-}lhs\text{-}vector\ p\ qs\ signs)) =$
 $(vec\text{-}of\text{-}list\ (mtx\text{-}row\ (characterize\text{-}consistent\text{-}signs\text{-}at\text{-}roots\text{-}copr\ p\ qs)\ I) \cdot$
 $(construct\text{-}lhs\text{-}vector\ p\ qs\ (characterize\text{-}consistent\text{-}signs\text{-}at\text{-}roots\text{-}copr\ p\ qs)))$
proof –
have $h0: \forall\ sgn. sgn \in set\ signs \wedge sgn \notin consistent\text{-}sign\text{-}vec\text{-}copr\ qs \text{ ‘ } set$
 $(characterize\text{-}root\text{-}list\text{-}p\ p) \wedge 0 < rat\text{-}of\text{-}nat\ (card$
 $\{xa. poly\ p\ xa = 0 \wedge consistent\text{-}sign\text{-}vec\text{-}copr\ qs\ xa = sgn\}) \longrightarrow z\ I$
 $sgn = 0$
proof –
have $\forall\ sgn. sgn \in set\ signs \wedge sgn \notin consistent\text{-}sign\text{-}vec\text{-}copr\ qs \text{ ‘ } set\ (characterize\text{-}root\text{-}list\text{-}p$
 $p) \wedge 0 < rat\text{-}of\text{-}int\ (card$
 $\{xa. poly\ p\ xa = 0 \wedge consistent\text{-}sign\text{-}vec\text{-}copr\ qs\ xa = sgn\}) \longrightarrow$
 $\{xa. poly\ p\ xa = 0 \wedge consistent\text{-}sign\text{-}vec\text{-}copr\ qs\ xa = sgn\} \neq \{\}$
by *fastforce*
then show *?thesis*
using *characterize-consistent-signs-at-roots-copr-def in-set nonzero welldefined*
by *auto*
qed
then have $\forall\ sgn. sgn \in set\ signs \wedge sgn \notin consistent\text{-}sign\text{-}vec\text{-}copr\ qs \text{ ‘ } set$
 $(characterize\text{-}root\text{-}list\text{-}p\ p) \longrightarrow ((0 = rat\text{-}of\text{-}nat\ (card$
 $\{xa. poly\ p\ xa = 0 \wedge consistent\text{-}sign\text{-}vec\text{-}copr\ qs\ xa = sgn\}) \vee z\ I$
 $sgn = 0))$
by *auto*
then have $hyp: \forall\ s. s \in set\ signs \wedge s \notin consistent\text{-}sign\text{-}vec\text{-}copr\ qs \text{ ‘ } set$
 $(characterize\text{-}root\text{-}list\text{-}p\ p) \longrightarrow (z\ I\ s * rat\text{-}of\text{-}nat\ (card\ \{x. poly\ p\ x = 0 \wedge consis\text{-}$
 $tent\text{-}sign\text{-}vec\text{-}copr\ qs\ x = s\}) = 0)$
by *auto*
then have $(\sum\ s \in set\ (signs). z\ I\ s * rat\text{-}of\text{-}nat\ (card\ \{x. poly\ p\ x = 0 \wedge consis\text{-}$
 $tent\text{-}sign\text{-}vec\text{-}copr\ qs\ x = s\})) =$
 $(\sum\ s \in (set\ (signs) \cap (consistent\text{-}sign\text{-}vec\text{-}copr\ qs \text{ ‘ } set\ (characterize\text{-}root\text{-}list\text{-}p$

$p))$. $z I s * \text{rat-of-nat} (\text{card} \{x. \text{poly } p x = 0 \wedge \text{consistent-sign-vec-copr } qs x = s\})$

proof –

have $\text{set}(\text{signs}) = (\text{set}(\text{signs}) \cap (\text{consistent-sign-vec-copr } qs \text{ ‘ set } (\text{characterize-root-list-p } p))) \cup$
 $(\text{set}(\text{signs}) - (\text{consistent-sign-vec-copr } qs \text{ ‘ set } (\text{characterize-root-list-p } p)))$

by *blast*

then have $\text{sum-rewrite}: (\sum s \in \text{set}(\text{signs}). z I s * \text{rat-of-nat} (\text{card} \{x. \text{poly } p x = 0 \wedge \text{consistent-sign-vec-copr } qs x = s\})) =$
 $(\sum s \in (\text{set}(\text{signs}) \cap (\text{consistent-sign-vec-copr } qs \text{ ‘ set } (\text{characterize-root-list-p } p))) \cup$
 $(\text{set}(\text{signs}) - (\text{consistent-sign-vec-copr } qs \text{ ‘ set } (\text{characterize-root-list-p } p))))$. $z I s * \text{rat-of-nat} (\text{card} \{x. \text{poly } p x = 0 \wedge \text{consistent-sign-vec-copr } qs x = s\})$

by *auto*

then have $\text{sum-split}: (\sum s \in (\text{set}(\text{signs}) \cap (\text{consistent-sign-vec-copr } qs \text{ ‘ set } (\text{characterize-root-list-p } p))) \cup$
 $(\text{set}(\text{signs}) - (\text{consistent-sign-vec-copr } qs \text{ ‘ set } (\text{characterize-root-list-p } p))))$. $z I s * \text{rat-of-nat} (\text{card} \{x. \text{poly } p x = 0 \wedge \text{consistent-sign-vec-copr } qs x = s\})$

$=$
 $(\sum s \in (\text{set}(\text{signs}) \cap (\text{consistent-sign-vec-copr } qs \text{ ‘ set } (\text{characterize-root-list-p } p))). z I s * \text{rat-of-nat} (\text{card} \{x. \text{poly } p x = 0 \wedge \text{consistent-sign-vec-copr } qs x = s\}))$
 $+ (\sum s \in (\text{set}(\text{signs}) - (\text{consistent-sign-vec-copr } qs \text{ ‘ set } (\text{characterize-root-list-p } p))). z I s * \text{rat-of-nat} (\text{card} \{x. \text{poly } p x = 0 \wedge \text{consistent-sign-vec-copr } qs x = s\}))$

by (*metis* (*no-types*, *lifting*) *List.finite-set sum.Int-Diff*)

have $\text{sum-zero}: (\sum s \in (\text{set}(\text{signs}) - (\text{consistent-sign-vec-copr } qs \text{ ‘ set } (\text{characterize-root-list-p } p))). z I s * \text{rat-of-nat} (\text{card} \{x. \text{poly } p x = 0 \wedge \text{consistent-sign-vec-copr } qs x = s\})) = 0$

using *hyp*

by (*simp add: hyp*)

show *?thesis using sum-rewrite sum-split sum-zero by linarith*

qed

then have $\text{set-eq}: \text{set}(\text{remdups}$
 $(\text{map}(\text{consistent-sign-vec-copr } qs)$
 $(\text{characterize-root-list-p } p))) = \text{set}(\text{signs}) \cap (\text{consistent-sign-vec-copr } qs$
 $\text{ ‘ set } (\text{characterize-root-list-p } p))$

using *all-info*

by (*simp add: characterize-consistent-signs-at-roots-copr-def subset-antisym*)

have $\text{hyp1}: (\sum s \leftarrow \text{signs}. z I s * \text{rat-of-nat} (\text{card} \{x. \text{poly } p x = 0 \wedge \text{consistent-sign-vec-copr } qs x = s\})) =$
 $(\sum s \in \text{set}(\text{signs}). z I s * \text{rat-of-nat} (\text{card} \{x. \text{poly } p x = 0 \wedge \text{consistent-sign-vec-copr } qs x = s\}))$

using *distinct-signs sum-list-distinct-conv-sum-set by blast*

have $\text{hyp2}: (\sum s \leftarrow \text{remdups}$
 $(\text{map}(\text{consistent-sign-vec-copr } qs)$
 $(\text{characterize-root-list-p } p))). z I s * \text{rat-of-nat} (\text{card} \{x. \text{poly } p x = 0 \wedge \text{consistent-sign-vec-copr } qs x = s\}))$

```

= (∑ s ∈ set (remdups
  (map (consistent-sign-vec-copr qs)
    (characterize-root-list-p p))). z I s * rat-of-nat (card {x. poly p x = 0 ∧
consistent-sign-vec-copr qs x = s}))
  using sum-list-distinct-conv-sum-set by blast
  have set-sum-eq: (∑ s ∈ (set (signs) ∩ (consistent-sign-vec-copr qs ' set (characterize-root-list-p
p))). z I s * rat-of-nat (card {x. poly p x = 0 ∧ consistent-sign-vec-copr qs x =
s})) =
  (∑ s ∈ set (remdups
    (map (consistent-sign-vec-copr qs)
      (characterize-root-list-p p))). z I s * rat-of-nat (card {x. poly p x = 0 ∧
consistent-sign-vec-copr qs x = s}))
    using set-eq by auto
  then have (∑ s ← signs. z I s * rat-of-nat (card {x. poly p x = 0 ∧ consis-
tent-sign-vec-copr qs x = s})) =
  (∑ s ← remdups
    (map (consistent-sign-vec-copr qs)
      (characterize-root-list-p p))). z I s * rat-of-nat (card {x. poly p x = 0 ∧
consistent-sign-vec-copr qs x = s}))
    using set-sum-eq hyp1 hyp2
  using ⟨(∑ s ∈ set signs. z I s * rat-of-nat (card {x. poly p x = 0 ∧ consis-
tent-sign-vec-copr qs x = s})) = (∑ s ∈ set signs ∩ consistent-sign-vec-copr qs ' set
(characterize-root-list-p p). z I s * rat-of-nat (card {x. poly p x = 0 ∧ consis-
tent-sign-vec-copr qs x = s}))⟩ by linarith
  then have consistent-sign-vec-copr qs ' set (characterize-root-list-p p) ⊆ set signs
  ⇒
  (∧ p qss.
    characterize-consistent-signs-at-roots-copr p qss =
    remdups (map (consistent-sign-vec-copr qss) (characterize-root-list-p p))) ⇒
  (∑ s ← signs. z I s * rat-of-nat (card {x. poly p x = 0 ∧ consistent-sign-vec-copr
qs x = s})) =
  (∑ s ← remdups
    (map (consistent-sign-vec-copr qs)
      (characterize-root-list-p p))). z I s * rat-of-nat (card {x. poly p x = 0 ∧
consistent-sign-vec-copr qs x = s}))
    by linarith
  then show ?thesis unfolding lhs-dot-rewrite[OF nonzero]
  apply (auto intro!: sum-list-distinct-filter simp add: distinct-signs character-
ize-consistent-signs-at-roots-copr-def)
  using all-info characterize-consistent-signs-at-roots-copr-def by auto[1]
qed

```

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 ≠ 0

```

```

assumes distinct-signs: distinct signs
assumes all-info:  $set (characterize-consistent-signs-at-roots-copr p qs) \subseteq set(signs)$ 
assumes welldefined: list-constr I (length qs)
assumes pairwise-rel-prime-1:  $\forall q. ((List.member qs q) \longrightarrow (coprime p q))$ 
shows  $(vec-of-list (mtx-row signs I) \cdot (construct-lhs-vector p qs signs)) =$ 
 $rat-of-int (card \{x. poly p x = 0 \wedge poly (prod-list (retrieve-polys qs I)) x > 0\})$ 
 $-$ 
 $rat-of-int (card \{x. poly p x = 0 \wedge poly (prod-list (retrieve-polys qs I)) x < 0\})$ 
proof  $-$ 
have finite  $(set (map (consistent-sign-vec-copr qs) (characterize-root-list-p p)))$ 
by auto
let ?gt  $= (set (map (consistent-sign-vec-copr qs) (characterize-root-list-p p)) \cap$ 
 $\{s. z I s = 1\})$ 
let ?lt  $= (set (map (consistent-sign-vec-copr qs) (characterize-root-list-p p)) \cap$ 
 $\{s. z I s = -1\})$ 
have eq:  $set (map (consistent-sign-vec-copr qs) (characterize-root-list-p p)) =$ 
 $?gt \cup ?lt$ 
using welldefined poly-roots-finite [of p] nonzero
apply (auto simp add: characterize-root-list-p-def)
using in-set z-lemma apply fastforce
done

from construct-lhs-vector-drop-consistent[OF assms(1-4)] have
 $vec-of-list (mtx-row signs I) \cdot construct-lhs-vector p qs signs =$ 
 $vec-of-list (mtx-row (characterize-consistent-signs-at-roots-copr p qs) I) \cdot$ 
 $construct-lhs-vector p qs (characterize-consistent-signs-at-roots-copr p qs) .$ 

from lhs-dot-rewrite[OF assms(1)]
moreover have  $\dots =$ 
 $(\sum s \leftarrow characterize-consistent-signs-at-roots-copr p qs.$ 
 $z I s * rat-of-int (card \{x. poly p x = 0 \wedge consistent-sign-vec-copr qs x = s\})) .$ 
moreover have  $\dots =$ 
 $(\sum s \in set (map (consistent-sign-vec-copr qs) (characterize-root-list-p p)).$ 
 $z I s * rat-of-int (card \{x. poly p x = 0 \wedge consistent-sign-vec-copr qs x = s\}))$ 
unfolding characterize-consistent-signs-at-roots-copr-def sum-code[symmetric]
by (auto)
ultimately have  $\dots =$ 
 $(\sum s \in ?gt. z I s * rat-of-int (card \{x. poly p x = 0 \wedge consistent-sign-vec-copr qs$ 
 $x = s\})) +$ 
 $(\sum s \in ?lt. z I s * rat-of-int (card \{x. poly p x = 0 \wedge consistent-sign-vec-copr qs$ 
 $x = s\}))$ 
apply (subst eq)
apply (rule sum.union-disjoint)
by auto

have setroots:  $set (characterize-root-list-p p) = \{x. poly p x = 0\}$  unfolding
characterize-root-list-p-def
using poly-roots-finite nonzero rsquarefree-def set-sorted-list-of-set by blast
have  $*$ :  $\bigwedge s. \{x. poly p x = 0 \wedge consistent-sign-vec-copr qs x = s\} =$ 

```

```

      {x ∈ {x. poly p x = 0}. consistent-sign-vec-copr qs x = s}
    by auto
  have lem-e1:  $\bigwedge x. x \in \{x. \text{poly } p \ x = 0\} \implies$ 
    card
      {s ∈ consistent-sign-vec-copr qs ‘ {x. poly p x = 0} ∩ {s. z I s = 1}.
        consistent-sign-vec-copr qs x = s} =
      (if 0 < poly (prod-list (retrieve-polys qs I)) x then 1 else 0)
  proof -
    fix x
    assume rt: x ∈ {x. poly p x = 0}
    then have 1: {s ∈ consistent-sign-vec-copr qs ‘ {x. poly p x = 0} ∩ {s. z I s
    = 1}. consistent-sign-vec-copr qs x = s} =
      {s. z I s = 1 ∧ consistent-sign-vec-copr qs x = s}
    by auto
    from horiz-vector-helper-pos[OF assms(1) assms(5) rt]
    have 2: ... = {s. (0 < poly (prod-list (retrieve-polys qs I)) x) ∧ consis-
    tent-sign-vec-copr qs x = s}
    using welldefined by blast
    have 3: ... = (if (0 < poly (prod-list (retrieve-polys qs I)) x) then {consistent-sign-vec-copr
    qs x} else {})
    by auto
    thus card {s ∈ consistent-sign-vec-copr qs ‘ {x. poly p x = 0} ∩ {s. z I s = 1}.
    consistent-sign-vec-copr qs x = s} =
      (if 0 < poly (prod-list (retrieve-polys qs I)) x then 1 else 0) using 1 2 3
  by auto
  qed
  have e1: ( $\sum s \in \text{consistent-sign-vec-copr } qs \text{ ‘ } \{x. \text{poly } p \ x = 0\} \cap \{s. z \ I \ s = 1\}.$ 
    card {x. poly p x = 0 ∧ consistent-sign-vec-copr qs x = s}) =
    (sum (λx. if (poly (prod-list (retrieve-polys qs I)) x) > 0 then 1 else 0) {x.
    poly p x = 0})
    unfolding * apply (rule sum-multicount-gen)
    using <finite (set (map (consistent-sign-vec-copr qs) (characterize-root-list-p
    p)))> setroots apply auto[1]
    apply (metis List.finite-set setroots)
    using lem-e1 by auto
    have gtchr: ( $\sum s \in ?gt. z \ I \ s * \text{rat-of-int } (\text{card } \{x. \text{poly } p \ x = 0 \wedge \text{consis-}$ 
    tent-sign-vec-copr qs x = s})) =
      rat-of-int (card {x. poly p x = 0 ∧ 0 < poly (prod-list (retrieve-polys qs I)) x})
    apply (auto simp add: setroots)
    apply (subst of-nat-sum[symmetric])
    apply (subst of-nat-eq-iff)
    apply (subst e1)
    apply (subst card-eq-sum)
    apply (rule sum.mono-neutral-cong-right)
    apply (metis List.finite-set setroots)
    by auto
  have lem-e2:  $\bigwedge x. x \in \{x. \text{poly } p \ x = 0\} \implies$ 
    card
      {s ∈ consistent-sign-vec-copr qs ‘ {x. poly p x = 0} ∩ {s. z I s = -1}.

```

$consistent\text{-}sign\text{-}vec\text{-}copr\ qs\ x = s\} =$
 $(if\ poly\ (prod\text{-}list\ (retrieve\text{-}polys\ qs\ I))\ x < 0\ then\ 1\ else\ 0)$

proof –

fix x

assume $rt: x \in \{x. poly\ p\ x = 0\}$

then have $1: \{s \in consistent\text{-}sign\text{-}vec\text{-}copr\ qs\ \ ' \ \{x. poly\ p\ x = 0\} \cap \{s. z\ I\ s = -1\}. consistent\text{-}sign\text{-}vec\text{-}copr\ qs\ x = s\} =$
 $\{s. z\ I\ s = -1 \wedge consistent\text{-}sign\text{-}vec\text{-}copr\ qs\ x = s\}$

by *auto*

from *horiz-vector-helper-neg*[*OF* *assms*(1) *assms*(5) *rt*]

have $2: \dots = \{s. (0 > poly\ (prod\text{-}list\ (retrieve\text{-}polys\ qs\ I))\ x) \wedge consistent\text{-}sign\text{-}vec\text{-}copr\ qs\ x = s\}$

using *welldefined* **by** *blast*

have $3: \dots = (if\ (0 > poly\ (prod\text{-}list\ (retrieve\text{-}polys\ qs\ I))\ x)\ then\ \{consistent\text{-}sign\text{-}vec\text{-}copr\ qs\ x\}\ else\ \{\})$

by *auto*

thus $card\ \{s \in consistent\text{-}sign\text{-}vec\text{-}copr\ qs\ \ ' \ \{x. poly\ p\ x = 0\} \cap \{s. z\ I\ s = -1\}. consistent\text{-}sign\text{-}vec\text{-}copr\ qs\ x = s\} =$
 $(if\ poly\ (prod\text{-}list\ (retrieve\text{-}polys\ qs\ I))\ x < 0\ then\ 1\ else\ 0)$ **using** 1 2 3 **by** *auto*

qed

have $e2: (\sum\ s \in consistent\text{-}sign\text{-}vec\text{-}copr\ qs\ \ ' \ \{x. poly\ p\ x = 0\} \cap \{s. z\ I\ s = -1\}. card\ \{x. poly\ p\ x = 0 \wedge consistent\text{-}sign\text{-}vec\text{-}copr\ qs\ x = s\}) =$
 $(sum\ (\lambda x. if\ (poly\ (prod\text{-}list\ (retrieve\text{-}polys\ qs\ I))\ x) < 0\ then\ 1\ else\ 0)\ \{x. poly\ p\ x = 0\})$

unfolding * **apply** (*rule* *sum-multicount-gen*)

using $\langle finite\ (set\ (map\ (consistent\text{-}sign\text{-}vec\text{-}copr\ qs)\ (characterize\text{-}root\text{-}list\text{-}p\ p))) \rangle$ *setroots* **apply** *auto*[1]

apply (*metis* *List.finite-set* *setroots*)

using *lem-e2* **by** *auto*

have $ltchr: (\sum\ s \in ?lt. z\ I\ s * rat\text{-}of\text{-}int\ (card\ \{x. poly\ p\ x = 0 \wedge consistent\text{-}sign\text{-}vec\text{-}copr\ qs\ x = s\})) =$
 $- rat\text{-}of\text{-}int\ (card\ \{x. poly\ p\ x = 0 \wedge 0 > poly\ (prod\text{-}list\ (retrieve\text{-}polys\ qs\ I))\ x\})$

apply (*auto* *simp* *add: setroots* *sum-negf*)

apply (*subst* *of-nat-sum*[*symmetric*])

apply (*subst* *of-nat-eq-iff*)

apply (*subst* *e2*)

apply (*subst* *card-eq-sum*)

apply (*rule* *sum.mono-neutral-cong-right*)

apply (*metis* *List.finite-set* *setroots*)

by *auto*

show *?thesis* **using** *gtchr* *ltchr*

using $\langle (\sum\ s \in set\ (map\ (consistent\text{-}sign\text{-}vec\text{-}copr\ qs)\ (characterize\text{-}root\text{-}list\text{-}p\ p)). z\ I\ s * rat\text{-}of\text{-}int\ (card\ \{x. poly\ p\ x = 0 \wedge consistent\text{-}sign\text{-}vec\text{-}copr\ qs\ x = s\})) =$
 $(\sum\ s \in set\ (map\ (consistent\text{-}sign\text{-}vec\text{-}copr\ qs)\ (characterize\text{-}root\text{-}list\text{-}p\ p)) \cap \{s. z\ I\ s = 1\}. z\ I\ s * rat\text{-}of\text{-}int\ (card\ \{x. poly\ p\ x = 0 \wedge consistent\text{-}sign\text{-}vec\text{-}copr\ qs\ x = s\})) + (\sum\ s \in set\ (map\ (consistent\text{-}sign\text{-}vec\text{-}copr\ qs)\ (characterize\text{-}root\text{-}list\text{-}p$

$p)) \cap \{s. z I s = - 1\}. z I s * \text{rat-of-int } (\text{card } \{x. \text{poly } p x = 0 \wedge \text{consistent-sign-vec-copr } qs x = s\}) \rangle \langle (\sum s \leftarrow \text{characterize-consistent-signs-at-roots-copr } p qs. z I s * \text{rat-of-int } (\text{card } \{x. \text{poly } p x = 0 \wedge \text{consistent-sign-vec-copr } qs x = s\})) \rangle$
 $= (\sum s \in \text{set } (\text{map } (\text{consistent-sign-vec-copr } qs) (\text{characterize-root-list-p } p)). z I s * \text{rat-of-int } (\text{card } \{x. \text{poly } p x = 0 \wedge \text{consistent-sign-vec-copr } qs x = s\})) \rangle \langle \text{vec-of-list } (\text{mtx-row } (\text{characterize-consistent-signs-at-roots-copr } p qs) I) \cdot \text{construct-lhs-vector } p qs (\text{characterize-consistent-signs-at-roots-copr } p qs) = (\sum s \leftarrow \text{characterize-consistent-signs-at-roots-copr } p qs. z I s * \text{rat-of-int } (\text{card } \{x. \text{poly } p x = 0 \wedge \text{consistent-sign-vec-copr } qs x = s\})) \rangle \langle \text{vec-of-list } (\text{mtx-row } \text{signs } I) \cdot \text{construct-lhs-vector } p qs \text{signs} = \text{vec-of-list } (\text{mtx-row } (\text{characterize-consistent-signs-at-roots-copr } p qs) I) \cdot \text{construct-lhs-vector } p qs (\text{characterize-consistent-signs-at-roots-copr } p qs) \rangle$
 by *linarith*
qed

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*: $\text{set } (\text{characterize-consistent-signs-at-roots-copr } p qs) \subseteq \text{set}(signs)$
assumes *welldefined*: $\text{list-constr } I (\text{length } qs)$
assumes *pairwise-rel-prime-1*: $\forall q. ((\text{List.member } qs q) \longrightarrow (\text{coprime } p q))$
shows $(\text{vec-of-list } (\text{mtx-row } \text{signs } I) \cdot (\text{construct-lhs-vector } p qs \text{signs})) = \text{construct-NoFI } p (\text{retrieve-polys } qs I)$
unfolding *construct-NoFI-prop*[*OF nonzero*]
using *matrix-equation-helper-step*[*OF assms*]
by *linarith*

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

assumes $\text{map } f xs = \text{map } g ys$
shows $\text{map-vec } f (\text{vec-of-list } xs) = \text{map-vec } g (\text{vec-of-list } ys)$
by (*metis assms vec-of-list-map*)

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*: $\text{set } (\text{characterize-consistent-signs-at-roots-copr } p qs) \subseteq \text{set}(signs)$
assumes *pairwise-rel-prime*: $\forall q. ((\text{List.member } qs q) \longrightarrow (\text{coprime } p q))$
assumes *welldefined*: $\text{all-list-constr } (subsets) (\text{length } qs)$
shows *satisfy-equation* $p qs subsets signs$
unfolding *satisfy-equation-def matrix-A-def*

construct-lhs-vector-def construct-rhs-vector-def all-list-constr-def
apply (*subst mult-mat-vec-of-list*)
apply (*auto simp add: mtx-row-length intro!: map-vec-vec-of-list-eq-intro*)
using *matrix-equation-main-step*[*OF assms(1-3) - assms(4)*, *unfolded construct-lhs-vector-def*]
using *all-list-constr-def welldefined* **by** *fastforce*

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)
unfolding *consistent-signs-at-roots-def characterize-consistent-signs-at-roots-def*
characterize-root-list-p-def
apply *auto*
apply (*subst set-sorted-list-of-set*)
using *assms poly-roots-finite* **apply** *blast*
unfolding *sgn-vec-def sgn-def signs-at-def squash-def o-def*
using *roots-def* **apply** *auto[1]*
by (*smt Collect-cong assms image-iff poly-roots-finite roots-def sorted-list-of-set(1)*)

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 $\bigwedge q. q \in \text{set } qs \implies \text{coprime } p \ q$
assumes *distinct signs*
assumes *consistent-signs-at-roots p qs \subseteq set signs*

assumes $\bigwedge l i. l \in \text{set subsets} \implies i \in \text{set } l \implies i < \text{length } qs$
shows $M\text{-mat signs subsets } *_v w\text{-vec } p \text{ } qs \text{ signs} = v\text{-vec } p \text{ } qs \text{ subsets}$
unfolding *satisfy-equation-def* [*symmetric*]
apply (*rule matrix-equation* [*OF assms(1) assms(3)*])
using $\langle p \neq 0 \rangle \langle \text{consistent-signs-at-roots } p \text{ } qs \subseteq \text{set signs} \rangle$
by (*auto simp add: assms(2) all-list-constr-def list-constr-def list-all-iff consistent-signs-at-roots-eq csa-list-copr-rel subset-iff intro: assms(5)*)

end

theory *BKR-Proofs*

imports *Matrix-Equation-Construction*

begin

definition *well-def-signs:: nat => rat list list => 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:: real poly => real poly list => nat list list => rat list list => rat mat => 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* $[[1],[-1]] [[],[0]] = (\text{mat-of-rows-list } 2 \text{ } [[1, 1], [1, -1]])$

unfolding *matrix-A-def mtx-row-def z-def* **by** (*auto simp add: numeral-2-eq-2*)

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]}*

unfolding *characterize-consistent-signs-at-roots-copr-def consistent-sign-vec-copr-def*

by *auto*

lemma *base-case-sgas-alt:*

fixes *p:: real poly*

fixes *qs:: real poly list*

assumes *len1: length qs = 1*

assumes *nonzero: p $\neq 0$*

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

shows *set (characterize-consistent-signs-at-roots-copr p qs) \subseteq {[1], [- 1]}*

```

proof –
  have ex-q:  $\exists (q::\text{real poly}). qs = [q]$ 
    using len1
    using length-Suc-conv[of qs 0] by auto
  then show ?thesis using base-case-sgas nonzero rel-prime
    apply (auto)
    using characterize-consistent-signs-at-roots-copr-def consistent-sign-vec-copr-def
by auto
qed

```

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]]

```

proof –

```

have h1: set (characterize-consistent-signs-at-roots-copr p [q])  $\subseteq \{[1], [-1]\}$ 
  using base-case-sgas assms by auto
have h2:  $\forall qa. \text{List.member } [q] qa \longrightarrow \text{coprime } p qa$  using rel-prime
  by simp
have h3: all-list-constr [[],[0]] (Suc 0) unfolding all-list-constr-def
  by (simp add: list-constr-def)
then show ?thesis using matrix-equation[where  $p = p$ , where  $qs = [q]$ , where
signs =  $[[1],[-1]]$ , where subsets =  $[[],[0]]$ ]
  nonzero h1 h2 h3 by auto

```

qed

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

```

fixes p:: real poly
fixes qs:: real poly list
assumes len1: 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]]

```

proof –

```

have ex-q:  $\exists (q::\text{real poly}). qs = [q]$ 
  using len1
  using length-Suc-conv[of qs 0] by auto
then show ?thesis using base-case-satisfy-equation nonzero rel-prime
  by auto

```

qed

lemma *base-case-matrix-eq*:

```

fixes q p:: real poly
assumes nonzero:  $p \neq 0$ 
assumes rel-prime: coprime p q
shows (mult-mat-vec (mat-of-rows-list 2 [[1, 1], [1, -1]]) (construct-lhs-vector
p [q] [[1],[-1]]) =
  (construct-rhs-vector p [q] [[],[0]]))

```

using *mat-base-case base-case-satisfy-equation* **unfolding** *satisfy-equation-def*
by (*simp add: nonzero rel-prime*)

lemma *less-two*:

shows $j < \text{Suc } (\text{Suc } 0) \longleftrightarrow j = 0 \vee j = 1$ **by** *auto*

lemma *inverse-mat-base-case*:

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

unfolding *inverts-mat-def mat-of-rows-list-def mat-eq-iff* **apply** *auto*

unfolding *less-two* **by** (*auto simp add: scalar-prod-def*)

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

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

unfolding *inverts-mat-def mat-of-rows-list-def mat-eq-iff* **apply** *auto*

unfolding *less-two* **by** (*auto simp add: scalar-prod-def*)

lemma *base-case-invertible-mat*:

shows *invertible-mat* (*matrix-A* $[[1], [-1]]$ $[[], [0]]$)

unfolding *invertible-mat-def* **using** *inverse-mat-base-case inverse-mat-base-case-2*

apply (*auto*)

apply (*simp add: mat-base-case mat-of-rows-list-def*)

using *mat-base-case* **by** *auto*

13 Inductive Step

13.1 Lemmas on smashing subsets and signs

lemma *signs-smash-property*:

fixes *signs1 signs2* :: 'a list list

fixes *a b*:: nat

shows $\forall (elem :: 'a list). (elem \in (set (signs-smash signs1 signs2))) \longrightarrow$

$(\exists n m :: nat.$

$elem = ((nth signs1 n)@(nth signs2 m))))$

proof *clarsimp*

fix *elem*

assume *assum*: $elem \in set (signs-smash signs1 signs2)$

show $\exists n m. elem = signs1 ! n @ signs2 ! m$

using *assum* **unfolding** *signs-smash-def* **apply** (*auto*)

by (*metis in-set-conv-nth*)

qed

lemma *signs-smash-property-set*:

fixes *signs1 signs2* :: 'a list list

fixes *a b*:: nat

shows $\forall (elem :: 'a list). (elem \in (set (signs-smash signs1 signs2))) \longrightarrow$

$(\exists (elem1 :: 'a list). \exists (elem2 :: 'a list).$

$(elem1 \in set(signs1) \wedge elem2 \in set(signs2) \wedge elem = (elem1@elem2))))$

```

proof clarsimp
  fix elem
  assume assum: elem ∈ set (signs-smash signs1 signs2)
  show ∃ elem1. elem1 ∈ set signs1 ∧ (∃ elem2. elem2 ∈ set signs2 ∧ elem =
elem1 @ elem2)
    using assum unfolding signs-smash-def by auto
qed

```

```

lemma subsets-smash-property:
  fixes subsets1 subsets2 :: nat list list
  fixes n:: nat
  shows ∀ (elem :: nat list). (List.member (subsets-smash n subsets1 subsets2)
elem) →
  (∃ (elem1::nat list). ∃ (elem2::nat list).
  (elem1 ∈ set(subsets1) ∧ elem2 ∈ set(subsets2) ∧ elem = (elem1 @ ((map
  ((+) n) elem2))))))
proof –
  let ?new-subsets = (map (map ((+) n) subsets2))

  have subsets-smash n subsets1 subsets2 = signs-smash subsets1 ?new-subsets
  unfolding subsets-smash-def signs-smash-def by (auto simp add: comp-def)
  then show ?thesis
    using signs-smash-property-set
    by auto fastforce
qed

```

13.2 Well-defined subsets preserved when smashing

```

lemma list-constr-append:
  list-constr a n ∧ list-constr b n → list-constr (a@b) n
  apply (auto) unfolding list-constr-def
  using list-all-append by blast

```

```

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))
proof –
  have h1: ∀ elem.
    List.member (subsets-smash (length qs1) subsets1 subsets2) elem →
    (∃ elem1 elem2. elem1 ∈ set subsets1 ∧ elem2 ∈ set subsets2 ∧ elem = elem1
    @ map ((+) (length qs1)) elem2)
    using subsets-smash-property by blast
  have h2: ∀ elem1 elem2. (elem1 ∈ set subsets1 ∧ elem2 ∈ set subsets2) →
  list-constr (elem1 @ map ((+) (length qs1)) elem2) (length (qs1 @ qs2))
  proof clarsimp

```

```

fix elem1
fix elem2
assume e1: elem1 ∈ set subsets1
assume e2: elem2 ∈ set subsets2
have h1: list-constr elem1 (length qs1 + length qs2)
proof –
  have h1: list-constr elem1 (length qs1) using e1 well-def-subsets1
    unfolding all-list-constr-def
    by simp
  then show ?thesis unfolding list-constr-def
    by (simp add: list-pred-mono-strong)
qed
have h2: list-constr (map ((+) (length qs1)) elem2) (length qs1 + length qs2)
proof –
  have h1: list-constr elem2 (length qs2) using e2 well-def-subsets2
    unfolding all-list-constr-def
    by simp
  then show ?thesis unfolding list-constr-def
    by (simp add: list-all-length)
qed
show list-constr (elem1 @ map ((+) (length qs1)) elem2) (length qs1 + length
qs2)
  using h1 h2 list-constr-append
  by blast
qed
then show ?thesis using h1 unfolding all-list-constr-def by auto
qed

```

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)
  using well-def1 well-def2 unfolding well-def-signs-def using signs-smash-property-set[of
signs1 signs2] length-append by auto

```

13.4 Distinct signs preserved when smashing

```

lemma distinct-map-append:
  assumes distinct ls
  shows distinct (map ((@) xs) ls)
  unfolding distinct-map inj-on-def using assms by auto

```

```

lemma length-eq-append:
  assumes length y = length b

```

shows $(x @ y = a @ b) \longleftrightarrow x=a \wedge y = b$
by (*simp add: assms*)

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)*
unfolding *signs-smash-def*
using *distinct1*
proof(*induction signs1*)
 case *Nil*
 then show *?case by auto*
next
 case (*Cons a signs1*)
 then show *?case apply (auto simp add: distinct2 distinct-map-append)*
 using *assms unfolding well-def-signs-def*
 by (*simp add: distinct1 distinct2 length-eq-append*)
qed

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 –
 have *h1: $\forall x \in \text{set (signs-smash a1 b1)}. x \in \text{set (signs-smash a2 b2)}$*
 proof *clarsimp*
 fix *x*
 assume *x-in: $x \in \text{set (signs-smash a1 b1)}$*
 have *h1: $\exists n m :: \text{nat}. x = (\text{nth a1 } n) @ (\text{nth b1 } m)$*
 using *signs-smash-property[of a1 b1] x-in*
 by *blast*
 then have *h2: $\exists p q :: \text{nat}. x = (\text{nth a2 } p) @ (\text{nth b2 } q)$*
 using *sub1 sub2 apply (auto)*
 by (*metis nth-find-first signs-smash-property-set subset-code(1) x-in*)
 then show *$x \in \text{set (signs-smash a2 b2)}$*
 unfolding *signs-smash-def apply (auto)*
 using *signs-smash-property-set sub1 sub2 x-in by fastforce*
qed
then show *?thesis*
 by *blast*
qed

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

have $x\text{-in-csv}$: $x \in \text{set}(\text{map}(\text{consistent-sign-vec-copr } (qs1 @ qs2))(\text{characterize-root-list-p } p))$
using $x\text{-in}$ **unfolding** *characterize-consistent-signs-at-roots-copr-def* **by** *simp*
have $csv\text{-hyp}$: $\forall r. \text{consistent-sign-vec-copr } (qs1 @ qs2) r = (\text{consistent-sign-vec-copr } qs1 r) @ (\text{consistent-sign-vec-copr } qs2 r)$
unfolding *consistent-sign-vec-copr-def* **by** *auto*
have $exr\text{-iff}$: $(\exists r \in \text{set}(\text{characterize-root-list-p } p). x = \text{consistent-sign-vec-copr } (qs1 @ qs2) r) \longleftrightarrow (\exists r \in \text{set}(\text{characterize-root-list-p } p). x = (\text{consistent-sign-vec-copr } qs1 r) @ (\text{consistent-sign-vec-copr } qs2 r))$
using $csv\text{-hyp}$ **by** *auto*
have exr : $x \in \text{set}(\text{map}(\text{consistent-sign-vec-copr } (qs1 @ qs2))(\text{characterize-root-list-p } p)) \longrightarrow (\exists r \in \text{set}(\text{characterize-root-list-p } p). x = \text{consistent-sign-vec-copr } (qs1 @ qs2) r)$
by *auto*
have $mem\text{-list1}$: $\forall r \in \text{set}(\text{characterize-root-list-p } p). (\text{consistent-sign-vec-copr } qs1 r) \in \text{set}(\text{map}(\text{consistent-sign-vec-copr } qs1)(\text{characterize-root-list-p } p))$
by *simp*
have $mem\text{-list2}$: $\forall r \in \text{set}(\text{characterize-root-list-p } p). (\text{consistent-sign-vec-copr } qs2 r) \in \text{set}(\text{map}(\text{consistent-sign-vec-copr } qs2)(\text{characterize-root-list-p } p))$
by *simp*
then show $\exists x1 \in \text{set}(\text{characterize-consistent-signs-at-roots-copr } p qs1).$
 $\exists x2 \in \text{set}(\text{characterize-consistent-signs-at-roots-copr } p qs2). x = x1 @ x2$
using $x\text{-in-csv}$ exr $mem\text{-list1}$ $mem\text{-list2}$ *characterize-consistent-signs-at-roots-copr-def* $exr\text{-iff}$ **by** *auto*
qed

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

shows $\text{set}(\text{characterize-consistent-signs-at-roots-copr } p (qs1 @ qs2))$
 $\subseteq \text{set}(signs\text{-smash } signs1\ signs2)$
proof –

have $h0: \forall x \in \text{set } (\text{characterize-consistent-signs-at-roots-copr } p \text{ (} qs1 \text{ @ } qs2)). \exists$
 $x1 \in \text{set } (\text{characterize-consistent-signs-at-roots-copr } p \text{ } qs1). \exists x2 \in \text{set } (\text{characterize-consistent-signs-at-roots-copr } p \text{ } qs2).$
 $(x = x1 @ x2)$ **using** *subset-helper*[of p $qs1$ $qs2$]
by *blast*
then have $\forall x \in \text{set } (\text{characterize-consistent-signs-at-roots-copr } p \text{ (} qs1 \text{ @ } qs2)).$
 $x \in \text{set } (\text{signs-smash } (\text{characterize-consistent-signs-at-roots-copr } p \text{ } qs1)$
 $(\text{characterize-consistent-signs-at-roots-copr } p \text{ } qs2))$
unfolding *signs-smash-def* **apply** (*auto*)
by *fastforce*
then have $\forall x \in \text{set } (\text{characterize-consistent-signs-at-roots-copr } p$
 $(qs1 \text{ @ } qs2)). x \in \text{set } (\text{signs-smash } signs1 \text{ } signs2)$
using *csa1 csa2 subset-smash-signs*[of - $signs1$ - $signs2$] **apply** (*auto*)
by *blast*
thus *?thesis*
by *blast*
qed

13.6 Main Results

lemma *dim-row-mat-of-rows-list*[*simp*]:
shows $\text{dim-row } (\text{mat-of-rows-list } nr \text{ } ls) = \text{length } ls$
unfolding *mat-of-rows-list-def* **by** *auto*

lemma *dim-col-mat-of-rows-list*[*simp*]:
shows $\text{dim-col } (\text{mat-of-rows-list } nr \text{ } ls) = nr$
unfolding *mat-of-rows-list-def* **by** *auto*

lemma *dim-row-matrix-A*[*simp*]:
shows $\text{dim-row } (\text{matrix-A } signs \text{ } subsets) = \text{length } subsets$
unfolding *matrix-A-def* **by** *auto*

lemma *dim-col-matrix-A*[*simp*]:
shows $\text{dim-col } (\text{matrix-A } signs \text{ } subsets) = \text{length } signs$
unfolding *matrix-A-def* **by** *auto*

lemma *length-signs-smash*:
shows
 $\text{length } (\text{signs-smash } signs1 \text{ } signs2) = \text{length } signs1 * \text{length } signs2$
unfolding *signs-smash-def* *length-concat*
by (*auto simp add: o-def sum-list-triv*)

lemma *length-subsets-smash*:
shows
 $\text{length } (\text{subsets-smash } n \text{ } subs1 \text{ } subs2) = \text{length } subs1 * \text{length } subs2$
unfolding *subsets-smash-def* *length-concat*
by (*auto simp add: o-def sum-list-triv*)

lemma *length-eq-concat*:

```

assumes  $\bigwedge x. x \in \text{set } ls \implies \text{length } x = n$ 
assumes  $i < n * \text{length } ls$ 
shows  $\text{concat } ls ! i = ls ! (i \text{ div } n) ! (i \text{ mod } n)$ 
using assms
proof (induct ls arbitrary: i)
  case Nil
  then show ?case
    by simp
next
from assms have  $\langle n > 0 \rangle$ 
  by (cases  $\langle n = 0 \rangle$ ) simp-all
case (Cons a ls)
show ?case
proof (cases  $\langle n \leq i \rangle$ )
  case False
  with Cons show ?thesis
    by (simp add: nth-append)
next
  case True
  moreover define j where  $\langle j = i - n \rangle$ 
  ultimately have  $\langle i = n + j \rangle$ 
    by simp
  with Cons  $\langle n > 0 \rangle$  show ?thesis
    by (simp add: nth-append div-add-self1)
qed
qed

```

```

lemma z-append:
assumes  $\bigwedge i. i \in \text{set } xs \implies i < \text{length } as$ 
shows  $z (xs @ (\text{map } (+) (\text{length } as) ys)) (as @ bs) = z xs as * z ys bs$ 
proof –
  have 1:  $\text{map } (!) (as @ bs) xs = \text{map } (!) as xs$ 
    unfolding list-eq-iff-nth-eq
    using assms by (auto simp add: nth-append)
  have 2:  $\text{map } (!) (as @ bs) \circ (+) (\text{length } as) ys = \text{map } (!) bs ys$ 
    unfolding list-eq-iff-nth-eq
    using assms by auto
  show ?thesis
    unfolding z-def apply auto
    unfolding 1 2 by auto
qed

```

```

lemma matrix-construction-is-kronecker-product:
fixes qs1 :: real poly list
fixes subs1 subs2 :: nat list list
fixes signs1 signs2 :: 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
  (matrix-A (signs-smash signs1 signs2) (subsets-smash n1 subs1 subs2)) =
  kroncker-product (matrix-A signs1 subs1) (matrix-A signs2 subs2)
unfolding mat-eq-iff dim-row-matrix-A dim-col-matrix-A
  length-subsets-smash length-signs-smash dim-kroncker
proof safe
  fix i j
  assume i:  $i < \text{length } \text{subs1} * \text{length } \text{subs2}$ 
    and j:  $j < \text{length } \text{signs1} * \text{length } \text{signs2}$ 
  have ld:  $i \text{ div } \text{length } \text{subs2} < \text{length } \text{subs1}$ 
     $j \text{ div } \text{length } \text{signs2} < \text{length } \text{signs1}$ 
    using i j less-mult-imp-div-less by auto
  have lm:  $i \text{ mod } \text{length } \text{subs2} < \text{length } \text{subs2}$ 
     $j \text{ mod } \text{length } \text{signs2} < \text{length } \text{signs2}$ 
    using i j less-mult-imp-mod-less by auto

  have n1:  $n1 = \text{length } (\text{signs1} ! (j \text{ div } \text{length } \text{signs2}))$ 
    using assms(2) ld(2) nth-mem by blast

  have 1: matrix-A (signs-smash signs1 signs2) (subsets-smash n1 subs1 subs2) $$
  (i, j) =
    z (subsets-smash n1 subs1 subs2 ! i) (signs-smash signs1 signs2 ! j)
    unfolding mat-of-rows-list-def matrix-A-def mtx-row-def
    using i j
    by (simp add: length-signs-smash length-subsets-smash)
  have 2: ... = z (subs1 ! ( $i \text{ div } \text{length } \text{subs2}$ ) @ map ((+) n1) (subs2 ! ( $i \text{ mod } \text{length } \text{subs2}$ ))))
    (signs1 ! ( $j \text{ div } \text{length } \text{signs2}$ ) @ signs2 ! ( $j \text{ mod } \text{length } \text{signs2}$ ))
    unfolding signs-smash-def subsets-smash-def
    apply (subst length-eq-concat)
    using i apply (auto simp add: mult.commute)
    apply (subst length-eq-concat)
    using j apply (auto simp add: mult.commute)
    using ld lm by auto
  have 3: ... =
    z (subs1 ! ( $i \text{ div } \text{length } \text{subs2}$ )) (signs1 ! ( $j \text{ div } \text{length } \text{signs2}$ )) *
    z (subs2 ! ( $i \text{ mod } \text{length } \text{subs2}$ )) (signs2 ! ( $j \text{ mod } \text{length } \text{signs2}$ ))
    unfolding n1
    apply (subst z-append)
    apply (auto simp add: n1[symmetric])
    using assms(1) ld(1) nth-mem by blast
  have 4: kroncker-product (matrix-A signs1 subs1) (matrix-A signs2 subs2) $$
  (i, j) =
    z (subs1 ! ( $i \text{ div } \text{length } \text{subs2}$ ))
    (signs1 ! ( $j \text{ div } \text{length } \text{signs2}$ )) *
    z (subs2 ! ( $i \text{ mod } \text{length } \text{subs2}$ ))
    (signs2 ! ( $j \text{ mod } \text{length } \text{signs2}$ ))
    unfolding kroncker-product-def matrix-A-def mat-of-rows-list-def mtx-row-def

```

```

using i j apply (auto simp add: Let-def)
apply (subst index-mat(1)[OF ld])
apply (subst index-mat(1)[OF lm])
using ld lm by auto
show matrix-A (signs-smash signs1 signs2) (subsets-smash n1 subs1 subs2) $$
(i, j) =
  kroncker-product (matrix-A signs1 subs1) (matrix-A signs2 subs2) $$ (i, j)
using 1 2 3 4 by auto
qed

```

lemma *inductive-step*:

```

fixes p: real poly
fixes qs1 qs2 :: real poly list
fixes subsets1 subsets2 :: nat list list
fixes signs1 signs2 :: rat list list
assumes nonzero: p ≠ 0
assumes nontriv1: length qs1 > 0
assumes nontriv2: length qs2 > 0
assumes pairwise-rel-prime1:  $\forall q. ((List.member\ qs1\ q) \longrightarrow (coprime\ p\ q))$ 
assumes pairwise-rel-prime2:  $\forall q. ((List.member\ qs2\ q) \longrightarrow (coprime\ p\ q))$ 
assumes welldefined-signs1: well-def-signs (length qs1) signs1
assumes welldefined-signs2: well-def-signs (length qs2) signs2
assumes distinct-signs1: distinct signs1
assumes distinct-signs2: distinct signs2
assumes all-info1: set (characterize-consistent-signs-at-roots-copr p qs1) ⊆ set(signs1)
assumes all-info2: set (characterize-consistent-signs-at-roots-copr p qs2) ⊆ set(signs2)
assumes welldefined-subsets1: all-list-constr (subsets1) (length qs1)
assumes welldefined-subsets2: all-list-constr (subsets2) (length qs2)
assumes invertibleMtx1: invertible-mat (matrix-A signs1 subsets1)
assumes invertibleMtx2: invertible-mat (matrix-A signs2 subsets2)
shows satisfy-equation p (qs1@qs2) (subsets-smash (length qs1) subsets1 sub-
sets2) (signs-smash signs1 signs2)
   $\wedge$  invertible-mat (matrix-A (signs-smash signs1 signs2) (subsets-smash (length
qs1) subsets1 subsets2))
proof –
  have h1: invertible-mat (matrix-A (signs-smash signs1 signs2) (subsets-smash
(length qs1) subsets1 subsets2))
    using matrix-construction-is-kroncker-product
      kroncker-invertible invertibleMtx1 invertibleMtx2
      welldefined-subsets1 welldefined-subsets2 unfolding all-list-constr-def list-constr-def
    using well-def-signs-def welldefined-signs1
    by (smt (verit, del-insts) Ball-set List.member-iff welldefined-signs1)
  have h2a: distinct (signs-smash signs1 signs2)
    using distinct-signs1 distinct-signs2 welldefined-signs1 welldefined-signs2 non-
triv1 nontriv2
      distinct-step by auto
  have h2ba:  $\forall q. List.member (qs1 @ qs2) q \longrightarrow (List.member\ qs1\ q \vee List.member\ qs2\ q)$ 

```

```

    by auto
  have h2b:  $\forall q. ((List.member (qs1@qs2) q) \longrightarrow (coprime p q))$ 
    using h2ba pairwise-rel-prime1 pairwise-rel-prime2 by auto
  have h2c: all-list-constr ((subsets-smash (length qs1) subsets1 subsets2)) (length
    (qs1@qs2))
    using well-def-step
      welldefined-subsets1 welldefined-subsets2
    by blast
  have h2d: set (characterize-consistent-signs-at-roots-copr p (qs1@qs2))  $\subseteq$  set(signs-smash
    signs1 signs2)
    using subset-step all-info1 all-info2
    by simp
  have h2: satisfy-equation p (qs1@qs2) (subsets-smash (length qs1) subsets1 sub-
    sets2) (signs-smash signs1 signs2)
    using matrix-equation[where p=p, where qs=qs1@qs2, where subsets =
    subsets-smash (length qs1) subsets1 subsets2,
      where signs = signs-smash signs1 signs2]
      h2a h2b h2c h2d apply (auto) using nonzero by blast
  show ?thesis using h1 h2 by blast
qed

```

14 Reduction Step Proofs

definition *get-matrix*:: (rat mat \times (nat list list \times rat list list)) \Rightarrow rat mat
 where *get-matrix* data = fst(data)

definition *get-subsets*:: (rat mat \times (nat list list \times rat list list)) \Rightarrow nat list list
 where *get-subsets* data = fst(snd(data))

definition *get-signs*:: (rat mat \times (nat list list \times rat list list)) \Rightarrow rat list list
 where *get-signs* data = snd(snd(data))

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

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

lemma *reduction-signs-is-get-signs*: reduction-signs p qs signs subsets m = *get-signs*
 (reduce-system p (qs, (m, (subsets, signs))))
 unfolding reduction-signs-def get-signs-def
 by (metis reduce-system.simps reduction-step.elims snd-conv)

lemma *reduction-subsets-is-get-subsets*: *reduction-subsets p qs signs subsets m = get-subsets (reduce-system p (qs, (m, (subsets, signs))))*
unfolding *reduction-subsets-def get-subsets-def*
by (*metis fst-conv reduce-system.simps reduction-step.elims snd-conv*)

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 –

have *h1*: $\forall n < (\text{dim-vec } \text{input-vec}). ((\text{input-vec } \$ n \neq 0) \longrightarrow \text{List.member } (\text{find-nonzeros-from-input-vec } \text{input-vec}) n)$

unfolding *find-nonzeros-from-input-vec-def* **by** *auto*

have *h2*: $\forall n < (\text{dim-vec } \text{input-vec}). (\text{List.member } (\text{find-nonzeros-from-input-vec } \text{input-vec}) n \longrightarrow (\text{input-vec } \$ n \neq 0))$

unfolding *find-nonzeros-from-input-vec-def* **by** *auto*

show *?thesis* **using** *h1 h2* **by** *auto*

qed

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 $\exists k < \text{length } \text{arb-list}.$

$(\text{take-indices } \text{arb-list } \text{index-list}) ! n = \text{arb-list} ! k$

using *lest well-def* **unfolding** *take-indices-def* **apply** (*auto*)

by (*metis nth-mem*)

lemma *invertible-means-mult-id*:

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

assumes *assm*: *invertible-mat A*

shows *matr-option (dim-row A)*

$(\text{mat-inverse } (A)) * A = 1_m (\text{dim-row } A)$

proof (*cases mat-inverse(A)*)

obtain *n* **where** *n*: $A \in \text{carrier-mat } n n$

using *assms invertible-mat-def square-mat.simps* **by** *blast*

case *None*

then **have** $A \notin \text{Units } (\text{ring-mat } \text{TYPE}('a) n n)$

by (*simp add: mat-inverse(1) n*)

thus *?thesis*

by (*meson assms det-non-zero-imp-unit invertible-Units n unit-imp-det-non-zero*)

```

next
  case (Some a)
  then show ?thesis
  by (metis assms carrier-matI invertible-mat-def mat-inverse(2) matr-option.simps(2)
square-mat.elims(2))
qed

```

```

lemma dim-invertible:
  fixes A:: 'a::field mat
  fixes n:: nat
  assumes asm: invertible-mat A
  assumes dim: A ∈ carrier-mat n n
  shows matr-option (dim-row A)
    (mat-inverse (A)) ∈ carrier-mat n n
proof (cases mat-inverse(A))
  obtain n where n: A ∈ carrier-mat n n
  using assms invertible-mat-def square-mat.simps by blast
  case None
  then have A ∉ Units (ring-mat TYPE('a) n n)
  by (simp add: mat-inverse(1) n)
  thus ?thesis
  by (meson assms det-non-zero-imp-unit invertible-Units n unit-imp-det-non-zero)

```

```

next
  case (Some a)
  then show ?thesis
  using dim mat-inverse(2) by auto
qed

```

```

lemma mult-assoc:
  fixes A B:: rat mat
  fixes v:: rat vec
  fixes n:: nat
  assumes A ∈ carrier-mat n n
  assumes B ∈ carrier-mat n n
  assumes dim-vec v = n
  shows A *v (mult-mat-vec B v) = (A*B)*v v
  using assms(1) assms(2) assms(3) by auto

```

```

lemma size-of-mat:
  fixes subsets :: nat list list
  fixes signs :: rat list list
  shows (matrix-A signs subsets) ∈ carrier-mat (length subsets) (length signs)
  unfolding matrix-A-def carrier-mat-def by auto

```

```

lemma size-of-lhs:
  fixes p:: real poly
  fixes qs :: real poly list
  fixes signs :: rat list list

```

shows $\text{dim-vec} (\text{construct-lhs-vector } p \text{ } qs \text{ } signs) = \text{length } signs$
unfolding $\text{construct-lhs-vector-def}$
by simp

lemma size-of-rhs :
fixes $p :: \text{real poly}$
fixes $qs :: \text{real poly list}$
fixes $subsets :: \text{nat list list}$
shows $\text{dim-vec} (\text{construct-rhs-vector } p \text{ } qs \text{ } subsets) = \text{length } subsets$
unfolding $\text{construct-rhs-vector-def}$
by simp

lemma same-size :
fixes $p :: \text{real poly}$
fixes $qs :: \text{real poly list}$
fixes $subsets :: \text{nat list list}$
fixes $signs :: \text{rat list list}$
assumes $\text{invertible-mat}: \text{invertible-mat} (\text{matrix-}A \text{ } signs \text{ } subsets)$
shows $\text{length } subsets = \text{length } signs$
using invertible-mat **unfolding** $\text{invertible-mat-def}$
using $\text{size-of-mat}[of \text{ } signs \text{ } subsets]$ $\text{size-of-lhs}[of \text{ } p \text{ } qs \text{ } signs]$ $\text{size-of-rhs}[of \text{ } p \text{ } qs \text{ } subsets]$
by simp

lemma $\text{mat-equal-list-lem}$:
fixes $A :: 'a :: \text{field mat}$
fixes $B :: 'a :: \text{field mat}$
shows $(\text{dim-row } A = \text{dim-row } B \wedge \text{dim-col } A = \text{dim-col } B \wedge \text{mat-to-list } A = \text{mat-to-list } B) \implies A = B$
proof –
assume $\text{hyp}: \text{dim-row } A = \text{dim-row } B \wedge \text{dim-col } A = \text{dim-col } B \wedge \text{mat-to-list } A = \text{mat-to-list } B$
then have $\bigwedge i j. i < \text{dim-row } A \implies j < \text{dim-col } A \implies B \text{ } \$\$ (i, j) = A \text{ } \$\$ (i, j)$
unfolding mat-to-list-def **by** auto
then show $?thesis$ **using** hyp
unfolding mat-to-list-def
using $\text{eq-matI}[of \text{ } A \text{ } B]$
by metis
qed

lemma mat-inverse-same : $\text{mat-inverse-var } A = \text{mat-inverse } A$
unfolding $\text{mat-inverse-var-def}$ mat-inverse-def mat-equal-def
by $(\text{smt} (\text{verit}) \text{mat-equal-list-lem} \text{split-cong})$

lemma $\text{construct-lhs-matches-solve-for-lhs}$:
fixes $p :: \text{real poly}$
fixes $qs :: \text{real poly list}$
fixes $subsets :: \text{nat list list}$

```

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)
by (smt (verit) carrier-vec-dim-vec dim-invertible dim-row-matrix-A invertible-mat
invertible-means-mult-id mat-inverse-same match mult-assoc one-mult-mat-vec same-size
satisfy-equation-def size-of-lhs size-of-mat solve-for-lhs-def)

```

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:  $A \subseteq \text{set}(B)$ 
assumes not-in-hyp:  $\forall n < \text{dim-vec } C. C \$ n = 0 \longrightarrow (\text{nth } B \ n) \notin A$ 
shows  $A \subseteq \text{set}(\text{take-indices } B$ 
    (find-nonzeros-from-input-vec C))
proof –
have unfold:  $\bigwedge x. (x \in A \implies x \in \text{set}(\text{take-indices } B$ 
    (find-nonzeros-from-input-vec C)))
proof –
fix x
assume in-a:  $x \in A$ 
have  $x \in \text{set}(B)$ 
using sub in-a by blast
then have  $\exists n < \text{length } B. \text{nth } B \ n = x$ 
by (simp add: in-set-conv-nth)
then have  $\exists n < \text{length } B. (\text{nth } B \ n = x \wedge (\text{List.member}(\text{find-nonzeros-from-input-vec}$ 
C) n) = True)
using not-in-hyp find-zeros-from-vec-prop[of C]
using assms(1) in-a by auto
thus  $x \in \text{set}(\text{take-indices } B$ 
    (find-nonzeros-from-input-vec C))
unfolding take-indices-def
by fastforce
qed
then show ?thesis
by blast
qed

```

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:  $A \subseteq \text{set}(B)$ 

```

```

assumes not-in-hyp:  $\forall n < \dim\text{-vec } C. C \$ n \neq 0 \longrightarrow (\text{nth } B \ n) \in A$ 
shows set (take-indices B (find-nonzeros-from-input-vec C))  $\subseteq A$ 
proof –
  have unfold:  $\bigwedge x. (x \in (\text{set } (\text{take-indices } B (\text{find-nonzeros-from-input-vec } C)))) \implies x \in A$ 
proof –
  fix x
  assume in-set:  $x \in \text{set } (\text{take-indices } B (\text{find-nonzeros-from-input-vec } C))$ 
  have h:  $\exists n < \dim\text{-vec } C. (C \$ n \neq 0 \wedge (\text{nth } B \ n) = x)$ 
proof –
  have h1:  $\exists n < \text{length } B. (\text{nth } B \ n) = x$ 
    using in-set unfolding take-indices-def find-nonzeros-from-input-vec-def eq-len by auto
  have h2:  $\forall n < \text{length } B. (\text{nth } B \ n = x \longrightarrow \text{List.member } (\text{find-nonzeros-from-input-vec } C) \ n)$ 
proof clarsimp
  fix n
  assume leq-hyp:  $n < \text{length } B$ 
  assume x-eq:  $x = B ! n$ 
  have h:  $(B ! n) \in \text{set } (\text{map } (!) \ B) (\text{find-nonzeros-from-input-vec } C)$ 
    using x-eq in-set
    by (simp add: take-indices-def)
  then have h2:  $\text{List.member } (\text{map } (!) \ B) (\text{find-nonzeros-from-input-vec } C) (B ! n)$ 
    using in-set by auto
  then have h3:  $\exists k < \text{length } B. (\text{List.member } (\text{find-nonzeros-from-input-vec } C) \ k \wedge ((B ! k) = (B ! n)))$ 
    by auto (smt (verit) atLeastLessThan-iff eq-len find-nonzeros-from-input-vec-def imageE mem-Collect-eq set-filter set-map set-upt)
  have h4:  $\forall v < \text{length } B. ((B ! v) = (B ! n) \longrightarrow v = n)$ 
    using dist by (metis find-first-unique leq-hyp)
  then show  $n \in \text{set } (\text{find-nonzeros-from-input-vec } C)$ 
    using h2 h3 h4 by auto
qed
  then have  $\forall n < \text{length } B. (\text{nth } B \ n = x \longrightarrow C \$ n \neq 0)$ 
    using find-zeros-from-vec-prop [of C]
    by (simp add: eq-len)
  then show ?thesis using h1 eq-len
    by auto
qed
thus  $x \in A$  using not-in-hyp
  by blast
qed
then show ?thesis
  by blast
qed

```

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 $nonzero$: $p \neq 0$
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 $set\ (characterize-consistent-signs-at-roots-copr\ p\ qs) \subseteq set(reduction-signs\ p\ qs\ signs\ subsets\ (matrix-A\ signs\ subsets))$
proof –
have $dim-hyp2$: $matr-option\ (dim-row\ (matrix-A\ signs\ subsets))$
 $(mat-inverse\ (matrix-A\ signs\ subsets)) \in carrier-mat\ (length\ signs)\ (length\ signs)$
using *invertible-mat dim-invertible*
using *same-size by fastforce*
have $(construct-lhs-vector\ p\ qs\ signs) = solve-for-lhs\ p\ qs\ subsets\ (matrix-A\ signs\ subsets)$
using *construct-lhs-matches-solve-for-lhs assms by auto*
then have $(solve-for-lhs\ p\ qs\ subsets\ (matrix-A\ signs\ subsets)) =$
 $vec-of-list\ (map\ rat-of-nat\ (map\ (\lambda s. card\ \{x. poly\ p\ x = 0 \wedge consistent-sign-vec-copr\ qs\ x = s\})\ signs))$
using *construct-lhs-vector-cleaner assms*
by (*metis* (*mono-tags, lifting*) *list.map-cong map-map o-apply of-int-of-nat-eq*)
then have $\forall n < (dim-vec\ (solve-for-lhs\ p\ qs\ subsets\ (matrix-A\ signs\ subsets)))$.
 $((solve-for-lhs\ p\ qs\ subsets\ (matrix-A\ signs\ subsets))\ \$\ n = 0) \longrightarrow$
 $(nth\ signs\ n) \notin set\ (characterize-consistent-signs-at-roots-copr\ p\ qs)$
proof –
have $h0$: $\forall n < (dim-vec\ (solve-for-lhs\ p\ qs\ subsets\ (matrix-A\ signs\ subsets)))$.
 $((solve-for-lhs\ p\ qs\ subsets\ (matrix-A\ signs\ subsets))\ \$\ n = 0) \longrightarrow$
 $rat-of-nat\ (card\ \{x. poly\ p\ x = 0 \wedge consistent-sign-vec-copr\ qs\ x = (nth\ signs\ n)\}) = 0$
by (*metis* (*mono-tags, lifting*) $\langle construct-lhs-vector\ p\ qs\ signs = solve-for-lhs\ p\ qs\ subsets\ (matrix-A\ signs\ subsets) \rangle$ *construct-lhs-vector-clean nonzero of-nat-0-eq-iff of-rat-of-nat-eq size-of-lhs*)
have $h1$: $\forall w. (rat-of-nat\ (card\ \{x. poly\ p\ x = 0 \wedge consistent-sign-vec-copr\ qs\ x = w\}) = 0 \longrightarrow$
 $(\nexists x. poly\ p\ x = 0 \wedge consistent-sign-vec-copr\ qs\ x = w))$
proof *clarsimp*
fix x
assume $card-asm$: $card\ \{xa. poly\ p\ xa = 0 \wedge consistent-sign-vec-copr\ qs\ xa = consistent-sign-vec-copr\ qs\ x\} = 0$
assume $zero-asm$: $poly\ p\ x = 0$
have $card-hyp$: $\{xa. poly\ p\ xa = 0 \wedge consistent-sign-vec-copr\ qs\ xa = consis-$

```

tent-sign-vec-copr qs x} = {}
  using card-eq-0-iff
  using card-asm nonzero poly-roots-finite by fastforce
  have x ∈ {xa. poly p xa = 0 ∧ consistent-sign-vec-copr qs xa = consis-
tent-sign-vec-copr qs x}
  using zero-asm by auto
  thus False using card-hyp
  by blast
qed
have h2: ∧ w. (rat-of-nat (card {x. poly p x = 0 ∧ consistent-sign-vec-copr qs
x = w}) = 0 ⇒
  (¬List.member (characterize-consistent-signs-at-roots-copr p qs) w))
  using poly-roots-finite [OF nonzero]
  by (auto simp add: characterize-consistent-signs-at-roots-copr-def character-
ize-root-list-p-def)
  then have h3: ∀ w. rat-of-nat (card {x. poly p x = 0 ∧ consistent-sign-vec-copr
qs x = w}) = 0 →
    w ∉ set (characterize-consistent-signs-at-roots-copr p qs)
  by simp
  show ?thesis using h0 h3
  by blast
qed
then have set (characterize-consistent-signs-at-roots-copr p qs) ⊆ set (take-indices
signs
  (find-nonzeros-from-input-vec (solve-for-lhs p qs subsets (matrix-A signs
subsets))))))
  using all-info
  reduction-signs-set-helper-lemma[where A = set (characterize-consistent-signs-at-roots-copr
p qs), where B = signs,
  where C = (solve-for-lhs p qs subsets (matrix-A signs subsets))]
  using dim-hyp2 solve-for-lhs-def by (simp add: mat-inverse-same)
  then show ?thesis unfolding reduction-signs-def by auto
qed

```

```

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) ⊆ 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 -
  have dim-hyp2: matr-option (dim-row (matrix-A signs subsets))

```

$(\text{mat-inverse } (\text{matrix-A signs subsets})) \in \text{carrier-mat } (\text{length signs}) (\text{length signs})$
using *invertible-mat dim-invertible*
using *same-size by fastforce*
have *supset*: $\text{set } (\text{characterize-consistent-signs-at-roots-copr } p \text{ } qs) \supseteq \text{set}(\text{reduction-signs } p \text{ } qs \text{ } \text{signs subsets } (\text{matrix-A signs subsets}))$
proof –
have $(\text{construct-lhs-vector } p \text{ } qs \text{ } \text{signs}) = \text{solve-for-lhs } p \text{ } qs \text{ } \text{subsets } (\text{matrix-A signs subsets})$
using *construct-lhs-matches-solve-for-lhs assms by auto*
then have $(\text{solve-for-lhs } p \text{ } qs \text{ } \text{subsets } (\text{matrix-A signs subsets})) = \text{vec-of-list } (\text{map } \text{rat-of-nat } (\text{map } (\lambda s. \text{card } \{x. \text{poly } p \text{ } x = 0 \wedge \text{consistent-sign-vec-copr } qs \text{ } x = s\}) \text{signs}))$
using *construct-lhs-vector-cleaner assms*
by $(\text{metis } (\text{mono-tags}, \text{lifting}) \text{list.map-cong map-map o-apply of-int-of-nat-eq})$
then have $\forall n < (\text{dim-vec } (\text{solve-for-lhs } p \text{ } qs \text{ } \text{subsets } (\text{matrix-A signs subsets})))$.
 $((\text{solve-for-lhs } p \text{ } qs \text{ } \text{subsets } (\text{matrix-A signs subsets})) \$ n \neq 0) \longrightarrow$
 $(\text{nth signs } n) \in \text{set } (\text{characterize-consistent-signs-at-roots-copr } p \text{ } qs)$
proof –
have $h0: \forall n < (\text{dim-vec } (\text{solve-for-lhs } p \text{ } qs \text{ } \text{subsets } (\text{matrix-A signs subsets})))$.
 $((\text{solve-for-lhs } p \text{ } qs \text{ } \text{subsets } (\text{matrix-A signs subsets})) \$ n = 0) \longrightarrow$
 $\text{rat-of-nat } (\text{card } \{x. \text{poly } p \text{ } x = 0 \wedge \text{consistent-sign-vec-copr } qs \text{ } x = (\text{nth signs } n)\}) = 0$
by $(\text{metis } (\text{mono-tags}, \text{lifting}) \langle \text{construct-lhs-vector } p \text{ } qs \text{ } \text{signs} = \text{solve-for-lhs } p \text{ } qs \text{ } \text{subsets } (\text{matrix-A signs subsets}) \rangle \text{construct-lhs-vector-clean nonzero of-nat-0-eq-iff of-rat-of-nat-eq size-of-lhs})$
have $h1: \forall w. (\text{rat-of-nat } (\text{card } \{x. \text{poly } p \text{ } x = 0 \wedge \text{consistent-sign-vec-copr } qs \text{ } x = w\}) \neq 0 \longrightarrow$
 $(\exists x. \text{poly } p \text{ } x = 0 \wedge \text{consistent-sign-vec-copr } qs \text{ } x = w))$
proof *clarsimp*
fix w
assume $\text{card-asm}: 0 < \text{card } \{x. \text{poly } p \text{ } x = 0 \wedge \text{consistent-sign-vec-copr } qs \text{ } x = w\}$
show $\exists x. \text{poly } p \text{ } x = 0 \wedge \text{consistent-sign-vec-copr } qs \text{ } x = w$
by $(\text{metis } (\text{mono-tags}, \text{lifting}) \text{Collect-empty-eq card-asm card-eq-0-iff gr-implies-not0})$
qed
have $h2: \bigwedge w. (\text{rat-of-nat } (\text{card } \{x. \text{poly } p \text{ } x = 0 \wedge \text{consistent-sign-vec-copr } qs \text{ } x = w\}) \neq 0 \implies$
 $(\text{List.member } (\text{characterize-consistent-signs-at-roots-copr } p \text{ } qs) w))$
proof *clarsimp*
fix w
assume $\text{card-asm}: 0 < \text{card } \{x. \text{poly } p \text{ } x = 0 \wedge \text{consistent-sign-vec-copr } qs \text{ } x = w\}$
have $h0: \exists x. \text{poly } p \text{ } x = 0 \wedge \text{consistent-sign-vec-copr } qs \text{ } x = w$
using *card-asm*
by $(\text{simp add: } h1)$

```

then show  $w \in \text{set}(\text{characterize-consistent-signs-at-roots-copr } p \text{ } qs)$ 
  unfolding characterize-consistent-signs-at-roots-copr-def
  using nonzero poly-roots-finite characterize-root-list-p-def by fastforce
qed
then have  $h3: \forall w. \text{rat-of-nat}(\text{card}\{x. \text{poly } p \ x = 0 \wedge \text{consistent-sign-vec-copr } qs \ x = w\}) \neq 0 \longrightarrow$ 
   $w \in \text{set}(\text{characterize-consistent-signs-at-roots-copr } p \text{ } qs)$ 
  by simp
show ?thesis using  $h0 \ h3$ 
  by (metis (no-types, lifting)  $\langle \text{solve-for-lhs } p \text{ } qs \text{ subsets } (\text{matrix-A signs subsets}) = \text{vec-of-list}(\text{map } \text{rat-of-nat}(\text{map}(\lambda s. \text{card}\{x. \text{poly } p \ x = 0 \wedge \text{consistent-sign-vec-copr } qs \ x = s\}) \text{ signs})) \rangle \text{dim-vec-of-list length-map nth-map vec-of-list-index}$ )
qed
then have  $\text{set}(\text{take-indices signs } (\text{find-nonzeros-from-input-vec}(\text{solve-for-lhs } p \text{ } qs \text{ subsets } (\text{matrix-A signs subsets})))) \subseteq$ 
 $\text{set}(\text{characterize-consistent-signs-at-roots-copr } p \text{ } qs)$ 
  using all-info
  reduction-signs-set-helper-lemma2 [where  $A = \text{set}(\text{characterize-consistent-signs-at-roots-copr } p \text{ } qs)$ , where  $B = \text{signs}$ ,
  where  $C = (\text{solve-for-lhs } p \text{ } qs \text{ subsets } (\text{matrix-A signs subsets}))$ ]
  using distinct-signs dim-hyp2 solve-for-lhs-def
  by (simp add: mat-inverse-same)
then show ?thesis unfolding reduction-signs-def by auto
qed
have  $\text{subset: set}(\text{characterize-consistent-signs-at-roots-copr } p \text{ } qs) \subseteq \text{set}(\text{reduction-signs } p \text{ } qs \text{ signs subsets } (\text{matrix-A signs subsets}))$ 
  using reduction-doesnt-break-things-signs[of p qs signs subsets] assms
  by blast
then show ?thesis using supset subset by auto
qed

```

theorem *reduce-system-sign-conditions:*

```

fixes  $p :: \text{real poly}$ 
fixes  $qs :: \text{real poly list}$ 
fixes  $\text{subsets} :: \text{nat list list}$ 
fixes  $\text{signs} :: \text{rat list list}$ 
assumes  $\text{nonzero: } p \neq 0$ 
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  $\text{set}(\text{get-signs}(\text{reduce-system } p \text{ } (qs, ((\text{matrix-A signs subsets}), (\text{subsets, signs})))))) = \text{set}(\text{characterize-consistent-signs-at-roots-copr } p \text{ } qs)$ 
  unfolding get-signs-def
  using reduction-deletes-bad-sign-conds[of p qs signs subsets] apply (auto)
  apply (simp add: all-info distinct-signs match nonzero reduction-signs-def welldefined-signs1)

```

```

using nonzero invertible-mat apply (metis snd-conv)
by (metis all-info distinct-signs invertible-mat match nonzero reduction-signs-def
snd-conv welldefined-signs1)

```

14.2 Showing matrix equation preserved when reducing

```

lemma rows-to-keep-lem:
  fixes A:: ('a::field) mat
  shows  $\bigwedge ell. ell \in set (rows-to-keep A) \implies ell < dim-row A$ 
  unfolding rows-to-keep-def
  apply auto
  using rref-pivot-positions
  by (metis carrier-mat-triv gauss-jordan-single(2) gauss-jordan-single(3) index-transpose-mat(3))

```

```

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 -
  have poly-type-hyp: p  $\neq 0$  using nonzero by auto
  have distinct-signs-hyp: distinct (snd (snd (reduce-system p (qs, ((matrix-A signs
subsets), (subsets, signs))))))
  proof -
  let ?sym = (find-nonzeros-from-input-vec (solve-for-lhs p qs subsets (matrix-A
signs subsets)))
  have h1:  $\forall i < length (take-indices signs ?sym). \forall j < length(take-indices signs
?sym).$ 
    i  $\neq j \longrightarrow nth (take-indices signs ?sym) i \neq nth (take-indices signs ?sym) j$ 
    using distinct-signs unfolding take-indices-def
  proof clarsimp
  fix i
  fix j
  assume distinct signs
  assume i < length
    (find-nonzeros-from-input-vec (solve-for-lhs p qs subsets (matrix-A
signs subsets)))

```

```

assume  $j < \text{length}$ 
      ( $\text{find-nonzeros-from-input-vec}$  ( $\text{solve-for-lhs}$   $p$   $qs$   $\text{subsets}$  ( $\text{matrix-A}$ 
 $\text{signs}$   $\text{subsets}$ ))))
assume  $\text{neq-hyp}: i \neq j$ 
assume  $\text{signs} ! (\text{find-nonzeros-from-input-vec}$  ( $\text{solve-for-lhs}$   $p$   $qs$   $\text{subsets}$ 
      ( $\text{matrix-A}$   $\text{signs}$   $\text{subsets}$ )) !  $i$ ) =
       $\text{signs} ! (\text{find-nonzeros-from-input-vec}$  ( $\text{solve-for-lhs}$   $p$   $qs$   $\text{subsets}$ 
      ( $\text{matrix-A}$   $\text{signs}$   $\text{subsets}$ )) !  $j$ )
have  $h1: \text{find-nonzeros-from-input-vec}$  ( $\text{solve-for-lhs}$   $p$   $qs$   $\text{subsets}$ 
      ( $\text{matrix-A}$   $\text{signs}$   $\text{subsets}$ )) !  $i \neq \text{find-nonzeros-from-input-vec}$  ( $\text{solve-for-lhs}$ 
 $p$   $qs$   $\text{subsets}$ 
      ( $\text{matrix-A}$   $\text{signs}$   $\text{subsets}$ )) !  $j$ 
      unfolding  $\text{find-nonzeros-from-input-vec-def}$  using  $\text{neq-hyp}$ 
      by ( $\text{metis}$   $\langle i < \text{length}$  ( $\text{find-nonzeros-from-input-vec}$  ( $\text{solve-for-lhs}$   $p$   $qs$   $\text{subsets}$ 
      ( $\text{matrix-A}$   $\text{signs}$   $\text{subsets}$ ))))  $\rangle \langle j < \text{length}$  ( $\text{find-nonzeros-from-input-vec}$  ( $\text{solve-for-lhs}$ 
       $p$   $qs$   $\text{subsets}$  ( $\text{matrix-A}$   $\text{signs}$   $\text{subsets}$ ))))  $\rangle \text{distinct-conv-nth}$   $\text{distinct-filter}$   $\text{distinct-upt}$ 
 $\text{find-nonzeros-from-input-vec-def}$ )
      then show  $\text{False}$  using  $\text{distinct-signs}$ 
      proof –
      have  $f1: \forall p$   $ns$   $n. ((n::\text{nat}) \in \{n \in \text{set } ns. p\ n\}) = (n \in \text{set } ns \wedge n \in \text{Collect}$ 
 $p)$ 
      by  $\text{simp}$ 
      then have  $f2: \text{filter}$  ( $\lambda n. \text{solve-for-lhs}$   $p$   $qs$   $\text{subsets}$  ( $\text{matrix-A}$   $\text{signs}$   $\text{subsets}$ )
       $\$ n \neq 0$ ) [ $0..<\text{dim-vec}$  ( $\text{solve-for-lhs}$   $p$   $qs$   $\text{subsets}$  ( $\text{matrix-A}$   $\text{signs}$   $\text{subsets}$ ))] !  $i \in$ 
 $\text{set}$  [ $0..<\text{length}$   $\text{signs}$ ]
      by ( $\text{metis}$  ( $\text{full-types}$ )  $\langle i < \text{length}$  ( $\text{find-nonzeros-from-input-vec}$  ( $\text{solve-for-lhs}$ 
       $p$   $qs$   $\text{subsets}$  ( $\text{matrix-A}$   $\text{signs}$   $\text{subsets}$ ))))  $\rangle \text{construct-lhs-matches-solve-for-lhs}$   $\text{find-nonzeros-from-input-vec-def}$ 
 $\text{invertible-mat}$   $\text{match}$   $\text{nth-mem}$   $\text{set-filter}$   $\text{size-of-lhs}$ )
      have  $\text{filter}$  ( $\lambda n. \text{solve-for-lhs}$   $p$   $qs$   $\text{subsets}$  ( $\text{matrix-A}$   $\text{signs}$   $\text{subsets}$ )  $\$ n$ 
       $\neq 0$ ) [ $0..<\text{dim-vec}$  ( $\text{solve-for-lhs}$   $p$   $qs$   $\text{subsets}$  ( $\text{matrix-A}$   $\text{signs}$   $\text{subsets}$ ))] !  $j \in \text{set}$ 
      [ $0..<\text{length}$   $\text{signs}$ ]
      using  $f1$  by ( $\text{metis}$  ( $\text{full-types}$ )  $\langle j < \text{length}$  ( $\text{find-nonzeros-from-input-vec}$ 
      ( $\text{solve-for-lhs}$   $p$   $qs$   $\text{subsets}$  ( $\text{matrix-A}$   $\text{signs}$   $\text{subsets}$ ))))  $\rangle \text{construct-lhs-matches-solve-for-lhs}$ 
 $\text{find-nonzeros-from-input-vec-def}$   $\text{invertible-mat}$   $\text{match}$   $\text{nth-mem}$   $\text{set-filter}$   $\text{size-of-lhs}$ )
      then show  $?thesis$ 
      using  $f2$  by ( $\text{metis}$   $\langle \text{signs} ! (\text{find-nonzeros-from-input-vec}$  ( $\text{solve-for-lhs}$ 
       $p$   $qs$   $\text{subsets}$  ( $\text{matrix-A}$   $\text{signs}$   $\text{subsets}$ )) !  $i$ ) =  $\text{signs} ! (\text{find-nonzeros-from-input-vec}$ 
      ( $\text{solve-for-lhs}$   $p$   $qs$   $\text{subsets}$  ( $\text{matrix-A}$   $\text{signs}$   $\text{subsets}$ )) !  $j$ )  $\rangle \text{atLeastLessThan-iff}$   $\text{dis-}$ 
 $\text{tinct-conv-nth}$   $\text{distinct-signs}$   $\text{find-nonzeros-from-input-vec-def}$   $h1$   $\text{set-upt}$ )
      qed
qed
then have  $\text{distinct}$  ( $\text{take-indices}$   $\text{signs}$  ( $\text{find-nonzeros-from-input-vec}$  ( $\text{solve-for-lhs}$ 
       $p$   $qs$   $\text{subsets}$  ( $\text{matrix-A}$   $\text{signs}$   $\text{subsets}$ ))))
      using  $\text{distinct-conv-nth}$  by  $\text{blast}$ 
      then show  $?thesis$ 
      using  $\text{get-signs-def}$   $\text{reduction-signs-def}$   $\text{reduction-signs-is-get-signs}$  by  $\text{auto}$ 
qed
have  $\text{all-info-hyp}: \text{set}$  ( $\text{characterize-consistent-signs-at-roots-copr}$   $p$   $qs$ )  $\subseteq \text{set}$ ( $\text{snd}$ 
      ( $\text{snd}$  ( $\text{reduce-system}$   $p$  ( $qs$ , (( $\text{matrix-A}$   $\text{signs}$   $\text{subsets}$ ), ( $\text{subsets}$ ,  $\text{signs}$ ))))))

```

```

using reduce-system-sign-conditions assms unfolding get-signs-def by auto
have pairwise-rel-prime-hyp:  $\forall q. ((\text{List.member } qs \ q) \longrightarrow (\text{coprime } p \ q))$ 
using pairwise-rel-prime by auto
have welldefined-hyp: all-list-constr (fst (snd (reduce-system p (qs, ((matrix-A
signs subsets), (subsets, signs)))))) (length qs)
using welldefined-subsets rows-to-keep-lem
unfolding all-list-constr-def list-constr-def list-all-def
apply (auto simp add: Let-def take-indices-def take-cols-from-matrix-def)
using nth-mem by fastforce
then show ?thesis using poly-type-hyp distinct-signs-hyp all-info-hyp pairwise-rel-prime-hyp
welldefined-hyp
using matrix-equation unfolding get-subsets-def get-signs-def
by blast
qed

```

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:  $\forall x. \text{List.member } S \ x \longrightarrow x < \text{length } signs$ 
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 –
have h0a:  $\text{dim-col } (\text{take-cols-from-matrix } (\text{alt-matrix-A } signs \ subsets) \ S) = \text{length}$ 
(take-indices signs S)
unfolding take-cols-from-matrix-def apply (auto) unfolding take-indices-def
by auto
have h0:  $\forall i < \text{length } (\text{take-indices } signs \ S). (\text{col } (\text{alt-matrix-A } (\text{take-indices } signs \ S) \ subsets) \ i =$ 
col (take-cols-from-matrix (alt-matrix-A signs subsets) S) i)
proof clarsimp
fix i
assume asm:  $i < \text{length } (\text{take-indices } signs \ S)$ 
have i-lt:  $i < \text{length } (\text{map } (!) (\text{cols } (\text{alt-matrix-A } signs \ subsets))) \ S)$  using
asm
apply (auto) unfolding take-indices-def by auto
have h0:  $\text{vec } (\text{length } subsets) (\lambda j. z (\text{subsets } ! \ j) (\text{map } (!) \ signs) \ S \ ! \ i)) =$ 
vec (length subsets) ( $\lambda j. z (\text{subsets } ! \ j) (\text{signs } ! \ (S \ ! \ i))$ ) using nth-map
by (metis <i < length (take-indices signs S)> length-map take-indices-def)
have dim: (map (!) (cols (alt-matrix-A signs subsets))) S) ! i  $\in \text{carrier-vec}$ 
(dim-row (alt-matrix-A signs subsets))
proof –
have dim-col (alt-matrix-A signs subsets) = length (signs)
by (simp add: alt-matrix-A-def)

```

```

have well-d:  $S ! i < \text{length } (\text{signs})$  using well-def-h
using i-lt by fastforce
have
  map-eq:  $(\text{map } (!) (\text{cols } (\text{alt-matrix-A signs subsets}))) S ! i = \text{nth } (\text{cols } (\text{alt-matrix-A signs subsets})) (S ! i)$ 
using i-lt by auto
have nth  $(\text{cols } (\text{alt-matrix-A signs subsets})) (S ! i) \in \text{carrier-vec } (\text{dim-row } (\text{alt-matrix-A signs subsets}))$ 
using col-dim unfolding cols-def using nth-map well-d
by (simp add:  $\langle \text{dim-col } (\text{alt-matrix-A signs subsets}) = \text{length signs} \rangle$ )
then show ?thesis using map-eq unfolding alt-matrix-A-def by auto
qed
have h1:  $\text{col } (\text{take-cols-from-matrix } (\text{alt-matrix-A signs subsets}) S) i = \text{col } (\text{mat-of-cols } (\text{dim-row } (\text{alt-matrix-A signs subsets})) (\text{map } (!) (\text{cols } (\text{alt-matrix-A signs subsets}))) S) i$ 
by (simp add: take-cols-from-matrix-def take-indices-def)
have h2:  $\text{col } (\text{mat-of-cols } (\text{dim-row } (\text{alt-matrix-A signs subsets})) (\text{map } (!) (\text{cols } (\text{alt-matrix-A signs subsets}))) S) i = \text{nth } (\text{map } (!) (\text{cols } (\text{alt-matrix-A signs subsets}))) S) i$ 
using dim i-lt asm col-mat-of-cols[where  $j = i$ , where  $n = (\text{dim-row } (\text{alt-matrix-A signs subsets}))$ ,
where  $vs = (\text{map } (!) (\text{cols } (\text{alt-matrix-A signs subsets}))) S]$ ]
by blast
have h3:  $\text{col } (\text{take-cols-from-matrix } (\text{alt-matrix-A signs subsets}) S) i = (\text{col } (\text{alt-matrix-A signs subsets}) (S ! i))$ 
using h1 h2 asm well-def-h i-lt by (auto simp add: alt-matrix-char)
have vec  $(\text{length subsets}) (\lambda j. z (\text{subsets } ! j) (\text{signs } ! (S ! i))) = (\text{col } (\text{alt-matrix-A signs subsets}) (S ! i))$ 
using asm well-def-h by (auto simp add: take-indices-def signs-are-cols)
then have vec  $(\text{length subsets}) (\lambda j. z (\text{subsets } ! j) (\text{take-indices signs } S ! i)) = \text{col } (\text{take-cols-from-matrix } (\text{alt-matrix-A signs subsets}) S) i$ 
using h0 h3
by (simp add: take-indices-def)
then show  $\text{col } (\text{alt-matrix-A } (\text{take-indices signs } S) \text{ subsets}) i = \text{col } (\text{take-cols-from-matrix } (\text{alt-matrix-A signs subsets}) S) i$ 
using asm signs-are-cols[where  $\text{signs} = (\text{take-indices signs } S)$ , where  $\text{subsets} = \text{subsets}$ ]
by auto
qed
then show ?thesis unfolding alt-matrix-A-def take-cols-from-matrix-def apply (auto)
using h0a mat-col-eqI
by (metis alt-matrix-A-def dim-col-mat(1) dim-row-mat(1) h0 mat-of-cols-def take-cols-from-matrix-def)
qed

```

lemma reduce-system-matrix-signs-helper:
fixes $p:: \text{real poly}$

```

fixes qs :: real poly list
fixes subsets :: nat list list
fixes signs :: rat list list
fixes S:: nat list
assumes well-def-h:  $\forall x. \text{List.member } S \ x \longrightarrow x < \text{length } \text{signs}$ 
assumes nonzero:  $p \neq 0$ 
shows matrix-A (take-indices signs S) subsets = take-cols-from-matrix (matrix-A
signs subsets) S
using reduce-system-matrix-signs-helper-aux alt-matrix-char assms by auto

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:  $\text{length } \text{subsets} \geq \text{length } \text{signs}$ 
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 signs (take-indices subsets S) = take-rows-from-matrix (alt-matrix-A
signs subsets) S
proof –
have h0a:  $\text{dim-row } (\text{take-rows-from-matrix } (\text{alt-matrix-A } \text{signs } \text{subsets}) \ S) =$ 
 $\text{length } (\text{take-indices } \text{subsets } \ S)$ 
unfolding take-rows-from-matrix-def apply (auto) unfolding take-indices-def
by auto
have h0:  $\forall i < \text{length } (\text{take-indices } \text{subsets } \ S). (\text{row } (\text{alt-matrix-A } \text{signs } (\text{take-indices}$ 
 $\text{subsets } \ S)) \ i =$ 
 $\text{row } (\text{take-rows-from-matrix } (\text{alt-matrix-A } \text{signs } \text{subsets}) \ S) \ i)$ 
proof clarsimp
fix i
assume asm:  $i < \text{length } (\text{take-indices } \text{subsets } \ S)$ 
have i-lt:  $i < \text{length } (\text{map } (!) (\text{rows } (\text{alt-matrix-A } \text{signs } \text{subsets}))) \ S)$  using
asm
apply (auto) unfolding take-indices-def by auto
have h0:  $\text{row } (\text{take-rows-from-matrix } (\text{alt-matrix-A } \text{signs } \text{subsets}) \ S) \ i =$ 
 $\text{row } (\text{mat-of-rows } (\text{dim-col } (\text{alt-matrix-A } \text{signs } \text{subsets})) (\text{map } (!) (\text{rows } (\text{alt-matrix-A}$ 
 $\text{signs } \text{subsets}))) \ S)) \ i$ 
unfolding take-rows-from-matrix-def take-indices-def by auto
have dim:  $(\text{map } (!) (\text{rows } (\text{alt-matrix-A } \text{signs } \text{subsets}))) \ S) ! i \in \text{carrier-vec}$ 
 $(\text{dim-col } (\text{alt-matrix-A } \text{signs } \text{subsets}))$ 
proof –
have dim-col (alt-matrix-A signs subsets) =  $\text{length } (\text{signs})$ 
by (simp add: alt-matrix-A-def)
then have lenh:  $\text{dim-col } (\text{alt-matrix-A } \text{signs } \text{subsets}) \leq \text{length } (\text{subsets})$ 
using assms
by auto
have well-d:  $S ! i < \text{length } (\text{subsets})$  using well-def-h
using i-lt by fastforce

```

```

have
  map-eq: (map (!) (rows (alt-matrix-A signs subsets))) S) ! i = nth (rows
(alt-matrix-A signs subsets)) (S ! i)
  using i-lt by auto
  have nth (rows (alt-matrix-A signs subsets)) (S ! i) ∈ carrier-vec (dim-col
(alt-matrix-A signs subsets))
  using col-dim unfolding rows-def using nth-map well-d
  using lenh
  by (simp add: alt-matrix-A-def)
  then show ?thesis using map-eq unfolding alt-matrix-A-def by auto
qed
  have h1: row (mat-of-rows (dim-col (alt-matrix-A signs subsets)) (map (!)
(rows (alt-matrix-A signs subsets))) S)) i
    = (row (alt-matrix-A signs subsets) (S ! i))
  using dim i-lt mat-of-rows-row[where i = i, where n = (dim-col (alt-matrix-A
signs subsets)),
    where vs = (map (!) (rows (alt-matrix-A signs subsets))) S]]
  using alt-matrix-char nth-mem well-def-h by fastforce
  have row (alt-matrix-A signs (take-indices subsets S)) i = (row (alt-matrix-A
signs subsets) (S ! i))
  using subsets-are-rows
proof –
  have f1: i < length S
    by (metis (no-types) asm length-map take-indices-def)
  then have List.member S (S ! i)
    by simp
  then show ?thesis
    using f1 by (simp add: ⟨ $\bigwedge$  subsets signs.  $\forall i < \text{length subsets. row (alt-matrix-A signs subsets) } i = \text{vec (length signs) } (\lambda j. z (\text{subsets ! } i) (\text{signs ! } j))$ ⟩ take-indices-def well-def-h)
qed
  then show (row (alt-matrix-A signs (take-indices subsets S)) i =
row (take-rows-from-matrix (alt-matrix-A signs subsets) S) i)
    using h0 h1 unfolding take-indices-def by auto
qed
then show ?thesis unfolding alt-matrix-A-def take-rows-from-matrix-def apply
(auto)
  using eq-rowI
  by (metis alt-matrix-A-def dim-col-mat(1) dim-row-mat(1) h0 length-map
mat-of-rows-def take-indices-def take-rows-from-matrix-def)
qed

```

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: $\text{length subsets} \geq \text{length signs}$
assumes well-def-h: $\forall x. \text{List.member } S \ x \longrightarrow x < \text{length subsets}$
shows matrix-A signs (take-indices subsets S) = take-rows-from-matrix (matrix-A signs subsets) S
using assms reduce-system-matrix-subsets-helper-aux alt-matrix-char
by auto

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: $\text{set} (\text{characterize-consistent-signs-at-roots-copr } p \text{ } qs) \subseteq \text{set}(\text{signs})$
assumes match: satisfy-equation p qs subsets signs
assumes inv: invertible-mat (matrix-A signs subsets)
shows matrix-A (get-signs (reduce-system p (qs, ((matrix-A signs subsets), (subsets, signs))))))
 (get-subsets (reduce-system p (qs, ((matrix-A signs subsets), (subsets, signs))))))
 =
 (get-matrix (reduce-system p (qs, ((matrix-A signs subsets), (subsets, signs))))))
proof –
let ?A = (matrix-A signs subsets)
let ?lhs-vec = (solve-for-lhs p qs subsets (matrix-A signs subsets))
let ?red-mtx = (take-rows-from-matrix (reduce-mat-cols (matrix-A signs subsets) ?lhs-vec) (rows-to-keep (reduce-mat-cols (matrix-A signs subsets) ?lhs-vec)))
have h1: matrix-A (get-signs (reduce-system p (qs, ((matrix-A signs subsets), (subsets, signs))))))
 (get-subsets (reduce-system p (qs, ((matrix-A signs subsets), (subsets, signs))))))
 = matrix-A (reduction-signs p qs signs subsets (matrix-A signs subsets)) (reduction-subsets p qs signs subsets (matrix-A signs subsets))
using reduction-signs-is-get-signs reduction-subsets-is-get-subsets **by** auto
have h1-var: matrix-A (get-signs (reduce-system p (qs, ((matrix-A signs subsets), (subsets, signs))))))
 (get-subsets (reduce-system p (qs, ((matrix-A signs subsets), (subsets, signs))))))
 = matrix-A (take-indices signs (find-nonzeros-from-input-vec ?lhs-vec)) (take-indices subsets (rows-to-keep (reduce-mat-cols ?A ?lhs-vec)))
using h1 reduction-signs-def reduction-subsets-def **by** auto
have h2: ?red-mtx = (take-rows-from-matrix (take-cols-from-matrix ?A (find-nonzeros-from-input-vec ?lhs-vec)) (rows-to-keep (take-cols-from-matrix ?A (find-nonzeros-from-input-vec ?lhs-vec))))
by simp
have h30: (construct-lhs-vector p qs signs) = ?lhs-vec
using assms construct-lhs-matches-solve-for-lhs
by simp
have h3a: $\forall x. \text{List.member} (\text{find-nonzeros-from-input-vec } ?\text{lhs-vec}) \ x \longrightarrow x < \text{length} (\text{signs})$

```

using h30 size-of-lhs unfolding find-nonzeros-from-input-vec-def apply (auto)
by (metis atLeastLessThan-iff filter-is-subset set-upt subset-eq)
have h3b-a:  $\forall x. \text{List.member (find-nonzeros-from-input-vec ?lhs-vec) } x \longrightarrow x <$ 
length (subsets)
using assms h30 size-of-lhs same-size unfolding find-nonzeros-from-input-vec-def
apply (auto)
by (simp add: find-nonzeros-from-input-vec-def h3a)
have h3ba: length
  (filter ( $\lambda i. \text{solve-for-lhs } p \text{ } qs \text{ } subsets \text{ (matrix-A signs subsets) } \$ i \neq 0$ )
    [0.. $\text{length subsets}$ ])
   $\leq$  length subsets using length-filter-le[where  $P = (\lambda i. \text{solve-for-lhs } p \text{ } qs \text{ } subsets$ 
(matrix-A signs subsets)  $\$ i \neq 0$ ),
where  $xs = [0.. $\text{length subsets}$ ]]$  length-upto by auto
have length subsets = dim-vec (solve-for-lhs p qs subsets (matrix-A signs subsets))
using h30 inv size-of-lhs same-size[of signs subsets] apply (auto)
by metis
then have length
  (filter ( $\lambda i. \text{solve-for-lhs } p \text{ } qs \text{ } subsets \text{ (matrix-A signs subsets) } \$ i \neq 0$ )
    [0.. $\text{dim-vec (solve-for-lhs } p \text{ } qs \text{ } subsets \text{ (matrix-A signs subsets))}$ ])
   $\leq$  length subsets using h3ba
by auto
then have h3b: length subsets  $\geq$  length (take-indices signs (find-nonzeros-from-input-vec
?lhs-vec))
unfolding take-indices-def find-nonzeros-from-input-vec-def by auto
have h3c:  $\forall x. \text{List.member (rows-to-keep (reduce-mat-cols ?A ?lhs-vec)) } x \longrightarrow x$ 
 $<$  length (subsets)
proof clarsimp
fix x
assume x-mem:  $x \in \text{set (rows-to-keep$ 
  (take-cols-from-matrix (matrix-A signs subsets)
    (find-nonzeros-from-input-vec (solve-for-lhs p qs subsets (matrix-A signs
subsets))))))
obtain nn :: rat list list  $\Rightarrow$  nat list  $\Rightarrow$  nat where
 $\forall x2 \ x3. (\exists v4. v4 \in \text{set } x3 \wedge \neg v4 < \text{length } x2) = (\text{nn } x2 \ x3 \in \text{set } x3 \wedge \neg$ 
nn x2 x3  $<$  length x2)
by moura
then have f4: nn signs (find-nonzeros-from-input-vec (solve-for-lhs p qs sub-
sets (matrix-A signs subsets)))  $\in$  set (find-nonzeros-from-input-vec (solve-for-lhs p
qs subsets (matrix-A signs subsets)))  $\wedge \neg$  nn signs (find-nonzeros-from-input-vec
(solve-for-lhs p qs subsets (matrix-A signs subsets)))  $<$  length signs  $\vee$  matrix-A
(take-indices signs (find-nonzeros-from-input-vec (solve-for-lhs p qs subsets (matrix-A
signs subsets)))) subsets = take-cols-from-matrix (matrix-A signs subsets) (find-nonzeros-from-input-vec
(solve-for-lhs p qs subsets (matrix-A signs subsets)))
using h3a nonzero reduce-system-matrix-signs-helper by auto
then have matrix-A (take-indices signs (find-nonzeros-from-input-vec (solve-for-lhs
p qs subsets (matrix-A signs subsets)))) subsets = take-cols-from-matrix (matrix-A
signs subsets) (find-nonzeros-from-input-vec (solve-for-lhs p qs subsets (matrix-A
signs subsets)))  $\wedge x \in \text{set (map snd (pivot-positions (gauss-jordan-single (take-cols-from-matrix$ 
(matrix-A signs subsets) (find-nonzeros-from-input-vec (solve-for-lhs p qs subsets

```

```

(matrix-A signs subsets))))T)))
  using f4 h3a x-mem by (auto simp add: rows-to-keep-def)
  thus x < length (subsets) using x-mem unfolding rows-to-keep-def
  by (metis (no-types) dim-row-matrix-A rows-to-keep-def rows-to-keep-lem)
qed
  have h3: matrix-A (take-indices signs (find-nonzeros-from-input-vec ?lhs-vec))
(take-indices subsets (rows-to-keep (reduce-mat-cols ?A ?lhs-vec))) =
  (take-rows-from-matrix (take-cols-from-matrix ?A (find-nonzeros-from-input-vec
?lhs-vec)) (rows-to-keep (take-cols-from-matrix ?A (find-nonzeros-from-input-vec ?lhs-vec))))

  using inv h3a h3b h3c reduce-system-matrix-subsets-helper reduce-system-matrix-signs-helper
  assms by auto
  show ?thesis using h1 h2 h3
  by (metis fst-conv get-matrix-def h1-var reduce-system.simps reduction-step.simps)
qed

```

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 = length signs
  assumes inv: invertible-mat (matrix-A 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-copr p qs) ⊆ set(signs)
  assumes match: satisfy-equation p qs subsets signs
  shows (∧j. j ∈ set (find-nonzeros-from-input-vec
    (solve-for-lhs p qs subsets (matrix-A signs subsets)))) ⇒
    j < length (cols (matrix-A signs subsets)))
proof -
  fix i
  fix j
  assume j-in: j ∈ set (find-nonzeros-from-input-vec
    (solve-for-lhs p qs subsets (matrix-A signs subsets)))
  let ?og-mat = (matrix-A signs subsets)
  let ?lhs = (solve-for-lhs p qs subsets ?og-mat)
  let ?new-mat = (take-rows-from-matrix (reduce-mat-cols ?og-mat ?lhs) (rows-to-keep
(reduce-mat-cols ?og-mat ?lhs)))
  have square-mat (matrix-A signs subsets) using inv
  using invertible-mat-def by blast
  then have mat-size: ?og-mat ∈ carrier-mat (length signs) (length signs)
  using size-of-mat
  by auto
  have dim-vec (solve-for-lhs p qs subsets (matrix-A signs subsets)) = (length signs)
  using size-of-lhs construct-lhs-matches-solve-for-lhs assms

```

```

    by (metis (full-types))
  then have h: j < (length signs)
    using j-in unfolding find-nonzeros-from-input-vec-def
    by simp
  then show j < length (cols (matrix-A signs subsets))
    using mat-size by auto
qed

```

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 = length signs
  assumes inv: invertible-mat (matrix-A 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-copr p qs) ⊆ 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))))
    ⊆ set (cols (matrix-A signs subsets))

```

proof –

```

  have well-def: (∧j. j ∈ set (find-nonzeros-from-input-vec
    (solve-for-lhs p qs subsets (matrix-A signs subsets))) ⇒
    j < length (cols (matrix-A signs subsets)))
    using assms well-def-find-zeros-from-lhs-vec by auto
  have ∀ x. x ∈ set (take-indices (cols (matrix-A signs subsets))
    (find-nonzeros-from-input-vec (solve-for-lhs p qs subsets (matrix-A signs
subsets))))
    → x ∈ set (cols (matrix-A signs subsets))

```

proof *clarsimp*

```

  fix x
  let ?og-list = (cols (matrix-A signs subsets))
  let ?ind-list = (find-nonzeros-from-input-vec
    (solve-for-lhs p qs subsets (matrix-A signs subsets)))
  assume x-in: x ∈ set (take-indices ?og-list ?ind-list)
  show x ∈ set (cols (matrix-A signs subsets))
    using x-in unfolding take-indices-def apply (auto)
    using in-set-conv-nth well-def by fastforce

```

qed

then show *?thesis*

by *blast*

qed

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 ≠ 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
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 –
  let ?og-mat = (matrix-A signs subsets)
  let ?lhs = (solve-for-lhs p qs subsets ?og-mat)
  let ?new-mat = (take-rows-from-matrix (reduce-mat-cols ?og-mat ?lhs) (rows-to-keep (reduce-mat-cols ?og-mat ?lhs)))
  have square-mat (matrix-A signs subsets) using inv using invertible-mat-def by blast
  then have mat-size: ?og-mat ∈ carrier-mat (length signs) (length signs) using size-of-mat by auto
  then have mat-size-alt: ?og-mat ∈ carrier-mat (length subsets) (length subsets) using size-of-mat same-size assms by auto
  have det-h: det ?og-mat ≠ 0 using invertible-det[where A = matrix-A signs subsets] mat-size using inv by blast
  then have rank-h: vec-space.rank (length signs) ?og-mat = (length signs) using vec-space.det-rank-iff mat-size by auto
  then have dist-cols: distinct (cols ?og-mat) using mat-size vec-space.non-distinct-low-rank[where A = ?og-mat, where n = length signs] by auto
  have well-def: (∧j. j ∈ set (find-nonzeros-from-input-vec (solve-for-lhs p qs subsets (matrix-A signs subsets))) ⇒ j < length (cols (matrix-A signs subsets))) using assms well-def-find-zeros-from-lhs-vec by auto
  have dist1: distinct (find-nonzeros-from-input-vec (solve-for-lhs p qs subsets (matrix-A signs subsets)))
  unfolding find-nonzeros-from-input-vec-def by auto
  have clear: set (take-indices (cols (matrix-A signs subsets)) (find-nonzeros-from-input-vec (solve-for-lhs p qs subsets (matrix-A signs subsets)))) ⊆ set (cols (matrix-A signs subsets)) using assms take-cols-subsets-og-cols by auto

```

then have *distinct* (*take-indices* (*cols* (*matrix-A signs subsets*))
(find-nonzeros-from-input-vec (*solve-for-lhs p qs subsets* (*matrix-A signs subsets*))))
unfolding *take-indices-def*
using *dist1 dist-cols well-def conjugatable-vec-space.distinct-map-nth* [**where**
ls = cols (*matrix-A signs subsets*), **where** *inds = (find-nonzeros-from-input-vec*
(solve-for-lhs p qs subsets (*matrix-A signs subsets*)))]
by *auto*
then have *unfold-thesis: vec-space.rank* (*length signs*) (*mat-of-cols* (*dim-row*
?og-mat) (*take-indices* (*cols* ?*og-mat*) (*find-nonzeros-from-input-vec* ?*lhs*)))
= (*length* (*find-nonzeros-from-input-vec* ?*lhs*))
using *clear inv conjugatable-vec-space.rank-invertible-subset-cols* [**where** *A =*
matrix-A signs subsets, **where** *B = (take-indices* (*cols* (*matrix-A signs subsets*))
(find-nonzeros-from-input-vec (*solve-for-lhs p qs subsets* (*matrix-A signs subsets*))))]
by (*simp add: len-eq mat-size take-indices-def*)
then show ?*thesis* **apply** (*simp*) **unfolding** *take-cols-from-matrix-def* **by** *auto*
qed

lemma *rechar-take-cols: take-cols-var A B = take-cols-from-matrix A B*
unfolding *take-cols-var-def take-cols-from-matrix-def take-indices-def* **by** *auto*

lemma *rows-and-cols-transpose: rows M = cols M^T*
using *row-transpose* **by** *simp*

lemma *take-rows-and-take-cols: (take-rows-from-matrix M r) = (take-cols-from-matrix*
M^T r)^T
unfolding *take-rows-from-matrix-def take-cols-from-matrix-def*
using *transpose-carrier-mat rows-and-cols-transpose* **apply** (*auto*)
by (*simp add: transpose-mat-of-cols*)

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 ≠ 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 –
let ?*og-mat* = (*matrix-A signs subsets*)
let ?*lhs* = (*solve-for-lhs p qs subsets* ?*og-mat*)
let ?*step1-mat* = (*reduce-mat-cols* ?*og-mat* ?*lhs*)

```

let ?new-mat = take-rows-from-matrix ?step1-mat (rows-to-keep ?step1-mat)
let ?ind-list = (find-nonzeros-from-input-vec ?lhs)
have square-mat (matrix-A signs subsets) using inv
  using invertible-mat-def by blast
then have og-mat-size: ?og-mat ∈ carrier-mat (length signs) (length signs)
  using size-of-mat
  by auto
have dim-col (mat-of-cols (dim-row ?og-mat) (take-indices (cols ?og-mat) ?ind-list))
  = (length (find-nonzeros-from-input-vec ?lhs))
  by (simp add: take-indices-def)
then have mat-of-cols (dim-row ?og-mat) (take-indices (cols ?og-mat) ?ind-list)
  ∈ carrier-mat (length signs) (length (find-nonzeros-from-input-vec ?lhs))
  by (simp add: len-eq mat-of-cols-def)
then have step1-mat-size: ?step1-mat ∈ carrier-mat (length signs) (length (find-nonzeros-from-input-vec
?lhs))
  by (simp add: take-cols-from-matrix-def)
then have dim-row ?step1-mat ≥ dim-col ?step1-mat
  by (metis carrier-matD(1) carrier-matD(2) construct-lhs-matches-solve-for-lhs
diff-zero find-nonzeros-from-input-vec-def inv length-filter-le length-upt match size-of-lhs)
then have gt-eq-asm: dim-col ?step1-matT ≥ dim-row ?step1-matT
  by simp
have det-h: det ?og-mat ≠ 0
  using invertible-det[where A = matrix-A signs subsets] og-mat-size
  using inv by blast
then have rank-h: vec-space.rank (length signs) ?og-mat = (length signs)
  using vec-space.det-rank-iff og-mat-size
  by auto
have rank-drop-cols: vec-space.rank (length signs) ?step1-mat = (dim-col ?step1-mat)
  using assms reduction-doesnt-break-things-invertibility-step1 step1-mat-size
  by auto
let ?step1-T = ?step1-matT
have rank-transpose: vec-space.rank (length signs) ?step1-mat = vec-space.rank
(length (find-nonzeros-from-input-vec ?lhs)) ?step1-T
  using transpose-rank[of ?step1-mat]
  using step1-mat-size by auto
have obv: ?step1-T ∈ carrier-mat (dim-row ?step1-T) (dim-col ?step1-T) by
auto
have should-have-this: vec-space.rank (length (find-nonzeros-from-input-vec ?lhs))
(take-cols ?step1-T (map snd (pivot-positions (gauss-jordan-single (?step1-T))))))
= vec-space.rank (length (find-nonzeros-from-input-vec ?lhs)) ?step1-T
  using obv gt-eq-asm conjugatable-vec-space.gauss-jordan-single-rank[where A
= ?step1-T, where n = dim-row ?step1-T, where nc = dim-col ?step1-T]
  by (simp add: take-cols-from-matrix-def take-indices-def)
then have vec-space.rank (length (find-nonzeros-from-input-vec ?lhs)) (take-cols
?step1-T (map snd (pivot-positions (gauss-jordan-single (?step1-T)))))) = dim-col
?step1-mat
  using rank-drop-cols rank-transpose by auto
then have with-take-cols-from-matrix: vec-space.rank (length (find-nonzeros-from-input-vec
?lhs)) (take-cols-from-matrix ?step1-T (map snd (pivot-positions (gauss-jordan-single

```

```

(?step1-T)))))) = dim-col ?step1-mat
  by (metis rechar-take-cols conjugatable-vec-space.gjs-and-take-cols-var gt-eq-assm
obv)
  have (take-rows-from-matrix ?step1-mat (rows-to-keep ?step1-mat)) = (take-cols-from-matrix
?step1-T (rows-to-keep ?step1-mat))T
    using take-rows-and-take-cols
    by blast
  then have rank-new-mat: vec-space.rank (dim-row ?new-mat) ?new-mat = dim-col
?step1-mat
    using with-take-cols-from-matrix transpose-rank
    by (metis carrier-matD(2) index-transpose-mat(2) mat-of-cols-carrier(2) rows-to-keep-def
step1-mat-size take-cols-from-matrix-def)
  have length (pivot-positions (gauss-jordan-single (?step1-matT))) ≤ (length (find-nonzeros-from-input-vec
?lhs))
    using obv length-pivot-positions-dim-row[where A = (gauss-jordan-single (?step1-matT))]
    by (metis carrier-matD(1) carrier-matD(2) gauss-jordan-single(2) gauss-jordan-single(3)
index-transpose-mat(2) step1-mat-size)
  then have len-lt-eq: length (pivot-positions (gauss-jordan-single (?step1-matT)))
≤ dim-col ?step1-mat
    using step1-mat-size
    by simp
  have len-gt-false: length (rows-to-keep ?step1-mat) < (dim-col ?step1-mat) ⇒
False
  proof -
    assume length-lt: length (rows-to-keep ?step1-mat) < (dim-col ?step1-mat)
    have h: dim-row ?new-mat < (dim-col ?step1-mat)
      by (metis Matrix.transpose-transpose index-transpose-mat(3) length-lt length-map
mat-of-cols-carrier(3) take-cols-from-matrix-def take-indices-def take-rows-and-take-cols)
    then show False using rank-new-mat
      by (metis Matrix.transpose-transpose carrier-matI index-transpose-mat(2)
nat-less-le not-less-iff-gr-or-eq transpose-rank vec-space.rank-le-nc)
    qed
  then have len-gt-eq: length (rows-to-keep ?step1-mat) ≥ (dim-col ?step1-mat)
    using not-less by blast
  have len-match: length (rows-to-keep ?step1-mat) = (dim-col ?step1-mat)
    using len-lt-eq len-gt-eq
    by (simp add: rows-to-keep-def)
  have mat-match: matrix-A (get-signs (reduce-system p (qs, ((matrix-A signs sub-
sets), (subsets, signs))))))
= (get-subsets (reduce-system p (qs, ((matrix-A signs subsets), (subsets, signs))))))
= (get-matrix (reduce-system p (qs, ((matrix-A signs subsets), (subsets, signs))))))
    using reduce-system-matrix-match[of p qs signs subsets] assms
    by blast
  have f2: length (get-subsets (take-rows-from-matrix (mat-of-cols (dim-row (matrix-A
signs subsets)) (map (!! (cols (matrix-A signs subsets))) (find-nonzeros-from-input-vec
(solve-for-lhs p qs subsets (matrix-A signs subsets)))))) (rows-to-keep (mat-of-cols
(dim-row (matrix-A signs subsets)) (map (!! (cols (matrix-A signs subsets))) (find-nonzeros-from-input-vec
(solve-for-lhs p qs subsets (matrix-A signs subsets)))))), map (!! subsets) (rows-to-keep

```

```

(mat-of-cols (dim-row (matrix-A signs subsets)) (map (!) (cols (matrix-A signs
subsets)))) (find-nonzeros-from-input-vec (solve-for-lhs p qs subsets (matrix-A signs
subsets))))), map (!) signs) (find-nonzeros-from-input-vec (solve-for-lhs p qs sub-
sets (matrix-A signs subsets)))) = length (find-nonzeros-from-input-vec (solve-for-lhs
p qs subsets (matrix-A signs subsets)))
  by (metis (no-types) ‹dim-col (mat-of-cols (dim-row (matrix-A signs sub-
sets)) (take-indices (cols (matrix-A signs subsets)) (find-nonzeros-from-input-vec
(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)))› ‹length (rows-to-keep (reduce-mat-cols
(matrix-A signs subsets) (solve-for-lhs p qs subsets (matrix-A signs subsets))))
= dim-col (reduce-mat-cols (matrix-A signs subsets) (solve-for-lhs p qs subsets
(matrix-A signs subsets)))› length-map reduce-mat-cols.simps reduce-system.simps
reduction-step.simps reduction-subsets-def reduction-subsets-is-get-subsets take-cols-from-matrix-def
take-indices-def)
  have f3: ∀ p ps rss nss m. map (!) rss) (find-nonzeros-from-input-vec (solve-for-lhs
p ps nss m)) = get-signs (reduce-system p (ps, m, nss, rss))
  by (metis (no-types) reduction-signs-def reduction-signs-is-get-signs take-indices-def)
  have square-final-mat: square-mat (get-matrix (reduce-system p (qs, ((matrix-A
signs subsets), (subsets, signs))))))
  using mat-match assms size-of-mat same-size
  apply (auto) using f2 f3
  by (metis (no-types, lifting) Matrix.transpose-transpose fst-conv get-matrix-def
index-transpose-mat(2) len-match length-map mat-of-cols-carrier(2) mat-of-cols-carrier(3)
reduce-mat-cols.simps take-cols-from-matrix-def take-indices-def take-rows-and-take-cols)

  have rank-match: vec-space.rank (dim-row ?new-mat) ?new-mat = dim-row ?new-mat
  using len-match rank-new-mat
  by (simp add: take-cols-from-matrix-def take-indices-def take-rows-and-take-cols)

  have invertible-mat ?new-mat
  using invertible-det og-mat-size
  using vec-space.det-rank-iff square-final-mat
  by (metis dim-col-matrix-A dim-row-matrix-A fst-conv get-matrix-def mat-match
rank-match reduce-system.simps reduction-step.simps size-of-mat square-mat.elims(2))
  then show ?thesis apply (simp)
  by (metis fst-conv get-matrix-def)
qed

```

14.5 Well def signs preserved when reducing

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

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

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

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

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

assumes $\text{length-init} : \forall x \in \text{set}(signs). \text{length } x = \text{length } qs$

assumes $\text{sat-eq} : \text{satisfy-equation } p \text{ } qs \text{ } subsets \text{ } signs$

assumes $\text{inv-mat} : \text{invertible-mat } (matrix-A \text{ } signs \text{ } subsets)$

shows $\forall x \in \text{set}(\text{reduction-signs } p \text{ } qs \text{ } signs \text{ } subsets \text{ } (matrix-A \text{ } signs \text{ } subsets)).$

```

    length x = length qs
proof clarsimp
  fix x
  assume x-in-set:  $x \in \text{set } (\text{reduction-signs } p \text{ } qs \text{ } \text{signs } \text{subsets } (\text{matrix-A } \text{signs } \text{subsets}))$ 
  have List.member ( $\text{reduction-signs } p \text{ } qs \text{ } \text{signs } \text{subsets } (\text{matrix-A } \text{signs } \text{subsets})$ ) x
  using x-in-set by simp
  then have take-ind:  $\text{List.member } (\text{take-indices } \text{signs } (\text{find-nonzeros-from-input-vec } (\text{solve-for-lhs } p \text{ } qs \text{ } \text{subsets } (\text{matrix-A } \text{signs } \text{subsets}))))$  x
  unfolding reduction-signs-def by auto
  have find-nz-len:  $\forall y. \text{List.member } (\text{find-nonzeros-from-input-vec } (\text{solve-for-lhs } p \text{ } qs \text{ } \text{subsets } (\text{matrix-A } \text{signs } \text{subsets}))) y \longrightarrow y < \text{length } \text{signs}$ 
  using sat-eq inv-mat construct-lhs-matches-solve-for-lhs [of p qs subsets signs]
  by (auto simp add: find-nonzeros-from-input-vec-def) (metis size-of-lhs)
  then have  $\exists n < \text{length } \text{signs}. x = \text{signs } ! n$ 
  using take-ind
  by (metis in-set-conv-nth reduction-signs-def take-indices-lem x-in-set)
  then show  $\text{length } x = \text{length } qs$  using length-init take-indices-lem
  using nth-mem by blast
qed

```

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 –
  have solve-construct:  $\text{construct-lhs-vector } p \text{ } qs \text{ } \text{signs} = \text{solve-for-lhs } p \text{ } qs \text{ } \text{subsets } (\text{matrix-A } \text{signs } \text{subsets})$ 
  using construct-lhs-matches-solve-for-lhs assms
  by simp
  have h1:  $\text{distinct } (\text{find-nonzeros-from-input-vec } (\text{solve-for-lhs } p \text{ } qs \text{ } \text{subsets } (\text{matrix-A } \text{signs } \text{subsets})))$ 
  unfolding find-nonzeros-from-input-vec-def
  using distinct-filter
  using distinct-upt by blast
  have h2:  $(\bigwedge j. j \in \text{set } (\text{find-nonzeros-from-input-vec } (\text{solve-for-lhs } p \text{ } qs \text{ } \text{subsets } (\text{matrix-A } \text{signs } \text{subsets})))) \implies j < \text{length } \text{signs}$ 
proof –
  fix j
  assume  $j \in \text{set } (\text{find-nonzeros-from-input-vec } (\text{solve-for-lhs } p \text{ } qs \text{ } \text{subsets } (\text{matrix-A } \text{signs } \text{subsets})))$ 

```

```

signs subsets)))
  show j < length signs using solve-construct size-of-lhs
  by (metis ⟨j ∈ set (find-nonzeros-from-input-vec (solve-for-lhs p qs subsets
(matrix-A signs subsets)))⟩ atLeastLessThan-iff filter-is-subset find-nonzeros-from-input-vec-def
set-upt subset-iff)
  qed
  then show ?thesis unfolding reduction-signs-def unfolding take-indices-def
  using distinct-init h1 h2 conjugatable-vec-space.distinct-map-nth[where ls =
signs, where inds = (find-nonzeros-from-input-vec (solve-for-lhs p qs subsets (matrix-A
signs subsets)))]
  by blast
  qed

```

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 ≠ 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 sub-
sets)) (length qs)
  unfolding all-list-constr-def
proof clarsimp
  fix x
  assume in-red-subsets: x ∈ set (reduction-subsets p qs signs subsets (matrix-A
signs subsets))
  have solve-construct: construct-lhs-vector p qs signs =
  solve-for-lhs p qs subsets (matrix-A signs subsets)
  using construct-lhs-matches-solve-for-lhs assms
  by simp
  have rows-to-keep-hyp: ∀ y. y ∈ set (rows-to-keep (reduce-mat-cols (matrix-A
signs subsets) (solve-for-lhs p qs subsets (matrix-A signs subsets)))) →
  y < length subsets
  proof clarsimp
  fix y
  assume in-set: y ∈ set (rows-to-keep
  (take-cols-from-matrix (matrix-A signs subsets) (find-nonzeros-from-input-vec
(solve-for-lhs p qs subsets (matrix-A signs subsets))))))
  have in-set-2: y ∈ set (rows-to-keep
  (take-cols-from-matrix (matrix-A signs subsets) (find-nonzeros-from-input-vec
(construct-lhs-vector p qs signs))))
  using in-set solve-construct by simp
  let ?lhs-vec = (solve-for-lhs p qs subsets (matrix-A signs subsets))
  have h30: (construct-lhs-vector p qs signs) = ?lhs-vec

```

```

using assms construct-lhs-matches-solve-for-lhs
by simp
have h3a:  $\forall x. \text{List.member (find-nonzeros-from-input-vec ?lhs-vec) } x \longrightarrow x < \text{length (signs)}$ 
using h30 size-of-lhs unfolding find-nonzeros-from-input-vec-def apply (auto)
by metis
have h3c:  $\forall x. \text{List.member (rows-to-keep (reduce-mat-cols (matrix-A signs subsets) (solve-for-lhs p qs subsets (matrix-A signs subsets)))) } x \longrightarrow x < \text{length (subsets)}$ 
proof clarsimp
fix x
assume x-mem:  $x \in \text{set (rows-to-keep (take-cols-from-matrix (matrix-A signs subsets) (find-nonzeros-from-input-vec (solve-for-lhs p qs subsets (matrix-A signs subsets))))})}$ 
obtain nn :: rat list list  $\Rightarrow$  nat list  $\Rightarrow$  nat where
 $\forall x2\ x3. (\exists v4. v4 \in \text{set } x3 \wedge \neg v4 < \text{length } x2) = (nn\ x2\ x3 \in \text{set } x3 \wedge \neg nn\ x2\ x3 < \text{length } x2)$ 
by moura
then have f4:  $nn\ \text{signs (find-nonzeros-from-input-vec (solve-for-lhs p qs subsets (matrix-A signs subsets)))} \in \text{set (find-nonzeros-from-input-vec (solve-for-lhs p qs subsets (matrix-A signs subsets)))} \wedge \neg nn\ \text{signs (find-nonzeros-from-input-vec (solve-for-lhs p qs subsets (matrix-A signs subsets)))} < \text{length signs} \vee \text{matrix-A (take-indices signs (find-nonzeros-from-input-vec (solve-for-lhs p qs subsets (matrix-A signs subsets)))) subsets} = \text{take-cols-from-matrix (matrix-A signs subsets) (find-nonzeros-from-input-vec (solve-for-lhs p qs subsets (matrix-A signs subsets)))}$ 
using h3a nonzero reduce-system-matrix-signs-helper by auto
then have matrix-A (take-indices signs (find-nonzeros-from-input-vec (solve-for-lhs p qs subsets (matrix-A signs subsets)))) subsets = take-cols-from-matrix (matrix-A signs subsets) (find-nonzeros-from-input-vec (solve-for-lhs p qs subsets (matrix-A signs subsets)))  $\wedge x \in \text{set (map snd (pivot-positions (gauss-jordan-single (take-cols-from-matrix (matrix-A signs subsets) (find-nonzeros-from-input-vec (solve-for-lhs p qs subsets (matrix-A signs subsets))))^T))}$ 
using h3a x-mem by (auto simp add: rows-to-keep-def)
thus  $x < \text{length (subsets)}$  using x-mem unfolding rows-to-keep-def
using dim-row-matrix-A h3a nonzero reduce-system-matrix-signs-helper rows-to-keep-def rows-to-keep-lem
by metis
qed
then show  $y < \text{length subsets}$  apply (auto)
using in-set-2 solve-construct by fastforce
qed
show list-constr x (length qs) using in-red-subsets unfolding reduction-subsets-def

using take-indices-lem [of - subsets] rows-to-keep-hyp apply auto
apply (metis List.member-iff all-list-constr-def in-set-conv-nth length-init)
done
qed

```

15 Overall Lemmas

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

lemma *getter-functions*: $\text{calculate-data } p qs = (\text{get-matrix } (\text{calculate-data } p qs),$
 $(\text{get-subsets } (\text{calculate-data } p qs), \text{get-signs } (\text{calculate-data } p qs)))$
unfolding *get-matrix-def get-subsets-def get-signs-def* **by** *auto*

15.1 Key properties preserved

15.1.1 Properties preserved when combining and reducing systems

lemma *combining-sys-satisfies-properties-helper*:
fixes *p*: *real poly*
fixes *qs1* :: *real poly list*
fixes *qs2* :: *real poly list*
fixes *subsets1 subsets2* :: *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 *pairwise-rel-prime1*: $\forall q. ((\text{List.member } qs1 q) \longrightarrow (\text{coprime } p q))$
assumes *nontriv2*: $\text{length } qs2 > 0$
assumes *pairwise-rel-prime2*: $\forall q. ((\text{List.member } qs2 q) \longrightarrow (\text{coprime } p q))$
assumes *satisfies-properties-sys1*: *satisfies-properties* *p qs1 subsets1 signs1 matrix1*
assumes *satisfies-properties-sys2*: *satisfies-properties* *p qs2 subsets2 signs2 matrix2*
shows *satisfies-properties* *p (qs1@qs2) (get-subsets (snd ((combine-systems p (qs1,(matrix1, (subsets1, signs1))) (qs2,(matrix2, (subsets2, signs2)))))) (get-signs (snd ((combine-systems p (qs1,(matrix1, (subsets1, signs1))) (qs2,(matrix2, (subsets2, signs2)))))) (get-matrix (snd ((combine-systems p (qs1,(matrix1, (subsets1, signs1))) (qs2,(matrix2, (subsets2, signs2))))))*
proof –
let *?subsets* = $(\text{get-subsets } (\text{snd } (\text{combine-systems } p (qs1, \text{matrix1}, \text{subsets1}, \text{signs1}) (qs2, \text{matrix2}, \text{subsets2}, \text{signs2}))))$
let *?signs* = $(\text{get-signs } (\text{snd } (\text{combine-systems } p (qs1, \text{matrix1}, \text{subsets1}, \text{signs1}) (qs2, \text{matrix2}, \text{subsets2}, \text{signs2}))))$
let *?matrix* = $(\text{get-matrix } (\text{snd } (\text{combine-systems } p (qs1, \text{matrix1}, \text{subsets1}, \text{signs1}) (qs2, \text{matrix2}, \text{subsets2}, \text{signs2}))))$
have *h1*: *all-list-constr* *?subsets* $(\text{length } (qs1 @ qs2))$
using *well-def-step*[*of subsets1 qs1 subsets2 qs2*] *assms*
by (*simp add: nontriv2 get-subsets-def satisfies-properties-def smash-systems-def*)

```

have h2: well-def-signs (length (qs1 @ qs2)) ?signs
  using well-def-signs-step[of qs1 qs2 signs1 signs2]
  using get-signs-def nontriv1 nontriv2 satisfies-properties-def satisfies-properties-sys1
satisfies-properties-sys2 smash-systems-def by auto
have h3: distinct ?signs
  using distinct-step[of - signs1 - signs2] assms
  using combine-systems.simps get-signs-def satisfies-properties-def smash-systems-def
snd-conv by auto
have h4: satisfy-equation p (qs1 @ qs2) ?subsets ?signs
  using assms inductive-step[of p qs1 qs2 signs1 signs2 subsets1 subsets2]
  using get-signs-def get-subsets-def satisfies-properties-def smash-systems-def
by auto
have h5: invertible-mat ?matrix
  using assms inductive-step[of p qs1 qs2 signs1 signs2 subsets1 subsets2]
  by (metis combining-to-smash fst-conv get-matrix-def kronecker-invertible sat-
isfies-properties-def smash-systems-def snd-conv)
have h6: ?matrix = matrix-A ?signs ?subsets
  unfolding get-matrix-def combine-systems.simps smash-systems-def get-signs-def
get-subsets-def
  apply simp
  apply (subst matrix-construction-is-kronecker-product [of subsets1 - signs1
signs2 subsets2])
  apply (metis Ball-set List.member-iff all-list-constr-def list-constr-def satis-
fies-properties-def
    satisfies-properties-sys1)
  using satisfies-properties-def satisfies-properties-sys1 well-def-signs-def apply
blast
  using satisfies-properties-def satisfies-properties-sys1 satisfies-properties-sys2
apply presburger
done
have h7: set (characterize-consistent-signs-at-roots-copr p (qs1 @ qs2))
  ⊆ set (?signs)
  using subset-step[of p qs1 signs1 qs2 signs2] assms
  by (simp add: nonzero get-signs-def satisfies-properties-def smash-systems-def)

then show ?thesis unfolding satisfies-properties-def using h1 h2 h3 h4 h5 h6
h7 by blast
qed

```

lemma combining-sys-satisfies-properties:

```

fixes p:: real poly
fixes qs1 :: real poly list
fixes qs2 :: real poly 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 satisfies-properties-sys1: satisfies-properties p qs1 (get-subsets (calculate-data

```

$p \text{ } qs1$) (get-signs (calculate-data $p \text{ } qs1$)) (get-matrix (calculate-data $p \text{ } qs1$))
assumes satisfies-properties-sys2: satisfies-properties $p \text{ } qs2$ (get-subsets (calculate-data
 $p \text{ } qs2$)) (get-signs (calculate-data $p \text{ } qs2$)) (get-matrix (calculate-data $p \text{ } qs2$))
shows satisfies-properties $p \text{ } (qs1 @ qs2)$ (get-subsets (snd ((combine-systems p
 $(qs1, calculate-data \text{ } p \text{ } qs1) (qs2, calculate-data \text{ } p \text{ } qs2))))$))
(get-signs (snd ((combine-systems $p \text{ } (qs1, calculate-data \text{ } p \text{ } qs1) (qs2, calculate-data$
 $p \text{ } qs2))))$))
(get-matrix (snd ((combine-systems $p \text{ } (qs1, calculate-data \text{ } p \text{ } qs1) (qs2, calculate-data$
 $p \text{ } qs2))))$))
using combining-sys-satisfies-properties-helper
by (metis getter-functions nontriv1 nontriv2 nonzero pairwise-rel-prime1 pair-
wise-rel-prime2 nonzero satisfies-properties-sys1 satisfies-properties-sys2)

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: $p \neq 0$
assumes nontriv: length $qs > 0$
assumes pairwise-rel-prime: $\forall q. ((List.member \text{ } qs \text{ } q) \longrightarrow (coprime \text{ } p \text{ } q))$
assumes satisfies-properties-sys: satisfies-properties $p \text{ } qs$ subsets signs matrix
shows satisfies-properties $p \text{ } qs$ (get-subsets (reduce-system $p \text{ } (qs, matrix, subsets, signs)$))
(get-signs (reduce-system $p \text{ } (qs, matrix, subsets, signs)$))
(get-matrix (reduce-system $p \text{ } (qs, matrix, subsets, signs)$))
proof –
have $h1$: all-list-constr (get-subsets (reduce-system $p \text{ } (qs, matrix, subsets, signs)$))
(length qs)
using reduction-doesnt-break-subsets assms reduction-subsets-is-get-subsets sat-
isfies-properties-def satisfies-properties-sys **by** auto
have $h2$: well-def-signs (length qs) (get-signs (reduce-system $p \text{ } (qs, matrix, sub-$
 $sets, signs)$))
using reduction-doesnt-break-length-signs[of signs $qs \text{ } p \text{ } subsets$] assms reduc-
tion-signs-is-get-signs satisfies-properties-def well-def-signs-def **by** auto
have $h3$: distinct (get-signs (reduce-system $p \text{ } (qs, matrix, subsets, signs)$))
using reduction-signs-are-distinct[of $p \text{ } qs \text{ } subsets \text{ } signs$] assms reduction-signs-is-get-signs
satisfies-properties-def **by** auto
have $h4$: satisfy-equation $p \text{ } qs$ (get-subsets (reduce-system $p \text{ } (qs, matrix, subsets,$
 $signs)$))
(get-signs (reduce-system $p \text{ } (qs, matrix, subsets, signs)$))
using reduce-system-matrix-equation-preserved[of $p \text{ } qs \text{ } signs \text{ } subsets$] assms sat-
isfies-properties-def **by** auto
have $h5$: invertible-mat (get-matrix (reduce-system $p \text{ } (qs, matrix, subsets, signs)$))
using reduction-doesnt-break-things-invertibility assms same-size satisfies-properties-def
by auto
have $h6$: get-matrix (reduce-system $p \text{ } (qs, matrix, subsets, signs)$) =
matrix-A (get-signs (reduce-system $p \text{ } (qs, matrix, subsets, signs)$))
(get-subsets (reduce-system $p \text{ } (qs, matrix, subsets, signs)$))

```

using reduce-system-matrix-match[of p qs signs subsets] assms satisfies-properties-def
by auto
have h7: set (characterize-consistent-signs-at-roots-copr p qs)  $\subseteq$  set (get-signs
(reduce-system p (qs, matrix, subsets, signs)))
using reduction-doesnt-break-things-signs[of p qs signs subsets] assms reduction-signs-is-get-signs satisfies-properties-def by auto
then show ?thesis unfolding satisfies-properties-def using h1 h2 h3 h4 h5 h6
h7
by blast
qed

```

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  $\neq$  0
assumes len1: length qs = 1
assumes pairwise-rel-prime:  $\forall q. ((List.member\ qs\ q) \longrightarrow (coprime\ p\ q))$ 
shows satisfies-properties p qs (get-subsets (calculate-data p qs)) (get-signs (calculate-data
p qs)) (get-matrix (calculate-data p qs))
proof -
have h1: all-list-constr [[],[0]] (length qs)
using len1 unfolding all-list-constr-def list-constr-def by auto
have h2: well-def-signs (length qs) [[1],[-1]]
unfolding well-def-signs-def using len1
by auto
have h3: distinct ([[1],[-1]]::rat list list)
unfolding distinct-def by auto
have h4: satisfy-equation p qs [[],[0]] [[1],[-1]]
using assms base-case-satisfy-equation-alt[of qs p] by auto
have h6: (mat-of-rows-list 2 [[1,1], [1,-1]]::rat mat) = (matrix-A ([[1],[-1]])
([[],[0]]) :: rat mat)
using mat-base-case by auto
then have h5: invertible-mat (mat-of-rows-list 2 [[1,1], [1,-1]]:: rat mat)
using base-case-invertible-mat
by simp
have h7: set (characterize-consistent-signs-at-roots-copr p qs)  $\subseteq$  set ([[1],[-1]])
using assms base-case-sgas-alt[of qs p]
by simp
have base-case-hyp: satisfies-properties p qs [[],[0]] [[1],[-1]] (mat-of-rows-list 2
[[1,1], [1,-1]])
using h1 h2 h3 h4 h5 h6 h7
using satisfies-properties-def by blast
then have key-hyp: satisfies-properties p qs (get-subsets (reduce-system p (qs,base-case-info)))
(get-signs (reduce-system p (qs,base-case-info))) (get-matrix (reduce-system p (qs,base-case-info)))
using reducing-sys-satisfies-properties

```

```

    by (metis base-case-info-def len1 nonzero pairwise-rel-prime nonzero zero-less-one-class.zero-less-one)

  show ?thesis
    by (simp add: key-hyp len1)
qed

```

15.1.3 For arbitrary qs

lemma *append-not-distinct-helper*: $(List.member\ l1\ m \wedge List.member\ l2\ m) \longrightarrow (distinct\ (l1@l2) = False)$

```

proof –
  have h1: List.member l1 m  $\longrightarrow$   $(\exists n. n < length\ l1 \wedge (nth\ l1\ n) = m)$ 
    using nth-find-first
    by (simp add: in-set-conv-nth)
  have h2:  $\forall n. n < length\ l1 \wedge (nth\ l1\ n) = m \longrightarrow (nth\ (l1@l2)\ n) = m$ 
proof clarsimp
  fix n
  assume lt: n < length l1
  assume nth-l1: m = l1 ! n
  show (l1 @ l2) ! n = l1 ! n
proof (induct l2)
  case Nil
  then show ?case
    by simp
  next
  case (Cons a l2)
  then show ?case
    by (simp add: lt nth-append)
  qed
qed
  have h3: List.member l1 m  $\longrightarrow$   $(\exists n. n < length\ l1 \wedge (nth\ (l1@l2)\ n) = m)$ 
    using h1 h2 by auto
  have h4: List.member l2 m  $\longrightarrow$   $(\exists n. (nth\ l2\ n) = m)$ 
    by (auto dest: nth-find-first)
  have h5:  $\forall n. (nth\ l2\ n) = m \longrightarrow (nth\ (l1@l2)\ (n + length\ l1)) = m$ 
proof clarsimp
  fix n
  assume nth-l2: m = l2 ! n
  show (l1 @ l2) ! (n + length l1) = l2 ! n
proof (induct l2)
  case Nil
  then show ?case
    by (metis add.commute nth-append-length-plus)
  next
  case (Cons a l2)
  then show ?case
    by (simp add: nth-append)
  qed
qed

```

```

have h6: List.member l2 m  $\longrightarrow$  ( $\exists n. (nth (l1@l2) (n + length l1)) = m$ )
  using h4 h5
  by blast
show ?thesis using h3 h6
  by auto
qed

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 ( $p \neq 0 \wedge (length\ qs > 0) \wedge (\forall q. ((List.member\ qs\ q) \longrightarrow (coprime\ p\ q)))$ )
)
   $\longrightarrow$  satisfies-properties p qs (get-subsets (calculate-data p qs)) (get-signs (calculate-data
p qs)) (get-matrix (calculate-data p qs))
proof (induction length qs arbitrary: qs rule: less-induct)
  case less
  have len1-h: length qs = 1  $\longrightarrow$  ( $p \neq 0 \wedge (length\ qs > 0) \wedge (\forall q. ((List.member\ qs\ q) \longrightarrow (coprime\ p\ q)))$ )
 $\longrightarrow$  satisfies-properties p qs (get-subsets (calculate-data
p qs)) (get-signs (calculate-data p qs)) (get-matrix (calculate-data p qs))
  using length-1-calculate-data-satisfies-properties
  by blast
  let ?len = length qs
  let ?q1 = take (?len div 2) qs
  let ?left = calculate-data p ?q1
  let ?q2 = drop (?len div 2) qs
  let ?right = calculate-data p ?q2
  let ?comb = combine-systems p (?q1, ?left) (?q2, ?right)
  let ?red = reduce-system p ?comb
  have h-q1-len: length qs > 1  $\longrightarrow$  (length ?q1 > 0) by auto
  have h-q2-len: length qs > 1  $\longrightarrow$  (length ?q2 > 0) by auto
  have h-q1-copr: ( $\forall q. ((List.member\ qs\ q) \longrightarrow (coprime\ p\ q))$ )  $\longrightarrow$  ( $\forall q. ((List.member\ ?q1\ q) \longrightarrow (coprime\ p\ q))$ )
  proof -
    have mem-hyp: ( $\forall q. ((List.member\ ?q1\ q) \longrightarrow (List.member\ qs\ q))$ )
      by (auto dest: in-set-takeD)
    then show ?thesis
      by blast
  qed
  have h-q2-copr: ( $\forall q. ((List.member\ qs\ q) \longrightarrow (coprime\ p\ q))$ )  $\longrightarrow$  ( $\forall q. ((List.member\ ?q2\ q) \longrightarrow (coprime\ p\ q))$ )
  proof -
    have mem-hyp: ( $\forall q. ((List.member\ ?q2\ q) \longrightarrow (List.member\ qs\ q))$ )
      by (auto dest: in-set-dropD)
    then show ?thesis
      by blast
  qed
have q1-sat-props: length qs > 1  $\longrightarrow$  ( $p \neq 0 \wedge (length\ qs > 0) \wedge (\forall q. ((List.member$ 

```

```

qs q) → (coprime p q))) ) → satisfies-properties p ?q1 (get-subsets (calculate-data
p ?q1)) (get-signs (calculate-data p ?q1)) (get-matrix (calculate-data p ?q1))
  using less.hyps[of ?q1] h-q1-len h-q1-copr
  by (auto simp add: Let-def)
  have q2-help: length qs > 1 → length (drop (length qs div 2) qs) < length qs
  using h-q1-len by auto
  then have q2-sat-props: length qs > 1 → (p ≠ 0 ∧ (length qs > 0) ∧ (∀ q.
((List.member qs q) → (coprime p q))) ) → satisfies-properties p ?q2 (get-subsets
(calculate-data p ?q2)) (get-signs (calculate-data p ?q2)) (get-matrix (calculate-data
p ?q2))
  using less.hyps[of ?q2] h-q2-len h-q2-copr
  by blast
  have put-tog: ?q1@?q2 = qs
  using append-take-drop-id by blast
  then have comb-sat-props: length qs > 1 → (p ≠ 0 ∧ (length qs > 0) ∧
(∀ q. ((List.member qs q) → (coprime p q))) ) → (satisfies-properties p (qs)
(get-subsets (snd ((combine-systems p (?q1, calculate-data p ?q1) (?q2, calculate-data
p ?q2)))))
  (get-signs (snd ((combine-systems p (?q1, calculate-data p ?q1) (?q2, calculate-data
p ?q2)))))
  (get-matrix (snd ((combine-systems p (?q1, calculate-data p ?q1) (?q2, calculate-data
p ?q2)))))
  using q1-sat-props q2-sat-props combining-sys-satisfies-properties
  using h-q1-copr h-q1-len h-q2-copr h-q2-len put-tog
  by metis
  then have comb-sat: length qs > 1 → (p ≠ 0 ∧ (length qs > 0) ∧ (∀ q.
((List.member qs q) → (coprime p q))) ) →
  (satisfies-properties p (qs) (get-subsets (snd ?comb)) (get-signs (snd ?comb))
(get-matrix (snd ?comb)))
  by blast
  have red-char: ?red = (reduce-system p (qs, (get-matrix (snd ?comb)), (get-subsets
(snd ?comb)), (get-signs (snd ?comb))))
  using getter-functions
  by (smt (verit, best) Pair-inject combining-to-smash get-matrix-def get-signs-def
get-subsets-def prod.collapse put-tog smash-systems-def)
  then have length qs > 1 → (p ≠ 0 ∧ (length qs > 0) ∧ (∀ q. ((List.member qs
q) → (coprime p q))) ) → (satisfies-properties p qs (get-subsets ?red) (get-signs
?red) (get-matrix ?red))
  using reducing-sys-satisfies-properties comb-sat by presburger
  then have len-gt1: length qs > 1 → (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))
  by (smt (verit, best) calculate-data.simps leD nat-less-le)
  then show ?case using len1-h len-gt1
  by (metis One-nat-def Suc-lessI)
qed

```

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 -
  let ?signs = [[1],[−1]]::rat list list
  let ?subsets = [[],[0]]::nat list list
  have mat-help: matrix-A [[1],[−1]] [[],[0]] = (mat-of-rows-list 2 [[1,1], [1,−1]])
    using mat-base-case by auto
  have well-def-signs: well-def-signs (length qs) ?signs unfolding well-def-signs-def

    using len1 by auto
  have distinct-signs: distinct ?signs
    unfolding distinct-def by auto
  have ex-q: ∃ (q::real poly). qs = [q]
    using len1
    using length-Suc-conv[of qs 0] by auto
  then have all-info: set (characterize-consistent-signs-at-roots-copr p qs) ⊆ set(?signs)
    using assms base-case-sgas apply (auto)
    by (meson in-mono insertE insert-absorb insert-not-empty)
  have match: satisfy-equation p qs ?subsets ?signs
    using ex-q base-case-satisfy-equation nonzero pairwise-rel-prime
    by auto

  have invertible-mat: invertible-mat (matrix-A ?signs ?subsets)
    using inverse-mat-base-case inverse-mat-base-case-2 unfolding invertible-mat-def
using mat-base-case
  by auto
  have h: set (get-signs (reduce-system p (qs, ((matrix-A ?signs ?subsets), (?subsets,
?signs)))))) =
    set (characterize-consistent-signs-at-roots-copr p qs)
    using nonzero nonzero well-def-signs distinct-signs all-info match invertible-mat
    reduce-system-sign-conditions[where p = p, where qs = qs, where signs =
[[1],[−1], where subsets = [[],[0]]]
    by blast
  then have set (snd (snd (reduce-system p (qs, ((mat-of-rows-list 2 [[1,1],
[1,−1]], ([[],[0]], [[1],[−1]])))))) =
    set (characterize-consistent-signs-at-roots-copr p qs)
    unfolding get-signs-def using mat-help by auto
  then have set (snd (snd (reduce-system p (qs, base-case-info)))) = set (characterize-consistent-signs-at-roots-
p qs)
    unfolding base-case-info-def

```

```

    by auto
  then show ?thesis using len1
    by (simp add: find-consistent-signs-at-roots-thm)
qed

```

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 –
  let ?signs = (get-signs (snd ((combine-systems p (qs1, calculate-data p qs1)
(qs2, calculate-data p qs2))))))
  let ?subsets = (get-subsets (snd ((combine-systems p (qs1, calculate-data p qs1)
(qs2, calculate-data p qs2))))))
  have h0: satisfies-properties p (qs1@qs2) ?subsets ?signs
    (get-matrix (snd ((combine-systems p (qs1, calculate-data p qs1) (qs2, calculate-data
p qs2))))))
  using calculate-data-satisfies-properties combining-sys-satisfies-properties
  using nontriv1 nontriv2 nonzero pairwise-rel-prime1 pairwise-rel-prime2 nonzero
  by simp
  then have h1: set(characterize-consistent-signs-at-roots-copr p (qs1@qs2)) ⊆ set
?signs
  unfolding satisfies-properties-def
  by linarith
  have h2: well-def-signs (length (qs1@qs2)) ?signs
  using calculate-data-satisfies-properties
  using h0 satisfies-properties-def by blast
  have h3: distinct ?signs
  using calculate-data-satisfies-properties
  using h0 satisfies-properties-def by blast
  have h4: satisfy-equation p (qs1@qs2) ?subsets ?signs
  using calculate-data-satisfies-properties
  using h0 satisfies-properties-def by blast
  have h5: invertible-mat (matrix-A ?signs ?subsets)

```

```

using calculate-data-satisfies-properties
using h0 satisfies-properties-def
by auto
have h: set (take-indices ?signs
  (find-nonzeros-from-input-vec (solve-for-lhs p (qs1@qs2) ?subsets (matrix-A
    ?signs ?subsets))))
  = set(characterize-consistent-signs-at-roots-copr p (qs1@qs2))
using h1 h2 h3 h4 h5 reduction-deletes-bad-sign-conds
using nonzero nonzero reduction-signs-def by auto
then have h: set (characterize-consistent-signs-at-roots-copr p (qs1@qs2)) =
  set (reduction-signs p (qs1@qs2) ?signs ?subsets (matrix-A ?signs ?subsets ))
unfolding reduction-signs-def get-signs-def
by blast
have help-h: reduction-signs p (qs1@qs2) ?signs ?subsets (matrix-A ?signs
  ?subsets)
  = (take-indices ?signs (find-nonzeros-from-input-vec (solve-for-lhs p (qs1@qs2)
    ?subsets (matrix-A ?signs ?subsets))))
unfolding reduction-signs-def by auto
have clear-signs: (signs-smash (get-signs (calculate-data p qs1)) (get-signs (calculate-data
  p qs2))) = (get-signs (snd ((combine-systems p (qs1, calculate-data p qs1) (qs2, calculate-data
  p qs2)))))
by (smt (verit) combining-to-smash get-signs-def getter-functions smash-systems-def
  snd-conv)
have clear-subsets: (subsets-smash (length qs1) (get-subsets(calculate-data p qs1))
  (get-subsets (calculate-data p qs2))) = (get-subsets (snd ((combine-systems p (qs1, calculate-data
  p qs1) (qs2, calculate-data p qs2)))))
by (smt (verit, best) Pair-inject combining-to-smash get-subsets-def prod.collapse
  smash-systems-def)
have well-def-signs (length qs1) (get-signs (calculate-data p qs1))
using calculate-data-satisfies-properties
using nontriv1 nonzero pairwise-rel-prime1 nonzero satisfies-properties-def
by auto
then have well-def-signs1: ( $\bigwedge j. j \in \text{set } (\text{get-signs } (\text{calculate-data } p \text{ } qs1)) \implies$ 
  length j = (length qs1))
using well-def-signs-def by blast
have all-list-constr (get-subsets(calculate-data p qs1)) (length qs1)
using calculate-data-satisfies-properties
using nontriv1 nonzero pairwise-rel-prime1 nonzero satisfies-properties-def
by auto
then have well-def-subsets1: ( $\bigwedge l i. l \in \text{set } (\text{get-subsets}(\text{calculate-data } p \text{ } qs1))$ 
 $\implies i \in \text{set } l \implies i < (\text{length } qs1)$ )
by (auto simp add: all-list-constr-def list-constr-def in-set-conv-nth list-all-length)

have extra-matrix-same: matrix-A (signs-smash (get-signs (calculate-data p qs1))
  (get-signs (calculate-data p qs2)))
  (subsets-smash (length qs1) (get-subsets(calculate-data p qs1)) (get-subsets
  (calculate-data p qs2)))
  = kronecker-product (get-matrix (calculate-data p qs1)) (get-matrix (calculate-data
  p qs2))

```

```

using well-def-signs1 well-def-subsets1
using matrix-construction-is-kronecker-product
using calculate-data-satisfies-properties nontriv1 nontriv2 nonzero pairwise-rel-prime1
pairwise-rel-prime2 nonzero satisfies-properties-def by auto
then have matrix-same: matrix-A ?signs ?subsets = kronecker-product (get-matrix
(calculate-data p qs1)) (get-matrix (calculate-data p qs2))
using clear-signs clear-subsets
by simp
have comb-sys-h: snd(snd(reduce-system p (combine-systems p (qs1, calculate-data
p qs1) (qs2, calculate-data p qs2)))) =
snd(snd(reduce-system p (qs1@qs2, (matrix-A ?signs ?subsets, (?subsets,
?signs))))))
unfolding get-signs-def get-subsets-def using matrix-same
by (smt combining-to-smash get-signs-def get-subsets-def getter-functions prod.collapse
prod.inject smash-systems-def)
then have extra-h: snd(snd(reduce-system p (qs1@qs2, (matrix-A ?signs ?subsets,
(?subsets, ?signs)))))) =
snd(snd(reduction-step (matrix-A ?signs ?subsets) ?signs ?subsets (solve-for-lhs
p (qs1@qs2) ?subsets (matrix-A ?signs ?subsets))))
by simp
then have same-h: set(snd(snd(reduce-system p (combine-systems p (qs1, calculate-data
p qs1) (qs2, calculate-data p qs2))))))
= set (reduction-signs p (qs1@qs2) ?signs ?subsets (matrix-A ?signs ?subsets
))
using comb-sys-h unfolding reduction-signs-def
by (metis get-signs-def help-h reduction-signs-is-get-signs)
then show ?thesis using h
by blast
qed

```

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 (induction length qs arbitrary: qs rule: less-induct)
case less
then show ?case
proof clarsimp
assume ind-hyp: (∧ qsa.
length qsa < length qs ⇒ qsa ≠ [] ∧ (∀ q. q ∈ set qsa → coprime p q) →
set (find-consistent-signs-at-roots p qsa) =
set (characterize-consistent-signs-at-roots-copr p qsa))
assume nonzero: p ≠ 0
assume nontriv: qs ≠ []
assume copr: ∀ q. q ∈ set qs → coprime p q
let ?len = length qs

```

```

let ?q1 = take ((?len) div 2) qs
let ?left = calculate-data p ?q1
let ?q2 = drop ((?len) div 2) qs
let ?right = calculate-data p ?q2
have nontriv-q1: length qs > 1  $\longrightarrow$  length ?q1 > 0
  by auto
have qs-more-q1: length qs > 1  $\longrightarrow$  length qs > length ?q1
  by auto
have pairwise-rel-prime-q1:  $\forall q. ((List.member ?q1 q) \longrightarrow (coprime p q))$ 
proof clarsimp
  fix q
  assume q  $\in$  set (take (length qs div 2) qs)
  then have List.member qs q using set-take-subset [where n = length qs div
2]
    by (auto dest: in-set-takeD)
  then show coprime p q
    using copr by auto
qed
have nontriv-q2: length qs > 1  $\longrightarrow$  length ?q2 > 0
  by auto
have qs-more-q2: length qs > 1  $\longrightarrow$  length qs > length ?q2
  by auto
have pairwise-rel-prime-q2:  $\forall q. ((List.member ?q2 q) \longrightarrow (coprime p q))$ 
proof clarsimp
  fix q
  assume q  $\in$  set (drop (length qs div 2) qs)
  then have List.member qs q using set-take-subset [where n = length qs div
2]
    by (auto dest: in-set-dropD)
  then show coprime p q
    using copr by auto
qed
have key-h: set (snd (snd (if ?len  $\leq$  Suc 0 then reduce-system p (qs, base-case-info)
  else Let (combine-systems p (?q1, ?left) (?q2, ?right))
    (reduce-system p)))) =
  set (characterize-consistent-signs-at-roots-copr p qs)
proof -
  have h-len1 : ?len = 1  $\longrightarrow$  set (snd (snd (if ?len  $\leq$  Suc 0 then reduce-system
p (qs, base-case-info)
  else Let (combine-systems p (?q1, ?left) (?q2, ?right))
    (reduce-system p)))) =
  set (characterize-consistent-signs-at-roots-copr p qs)
  using find-consistent-signs-at-roots-len1 [of p qs] copr nonzero nontriv
  by (simp add: find-consistent-signs-at-roots-thm)
  have h-len-gt1 : ?len > 1  $\longrightarrow$  set (snd (snd (if ?len  $\leq$  Suc 0 then reduce-system
p (qs, base-case-info)
  else Let (combine-systems p (?q1, ?left) (?q2, ?right))
    (reduce-system p)))) =
  set (characterize-consistent-signs-at-roots-copr p qs)

```

```

proof –
  have h-imp-a:  $?len > 1 \longrightarrow \text{set}(\text{snd}(\text{snd}(\text{reduce-system } p (\text{combine-systems } p (?q1, ?left) (?q2, ?right)))))) =$ 
     $\text{set}(\text{characterize-consistent-signs-at-roots-copr } p \text{ } qs)$ 
  proof –
    have h1:  $?len > 1 \longrightarrow \text{set}(\text{snd}(\text{snd} (?left))) = \text{set}(\text{characterize-consistent-signs-at-roots-copr } p ?q1)$ 
      using nontriv-q1 pairwise-rel-prime-q1 ind-hyp [of ?q1] qs-more-q1
      apply (auto simp add: find-consistent-signs-at-roots-thm)
      apply (smt (verit, del-insts) One-nat-def list.size(3) nontriv-q1 order-less-irrefl take0)
      apply (smt (verit, ccfv-threshold) length-greater-0-conv nontriv-q1 numeral-1-eq-Suc-0 numeral-eq-one-iff take0)
    done
    have h2:  $?len > 1 \longrightarrow \text{set}(\text{snd}(\text{snd} (?right))) = \text{set}(\text{characterize-consistent-signs-at-roots-copr } p ?q2)$ 
      using nontriv-q2 pairwise-rel-prime-q2 ind-hyp [of ?q2] qs-more-q2
      by (auto simp add: find-consistent-signs-at-roots-thm)
      show ?thesis using nonzero nontriv-q1 pairwise-rel-prime-q1 nontriv-q2 pairwise-rel-prime-q2 h1 h2 smaller-sys-are-good
      by (metis append-take-drop-id find-consistent-signs-at-roots-thm)
    qed
    then have h-imp:  $?len > 1 \longrightarrow \text{set}(\text{snd}(\text{snd}(\text{Let}(\text{combine-systems } p (?q1, ?left) (?q2, ?right))(\text{reduce-system } p)))) =$ 
       $\text{set}(\text{characterize-consistent-signs-at-roots-copr } p \text{ } qs)$ 
      by auto
      then show ?thesis by auto
    qed
    show ?thesis using h-len1 h-len-gt1
      by (meson <qs ≠ []> length-0-conv less-one nat-neq-iff)
    qed
    then show  $\text{set}(\text{find-consistent-signs-at-roots } p \text{ } qs) = \text{set}(\text{characterize-consistent-signs-at-roots-copr } p \text{ } qs)$ 
      by (smt One-nat-def calculate-data.simps find-consistent-signs-at-roots-thm length-0-conv nontriv)
    qed
  qed

```

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

fixes *p*: *real poly*

assumes $p \neq 0$

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

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

proof –

obtain *a b c* **where** *abc*: $\text{reduce-system } p ([1], \text{base-case-info}) = (a, b, c)$

using *prod-cases3 by blast*

```

have find-consistent-signs-at-roots  $p [1] = c$  using abc
  by (simp add: find-consistent-signs-at-roots-thm)
have *: set (find-consistent-signs-at-roots  $p [1]) = \text{set} (\text{characterize-consistent-signs-at-roots-copr}$ 
 $p [1])$ 
  apply (subst find-consistent-signs-at-roots-1)
  using assms by auto
have set(characterize-consistent-signs-at-roots-copr  $p [])$  = drop 1 ' set(characterize-consistent-signs-at-roots-
 $p [1])$ 
  unfolding characterize-consistent-signs-at-roots-copr-def consistent-sign-vec-copr-def
apply simp
  by (smt drop0 drop-Suc-Cons image-cong image-image)
  thus ?thesis using abc *
  apply (auto) apply (simp add: find-consistent-signs-at-roots-thm)
  by (simp add: find-consistent-signs-at-roots-thm)
qed

```

```

lemma find-consistent-signs-at-roots-copr:
  fixes  $p$ :: real poly
  fixes  $qs$ :: real poly list
  assumes  $p \neq 0$ 
  assumes  $\bigwedge q. q \in \text{set } qs \implies \text{coprime } p q$ 
  shows set(find-consistent-signs-at-roots  $p qs) = \text{set}(\text{characterize-consistent-signs-at-roots-copr}$ 
 $p qs)$ 
  using assms find-consistent-signs-at-roots-0 [of p] find-consistent-signs-at-roots-1
  [of p qs]
  by auto

```

```

lemma find-consistent-signs-at-roots:
  fixes  $p$ :: real poly
  fixes  $qs$ :: real poly list
  assumes  $p \neq 0$ 
  assumes  $\bigwedge q. q \in \text{set } qs \implies \text{coprime } p q$ 
  shows set(find-consistent-signs-at-roots  $p qs) = \text{set}(\text{characterize-consistent-signs-at-roots}$ 
 $p qs)$ 
  using assms csa-list-copr-rel [of p qs] find-consistent-signs-at-roots-copr [of p qs]
  by auto

```

```

theorem find-consistent-signs-at-roots-alt:
  assumes  $p \neq 0$ 
  assumes  $\bigwedge q. q \in \text{set } qs \implies \text{coprime } p q$ 
  shows set (find-consistent-signs-at-roots  $p qs) = \text{consistent-signs-at-roots } p qs$ 
  using consistent-signs-at-roots-eq assms(1) assms(2) find-consistent-signs-at-roots
  by auto

```

```

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* ($\lambda(f,i).$ (*lead-coeff* f) \wedge i) fs) ;
 fs = *map* ($\lambda(f,i).$ (*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* ($\lambda(c,fs).$ (c, *map* ($\lambda(f,pow).$ (*index-of* factors f, pow)) fs))
 fact-ps
 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* ($\lambda(f,pow).$ (signs ! f) \wedge 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* ($\lambda l.$ *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* (λ *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* (λ *i*.
map (λ *v*. *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* (λ *x*. *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 \neq *None*
)

17 Proofs

17.1 Parsing and Semantics

primrec *real-sem* :: *real poly fml* \Rightarrow *real* \Rightarrow *bool*

where
real-sem (And l r) x = (*real-sem l x* \wedge *real-sem r x*)
 | *real-sem (Or l r) x* = (*real-sem l x* \vee *real-sem r x*)
 | *real-sem (Gt p) x* = (*poly p x* $>$ 0)
 | *real-sem (Geq p) x* = (*poly p x* \geq 0)
 | *real-sem (Lt p) x* = (*poly p x* $<$ 0)
 | *real-sem (Leq p) x* = (*poly p x* \leq 0)
 | *real-sem (Eq p) x* = (*poly p x* = 0)
 | *real-sem (Neq p) x* = (*poly p x* \neq 0)

primrec *fml-sem* :: *rat poly fml* \Rightarrow *real* \Rightarrow *bool*

where
fml-sem (And l r) x = (*fml-sem l x* \wedge *fml-sem r x*)
 | *fml-sem (Or l r) x* = (*fml-sem l x* \vee *fml-sem r x*)
 | *fml-sem (Gt p) x* = (*rpoly p x* $>$ 0)
 | *fml-sem (Geq p) x* = (*rpoly p x* \geq 0)
 | *fml-sem (Lt p) x* = (*rpoly p x* $<$ 0)
 | *fml-sem (Leq p) x* = (*rpoly p x* \leq 0)
 | *fml-sem (Eq p) x* = (*rpoly p x* = 0)
 | *fml-sem (Neq p) x* = (*rpoly p x* \neq 0)

lemma *poly-list-set-fml*:

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

apply (*induction*) **by** *auto*

lemma *convert-semantic-lem*:

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

$ls ! (\text{index-of } ps \ p) = \text{rpoly } p \ x$

shows $fml\text{-sem } fml \ x = \text{lookup-sem } (\text{map-fml } (\text{index-of } ps) \ fml) \ ls$

using *assms* **apply** (*induct fml*)

by *auto*

lemma *index-of-aux-more*:

shows $\text{index-of-aux } ls \ p \ n \geq n$

apply (*induct ls arbitrary: n*)

apply *auto*

using *Suc-leD* **by** *blast*

lemma *index-of-aux-lookup*:

assumes $p \in \text{set } ls$

shows $(\text{index-of-aux } ls \ p \ n) - n < \text{length } ls$

$ls ! ((\text{index-of-aux } ls \ p \ n) - n) = p$

using *assms* **apply** (*induct ls arbitrary: n*)

apply *auto*

apply (*metis Suc-diff-Suc index-of-aux-more lessI less-Suc-eq-0-disj less-le-trans*)

by (*metis Suc-diff-Suc index-of-aux-more lessI less-le-trans nth-Cons-Suc*)

lemma *index-of-lookup*:

assumes $p \in \text{set } ls$

shows $\text{index-of } ls \ p < \text{length } ls$

$ls ! (\text{index-of } ls \ p) = p$

apply (*metis assms index-of-aux-lookup(1) index-of-def minus-nat.diff-0*)

by (*metis assms index-of-aux-lookup(2) index-of-def minus-nat.diff-0*)

lemma *convert-semantic*:

shows $fml\text{-sem } fml \ x = \text{lookup-sem } (\text{fst } (\text{convert } fml)) (\text{map } (\lambda p. \text{rpoly } p \ x) (\text{snd } (\text{convert } fml)))$

unfolding *convert-def Let-def* **apply** *simp*

apply (*intro convert-semantic-lem*)

by (*simp add: index-of-lookup(1) index-of-lookup(2)*)

lemma *convert-closed*:

shows $\bigwedge i. i \in \text{set-fml } (\text{fst } (\text{convert } fml)) \implies i < \text{length } (\text{snd } (\text{convert } fml))$

unfolding *convert-def Let-def*

apply (*auto simp add: fml.set-map*)

by (*simp add: index-of-lookup(1) poly-list-set-fml*)

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

where *sign-vec qs x \equiv*

map (squash \circ ($\lambda p. \text{rpoly } p \ x$)) qs

definition *consistent-sign-vectors::rat poly list \Rightarrow real set \Rightarrow rat list set*
where *consistent-sign-vectors qs S = (sign-vec qs) ‘ S*

lemma *sign-vec-semantics:*

assumes $\bigwedge i. i \in \text{set-fml fml} \implies i < \text{length ls}$
shows $\text{lookup-sem fml (map (\lambda p. rpoly p x) ls)} = \text{lookup-sem fml (sign-vec ls x)}$
using *assms apply (induction)*
by *(auto simp add: sign-vec-def squash-def)*

lemma *universal-lookup-sem:*

assumes $\bigwedge i. i \in \text{set-fml fml} \implies i < \text{length qs}$
assumes *set signs = consistent-sign-vectors qs UNIV*
shows $(\forall x::\text{real}. \text{lookup-sem fml (map (\lambda p. rpoly p x) qs)}) \longleftrightarrow$
 $\text{list-all (lookup-sem fml) signs}$
using *assms(2) unfolding consistent-sign-vectors-def list-all-iff*
by *(simp add: assms(1) sign-vec-semantics)*

lemma *existential-lookup-sem:*

assumes $\bigwedge i. i \in \text{set-fml fml} \implies i < \text{length qs}$
assumes *set signs = consistent-sign-vectors qs UNIV*
shows $(\exists x::\text{real}. \text{lookup-sem fml (map (\lambda p. rpoly p x) qs)}) \longleftrightarrow$
 $\text{find (lookup-sem fml) signs} \neq \text{None}$
using *assms(2) unfolding consistent-sign-vectors-def find-None-iff*
by *(simp add: assms(1) sign-vec-semantics)*

17.2 Factoring Lemmas

interpretation *of-rat-poly-hom: map-poly-comm-semiring-hom of-rat..*

interpretation *of-rat-poly-hom: map-poly-comm-ring-hom of-rat..*

interpretation *of-rat-poly-hom: map-poly-idom-hom of-rat..*

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

shows $\text{poly (real-of-rat-poly (\prod (a,b) \in s. f a b)) y} = (\prod (a,b) \in s. \text{poly (real-of-rat-poly (f a b)) y})$
apply *(simp add: of-rat-poly-hom.hom-prod poly-prod)*
by *(simp add: case-prod-app prod.case-distrib)*

lemma *sign-vec-index-of:*

assumes $f \in \text{set ftrs}$
shows $\text{sign-vec ftrs x} ! (\text{index-of ftrs f}) = \text{squash (rpoly f x)}$
by *(simp add: assms index-of-lookup(1) index-of-lookup(2) sign-vec-def)*

lemma *squash-idem:*

shows $\text{squash (squash x)} = \text{squash x}$
unfolding *squash-def* **by** *auto*

lemma *squash-mult:*

shows $\text{squash } ((a::\text{real}) * b) = \text{squash } a * \text{squash } b$
unfolding *squash-def* **apply** *auto*
using *less-not-sym mult-neg-neg* **apply** *blast*
using *mult-less-0-iff* **by** *blast*

lemma *squash-prod-list*:
shows $\text{squash } (\text{prod-list } (ls::\text{real list})) = \text{prod-list } (\text{map } \text{squash } ls)$
apply (*induction ls*)
unfolding *squash-def* **apply** *auto*
apply (*simp add: mult-less-0-iff*)
by (*simp add: zero-less-mult-iff*)

lemma *squash-pow*:
shows $\text{squash } ((x::\text{real}) ^ (y::\text{nat})) = (\text{squash } x) ^ y$
unfolding *squash-def* **apply** *auto*
by (*auto simp add: zero-less-power-eq*)

lemma *squash-real-of-rat*[*simp*]:
shows $\text{squash } (\text{real-of-rat } x) = \text{squash } x$
unfolding *squash-def* **by** *auto*

lemma *factorize-rat-poly-monic-irreducible-monic*:
assumes $\text{factorize-rat-poly-monic } f = (c, fs)$
assumes $(fi, i) \in \text{set } fs$
shows $\text{irreducible } fi \wedge \text{monic } fi$
proof –
obtain $c' fs'$ **where** $cfs: \text{factorize-rat-poly } f = (c', fs')$
by (*meson surj-pair*)
then have $fs: fs = \text{map } (\lambda(f, i). (\text{normalize } f, i)) fs'$
using *factorize-rat-poly-monic-def* **assms** **by** *auto*
obtain fi' **where** $(fi', i) \in \text{set } fs' \wedge fi = \text{normalize } fi'$
using *assms(2)* **unfolding** fs **by** *auto*
thus *?thesis* **using** *factorize-rat-poly irreducible-normalize-iff*
by (*metis cfs monic-normalize not-irreducible-zero*)
qed

lemma *square-free-normalize*:
assumes *square-free p*
shows *square-free (normalize p)*
by (*metis assms square-free-multD(3) unit-factor-mult-normalize*)

lemma *coprime-normalize*:
assumes *coprime a b*
shows *coprime (normalize a) b*
using *assms* **by** *auto*

lemma *undo-normalize*:
shows $a = \text{Polynomial.smult } (\text{unit-factor } (\text{lead-coeff } a)) (\text{normalize } a)$
by (*metis add.right-neutral mult-pCons-right mult-zero-right normalize-mult-unit-factor*)

pCons-0-hom.hom-zero unit-factor-poly-def)

lemma *finite-smult-distr*:

assumes *distinct fs*
shows $(\prod_{(x,y)\in set\ fs} Polynomial.smult\ ((f\ x\ y)::rat)\ (g\ x\ y)) =$
 $Polynomial.smult\ (\prod_{(x,y)\in set\ fs} f\ x\ y)\ (\prod_{(x,y)\in set\ fs} g\ x\ y)$
using *assms*
proof (*induction fs*)
case *Nil*
then show *?case by auto*
next
case (*Cons a fs*)
then show *?case apply auto*
using *mult.commute mult-smult-right prod.case-distrib smult-smult split-cong*
split-conv
by (*simp add: Groups.mult-ac(2) split-beta*)
qed

lemma *normalize-coprime-degree*:

assumes *normalize (f::rat poly) = normalize g*
assumes *coprime f g*
shows *degree f = 0*
proof –
have *f dvd g* **by** (*simp add: assms(1) associatedD2*)
then have *f dvd 1*
using *assms(2) associatedD1 by auto*
thus *?thesis*
using *Missing-Polynomial-Factorial.is-unit-field-poly by blast*
qed

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 (*unfold square-free-factorization-def split, intro conjI impI allI*)
obtain *c' fs'* **where** *cfs: factorize-rat-poly f = (c',fs')*
by (*meson surj-pair*)
then have *fs: fs = map (\(f,i). (normalize f, i)) fs'*
using *factorize-rat-poly-monic-def assms by auto*
have *sq: square-free-factorization f (c',fs')*
using *cfs factorize-rat-poly(1) by blast*
obtain *lcs* **where** *lcs: lcs = prod-list (map (\(f,i). lead-coeff f ^ i) fs')* **by force**
have *c: c = c' * lcs* **using** *assms unfolding factorize-rat-poly-monic-def cfs*
Let-def lcs by auto
show *f = 0 \implies c = 0* **using** *c cfs by auto*
show *f = 0 \implies fs = []* **using** *fs cfs by auto*
have *dist: distinct fs'* **using** *sq square-free-factorizationD(5) by blast*
show *dist2: distinct fs* **unfolding** *fs*
unfolding *distinct-conv-nth* **apply** *auto*
proof –

```

fix  $i j$ 
assume  $ij$ :  $i < \text{length } fs' \ j < \text{length } fs' \ i \neq j$ 
assume  $eq$ : (case  $fs' ! i$  of
  ( $f, x$ )  $\Rightarrow$  (normalize  $f, x$ ) =
  (case  $fs' ! j$  of
    ( $f, x$ )  $\Rightarrow$  (normalize  $f, x$ ))
obtain  $f a$  where  $fa$ :  $fs' ! i = (f, a)$  by force
obtain  $g$  where  $g$ :  $fs' ! j = (g, a)$  normalize f = normalize g
  using  $eq fa$  apply auto
  by (metis case-prod-conv prod.collapse prod.inject)
have  $f \neq g$  using dist ij fa g
  using nth-eq-iff-index-eq by fastforce
then have coprime f g
  using square-free-factorizationD(3)[OF sq, of f a g a]  $fa g ij$ 
  apply auto
  using nth-mem by force
then have degree f = 0
  by (simp add: g(2) normalize-coprime-degree)
thus False
  using  $fa ij(1)$  nth-mem sq square-free-factorizationD'(3) by fastforce
qed
have  $ceq$ :  $c = c' * (\prod (a, i) \in \text{set } fs'. (\text{lead-coeff } a) \wedge i)$  using c lcs
  by (simp add: dist prod.distinct-set-conv-list)
have  $fseq$ :  $(\prod (a, i) \in \text{set } fs'. a \wedge i) = (\prod (a, i) \in \text{set } fs'. (\text{normalize } a) \wedge i)$ 
  apply (subst prod.distinct-set-conv-list[OF dist])
  apply (subst prod.distinct-set-conv-list[OF dist2])
  unfolding  $fs$  apply (auto simp add: o-def)
  by (metis (no-types, lifting) case-prod-conv old.prod.exhaust)

have  $f = \text{Polynomial.smult } c' (\prod (a, i) \in \text{set } fs'. a \wedge i)$  using sq square-free-factorizationD(1)
by blast
moreover have  $\dots = \text{Polynomial.smult } c' (\prod (a, i) \in \text{set } fs'. (\text{Polynomial.smult } ((\text{unit-factor } (\text{lead-coeff } a))) (\text{normalize } a)) \wedge i)$ 
  apply (subst undo-normalize[symmetric]) by auto
moreover have  $\dots = \text{Polynomial.smult } c' (\prod (a, i) \in \text{set } fs'. (\text{Polynomial.smult } ((\text{lead-coeff } a) \wedge i) ((\text{normalize } a) \wedge i)))$ 
  apply (subst smult-power) by auto
moreover have  $\dots = \text{Polynomial.smult } c' (\text{Polynomial.smult } (\prod (a, i) \in \text{set } fs'. ((\text{lead-coeff } a) \wedge i)) (\prod (a, i) \in \text{set } fs'. (\text{normalize } a) \wedge i))$ 
  apply (subst finite-smult-distr) by (auto simp add: dist)
moreover have  $\dots = \text{Polynomial.smult } (c' * (\prod (a, i) \in \text{set } fs'. (\text{lead-coeff } a) \wedge i)) (\prod (a, i) \in \text{set } fs'. (\text{normalize } a) \wedge i)$ 
  using smult-smult by blast
moreover have  $\dots = \text{Polynomial.smult } c (\prod (a, i) \in \text{set } fs'. a \wedge i)$ 
  unfolding  $ceq fseq$  by auto
ultimately show  $f = \text{Polynomial.smult } c (\prod (a, i) \in \text{set } fs'. a \wedge i)$  by auto
fix  $a i$ 

```

assume $ai: (a,i) \in \text{set } fs$
obtain a' **where** $a': (a',i) \in \text{set } fs' \ a = \text{normalize } a'$ **using** ai **unfolding** fs **by**
auto
show *square-free* a **using** *square-free-normalize* a'
using sq *square-free-factorizationD(2)* **by** *blast*
show $0 < \text{degree } a \ 0 < i$ **using** *degree-normalize* a'
using sq *square-free-factorizationD'(3)* **by** *fastforce+*
fix $b \ j$
assume $bj: (b,j) \in \text{set } fs \ (a,i) \neq (b,j)$
obtain b' **where** $b': (b',j) \in \text{set } fs' \ b = \text{normalize } b'$ **using** bj **unfolding** fs **by**
auto
show *algebraic-semidom-class.coprime* $a \ b$ **using** $a' \ b'$ **apply** *auto*
using $bj(2)$ sq *square-free-factorizationD(3)* **by** *fastforce*
qed

lemma *undo-factorize-correct*:

assumes *factorize-rat-poly-monic* $p = (c,fs)$
assumes $\bigwedge f \ p. (f,p) \in \text{set } fs \implies f \in \text{set } ftrs$
shows *undo-factorize* $(c, \text{map } (\lambda(f,pow). (\text{index-of } ftrs \ f, \ pow))) \ fs) \ (\text{sign-vec } ftrs \ x) = \text{squash } (rpoly \ p \ x)$
proof –
have $p: p = \text{smult } c \ (\prod (a, i) \in \text{set } fs. a \ ^i)$
using *assms(1)* *factorize-rat-poly-monic-square-free-factorization* *square-free-factorizationD(1)*
by *blast*
have $fs: \text{distinct } fs$
using *assms(1)* *factorize-rat-poly-monic-square-free-factorization* *square-free-factorizationD(5)*
by *blast*
have $rpoly \ p \ x = ((\text{real-of-rat } c) * rpoly \ (\prod (a, i) \in \text{set } fs. a \ ^i) \ x)$
using p **by** (*simp add: of-rat-hom.map-poly-hom-smult*)
moreover **have** $\dots = ((\text{real-of-rat } c) * rpoly \ (\prod ai \in \text{set } fs. \text{case } ai \ \text{of } (a,i) \Rightarrow a \ ^i) \ x)$
by *blast*
moreover **have** $\dots = ((\text{real-of-rat } c) * (\prod ai \in \text{set } fs. \text{case } ai \ \text{of } (a,i) \Rightarrow rpoly \ (a \ ^i) \ x))$
by (*simp add: finite-prod-map-of-rat-poly-hom*)
moreover **have** $\dots = ((\text{real-of-rat } c) * (\prod ai \in \text{set } fs. \text{case } ai \ \text{of } (a,i) \Rightarrow (rpoly \ a \ x) \ ^i))$
by (*metis (mono-tags, lifting) of-rat-poly-hom.hom-power poly-hom.hom-power split-cong*)
moreover **have** $\dots = ((\text{real-of-rat } c) * (\text{prod-list } (\text{map } (\lambda ai. \text{case } ai \ \text{of } (a,i) \Rightarrow (rpoly \ a \ x) \ ^i) \ fs)))$
by (*simp add: fs prod.distinct-set-conv-list*)
ultimately **have** $rpoly \ p \ x = ((\text{real-of-rat } c) * (\text{prod-list } (\text{map } (\lambda ai. \text{case } ai \ \text{of } (a,i) \Rightarrow (rpoly \ a \ x) \ ^i) \ fs)))$ **by** *auto*

then **have** $\text{squash } (rpoly \ p \ x) = \text{squash } c * \text{prod-list } (\text{map } \text{squash } (\text{map } (\lambda ai. \text{case } ai \ \text{of } (a,i) \Rightarrow (rpoly \ a \ x) \ ^i) \ fs))$
by (*auto simp add: squash-mult squash-prod-list o-def*)
moreover **have** $\dots = \text{squash } c * \text{prod-list } (\text{map } (\lambda ai. \text{case } ai \ \text{of } (a,i) \Rightarrow \text{squash}$

```

((rpoly a x ^ i) fs)
  apply (simp add: o-def)
  by (simp add: prod.case-distrib)
  ultimately have rp:squash(rpoly p x) = squash c * prod-list (map ( $\lambda i. \text{case } ai$ 
of (a, i)  $\Rightarrow$  squash (rpoly a x ^ i) fs)
  using squash-pow
  by presburger
have undo-factorize
  (c, map ( $\lambda(f, pow).(\text{index-of ftrs } f, pow)$ ) fs) (sign-vec ftrs x) =
squash
  (c * ( $\prod xa \leftarrow fs. \text{case } xa \text{ of } (f, y) \Rightarrow \text{sign-vec ftrs } x ! \text{index-of ftrs } f \wedge y$ ))
  unfolding undo-factorize-def apply (auto simp add: o-def)
  by (metis (mono-tags, lifting) case-prod-conv old.prod.exhaust)
moreover have ... = squash
  (c * ( $\prod xa \leftarrow fs. \text{case } xa \text{ of } (f, y) \Rightarrow (\text{squash } (\text{rpoly } f \ x)) \wedge y$ ))
  using assms(2) sign-vec-index-of map-eq-conv split-cong by (smt (verit, del-insts))
ultimately show ?thesis using rp
  by (metis (mono-tags, lifting) of-rat-hom.hom-mult squash-idem squash-mult
squash-real-of-rat)
qed

```

```

lemma length-sign-vec[simp]:
  shows length (sign-vec ps x) = length ps unfolding sign-vec-def by auto

```

```

lemma factorize-polys-has-factors:
  assumes factorize-polys ps = (ftrs, data)
  assumes p  $\in$  set ps
  assumes factorize-rat-poly-monic p = (c, fs)
  shows set (map fst fs)  $\subseteq$  set ftrs
  using assms unfolding factorize-polys-def Let-def apply auto
  by (metis UN-iff fst-conv image-eqI snd-conv)

```

```

lemma factorize-polys-undo-factorize-polys:
  assumes factorize-polys ps = (ftrs, data)
  shows undo-factorize-polys data (sign-vec ftrs x) = sign-vec ps x
  unfolding list-eq-iff-nth-eq undo-factorize-polys-def apply auto
proof –
  show leq: length data = length ps
    using assms unfolding factorize-polys-def by (auto simp add: Let-def)
  fix i
  assume il: i < length data
  obtain c fs where cfs: factorize-rat-poly-monic (ps ! i) = (c, fs)
    by (meson surj-pair)
  then have fst: set (map fst fs)  $\subseteq$  set ftrs
    using assms factorize-polys-has-factors il leq nth-mem by fastforce
  have *: data ! i = (c, map ( $\lambda(f, pow). (\text{index-of ftrs } f, pow)$ ) fs)
    using assms unfolding factorize-polys-def
    using cfs il by (auto simp add: Let-def cfs)
  have undo-factorize (data ! i) (sign-vec ftrs x) = squash (rpoly (ps ! i) x) un-

```

```

folding *
  apply (subst undo-factorize-correct[of ps ! i])
  apply (auto simp add: cfs)
  using fsts by auto
  thus undo-factorize (data ! i) (sign-vec ftrs x) = sign-vec ps x ! i
  using leq il sign-vec-def by auto
qed

lemma factorize-polys-irreducible-monic:
  assumes factorize-polys ps = (fs,data)
  shows distinct fs  $\bigwedge$  f. f  $\in$  set fs  $\implies$  irreducible f  $\wedge$  monic f
  using assms unfolding factorize-polys-def Let-def apply auto
  using factorize-rat-poly-monic-irreducible-monic
  apply (metis prod.collapse)
  using factorize-rat-poly-monic-irreducible-monic
  by (metis prod.collapse)

lemma factorize-polys-square-free:
  assumes factorize-polys ps = (fs,data)
  shows  $\bigwedge$  f. f  $\in$  set fs  $\implies$  square-free f
  using assms factorize-polys-irreducible-monic(2) irreducible-imp-square-free by
blast

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
  by (metis (no-types, lifting) assms(5) coprime-0(2) coprime-def' f(1) f(2) g(1)
g(2) irreducible-normalized-divisors normalize-dvd-iff normalize-idem normalize-monic)

lemma factorize-polys-coprime:
  assumes factorize-polys ps = (fs,data)
  shows  $\bigwedge$  f g. f  $\in$  set fs  $\implies$  g  $\in$  set fs  $\implies$  f  $\neq$  g  $\implies$  coprime f g
  using assms factorize-polys-irreducible-monic(2) irreducible-monic-coprime by
auto

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)
  by (metis assms gcd-dvd-1 of-rat-hom.map-poly-gcd of-rat-poly-hom.hom-dvd-1)

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 –
  have forward: coprime p (q::rat poly)  $\longrightarrow$  coprime (real-of-rat-poly p) ((real-of-rat-poly
q)::real poly)
  using coprime-rat-poly-real-poly by auto

```

```

have backward: coprime (real-of-rat-poly p) ((real-of-rat-poly q)::real poly)  $\implies$ 
coprime p (q::rat poly)
proof -
  assume copr-real: comm-monoid-mult-class.coprime (real-of-rat-poly p) (real-of-rat-poly
q)
  have degree (gcd p (q::rat poly)) > 0  $\implies$  False
  proof -
    assume deg: degree (gcd p (q::rat poly)) > 0
    then have  $\exists y. y \text{ dvd } p \wedge y \text{ dvd } q \wedge \text{degree } y > 0$ 
      by blast
    then obtain y where yprop: y dvd p  $\wedge$  y dvd q  $\wedge$  degree y > 0
      by auto
    then have (real-of-rat-poly y) dvd (real-of-rat-poly p)  $\wedge$ 
(real-of-rat-poly y) dvd (real-of-rat-poly q)  $\wedge$  degree y > 0
      by simp
    then show False
      using copr-real apply (auto)
      by fastforce
  qed
  then show comm-monoid-mult-class.coprime p (q::rat poly)
    using comm-monoid-gcd-class.gcd-dvd-1
    by (metis Missing-Polynomial-Factorial.is-unit-field-poly copr-real gcd-zero-iff'
neq0-conv of-rat-poly-hom.hom-zero)
  qed
  show ?thesis
    using forward backward by auto
qed

```

```

lemma factorize-polys-map-distinct:
assumes factorize-polys ps = (fs,data)
assumes fss = map real-of-rat-poly fs
shows distinct fss
using factorize-polys-irreducible-monic[OF assms(1)]
unfolding assms(2)
apply (simp add: distinct-conv-nth)
by (metis of-rat-eq-iff of-rat-hom.coeff-map-poly-hom poly-eqI)

```

```

lemma factorize-polys-map-square-free:
assumes factorize-polys ps = (fs,data)
assumes fss = map real-of-rat-poly fs
shows  $\bigwedge f. f \in \text{set } fss \implies \text{square-free } f$ 
using factorize-polys-square-free[OF assms(1)]
using assms(2) field-hom-0'.square-free-map-poly of-rat-hom.field-hom-0'-axioms
by auto

```

```

lemma factorize-polys-map-coprime:
assumes factorize-polys ps = (fs,data)
assumes fss = map real-of-rat-poly fs
shows  $\bigwedge f g. f \in \text{set } fss \implies g \in \text{set } fss \implies f \neq g \implies \text{coprime } f g$ 

```

```

using factorize-polys-coprime[OF assms(1)] coprime-rat-poly-real-poly unfolding
assms(2)
by auto

```

```

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  $\bigwedge p. p \in \text{set } ps \implies \text{coprime } p \ q$ 
proof –
  fix p
  assume  $p \in \text{set } ps$ 
  then obtain r where  $r: \text{prod-list } ps = r * p$ 
    using remove1-retains-prod by blast
  show coprime p q
    apply (rule coprime-prod[of r 1])
    using assms r apply auto
    by blast
qed

```

```

lemma factorize-polys-square-free-prod-list:
  assumes factorize-polys ps = (fs,data)
  shows square-free (prod-list fs)
proof (rule square-freeI)
  from factorize-polys-coprime[OF assms]
  have coprime:  $\bigwedge p \ q. p \in \text{set } fs \implies q \in \text{set } fs \implies p \neq q \implies \text{coprime } p \ q$  .
  from factorize-polys-square-free[OF assms]
  have sq:  $\bigwedge p. p \in \text{set } fs \implies \text{square-free } p$  .
  thus prod-list fs  $\neq 0$  unfolding prod-list-zero-iff
    using square-free-def by blast
  fix q
  assume degree q > 0  $q * q \ \text{dvd} \ \text{prod-list } fs$ 
  from irreduciblea-factor[OF this(1)] this(2) obtain q where
    irr: irreducible q and dvd:  $q * q \ \text{dvd} \ \text{prod-list } fs$  unfolding dvd-def by auto
  hence dvd':  $q \ \text{dvd} \ \text{prod-list } fs$  unfolding dvd-def by auto
  from irreducible-dvd-prod-list[OF irr dvd'] obtain b where
    mem:  $b \in \text{set } fs$  and dvd1:  $q \ \text{dvd} \ b$  by auto
  from dvd1 obtain k where  $b: b = q * k$  unfolding dvd-def by auto
  from split-list[OF mem] b obtain bs1 bs2 where  $bs: fs = bs1 \ @ \ b \ \# \ bs2$  by
auto
  from irr have q0:  $q \neq 0$  and dq: degree q > 0 unfolding irreduciblea-def by
auto
  have square-free (q * k) using sq b mem by auto
  from this[unfolded square-free-def, THEN conjunct2, rule-format, OF dq]
  have qk:  $\neg q \ \text{dvd} \ k$  by simp
  from dvd[unfolded bs b] have  $q * q \ \text{dvd} \ q * (k * \text{prod-list } (bs1 \ @ \ bs2))$ 
    by (auto simp: ac-simps)
  with q0 have  $q \ \text{dvd} \ k * \text{prod-list } (bs1 \ @ \ bs2)$  by auto
  with irr qk have  $q \ \text{dvd} \ \text{prod-list } (bs1 \ @ \ bs2)$  by auto

```

```

from irreducible-dvd-prod-list[OF irr this] obtain b' where
  mem': b' ∈ set (bs1 @ bs2) and dvd2: q dvd b' by fastforce
from dvd1 dvd2 have q dvd gcd b b' by auto
with dq is-unit-iff-degree[OF q0] have cop: ¬ coprime b b' by force
from mem' have b' ∈ set fs unfolding bs by auto
have b': b' = b using coprime
  using ⟨b' ∈ set fs⟩ cop mem by blast
with mem' bs factorize-polys-irreducible-monic(1)[OF assms] show False by
auto
qed

```

```

lemma factorize-polys-map-square-free-prod-list:
  assumes factorize-polys ps = (fs,data)
  assumes fss = map real-of-rat-poly fs
  shows square-free (prod-list fss)
  using factorize-polys-square-free-prod-list[OF assms(1)] unfolding assms(2)
  by (simp add: of-rat-hom.square-free-map-poly)

```

```

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

```

```

proof –
  fix f
  assume f: f ∈ set fss
  from factorize-polys-map-square-free[OF assms]
  have sq:  $\bigwedge p. p \in \text{set } fss \implies \text{square-free } p$  .
  have z:  $\bigwedge p. p \in \text{set } fss \implies p \neq 0$  using sq square-free-def by blast
  have c: coprime (prod-list fss) (pderiv (prod-list fss))
    apply (simp add: separable-def[symmetric] square-free-iff-separable[symmetric])
    using factorize-polys-map-square-free-prod-list[OF assms] .
  from coprime-prod-list[OF z c f]
  show coprime f (pderiv (prod-list fss)) by auto
qed

```

```

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:: rat poly list
  shows pairwise-coprime-list (fst(factorize-polys qs))
proof –
  let ?fs = fst(factorize-polys qs)
  have ( $\forall m < \text{length } ?fs. \forall n < \text{length } ?fs.$ 
     $m \neq n \longrightarrow \text{coprime } (?fs ! n) (?fs ! m)$ )
  proof clarsimp

```

```

fix m n
assume m < length (fst (factorize-polys qs))
assume n < length (fst (factorize-polys qs))
assume m ≠ n
show algebraic-semidom-class.coprime (fst (factorize-polys qs) ! n)
      (fst (factorize-polys qs) ! m)
  by (metis ⟨m < length (fst (factorize-polys qs))⟩ ⟨m ≠ n⟩ ⟨n < length (fst
(factorize-polys qs))⟩ coprime-iff-coprime distinct-conv-nth factorize-polys-coprime
factorize-polys-def factorize-polys-irreducible-monic(1) fstI nth-mem)
qed
then show ?thesis unfolding pairwise-coprime-list-def by auto
qed

```

lemma squarefree-factorization-degree:

```

assumes square-free-factorization p (c,fs)
shows degree p = sum-list (map (λ(f,c). c * degree f) fs)
proof –
  have p =
    Polynomial.smult c
      (∏ (a, i)∈set fs. a ^ i) using assms unfolding square-free-factorization-def
  by blast
  then have degree p = degree (∏ (a, i)∈set fs. a ^ i)
    using assms square-free-factorizationD(4) by fastforce
  also have ... = degree (prod-list (map (λ(f,c). f ^ c) fs))
    by (metis assms prod.distinct-set-conv-list square-free-factorizationD(5))
  also have ... = (∑ (a, i)←fs. degree (a ^ i))
    apply (subst degree-prod-list-eq)
    apply (auto simp add: o-def)
    using assms degree-0 square-free-factorizationD(2) apply blast
    by (simp add: prod.case-distrib)
  ultimately show ?thesis
    by (smt (verit, ccfv-SIG) Polynomial.degree-power-eq Suc-eq-plus1 assms de-
gree-0 map-eq-conv split-cong square-free-factorizationD(2))
qed

```

17.3 Auxiliary Polynomial Lemmas

definition roots-of-coprime-r:: real poly list ⇒ real set

where roots-of-coprime-r qs = {x. poly (coprime-r qs) x = 0}

lemma crb-lem-pos:

```

fixes x:: real
fixes p:: real poly
assumes x: poly p x = 0
assumes p: p ≠ 0
shows x < crb p
using cauchy-root-bound[of p x] apply (auto)
unfolding crb-def apply (auto)
using p x

```

by *linarith*

lemma *crb-lem-neg*:
fixes *x*:: real
fixes *p*:: real poly
assumes *x*: poly *p* *x* = 0
assumes *p*: *p* ≠ 0
shows *x* > -*crb* *p*
using *cauchy-root-bound*[of *p* *x*] **apply** (*auto*)
unfolding *crb-def* **apply** (*auto*)
using *p* *x* **by** *linarith*

lemma *prod-zero*:
shows $\forall x . \text{poly } (\text{prod-list } (qs:: \text{rat poly list})) x = 0 \iff (\exists q \in \text{set } (qs). \text{poly } q x = 0)$
apply *auto*
using *poly-prod-list-zero-iff* **apply** *blast*
using *poly-prod-list-zero-iff* **by** *blast*

lemma *coprime-r-zero1*: poly (*coprime-r* *qs*) (*crb* (*prod-list* *qs*)) = 0
by (*simp* *add: coprime-r-def*)

lemma *coprime-r-zero2*: poly (*coprime-r* *qs*) (-*crb* (*prod-list* *qs*)) = 0
by (*simp* *add: coprime-r-def*)

lemma *coprime-mult*:
fixes *a*:: real poly
fixes *b*:: real poly
fixes *c*:: real poly
assumes *algebraic-semidom-class.coprime* *a* *b*
assumes *algebraic-semidom-class.coprime* *a* *c*
shows *algebraic-semidom-class.coprime* *a* (*b***c*)
using *assms*(1) *assms*(2) **by** *auto*

lemma *coprime-r-coprime-prop*:
fixes *ps*:: rat poly list
assumes *factorize-polys* *ps* = (*fs*,*data*)
assumes *fss* = *map* *real-of-rat-poly* *fs*
shows $\bigwedge f. f \in \text{set } fss \implies \text{coprime } f (\text{coprime-r } fss)$
proof *clarsimp*
fix *f*:: real poly
assume *f-in*: *f* ∈ *set* *fss*
have *nonz-prod*: *prod-list* *fss* ≠ 0 **using** *factorize-polys-map-square-free* **apply** (*auto*)
using *assms*(1) *assms*(2) *square-free-def* **by** *fastforce*
have *nonz-f*: *f* ≠ 0 **using** *f-in* *factorize-polys-map-square-free* **apply** (*auto*)
using *assms*(1) *assms*(2) *square-free-def* **by** *fastforce*

```

have copr-pderiv: algebraic-semidom-class.coprime f (pderiv (prod-list fss)) using
factorize-polys-map-coprime-pderiv
  apply (auto)
  using f-in assms(1) assms(2) by auto
have z-iff:  $\forall x. \text{poly } f x = 0 \longrightarrow \text{poly } (\text{prod-list } fss) x = 0$ 
  using f-in apply (auto)
  using poly-prod-list-zero-iff by blast
let ?inf-p = [:(crb (prod-list fss)),1:]::real poly
have copr-inf: algebraic-semidom-class.coprime f ([:(crb (prod-list fss)),1:])
proof -
  have zero-prop:  $\forall x. \text{poly } ?inf-p x = 0 \longleftrightarrow x = \text{crb } (\text{prod-list } fss)$ 
    by auto
  have poly (prod-list fss) (crb (prod-list fss))  $\neq 0$ 
proof -
  have h:  $\forall x. \text{poly } (\text{prod-list } fss) x = 0 \longrightarrow x < (\text{crb } (\text{prod-list } fss))$ 
    using nonz-prod crb-lem-pos[where p = prod-list fss]
    by auto
  then show ?thesis by auto
qed
then have nonzero:  $\text{poly } f (\text{crb } (\text{prod-list } fss)) \neq 0$ 
  using z-iff by auto
then have  $\neg(\exists x. \text{poly } f x = 0 \wedge \text{poly } ?inf-p x = 0)$ 
  by simp
have is-unit-gcd: is-unit (gcd ?inf-p f)
  using prime-elem-imp-gcd-eq prime-elem-iff-irreducible linear-irreducible-field
  apply (auto) using nonzero
proof -
  have f1:  $\forall x0. - (x0::real) = - 1 * x0$ 
    by simp
  have (1::real)  $\neq 0$ 
    by auto
  then have is-unit (gcd (pCons (- 1 * real-of-int (crb (prod-list fss))) 1) f)
    using f1 by (metis (no-types) is-unit-gcd nonzero one-poly-eq-simps(1)
poly-eq-0-iff-dvd prime-elem-imp-coprime prime-elem-linear-field-poly)
  then show degree (gcd (pCons (- real-of-int (crb (prod-list fss))) 1) f) = 0
    by simp
qed
then show ?thesis
  using is-unit-gcd
  by (metis gcd.commute gcd-eq-1-imp-coprime is-unit-gcd-iff)
qed
let ?ninf-p = [:(crb (prod-list fss)),1:]::real poly
have copr-neg-inf: algebraic-semidom-class.coprime f ([:(crb (prod-list fss)),1:])
proof -
  have h:  $\forall x. \text{poly } f x = 0 \longrightarrow \text{poly } (\text{prod-list } fss) x = 0$ 
    using f-in apply (auto)
    using poly-prod-list-zero-iff by blast
  have zero-prop:  $\forall x. \text{poly } ?ninf-p x = 0 \longleftrightarrow x = -\text{crb } (\text{prod-list } fss)$ 
    by auto

```

```

have poly (prod-list fss) (-crb (prod-list fss)) ≠ 0
proof -
  have h: ∀ x. poly (prod-list fss) x = 0 → x > (-crb (prod-list fss))
    using nonz-prod crb-lem-neg[where p = prod-list fss]
    by auto
  then show ?thesis by auto
qed
then have nonzero: poly f (-crb (prod-list fss)) ≠ 0
  using z-iff by auto
then have ¬(∃ x. poly f x = 0 ∧ poly ?ninf-p x = 0)
  using zero-prop by auto
have is-unit-gcd: is-unit (gcd ?ninf-p f)
  using prime-elem-imp-gcd-eq prime-elem-iff-irreducible linear-irreducible-field
  apply (auto) using nonzero
proof -
  have f1: (1::real) ≠ 0
    by auto
  have ¬ pCons (real-of-int (crb (prod-list fss))) 1 dvd f
    using nonzero by auto
  then show degree (gcd (pCons (real-of-int (crb (prod-list fss))) 1) f) = 0
    using f1 by (metis (no-types) Missing-Polynomial-Factorial.is-unit-field-poly
coprime-imp-gcd-eq-1 is-unit-gcd-iff one-poly-eq-simps(1) prime-elem-imp-coprime
prime-elem-linear-field-poly)
  qed
  then show ?thesis
    using is-unit-gcd
    by (metis gcd.commute gcd-eq-1-imp-coprime is-unit-gcd-iff)
qed
show algebraic-semidom-class.coprime f (coprime-r fss)
  using copr-pderiv coprime-mult unfolding coprime-r-def
  using copr-inf copr-neg-inf by blast
qed

lemma coprime-r-nonzero:
  fixes ps:: rat poly list
  assumes factorize-polys ps = (fs,data)
  assumes nonempty-fs: fs ≠ []
  assumes fss-is: fss = map real-of-rat-poly fs
  shows (coprime-r fss) ≠ 0
proof -
  have nonempty-fss: fss ≠ [] using nonempty-fs fss-is by auto
  have deg-f: ∀ f ∈ set (fs). degree f > 0
    using factorize-polys-irreducible-monic
    apply (auto)
    using asms(1) irreducible-degree-field by blast
  then have deg-fss: ∀ f ∈ set (fss). degree f > 0
    using fss-is by simp
  then have fss-nonz: ∀ f ∈ set (fss). f ≠ 0
    by auto

```

```

have fss ≠ [] → ((∀ f ∈ set (fss). (degree f > 0 ∧ f ≠ 0)) → degree (prod-list
fss) > 0)
proof (induct fss)
  case Nil
  then show ?case
  by blast
next
  case (Cons a fss)
  show ?case
  proof clarsimp
    assume z-lt: 0 < degree a
    assume anonz: a ≠ 0
    assume fnonz: ∀ f ∈ set fss. 0 < degree f ∧ f ≠ 0
    have h: degree (a * prod-list fss) = degree a + degree (prod-list fss)
      using degree-mult-eq[where p = a, where q = prod-list fss] anonz fnonz
      by auto
    then show 0 < degree (a * prod-list fss)
      using z-lt Cons.hyps by auto
  qed
qed
then have degree (prod-list fss) > 0
  using nonempty-fss deg-fss fss-nonz by auto
then have pderiv-nonzero: pderiv (prod-list fss) ≠ 0
  by (simp add: pderiv-eq-0-iff)
have (([:-(crb (prod-list fss)),1:] * [:(crb (prod-list fss)),1:])) ≠ 0
  by auto
then show ?thesis using pderiv-nonzero
  unfolding coprime-r-def apply (auto)
  by (metis offset-poly-eq-0-lemma right-minus-eq synthetic-div-unique-lemma)
qed

```

lemma Rolle-pderiv:

```

fixes q:: real poly
fixes x1 x2:: real
shows (x1 < x2 ∧ poly q x1 = 0 ∧ poly q x2 = 0) → (∃ w. x1 < w ∧ w < x2
∧ poly (pderiv q) w = 0)
using Rolle-deriv apply (auto)
by (metis DERIV-unique Rolle continuous-at-imp-continuous-on poly-DERIV
poly-differentiable poly-isCont)

```

lemma coprime-r-roots-prop:

```

fixes qs:: real poly list
assumes pairwise-rel-prime: ∀ q1 q2. (q1 ≠ q2 ∧ (List.member qs q1) ∧ (List.member
qs q2)) → coprime q1 q2
shows ∀ x1. ∀ x2. ((x1 < x2 ∧ (∃ q1 ∈ set (qs). (poly q1 x1) = 0) ∧ (∃ q2 ∈
set (qs). (poly q2 x2) = 0)) → (∃ q. x1 < q ∧ q < x2 ∧ poly (coprime-r qs) q =
0))
proof clarsimp
  fix x1:: real

```

```

fix x2:: real
fix q1:: real poly
fix q2:: real poly
assume x1 < x2
assume q1-in: q1 ∈ set qs
assume q1-0: poly q1 x1 = 0
assume q2-in: q2 ∈ set qs
assume q2-0: poly q2 x2 = 0
have prod-z-x1: poly (prod-list qs) x1 = 0 using q1-in q1-0
  using poly-prod-list-zero-iff by blast
have prod-z-x2: poly (prod-list qs) x2 = 0 using q2-in q2-0
  using poly-prod-list-zero-iff by blast
have ∃ w>x1. w < x2 ∧ poly (pderiv (prod-list qs)) w = 0
  using Rolle-pderiv[where q = prod-list qs] prod-z-x1 prod-z-x2
  using ⟨x1 < x2⟩ by blast
then obtain w where w-def: w > x1 ∧ w < x2 ∧ poly (pderiv (prod-list qs)) w
= 0
  by auto
then have poly (coprime-r qs) w = 0
  unfolding coprime-r-def
  by simp
then show ∃ q>x1. q < x2 ∧ poly (coprime-r qs) q = 0
  using w-def by blast
qed

```

17.4 Setting Up the Procedure: Lemmas

definition *has-no-zeros*::rat list ⇒ bool
where *has-no-zeros* l = (0 ∉ set l)

lemma *hnz-prop*: *has-no-zeros* l ⇔ ¬(∃ k < length l. l ! k = 0)
unfolding *has-no-zeros-def*
by (*simp add: in-set-conv-nth*)

definition *cast-rat-list*:: rat poly list ⇒ real poly list
where *cast-rat-list* qs = map real-of-rat-poly qs

definition *consistent-sign-vectors-r*::real poly list ⇒ real set ⇒ rat list set
where *consistent-sign-vectors-r* qs S = (*signs-at* qs) ‘ S

lemma *consistent-sign-vectors-consistent-sign-vectors-r*:
shows*consistent-sign-vectors-r* (cast-rat-list qs) S = *consistent-sign-vectors* qs S
unfolding *consistent-sign-vectors-r-def* *cast-rat-list-def* *consistent-sign-vectors-def*
sign-vec-def *signs-at-def*
by auto

lemma *coprime-over-reals-coprime-over-rats*:
fixes qs:: 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  $\neg \text{algebraic-semidom-class.coprime } (nth \ qs \ p1) \ (nth \ qs \ p2)$ 
proof -
  have isx:  $\exists (x::\text{real}). \ csa = (\text{sign-vec } qs \ x)$ 
  using csa-in unfolding consistent-sign-vectors-def by auto
  then obtain x where havex:  $csa = (\text{sign-vec } qs \ x)$  by auto
  then have expolys:  $\text{poly } (\text{real-of-rat-poly } (nth \ qs \ p1)) \ x = 0 \wedge \text{poly } (\text{real-of-rat-poly } (nth \ qs \ p2)) \ x = 0$ 
  using havex unfolding sign-vec-def squash-def
  by (smt (verit) class-field.neg-1-not-0 length-map map-map nth-map one-neq-zero p1p2)
  then have  $\neg \text{coprime } (\text{real-of-rat-poly } (nth \ qs \ p1)) \ ((\text{real-of-rat-poly } (nth \ qs \ p2))::\text{real poly})$ 
  using coprime-poly-0 by force
  then show ?thesis
  using coprime-rat-poly-real-poly by auto
qed

```

lemma *zero-above*:

```

fixes qs:: rat poly list
fixes x1:: real
assumes hnz: has-no-zeros (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 clarsimp
  fix x2
  assume x1-lt:  $x1 < x2$ 
  assume diff-sign-vec:  $\text{sign-vec } qs \ x1 \neq \text{sign-vec } qs \ x2$ 
  then have  $\exists q \in \text{set } qs. \ \text{squash } (\text{rpoly } q \ x1) \neq \text{squash } (\text{rpoly } q \ x2)$ 
  unfolding sign-vec-def
  by simp
  then obtain q where q-prop:  $q \in \text{set } qs \wedge \text{squash } (\text{rpoly } q \ x1) \neq \text{squash } (\text{rpoly } q \ x2)$ 
  by auto
  then have q-in:  $q \in \text{set } qs$  by auto
  have poss1:  $\text{squash } (\text{rpoly } q \ x1) = -1 \wedge \text{squash } (\text{rpoly } q \ x2) = 1 \longrightarrow (\exists r > x1. \ r \leq x2 \wedge (\exists q \in \text{set } qs. \ \text{rpoly } q \ r = 0))$ 
  using poly-IVT-pos[of x1 x2] using x1-lt unfolding squash-def apply (auto)
  using q-prop by fastforce
  have poss2:  $\text{squash } (\text{rpoly } q \ x1) = 1 \wedge \text{squash } (\text{rpoly } q \ x2) = -1 \longrightarrow (\exists r > x1. \ r \leq x2 \wedge (\exists q \in \text{set } qs. \ \text{rpoly } q \ r = 0))$ 
  using poly-IVT-neg[of x1 x2] using x1-lt unfolding squash-def apply (auto)
  using q-prop by fastforce
  have poss3:  $\text{squash } (\text{rpoly } q \ x2) = 0 \longrightarrow (\exists r > x1. \ r \leq x2 \wedge (\exists q \in \text{set } qs. \ \text{rpoly } q \ r = 0))$ 
  using x1-lt unfolding squash-def apply (auto)

```

```

    using q-prop by blast
  have (q ∈ set qs ∧ rpoly q x1 = 0) → ¬has-no-zeros (sign-vec qs x1)
    unfolding has-no-zeros-def sign-vec-def
    by (smt (verit) image-eqI list.set-map o-apply squash-def)
  have not-poss4: squash (rpoly q x1) ≠ 0
    using hnz q-in unfolding squash-def
    using ⟨q ∈ set qs ∧ rpoly q x1 = 0 → ¬ has-no-zeros (sign-vec qs x1)⟩ by
  auto
  then show ∃ r > x1. r ≤ x2 ∧ (∃ q ∈ set qs. rpoly q r = 0)
    using q-prop poss1 poss2 poss3 not-poss4 by (metis (no-types, lifting) squash-def)

```

qed

lemma zero-below:

```

  fixes qs:: rat poly list
  fixes x1:: real
  assumes hnz: has-no-zeros (sign-vec qs x1)
  shows ∀ x2 < x1. ((sign-vec qs x1) ≠ (sign-vec qs x2)) →
    (∃ (r::real) < x1. (r ≥ x2 ∧ (∃ q ∈ set(qs). rpoly q r = 0)))
proof clarsimp
  fix x2
  assume x1-gt: x2 < x1
  assume diff-sign-vec: sign-vec qs x1 ≠ sign-vec qs x2
  then have ∃ q ∈ set qs. squash (rpoly q x1) ≠ squash (rpoly q x2)
    unfolding sign-vec-def
    by simp
  then obtain q where q-prop: q ∈ set qs ∧ squash (rpoly q x1) ≠ squash (rpoly
  q x2)
    by auto
  then have q-in: q ∈ set qs by auto
  have poss1: squash (rpoly q x1) = -1 ∧ squash (rpoly q x2) = 1 → (∃ r < x1.
  (r ≥ x2 ∧ (∃ q ∈ set(qs). rpoly q r = 0)))
    using poly-IVT-neg[of x2 x1] using x1-gt unfolding squash-def apply (auto)
    using q-prop by fastforce
  have poss2: squash (rpoly q x1) = 1 ∧ squash (rpoly q x2) = -1 → (∃ r < x1.
  (r ≥ x2 ∧ (∃ q ∈ set(qs). rpoly q r = 0)))
    using poly-IVT-pos[of x2 x1] using x1-gt unfolding squash-def apply (auto)
    using q-prop by fastforce
  have poss3: squash (rpoly q x2) = 0 → (∃ r < x1. (r ≥ x2 ∧ (∃ q ∈ set(qs).
  rpoly q r = 0)))
    using x1-gt unfolding squash-def apply (auto)
    using q-prop by blast
  have (q ∈ set qs ∧ rpoly q x1 = 0) → ¬has-no-zeros (sign-vec qs x1)
    unfolding has-no-zeros-def sign-vec-def
    by (smt (verit) comp-apply image-eqI list.set-map squash-def)
  have not-poss4: squash (rpoly q x1) ≠ 0
    using hnz q-in unfolding squash-def
    using ⟨q ∈ set qs ∧ rpoly q x1 = 0 → ¬ has-no-zeros (sign-vec qs x1)⟩ by
  auto

```

```

then show ( $\exists r < x1. (r \geq x2 \wedge (\exists q \in \text{set}(qs). \text{rpoly } q \ r = 0))$ )
using q-prop poss1 poss2 poss3 not-poss4 by (metis (no-types, lifting) squash-def)
qed

```

lemma *sorted-list-lemma:*

```

fixes l:: real list
fixes a b:: real
fixes n:: nat
assumes  $a < b$ 
assumes  $(n + 1) < \text{length } l$ 
assumes strict-sort: sorted-wrt (<) l
assumes lt-a: (l ! n) < a
assumes b-lt: b < (l ! (n+1))
shows  $\neg(\exists(x::\text{real}). (\text{List.member } l \ x \wedge a \leq x \wedge x \leq b))$ 
proof -
  have sorted-hyp-var:  $\forall q1 < \text{length } l. \forall q2 < \text{length } l. (q1 < q2 \longrightarrow$ 
    (l ! q1) < (l ! q2))
    using strict-sort by (auto simp: sorted-wrt-iff-nth-less)
  then have sorted-hyp-var2:  $\forall q1 < \text{length } l. \forall q2 < \text{length } l. ((l ! q1) < (l !$ 
q2))  $\longrightarrow q1 < q2$ 
    using linorder-neqE-nat
    by (metis Groups.add-ac(2) add-mono-thms-linordered-field(5) less-irrefl)
  have  $(\exists(x::\text{real}). (\text{List.member } l \ x \wedge a \leq x \wedge x \leq b)) \implies \text{False}$ 
  proof -
    assume  $(\exists(x::\text{real}). (\text{List.member } l \ x \wedge a \leq x \wedge x \leq b))$ 
    then obtain x where x-prop: List.member l x  $\wedge a \leq x \wedge x \leq b$  by auto
    then have l-prop: List.member l x  $\wedge (l ! n) < x \wedge x < (l ! (n+1))$ 
      using lt-a b-lt by auto
    have nth-l: l ! n < x using l-prop by auto
    have np1th-l: x < l ! (n+1) using l-prop by auto
    have  $\exists k. k < \text{length } l \wedge \text{nth } l \ k = x$  using l-prop
      using index-of-lookup by auto force
    then obtain k where k-prop: k < length l  $\wedge \text{nth } l \ k = x$  by auto
    have n-lt: n < k
      using nth-l sorted-hyp-var k-prop add-lessD1 assms(2) linorder-neqE-nat
nat-SN.gt-trans
      by (meson sorted-hyp-var2)
    have k-gt: k < n + 1
      using sorted-hyp-var np1th-l k-prop
      using assms(2) sorted-hyp-var2 by blast
    show False
      using n-lt k-gt by auto
  qed
  then show ?thesis by auto
qed

```

lemma *roots-of-coprime-r-location-property:*

```

fixes qs:: rat poly list

```

```

fixes sga:: rat list
fixes zer-list
assumes pairwise-rel-prime: pairwise-coprime-list qs
assumes all-squarefree:  $\bigwedge q. q \in \text{set } qs \implies \text{rsquarefree } q$ 
assumes x1-prop: sga = sign-vec qs x1
assumes hnz: has-no-zeros sga
assumes zer-list-prop: zer-list = sorted-list-of-set  $\{(x::\text{real}). (\exists q \in \text{set}(qs). (\text{rpoly } q \ x = 0))\}$ 
shows zer-list  $\neq [] \implies ((x1 < (\text{zer-list } ! 0)) \vee (x1 > (\text{zer-list } ! (\text{length zer-list} - 1))) \vee$ 
 $(\exists n < (\text{length zer-list} - 1). x1 > (\text{zer-list } ! n) \wedge x1 < (\text{zer-list } ! (n+1))))$ 
proof -
let ?zer-list = sorted-list-of-set  $\{(x::\text{real}). (\exists q \in \text{set}(qs). (\text{rpoly } q \ x = 0))\} :: \text{real list}$ 
show ?thesis
proof -
have  $((\forall q. (\text{List.member } qs \ q) \implies q \neq 0) \wedge \text{has-no-zeros } (\text{sign-vec } qs \ x1)) \implies$ 
 $\neg \text{List.member } ?zer\text{-list } x1$ 
proof (induct qs)
case Nil
then show ?case
by auto
next
case (Cons a qs)
then show ?case
proof (clarsimp simp del: List.member-iff)
assume imp:  $((\forall q. \text{List.member } qs \ q \implies q \neq 0) \wedge$ 
 $\text{has-no-zeros } (\text{sign-vec } qs \ x1)) \implies$ 
 $\neg \text{List.member } (\text{sorted-list-of-set } \{x. \exists q \in \text{set } qs. \text{rpoly } q \ x = 0\})$ 
 $x1$ 
assume nonz:  $\forall q. \text{List.member } (a \# qs) \ q \implies q \neq 0$ 
assume hnz: has-no-zeros (sign-vec (a # qs) x1)
assume mem-list: List.member
 $(\text{sorted-list-of-set } \{x. \text{rpoly } a \ x = 0 \vee (\exists q \in \text{set } qs. \text{rpoly } q \ x = 0)\})$ 
 $x1$ 
have has-no-zeros (sign-vec (a # qs) x1)  $\implies$  has-no-zeros (sign-vec qs x1)
proof -
assume hnz: has-no-zeros (sign-vec (a # qs) x1)
have same-vec:  $\text{sign-vec } (a \# qs) \ x1 = ((\text{if } \text{rpoly } a \ x1 > 0 \text{ then } 1 \text{ else if } \text{rpoly } a \ x1 = 0 \text{ then } 0 \text{ else } -1) \# \text{sign-vec } (qs) \ x1)$ 
unfolding sign-vec-def squash-def by auto
have has-no-zeros  $((\text{if } \text{rpoly } a \ x1 > 0 \text{ then } 1 \text{ else if } \text{rpoly } a \ x1 = 0 \text{ then } 0$ 
 $\text{else } -1) \# \text{sign-vec } (qs) \ x1)$ 
 $\implies$  has-no-zeros (sign-vec (qs) x1)
by (simp add: has-no-zeros-def)
then show has-no-zeros (sign-vec qs x1) using hnz same-vec by auto
qed
then have nmem:  $\neg \text{List.member } (\text{sorted-list-of-set } \{x. \exists q \in \text{set } qs. \text{rpoly } q \ x = 0\}) \ x1$ 

```

```

    using hnz nonz imp by auto
  have  $\forall q \in \text{set } qs. q \neq 0$ 
    using nonz by auto
  then have  $\forall q \in \text{set } qs. \text{finite } \{x. \text{rpoly } q \ x = 0\}$ 
    by (simp add: poly-roots-finite)
  then have fin-set:  $\text{finite } \{x. \exists q \in \text{set } qs. \text{rpoly } q \ x = 0\}$ 
    by auto
    have not-in:  $x1 \notin \{x. \exists q \in \text{set } qs. \text{rpoly } q \ x = 0\}$  using fin-set nmem
set-sorted-list-of-set
  all-squarefree
  by (simp add:  $\langle \text{finite } \{x. \exists q \in \text{set } qs. \text{rpoly } q \ x = 0\} \rangle$ )
  have x1-in:  $x1 \in \{x. \text{rpoly } a \ x = 0 \vee (\exists q \in \text{set } qs. \text{rpoly } q \ x = 0)\}$ 
    using mem-list sorted-list-of-set
  proof -
    have f1:  $\forall r \ R. ((r::\text{real}) \in R \vee \neg \text{List.member } (\text{sorted-list-of-set } R) \ r) \vee$ 
infinite R
    by auto
    have finite  $\{r. \text{rpoly } a \ (r::\text{real}) = 0\}$ 
      apply (subst real-roots-of-rat-poly(1) [symmetric])
      using nonz
      apply simp-all
    done
    then show ?thesis
      using f1  $\langle \text{finite } \{x. \exists q \in \text{set } qs. \text{rpoly } q \ x = 0\} \rangle$  mem-list by fastforce
  qed
  have  $\text{rpoly } a \ x1 \neq 0$  using hnz
    unfolding has-no-zeros-def sign-vec-def squash-def by auto
  then show False using not-in x1-in
    by auto
  qed
  then have non-mem:  $\neg \text{List.member } ?\text{zer-list } x1$ 
    using all-squarefree unfolding rsquarefree-def hnz apply (auto)
    using hnz x1-prop
    by simp
  have  $?\text{zer-list} \neq [] \implies ((x1 \geq (?\text{zer-list} ! 0)) \wedge (x1 \leq (?\text{zer-list} ! (\text{length } ?\text{zer-list} - 1))))$ 
 $\implies (\exists n < (\text{length } ?\text{zer-list} - 1). x1 > (?\text{zer-list} ! n) \wedge x1 < (?\text{zer-list} ! (n+1)))$ 
  proof -
    assume nonempty:  $?\text{zer-list} \neq []$ 
    assume x1-asm:  $(x1 \geq (?\text{zer-list} ! 0)) \wedge (x1 \leq (?\text{zer-list} ! (\text{length } ?\text{zer-list} - 1)))$ 
  have nm1:  $x1 \neq ?\text{zer-list} ! 0$  using non-mem
    using  $\langle \text{sorted-list-of-set } \{x. \exists q \in \text{set } qs. \text{rpoly } q \ x = 0\} \neq [] \rangle$  apply auto
    by (metis (no-types, lifting) in-set-conv-nth length-greater-0-conv)
  have nm2:  $x1 \neq ?\text{zer-list} ! (\text{length } ?\text{zer-list} - 1)$ 
    using non-mem  $\langle \text{sorted-list-of-set } \{x. \exists q \in \text{set } qs. \text{rpoly } q \ x = 0\} \neq [] \rangle$ 
    by auto (metis (lifting) diff-Suc-less in-set-conv-nth length-greater-0-conv
      sorted-list-of-set.length-sorted-key-list-of-set)

```

```

then have x1-asm-var:  $x1 > (?zer-list ! 0) \wedge x1 < ?zer-list ! (length ?zer-list - 1)$ 
using x1-asm nm1 nm2 by auto
have  $(\forall n. (n < (length ?zer-list - 1) \wedge x1 \geq (?zer-list ! n) \longrightarrow x1 \geq (?zer-list ! (n+1)))) \implies False$ 
proof -
  assume assump:  $(\forall n. (n < (length ?zer-list - 1) \wedge x1 \geq (?zer-list ! n) \longrightarrow x1 \geq (?zer-list ! (n+1))))$ 
have zer-case:  $x1 \geq ?zer-list ! 0$  using x-asm-var by auto
have all-n:  $\bigwedge n. (n < (length ?zer-list - 1) \longrightarrow x1 \geq ?zer-list ! n)$ 
proof -
  fix n
  show n-lt:  $(n < (length ?zer-list - 1) \longrightarrow x1 \geq ?zer-list ! n)$ 
  proof (induct n)
    case 0
    then show ?case using zer-case
      by blast
    next
    case (Suc n)
    then show ?case
      using assump
      using Suc-eq-plus1 Suc-lessD by presburger
  qed
qed
have  $(length ?zer-list - 2) \leq length ?zer-list - 1$ 
  using diff-le-mono2 one-le-numeral by blast
have  $x1 \geq ?zer-list ! (length ?zer-list - 1)$ 
proof -
  have h1:  $length ?zer-list = 1 \longrightarrow x1 \geq ?zer-list ! (length ?zer-list - 1)$ 
    using assump zer-case by auto
  have h2:  $length ?zer-list > 1 \longrightarrow x1 \geq ?zer-list ! (length ?zer-list - 1)$ 
    using all-n assump apply (auto)
    by (metis (mono-tags, lifting) Suc-diff-Suc lessI)
  then show ?thesis using h1 h2 apply (auto)
    using zer-case by blast
qed
then show False using all-n x-asm-var
  by linarith
qed
then show ?thesis
  using x1-asm nonempty non-mem by auto
  (metis (lifting) Suc-lessD add.right-neutral add-Suc-right length-sorted-list-of-set less-diff-conv less-le not-less nth-mem)
qed
then have h1:  $(?zer-list \neq [] \wedge (x1 \geq (?zer-list ! 0)) \wedge (x1 \leq (?zer-list ! (length ?zer-list - 1)))) \implies$ 
 $(\exists n < (length ?zer-list - 1). x1 > (?zer-list ! n) \wedge x1 < (?zer-list ! (n+1)))$ 
by blast
then show ?thesis

```

```

    using zer-list-prop not-less by auto
  qed
qed

lemma roots-of-coprime-r-capture-sgas-without-zeros:
  fixes qs:: rat poly list
  fixes sga:: rat list
  assumes pairwise-rel-prime: pairwise-coprime-list qs
  assumes all-squarefree:  $\bigwedge q. q \in \text{set } qs \implies \text{rsquarefree } q$ 
  assumes ex-x1:  $\text{sga} = \text{sign-vec } qs \ x1$ 
  assumes hnz: has-no-zeros sga
  shows  $(\exists w \in (\text{roots-of-coprime-r } (\text{cast-rat-list } qs)). \text{sga} = (\text{sign-vec } qs \ w))$ 
proof -
  obtain x1 where x1-prop:  $\text{sga} = (\text{sign-vec } qs \ x1)$  using ex-x1 by auto
  let ?zer-list = sorted-list-of-set  $\{(x::\text{real}). (\exists q \in \text{set}(qs). (\text{rpoly } q \ x = 0))\} :: \text{real list}$ 
  have strict-sorted-h: sorted-wrt ( $<$ ) ?zer-list using sorted-sorted-list-of-set
    strict-sorted-iff by auto
  then have sorted-hyp:  $\forall q < \text{length } ?zer\text{-list}. (q + 1 < \text{length } ?zer\text{-list}) \longrightarrow$ 
     $(?zer\text{-list } ! \ q) < (?zer\text{-list } ! \ (q + 1))$ 
  using strict-sorted-h by (auto simp: sorted-wrt-iff-nth-less)
  then have sorted-hyp-var:  $\forall q1 < \text{length } ?zer\text{-list}. \forall q2 < \text{length } ?zer\text{-list}. (q1 <$ 
 $q2 \longrightarrow$ 
     $(?zer\text{-list } ! \ q1) < (?zer\text{-list } ! \ q2))$ 
  using sorted-wrt-iff-nth-less strict-sorted-h by blast
  then have sorted-hyp-var2:  $\forall q1 < \text{length } ?zer\text{-list}. ((?zer\text{-list } ! \ q1)::\text{real}) \leq$ 
 $(?zer\text{-list } ! \ (\text{length } ?zer\text{-list} - 1))$ 
  by (smt (verit, ccfv-SIG) One-nat-def Suc-pred bot-nat-0.extremum less-Suc-eq-le
    less-le not-less)
  have nonz-q:  $\forall q \in \text{set } qs. q \neq 0$ 
  using all-squarefree unfolding rsquarefree-def by auto
  then have  $\forall q \in \text{set } qs. \text{finite } \{x. \text{rpoly } q \ x = 0\}$ 
  by (simp add: poly-roots-finite)
  then have fin-set:  $\text{finite } \{x. \exists q \in \text{set } qs. \text{rpoly } q \ x = 0\}$ 
  by auto
  have x1-and-roots:  $?zer\text{-list} \neq [] \longrightarrow ((x1 < (?zer\text{-list } ! \ 0)) \vee (x1 > (?zer\text{-list } !$ 
 $(\text{length } ?zer\text{-list} - 1))) \vee$ 
 $(\exists n < (\text{length } ?zer\text{-list} - 1). x1 > (?zer\text{-list } ! \ n) \wedge x1 < (?zer\text{-list } ! \ (n+1))))$ 
  using roots-of-coprime-r-location-property x1-prop assms by auto
  have x2gt:  $\forall x2 > x1. \text{sign-vec } qs \ x1 \neq \text{sign-vec } qs \ x2 \longrightarrow (\exists r > x1. r \leq x2 \wedge$ 
 $(\exists q \in \text{set } qs. \text{rpoly } q \ r = 0))$ 
  using hnz x1-prop zero-above[of qs x1] by auto
  have x2lt:  $\forall x2 < x1. \text{sign-vec } qs \ x1 \neq \text{sign-vec } qs \ x2 \longrightarrow (\exists r < x1. x2 \leq r \wedge$ 
 $(\exists q \in \text{set } qs. \text{rpoly } q \ r = 0))$ 
  using hnz x1-prop zero-below[of qs x1] by (auto)
  have triv-neg-inf-h:  $?zer\text{-list} = [] \implies \text{sga} = (\text{sign-vec } qs \ (-\text{crb } (\text{prod-list } (\text{cast-rat-list } qs))))$ 

```

```

proof –
  assume empty-zer: (?zer-list:: real list) = []
  let ?zer-set = {x. ∃ q∈set qs. rpoly q x = 0}:: real set
  have fin-zer: finite ?zer-set using fin-set by auto
  have finite ?zer-set  $\implies$  (sorted-list-of-set ?zer-set = []) = (?zer-set = {})
    using fin-zer sorted-list-of-set-eq-Nil-iff[where A = ?zer-set] by auto
  then have (sorted-list-of-set ?zer-set = []) = (?zer-set = {})
    using fin-zer by auto
  then have nozers: ?zer-set = {}
    using empty-zer by auto
  then have  $\neg(\exists(r::real). (\exists(q::rat poly)\in set qs. rpoly q r = 0))$ 
    using nozers by auto
  then have  $\forall y. sign-vec qs x1 = sign-vec qs y$ 
proof –
  fix y
  have gt-prop:  $x1 > y \implies sign-vec qs x1 = sign-vec qs y$ 
    using hnz x1-prop zero-below[of qs x1] apply (auto)
    using  $\langle \#r. \exists q\in set qs. rpoly q r = 0 \rangle$  by blast
  have lt-prop:  $x1 < y \implies sign-vec qs x1 = sign-vec qs y$ 
    using zero-above[of qs x1] apply (auto)
    using  $\langle \#r. \exists q\in set qs. rpoly q r = 0 \rangle$  by blast
  show ?thesis using gt-prop lt-prop apply (auto)
  apply (metis  $\langle \#r. \exists q\in set qs. rpoly q r = 0 \rangle$  linorder-neqE-linordered-idom
x2gt x2lt)
    using x2gt x2lt apply (auto)
  apply (metis  $\langle \#r. \exists q\in set qs. rpoly q r = 0 \rangle$  linorder-neqE-linordered-idom)
  apply (metis  $\langle \#r. \exists q\in set qs. rpoly q r = 0 \rangle$  linorder-neqE-linordered-idom)
  by (metis  $\langle \#r. \exists q\in set qs. rpoly q r = 0 \rangle$  linorder-neqE-linordered-idom)
qed
then show ?thesis
  by (simp add: x1-prop)
qed
have neg-inf-h: ?zer-list  $\neq [] \implies (x1 < (?zer-list ! 0) \implies sga = (sign-vec qs$ 
(-crb (prod-list (cast-rat-list qs)))))
proof –
  let ?neg-crb = -crb (prod-list (cast-rat-list qs))
  assume len-nontriv: ?zer-list  $\neq []$ 
  assume x1-lt:  $x1 < ?zer-list ! 0$ 
  have r-gt:  $\forall r. (\exists q\in set qs. rpoly q r = 0) \implies r \geq (?zer-list ! 0)$ 
proof clarsimp
  fix q::rat poly
  fix r:: real
  assume q-in:  $q \in set qs$ 
  assume rpoly q r = 0
  then have  $r \in \{x. \exists q\in set qs. rpoly q x = 0\}$  using q-in by auto
  then have List.member (sorted-list-of-set {x. ∃ q∈set qs. rpoly q x = 0}) r
    by (auto simp add: fin-set)
  then show sorted-list-of-set {x. ∃ q∈set qs. rpoly q x = 0} ! 0 ≤ r
    using sorted-hyp-var by auto

```

(metis (no-types, lifting) in-set-conv-nth less-le not-less not-less-zero sorted-hyp-var)

qed

have *prod-zero*: $\forall x. (\exists q \in \text{set } qs. \text{rpoly } q \ x = 0) \longrightarrow (\text{poly } (\text{prod-list } (\text{cast-rat-list } qs)) \ x) = 0$

using *prod-list-zero-iff* [**where** *xs* = (cast-rat-list qs)]

by (metis cast-rat-list-def image-eqI list.set-map poly-prod-list-zero-iff)

have *?zer-list* $\neq [] \longrightarrow \text{List.member } (\text{sorted-list-of-set } \{x. \exists q \in \text{set } qs. \text{rpoly } q \ x = 0\}) \ (\text{?zer-list } ! 0)$

using *nth-Cons-0* **apply** (auto)

by (meson grOI length-0-conv nth-mem)

then have *?zer-list* $\neq [] \longrightarrow (\text{?zer-list } ! 0) \in \{x. \exists q \in \text{set } qs. \text{rpoly } q \ x = 0\}$

by (auto simp add: fin-set)

then have *?zer-list* $\neq [] \longrightarrow (\exists q \in \text{set } qs. \text{rpoly } q \ (\text{?zer-list } ! 0) = 0)$

by blast

then have *poly-zero*: *?zer-list* $\neq [] \longrightarrow (\text{poly } (\text{prod-list } (\text{cast-rat-list } qs)) \ (\text{?zer-list } ! 0)) = 0$

using *prod-zero* **by** auto

have $\forall q. \text{List.member } (\text{cast-rat-list } qs) \ q \longrightarrow q \neq 0$ **using** *nonz-q*

by (auto simp add: cast-rat-list-def)

then have (prod-list (cast-rat-list qs)) $\neq 0$

using *prod-list-zero-iff* **by** fastforce

then have *crb-lt*: *?zer-list* $\neq [] \longrightarrow \text{?neg-crb} < \text{?zer-list } ! 0$

using *crb-lem-neg* [**where** *p* = (prod-list (cast-rat-list qs)), **where** *x* = sorted-list-of-set {x. $\exists q \in \text{set } qs. \text{rpoly } q \ x = 0$ } ! 0] **apply** (auto)

using *poly-zero*

by blast

have *crb-gt-x1*: *?zer-list* $\neq [] \longrightarrow (\text{?neg-crb} > x1 \longrightarrow (\text{sga } \neq (\text{sign-vec } qs \ \text{?neg-crb})) \longrightarrow (\exists r > x1. r \leq \text{?neg-crb} \wedge (\exists q \in \text{set } qs. \text{rpoly } q \ r = 0)))$

using *x2gt crb-lt r-gt x1-prop* **by** fastforce

have *crb-lt-x1*: *?neg-crb* $< x1 \longrightarrow (\text{sga } \neq (\text{sign-vec } qs \ \text{?neg-crb})) \longrightarrow (\exists r < x1. \text{?neg-crb} \leq r \wedge (\exists q \in \text{set } qs. \text{rpoly } q \ r = 0))$

using *x2lt x1-lt r-gt x1-prop* **by** fastforce

show *?thesis* **using** *len-nontriv crb-gt-x1 crb-lt-x1 x1-lt crb-lt r-gt*

using *x1-prop* **by** auto

qed

have *pos-inf-h*: *?zer-list* $\neq [] \implies (x1 > (\text{?zer-list } ! (\text{length } \text{?zer-list} - 1)) \implies \text{sga} = (\text{sign-vec } qs \ (\text{crb } (\text{prod-list } (\text{cast-rat-list } qs))))$

proof –

let *?pos-crb* = *crb* (prod-list (cast-rat-list qs))

assume *len-nontriv*: *?zer-list* $\neq []$

assume *x1-lt*: *x1* $> \text{?zer-list } ! (\text{length } \text{?zer-list} - 1)$

have *r-gt*: $\bigwedge r. (\exists q \in \text{set } qs. \text{rpoly } q \ r = 0) \implies r \leq (\text{?zer-list } ! (\text{length } \text{?zer-list} - 1))$

proof –

fix *r*:: real

assume *q-in*: $(\exists q \in \text{set } qs. \text{rpoly } q \ r = 0)$

then obtain *q*::rat *poly* **where** *q-prop*: $q \in \text{set } qs \wedge \text{rpoly } q \ r = 0$ **by** auto

```

then have  $r \in \{x. \exists q \in \text{set } qs. \text{rpoly } q \ x = 0\}$  using q-in by auto
then have  $\text{List.member } (\text{sorted-list-of-set } \{x. \exists q \in \text{set } qs. \text{rpoly } q \ x = 0\}) \ r$ 
  by (auto simp add: fin-set)
then have  $\exists n < (\text{length } ?\text{zer-list}). \ r = ?\text{zer-list } ! \ n$ 
  by (auto simp add: in-set-conv-nth)
then obtain  $n$  where  $n\text{-prop}: \ n < \text{length } ?\text{zer-list} \wedge \ r = ?\text{zer-list } ! \ n$ 
  by auto
then show  $r \leq (?\text{zer-list } ! \ (\text{length } ?\text{zer-list} - 1))$ 
proof –
  have  $\forall q1. \ q1 < \text{length } ?\text{zer-list} \longrightarrow (?\text{zer-list } ! \ q1) \leq (?\text{zer-list } ! \ (\text{length } ?\text{zer-list} - 1:: \text{nat}))$ 
    using sorted-hyp-var2 by auto
  then have  $(?\text{zer-list } ! \ n) \leq (?\text{zer-list } ! \ (\text{length } ?\text{zer-list} - 1))$ 
    using n-prop by auto
  then show ?thesis using n-prop by auto
qed
qed
have  $\text{prod-zer}: \forall x. \ (\exists q \in \text{set } qs. \ \text{rpoly } q \ x = 0) \longrightarrow (\text{poly } (\text{prod-list } (\text{cast-rat-list } qs)) \ x) = 0$ 
  using prod-list-zero-iff [where  $xs = (\text{cast-rat-list } qs)$ ] unfolding cast-rat-list-def
  apply (auto)
  using nonz-q apply blast
  by (metis (no-types, opaque-lifting) image-eqI list.set-map of-rat-poly-hom.prod-list-map-hom poly-prod-list-zero-iff)
  have  $?\text{zer-list} \neq [] \longrightarrow \text{List.member } (\text{sorted-list-of-set } \{x. \exists q \in \text{set } qs. \ \text{rpoly } q \ x = 0\})$ 
     $(?\text{zer-list } ! \ (\text{length } ?\text{zer-list} - 1))$ 
    using nth-Cons-0 apply (auto)
    by (metis (no-types, lifting) diff-less in-set-conv-nth length-greater-0-conv length-sorted-list-of-set zero-less-Suc)
  then have  $?\text{zer-list} \neq [] \longrightarrow (?\text{zer-list } ! \ (\text{length } ?\text{zer-list} - 1))$ 
     $\in \{x. \exists q \in \text{set } qs. \ \text{rpoly } q \ x = 0\}$ 
    by auto (simp add: fin-set)
  then have  $?\text{zer-list} \neq [] \longrightarrow (\exists q \in \text{set } qs. \ \text{rpoly } q \ (?\text{zer-list } ! \ (\text{length } ?\text{zer-list} - 1)) = 0)$ 
    by blast
  then have  $\text{poly-zer}: \ ?\text{zer-list} \neq [] \longrightarrow (\text{poly } (\text{prod-list } (\text{cast-rat-list } qs)) \ (?\text{zer-list } ! \ (\text{length } ?\text{zer-list} - 1))) = 0$ 
    using prod-zer by auto
  have  $\forall q. \ \text{List.member } (\text{cast-rat-list } qs) \ q \longrightarrow q \neq 0$  using nonz-q
    by (auto simp add: cast-rat-list-def)
  then have  $(\text{prod-list } (\text{cast-rat-list } qs)) \neq 0$ 
    using prod-list-zero-iff by auto
  then have  $\text{crb-lt}: \ ?\text{zer-list} \neq [] \longrightarrow ?\text{pos-crb} > ?\text{zer-list } ! \ (\text{length } ?\text{zer-list} - 1)$ 
    using crb-lem-pos [where  $p = (\text{prod-list } (\text{cast-rat-list } qs))$ ], where  $x = \text{sorted-list-of-set } \{x. \exists q \in \text{set } qs. \ \text{rpoly } q \ x = 0\} \ ! \ (\text{length } ?\text{zer-list} - 1)$  apply (auto)
    using poly-zer
    by simp

```

```

have crb-gt-x1: ?zer-list ≠ [] → ((?pos-crb::real) > (x1::real) → (sga ≠
(sign-vec (qs::rat poly list) (?pos-crb::real))) → (∃ (r::real) < x1. r ≥ ?pos-crb ∧
(∃ (q::rat poly) ∈ set qs. r poly q r = 0)))
using x2gt apply auto
using crb-lt r-gt x1-prop
using x1-lt by fastforce
have crb-lt-x1: ?pos-crb < x1 → (sga ≠ (sign-vec qs ?pos-crb)) → (∃ r > x1.
?pos-crb ≥ r ∧ (∃ q ∈ set qs. poly (real-of-rat-poly q) r = 0))
using x2lt apply auto
using x1-lt r-gt x1-prop
by (meson ⟨prod-list (cast-rat-list qs) ≠ 0⟩ crb-lem-pos not-less prod-zer)
show ?thesis using len-nontriv crb-gt-x1 crb-lt-x1 x1-lt crb-lt r-gt apply auto
using x1-prop
by blast
qed
have between-h: (∃ n < (length ?zer-list - 1). x1 > (?zer-list ! n) ∧ x1 <
(?zer-list ! (n+1))) ⇒ (∃ w ∈ (roots-of-coprime-r (cast-rat-list qs)). sga = (sign-vec
qs w))
proof -
assume (∃ n < (length ?zer-list - 1). x1 > (?zer-list ! n) ∧ x1 < (?zer-list !
(n+1)))
then obtain n where n-prop: n < (length ?zer-list - 1) ∧ x1 > (?zer-list !
n) ∧ x1 < (?zer-list ! (n+1))
by auto
have ∀ q1 q2. (q1 ≠ q2 ∧ (List.member (cast-rat-list qs) q1) ∧ (List.member
(cast-rat-list qs) q2)) → coprime q1 q2
using pairwise-rel-prime
apply (auto simp add: pairwise-coprime-list-def cast-rat-list-def coprime-rat-poly-iff-coprimereal-poly
in-set-conv-nth)
using coprime-rat-poly-real-poly
apply auto
done
then have all-prop: ∀ x1. ∀ x2. ((x1 < x2 ∧ (∃ q1 ∈ set (cast-rat-list(qs)). (poly
q1 x1) = 0) ∧ (∃ q2 ∈ set((cast-rat-list(qs))). (poly q2 x2) = 0)) → (∃ q. x1 < q
∧ q < x2 ∧ poly (coprime-r (cast-rat-list qs) q) = 0))
using coprime-r-roots-prop
by auto
have exq1: (∃ q1 ∈ set (cast-rat-list(qs)). (poly q1 (?zer-list ! n)) = 0)
unfolding cast-rat-list-def using n-prop apply auto
by (smt (verit, ccfv-SIG) One-nat-def Suc-eq-plus1 Suc-lessD fin-set length-sorted-list-of-set
less-diff-conv mem-Collect-eq nth-mem set-sorted-list-of-set)
have exq2: (∃ q2 ∈ set (cast-rat-list(qs)). (poly q2 (?zer-list ! (n+1))) = 0)
unfolding cast-rat-list-def using n-prop One-nat-def Suc-eq-plus1 fin-set
less-diff-conv mem-Collect-eq nth-mem set-sorted-list-of-set
by (smt (verit, ccfv-SIG) image-eqI set-map)
have n-prop2: (((?zer-list ! n) < (?zer-list ! (n+1)) ∧ (∃ q1 ∈ set (cast-rat-list(qs)).
(poly q1 (?zer-list ! n)) = 0) ∧ (∃ q2 ∈ set((cast-rat-list(qs))). (poly q2 (?zer-list !
(n+1)))) = 0)))
using exq1 exq2 sorted-hyp n-prop by auto

```

then have $(\exists q. (?zer-list ! n) < q \wedge q < (?zer-list ! (n+1)) \wedge poly (coprime-r (cast-rat-list qs)) q = 0)$
using *all-prop n-prop2* **by** *auto*
then have $\exists w \in (roots-of-coprime-r (cast-rat-list qs)). (?zer-list ! n) < w \wedge w < (?zer-list ! (n+1))$
apply (*auto*)
using *roots-of-coprime-r-def* **by** *auto*
then obtain w **where** $w-prop: w \in (roots-of-coprime-r (cast-rat-list qs)) \wedge (?zer-list ! n) < w \wedge w < (?zer-list ! (n+1))$ **by** *auto*
have $n-lt1: n < length ?zer-list$ **using** $n-prop$
using *add-lessD1 less-diff-conv* **by** *blast*
have $n-lt2: n + 1 < length ?zer-list$ **using** $n-prop$
using *less-diff-conv* **by** *blast*
have $sorted-hyp-var3: ?zer-list ! n < ?zer-list ! (n + 1)$ **using** *sorted-hyp n-lt1 n-lt2* **by** *auto*
then have $helper: w > x1 \longrightarrow \neg(\exists(x::real). (List.member ?zer-list x \wedge x1 \leq x \wedge x \leq w))$
using $n-prop w-prop x1-prop strict-sorted-h sorted-list-lemma$ **where** $n = n,$
where $l = ?zer-list,$ **where** $a = x1,$ **where** $b = w$ *sorted-hyp-var3*
by *auto*
have $no-root1: w > x1 \implies \neg(\exists r > x1. r \leq w \wedge (\exists q \in set qs. rpoly q r = 0))$
proof –
assume $w > x1$
then have $nex: \neg(\exists(x::real). (List.member ?zer-list x \wedge x1 \leq x \wedge x \leq w))$
using *helper* **by** *auto*
have $(\exists r > x1. r \leq w \wedge (\exists q \in set qs. rpoly q r = 0)) \implies False$
proof –
assume $(\exists r > x1. r \leq w \wedge (\exists q \in set qs. rpoly q r = 0))$
then obtain r **where** $r-prop: r > x1 \wedge r \leq w \wedge (\exists q \in set qs. rpoly q r = 0)$
by *auto*
then have $List.member ?zer-list r \wedge x1 \leq r \wedge x1 \leq w$
by *simp (simp add: fin-set)*
then show *?thesis* **using** $nex r-prop$
by *blast*
qed
then show *?thesis* **by** *auto*
qed
have $helper2: w < x1 \longrightarrow \neg(\exists(x::real). (List.member ?zer-list x \wedge w \leq x \wedge x \leq x1))$
using $n-lt2 n-prop sorted-list-lemma strict-sorted-h w-prop$ **by** *blast*
have $no-root2: w < x1 \implies \neg(\exists r < x1. w \leq r \wedge (\exists q \in set qs. rpoly q r = 0))$
proof –
assume $w < x1$
then have $nex: \neg(\exists(x::real). (List.member ?zer-list x \wedge w \leq x \wedge x \leq x1))$
using *helper2* **by** *auto*
have $(\exists r < x1. w \leq r \wedge (\exists q \in set qs. rpoly q r = 0)) \implies False$
proof –
assume $(\exists r < x1. w \leq r \wedge (\exists q \in set qs. rpoly q r = 0))$
then obtain r **where** $r-prop: r < x1 \wedge w \leq r \wedge (\exists q \in set qs. rpoly q r = 0)$

```

by auto
  then have List.member ?zer-list r  $\wedge$  w  $\leq$  r  $\wedge$  r  $\leq$  x1
    by simp (simp add: fin-set r-prop)

  then show ?thesis using nex r-prop
    by blast
qed
then show ?thesis by auto
qed
then have w-gt: w > x1  $\longrightarrow$  (sign-vec qs x1) = (sign-vec qs w)
  using no-root1 n-prop x2gt by auto
have w-lt: w < x1  $\longrightarrow$  (sign-vec qs x1) = (sign-vec qs w)
  using no-root2 n-prop x2lt by auto
then have sga = (sign-vec qs w) using w-gt w-lt x1-prop by auto
then show ( $\exists$  w  $\in$  (roots-of-coprime-r (cast-rat-list qs)). sga = (sign-vec qs w))
  using w-prop by auto
qed
show ?thesis using triv-neg-inf-h neg-inf-h pos-inf-h between-h x1-and-roots
  by (metis (mono-tags, lifting) coprime-r-zero1 coprime-r-zero2 mem-Collect-eq
roots-of-coprime-r-def)
qed

```

lemma find_csas-lemma-nozeros:

```

fixes qs:: rat poly list
assumes fs: factorize-polys qs = (fs,data)
assumes fs  $\neq$  []
shows (csa  $\in$  (consistent-sign-vectors fs UNIV)  $\wedge$  has-no-zeros csa)  $\longleftrightarrow$ 
  csa  $\in$  set (find-consistent-signs-at-roots (coprime-r (cast-rat-list fs)) (cast-rat-list
fs))
proof -
  let ?new-l = cast-rat-list fs
  let ?copr = coprime-r ?new-l
  have copr-nonz: ?copr  $\neq$  0
    using coprime-r-nonzero[OF assms(1-2)] unfolding cast-rat-list-def by auto
  have nontriv-list: 0 < length ?new-l
    using assms cast-rat-list-def by (auto)
  have pairwise-cp: ( $\bigwedge$  q. q  $\in$  set ?new-l  $\implies$ 
    algebraic-semidom-class.coprime ?copr q) using coprime-r-coprime-prop[OF
assms(1)]
    apply (auto)
  by (metis cast-rat-list-def comm-monoid-mult-class.coprime-commute coprime-iff-coprime
list.set-map)
  have set-fsga: set(find-consistent-signs-at-roots ?copr ?new-l) = set(characterize-consistent-signs-at-roots
?copr ?new-l)
    using find-consistent-signs-at-roots[OF copr-nonz pairwise-cp] by auto
  then have csa-in-hyp: csa  $\in$  set (find-consistent-signs-at-roots ?copr ?new-l)
 $\longleftrightarrow$  csa  $\in$  set(characterize-consistent-signs-at-roots ?copr ?new-l) by auto
  have forward: (csa  $\in$  (consistent-sign-vectors fs UNIV)  $\wedge$  has-no-zeros csa)

```

```

    => csa ∈ set(characterize-consistent-signs-at-roots ?copr ?new-l)
  proof -
    assume csa-in: (csa ∈ (consistent-sign-vectors fs UNIV) ∧ has-no-zeros csa)
    have fin: finite {x. poly (coprime-r (cast-rat-list fs)) x = 0}
      using copr-nonz poly-roots-finite
      by (simp add: poly-roots-finite fs)
    have pcl: pairwise-coprime-list fs
      using coprime-factorize fs
      by (metis fst-conv)
    have sqf:  $\bigwedge q. q \in \text{set } fs \implies \text{rsquarefree } q$ 
      using factorize-polys-square-free[OF assms(1)]
      by (metis square-free-rsquarefree)
    obtain x1 where x1: csa = sign-vec fs x1
      using consistent-sign-vectors-def csa-in by auto
    have hnz: has-no-zeros csa using csa-in by auto
    obtain w where w-prop:  $w \in \text{roots-of-coprime-r (cast-rat-list fs) csa} = \text{sign-vec}$ 
      fs w
      using roots-of-coprime-r-capture-sgas-without-zeros[OF pcl sqf x1 hnz]
      by auto
    have w-root: poly (coprime-r (cast-rat-list fs)) w = 0
      using w-prop
      by (simp add: roots-of-coprime-r-def)
    then have w ∈ {x. poly (coprime-r (cast-rat-list fs)) x = 0}
      by auto
    then have w-ins:  $w \in \text{set (characterize-root-list-p (coprime-r (cast-rat-list fs)))}$ 
      using fin set-sorted-list-of-set[where A={x. poly (coprime-r (cast-rat-list fs))
x = 0}]
      unfolding characterize-root-list-p-def
      by auto
    have map (( $\lambda x. \text{if } 0 < x \text{ then } 1 \text{ else if } x < 0 \text{ then } -1 \text{ else } 0$ )  $\circ$  ( $\lambda p. \text{rpoly } p$ 
w)) fs =
      map (( $\lambda x. \text{if } 0 < x \text{ then } 1 \text{ else } -1$ )  $\circ$  ( $\lambda p. \text{rpoly } p$  w)) fs
  proof -
    have  $\neg(\exists x \in \text{set } fs. \text{rpoly } x \text{ w} = 0)$ 
  proof clarsimp
    fix x
    assume x-in:  $x \in \text{set } fs$ 
    assume x-zer:  $\text{rpoly } x \text{ w} = 0$ 
    then have  $\exists k < \text{length } fs. \text{nth } fs \ k = x$ 
      using x-in
      by (simp add: in-set-conv-nth)
    then obtain k where k-prop:  $k < \text{length } fs \wedge fs \ ! \ k = x$ 
      by auto
    then have  $(\text{sign-vec } fs \ w) \ ! \ k = 0$  using x-zer apply (auto)
      unfolding sign-vec-def squash-def by auto
    then have  $\neg(\text{has-no-zeros } (\text{sign-vec } fs \ w))$ 
      apply (auto)
      by (simp add: hnz-prop k-prop)
    then show False using hnz unfolding sign-vec-def squash-def

```

```

    using (¬ has-no-zeros (sign-vec fs w) w-prop(2)) by blast
  qed
  then show ?thesis using hnz unfolding sign-vec-def squash-def by auto
  qed
  then have map ((λx. if 0 < x then 1 else if x < 0 then - 1 else 0) ∘ (λp.
rpolynomial p w)) fs =
  map (λq. if 0 < polynomial q w then 1 else - 1)
  (cast-rat-list fs)
  unfolding cast-rat-list-def by auto
  then have csa = map (λq. if 0 < polynomial q w then 1 else - 1)
  (cast-rat-list fs)
  by (simp add: comp-def sign-vec-def squash-def w-prop(2))
  then show ?thesis unfolding characterize-consistent-signs-at-roots-def
  apply (auto) unfolding signs-at-def using w-ins w-prop
  using consistent-sign-vectors-consistent-sign-vectors-r consistent-sign-vectors-def
consistent-sign-vectors-r-def signs-at-def by force
  qed
  have backward: csa ∈ set(characterize-consistent-signs-at-roots ?copr ?new-l) ⇒
  (csa ∈ (consistent-sign-vectors fs UNIV) ∧ has-no-zeros csa)
  proof -
    assume csa-in: csa ∈ set(characterize-consistent-signs-at-roots ?copr ?new-l)
    have csa-in-old-set: csa ∈ set (characterize-consistent-signs-at-roots-copr ?copr
?new-l)
    using csa-list-copr-rel copr-nonz csa-in find-consistent-signs-at-roots-copr pair-
wise-cp set-fsga by auto
    have ∀ (x::real). ∀ (l::real polynomial list). rec-list True (λh T. If (h = 0) False)
  (map (λq. if 0 < polynomial q x then (1::rat) else (-1::rat))
  l)
  proof clarsimp
    fix x::real
    fix l::real polynomial list
    show rec-list True (λh T. If (h = 0) False)
  (map (λq. if 0 < polynomial q x then (1::rat) else (-1::rat)) l)
  proof (induct l)
    case Nil
    then show ?case
    by simp
  next
    case (Cons a l)
    then show ?case by auto
  qed
  qed
  then have ∀ x. rec-list True (λh T. If (h = 0) False)
  (map (λq. if 0 < polynomial q x then (1::rat) else - 1)
  (cast-rat-list fs))
  by auto
  then have hnz: has-no-zeros csa
  using csa-in-old-set
  unfolding characterize-consistent-signs-at-roots-copr-def consistent-sign-vec-copr-def

```

```

apply (auto) unfolding has-no-zeros-def
by auto
have  $\exists w \in \text{set}(\text{characterize-root-list-p } ?\text{copr})$ . csa = consistent-sign-vec-copr
?new-l w
using csa-in-old-set using characterize-consistent-signs-at-roots-copr-def by
auto
then obtain w where w-prop:  $w \in \text{set}(\text{characterize-root-list-p } ?\text{copr}) \wedge \text{csa}$ 
= consistent-sign-vec-copr ?new-l w
by auto
then have poly ?copr w = 0 unfolding characterize-root-list-p-def
by (simp add: copr-nonz poly-roots-finite)
then have copr-prop:  $\forall p \in \text{set}(?\text{new-l})$ . poly p w  $\neq 0$ 
using w-prop coprime-r-coprime-prop apply (auto)
by (meson coprime-poly-0 pairwise-cp)
then have consistent-sign-vec-copr ?new-l w = sign-vec fs w
unfolding sign-vec-def squash-def consistent-sign-vec-copr-def
cast-rat-list-def by auto
then show ?thesis using hnz w-prop apply (auto)
using consistent-sign-vectors-def by blast
qed
have (csa  $\in$  (consistent-sign-vectors fs UNIV)  $\wedge$  has-no-zeros csa)
 $\longleftrightarrow$  csa  $\in$  set(characterize-consistent-signs-at-roots ?copr ?new-l)
using forward backward by blast
then show ?thesis using csa-in-hyp by auto
qed

```

lemma *range-eq*:

```

fixes a
fixes b
shows  $a = b \longrightarrow \text{range } a = \text{range } b$ 
by simp

```

lemma *removeAt-distinct*:

```

shows distinct fss  $\implies$  distinct (removeAt i fss)
unfolding removeAt-def
by (simp add: set-take-disj-set-drop-if-distinct)

```

lemma *consistent-signs-atw*:

```

assumes  $\bigwedge p. p \in \text{set } fs \implies \text{poly } p \ x \neq 0$ 
shows consistent-sign-vec-copr fs x = signs-at fs x
unfolding consistent-sign-vec-copr-def signs-at-def squash-def o-def
by (simp add: assms)

```

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 set(characterize-consistent-signs-at-roots p fs) =

```

$consistent\text{-}sign\text{-}vectors\text{-}r\ fs\ (\{x.\ poly\ p\ x = 0\})$
by (*simp* *add: assms(1) characterize-consistent-signs-at-roots-def consistent-sign-vectors-r-def poly-roots-finite characterize-root-list-p-def*)

lemma *signs-at-insert*:
assumes $i \leq length\ qs$
shows $insertAt\ i\ (squash\ (poly\ p\ x))\ (signs\text{-}at\ qs\ x) =$
 $signs\text{-}at\ (insertAt\ i\ p\ qs)\ x$
unfolding *insertAt-def signs-at-def o-def* **using** *assms take-map apply auto*
apply (*subst drop-map*) **by** *simp*

lemma *length-removeAt*:
assumes $i < length\ ls$
shows $length\ (removeAt\ i\ ls) = length\ ls - 1$
unfolding *removeAt-def* **using** *assms* **by** *auto*

lemma *insertAt-removeAt*:
assumes $i < length\ ls$
shows $insertAt\ i\ (ls!i)\ (removeAt\ i\ ls) = ls$
unfolding *insertAt-def removeAt-def* **using** *assms* **apply** *auto*
by (*simp add: Cons-nth-drop-Suc*)

lemma *insertAt-nth*:
assumes $i \leq length\ ls$
shows $insertAt\ i\ n\ ls ! i = n$
unfolding *insertAt-def* **using** *assms*
by (*simp add: nth-append-take*)

lemma *length-signs-at[simp]*:
shows $length\ (signs\text{-}at\ qs\ x) = length\ qs$
unfolding *signs-at-def* **by** *auto*

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

proof –
from *removeAt-distinct[OF assms(4)]*
have *neq: fss ! i ≠ 0* **using** *assms* **by** *simp*
have *cop: $\bigwedge q.\ q \in set\ (removeAt\ i\ fss) \implies coprime\ (fss ! i)\ q$* **using** *assms*
unfolding *removeAt-def*
apply *auto*
apply (*metis distinct-append in-set-takeD list-update-id not-distinct-conv-prefix nth-mem upd-conv-take-nth-drop*)

by (*metis Cons-nth-drop-Suc distinct.simps(2) distinct-drop in-set-dropD nth-mem*)
from *find-consistent-signs-at-roots[OF neq]*
have *set (find-consistent-signs-at-roots (fss ! i) (removeAt i fss)) = set (characterize-consistent-signs-at-roots (fss ! i) (removeAt i fss))*
using *cop by auto*
then have *eq: set (map (insertAt i 0) (find-consistent-signs-at-roots (fss ! i) (removeAt i fss))) = insertAt i 0 ' set (characterize-consistent-signs-at-roots (fss ! i) (removeAt i fss))*
by *simp*
from *characterize-consistent-signs-at-roots-rw[OF neq cop]*
have *eq2: set (characterize-consistent-signs-at-roots (fss ! i) (removeAt i fss)) = consistent-sign-vectors-r (removeAt i fss) {x. poly (fss ! i) x = 0} .*
have *ile: i ≤ length (removeAt i fss)*
using *length-removeAt[OF assms(1)] assms(1) by linarith*
have *1: ∧xb. poly (fss ! i) xb = 0 ⇒ insertAt i 0 (signs-at (removeAt i fss) xb) ∈ range (signs-at fss)*
proof –
fix *x*
assume *poly (fss ! i) x = 0*
then have *insertAt i 0 (signs-at (removeAt i fss) x) = insertAt i (squash (poly (fss ! i) x)) (signs-at (removeAt i fss) x)* **by** (*auto simp add: squash-def*)
also have *... = signs-at (insertAt i (fss ! i) (removeAt i fss)) x*
apply (*intro signs-at-insert*)
using *length-removeAt[OF assms(1)]*
using *assms(1) by linarith*
also have *... = signs-at fss x* **unfolding** *insertAt-removeAt[OF assms(1)]* **by** *auto*
ultimately have **:insertAt i 0 (signs-at (removeAt i fss) x) = signs-at fss x*
by *auto*
thus *insertAt i 0 (signs-at (removeAt i fss) x) ∈ range (signs-at fss)* **by** *auto*
qed
have *2: ∧xa. signs-at fss xa ! i = 0 ⇒ i ≤ length (removeAt i fss) ⇒ signs-at fss xa ∈ insertAt i 0 ' signs-at (removeAt i fss) ' {x. poly (fss ! i) x = 0}*
proof –
fix *x*
assume *signs-at fss x ! i = 0*
then have *z:poly (fss ! i) x = 0* **unfolding** *signs-at-def o-def squash-def*
by (*smt (verit, best) assms(1) class-field.zero-not-one nth-map zero-neq-neg-one*)
then have *insertAt i 0 (signs-at (removeAt i fss) x) = insertAt i (squash (poly (fss ! i) x)) (signs-at (removeAt i fss) x)* **by** (*auto simp add: squash-def*)
also have *... = signs-at (insertAt i (fss ! i) (removeAt i fss)) x*

```

    apply (intro signs-at-insert)
    using length-removeAt[OF assms(1)]
    using assms(1) by linarith
    also have ... = signs-at fss x unfolding insertAt-removeAt[OF assms(1)] by
auto
    ultimately have *:insertAt i 0 (signs-at (removeAt i fss) x) = signs-at fss x
by auto
    thus signs-at fss x ∈ insertAt i 0 ‘ signs-at (removeAt i fss) ‘ {x. poly (fss ! i)
x = 0}
    using z
    by (metis (mono-tags, lifting) mem-Collect-eq rev-image-eqI)
qed
    have insertAt i 0 ‘ consistent-sign-vectors-r (removeAt i fss) {x. poly (fss ! i) x
= 0} =
    {v ∈ consistent-sign-vectors-r fss UNIV. v ! i = 0}
    unfolding consistent-sign-vectors-r-def
    apply (auto simp add: 1)
    apply (subst insertAt-nth)
    using ile by (auto simp add: 2)
    thus ?thesis unfolding eq eq2 .
qed

```

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

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 $(csa \in (\text{consistent-sign-vectors-r } fss \text{ } UNIV) \wedge \neg(\text{has-no-zeros } csa))$
 $\iff csa \in \text{set } (\text{find-sgas-aux } fss)$
proof –
have $a: (\bigwedge p \ q. p \in \text{set } fss \implies$
 $q \in \text{set } fss \implies$
 $p \neq q \implies \text{coprime } p$
 $q)$
using *cast-rat-list-def factorize-polys-map-coprime fs fss* **by** *blast*
have $b: (\bigwedge p. p \in \text{set } fss \implies p \neq 0)$
using *factorize-polys-map-square-free-prod-list semidom-class.prod-list-zero-iff*
square-free-def
using *cast-rat-list-def fs fss* **by** *blast*
have $c: \text{distinct } fss$
using *factorize-polys-map-distinct[OF assms(1)] assms(2)* **unfolding** *cast-rat-list-def*
by *auto*
have forwards: $csa \in (\text{consistent-sign-vectors-r } fss \text{ } UNIV) \implies$
 $\neg (\text{has-no-zeros } csa)$

```

    ⇒ csa ∈ set (find-sgas-aux fss)
  unfolding find-sgas-aux-def
  proof -
    assume csa: csa ∈ (consistent-sign-vectors-r fss UNIV)
    assume hnz: ¬(has-no-zeros csa)
    then obtain i where i: i < length csa csa ! i = 0 unfolding hnz-prop by
  auto
    then have cin: csa ∈ {v ∈ consistent-sign-vectors-r fss UNIV. v ! i = 0} using
  csa by auto
    have 1:i < length fss using i csa unfolding consistent-sign-vectors-r-def by
  auto
    from find-csa-at-index[OF 1 a b c]
    have eq: set (map (λv. insertAt i 0 v) (find-consistent-signs-at-roots (fss ! i)
  (removeAt i fss))) =
      {v ∈ consistent-sign-vectors-r fss UNIV. v ! i = 0} by auto
    show csa ∈ set (concat (map (λi. map (insertAt i 0) (find-consistent-signs-at-roots
  (fss ! i) (removeAt i fss))) [0..<length fss]))
      using cin i unfolding eq[symmetric]
      apply (intro in-set-concat-map[of i])
      using 1 by auto
  qed
  have backwards: csa ∈ set (find-sgas-aux fss) ⇒
    ¬ (has-no-zeros csa) ∧ csa ∈ (consistent-sign-vectors-r fss UNIV)
  proof -
    assume *: csa ∈ set (find-sgas-aux fss)
    then obtain i where i: i < length fss
      csa ∈ set (map (λv. insertAt i 0 v) (find-consistent-signs-at-roots (fss ! i)
  (removeAt i fss)))
      unfolding find-sgas-aux-def
      by auto
    from find-csa-at-index[OF i(1) a b c]
    have eq: set (map (λv. insertAt i 0 v) (find-consistent-signs-at-roots (fss ! i)
  (removeAt i fss))) =
      {v ∈ consistent-sign-vectors-r fss UNIV. v ! i = 0} by auto
    have *: csa ∈ {v ∈ consistent-sign-vectors-r fss UNIV. v ! i = 0} using i eq
  by auto
    then have length csa = length fss unfolding consistent-sign-vectors-r-def by
  auto
    thus ?thesis unfolding has-no-zeros-def using * apply (auto simp add:in-set-conv-nth)
      using i(1) by auto
  qed
  show ?thesis
    using forwards backwards by blast
  qed

lemma mem-append: List.member (l1@l2) m ⇔ (List.member l1 m ∨ List.member
  l2 m)
  by simp

```

```

lemma same-sign-cond-rationals-reals:
  fixes qs:: rat poly list
  assumes lenh: length (fst(factorize-polys qs)) > 0
  shows set(find-sgas (map (map-poly of-rat) (fst(factorize-polys qs)))) = consis-
tent-sign-vectors (fst(factorize-polys qs)) UNIV
proof -
  let ?ftrs = (fst(factorize-polys qs))
  have pairwise-rel-prime: pairwise-coprime-list (fst(factorize-polys qs))
    using factorize-polys-coprime
    by (simp add: coprime-factorize)
  have all-squarefree:  $\forall q. (List.member (fst(factorize-polys qs)) q) \longrightarrow (rsquarefree\ q)$ 
    by (auto intro!: square-free-rsquarefree factorize-polys-square-free simp add:
prod-eq-iff)
  have allnonzero:  $\forall q. (List.member\ ?ftrs\ q) \longrightarrow q \neq 0$ 
    using all-squarefree apply (auto)
    using rsquarefree-def by blast
  have h1:  $\forall csa. (csa \in (consistent-sign-vectors\ ?ftrs\ UNIV) \wedge \neg (has-no-zeros\ csa))$ 
 $\longleftrightarrow csa \in set (find-sgas-aux (cast-rat-list\ ?ftrs))$ 
    using lenh find-csas-lemma-onezero-gen pairwise-rel-prime allnonzero
    by (metis consistent-sign-vectors-consistent-sign-vectors-r eq-fst-iff)
  have h2:  $\forall csa. (csa \in (consistent-sign-vectors\ ?ftrs\ UNIV) \wedge has-no-zeros\ csa)$ 
 $\longleftrightarrow$ 
    List.member (find-consistent-signs-at-roots (coprime-r (cast-rat-list ?ftrs)) (cast-rat-list
?ftrs)) csa
    using lenh allnonzero pairwise-rel-prime
    apply (simp add: find-csas-lemma-nozeros)
    apply (subst find-csas-lemma-nozeros)
    apply (auto simp add: prod-eq-iff)
  done
  have h3:  $\forall csa. List.member (find-sgas (map (map-poly of-rat) ?ftrs)) csa \longleftrightarrow$ 
 $((List.member (find-sgas-aux (cast-rat-list ?ftrs)) csa) \vee (List.member (find-consistent-signs-at-roots$ 
(coprime-r (cast-rat-list ?ftrs)) (cast-rat-list ?ftrs)) csa))
    unfolding find-sgas-def cast-rat-list-def using mem-append
    by metis
  have h4:  $\forall csa. List.member (find-sgas (map (map-poly of-rat) ?ftrs)) csa \longleftrightarrow$ 
 $((csa \in (consistent-sign-vectors\ ?ftrs\ UNIV) \wedge has-no-zeros\ csa) \vee (csa \in$ 
(consistent-sign-vectors ?ftrs UNIV)  $\wedge \neg (has-no-zeros\ csa)))$ 
    using h1 h2 h3 by auto
  have h5:  $\forall csa. (csa \in (consistent-sign-vectors\ ?ftrs\ UNIV) \wedge has-no-zeros\ csa)$ 
 $\vee (csa \in (consistent-sign-vectors\ ?ftrs\ UNIV) \wedge \neg (has-no-zeros\ csa))$ 
 $\longleftrightarrow csa \in (consistent-sign-vectors\ ?ftrs\ UNIV)$ 
    by auto
  then have  $\forall csa. List.member (find-sgas (map (map-poly of-rat) ?ftrs)) csa \longleftrightarrow$ 
 $csa \in (consistent-sign-vectors\ ?ftrs\ UNIV)$ 
    using h4
    by blast
  then show ?thesis

```

by auto
qed

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

assumes *factorize-polys* $ps = (ftrs, data)$

assumes *sgas* = *find-sgas* (*map* (*map-poly of-rat*) *ftrs*)

assumes *sgas-set*: *set* (*sgas*) = *consistent-sign-vectors* *ftrs UNIV*

shows *set* (*map* (*undo-factorize-polys* *data*) *sgas*) = *consistent-sign-vectors* *ps UNIV*

proof –

have *h*: $\forall x. \text{undo-factorize-polys } data \text{ (sign-vec } ftrs \ x) = \text{sign-vec } ps \ x$

using *factorize-polys-undo-factorize-polys*

by (*simp add: assms(1)*)

have *h1*: $\forall x. \text{sign-vec } ps \ x \in \text{set} \text{ (map (undo-factorize-polys } data) \text{ sgas)}$

using *sgas-set*

by (*metis UNIV-I consistent-sign-vectors-def h image-eqI image-set*)

then have *subset-h*: *consistent-sign-vectors* *ps UNIV* $\subseteq \text{set} \text{ (map (undo-factorize-polys } data) \text{ sgas)}$

unfolding *consistent-sign-vectors-def* by auto

have *supset-h*: *consistent-sign-vectors* *ps UNIV* $\supseteq \text{set} \text{ (map (undo-factorize-polys } data) \text{ sgas)}$

proof –

have $\forall \text{ ele. } \text{ele} \in \text{set} \text{ (map (undo-factorize-polys } data) \text{ sgas)} \longrightarrow$

$(\exists n < \text{length } \text{sgas. } \text{ele} = \text{undo-factorize-polys } data \text{ (nth } \text{sgas } n))$

using *index-of-lookup(1)* *index-of-lookup(2)* by *fastforce*

then have $\forall \text{ ele. } \text{ele} \in \text{set} \text{ (map (undo-factorize-polys } data) \text{ sgas)} \longrightarrow$

$(\exists x. \text{ele} = \text{undo-factorize-polys } data \text{ (sign-vec } ftrs \ x))$

using *sgas-set* *apply* (*auto*) using *consistent-sign-vectors-def* by auto

then show *?thesis* using *h*

using *consistent-sign-vectors-def* by auto

qed

show *?thesis* using *subset-h* *supset-h*

by *blast*

qed

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*

proof –

have *set-eq*: *set* (*find-consistent-signs* *qs*) = $\{(\text{map } (\lambda x. \text{if poly } x \ 0 < 0 \text{ then } -1 \text{ else if poly } x \ 0 = 0 \text{ then } 0 \text{ else } 1) \text{ } qs)\}$

using *empty* *unfolding* *find-consistent-signs-def* *apply* (*auto*)

apply (*metis fst-conv*)

by (*metis prod.collapse*)

have *deg-q-prop*: *fst*(*factorize-polys* *qs*) = [] $\implies (\forall q \in \text{set}(qs). \text{degree } q = 0)$

apply (*rule ccontr*)

proof *clarsimp*

fix *q*

```

assume *:fst(factorize-polys qs) = []
assume q: q ∈ set qs 0 < degree q
obtain arb where factorize-rat-poly-monic q = (arb,[])
using * q unfolding factorize-polys-def apply (auto simp add:Let-def)
by (metis prod.collapse)
from squarefree-factorization-degree[OF factorize-rat-poly-monic-square-free-factorization[OF
this]]
have degree q = 0 by auto
thus False using q by auto
qed
have deg-q-cons: (∀ q ∈ set(qs). degree q = 0) ⇒ (consistent-sign-vectors qs
UNIV = {(map (λx. if poly x 0 < 0 then -1 else if poly x 0 = 0 then 0 else 1)
qs)})
proof -
assume deg-z: (∀ q ∈ set(qs). degree q = 0)
then have ∀ q ∈ set(qs). ∀ x y. poly q x = poly q y
apply (auto)
by (meson constant-def constant-degree)
then have csv: consistent-sign-vectors qs UNIV = {sign-vec qs 0}
unfolding consistent-sign-vectors-def sign-vec-def apply (auto)
apply (metis deg-z degree-0-id of-rat-hom.map-poly-hom-coeff-lift poly-0-coeff-0
poly-const-conv squash-real-of-rat)
by (metis (mono-tags, lifting) class-semiring.add.one-closed comp-def image-iff
list.map-cong0 of-rat-hom.poly-map-poly-0)
have sign-vec qs 0 = (map (λx. if poly x 0 < 0 then -1 else if poly x 0 = 0
then 0 else 1) qs)
unfolding sign-vec-def squash-def by auto
then show consistent-sign-vectors qs UNIV = {(map (λx. if poly x 0 < 0 then
-1 else if poly x 0 = 0 then 0 else 1) qs)}
using csv by auto
qed
then show ?thesis
using deg-q-prop deg-q-cons set-eq
by (simp add: empty)
qed

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
proof -
let ?fs = fst(factorize-polys qs)
let ?data = snd(factorize-polys qs)
let ?sgas = find-sgas (map (map-poly of-rat) ?fs)
have h0: set (?sgas) = consistent-sign-vectors ?fs UNIV
using lenh same-sign-cond-rationals-reals coprime-factorize by auto
then have h1: set (map (undo-factorize-polys ?data) ?sgas) = consistent-sign-vectors
qs UNIV
using factorize-polys-undo-factorize-polys-set

```

```

    by simp
  then show ?thesis using lenh
    by (smt (verit, best) find-consistent-signs-def main-step-aux1 prod.case prod.collapse)
qed

```

```

lemma main-step:
  fixes qs:: rat poly list
  shows set (find-consistent-signs qs) = consistent-sign-vectors qs UNIV
  using main-step-aux1 main-step-aux2 by auto

```

17.5 Decision Procedure: Main Lemmas

```

lemma decide-univ-lem-helper:
  assumes (fml-struct, polys) = convert fml
  shows ( $\forall x::\text{real}. \text{lookup-sem fml-struct (map (\lambda p. \text{rpoly } p \ x) \ \text{polys}))} \longleftrightarrow$ 
    (decide-universal fml))
  using universal-lookup-sem main-step unfolding decide-universal-def apply
  (auto)
  apply (metis assms convert-closed fst-conv snd-conv)
  by (metis (full-types) assms convert-closed fst-conv snd-conv)

```

```

lemma decide-exis-lem-helper:
  assumes (fml-struct, polys) = convert fml
  shows ( $\exists x::\text{real}. \text{lookup-sem fml-struct (map (\lambda p. \text{rpoly } p \ x) \ \text{polys}))} \longleftrightarrow$ 
    (decide-existential fml))
  using existential-lookup-sem main-step unfolding decide-existential-def apply
  (auto)
  apply (metis assms convert-closed fst-conv snd-conv)
  by (metis (full-types) assms convert-closed fst-conv snd-conv)

```

```

theorem decision-procedure:
  shows ( $\forall x::\text{real}. \text{fml-sem fml } x \longleftrightarrow$  (decide-universal fml))
    ( $\exists x::\text{real}. \text{fml-sem fml } x \longleftrightarrow$  (decide-existential fml))
  using convert-semantics-lem decide-univ-lem-helper apply (auto)
  apply (simp add: convert-semantics)
  apply (metis convert-def convert-semantics fst-conv snd-conv)
  using convert-semantics-lem
  by (metis convert-def convert-semantics decide-exis-lem-helper fst-conv snd-conv)

```

```

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]), [[1],[0],[-1]]))*

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 subsets* (*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,* Σ) = *calculate-data-R* *p qs* in Σ)

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

shows *find-consistent-signs-at-roots-R* *p qs* = *snd* (*snd* (*calculate-data-R* *p qs*))
by (*simp* *add: case-prod-beta find-consistent-signs-at-roots-R-def*)

end

```

theory Renegar-Proofs
  imports Renegar-Algorithm
    BKR-Proofs
begin

```

22 Tarski Queries Changed

```

lemma construct-NofI-R-relation:
  fixes p:: real poly
  fixes I1:: real poly list
  fixes I2:: real poly list
  shows construct-NofI-R p I1 I2 =
    construct-NofI (sum-list (map power2 (p # I1))) I2
  unfolding construct-NofI-R-def construct-NofI-def
  by metis

```

```

lemma sum-list-map-power2:
  shows sum-list (map power2 ls) ≥ (0::real poly)
  apply (induct ls)
  by auto

```

```

lemma sum-list-map-power2-poly:
  shows poly (sum-list (map power2 (ls::real poly list))) x ≥ (0::real)
  apply (induct ls)
  by auto

```

```

lemma construct-NofI-R-prop-helper:
  fixes p:: real poly
  fixes I1:: real poly list
  fixes I2:: real poly list
  assumes nonzero: p ≠ 0
  shows construct-NofI-R p I1 I2 =
    rat-of-int (int (card {x. poly (sum-list (map (λx. x2) (p # I1))) x = 0 ∧ poly
(prod-list I2) x > 0}) –
    int (card {x. poly (sum-list (map (λx. x2) (p # I1))) x = 0 ∧ poly (prod-list
I2) x < 0}))
  proof –
    show ?thesis unfolding construct-NofI-R-relation[of p I1 I2]
    apply (subst construct-NofI-prop[of - I2])
    apply auto
    using assms sum-list-map-power2
    by (metis le-add-same-cancel1 power2-less-eq-zero-iff)
qed

```

```

lemma zer-iff:
  fixes p:: real poly
  fixes ls:: real poly list
  shows poly (sum-list (map (λx. x2) ls)) x = 0 ↔ (∀ i ∈ set ls. poly i x = 0)
  proof (induct ls)

```

```

case Nil
then show ?case by auto
next
case (Cons a I1)
then show ?case
  apply simp
  apply (subst add-nonneg-eq-0-iff)
  using zero-le-power2 apply blast
  using sum-list-map-power2-poly apply presburger
  by simp
qed

```

lemma *construct-NofI-prop-R*:

```

fixes p:: real poly
fixes I1:: real poly list
fixes I2:: real poly list
assumes nonzero: p≠0
shows construct-NofI-R p I1 I2 =
  (rat-of-int (int (card {x. poly p x = 0 ∧ (∀ q ∈ set I1. poly q x = 0) ∧ poly
(prod-list I2) x > 0}) -
  int (card {x. poly p x = 0 ∧ (∀ q ∈ set I1. poly q x = 0) ∧ poly (prod-list I2)
x < 0}))
  unfolding construct-NofI-R-prop-helper[OF nonzero]
  using zer-iff
  apply auto
  by (smt (verit, del-insts) Collect-cong sum-list-map-power2-poly zero-le-power2
zero-less-power2)

```

23 Matrix Equation

definition *map-sgas*:: rat list ⇒ rat list
where *map-sgas* l = map (λr. (1 - r²)) l

definition *z-R*:: (nat list*nat list) ⇒ rat list ⇒ rat
where *z-R* index-list sign-asg ≡ (prod-list (map (nth (map-sgas sign-asg)) (fst(index-list))))*(prod-list (map (nth sign-asg) (snd(index-list))))

definition *mtx-row-R*:: rat list list ⇒ (nat list * nat list) ⇒ rat list
where *mtx-row-R* sign-list index-list ≡ (map ((z-R index-list)) sign-list)

definition *matrix-A-R*:: rat list list ⇒ (nat list * nat list) list ⇒ rat mat
where *matrix-A-R* sign-list subset-list =
(mat-of-rows-list (length sign-list) (map (λi .(mtx-row-R sign-list i)) subset-list))

definition *all-list-constr-R*:: (nat list*nat list) list ⇒ nat ⇒ bool
where *all-list-constr-R* L n ≡ (∀ x. List.member L x → (list-constr (fst x) n ∧ list-constr (snd x) n))

definition *alt-matrix-A-R*:: rat list list ⇒ (nat list*nat list) list ⇒ rat mat

where *alt-matrix-A-R signs subsets* = (mat (length subsets) (length signs)
 (λ(i, j). z-R (subsets ! i) (signs ! j)))

lemma *alt-matrix-char-R: alt-matrix-A-R signs subsets = matrix-A-R signs subsets*

proof –

have *h0*: (∧ i j. i < length subsets ⇒
 j < length signs ⇒
 map (λ index-list. map (z-R index-list) signs) subsets ! i ! j = z-R (subsets
 ! i) (signs ! j))

proof –

fix *i*

fix *j*

assume *i-lt*: i < length subsets

assume *j-lt*: j < length signs

show ((map (λ index-list. map (z-R index-list) signs) subsets) ! i) ! j = z-R
 (subsets ! i) (signs ! j)

proof –

have *h0*: (map (λ index-list. map (z-R index-list) signs) subsets) ! i = map
 (z-R (subsets ! i)) signs

using *nth-map i-lt*

by *blast*

then show *?thesis* **using** *nth-map j-lt*

by *simp*

qed

qed

have *h*: mat (length subsets) (length signs) (λ(i, j). z-R (subsets ! i) (signs ! j))

=

mat (length subsets) (length signs) (λ(i, y). map (λ index-list. map (z-R in-
 dex-list) signs) subsets ! i ! y)

using *h0 eq-matI*[**where** *A* = mat (length subsets) (length signs) (λ(i, j). z-R
 (subsets ! i) (signs ! j)),

where *B* = mat (length subsets) (length signs) (λ(i, y). map (λ index-list.
 map (z-R index-list) signs) subsets ! i ! y)]

by *auto*

show *?thesis* **unfolding** *alt-matrix-A-R-def matrix-A-R-def mat-of-rows-list-def*

apply (*auto*) **unfolding** *mtx-row-R-def*

using *h* **by** *blast*

qed

lemma *subsets-are-rows-R*: ∀ i < (length subsets). row (alt-matrix-A-R signs sub-
 sets) i = vec (length signs) (λ j. z-R (subsets ! i) (signs ! j))

unfolding *row-def* **unfolding** *alt-matrix-A-R-def* **by** *auto*

lemma *signs-are-cols-R*: ∀ i < (length signs). col (alt-matrix-A-R signs subsets) i
 = vec (length subsets) (λ j. z-R (subsets ! j) (signs ! i))

unfolding *col-def* **unfolding** *alt-matrix-A-R-def* **by** *auto*

definition *consistent-sign-vec::real poly list ⇒ real ⇒ rat list*

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

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

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

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

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

assumes $p \neq 0$

assumes $i < \text{length } signs$

shows $(\text{construct-lhs-vector-R } p \text{ } qs \text{ } signs) \ \$ \ i =$

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

proof –

from *poly-roots-finite*[*OF* *assms*(1)] **have** $\text{finite } \{x. \text{poly } p \ x = 0\}$.

then have *eq*: (*Collect*

$((\lambda v. v = signs \ ! \ i) \circ$

$\text{consistent-sign-vec } qs) \cap$

$\text{set } (\text{sorted-list-of-set}$

$\{x. \text{poly } p \ x = 0\}) =$

$\{x. \text{poly } p \ x = 0 \wedge \text{consistent-sign-vec } qs \ x = signs \ ! \ i\}$

by *auto*

show *?thesis*

unfolding *construct-lhs-vector-R-def* *vec-of-list-index* *characterize-root-list-p-def*

apply *auto*

apply (*subst* *nth-map*[*OF* *assms*(2)])

apply *auto*

apply (*subst* *distinct-length-filter*)

apply (*auto*)

using *eq*

by *auto*

qed

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

assumes $p \neq 0$

shows $(\text{construct-lhs-vector-R } p \text{ } qs \text{ } signs) =$

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

```

x) = s))) signs)
apply (rule eq-vecI)
apply (auto simp add: construct-lhs-vector-clean-R[OF assms] )
apply (simp add: vec-of-list-index)
unfolding construct-lhs-vector-R-def
using assms construct-lhs-vector-clean-R construct-lhs-vector-def apply auto[1]
apply (simp add: construct-lhs-vector-R-def)
by auto

```

lemma *z-signs-R2*:

```

fixes I:: nat list
fixes signs:: rat list
assumes lf: list-all ( $\lambda i. i < \text{length signs}$ ) 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 using assms
proof (induct I)
  case Nil
  then show ?case by auto
next
  case (Cons a I)
  moreover have eo: signs ! a = 1  $\vee$  signs ! a = 0  $\vee$  signs ! a = -1
  using assms Cons by (simp add: list-all-iff)
  have prod-list (map (!) (map-sgas signs)) (a # I) = (1 - (signs ! a)2)*prod-list
  (map (!) (map-sgas signs)) (I)
  unfolding map-sgas-def apply (auto)
  using calculation(2) by auto
  then show ?case using eo
  using Cons.hyps calculation(2) la by auto
qed

```

lemma *z-signs-R1*:

```

fixes I:: nat list
fixes signs:: rat list
assumes lf: list-all ( $\lambda i. i < \text{length signs}$ ) 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 using assms
proof (induct I)
  case Nil
  then show ?case by auto
next
  case (Cons a I)
  moreover have signs ! a = 1  $\vee$  signs ! a = 0  $\vee$  signs ! a = -1
  using assms Cons by (simp add: list-all-iff)

then have eo: 1 - (signs ! a)2 = 1  $\vee$  1 - (signs ! a)2 = 0

```

```

using cancel-comm-monoid-add-class.diff-cancel diff-zero by fastforce
have prod-list (map (!) (map-sgas signs)) (a # I) = (1 - (signs ! a)2)*prod-list
(map (!) (map-sgas signs)) (I)
unfolding map-sgas-def apply (auto)
using calculation(2) by auto
then show ?case using eo
using Cons.hyps calculation(2) la by auto
qed

```

lemma *z-signs-R*:

```

fixes I:: (nat list * nat list)
fixes signs:: rat list
assumes lf: list-all (λi. i < length signs) (fst I)
assumes ls: list-all (λi. i < length signs) (snd I)
assumes la: list-all (λs. s = 1 ∨ s = 0 ∨ s = -1) signs
shows (z-R I signs = 1) ∨ (z-R I signs = 0) ∨ (z-R I signs = -1)
using assms z-signs-R2 z-signs-R1 unfolding z-R-def apply (auto)
by (metis (no-types, lifting) mult-cancel-left1 mult-minus1-right)

```

lemma *z-lemma-R*:

```

fixes I:: nat list * nat list
fixes sign:: rat list
assumes consistent: sign ∈ set (characterize-consistent-signs-at-roots p qs)
assumes welldefined1: list-constr (fst I) (length qs)
assumes welldefined2: list-constr (snd I) (length qs)
shows (z-R I sign = 1) ∨ (z-R I sign = 0) ∨ (z-R I sign = -1)
proof (rule z-signs-R)
have same: length sign = length qs using consistent
using characterize-consistent-signs-at-roots-def signs-at-def by fastforce
thus (list-all (λi. i < length sign) (fst I))
using welldefined1
by (auto simp add: list-constr-def characterize-consistent-signs-at-roots-def con-
sistent-sign-vec-copr-def)
thus (list-all (λi. i < length sign) (snd I))
using same welldefined2
by (auto simp add: list-constr-def characterize-consistent-signs-at-roots-def con-
sistent-sign-vec-copr-def)
show list-all (λs. s = 1 ∨ s = 0 ∨ s = -1) sign using consistent
apply (auto simp add: list.pred-map characterize-consistent-signs-at-roots-def
consistent-sign-vec-def)
using Ball-set
by (simp add: list-all-length signs-at-def squash-def)
qed

```

lemma *in-set-R*:

```

fixes p:: real poly
assumes nonzero: p ≠ 0
fixes qs:: real poly list

```

```

fixes sign:: rat list
fixes x:: real
assumes root-p:  $x \in \{x. \text{poly } p \ x = 0\}$ 
assumes sign-fix:  $\text{sign} = \text{consistent-sign-vec } qs \ x$ 
shows  $\text{sign} \in \text{set } (\text{characterize-consistent-signs-at-roots } p \ qs)$ 
proof –
  have h1:  $\text{consistent-sign-vec } qs \ x \in$ 
     $\text{set } (\text{remdups } (\text{map } (\text{signs-at } qs) (\text{sorted-list-of-set } \{x. \text{poly } p \ x = 0\})))$ 
  unfolding consistent-sign-vec-def signs-at-def squash-def
  using root-p nonzero poly-roots-finite set-sorted-list-of-set apply (auto)
  by (smt (verit, ccfv-SIG) Collect-cong comp-def image-eqI map-eq-conv mem-Collect-eq
poly-roots-finite set-sorted-list-of-set)
  thus ?thesis unfolding characterize-consistent-signs-at-roots-def characterize-root-list-p-def
using sign-fix
  by blast
qed

```

lemma *consistent-signs-prop-R*:

```

fixes p:: real poly
assumes nonzero:  $p \neq 0$ 
fixes qs:: real poly list
fixes sign:: rat list
fixes x:: real
assumes root-p:  $x \in \{x. \text{poly } p \ x = 0\}$ 
assumes sign-fix:  $\text{sign} = \text{consistent-sign-vec } qs \ x$ 
shows  $\text{list-all } (\lambda s. s = 1 \vee s = 0 \vee s = -1) \ \text{sign}$ 
using assms unfolding consistent-sign-vec-def squash-def apply (auto)
by (smt (z3) length-map list-all-length nth-map)

```

lemma *horiz-vector-helper-pos-ind-R1*:

```

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:  $x \in \{x. \text{poly } p \ x = 0\}$ 
assumes sign-fix:  $\text{sign} = \text{consistent-sign-vec } qs \ x$ 
assumes asm:  $\text{list-constr } I \ (\text{length } qs)$ 
shows  $(\text{prod-list } (\text{map } (\text{nth } (\text{map-sgas } \text{sign})) \ I)) = 1 \iff$ 
   $(\forall p \in \text{set } (\text{retrieve-polys } qs \ I). \text{poly } p \ x = 0)$ 
using asm
proof (induction I)
  case Nil
  then show ?case unfolding map-sgas-def apply (auto)
  by (simp add: retrieve-polys-def)
next
  case (Cons a xa)

```

```

then have same0: (prod-list (map (nth (map-sgas sign)) xa)) = 1  $\longleftrightarrow$ 
  ( $\forall p \in \text{set (retrieve-polys qs xa)}. \text{poly } p \ x = 0$ ) unfolding list-constr-def by
  auto
have welldef:  $a < \text{length } \text{qs}$  using Cons.premss assms unfolding list-constr-def
by auto
then have h: prod-list (map (!! (map-sgas sign)) (a#xa)) = (1 - (sign !
a)2)*(prod-list (map (!! (map-sgas sign)) (xa)))
unfolding map-sgas-def using assms apply (auto)
by (metis (no-types, lifting) consistent-sign-vec-def length-map nth-map)
have list-all ( $\lambda s. s = 1 \vee s = 0 \vee s = -1$ ) sign using sign-fix unfolding
consistent-sign-vec-def squash-def
apply (auto)
by (smt (z3) length-map list-all-length nth-map)
then have eo: (prod-list (map (!! (map-sgas sign)) (xa))) = 0  $\vee$  (prod-list (map
 (!! (map-sgas sign)) (xa))) = 1
using assms Cons.premss z-signs-R1
by (clarsimp simp add: consistent-sign-vec-def list-all-iff list-constr-def im-
age-iff)
  (smt (verit, del-insts) image-iff length-map list.set-map)
have (sign ! a)2 = 1  $\vee$  (sign ! a)2 = 0 using sign-fix welldef unfolding
consistent-sign-vec-def
by auto
then have s1: (prod-list (map (nth (map-sgas sign)) (a#xa))) = 1  $\longleftrightarrow$ 
  ((sign ! a)2 = 0  $\wedge$  (prod-list (map (!! (map-sgas sign)) (xa))) = 1)
using eo h
by (metis (mono-tags, opaque-lifting) cancel-comm-monoid-add-class.diff-cancel
diff-zero mult.left-neutral mult-zero-left)
have (sign ! a)2 = 0  $\longleftrightarrow$  (poly (qs ! a) x = 0)
using sign-fix unfolding consistent-sign-vec-def welldef apply (auto)
apply (smt (z3) class-field.neg-1-not-0 class-field.zero-not-one nth-map welldef)
by (smt (z3) nth-map welldef)
then have same1:(prod-list (map (nth (map-sgas sign)) (a#xa))) = 1  $\longleftrightarrow$ 
  ((poly (qs ! a) x = 0)  $\wedge$  (prod-list (map (!! (map-sgas sign)) (xa))) = 1) using
s1 by auto
have set (retrieve-polys qs (a#xa)) = {(qs ! a)}  $\cup$  set (retrieve-polys qs xa)
by (metis (no-types, lifting) insert-is-Un list.simps(15) list.simps(9) retrieve-polys-def)
then have same2:( $\forall p \in \text{set (retrieve-polys qs (a#xa))}. \text{poly } p \ x = 0$ ) = ((poly
(qs ! a) x = 0)
 $\wedge$  ( $\forall p \in \text{set (retrieve-polys qs (xa))}. \text{poly } p \ x = 0$ ))
by auto
then show ?case using same0 same1 same2
  assms by auto
qed

```

lemma csv-length-same-as-qlist:

```

fixes p:: real poly
assumes nonzero:  $p \neq 0$ 
fixes qs:: real poly list
fixes sign:: 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  $\text{length } \text{sign} = \text{length } qs$ 
using  $\text{assms}$  unfolding  $\text{consistent-sign-vec-def}$  by  $\text{auto}$ 

lemma  $\text{horiz-vector-helper-zer-ind-R2}$ :
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  $(\text{prod-list } (\text{map } (\text{nth } \text{sign}) \ I)) = 0 \longleftrightarrow$ 
 $(\text{poly } (\text{prod-list } (\text{retrieve-polys } qs \ I)) \ x = 0)$ 
using  $\text{asm}$ 
proof  $(\text{induction } I)$ 
case  $\text{Nil}$ 
then show  $?case$  unfolding  $\text{map-sgas-def}$  apply  $(\text{auto})$  unfolding  $\text{retrieve-polys-def}$ 
by  $\text{auto}$  next
case  $(\text{Cons } a \ xa)$ 
have  $\text{poly } (\text{prod-list } (\text{retrieve-polys } qs \ (a \ \# \ xa))) \ x = (\text{poly } (qs \ ! \ a) \ x) * \text{poly}$ 
 $(\text{prod-list } (\text{retrieve-polys } qs \ (xa))) \ x$ 
by  $(\text{simp add: retrieve-polys-def})$ 
then show  $?case$  using  $\text{Cons}$ 
by  $(\text{simp add: list-all-iff consistent-sign-vec-def list-constr-def sign-fix})$ 
qed

lemma  $\text{horiz-vector-helper-pos-ind-R2}$ :
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  $(\text{prod-list } (\text{map } (\text{nth } \text{sign}) \ I)) = 1 \longleftrightarrow$ 
 $(\text{poly } (\text{prod-list } (\text{retrieve-polys } qs \ I)) \ x > 0)$ 
using  $\text{asm}$ 
proof  $(\text{induction } I)$ 
case  $\text{Nil}$ 
then show  $?case$  unfolding  $\text{map-sgas-def}$  apply  $(\text{auto})$  unfolding  $\text{retrieve-polys-def}$ 
by  $\text{auto}$  next
case  $(\text{Cons } a \ xa)$ 

```

```

then have ih: (prod-list (map (!! sign) xa) = 1) = (0 < poly (prod-list (retrieve-polys
qs xa)) x)
  unfolding list-constr-def by auto
  have lensame: length sign = length qs using sign-fix csv-length-same-as-qlist[of
p x sign qs]
    nonzero root-p by auto
  have poly (prod-list (retrieve-polys qs (a # xa))) x = (poly (qs ! a) x)*poly
(prod-list (retrieve-polys qs (xa))) x
  by (simp add: retrieve-polys-def)
  then have iff1: (poly (prod-list (retrieve-polys qs (a#xa))) x > 0)  $\longleftrightarrow$ 
    (((poly (qs ! a) x) > 0  $\wedge$  poly (prod-list (retrieve-polys qs (xa))) x > 0)  $\vee$ 
    ((poly (qs ! a) x) < 0  $\wedge$  poly (prod-list (retrieve-polys qs (xa))) x < 0))
  by (metis zero-less-mult-iff)
  have prodsame: (prod-list (map (nth sign) (a#xa))) = (sign ! a)* (prod-list (map
(nth sign) (xa)))
  using lensame Cons.premis unfolding list-constr-def by auto
  have sagt: sign ! a = 1  $\longleftrightarrow$  (poly (qs ! a) x) > 0
  using assms Cons.premis
  by (simp add: list-all-iff consistent-sign-vec-def list-constr-def)
  have salt: sign ! a = -1  $\longleftrightarrow$  (poly (qs ! a) x) < 0
  using assms Cons.premis
  by (simp add: list-all-iff consistent-sign-vec-def list-constr-def)
  have h1: ((poly (qs ! a) x) > 0  $\wedge$  poly (prod-list (retrieve-polys qs (xa))) x > 0)
 $\longrightarrow$ 
    (prod-list (map (nth sign) (a#xa))) = 1
  using prodsame sagt ih by auto
  have eo: (prod-list (map (!! sign) xa) = 1)  $\vee$  (prod-list (map (!! sign) xa) =
0)  $\vee$ 
    (prod-list (map (!! sign) xa) = -1)
  using Cons.premis consistent-signs-prop-R lensame list-constr-def nonzero root-p
sign-fix z-signs-R2 by auto
  have d1: (prod-list (map (!! sign) xa) = -1)  $\implies$  (0 > poly (prod-list (retrieve-polys
qs xa)) x)
  proof -
    assume (prod-list (map (!! sign) xa) = -1)
    then show (0 > poly (prod-list (retrieve-polys qs xa)) x)
      using assms Cons.premis prodsame salt ih
      apply (cases  $\langle$ poly (prod-list (retrieve-polys qs xa)) x = 0 $\rangle$ )
      apply (auto simp add: list-constr-def horiz-vector-helper-zer-ind-R2 not-less)
      apply (metis horiz-vector-helper-zer-ind-R2 list-constr-def mult-minus1 mult-zero-left
neg-equal-iff-equal one-neq-zero root-p)
      apply (metis horiz-vector-helper-zer-ind-R2 list-constr-def root-p zero-neq-neg-one)
      done
  qed
  have d2: (0 > poly (prod-list (retrieve-polys qs xa)) x)  $\longrightarrow$  (prod-list (map (!!
sign) xa) = -1)
  using Cons.premis ih eo assms horiz-vector-helper-zer-ind-R2 [where p = p,
where x = x, where sign = sign, where I = I]
  apply (auto simp add: list-constr-def list-all-iff)

```

```

apply (metis (mono-tags, lifting) Cons.premis eo horiz-vector-helper-zer-ind-R2
list.pred-inject(2) list-constr-def mem-Collect-eq order-less-irrefl)
done
have (prod-list (map (!) sign) xa) = -1) = (0 > poly (prod-list (retrieve-polys
qs xa)) x)
using d1 d2
by blast
then have h2: ((poly (qs ! a) x) < 0 ∧ poly (prod-list (retrieve-polys qs (xa))) x
< 0) →
(prod-list (map (nth sign) (a#xa))) = 1
using prodsame salt ih by auto
have h3: (prod-list (map (nth sign) (a#xa))) = 1 → (((poly (qs ! a) x) > 0 ∧
poly (prod-list (retrieve-polys qs (xa))) x > 0) ∨
((poly (qs ! a) x) < 0 ∧ poly (prod-list (retrieve-polys qs (xa))) x < 0))
using prodsame salt ih assms horiz-vector-helper-zer-ind-R2[where p = p,
where x = x, where sign = sign, where I = I]
by (smt (verit, ccfv-threshold) Cons.premis ⟨poly (prod-list (retrieve-polys qs (a #
xa))) x = poly (qs ! a) x * poly (prod-list (retrieve-polys qs xa)) x⟩ horiz-vector-helper-zer-ind-R2
mem-Collect-eq mult-cancel-left1 mult-not-zero sagt)
then show ?case using h1 h2 h3 iff1 Cons.premis by auto
qed

```

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

```

fixes p:: real poly
assumes nonzero: p≠0
fixes qs:: real poly list
fixes I:: nat list * nat list
fixes sign:: rat list
fixes x:: real
assumes root-p: x ∈ {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 ((∀ p ∈ set (retrieve-polys qs (fst I)). poly p x = 0) ∧ (poly (prod-list
(retrieve-polys qs (snd I))) x > 0)) ↔ (z-R I sign = 1)
proof -
have len: length sign = length qs using sign-fix csv-length-same-as-qlist[of p x
sign qs]
nonzero root-p by auto
have d1: ((∀ p ∈ set (retrieve-polys qs (fst I)). poly p x = 0) ∧ (poly (prod-list
(retrieve-polys qs (snd I))) x > 0)) → (z-R I sign = 1)
using assms horiz-vector-helper-pos-ind-R1[where p = p, where qs = qs,
where sign = sign, where x = x, where I = fst I]
horiz-vector-helper-pos-ind-R2[where p = p, where qs = qs, where sign =
sign, where x = x, where I = snd I] unfolding z-R-def
by auto
have d2: (z-R I sign = 1) → ((∀ p ∈ set (retrieve-polys qs (fst I)). poly p x =
0) ∧ (poly (prod-list (retrieve-polys qs (snd I))) x > 0))
proof -

```

```

have h1: (z-R I sign = 1) → (∀ p ∈ set (retrieve-polys qs (fst I)). poly p x =
0)
proof –
  have (prod-list (map (nth (map-sgas sign)) (fst I))) = 1 ∨ (prod-list (map
(nth (map-sgas sign)) (fst I))) = 0
  using len consistent-signs-prop-R[where p = p, where qs = qs, where x
= x, where sign = sign] z-signs-R1[where signs = sign, where I = fst I] assms

  unfolding list-constr-def
  by auto
  then show ?thesis
  using z-signs-R1[where signs = sign, where I = fst I] horiz-vector-helper-pos-ind-R1[where
sign = sign, where I = fst I, where p = p, where x = x] assms
  apply (auto)
  by (metis (mono-tags, opaque-lifting) ⟨prod-list (map (!) (map-sgas sign))
(fst I) = 1 ∨ prod-list (map (!) (map-sgas sign)) (fst I) = 0⟩ mult-zero-left
z-R-def)
  qed
  then have h2: (z-R I sign = 1) → (poly (prod-list (retrieve-polys qs (snd I)))
x > 0)
  unfolding z-R-def using assms horiz-vector-helper-pos-ind-R2[where p = p,
where x = x, where sign = sign, where qs = qs, where I = snd I]
  by (metis horiz-vector-helper-pos-ind-R1 mult.left-neutral)
  show ?thesis using h1 h2 by auto
  qed
  show ?thesis using d1 d2 by blast
qed

```

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

```

fixes p:: real poly
assumes nonzero: p≠0
fixes qs:: real poly list
fixes I:: nat list*nat list
fixes sign:: rat list
fixes x:: real
assumes root-p: x ∈ {x. poly p x = 0}
assumes sign-fix: sign = consistent-sign-vec qs x
assumes welldefined1: list-constr (fst I) (length qs)
assumes welldefined2: list-constr (snd I) (length qs)
shows ((∀ p ∈ set (retrieve-polys qs (fst I)). poly p x = 0) ∧ (poly (prod-list
(retrieve-polys qs (snd I))) x > 0)) ↔ (z-R I sign = 1)
using horiz-vector-helper-pos-ind-R
using nonzero root-p sign-fix welldefined1 welldefined2 by blast

```

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

```

fixes p:: real poly
assumes nonzero: p≠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 –
  have set-hyp:  $\text{sign} \in \text{set } (\text{characterize-consistent-signs-at-roots } p \ qs)$ 
    using in-set-R[of p x sign qs] nonzero root-p sign-fix by blast
  have z-hyp:  $((z\text{-R } I \ \text{sign} = 1) \vee (z\text{-R } I \ \text{sign} = 0) \vee (z\text{-R } I \ \text{sign} = -1))$ 
    using welldefined1 welldefined2 set-hyp z-lemma-R[where sign=sign, where
I = I, where p=p, where qs=qs] by blast
  have d1:  $((\forall p \in \text{set } (\text{retrieve-polys } qs \ (\text{fst } I)). \text{poly } p \ x = 0) \wedge (\text{poly } (\text{prod-list } (\text{retrieve-polys } qs \ (\text{snd } I))) \ x < 0)) \implies (z\text{-R } I \ \text{sign} = -1)$ 
    using horiz-vector-helper-pos-R
    using nonzero root-p sign-fix welldefined1 welldefined2
    by (smt (verit, ccfv-threshold) horiz-vector-helper-pos-ind-R1 horiz-vector-helper-zer-ind-R2 mem-Collect-eq mult-eq-0-iff z-R-def z-hyp)
  have d2:  $(z\text{-R } I \ \text{sign} = -1) \implies ((\forall p \in \text{set } (\text{retrieve-polys } qs \ (\text{fst } I)). \text{poly } p \ x = 0) \wedge (\text{poly } (\text{prod-list } (\text{retrieve-polys } qs \ (\text{snd } I))) \ x < 0))$ 
    using horiz-vector-helper-pos-ind-R1 horiz-vector-helper-zer-ind-R2 nonzero root-p sign-fix welldefined1 welldefined2
    by (smt (verit, ccfv-threshold) class-field.neg-1-not-0 consistent-sign-vec-def consistent-signs-prop-R equal-neg-zero horiz-vector-helper-pos-ind-R2 length-map list-all-length list-constr-def mem-Collect-eq mem-Collect-eq mult-cancel-left1 mult-not-zero retrieve-polys-def z-R-def z-signs-R1 zero-neg-one)
  then show ?thesis using d1 d2
    by linarith
qed

```

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 –
  have  $p \neq 0$  using nonzero by auto
  from construct-lhs-vector-cleaner[OF this]
  have rhseq:  $\text{construct-lhs-vector-R } p \ qs \ signs =$ 
     $\text{vec-of-list}$ 
     $(\text{map } (\lambda s. \text{rat-of-int } (\text{card } \{x. \text{poly } p \ x = 0 \wedge \text{consistent-sign-vec } qs \ x = s\}))$ 
     $signs)$ 

```

```

using construct-lhs-vector-cleaner-R nonzero by presburger
have (vec-of-list (mtx-row-R signs I) · (construct-lhs-vector-R p qs signs)) =
  sum-list (map2 (*) (mtx-row-R signs I) (map (λs. rat-of-int (card {x. poly p x
= 0 ∧ consistent-sign-vec qs x = s))) signs))
unfolding rhseq
apply (intro vec-of-list-dot-rewrite)
by (auto simp add: mtx-row-R-def)
thus ?thesis unfolding mtx-row-R-def
using map2-map-map
by (auto simp add: map2-map-map)
qed

```

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 ≠ 0
assumes distinct-signs: distinct signs
assumes all-info: set (characterize-consistent-signs-at-roots p qs) ⊆ set(signs)
assumes welldefined1: list-constr (fst I) (length qs)
assumes welldefined2: list-constr (snd I) (length qs)
shows
  (vec-of-list (mtx-row-R signs I) · (construct-lhs-vector-R p qs signs)) =
  (vec-of-list (mtx-row-R (characterize-consistent-signs-at-roots p qs) I) ·
  (construct-lhs-vector-R p qs (characterize-consistent-signs-at-roots p qs)))
proof –
have h0: ∀ sgn. sgn ∈ set signs ∧ sgn ∉ consistent-sign-vec qs ‘ set (characterize-root-list-p
p) ∧
  0 < rat-of-nat (card {xa. poly p xa = 0 ∧ consistent-sign-vec qs xa = sgn})
   $\longrightarrow$  z-R I sgn = 0
proof –
have  $\forall$  sgn. sgn ∈ set signs ∧ sgn ∉ consistent-sign-vec qs ‘ set (characterize-root-list-p
p) ∧ 0 < rat-of-int (card
  {xa. poly p xa = 0 ∧ consistent-sign-vec qs xa = sgn})  $\longrightarrow$  {xa. poly
p xa = 0 ∧ consistent-sign-vec qs xa = sgn} ≠ {}
by fastforce
then show ?thesis
proof –
  { fix iis :: rat list
    have ff1: 0 ≠ p
      using nonzero rsquarefree-def by blast
    obtain rr :: (real ⇒ bool) ⇒ real where
      ff2: ∧p. p (rr p) ∨ Collect p = {}
      by (metis Collect-empty-eq)
    { assume  $\exists$  is. is = iis ∧ {r. poly p r = 0 ∧ consistent-sign-vec qs r = is}
   $\neq \{\}$ 
      then have  $\exists$  is. consistent-sign-vec qs (rr (λr. poly p r = 0 ∧ consis-

```

$tent\text{-}sign\text{-}vec\ qs\ r = is)) = iis \wedge \{r.\ poly\ p\ r = 0 \wedge consistent\text{-}sign\text{-}vec\ qs\ r = is\}$
 $\neq \{\}$

using *ff2*
by (*metis* (*mono-tags*, *lifting*))
then have $\exists r.\ poly\ p\ r = 0 \wedge consistent\text{-}sign\text{-}vec\ qs\ r = iis$
using *ff2*
by *smt*
then have $iis \in consistent\text{-}sign\text{-}vec\ qs\ \text{'set (sorted-list-of-set \{r.\ poly\ p\ r$
 $= 0\})}$

using *ff1 poly-roots-finite*
by (*metis* (*mono-tags*) *imageI mem-Collect-eq set-sorted-list-of-set*) }
then have $iis \notin set\ signs \vee iis \in consistent\text{-}sign\text{-}vec\ qs\ \text{'set (characterize-root-list-p$
 $p) \vee \neg 0 < rat\text{-}of\text{-}int\ (int\ (card\ \{r.\ poly\ p\ r = 0 \wedge consistent\text{-}sign\text{-}vec\ qs\ r = iis\}))$
by (*metis* (*no-types*) $\langle \forall sgn.\ sgn \in set\ signs \wedge sgn \notin consistent\text{-}sign\text{-}vec\ qs$
 $\text{'set (characterize-root-list-p p) \wedge 0 < rat\text{-}of\text{-}int\ (int\ (card\ \{xa.\ poly\ p\ xa = 0 \wedge$
 $consistent\text{-}sign\text{-}vec\ qs\ xa = sgn\}) \rangle \longrightarrow \{xa.\ poly\ p\ xa = 0 \wedge consistent\text{-}sign\text{-}vec\ qs$
 $xa = sgn\} \neq \{\}$ $\rangle characterize\text{-}root\text{-}list\text{-}p\text{-}def$) }
then show *?thesis*
by *fastforce*

qed
qed

then have $\forall sgn.\ sgn \in set\ signs \wedge sgn \notin consistent\text{-}sign\text{-}vec\ qs\ \text{'set (characterize-root-list-p$
 $p) \longrightarrow ((0 = rat\text{-}of\text{-}nat\ (card$
 $\{xa.\ poly\ p\ xa = 0 \wedge consistent\text{-}sign\text{-}vec\ qs\ xa = sgn\}) \vee z\text{-}R\ I\ sgn$
 $= 0))$

by *auto*

then have *hyp*: $\forall s.\ s \in set\ signs \wedge s \notin consistent\text{-}sign\text{-}vec\ qs\ \text{'set (characterize-root-list-p$
 $p) \longrightarrow (z\text{-}R\ I\ s * rat\text{-}of\text{-}nat\ (card\ \{x.\ poly\ p\ x = 0 \wedge consistent\text{-}sign\text{-}vec\ qs\ x =$
 $s\}) = 0)$

by *auto*

then have $(\sum_{s \in set(signs)} z\text{-}R\ I\ s * rat\text{-}of\text{-}nat\ (card\ \{x.\ poly\ p\ x = 0 \wedge$
 $consistent\text{-}sign\text{-}vec\ qs\ x = s\})) =$
 $(\sum_{s \in (set\ (signs) \cap (consistent\text{-}sign\text{-}vec\ qs\ \text{'set (characterize-root-list-p p))})} z\text{-}R\ I\ s * rat\text{-}of\text{-}nat\ (card\ \{x.\ poly\ p\ x = 0 \wedge consistent\text{-}sign\text{-}vec\ qs\ x = s\}))$

proof –
have $set(signs) = (set\ (signs) \cap (consistent\text{-}sign\text{-}vec\ qs\ \text{'set (characterize-root-list-p$
 $p))) \cup$
 $(set(signs) - (consistent\text{-}sign\text{-}vec\ qs\ \text{'set (characterize-root-list-p p)))$

by *blast*

then have *sum-rewrite*: $(\sum_{s \in set(signs)} z\text{-}R\ I\ s * rat\text{-}of\text{-}nat\ (card\ \{x.\ poly\ p$
 $x = 0 \wedge consistent\text{-}sign\text{-}vec\ qs\ x = s\})) =$
 $(\sum_{s \in (set\ (signs) \cap (consistent\text{-}sign\text{-}vec\ qs\ \text{'set (characterize-root-list-p$
 $p))) \cup$
 $(set(signs) - (consistent\text{-}sign\text{-}vec\ qs\ \text{'set (characterize-root-list-p p)))} z\text{-}R\ I\ s * rat\text{-}of\text{-}nat\ (card\ \{x.\ poly\ p\ x = 0 \wedge consistent\text{-}sign\text{-}vec\ qs\ x = s\}))$

by *auto*

then have *sum-split*: $(\sum_{s \in (set\ (signs) \cap (consistent\text{-}sign\text{-}vec\ qs\ \text{'set (characterize-root-list-p$
 $p))) \cup$
 $(set(signs) - (consistent\text{-}sign\text{-}vec\ qs\ \text{'set (characterize-root-list-p p)))} z\text{-}R\ I\ s * rat\text{-}of\text{-}nat\ (card\ \{x.\ poly\ p\ x = 0 \wedge consistent\text{-}sign\text{-}vec\ qs\ x = s\}))$

$z\text{-}R\ I\ s * \text{rat-of-nat} (\text{card} \{x. \text{poly } p\ x = 0 \wedge \text{consistent-sign-vec } qs\ x = s\})$
 $=$
 $(\sum s \in (\text{set } (signs) \cap (\text{consistent-sign-vec } qs \text{ ' set } (\text{characterize-root-list-p } p)))) . z\text{-}R\ I\ s * \text{rat-of-nat} (\text{card} \{x. \text{poly } p\ x = 0 \wedge \text{consistent-sign-vec } qs\ x = s\})$
 $+ (\sum s \in (\text{set } (signs) - (\text{consistent-sign-vec } qs \text{ ' set } (\text{characterize-root-list-p } p)))) . z\text{-}R\ I\ s * \text{rat-of-nat} (\text{card} \{x. \text{poly } p\ x = 0 \wedge \text{consistent-sign-vec } qs\ x = s\})$
by (*metis* (*no-types*, *lifting*) *List.finite-set sum.Int-Diff*)
have *sum-zero*: $(\sum s \in (\text{set } (signs) - (\text{consistent-sign-vec } qs \text{ ' set } (\text{characterize-root-list-p } p)))) . z\text{-}R\ I\ s * \text{rat-of-nat} (\text{card} \{x. \text{poly } p\ x = 0 \wedge \text{consistent-sign-vec } qs\ x = s\})$
 $= 0$
using *hyp*
by (*simp add: hyp*)
show *?thesis* **using** *sum-rewrite sum-split sum-zero* **by** *linarith*
qed
then have *set-eq*: $\text{set } (\text{remdups } (\text{map } (\text{consistent-sign-vec } qs) (\text{characterize-root-list-p } p))) = \text{set } (signs) \cap (\text{consistent-sign-vec } qs \text{ ' set } (\text{characterize-root-list-p } p))$
using *all-info* **apply** (*simp add: characterize-consistent-signs-at-roots-def subset-antisym*)
by (*smt* (*z3*) *Int-subset-iff consistent-sign-vec-def list.set-map map-eq-conv o-apply signs-at-def squash-def subsetI subset-antisym*)
have *hyp1*: $(\sum s \leftarrow \text{signs} . z\text{-}R\ I\ s * \text{rat-of-nat} (\text{card} \{x. \text{poly } p\ x = 0 \wedge \text{consistent-sign-vec } qs\ x = s\})) =$
 $(\sum s \in \text{set } (signs) . z\text{-}R\ I\ s * \text{rat-of-nat} (\text{card} \{x. \text{poly } p\ x = 0 \wedge \text{consistent-sign-vec } qs\ x = s\}))$
using *distinct-signs sum-list-distinct-conv-sum-set* **by** *blast*
have *hyp2*: $(\sum s \leftarrow \text{remdups } (\text{map } (\text{consistent-sign-vec } qs) (\text{characterize-root-list-p } p))) . z\text{-}R\ I\ s * \text{rat-of-nat} (\text{card} \{x. \text{poly } p\ x = 0 \wedge \text{consistent-sign-vec } qs\ x = s\})$
 $= (\sum s \in \text{set } (\text{remdups } (\text{map } (\text{consistent-sign-vec } qs) (\text{characterize-root-list-p } p)))) . z\text{-}R\ I\ s * \text{rat-of-nat} (\text{card} \{x. \text{poly } p\ x = 0 \wedge \text{consistent-sign-vec } qs\ x = s\})$
using *sum-list-distinct-conv-sum-set* **by** *blast*
have *set-sum-eq*: $(\sum s \in (\text{set } (signs) \cap (\text{consistent-sign-vec } qs \text{ ' set } (\text{characterize-root-list-p } p)))) . z\text{-}R\ I\ s * \text{rat-of-nat} (\text{card} \{x. \text{poly } p\ x = 0 \wedge \text{consistent-sign-vec } qs\ x = s\})$
 $=$
 $(\sum s \in \text{set } (\text{remdups } (\text{map } (\text{consistent-sign-vec } qs) (\text{characterize-root-list-p } p)))) . z\text{-}R\ I\ s * \text{rat-of-nat} (\text{card} \{x. \text{poly } p\ x = 0 \wedge \text{consistent-sign-vec } qs\ x = s\})$
using *set-eq* **by** *auto*
then have $(\sum s \leftarrow \text{signs} . z\text{-}R\ I\ s * \text{rat-of-nat} (\text{card} \{x. \text{poly } p\ x = 0 \wedge \text{consistent-sign-vec } qs\ x = s\})) =$
 $(\sum s \leftarrow \text{remdups } (\text{map } (\text{consistent-sign-vec } qs) (\text{characterize-root-list-p } p))) . z\text{-}R\ I\ s * \text{rat-of-nat} (\text{card} \{x. \text{poly } p\ x = 0$

\wedge *consistent-sign-vec* qs $x = s$))
using *set-sum-eq hyp1 hyp2*
using $\langle (\sum s \in \text{set } \text{signs}. z\text{-}R\ I\ s * \text{rat-of-nat} (\text{card } \{x. \text{poly } p\ x = 0 \wedge \text{consistent-sign-vec } qs\ x = s\})) = (\sum s \in \text{set } \text{signs} \cap \text{consistent-sign-vec } qs \text{ 'set (characterize-root-list-p } p). z\text{-}R\ I\ s * \text{rat-of-nat} (\text{card } \{x. \text{poly } p\ x = 0 \wedge \text{consistent-sign-vec } qs\ x = s\})) \rangle$
by *linarith*
then have *consistent-sign-vec* qs 'set (characterize-root-list-p p) \subseteq set *signs* \implies
 $(\wedge p$ *qss*.
characterize-consistent-signs-at-roots p *qss* =
remdups (*map* (*consistent-sign-vec* *qss*) (*characterize-root-list-p* p))) \implies
 $(\sum s \leftarrow \text{signs}. z\text{-}R\ I\ s * \text{rat-of-nat} (\text{card } \{x. \text{poly } p\ x = 0 \wedge \text{consistent-sign-vec } qs\ x = s\})) =$
 $(\sum s \leftarrow \text{remdups}$
 $(\text{map } (\text{consistent-sign-vec } qs)$
 $(\text{characterize-root-list-p } p)). z\text{-}R\ I\ s * \text{rat-of-nat} (\text{card } \{x. \text{poly } p\ x = 0$
 $\wedge \text{consistent-sign-vec } qs\ x = s\}))$
by *linarith*
then show *?thesis unfolding lhs-dot-rewrite[OF nonzero]*
apply (*auto intro!*: *sum-list-distinct-filter simp add: distinct-signs consistent-sign-vec-def characterize-consistent-signs-at-roots-def*)
using *all-info consistent-sign-vec-def characterize-consistent-signs-at-roots-def*
by (*smt (z3) list.set-map map-eq-conv o-apply set-remdups signs-at-def squash-def*)

qed

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 (*vec-of-list (mtx-row-R* $signs$ I) \cdot (*construct-lhs-vector-R* p qs $signs$)) =
 $\text{rat-of-int } (\text{card } \{x. \text{poly } p\ x = 0 \wedge (\forall p \in \text{set } (\text{retrieve-polys } qs\ (\text{fst } I)). \text{poly } p\ x = 0) \wedge \text{poly } (\text{prod-list } (\text{retrieve-polys } qs\ (\text{snd } I)))\ x > 0\}) -$
 $\text{rat-of-int } (\text{card } \{x. \text{poly } p\ x = 0 \wedge (\forall p \in \text{set } (\text{retrieve-polys } qs\ (\text{fst } I)). \text{poly } p\ x = 0) \wedge \text{poly } (\text{prod-list } (\text{retrieve-polys } qs\ (\text{snd } I)))\ x < 0\})$
proof –
have *finset*: *finite (set (map (consistent-sign-vec* qs) (*characterize-root-list-p* p)))
by *auto*
let *?gt* = (*set (map (consistent-sign-vec* qs) (*characterize-root-list-p* p)) \cap $\{s. z\text{-}R\ I\ s = 1\}$)
let *?lt* = (*set (map (consistent-sign-vec* qs) (*characterize-root-list-p* p)) \cap $\{s. z\text{-}R\ I\ s = -1\}$)
let *?zer* = (*set (map (consistent-sign-vec* qs) (*characterize-root-list-p* p)) \cap $\{s. z\text{-}R\ I\ s = 0\}$)

have $eq: set (map (consistent-sign-vec qs) (characterize-root-list-p p)) = (?gt \cup ?lt) \cup ?zer$
proof safe
fix x
assume $h: x \in set (map (consistent-sign-vec qs) (characterize-root-list-p p))$
 $z-R I x \neq 0 \ z-R I x \neq -1$
then have $x \in set (characterize-consistent-signs-at-roots p qs)$
unfolding $characterize-consistent-signs-at-roots-def$
by ($smt (verit, del-insts) characterize-consistent-signs-at-roots-def characterize-root-list-p-def imageE in-set-R nonzero poly-roots-finite set-map sorted-list-of-set(1)$)
thus $z-R I x = 1$
using $h welldefined1 welldefined2 z-lemma-R$ **by** $blast$
qed
have $sumeq: (\sum s \in (?gt \cup ?lt). z-R I s * rat-of-int (card \{x. poly p x = 0 \wedge consistent-sign-vec qs x = s\}))$
 $= (\sum s \in ?gt. z-R I s * rat-of-int (card \{x. poly p x = 0 \wedge consistent-sign-vec qs x = s\})) +$
 $(\sum s \in ?lt. z-R I s * rat-of-int (card \{x. poly p x = 0 \wedge consistent-sign-vec qs x = s\}))$
apply ($rule sum.union-disjoint$) **by** $auto$

from $construct-lhs-vector-drop-consistent-R[OF assms(1-5)]$ **have**
 $vec-of-list (mtx-row-R signs I) \cdot construct-lhs-vector-R p qs signs =$
 $vec-of-list (mtx-row-R (characterize-consistent-signs-at-roots p qs) I) \cdot$
 $construct-lhs-vector-R p qs (characterize-consistent-signs-at-roots p qs) .$

from $lhs-dot-rewrite[OF assms(1)]$
moreover have $... =$
 $(\sum s \leftarrow characterize-consistent-signs-at-roots p qs.$
 $z-R I s * rat-of-int (card \{x. poly p x = 0 \wedge consistent-sign-vec qs x = s\})) .$
moreover have $... =$
 $(\sum s \in set (map (consistent-sign-vec qs) (characterize-root-list-p p)).$
 $z-R I s * rat-of-int (card \{x. poly p x = 0 \wedge consistent-sign-vec qs x = s\}))$

unfolding $characterize-consistent-signs-at-roots-def sum-code[symmetric]$
apply ($auto$)
by ($smt (verit, best) consistent-sign-vec-def list.set-map map-eq-conv o-apply signs-at-def squash-def sum.set-conv-list$)
moreover have $setc1: ... =$
 $(\sum s \in (?gt \cup ?lt). z-R I s * rat-of-int (card \{x. poly p x = 0 \wedge consistent-sign-vec qs x = s\})) +$
 $(\sum s \in ?zer. z-R I s * rat-of-int (card \{x. poly p x = 0 \wedge consistent-sign-vec qs x = s\}))$
apply ($subst eq$)
apply ($rule sum.union-disjoint$) **by** $auto$
ultimately have $setc: ... =$
 $(\sum s \in ?gt. z-R I s * rat-of-int (card \{x. poly p x = 0 \wedge consistent-sign-vec qs x = s\})) +$
 $(\sum s \in ?lt. z-R I s * rat-of-int (card \{x. poly p x = 0 \wedge consistent-sign-vec qs x = s\})) +$

```

( $\sum s \in ?zer. z-R I s * rat-of-int (card \{x. poly p x = 0 \wedge consistent-sign-vec qs x = s\})$ )
  using sumeq by auto
  have  $\forall s \in ?zer. (z-R I s * rat-of-int (card \{x. poly p x = 0 \wedge consistent-sign-vec qs x = s\})) = 0$ 
  by auto
  then have obvzer:  $(\sum s \in ?zer. z-R I s * rat-of-int (card \{x. poly p x = 0 \wedge consistent-sign-vec qs x = s\})) = 0$ 
  by auto

  have setroots:  $set (characterize-root-list-p p) = \{x. poly p x = 0\}$  unfolding
characterize-root-list-p-def
  using poly-roots-finite nonzero rsquarefree-def set-sorted-list-of-set by blast
  have *:  $\bigwedge s. \{x. poly p x = 0 \wedge consistent-sign-vec qs x = s\} =$ 
 $\{x \in \{x. poly p x = 0\}. consistent-sign-vec qs x = s\}$ 
  by auto
  have e1:  $(\sum s \in consistent-sign-vec qs ' \{x. poly p x = 0\} \cap \{s. z-R I s = 1\}.$ 
 $card \{x. poly p x = 0 \wedge consistent-sign-vec qs x = s\} =$ 
 $(sum (\lambda x. if (\forall p \in set (retrieve-polys qs (fst I)). poly p x = 0) \wedge (poly (prod-list$ 
 $(retrieve-polys qs (snd I))) x) > 0 then 1 else 0) \{x. poly p x = 0\})$ 
  unfolding * apply (rule sum-multicount-gen)
  using <finite (set (map (consistent-sign-vec qs) (characterize-root-list-p p)))>
setroots apply auto[1]
  apply (metis List.finite-set setroots)
  proof safe
    fix x
    assume rt:  $poly p x = 0$ 
    then have 1:  $\{s \in consistent-sign-vec qs ' \{x. poly p x = 0\} \cap \{s. z-R I s =$ 
 $1\}. consistent-sign-vec qs x = s\} =$ 
 $\{s. z-R I s = 1 \wedge consistent-sign-vec qs x = s\}$ 
    by auto
    have 2:  $... = \{s. (\forall p \in set (retrieve-polys qs (fst I)). poly p x = 0) \wedge (0 <$ 
 $poly (prod-list (retrieve-polys qs (snd I))) x) \wedge consistent-sign-vec qs x = s\}$ 
    using horiz-vector-helper-pos-R assms welldefined1 welldefined2 rt by blast
    have 3:  $... = (if (\forall p \in set (retrieve-polys qs (fst I)). poly p x = 0) \wedge (0 < poly$ 
 $(prod-list (retrieve-polys qs (snd I))) x) then \{consistent-sign-vec qs x\} else \{\})$ 
    by auto
    then have  $card \{s \in consistent-sign-vec qs ' \{x. poly p x = 0\} \cap \{s. z-R I s =$ 
 $1\}. consistent-sign-vec qs x = s\} =$ 
 $(if ((\forall p \in set (retrieve-polys qs (fst I)). poly p x = 0) \wedge 0 < poly (prod-list$ 
 $(retrieve-polys qs (snd I))) x)$ 
 $then 1 else 0)$  using 1 2 3 by auto
    thus card
 $\{s \in consistent-sign-vec qs ' \{x. poly p x = 0\} \cap \{s. z-R I s = 1\}.$ 
 $consistent-sign-vec qs x = s\} =$ 
 $(if (\forall p \in set (retrieve-polys qs (fst I)). poly p x = 0) \wedge$ 
 $0 < poly (prod-list (retrieve-polys qs (snd I))) x$ 
 $then 1 else 0)$ 
    by auto

```

qed

have *gtchr*: $(\sum s \in ?gt. z-R I s * \text{rat-of-int} (\text{card} \{x. \text{poly } p x = 0 \wedge \text{consistent-sign-vec } qs x = s\})) =$
 $\text{rat-of-int} (\text{card} \{x. \text{poly } p x = 0 \wedge (\forall p \in \text{set} (\text{retrieve-polys } qs (fst I)). \text{poly } p x = 0) \wedge 0 < \text{poly} (\text{prod-list} (\text{retrieve-polys } qs (snd I))) x\})$
apply (*auto simp add: setroots*)
apply (*subst of-nat-sum[symmetric]*)
apply (*subst of-nat-eq-iff*)
apply (*subst e1*)
apply (*subst card-eq-sum*)
apply (*rule sum.mono-neutral-cong-right*)
apply (*metis List.finite-set setroots*)
by auto
have *e2*: $(\sum s \in \text{consistent-sign-vec } qs \text{ ' } \{x. \text{poly } p x = 0\} \cap \{s. z-R I s = -1\}.$
 $\text{card} \{x. \text{poly } p x = 0 \wedge \text{consistent-sign-vec } qs x = s\}) =$
 $(\text{sum} (\lambda x. \text{if } (\forall p \in \text{set} (\text{retrieve-polys } qs (fst I)). \text{poly } p x = 0) \wedge (\text{poly} (\text{prod-list} (\text{retrieve-polys } qs (snd I))) x) < 0 \text{ then } 1 \text{ else } 0) \{x. \text{poly } p x = 0\})$
unfolding * **apply** (*rule sum-multicount-gen*)
using *finite* (*set* (*map* (*consistent-sign-vec* *qs*) (*characterize-root-list-p* *p*)))
setroots **apply** *auto[1]*
apply (*metis List.finite-set setroots*)
proof safe
fix *x*
assume *rt*: $\text{poly } p x = 0$
then have *1*: $\{s \in \text{consistent-sign-vec } qs \text{ ' } \{x. \text{poly } p x = 0\} \cap \{s. z-R I s = -1\}.$
 $\text{consistent-sign-vec } qs x = s\} =$
 $\{s. z-R I s = -1 \wedge \text{consistent-sign-vec } qs x = s\}$
by auto
have *2*: $\dots = \{s. ((\forall p \in \text{set} (\text{retrieve-polys } qs (fst I)). \text{poly } p x = 0) \wedge 0 > \text{poly} (\text{prod-list} (\text{retrieve-polys } qs (snd I))) x) \wedge \text{consistent-sign-vec } qs x = s\}$
using *horiz-vector-helper-neg-R* *assms* *rt* *welldefined1* *welldefined2* **by** *blast*
have *3*: $\dots = (\text{if } (\forall p \in \text{set} (\text{retrieve-polys } qs (fst I)). \text{poly } p x = 0) \wedge (0 > \text{poly} (\text{prod-list} (\text{retrieve-polys } qs (snd I))) x) \text{ then } \{\text{consistent-sign-vec } qs x\} \text{ else } \{\})$
by auto
thus $\text{card} \{s \in \text{consistent-sign-vec } qs \text{ ' } \{x. \text{poly } p x = 0\} \cap \{s. z-R I s = -1\}.$
 $\text{consistent-sign-vec } qs x = s\} =$
 $(\text{if } (\forall p \in \text{set} (\text{retrieve-polys } qs (fst I)). \text{poly } p x = 0) \wedge 0 > \text{poly} (\text{prod-list} (\text{retrieve-polys } qs (snd I))) x$
 $\text{then } 1 \text{ else } 0) \text{ using } 1\ 2\ 3 \text{ by auto}$

qed

have *ltchr*: $(\sum s \in ?lt. z-R I s * \text{rat-of-int} (\text{card} \{x. \text{poly } p x = 0 \wedge \text{consistent-sign-vec } qs x = s\})) =$
 $\text{rat-of-int} (\text{card} \{x. \text{poly } p x = 0 \wedge (\forall p \in \text{set} (\text{retrieve-polys } qs (fst I)). \text{poly } p x = 0) \wedge 0 > \text{poly} (\text{prod-list} (\text{retrieve-polys } qs (snd I))) x\})$
apply (*auto simp add: setroots sum-negf*)
apply (*subst of-nat-sum[symmetric]*)

```

apply (subst of-nat-eq-iff)
apply (subst e2)
apply (subst card-eq-sum)
apply (rule sum.mono-neutral-cong-right)
apply (metis List.finite-set setroots)
by auto
show ?thesis using gtchr ltchr obvzer setc
using ⟨(∑ s←characterize-consistent-signs-at-roots p qs. z-R I s * rat-of-int
(int (card {x. poly p x = 0 ∧ consistent-sign-vec qs x = s}))) = (∑ s∈set (map
(consistent-sign-vec qs) (characterize-root-list-p p)). z-R I s * rat-of-int (int (card
{x. poly p x = 0 ∧ consistent-sign-vec qs x = s})))⟩ ⟨vec-of-list (mtx-row-R (characterize-consistent-signs-at-roots
p qs) I) · construct-lhs-vector-R p qs (characterize-consistent-signs-at-roots p qs) =
(∑ s←characterize-consistent-signs-at-roots p qs. z-R I s * rat-of-int (int (card {x.
poly p x = 0 ∧ consistent-sign-vec qs x = s})))⟩ ⟨vec-of-list (mtx-row-R signs I) ·
construct-lhs-vector-R p qs signs = vec-of-list (mtx-row-R (characterize-consistent-signs-at-roots
p qs) I) · construct-lhs-vector-R p qs (characterize-consistent-signs-at-roots p qs)⟩
setc1 by linarith
qed

```

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≠0
assumes distinct-signs: distinct signs
assumes all-info: set (characterize-consistent-signs-at-roots p qs) ⊆ set(signs)
assumes welldefined1: list-constr (fst I) (length qs)
assumes welldefined2: list-constr (snd I) (length qs)
shows (vec-of-list (mtx-row-R signs I) ·
(construct-lhs-vector-R p qs signs)) =
construct-NofI-R p (retrieve-polys qs (fst I)) (retrieve-polys qs (snd I))

```

proof –

```

show ?thesis
unfolding construct-NofI-prop-R[OF nonzero]
using matrix-equation-helper-step-R[OF assms]
by linarith

```

qed

lemma *mtx-row-length-R:*

```

list-all (λr. length r = length signs) (map (mtx-row-R signs) ls)
apply (induction ls)
by (auto simp add: mtx-row-R-def)

```

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* $p \text{ } qs \text{ } \text{subsets } \text{signs}$
unfolding *satisfy-equation-R-def matrix-A-R-def*
construct-lhs-vector-R-def construct-rhs-vector-R-def all-list-constr-R-def
apply (*subst mult-mat-vec-of-list*)
apply (*auto simp add: mtx-row-length-R intro!: map-vec-vec-of-list-eq-intro*)
using *matrix-equation-main-step-R[OF assms(1-3), unfolded construct-lhs-vector-R-def]*
using *all-list-constr-R-def welldefined* **by** *fastforce*

lemma *consistent-signs-at-roots-eq*:
assumes $p \neq 0$
shows *consistent-signs-at-roots* $p \text{ } qs =$
set (characterize-consistent-signs-at-roots } p \text{ } qs)
unfolding *consistent-signs-at-roots-def characterize-consistent-signs-at-roots-def*
characterize-root-list-p-def
apply *auto*
apply (*subst set-sorted-list-of-set*)
using *assms poly-roots-finite* **apply** *blast*
unfolding *sgn-vec-def sgn-def signs-at-def squash-def o-def*
using *roots-def* **apply** *auto[1]*
by (*smt Collect-cong assms image-iff poly-roots-finite roots-def sorted-list-of-set(1)*)

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 $\bigwedge a \ b \ i. (a, b) \in \text{set}(\text{subsets}) \implies (i \in \text{set } a \vee i \in \text{set } b) \implies i <$
length qs
shows *M-mat-R signs subsets* $*_v$ *w-vec-R* $p \text{ } qs \text{ } \text{signs} = \text{v-vec-R } p \text{ } qs \text{ } \text{subsets}$
unfolding *satisfy-equation-R-def[symmetric]*
using *matrix-equation-R[of p signs qs subsets] assms*
using *consistent-signs-at-roots-eq* **unfolding** *all-list-constr-R-def list-constr-def*

apply (*auto*)
by (*metis (no-types, lifting) Ball-set*)

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

lemma *mat-base-case-R*:

shows *matrix-A-R* $[[1],[0],[-1]]$ $[([], []),([0], []),([], [0])]$ = (*mat-of-rows-list* 3
 $[[1,1,1], [0,1,0], [1,0,-1]]$)
unfolding *matrix-A-R-def mtr-row-R-def z-R-def map-sgas-def* **apply** (*auto*)
by (*simp add: numeral-3-eq-3*)

lemma *base-case-sgas-R*:

fixes *q p*:: *real poly*
assumes *nonzero*: *p* \neq 0
shows *set (characterize-consistent-signs-at-roots p [q])* \subseteq $\{[1],[0], [- 1]\}$
unfolding *characterize-consistent-signs-at-roots-def signs-at-def* **apply** (*auto*)
by (*meson squash-def*)

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

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

proof –

have *ex-q*: $\exists (q::\text{real poly}). qs = [q]$
using *len1*
using *length-Suc-conv[of qs 0]* **by** *auto*
then show *?thesis* **using** *base-case-sgas-R nonzero*
by *auto*

qed

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

fixes *q p*:: *real poly*
assumes *nonzero*: *p* \neq 0
shows *satisfy-equation-R p [q]* $[([], []),([0], []),([], [0])]$ $[[1],[0],[-1]]$

proof –

have *h1*: *set (characterize-consistent-signs-at-roots p [q])* \subseteq $\{[1], [0],[- 1]\}$
using *base-case-sgas-R assms* **by** *auto*

have $h2$: *all-list-constr-R* $[(\ [], \ []), ([0], \ []), (\ [], [0])]$ (*Suc 0*) **unfolding** *all-list-constr-R-def*
by (*simp add: list-constr-def*)
then show *?thesis* **using** *matrix-equation-R* [**where** $p = p$, **where** $qs = [q]$,
where $signs = [[1], [0], [-1]]$, **where** $subsets = [(\ [], \ []), ([0], \ []), (\ [], [0])]$]
nonzero h1 h2 **by** *auto*
qed

lemma *base-case-satisfy-equation-alt-R*:
fixes p :: *real poly*
fixes qs :: *real poly list*
assumes $len1$: *length qs = 1*
assumes $nonzero$: $p \neq 0$
shows *satisfy-equation-R* p qs $[(\ [], \ []), ([0], \ []), (\ [], [0])]$ $[[1], [0], [-1]]$
proof –
have $ex-q$: $\exists (q::\text{real poly}). qs = [q]$
using $len1$
using *length-Suc-conv*[of qs 0] **by** *auto*
then show *?thesis* **using** *base-case-satisfy-equation-R nonzero*
by *auto*
qed

lemma *base-case-matrix-eq*:
fixes q p :: *real poly*
assumes $nonzero$: $p \neq 0$
shows (*mult-mat-vec* (*mat-of-rows-list* 3 $[[1, 1, 1], [0, 1, 0], [1, 0, -1]]$) (*construct-lhs-vector-R*
 p $[q]$ $[[1], [0], [-1]]$) =
(*construct-rhs-vector-R* p $[q]$ $[(\ [], \ []), ([0], \ []), (\ [], [0])]$)
using *mat-base-case-R base-case-satisfy-equation-R* **unfolding** *satisfy-equation-R-def*
by (*simp add: nonzero*)

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

lemma *inverse-mat-base-case-R*:
shows *inverts-mat* (*mat-of-rows-list* 3 $[[1/2, -1/2, 1/2], [0, 1, 0], [1/2, -1/2, -1/2]]::\text{rat mat}$) (*mat-of-rows-list* 3 $[[1, 1, 1], [0, 1, 0], [1, 0, -1]]::\text{rat mat}$)
unfolding *inverts-mat-def mat-of-rows-list-def mat-eq-iff* **apply** *auto*
unfolding *less-three* **by** (*auto simp add: scalar-prod-def*)

lemma *inverse-mat-base-case-2-R*:
shows *inverts-mat* (*mat-of-rows-list* 3 $[[1, 1, 1], [0, 1, 0], [1, 0, -1]]::\text{rat mat}$) (*mat-of-rows-list*
3 $[[1/2, -1/2, 1/2], [0, 1, 0], [1/2, -1/2, -1/2]]::\text{rat mat}$)
unfolding *inverts-mat-def mat-of-rows-list-def mat-eq-iff* **apply** *auto*
unfolding *less-three* **by** (*auto simp add: scalar-prod-def*)

lemma *base-case-invertible-mat-R*:
shows *invertible-mat* (*matrix-A-R* $[[1], [0], [-1]]$ $[(\ [], \ []), ([0], \ []), (\ [], [0])]$)
unfolding *invertible-mat-def* **using** *inverse-mat-base-case-R inverse-mat-base-case-2-R*
apply (*auto*)

apply (*simp add: mat-base-case mat-of-rows-list-def*)
using *mat-base-case-R* **by** *auto*

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* ($\lambda l1. \text{map } (\lambda l2. (fst\ l1\ @\ fst\ l2, snd\ l1\ @\ snd\ l2))\ s2$) *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 *clarsimp*

fix *a b*

assume *assum*: (*a, b*) \in *set* (*smash-list-list* *l1 l2*)

show $\exists aa\ ba. (aa, ba) \in set\ l1 \wedge (\exists ab\ bb. (ab, bb) \in set\ l2 \wedge a = aa\ @\ ab \wedge b = ba\ @\ bb)$

using *assum* **unfolding** *smash-list-list-def*

apply (*auto*) **by** *blast*

qed

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 –

let *?fst-component2* = *subsets-first-component-list* *subsets2*

let *?snd-component2* = *subsets-second-component-list* *subsets2*

let *?new-subsets* = *map* ($\lambda I. ((map\ ((+)\ n))\ (fst\ I), (map\ ((+)\ n))\ (snd\ I)))$ *subsets2*

```

have subsets-smash-R n subsets1 subsets2 = smash-list-list subsets1 ?new-subsets

  unfolding subsets-smash-R-def smash-list-list-def apply (auto)
  by (simp add: comp-def)
  then show ?thesis using smash-list-list-property-set[of subsets1 ?new-subsets]
apply (auto)
  using imageE set-map smash-list-list-property-set
  by (smt (z3) prod.exhaust-sel)
qed

```

25.2 Well-defined subsets preserved when smashing

lemma *well-def-step-R*:

```

fixes qs1 qs2 :: real poly list
fixes subsets1 subsets2 :: (nat list*nat list) list
assumes well-def-subsets1: all-list-constr-R (subsets1) (length qs1)
assumes well-def-subsets2: all-list-constr-R (subsets2) (length qs2)
shows all-list-constr-R ((subsets-smash-R (length qs1) subsets1 subsets2))
  (length (qs1 @ qs2))
proof –
  let ?n = (length qs1)
  have h1:  $\forall$  elem.
    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))))))
    using subsets-smash-property-R by blast
  have h2:  $\forall$  elem1 elem2. (elem1  $\in$  set subsets1  $\wedge$  elem2  $\in$  set subsets2)  $\longrightarrow$ 
  list-constr ((fst elem1) @ map ((+) (length qs1)) (fst elem2)) (length (qs1 @ qs2))
  proof clarsimp
    fix elem1 b elem2 ba
    assume e1: (elem1, b)  $\in$  set subsets1
    assume e2: (elem2, ba)  $\in$  set subsets2
    have h1: list-constr elem1 (length qs1 + length qs2)
    proof –
      have h1: list-constr elem1 (length qs1) using e1 well-def-subsets1
      unfolding all-list-constr-R-def by auto
      then show ?thesis unfolding list-constr-def
      by (simp add: list.pred-mono-strong)
    qed
  have h2: list-constr (map ((+) (length qs1)) elem2) (length qs1 + length qs2)
  proof –
    have h1: list-constr elem2 (length qs2) using e2 well-def-subsets2
    unfolding all-list-constr-R-def
    by simp
    then show ?thesis unfolding list-constr-def
    by (simp add: list-all-length)
  qed
  show list-constr (elem1 @ map ((+) (length qs1)) elem2) (length qs1 + length

```

```

qs2)
  using h1 h2 list-constr-append
  by blast
qed
have h3:  $\forall \text{elem1 elem2. } (\text{elem1} \in \text{set subsets1} \wedge \text{elem2} \in \text{set subsets2}) \longrightarrow$ 
list-constr ((snd elem1) @ map ((+) (length qs1)) (snd elem2)) (length (qs1 @
qs2))
proof clarsimp
  fix elem1 b elem2 ba
  assume e1: (b, elem1)  $\in$  set subsets1
  assume e2: (ba, elem2)  $\in$  set subsets2
  have h1: list-constr elem1 (length qs1 + length qs2)
  proof -
    have h1: list-constr elem1 (length qs1) using e1 well-def-subsets1
      unfolding all-list-constr-R-def
      by auto
    then show ?thesis unfolding list-constr-def
      by (simp add: list-pred-mono-strong)
  qed
  have h2: list-constr (map ((+) (length qs1)) elem2) (length qs1 + length qs2)
  proof -
    have h1: list-constr elem2 (length qs2) using e2 well-def-subsets2
      unfolding all-list-constr-R-def
      by simp
    then show ?thesis unfolding list-constr-def
      by (simp add: list-all-length)
  qed
  show list-constr (elem1 @ map ((+) (length qs1)) elem2) (length qs1 + length
qs2)
  using h1 h2 list-constr-append
  by blast
qed
show ?thesis using h1 h2 h3 unfolding all-list-constr-def
  by (metis all-list-constr-R-def fst-conv snd-conv)
qed

```

25.3 Consistent Sign Assignments Preserved When Smashing

```

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

```

```

assume x-in:  $x \in \text{set}(\text{characterize-consistent-signs-at-roots } p \text{ (} qs1 \text{ @ } qs2))$ 
have x-in-csv:  $x \in \text{set}(\text{map}(\text{consistent-sign-vec}(qs1 \text{ @ } qs2))(\text{characterize-root-list-} p))$ 
using x-in unfolding characterize-consistent-signs-at-roots-def
by (smt (z3) consistent-sign-vec-def map-eq-conv o-apply set-remdups signs-at-def squash-def)
have csv-hyp:  $\forall r. \text{consistent-sign-vec}(qs1 \text{ @ } qs2) \ r = (\text{consistent-sign-vec } qs1 \ r) \text{@}(\text{consistent-sign-vec } qs2 \ r)$ 
unfolding consistent-sign-vec-def by auto
have exr-iff:  $(\exists r \in \text{set}(\text{characterize-root-list-} p). \ x = \text{consistent-sign-vec}(qs1 \text{ @ } qs2) \ r) \longleftrightarrow (\exists r \in \text{set}(\text{characterize-root-list-} p). \ x = (\text{consistent-sign-vec } qs1 \ r) \text{@}(\text{consistent-sign-vec } qs2 \ r))$ 
using csv-hyp by auto
have exr:  $x \in \text{set}(\text{map}(\text{consistent-sign-vec}(qs1 \text{ @ } qs2))(\text{characterize-root-list-} p)) \longrightarrow (\exists r \in \text{set}(\text{characterize-root-list-} p). \ x = \text{consistent-sign-vec}(qs1 \text{ @ } qs2) \ r)$ 
by auto
have mem-list1:  $\forall r \in \text{set}(\text{characterize-root-list-} p). \ (\text{consistent-sign-vec } qs1 \ r) \in \text{set}(\text{map}(\text{consistent-sign-vec } qs1)(\text{characterize-root-list-} p))$ 
by simp
have mem-list2:  $\forall r \in \text{set}(\text{characterize-root-list-} p). \ (\text{consistent-sign-vec } qs2 \ r) \in \text{set}(\text{map}(\text{consistent-sign-vec } qs2)(\text{characterize-root-list-} p))$ 
by simp
then show  $\exists x1 \in \text{set}(\text{characterize-consistent-signs-at-roots } p \ qs1). \ \exists x2 \in \text{set}(\text{characterize-consistent-signs-at-roots } p \ qs2). \ x = x1 \text{ @ } x2$ 
using x-in-csv exr mem-list1 mem-list2 characterize-consistent-signs-at-roots-def exr-iff
using imageE image-eqI map-append set-map set-remdups signs-at-def x-in
by auto
qed

```

lemma *subset-step-R*:

```

fixes p:: real poly
fixes qs1 qs2 :: real poly list
fixes signs1 signs2 :: rat list list
assumes csa1:  $\text{set}(\text{characterize-consistent-signs-at-roots } p \ qs1) \subseteq \text{set}(\text{signs1})$ 
assumes csa2:  $\text{set}(\text{characterize-consistent-signs-at-roots } p \ qs2) \subseteq \text{set}(\text{signs2})$ 
shows  $\text{set}(\text{characterize-consistent-signs-at-roots } p \ (qs1 \text{ @ } qs2)) \subseteq \text{set}(\text{signs-smash } \text{signs1 } \text{signs2})$ 
proof –
have h0:  $\forall x \in \text{set}(\text{characterize-consistent-signs-at-roots } p \ (qs1 \text{ @ } qs2)). \ \exists x1 \in \text{set}(\text{characterize-consistent-signs-at-roots } p \ qs1). \ \exists x2 \in \text{set}(\text{characterize-consistent-signs-at-roots } p \ qs2). \ (x = x1 \text{@} x2)$  using subset-helper-R[of p qs1 qs2]
by blast
then have  $\forall x \in \text{set}(\text{characterize-consistent-signs-at-roots } p \ (qs1 \text{ @ } qs2)). \ x \in \text{set}(\text{signs-smash}(\text{characterize-consistent-signs-at-roots } p \ qs1)(\text{characterize-consistent-signs-at-roots } p \ qs2))$ 

```

```

  unfolding signs-smash-def apply (auto)
  by fastforce
then have  $\forall x \in \text{set } (\text{characterize-consistent-signs-at-roots } p$ 
   $(qs1 \text{ @ } qs2)). x \in \text{set } (\text{signs-smash } signs1 \text{ } signs2)$ 
  using  $csa1 \text{ } csa2 \text{ subset-smash-signs[of - signs1 - signs2]$  apply (auto)
  by blast
thus ?thesis
  by blast
qed

```

25.4 Main Results

```

lemma dim-row-matrix-A-R[simp]:
  shows dim-row (matrix-A-R signs subsets) = length subsets
  unfolding matrix-A-R-def by auto

```

```

lemma dim-col-matrix-A-R[simp]:
  shows dim-col (matrix-A-R signs subsets) = length signs
  unfolding matrix-A-R-def by auto

```

```

lemma length-subsets-smash-R:
  shows
    length (subsets-smash-R n subs1 subs2) = length subs1 * length subs2
  unfolding subsets-smash-R-def length-concat
  by (auto simp add: o-def sum-list-triv)

```

```

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

```

proof –

```

  have 1: map (!! (as @ bs)) (fst xs) = map (!! as) (fst xs)
    unfolding list-eq-iff-nth-eq
    using assms by (auto simp add: nth-append)
  have 2: map (!! (as @ bs) o (+) (length as)) (fst ys) = map (!! bs) (fst ys)
    unfolding list-eq-iff-nth-eq
    using assms by auto
  have 3: map (!! (as @ bs)) (snd xs) = map (!! as) (snd xs)
    unfolding list-eq-iff-nth-eq
    using assms by (auto simp add: nth-append)
  have 4: map (!! (as @ bs) o (+) (length as)) (snd ys) = map (!! bs) (snd ys)
    unfolding list-eq-iff-nth-eq
    using assms by auto
  show ?thesis
    unfolding z-R-def apply auto
    unfolding 1 2 3 4 apply (auto)
    by (smt (z3) assms(1) comp-apply length-map map-append map-eq-conv map-sgas-def

```

nth-append nth-append-length-plus)
qed

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 $\bigwedge l i. l \in \text{set } \text{subs1} \implies (i \in \text{set } (\text{fst } l) \vee i \in \text{set } (\text{snd } l)) \implies i < n1$

assumes $\bigwedge j. j \in \text{set } \text{signs1} \implies \text{length } j = n1$

shows $(\text{matrix-A-R } (\text{signs-smash } \text{signs1 } \text{signs2}) (\text{subsets-smash-R } n1 \text{ subs1 } \text{subs2}))$

=

$\text{kronecker-product } (\text{matrix-A-R } \text{signs1 } \text{subs1}) (\text{matrix-A-R } \text{signs2 } \text{subs2})$

unfolding *mat-eq-iff dim-row-matrix-A-R dim-col-matrix-A-R*

length-subsets-smash-R length-signs-smash dim-kronecker

proof *safe*

fix *i j*

assume *i*: $i < \text{length } \text{subs1} * \text{length } \text{subs2}$

assume *j*: $j < \text{length } \text{signs1} * \text{length } \text{signs2}$

have *ld*: $i \text{ div } \text{length } \text{subs2} < \text{length } \text{subs1}$

$j \text{ div } \text{length } \text{signs2} < \text{length } \text{signs1}$

using *i j less-mult-imp-div-less* **by** *auto*

have *lm*: $i \text{ mod } \text{length } \text{subs2} < \text{length } \text{subs2}$

$j \text{ mod } \text{length } \text{signs2} < \text{length } \text{signs2}$

using *i j less-mult-imp-mod-less* **by** *auto*

have *n1*: $n1 = \text{length } (\text{signs1 } ! (j \text{ div } \text{length } \text{signs2}))$

using *assms(2) ld(2) nth-mem* **by** *blast*

have *1*: $\text{matrix-A-R } (\text{signs-smash } \text{signs1 } \text{signs2}) (\text{subsets-smash-R } n1 \text{ subs1 } \text{subs2}) \text{ \$\$ } (i, j) =$

$z\text{-R } (\text{subsets-smash-R } n1 \text{ subs1 } \text{subs2 } ! i) (\text{signs-smash } \text{signs1 } \text{signs2 } ! j)$

unfolding *mat-of-rows-list-def matrix-A-R-def mtx-row-R-def*

using *i j* **by** *(auto simp add: length-signs-smash length-subsets-smash-R)*

have *2*: $\dots = z\text{-R } ((\text{fst } (\text{subs1 } ! (i \text{ div } \text{length } \text{subs2}))) @ \text{map } ((+) n1) (\text{fst } (\text{subs2 } ! (i \text{ mod } \text{length } \text{subs2}))))),$

$(\text{snd } (\text{subs1 } ! (i \text{ div } \text{length } \text{subs2}))) @ \text{map } ((+) n1) (\text{snd } (\text{subs2 } ! (i \text{ mod } \text{length } \text{subs2}))))))$

$(\text{signs1 } ! (j \text{ div } \text{length } \text{signs2}) @ \text{signs2 } ! (j \text{ mod } \text{length } \text{signs2}))$

unfolding *signs-smash-def subsets-smash-R-def*

apply *(subst length-eq-concat)*

using *i* **apply** *(auto simp add: mult.commute)*

apply *(subst length-eq-concat)*

using *j* **apply** *(auto simp add: mult.commute)*

using *ld lm* **by** *auto*

have *3*: $\dots =$

$z\text{-R } (\text{subs1 } ! (i \text{ div } \text{length } \text{subs2})) (\text{signs1 } ! (j \text{ div } \text{length } \text{signs2})) *$

```

z-R (subs2 ! (i mod length subs2)) (signs2 ! (j mod length signs2))
  unfolding n1
  apply (subst z-append-R)
  apply (auto simp add: n1[symmetric])
  using assms(1) ld(1) nth-mem
  apply blast
  using assms(1) ld(1) nth-mem by blast
have 4: kronecker-product (matrix-A-R signs1 subs1) (matrix-A-R signs2 subs2)
  $$ (i,j) =
  z-R (subs1 ! (i div length subs2))
    (signs1 ! (j div length signs2)) *
  z-R (subs2 ! (i mod length subs2))
    (signs2 ! (j mod length signs2))
  unfolding kronecker-product-def matrix-A-R-def mat-of-rows-list-def mtx-row-R-def
  using i j apply (auto simp add: Let-def)
  apply (subst index-mat(1)[OF ld])
  apply (subst index-mat(1)[OF lm])
  using ld lm by auto
  show matrix-A-R (signs-smash signs1 signs2) (subsets-smash-R n1 subs1 subs2)
  $$ (i, j) =
    kronecker-product (matrix-A-R signs1 subs1) (matrix-A-R signs2 subs2) $$
  (i, j)
  using 1 2 3 4 by auto
qed

```

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 ≠ 0
assumes nontriv1: length qs1 > 0
assumes nontriv2: length qs2 > 0
assumes welldefined-signs1: well-def-signs (length qs1) signs1
assumes welldefined-signs2: well-def-signs (length qs2) signs2
assumes distinct-signs1: distinct signs1
assumes distinct-signs2: distinct signs2
assumes all-info1: set (characterize-consistent-signs-at-roots p qs1) ⊆ set(signs1)
assumes all-info2: set (characterize-consistent-signs-at-roots p qs2) ⊆ set(signs2)
assumes welldefined-subsets1: all-list-constr-R (subsets1) (length qs1)
assumes welldefined-subsets2: all-list-constr-R (subsets2) (length qs2)
assumes invertibleMtx1: invertible-mat (matrix-A-R signs1 subsets1)
assumes invertibleMtx2: invertible-mat (matrix-A-R signs2 subsets2)
shows satisfy-equation-R p (qs1@qs2) (subsets-smash-R (length qs1) subsets1
subsets2) (signs-smash signs1 signs2)
  ∧ invertible-mat (matrix-A-R (signs-smash signs1 signs2) (subsets-smash-R
(length qs1) subsets1 subsets2))
proof -

```

```

have h1: invertible-mat (matrix-A-R (signs-smash signs1 signs2) (subsets-smash-R
(length qs1) subsets1 subsets2))
  using matrix-construction-is-kronecker-product-R
    kronecker-invertible invertibleMtx1 invertibleMtx2
    welldefined-subsets1 welldefined-subsets2 unfolding all-list-constr-R-def list-constr-def
using Ball-set well-def-signs-def welldefined-signs1 in-set-conv-nth list-all-length
apply (auto)
by (smt (z3) Ball-set kronecker-invertible)
have h2a: distinct (signs-smash signs1 signs2)
  using distinct-signs1 distinct-signs2 welldefined-signs1 welldefined-signs2 non-
triv1 nontriv2
    distinct-step by auto
have h2c: all-list-constr-R ((subsets-smash-R (length qs1) subsets1 subsets2))
(length (qs1@qs2))
  using well-def-step-R
    welldefined-subsets1 welldefined-subsets2
by blast
have h2d: set (characterize-consistent-signs-at-roots p (qs1@qs2))  $\subseteq$  set(signs-smash
signs1 signs2)
  using subset-step-R all-info1 all-info2
by simp
have h2: satisfy-equation-R p (qs1@qs2) (subsets-smash-R (length qs1) subsets1
subsets2) (signs-smash signs1 signs2)
  using matrix-equation-R[where p=p, where qs=qs1@qs2, where subsets =
subsets-smash-R (length qs1) subsets1 subsets2,
  where signs = signs-smash signs1 signs2]
  h2a h2c h2d apply (auto) using nonzero by blast
show ?thesis using h1 h2 by blast
qed

```

26 Reduction Step Proofs

definition *get-matrix-R*:: (rat mat \times ((nat list*nat list) list \times rat list list)) \Rightarrow rat mat

where *get-matrix-R* data = fst(data)

definition *get-subsets-R*:: (rat mat \times ((nat list*nat list) list \times rat list list)) \Rightarrow (nat list*nat list) list

where *get-subsets-R* data = fst(snd(data))

definition *get-signs-R*:: (rat mat \times ((nat list*nat list) list \times rat list list)) \Rightarrow rat list list

where *get-signs-R* data = snd(snd(data))

definition *reduction-signs-R*:: real poly \Rightarrow real poly list \Rightarrow rat list list \Rightarrow (nat list*nat list) list \Rightarrow rat mat \Rightarrow 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* \Rightarrow *real poly list* \Rightarrow *rat list list* \Rightarrow (*nat list*nat list*) *list* \Rightarrow *rat mat* \Rightarrow (*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))))
unfolding *reduction-signs-R-def get-signs-R-def* **apply** (*simp*)
using *reduction-step-R.elims snd-conv*
by *metis*

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))))*
unfolding *reduction-subsets-R-def get-subsets-R-def*
using *reduce-system.simps reduction-step.elims fst-conv snd-conv*
by (*metis reduce-system-R.simps reduction-step-R.simps*)

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*: $n < \text{length } (\text{take-indices } \text{arb-list } \text{index-list})$
assumes *well-def*: $\forall q. (\text{List.member } \text{index-list } q \longrightarrow q < \text{length } \text{arb-list})$
shows $\exists k < \text{length } \text{arb-list}.$
 (*take-indices arb-list index-list*) ! *n* = *arb-list* ! *k*
using *lest well-def* **unfolding** *take-indices-def* **apply** (*auto*)
by (*metis nth-mem*)

lemma *size-of-mat-R*:
fixes *subsets*:: (*nat list*nat list*) *list*
fixes *signs*:: *rat list list*
shows (*matrix-A-R signs subsets*) \in *carrier-mat (length subsets) (length signs)*
unfolding *matrix-A-R-def carrier-mat-def* **by** *auto*

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*
unfolding *construct-lhs-vector-R-def*
by *simp*

```

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
  unfolding construct-rhs-vector-R-def
  by simp

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
  using invertible-mat unfolding invertible-mat-def
  using size-of-mat-R[of signs subsets] size-of-lhs-R[of p qs signs] size-of-rhs-R[of
p qs subsets]
  by simp

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 –
    have matrix-equation-hyp: (mult-mat-vec (matrix-A-R signs subsets) (construct-lhs-vector-R
p qs signs)) = (construct-rhs-vector-R p qs subsets)
      using match unfolding satisfy-equation-R-def by blast
    then have eqn-hyp: ((matr-option (dim-row (matrix-A-R signs subsets))
      (mat-inverse (matrix-A-R signs subsets))) *v (mult-mat-vec (matrix-A-R signs
subsets) (construct-lhs-vector-R p qs signs))) =
      mult-mat-vec (matr-option (dim-row (matrix-A-R signs subsets))
      (mat-inverse (matrix-A-R signs subsets))) (construct-rhs-vector-R p qs subsets)
      using invertible-mat
    by (simp add: matrix-equation-hyp)
    have match-hyp: length subsets = length signs using same-size-R invertible-mat
  by auto
    then have dim-hyp1: matrix-A-R signs subsets ∈ carrier-mat (length signs)
      (length signs)
      using size-of-mat
      by auto
    then have dim-hyp2: matr-option (dim-row (matrix-A-R signs subsets))
      (mat-inverse (matrix-A-R signs subsets)) ∈ carrier-mat (length signs) (length
signs)

```

```

using invertible-mat dim-invertible
by blast
have mult-assoc-hyp: (((matr-option (dim-row (matrix-A-R signs subsets)))
  (mat-inverse (matrix-A-R signs subsets))) *v (mult-mat-vec (matrix-A-R signs
subsets) (construct-lhs-vector-R p qs signs)))
  = (((matr-option (dim-row (matrix-A-R signs subsets)))
  (mat-inverse (matrix-A-R signs subsets))) * (matrix-A-R signs subsets)) *v
(construct-lhs-vector-R p qs signs))
using mult-assoc dim-hyp1 dim-hyp2 size-of-lhs-R by auto
have cancel-helper: (((matr-option (dim-row (matrix-A-R signs subsets)))
  (mat-inverse (matrix-A-R signs subsets))) * (matrix-A-R signs subsets)) *v
(construct-lhs-vector-R p qs signs))
  = (1m (length signs)) *v (construct-lhs-vector-R p qs signs)
using invertible-means-mult-id[where A = matrix-A-R signs subsets] dim-hyp1
by (simp add: invertible-mat match-hyp)
then have cancel-hyp: (((matr-option (dim-row (matrix-A-R signs subsets)))
  (mat-inverse (matrix-A-R signs subsets))) * (matrix-A-R signs subsets)) *v
(construct-lhs-vector-R p qs signs))
  = (construct-lhs-vector-R p qs signs)
apply (auto)
by (metis carrier-vec-dim-vec one-mult-mat-vec size-of-lhs-R)
then have mult-hyp: (((matr-option (dim-row (matrix-A-R signs subsets)))
  (mat-inverse (matrix-A-R signs subsets))) *v (mult-mat-vec (matrix-A-R signs
subsets) (construct-lhs-vector-R p qs signs)))
  = (construct-lhs-vector-R p qs signs)
using mult-assoc-hyp cancel-hyp
by simp
then have (construct-lhs-vector-R p qs signs) = (mult-mat-vec (matr-option
(dim-row (matrix-A-R signs subsets))
  (mat-inverse (matrix-A-R signs subsets))) (construct-rhs-vector-R p qs subsets))

using eqn-hyp
by simp
then show ?thesis
unfolding solve-for-lhs-R-def
by (simp add: mat-inverse-same)
qed

```

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 ≠ 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 –
have *dim-hyp2*: *matr-option* (*dim-row* (*matrix-A-R signs subsets*))
(mat-inverse (*matrix-A-R signs subsets*)) \in *carrier-mat* (*length signs*) (*length signs*)
using *invertible-mat dim-invertible*
using *same-size-R by fastforce*
have (*construct-lhs-vector-R p qs signs*) = *solve-for-lhs-R p qs subsets* (*matrix-A-R signs subsets*)
using *construct-lhs-matches-solve-for-lhs-R assms by auto*
then have (*solve-for-lhs-R p qs subsets* (*matrix-A-R signs subsets*)) =
vec-of-list (*map rat-of-nat* (*map* (λs . *card* {*x*. *poly p x = 0* \wedge *consistent-sign-vec qs x = s*})) *signs*))
using *construct-lhs-vector-cleaner-R assms*
by (*metis* (*mono-tags, lifting*) *list.map-cong map-map o-apply of-int-of-nat-eq*)
then have $\forall n < (\text{dim-vec } (\text{solve-for-lhs-R } p \text{ qs subsets } (\text{matrix-A-R signs subsets})))$.
 $((\text{solve-for-lhs-R } p \text{ qs subsets } (\text{matrix-A-R signs subsets})) \$ n = 0) \longrightarrow$
 $(\text{nth signs } n) \notin \text{set } (\text{characterize-consistent-signs-at-roots } p \text{ qs})$
proof –
have *h0*: $\forall n < (\text{dim-vec } (\text{solve-for-lhs-R } p \text{ qs subsets } (\text{matrix-A-R signs subsets})))$.
 $((\text{solve-for-lhs-R } p \text{ qs subsets } (\text{matrix-A-R signs subsets})) \$ n = 0) \longrightarrow$
 $\text{rat-of-nat } (\text{card } \{x. \text{poly } p \text{ } x = 0 \wedge \text{consistent-sign-vec } qs \text{ } x = (\text{nth signs } n)\})$
 $= 0$
by (*metis* (*mono-tags, lifting*) \langle *construct-lhs-vector-R p qs signs = solve-for-lhs-R p qs subsets* (*matrix-A-R signs subsets*) \rangle *construct-lhs-vector-clean-R nonzero of-nat-0-eq-iff of-rat-of-nat-eq size-of-lhs-R*)
have *h1*: $\forall w$. (*rat-of-nat* (*card* {*x*. *poly p x = 0* \wedge *consistent-sign-vec qs x = w*}) = 0 \longrightarrow
 $(\nexists x. \text{poly } p \text{ } x = 0 \wedge \text{consistent-sign-vec } qs \text{ } x = w)$)
proof *clarsimp*
fix *x*
assume *card-asm*: *card* {*xa*. *poly p xa = 0* \wedge *consistent-sign-vec qs xa = consistent-sign-vec qs x*} = 0
assume *zero-asm*: *poly p x = 0*
have *card-hyp*: {*xa*. *poly p xa = 0* \wedge *consistent-sign-vec qs xa = consistent-sign-vec qs x*} = {}
using *card-eq-0-iff*
using *card-asm nonzero poly-roots-finite*
by (*metis* (*full-types*) *finite-Collect-conjI*)
have *x* \in {*xa*. *poly p xa = 0* \wedge *consistent-sign-vec qs xa = consistent-sign-vec qs x*}
using *zero-asm by auto*
thus *False* **using** *card-hyp*
by *blast*
qed

```

have h2:  $\bigwedge w. (\text{rat-of-nat } (\text{card } \{x. \text{poly } p \ x = 0 \wedge \text{consistent-sign-vec } qs \ x = w\}) = 0 \implies$ 
  ( $\neg \text{List.member } (\text{characterize-consistent-signs-at-roots } p \ qs) \ w))$ 
proof clarsimp
  fix w
  assume card-asm:  $\text{card } \{x. \text{poly } p \ x = 0 \wedge \text{consistent-sign-vec } qs \ x = w\} = 0$ 
  assume mem-asm:  $w \in \text{set } (\text{characterize-consistent-signs-at-roots } p \ qs)$ 
  have h0:  $\nexists x. \text{poly } p \ x = 0 \wedge \text{consistent-sign-vec } qs \ x = w$  using h1 card-asm
    by (simp add: h1)
  have h1:  $\exists x. \text{poly } p \ x = 0 \wedge \text{consistent-sign-vec } qs \ x = w$  using mem-asm
  unfolding characterize-consistent-signs-at-roots-def characterize-root-list-p-def
  proof -
    have  $w \in \text{Collect } (\text{List.member } (\text{remdups } (\text{map } (\text{consistent-sign-vec } qs)$ 
  ( $\text{sorted-list-of-set } \{r. \text{poly } p \ r = 0\})))$ 
    using mem-asm poly-roots-finite [of p]  $\langle p \neq 0 \rangle$ 
    by (auto simp add: characterize-consistent-signs-at-roots-def characterize-root-list-p-def
  consistent-sign-vec-def squash-def signs-at-def image-iff)
    then have f1:  $w \in \text{set } (\text{map } (\text{consistent-sign-vec } qs) (\text{sorted-list-of-set } \{r.$ 
   $\text{poly } p \ r = 0\}))$ 
    by simp
    have finite  $\{r. \text{poly } p \ r = 0\}$ 
    using nonzero poly-roots-finite by blast
    then show ?thesis
    using f1 by auto
  qed
  show False using h0 h1 by auto
  qed
  then have h3:  $\forall w. \text{rat-of-nat } (\text{card } \{x. \text{poly } p \ x = 0 \wedge \text{consistent-sign-vec } qs$ 
   $x = w\}) = 0 \implies$ 
   $w \notin \text{set } (\text{characterize-consistent-signs-at-roots } p \ qs)$ 
  by simp
  show ?thesis using h0 h3
  by blast
  qed
  then have  $\text{set } (\text{characterize-consistent-signs-at-roots } p \ qs) \subseteq \text{set } (\text{take-indices}$ 
  signs
  ( $\text{find-nonzeros-from-input-vec } (\text{solve-for-lhs-R } p \ qs \ \text{subsets } (\text{matrix-A-R}$ 
  signs subsets))))
  using all-info
  reduction-signs-set-helper-lemma[where  $A = \text{set } (\text{characterize-consistent-signs-at-roots}$ 
   $p \ qs)$ , where  $B = \text{signs}$ ,
  where  $C = (\text{solve-for-lhs-R } p \ qs \ \text{subsets } (\text{matrix-A-R} \ \text{signs} \ \text{subsets}))]$ 
  using dim-hyp2 solve-for-lhs-R-def by (simp add: mat-inverse-same)
  then show ?thesis unfolding reduction-signs-R-def by auto
  qed

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:  $p \neq 0$ 
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)  $\subseteq$  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 –
  have dim-hyp2: matr-option (dim-row (matrix-A-R signs subsets))
    (mat-inverse (matrix-A-R signs subsets))  $\in$  carrier-mat (length signs) (length
signs)
    using invertible-mat dim-invertible
    using same-size-R by fastforce
  have supset: set (characterize-consistent-signs-at-roots p qs)  $\supseteq$  set(reduction-signs-R
p qs signs subsets (matrix-A-R signs subsets))
  proof –
    have (construct-lhs-vector-R p qs signs) = solve-for-lhs-R p qs subsets (matrix-A-R
signs subsets)
      using construct-lhs-matches-solve-for-lhs-R assms by auto
    then have (solve-for-lhs-R p qs subsets (matrix-A-R signs subsets)) =
      vec-of-list (map rat-of-nat (map ( $\lambda s$ . card { $x$ . poly p x = 0  $\wedge$  consistent-sign-vec
qs x = s}) signs))
      using construct-lhs-vector-cleaner-R assms
    by (metis (mono-tags, lifting) list.map-cong map-map o-apply of-int-of-nat-eq)
    then have  $\forall n < (\text{dim-vec } (\text{solve-for-lhs-R } p \text{ qs subsets } (\text{matrix-A-R signs
subsets})))$ .
      ( $((\text{solve-for-lhs-R } p \text{ qs subsets } (\text{matrix-A-R signs subsets})) \$ n \neq 0) \longrightarrow$ 
      ( $\text{nth signs } n) \in \text{set } (\text{characterize-consistent-signs-at-roots } p \text{ qs})$ )
    proof –
      have h0:  $\forall n < (\text{dim-vec } (\text{solve-for-lhs-R } p \text{ qs subsets } (\text{matrix-A-R signs
subsets})))$ .
        ( $((\text{solve-for-lhs-R } p \text{ qs subsets } (\text{matrix-A-R signs subsets})) \$ n = 0) \longrightarrow$ 
         $\text{rat-of-nat } (\text{card } \{x. \text{poly } p \text{ x} = 0 \wedge \text{consistent-sign-vec } qs \text{ x} = (\text{nth signs }
n)\}) = 0$ )
        by (simp add:  $\langle \text{solve-for-lhs-R } p \text{ qs subsets } (M\text{-mat-R signs subsets}) =$ 
 $\text{vec-of-list } (\text{map rat-of-nat } (\text{map } (\lambda s. \text{card } \{x. \text{poly } p \text{ x} = 0 \wedge \text{consistent-sign-vec}$ 
 $qs \text{ x} = s\}) \text{ signs}) \rangle \text{vec-of-list-index}$ )
      have h1:  $\forall w. (\text{rat-of-nat } (\text{card } \{x. \text{poly } p \text{ x} = 0 \wedge \text{consistent-sign-vec } qs \text{ x} =
w\}) \neq 0 \longrightarrow$ 
        ( $\exists x. \text{poly } p \text{ x} = 0 \wedge \text{consistent-sign-vec } qs \text{ x} = w$ )
    proof clarsimp
      fix w
      assume card-asm:  $0 < \text{card } \{x. \text{poly } p \text{ x} = 0 \wedge \text{consistent-sign-vec } qs \text{ x} =
w\}$ 
      show  $\exists x. \text{poly } p \text{ x} = 0 \wedge \text{consistent-sign-vec } qs \text{ x} = w$ 
        by (metis (mono-tags, lifting) Collect-empty-eq card-asm card-eq-0-iff

```

```

gr-implies-not0)
  qed
  have h2:  $\bigwedge w. (\text{rat-of-nat } (\text{card } \{x. \text{poly } p \ x = 0 \wedge \text{consistent-sign-vec } qs \ x = w\}) \neq 0 \implies$ 
    (List.member (characterize-consistent-signs-at-roots p qs) w))
  proof clarsimp
    fix w
    assume card-asm:  $0 < \text{card } \{x. \text{poly } p \ x = 0 \wedge \text{consistent-sign-vec } qs \ x = w\}$ 
    have h0:  $\exists x. \text{poly } p \ x = 0 \wedge \text{consistent-sign-vec } qs \ x = w$ 
      using card-asm
      by (simp add: h1)
    then show  $w \in \text{set } (\text{characterize-consistent-signs-at-roots } p \ qs)$ 
      unfolding characterize-consistent-signs-at-roots-def
      using nonzero poly-roots-finite characterize-root-list-p-def
      by (smt (verit) characterize-consistent-signs-at-roots-def in-set-R mem-Collect-eq)
    qed
    then have h3:  $\forall w. \text{rat-of-nat } (\text{card } \{x. \text{poly } p \ x = 0 \wedge \text{consistent-sign-vec } qs \ x = w\}) \neq 0 \longrightarrow$ 
       $w \in \text{set } (\text{characterize-consistent-signs-at-roots } p \ qs)$ 
      by simp
    show ?thesis using h0 h3
      by (metis (no-types, lifting)  $\langle \text{solve-for-lhs-R } p \ qs \ \text{subsets } (\text{matrix-A-R signs subsets}) = \text{vec-of-list } (\text{map rat-of-nat } (\text{map } (\lambda s. \text{card } \{x. \text{poly } p \ x = 0 \wedge \text{consistent-sign-vec } qs \ x = s\}) \ \text{signs})) \rangle \ \text{dim-vec-of-list length-map nth-map vec-of-list-index}$ )
    qed
    then have  $\text{set } (\text{take-indices } \text{signs } (\text{find-nonzeros-from-input-vec } (\text{solve-for-lhs-R } p \ qs \ \text{subsets } (\text{matrix-A-R signs subsets})))) \subseteq$ 
       $\text{set } (\text{characterize-consistent-signs-at-roots } p \ qs)$ 
      using all-info reduction-signs-set-helper-lemma2 [where  $A = \text{set } (\text{characterize-consistent-signs-at-roots } p \ qs)$ , where  $B = \text{signs}$ , where  $C = (\text{solve-for-lhs-R } p \ qs \ \text{subsets } (\text{matrix-A-R signs subsets}))$ ]
      using distinct-signs dim-hyp2 solve-for-lhs-R-def
      by (simp add: mat-inverse-same)
    then show ?thesis unfolding reduction-signs-R-def by auto
  qed
  have  $\text{subset: set } (\text{characterize-consistent-signs-at-roots } p \ qs) \subseteq \text{set } (\text{reduction-signs-R } p \ qs \ \text{signs subsets } (\text{matrix-A-R signs subsets}))$ 
    using reduction-doesnt-break-things-signs-R [of  $p \ qs \ \text{signs subsets}$ ] assms
    by blast
  then show ?thesis using supset subset by auto
qed

```

```

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:  $p \neq 0$ 
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)  $\subseteq$  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)
using reduction-deletes-bad-sign-conds-R [of p qs signs subsets] invertible-mat
by (simp-all add: all-info distinct-signs match nonzero get-signs-R-def reduc-
tion-signs-def welldefined-signs1 reduction-signs-R-def Let-def)

```

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:  $p \neq 0$ 
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 –
have poly-type-hyp:  $p \neq 0$  using nonzero by auto
have distinct-signs-hyp: distinct (snd (snd (reduce-system-R p (qs, ((matrix-A-R
signs subsets), (subsets, signs))))))
proof –
let ?sym = (find-nonzeros-from-input-vec (solve-for-lhs-R p qs subsets (matrix-A-R
signs subsets)))
have h1:  $\forall i < \text{length}(\text{take-indices signs ?sym}). \forall j < \text{length}(\text{take-indices signs ?sym}).$ 
 $i \neq j \longrightarrow \text{nth}(\text{take-indices signs ?sym}) i \neq \text{nth}(\text{take-indices signs ?sym}) j$ 
using distinct-signs unfolding take-indices-def
proof clarsimp
fix i
fix j
assume distinct signs
assume  $i < \text{length}$ 
(find-nonzeros-from-input-vec (solve-for-lhs-R p qs subsets (matrix-A-R
signs subsets)))

```

```

assume  $j < \text{length}$ 
  ( $\text{find-nonzeros-from-input-vec}$  ( $\text{solve-for-lhs-R}$   $p$   $qs$   $\text{subsets}$  ( $\text{matrix-A-R}$ 
 $\text{signs}$   $\text{subsets}$ )))
assume  $\text{neq-hyp}: i \neq j$ 
assume  $\text{signs} ! (\text{find-nonzeros-from-input-vec}$  ( $\text{solve-for-lhs-R}$   $p$   $qs$   $\text{subsets}$ 
  ( $\text{matrix-A-R}$   $\text{signs}$   $\text{subsets}$ )) !  $i$ ) =
   $\text{signs} ! (\text{find-nonzeros-from-input-vec}$  ( $\text{solve-for-lhs-R}$   $p$   $qs$   $\text{subsets}$ 
  ( $\text{matrix-A-R}$   $\text{signs}$   $\text{subsets}$ )) !  $j$ )
have  $h1: \text{find-nonzeros-from-input-vec}$  ( $\text{solve-for-lhs-R}$   $p$   $qs$   $\text{subsets}$ 
  ( $\text{matrix-A-R}$   $\text{signs}$   $\text{subsets}$ )) !  $i \neq \text{find-nonzeros-from-input-vec}$ 
( $\text{solve-for-lhs-R}$   $p$   $qs$   $\text{subsets}$ 
  ( $\text{matrix-A-R}$   $\text{signs}$   $\text{subsets}$ )) !  $j$ 
unfolding  $\text{find-nonzeros-from-input-vec-def}$  using  $\text{neq-hyp}$ 
by ( $\text{metis}$   $\langle i < \text{length}$  ( $\text{find-nonzeros-from-input-vec}$  ( $\text{solve-for-lhs-R}$   $p$ 
 $qs$   $\text{subsets}$  ( $\text{matrix-A-R}$   $\text{signs}$   $\text{subsets}$ )))  $\langle j < \text{length}$  ( $\text{find-nonzeros-from-input-vec}$ 
( $\text{solve-for-lhs-R}$   $p$   $qs$   $\text{subsets}$  ( $\text{matrix-A-R}$   $\text{signs}$   $\text{subsets}$ )))  $\rangle$   $\text{distinct-conv-nth}$   $\text{distinct-filter}$ 
 $\text{distinct-upt}$   $\text{find-nonzeros-from-input-vec-def}$ )
then show  $\text{False}$  using  $\text{distinct-signs}$ 
proof –
have  $f1: \forall p$   $ns$   $n. ((n::\text{nat}) \in \{n \in \text{set } ns. p\ n\}) = (n \in \text{set } ns \wedge n \in \text{Collect}$ 
 $p)$ 
by  $\text{simp}$ 
then have  $f2: \text{filter}$  ( $\lambda n. \text{solve-for-lhs-R}$   $p$   $qs$   $\text{subsets}$  ( $\text{matrix-A-R}$   $\text{signs}$ 
 $\text{subsets}$ )  $\$$   $n \neq 0$ ) [ $0..<\text{dim-vec}$  ( $\text{solve-for-lhs-R}$   $p$   $qs$   $\text{subsets}$  ( $\text{matrix-A-R}$   $\text{signs}$ 
 $\text{subsets}$ ))] !  $i \in \text{set}$  [ $0..<\text{length}$   $\text{signs}$ ]
by ( $\text{metis}$  ( $\text{full-types}$ )  $\langle i < \text{length}$  ( $\text{find-nonzeros-from-input-vec}$  ( $\text{solve-for-lhs-R}$ 
 $p$   $qs$   $\text{subsets}$  ( $\text{matrix-A-R}$   $\text{signs}$   $\text{subsets}$ )))  $\rangle$   $\text{construct-lhs-matches-solve-for-lhs-R}$ 
 $\text{find-nonzeros-from-input-vec-def}$   $\text{invertible-mat}$   $\text{match}$   $\text{nth-mem}$   $\text{set-filter}$ 
 $\text{size-of-lhs-R}$ )
have  $\text{filter}$  ( $\lambda n. \text{solve-for-lhs-R}$   $p$   $qs$   $\text{subsets}$  ( $\text{matrix-A-R}$   $\text{signs}$ 
 $\text{subsets}$ )  $\$$   $n \neq 0$ ) [ $0..<\text{dim-vec}$  ( $\text{solve-for-lhs-R}$   $p$   $qs$   $\text{subsets}$  ( $\text{matrix-A-R}$ 
 $\text{signs}$   $\text{subsets}$ ))] !  $j \in \text{set}$  [ $0..<\text{length}$   $\text{signs}$ ]
using  $f1$  by ( $\text{metis}$  ( $\text{full-types}$ )  $\langle j < \text{length}$  ( $\text{find-nonzeros-from-input-vec}$ 
( $\text{solve-for-lhs-R}$   $p$   $qs$   $\text{subsets}$  ( $\text{matrix-A-R}$   $\text{signs}$   $\text{subsets}$ )))  $\rangle$   $\text{construct-lhs-matches-solve-for-lhs-R}$ 
 $\text{find-nonzeros-from-input-vec-def}$   $\text{invertible-mat}$   $\text{match}$   $\text{nth-mem}$   $\text{set-filter}$ 
 $\text{size-of-lhs-R}$ )
then show  $?thesis$ 
using  $f2$  by ( $\text{metis}$   $\langle \text{signs} ! (\text{find-nonzeros-from-input-vec}$  ( $\text{solve-for-lhs-R}$   $p$ 
 $qs$   $\text{subsets}$  ( $\text{matrix-A-R}$   $\text{signs}$   $\text{subsets}$ )) !  $i$ ) =  $\text{signs} ! (\text{find-nonzeros-from-input-vec}$ 
( $\text{solve-for-lhs-R}$   $p$   $qs$   $\text{subsets}$  ( $\text{matrix-A-R}$   $\text{signs}$   $\text{subsets}$ )) !  $j$ )  $\rangle$   $\text{atLeastLessThan-iff}$ 
 $\text{distinct-conv-nth}$   $\text{distinct-signs}$   $\text{find-nonzeros-from-input-vec-def}$   $h1$   $\text{set-upt}$ )
qed
qed
then have  $\text{distinct}$  ( $\text{take-indices}$   $\text{signs}$  ( $\text{find-nonzeros-from-input-vec}$  ( $\text{solve-for-lhs-R}$ 
 $p$   $qs$   $\text{subsets}$  ( $\text{matrix-A-R}$   $\text{signs}$   $\text{subsets}$ ))))
using  $\text{distinct-conv-nth}$  by  $\text{blast}$ 
then show  $?thesis$ 
using  $\text{get-signs-R-def}$   $\text{reduction-signs-R-def}$   $\text{reduction-signs-is-get-signs-R}$  by
 $\text{auto}$ 
qed
have  $\text{all-info-hyp}: \text{set}$  ( $\text{characterize-consistent-signs-at-roots}$   $p$   $qs$ )  $\subseteq \text{set}$ ( $\text{snd}$  ( $\text{snd}$ 

```

```

(reduce-system-R p (qs, ((matrix-A-R signs subsets), (subsets, signs))))))
  using reduce-system-sign-conditions-R assms unfolding get-signs-R-def by
  auto
  have welldefined-hyp: all-list-constr-R (fst (snd (reduce-system-R p (qs, ((matrix-A-R
signs subsets), (subsets, signs)))))) (length qs)
    using welldefined-subsets rows-to-keep-lem
    unfolding all-list-constr-R-def list-constr-def list-all-def
    apply (auto simp add: Let-def take-indices-def take-cols-from-matrix-def)
    using nth-mem
    apply (smt (z3) mat-of-cols-carrier(2) rows-to-keep-lem)
    by (smt (z3) mat-of-cols-carrier(2) nth-mem rows-to-keep-lem)
  then show ?thesis using poly-type-hyp distinct-signs-hyp all-info-hyp wellde-
fined-hyp
    using matrix-equation-R unfolding get-subsets-R-def get-signs-R-def
    by blast
qed

```

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:  $\forall x. \text{List.member } S \ x \longrightarrow x < \text{length signs}$ 
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 –
  have h0a: dim-col (take-cols-from-matrix (alt-matrix-A-R signs subsets) S) =
length (take-indices signs S)
    unfolding take-cols-from-matrix-def apply (auto) unfolding take-indices-def
by auto
  have h0:  $\forall i < \text{length (take-indices signs S)}. (\text{col (alt-matrix-A-R (take-indices
signs S) subsets) } i =$ 
col (take-cols-from-matrix (alt-matrix-A-R signs subsets) S) i)
    proof clarsimp
    fix i
    assume asm:  $i < \text{length (take-indices signs S)}$ 
    have i-lt:  $i < \text{length (map (!) (cols (alt-matrix-A-R signs subsets))) } S$  using
asm
    apply (auto) unfolding take-indices-def by auto
    have h0:  $\text{vec (length subsets) } (\lambda j. z\text{-R (subsets ! } j) (\text{map (!) signs } S ! i)) =$ 
 $\text{vec (length subsets) } (\lambda j. z\text{-R (subsets ! } j) (\text{signs ! (S ! i)})$  using nth-map
    by (metis  $\langle i < \text{length (take-indices signs S)} \rangle$  length-map take-indices-def)
    have dim:  $(\text{map (!) (cols (alt-matrix-A-R signs subsets)) } S) ! i \in \text{carrier-vec}$ 
(dim-row (alt-matrix-A-R signs subsets))
    proof –

```

```

have dim-col (alt-matrix-A-R signs subsets) = length (signs)
  by (simp add: alt-matrix-A-R-def)
have well-d: S ! i < length (signs) using well-def-h
  using i-lt by fastforce
have
  map-eq: (map (!! (cols (alt-matrix-A-R signs subsets))) S) ! i = nth (cols
(alt-matrix-A-R signs subsets)) (S ! i)
  using i-lt by auto
have nth (cols (alt-matrix-A-R signs subsets)) (S ! i) ∈ carrier-vec (dim-row
(alt-matrix-A-R signs subsets))
  using col-dim unfolding cols-def using nth-map well-d
  by (simp add: ⟨dim-col (alt-matrix-A-R signs subsets) = length signs⟩)
then show ?thesis using map-eq by auto
qed
have h1: col (take-cols-from-matrix (alt-matrix-A-R signs subsets) S) i =
  col (mat-of-cols (dim-row (alt-matrix-A-R signs subsets)) (map (!! (cols
(alt-matrix-A-R signs subsets))) S)) i
  by (simp add: take-cols-from-matrix-def take-indices-def)
have h2: col (mat-of-cols (dim-row (alt-matrix-A-R signs subsets)) (map (!!
(cols (alt-matrix-A-R signs subsets))) S)) i
  = nth (map (!! (cols (alt-matrix-A-R signs subsets))) S) i
  using dim i-lt asm col-mat-of-cols[where j = i, where n = (dim-row
(alt-matrix-A-R signs subsets)),
  where vs = (map (!! (cols (alt-matrix-A-R signs subsets))) S)]
  by blast
have h3: col (take-cols-from-matrix (alt-matrix-A-R signs subsets) S) i = (col
(alt-matrix-A-R signs subsets) (S ! i))
  using h1 h2 well-def-h asm by (auto simp add: take-indices-def mat-of-rows-list-def
matrix-A-R-def alt-matrix-char-R)
have vec (length subsets) (λj. z-R (subsets ! j) (signs ! (S ! i))) = (col
(alt-matrix-A-R signs subsets) (S ! i))
  using asm well-def-h by (auto simp add: signs-are-cols-R take-indices-def)
then have vec (length subsets) (λj. z-R (subsets ! j) (take-indices signs S ! i))
=
  col (take-cols-from-matrix (alt-matrix-A-R signs subsets) S) i
  using h0 h3
  by (simp add: take-indices-def)
then show col (alt-matrix-A-R (take-indices signs S) subsets) i =
  col (take-cols-from-matrix (alt-matrix-A-R signs subsets) S) i
  using asm signs-are-cols-R[where signs = (take-indices signs S), where
subsets = subsets]
  by auto
qed
then show ?thesis unfolding alt-matrix-A-R-def take-cols-from-matrix-def
apply (auto)
  using h0a mat-col-eqI
  by (metis alt-matrix-A-R-def dim-col-mat(1) dim-row-mat(1) h0 mat-of-cols-def
take-cols-from-matrix-def)
qed

```

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*: $\forall x. List.member\ S\ x \longrightarrow x < length\ signs$
assumes *nonzero*: $p \neq 0$
shows *matrix-A-R* (*take-indices signs* S) *subsets* = *take-cols-from-matrix* (*matrix-A-R signs subsets*) S
using *reduce-system-matrix-signs-helper-aux-R alt-matrix-char-R assms* **by** *auto*

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*: $length\ subsets \geq length\ signs$
assumes *well-def-h*: $\forall x. List.member\ S\ x \longrightarrow x < length\ subsets$
assumes *nonzero*: $p \neq 0$
shows *alt-matrix-A-R signs* (*take-indices subsets* S) = *take-rows-from-matrix* (*alt-matrix-A-R signs subsets*) S
proof –
have $h0a$: $dim\text{-}row\ (take\text{-}rows\text{-}from\text{-}matrix\ (alt\text{-}matrix\text{-}A\text{-}R\ signs\ subsets)\ S) = length\ (take\text{-}indices\ subsets\ S)$
unfolding *take-rows-from-matrix-def* **apply** (*auto*) **unfolding** *take-indices-def* **by** *auto*
have $h0$: $\forall i < length\ (take\text{-}indices\ subsets\ S). (row\ (alt\text{-}matrix\text{-}A\text{-}R\ signs\ (take\text{-}indices\ subsets\ S))\ i = row\ (take\text{-}rows\text{-}from\text{-}matrix\ (alt\text{-}matrix\text{-}A\text{-}R\ signs\ subsets)\ S)\ i)$
proof *clarsimp*
fix i
assume *asm*: $i < length\ (take\text{-}indices\ subsets\ S)$
have $i\text{-}lt$: $i < length\ (map\ (!)\ (rows\ (alt\text{-}matrix\text{-}A\text{-}R\ signs\ subsets)))\ S)$ **using** *asm*
apply (*auto*) **unfolding** *take-indices-def* **by** *auto*
have $h0$: $row\ (take\text{-}rows\text{-}from\text{-}matrix\ (alt\text{-}matrix\text{-}A\text{-}R\ signs\ subsets)\ S)\ i = row\ (mat\text{-}of\text{-}rows\ (dim\text{-}col\ (alt\text{-}matrix\text{-}A\text{-}R\ signs\ subsets))\ (map\ (!)\ (rows\ (alt\text{-}matrix\text{-}A\text{-}R\ signs\ subsets))))\ S)\ i$
unfolding *take-rows-from-matrix-def take-indices-def* **by** *auto*
have dim : $(map\ (!)\ (rows\ (alt\text{-}matrix\text{-}A\text{-}R\ signs\ subsets)))\ S)\ !\ i \in carrier\text{-}vec\ (dim\text{-}col\ (alt\text{-}matrix\text{-}A\text{-}R\ signs\ subsets))$
proof –
have $dim\text{-}col\ (alt\text{-}matrix\text{-}A\text{-}R\ signs\ subsets) = length\ (signs)$
by (*simp add: alt-matrix-A-R-def*)
then have $lenh$: $dim\text{-}col\ (alt\text{-}matrix\text{-}A\text{-}R\ signs\ subsets) \leq length\ (subsets)$

```

    using assms
    by auto
  have well-d:  $S ! i < \text{length } (\text{subsets})$  using well-def-h
    using i-lt by fastforce
  have
    map-eq:  $(\text{map } (!) (\text{rows } (\text{alt-matrix-A-R signs subsets}))) S ! i = \text{nth } (\text{rows } (\text{alt-matrix-A-R signs subsets})) (S ! i)$ 
    using i-lt by auto
  have  $\text{nth } (\text{rows } (\text{alt-matrix-A-R signs subsets})) (S ! i) \in \text{carrier-vec } (\text{dim-col } (\text{alt-matrix-A-R signs subsets}))$ 
    using col-dim unfolding rows-def using nth-map well-d
    using lenh
    by (simp add: alt-matrix-A-R-def)
  then show ?thesis using map-eq unfolding alt-matrix-A-R-def by auto
qed
  have h1:  $\text{row } (\text{mat-of-rows } (\text{dim-col } (\text{alt-matrix-A-R signs subsets})) (\text{map } (!) (\text{rows } (\text{alt-matrix-A-R signs subsets}))) S)) i = (\text{row } (\text{alt-matrix-A-R signs subsets}) (S ! i))$ 
    using dim i-lt mat-of-rows-row [where  $i = i$ , where  $n = (\text{dim-col } (\text{alt-matrix-A-R signs subsets}))$ ],
    where  $vs = (\text{map } (!) (\text{rows } (\text{alt-matrix-A-R signs subsets}))) S]$ 
    using alt-matrix-char-R nth-mem well-def-h
    by fastforce
  have  $\text{row } (\text{alt-matrix-A-R signs } (\text{take-indices subsets } S)) i = (\text{row } (\text{alt-matrix-A-R signs subsets}) (S ! i))$ 
    using subsets-are-rows-R
  proof –
  have f1:  $i < \text{length } S$ 
    by (metis (no-types) asm length-map take-indices-def)
  then have List.member  $S (S ! i)$ 
    by simp
  then show ?thesis
    using f1
    by (simp add:  $\langle \bigwedge \text{subsets signs. } \forall i < \text{length } \text{subsets. } \text{row } (\text{alt-matrix-A-R signs subsets}) i = \text{vec } (\text{length } \text{signs}) (\lambda j. z\text{-R } (\text{subsets } ! i) (\text{signs } ! j)) \rangle$  take-indices-def well-def-h)
  qed
  then show  $(\text{row } (\text{alt-matrix-A-R signs } (\text{take-indices subsets } S)) i = \text{row } (\text{take-rows-from-matrix } (\text{alt-matrix-A-R signs subsets}) S) i)$ 
    using h0 h1 unfolding take-indices-def by auto
  qed
  then show ?thesis unfolding alt-matrix-A-R-def take-rows-from-matrix-def apply (auto)
    using eq-rowI
    by (metis alt-matrix-A-R-def dim-col-mat(1) dim-row-mat(1) h0 h0a mat-of-rows-def take-rows-from-matrix-def)
  qed

```

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
using *assms* *reduce-system-matrix-subsets-helper-aux-R* *alt-matrix-char-R*
by *auto*

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* ($\text{length } qs$) $signs$
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 –
let $?A$ = (*matrix-A-R* $signs$ $subsets$)
let $?lhs\text{-vec}$ = (*solve-for-lhs-R* p qs $subsets$ (*matrix-A-R* $signs$ $subsets$))
let $?red\text{-mtx}$ = (*take-rows-from-matrix* (*reduce-mat-cols* (*matrix-A-R* $signs$ $subsets$) $?lhs\text{-vec}$))
have $h1$: *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* $signs$ $subsets$), ($subsets$, $signs$))))))
= *matrix-A-R* (*reduction-signs-R* p qs $signs$ $subsets$ (*matrix-A-R* $signs$ $subsets$))
(*reduction-subsets-R* p qs $signs$ $subsets$ (*matrix-A-R* $signs$ $subsets$))
using *reduction-signs-is-get-signs-R* *reduction-subsets-is-get-subsets-R* **by** *auto*
have $h1\text{-var}$: *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* $signs$ $subsets$), ($subsets$, $signs$))))))
= *matrix-A-R* (*take-indices* $signs$ (*find-nonzeros-from-input-vec* $?lhs\text{-vec}$)) (*take-indices* $subsets$ (*rows-to-keep* (*reduce-mat-cols* $?A$ $?lhs\text{-vec}$)))
using $h1$ *reduction-signs-R-def* *reduction-subsets-R-def* **by** *auto*

```

have h2: ?red-mtx = (take-rows-from-matrix (take-cols-from-matrix ?A (find-nonzeros-from-input-vec
?lhs-vec)) (rows-to-keep (take-cols-from-matrix ?A (find-nonzeros-from-input-vec ?lhs-vec))))
  by simp
have h30: (construct-lhs-vector-R p qs signs) = ?lhs-vec
  using assms construct-lhs-matches-solve-for-lhs-R
  by simp
have h3a:  $\forall x. \text{List.member (find-nonzeros-from-input-vec ?lhs-vec) } x \longrightarrow x < \text{length (signs)}$ 
  using h30 size-of-lhs-R[of p qs signs]
  unfolding find-nonzeros-from-input-vec-def by auto
have h3b-a:  $\forall x. \text{List.member (find-nonzeros-from-input-vec ?lhs-vec) } x \longrightarrow x < \text{length (subsets)}$ 
  using assms h30 size-of-lhs-R same-size-R unfolding find-nonzeros-from-input-vec-def
apply (auto)
  by (simp add: find-nonzeros-from-input-vec-def h3a)
have h3ba: length
  (filter ( $\lambda i. \text{solve-for-lhs-R } p \text{ } qs \text{ subsets (matrix-A-R signs subsets) } \$ i \neq 0$ )
  [0.. $\text{length subsets}$ ])
   $\leq \text{length subsets}$  using length-filter-le[where  $P = (\lambda i. \text{solve-for-lhs-R } p \text{ } qs \text{ subsets (matrix-A-R signs subsets) } \$ i \neq 0)$ ,
  where  $xs = [0.. $\text{length subsets}$ ]] \text{length-upto}$  by auto
have length subsets = dim-vec (solve-for-lhs-R p qs subsets (matrix-A-R signs
subsets))
  using h30 inv size-of-lhs-R same-size-R[of signs subsets] apply (auto)
  by metis
then have length
  (filter ( $\lambda i. \text{solve-for-lhs-R } p \text{ } qs \text{ subsets (matrix-A-R signs subsets) } \$ i \neq 0$ )
  [0.. $\text{dim-vec (solve-for-lhs-R } p \text{ } qs \text{ subsets (matrix-A-R signs subsets))}$ ])
   $\leq \text{length subsets}$  using h3ba
  by auto
then have h3b: length subsets  $\geq \text{length (take-indices signs (find-nonzeros-from-input-vec
?lhs-vec))}$ 
  unfolding take-indices-def find-nonzeros-from-input-vec-def by auto
have h3c:  $\forall x. \text{List.member (rows-to-keep (reduce-mat-cols ?A ?lhs-vec)) } x \longrightarrow x < \text{length (subsets)}$ 
proof clarsimp
  fix x
  assume x-mem:  $x \in \text{set (rows-to-keep (take-cols-from-matrix (matrix-A-R signs subsets) (find-nonzeros-from-input-vec (solve-for-lhs-R p qs subsets (matrix-A-R signs subsets))))}$ 
  obtain nn :: rat list list  $\Rightarrow$  nat list  $\Rightarrow$  nat where
     $\forall x2 \ x3. (\exists v4. v4 \in \text{set } x3 \wedge \neg v4 < \text{length } x2) = (nn \ x2 \ x3 \in \text{set } x3 \wedge \neg nn \ x2 \ x3 < \text{length } x2)$ 
  by moura
  then have f4:  $nn \ \text{signs (find-nonzeros-from-input-vec (solve-for-lhs-R } p \text{ } qs \text{ subsets (matrix-A-R signs subsets)))} \in \text{set (find-nonzeros-from-input-vec (solve-for-lhs-R } p \text{ } qs \text{ subsets (matrix-A-R signs subsets)))} \wedge \neg nn \ \text{signs (find-nonzeros-from-input-vec (solve-for-lhs-R } p \text{ } qs \text{ subsets (matrix-A-R signs subsets)))} < \text{length signs} \vee \text{ma-}$ 

```

```

trix-A-R (take-indices signs (find-nonzeros-from-input-vec (solve-for-lhs-R p qs sub-
sets (matrix-A-R signs subsets)))) subsets = take-cols-from-matrix (matrix-A-R
signs subsets) (find-nonzeros-from-input-vec (solve-for-lhs-R p qs subsets (matrix-A-R
signs subsets)))
  using nonzero
  using h3a reduce-system-matrix-signs-helper-R by auto
  then have matrix-A-R (take-indices signs (find-nonzeros-from-input-vec (solve-for-lhs-R
p qs subsets (matrix-A-R signs subsets)))) subsets = take-cols-from-matrix (matrix-A-R
signs subsets) (find-nonzeros-from-input-vec (solve-for-lhs-R p qs subsets (matrix-A-R
signs subsets)))  $\wedge x \in \text{set} (\text{map snd} (\text{pivot-positions} (\text{gauss-jordan-single} (\text{take-cols-from-matrix}
(\text{matrix-A-R signs subsets}) (\text{find-nonzeros-from-input-vec} (\text{solve-for-lhs-R p qs sub-
sets (matrix-A-R signs subsets))))^T)))$ 
    using f4 x-mem h3a by (auto simp add: rows-to-keep-def)
    thus  $x < \text{length} (\text{subsets})$  using x-mem unfolding rows-to-keep-def
    by (metis dim-row-matrix-A-R rows-to-keep-def rows-to-keep-lem)
  qed
  have h3: matrix-A-R (take-indices signs (find-nonzeros-from-input-vec ?lhs-vec))
(take-indices subsets (rows-to-keep (reduce-mat-cols ?A ?lhs-vec))) =
  (take-rows-from-matrix (take-cols-from-matrix ?A (find-nonzeros-from-input-vec
?lhs-vec)) (rows-to-keep (take-cols-from-matrix ?A (find-nonzeros-from-input-vec ?lhs-vec))))

  using inv h3a h3b h3c reduce-system-matrix-subsets-helper-R reduce-system-matrix-signs-helper-R
  assms
  by auto
  show ?thesis using h1 h2 h3
  by (metis fst-conv get-matrix-R-def h1-var reduce-system-R.simps reduction-step-R.simps)
qed

```

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: length subsets = 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: set (characterize-consistent-signs-at-roots p qs)  $\subseteq \text{set}(\text{signs})$ 
assumes match: satisfy-equation-R p qs subsets signs
shows ( $\bigwedge j. j \in \text{set} (\text{find-nonzeros-from-input-vec}
(\text{solve-for-lhs-R p qs subsets (matrix-A-R signs subsets)))) \implies$ 

```

```

      j < length (cols (matrix-A-R signs subsets)))
proof –
  fix i
  fix j
  assume j-in: j ∈ set (find-nonzeros-from-input-vec
    (solve-for-lhs-R p qs subsets (matrix-A-R signs subsets)))
  let ?og-mat = (matrix-A-R signs subsets)
  let ?lhs = (solve-for-lhs-R p qs subsets ?og-mat)
  let ?new-mat = (take-rows-from-matrix (reduce-mat-cols ?og-mat ?lhs) (rows-to-keep
    (reduce-mat-cols ?og-mat ?lhs)))
  have square-mat (matrix-A-R signs subsets) using inv
    using invertible-mat-def by blast
  then have mat-size: ?og-mat ∈ carrier-mat (length signs) (length signs)
    using size-of-mat
    by auto
  have dim-vec (solve-for-lhs-R p qs subsets (matrix-A-R signs subsets)) = (length
    signs)
    using size-of-lhs-R construct-lhs-matches-solve-for-lhs-R assms
    by (metis (full-types))
  then have h: j < (length signs)
    using j-in unfolding find-nonzeros-from-input-vec-def
    by simp
  then show j < length (cols (matrix-A-R signs subsets))
    using mat-size by auto
qed

```

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))
  (find-nonzeros-from-input-vec (solve-for-lhs-R p qs subsets (matrix-A-R
    signs subsets))))
  ⊆ set (cols (matrix-A-R signs subsets))
proof –
  have well-def: (∧j. j ∈ set (find-nonzeros-from-input-vec
    (solve-for-lhs-R p qs subsets (matrix-A-R signs subsets)))) ⇒
    j < length (cols (matrix-A-R signs subsets))
    using assms well-def-find-zeros-from-lhs-vec-R by auto
  have ∀ x. x ∈ set (take-indices (cols (matrix-A-R signs subsets))

```

```

      (find-nonzeros-from-input-vec (solve-for-lhs-R p qs subsets (matrix-A-R
signs subsets))))
    → x ∈ set (cols (matrix-A-R signs subsets))
  proof clarsimp
  fix x
  let ?og-list = (cols (matrix-A-R signs subsets))
  let ?ind-list = (find-nonzeros-from-input-vec
    (solve-for-lhs-R p qs subsets (matrix-A-R signs subsets)))
  assume x-in: x ∈ set (take-indices ?og-list ?ind-list)
  show x ∈ set (cols (matrix-A-R signs subsets))
  using x-in unfolding take-indices-def apply (auto)
  using in-set-conv-nth well-def by fastforce
qed
then show ?thesis
  by blast
qed

```

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)
(solve-for-lhs-R p qs subsets (matrix-A-R signs subsets))) =
  (length (find-nonzeros-from-input-vec (solve-for-lhs-R p qs subsets (matrix-A-R
signs subsets))))
  proof -
  let ?og-mat = (matrix-A-R signs subsets)
  let ?lhs = (solve-for-lhs-R p qs subsets ?og-mat)
  let ?new-mat = (take-rows-from-matrix (reduce-mat-cols ?og-mat ?lhs) (rows-to-keep
(reduce-mat-cols ?og-mat ?lhs)))
  have square-mat (matrix-A-R signs subsets) using inv
  using invertible-mat-def by blast
  then have mat-size: ?og-mat ∈ carrier-mat (length signs) (length signs)
  using size-of-mat
  by auto
  then have mat-size-alt: ?og-mat ∈ carrier-mat (length subsets) (length subsets)
  using size-of-mat same-size assms
  by auto
  have det-h: det ?og-mat ≠ 0
  using invertible-det[where A = matrix-A-R signs subsets] mat-size

```

```

    using inv by blast
  then have rank-h: vec-space.rank (length signs) ?og-mat = (length signs)
    using vec-space.det-rank-iff mat-size
    by auto
  then have dist-cols: distinct (cols ?og-mat) using mat-size vec-space.non-distinct-low-rank[where
  A = ?og-mat, where n = length signs]
    by auto
  have well-def: ( $\bigwedge j. j \in \text{set } (\text{find-nonzeros-from-input-vec }
    (\text{solve-for-lhs-R } p \text{ qs subsets } (\text{matrix-A-R signs subsets}))) \implies
    j < \text{length } (\text{cols } (\text{matrix-A-R signs subsets})))$ )
    using assms well-def-find-zeros-from-lhs-vec-R by auto
  have dist1: distinct
    (find-nonzeros-from-input-vec (solve-for-lhs-R p qs subsets (matrix-A-R signs
  subsets)))
    unfolding find-nonzeros-from-input-vec-def by auto
  have clear: set (take-indices (cols (matrix-A-R signs subsets))
    (find-nonzeros-from-input-vec (solve-for-lhs-R p qs subsets (matrix-A-R
  signs subsets))))
     $\subseteq$  set (cols (matrix-A-R signs subsets))
    using assms take-cols-subsets-og-cols-R by auto
  then have distinct (take-indices (cols (matrix-A-R signs subsets))
    (find-nonzeros-from-input-vec (solve-for-lhs-R p qs subsets (matrix-A-R
  signs subsets))))
    unfolding take-indices-def
    using dist1 dist-cols well-def conjugatable-vec-space.distinct-map-nth[where ls
  = cols (matrix-A-R signs subsets), where inds = (find-nonzeros-from-input-vec
  (solve-for-lhs-R p qs subsets (matrix-A-R signs subsets)))]
    by auto
  then have unfold-thesis: vec-space.rank (length signs) (mat-of-cols (dim-row
  ?og-mat) (take-indices (cols ?og-mat) (find-nonzeros-from-input-vec ?lhs)))
  = (length (find-nonzeros-from-input-vec ?lhs))
    using clear inv conjugatable-vec-space.rank-invertible-subset-cols[where A= ma-
  trix-A-R signs subsets, where B = (take-indices (cols (matrix-A-R signs subsets))
  (find-nonzeros-from-input-vec (solve-for-lhs-R p qs subsets (matrix-A-R signs
  subsets)))))]
    by (simp add: len-eq mat-size take-indices-def)
  then show ?thesis apply (simp) unfolding take-cols-from-matrix-def by auto
qed

```

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  $\neq$  0
  assumes welldefined-signs1: well-def-signs (length qs) signs

```

```

assumes distinct-signs: distinct signs
assumes all-info:  $set (characterize-consistent-signs-at-roots p qs) \subseteq set(signs)$ 
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 –
  let ?og-mat = (matrix-A-R signs subsets)
  let ?lhs = (solve-for-lhs-R p qs subsets ?og-mat)
  let ?step1-mat = (reduce-mat-cols ?og-mat ?lhs)
  let ?new-mat = (take-rows-from-matrix ?step1-mat (rows-to-keep ?step1-mat))
  let ?ind-list = (find-nonzeros-from-input-vec ?lhs)
  have square-mat (matrix-A-R signs subsets) using inv
    using invertible-mat-def by blast
  then have og-mat-size:  $?og-mat \in carrier-mat (length\ signs) (length\ signs)$ 
    using size-of-mat
    by auto
  have dim-col (mat-of-cols (dim-row ?og-mat) (take-indices (cols ?og-mat) ?ind-list))
    = ( $length (find-nonzeros-from-input-vec ?lhs)$ )
    by (simp add: take-indices-def)
  then have mat-of-cols (dim-row ?og-mat) (take-indices (cols ?og-mat) ?ind-list)
     $\in carrier-mat (length\ signs) (length (find-nonzeros-from-input-vec ?lhs))$ 
    by (simp add: len-eq mat-of-cols-def)
  then have step1-mat-size:  $?step1-mat \in carrier-mat (length\ signs) (length (find-nonzeros-from-input-vec ?lhs))$ 
    by (simp add: take-cols-from-matrix-def)
  then have dim-row ?step1-mat  $\geq dim-col\ ?step1-mat$ 
    by (metis carrier-matD(1) carrier-matD(2) construct-lhs-matches-solve-for-lhs-R diff-zero find-nonzeros-from-input-vec-def inv length-filter-le length-upt match size-of-lhs-R)
  then have gt-eq-asm:  $dim-col\ ?step1-mat^T \geq dim-row\ ?step1-mat^T$ 
    by simp
  have det-h:  $det\ ?og-mat \neq 0$ 
    using invertible-det [where  $A = matrix-A-R\ signs\ subsets$ ] og-mat-size
    using inv by blast
  then have rank-h:  $vec-space.rank (length\ signs)\ ?og-mat = (length\ signs)$ 
    using vec-space.det-rank-iff og-mat-size
    by auto
  have rank-drop-cols:  $vec-space.rank (length\ signs)\ ?step1-mat = (dim-col\ ?step1-mat)$ 
    using assms reduction-doesnt-break-things-invertibility-step1-R step1-mat-size
    by auto
  let ?step1-T =  $?step1-mat^T$ 
  have rank-transpose:  $vec-space.rank (length\ signs)\ ?step1-mat = vec-space.rank (length (find-nonzeros-from-input-vec ?lhs))\ ?step1-T$ 
    using transpose-rank [of  $?step1-mat$ ]
    using step1-mat-size by auto
  have obv:  $?step1-T \in carrier-mat (dim-row\ ?step1-T) (dim-col\ ?step1-T)$  by auto
  have should-have-this:  $vec-space.rank (length (find-nonzeros-from-input-vec ?lhs)) (take-cols\ ?step1-T (map\ snd (pivot-positions (gauss-jordan-single (?step1-T)))))) = vec-space.rank (length (find-nonzeros-from-input-vec ?lhs))\ ?step1-T$ 

```

using *obv gt-eq-asm conjugatable-vec-space.gauss-jordan-single-rank* **where** $A = ?step1-T$, **where** $n = \text{dim-row } ?step1-T$, **where** $nc = \text{dim-col } ?step1-T$
by (*simp add: take-cols-from-matrix-def take-indices-def*)
then have $\text{vec-space.rank } (\text{length } (\text{find-nonzeros-from-input-vec } ?lhs)) (\text{take-cols } ?step1-T (\text{map snd } (\text{pivot-positions } (\text{gauss-jordan-single } (?step1-T)))))) = \text{dim-col } ?step1-mat$
using *rank-drop-cols rank-transpose by auto*
then have *with-take-cols-from-matrix: vec-space.rank* ($\text{length } (\text{find-nonzeros-from-input-vec } ?lhs) (\text{take-cols-from-matrix } ?step1-T (\text{map snd } (\text{pivot-positions } (\text{gauss-jordan-single } (?step1-T)))))) = \text{dim-col } ?step1-mat$)
using *rank-transpose rechar-take-cols conjugatable-vec-space.gjs-and-take-cols-var*
apply (*auto*)
by (*smt conjugatable-vec-space.gjs-and-take-cols-var gt-eq-asm obv rechar-take-cols reduce-mat-cols.simps*)
have ($\text{take-rows-from-matrix } ?step1-mat (\text{rows-to-keep } ?step1-mat) = (\text{take-cols-from-matrix } ?step1-T (\text{rows-to-keep } ?step1-mat))^T$)
using *take-rows-and-take-cols*
by *blast*
then have *rank-new-mat: vec-space.rank* ($\text{dim-row } ?new-mat$) $?new-mat = \text{dim-col } ?step1-mat$
using *with-take-cols-from-matrix transpose-rank apply (auto)*
by (*smt (verit, ccfv-threshold) carrier-matD(2) index-transpose-mat(2) mat-of-cols-carrier(2) reduce-mat-cols.simps rows-to-keep-def step1-mat-size take-cols-from-matrix-def transpose-rank*)
have $\text{length } (\text{pivot-positions } (\text{gauss-jordan-single } (?step1-mat^T))) \leq (\text{length } (\text{find-nonzeros-from-input-vec } ?lhs))$
using *obv length-pivot-positions-dim-row* **where** $A = (\text{gauss-jordan-single } (?step1-mat^T))$
by (*metis carrier-matD(1) carrier-matD(2) gauss-jordan-single(2) gauss-jordan-single(3) index-transpose-mat(2) step1-mat-size*)
then have *len-lt-eq: length* ($\text{pivot-positions } (\text{gauss-jordan-single } (?step1-mat^T))) \leq \text{dim-col } ?step1-mat$)
using *step1-mat-size*
by *simp*
have *len-gt-false: length* ($\text{rows-to-keep } ?step1-mat$) $< (\text{dim-col } ?step1-mat) \implies \text{False}$
proof –
assume *length-lt: length* ($\text{rows-to-keep } ?step1-mat$) $< (\text{dim-col } ?step1-mat)$
have *h: dim-row* $?new-mat < (\text{dim-col } ?step1-mat)$
by (*metis Matrix.transpose-transpose index-transpose-mat(3) length-lt length-map mat-of-cols-carrier(3) take-cols-from-matrix-def take-indices-def take-rows-and-take-cols*)
then show *False using rank-new-mat*
by (*metis Matrix.transpose-transpose carrier-matI index-transpose-mat(2) nat-less-le not-less-iff-gr-or-eq transpose-rank vec-space.rank-le-nc*)
qed
then have *len-gt-eq: length* ($\text{rows-to-keep } ?step1-mat$) $\geq (\text{dim-col } ?step1-mat)$
using *not-less by blast*
have *len-match: length* ($\text{rows-to-keep } ?step1-mat$) $= (\text{dim-col } ?step1-mat)$
using *len-lt-eq len-gt-eq*
by (*simp add: rows-to-keep-def*)

have *mat-match*: *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*))))))
using *reduce-system-matrix-match-R*[*of p qs signs subsets*] *assms*
by *blast*
have *f2*: *length* (*get-subsets-R* (*take-rows-from-matrix* (*mat-of-cols* (*dim-row* (*matrix-A-R* *signs* *subsets*)) (*map* (!) (*cols* (*matrix-A-R* *signs* *subsets*))) (*find-nonzeros-from-input-vec* (*solve-for-lhs-R* *p* *qs* *subsets* (*matrix-A-R* *signs* *subsets*)))))) (*rows-to-keep* (*mat-of-cols* (*dim-row* (*matrix-A-R* *signs* *subsets*)) (*map* (!) (*cols* (*matrix-A-R* *signs* *subsets*))) (*find-nonzeros-from-input-vec* (*solve-for-lhs-R* *p* *qs* *subsets* (*matrix-A-R* *signs* *subsets*))))))))) (*map* (!) *subsets*) (*rows-to-keep* (*mat-of-cols* (*dim-row* (*matrix-A-R* *signs* *subsets*)) (*map* (!) (*cols* (*matrix-A-R* *signs* *subsets*))) (*find-nonzeros-from-input-vec* (*solve-for-lhs-R* *p* *qs* *subsets* (*matrix-A-R* *signs* *subsets*))))))))) (*map* (!) (*cols* (*matrix-A-R* *signs* *subsets*))) (*find-nonzeros-from-input-vec* (*solve-for-lhs-R* *p* *qs* *subsets* (*matrix-A-R* *signs* *subsets*)))))) = *length* (*find-nonzeros-from-input-vec* (*solve-for-lhs-R* *p* *qs* *subsets* (*matrix-A-R* *signs* *subsets*)))
by (*metis* (*no-types*) <*dim-col* (*mat-of-cols* (*dim-row* (*matrix-A-R* *signs* *subsets*)) (*take-indices* (*cols* (*matrix-A-R* *signs* *subsets*)) (*find-nonzeros-from-input-vec* (*solve-for-lhs-R* *p* *qs* *subsets* (*matrix-A-R* *signs* *subsets*)))))) = *length* (*find-nonzeros-from-input-vec* (*solve-for-lhs-R* *p* *qs* *subsets* (*matrix-A-R* *signs* *subsets*))) > <*length* (*rows-to-keep* (*reduce-mat-cols* (*matrix-A-R* *signs* *subsets*) (*solve-for-lhs-R* *p* *qs* *subsets* (*matrix-A-R* *signs* *subsets*)))) = *dim-col* (*reduce-mat-cols* (*matrix-A-R* *signs* *subsets*) (*solve-for-lhs-R* *p* *qs* *subsets* (*matrix-A-R* *signs* *subsets*))) > *length-map* *reduce-mat-cols.simps* *reduce-system-R.simps* *reduction-step-R.simps* *reduction-subsets-R-def* *reduction-subsets-is-get-subsets-R* *take-cols-from-matrix-def* *take-indices-def*)
have *f3*: $\forall p$ *ps* *rss* *nss* *m*. *map* (!) *rss* (*find-nonzeros-from-input-vec* (*solve-for-lhs-R* *p* *ps* *nss* *m*)) = *get-signs-R* (*reduce-system-R* *p* (*ps*, *m*, *nss*, *rss*))
by (*metis* (*no-types*) *reduction-signs-R-def* *reduction-signs-is-get-signs-R* *take-indices-def*)
have *square-final-mat*: *square-mat* (*get-matrix-R* (*reduce-system-R* *p* (*qs*, ((*matrix-A-R* *signs* *subsets*), (*subsets*, *signs*))))))
using *mat-match* *assms* *size-of-mat-R* *same-size-R*
apply (*auto*) **using** *f2* *f3*
by (*metis* (*no-types*, *lifting*) *Matrix.transpose-transpose* *fst-conv* *get-matrix-R-def* *index-transpose-mat*(2) *len-match* *length-map* *mat-of-cols-carrier*(2) *mat-of-cols-carrier*(3) *reduce-mat-cols.simps* *take-cols-from-matrix-def* *take-indices-def* *take-rows-and-take-cols*)
have *rank-match*: *vec-space.rank* (*dim-row* ?*new-mat*) ?*new-mat* = *dim-row* ?*new-mat*
using *len-match* *rank-new-mat*
by (*simp* *add*: *take-cols-from-matrix-def* *take-indices-def* *take-rows-and-take-cols*)
have *invertible-mat* ?*new-mat*
using *invertible-det* *og-mat-size*
using *vec-space.det-rank-iff* *square-final-mat*
by (*metis* *dim-col-matrix-A-R* *dim-row-matrix-A-R* *fst-conv* *get-matrix-R-def* *mat-match* *rank-match* *reduce-system-R.simps* *reduction-step-R.simps* *size-of-mat-R* *square-mat.elims*(2))
then show ?*thesis* **apply** (*simp*)

by (*metis fst-conv get-matrix-R-def*)
 qed

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

using *assms*

by (*auto simp add: reduction-signs-def find-nonzeros-from-input-vec-def*
reduction-signs-R-def size-of-lhs-R take-indices-def simp flip: construct-lhs-matches-solve-for-lhs-R)

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 –

have *solve-construct*: *construct-lhs-vector-R p qs signs =*
solve-for-lhs-R p qs subsets (matrix-A-R signs subsets)

using *construct-lhs-matches-solve-for-lhs-R assms*

by *simp*

have *h1*: *distinct (find-nonzeros-from-input-vec (solve-for-lhs-R p qs subsets (matrix-A-R signs subsets)))*

unfolding *find-nonzeros-from-input-vec-def*

using *distinct-filter*

using *distinct-upt* **by** *blast*

have *h2*: $(\bigwedge j. j \in \text{set} (\text{find-nonzeros-from-input-vec} (\text{solve-for-lhs-R } p \text{ } qs \text{ } subsets \text{ } (\text{matrix-A-R } signs \text{ } subsets)))) \implies$
 $j < \text{length } signs$)

proof –

fix *j*

assume $j \in \text{set} (\text{find-nonzeros-from-input-vec} (\text{solve-for-lhs-R } p \text{ } qs \text{ } subsets \text{ } (\text{matrix-A-R } signs \text{ } subsets)))$

show $j < \text{length } signs$ **using** *solve-construct size-of-lhs-R*

by (*metis* $\langle j \in \text{set} (\text{find-nonzeros-from-input-vec} (\text{solve-for-lhs-R } p \text{ } qs \text{ } subsets$

```

(matrix-A-R signs subsets)))› atLeastLessThan-iff filter-is-subset find-nonzeros-from-input-vec-def
set-upt subset-iff)
qed
then show ?thesis unfolding reduction-signs-R-def unfolding take-indices-def
using distinct-init h1 h2 conjugatable-vec-space.distinct-map-nth[where ls =
signs, where inds = (find-nonzeros-from-input-vec (solve-for-lhs-R p qs subsets
(matrix-A-R signs subsets)))]
by blast
qed

```

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)
unfolding all-list-constr-R-def
proof clarsimp
fix a b
assume in-red-subsets: (a, b) ∈ set (reduction-subsets-R p qs signs subsets (matrix-A-R
signs subsets))
have solve-construct: construct-lhs-vector-R p qs signs =
solve-for-lhs-R p qs subsets (matrix-A-R signs subsets)
using construct-lhs-matches-solve-for-lhs-R assms
by simp
have rows-to-keep-hyp: ∀ y. y ∈ set (rows-to-keep (reduce-mat-cols (matrix-A-R
signs subsets) (solve-for-lhs-R p qs subsets (matrix-A-R signs subsets)))) →
y < length subsets
proof clarsimp
fix y
assume in-set: y ∈ set (rows-to-keep
(take-cols-from-matrix (matrix-A-R signs subsets) (find-nonzeros-from-input-vec
(solve-for-lhs-R p qs subsets (matrix-A-R signs subsets)))))
have in-set-2: y ∈ set (rows-to-keep
(take-cols-from-matrix (matrix-A-R signs subsets) (find-nonzeros-from-input-vec
(construct-lhs-vector-R p qs signs))))
using in-set solve-construct by simp
let ?lhs-vec = (solve-for-lhs-R p qs subsets (matrix-A-R signs subsets))
have h30: (construct-lhs-vector-R p qs signs) = ?lhs-vec
using assms construct-lhs-matches-solve-for-lhs-R
by simp
have h3a: ∀ x. List.member (find-nonzeros-from-input-vec ?lhs-vec) x → x <

```

```

length (signs)
  using h30 size-of-lhs-R unfolding find-nonzeros-from-input-vec-def
  using atLeastLessThan-iff filter-is-subset set-upt subset-eq apply (auto)
  by (smt (verit, best) atLeastLessThan-iff mem-Collect-eq set-filter set-upt)
  have h3c:  $\forall x. \text{List.member (rows-to-keep (reduce-mat-cols (matrix-A-R signs subsets) (solve-for-lhs-R p qs subsets (matrix-A-R signs subsets))))} x \rightarrow x < \text{length (subsets)}$ 
  proof clarsimp
  fix x
  assume x-mem:  $x \in \text{set (rows-to-keep (take-cols-from-matrix (matrix-A-R signs subsets) (find-nonzeros-from-input-vec (solve-for-lhs-R p qs subsets (matrix-A-R signs subsets))))}$ 
  show  $x < \text{length (subsets)}$  using x-mem unfolding rows-to-keep-def using pivot-positions
  using dim-row-matrix-A h3a nonzero reduce-system-matrix-signs-helper-R rows-to-keep-def rows-to-keep-lem dim-row-matrix-A-R rows-to-keep-def rows-to-keep-lem
  apply auto
  apply (smt (verit) dim-row-matrix-A-R rows-to-keep-lem x-mem)
  done
qed
then show  $y < \text{length subsets}$ 
  using in-set-2 solve-construct by auto
qed
show list-constr a (length qs)  $\wedge$  list-constr b (length qs) using in-red-subsets
unfolding reduction-subsets-def
  using take-indices-lem-R [of - subsets] rows-to-keep-hyp
  using all-list-constr-R-def in-set-conv-nth length-init
  apply (auto simp add: in-set-conv-nth)
  apply (metis in-set-conv-nth reduction-subsets-R-def rows-to-keep-hyp)
  apply (metis in-set-conv-nth reduction-subsets-R-def rows-to-keep-hyp)
  done
qed

```

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
by simp

```

```

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)))
unfolding get-matrix-R-def get-subsets-R-def get-signs-R-def by auto

```

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: length qs1 > 0
assumes nontriv2: 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 –
  let ?subsets = (get-subsets-R (snd (combine-systems-R p (qs1, matrix1, subsets1, signs1) (qs2, matrix2, subsets2, signs2))))
  let ?signs = (get-signs-R (snd (combine-systems-R p (qs1, matrix1, subsets1, signs1) (qs2, matrix2, subsets2, signs2))))
  let ?matrix = (get-matrix-R (snd (combine-systems-R p (qs1, matrix1, subsets1, signs1) (qs2, matrix2, subsets2, signs2))))
  have h1: all-list-constr-R ?subsets (length (qs1 @ qs2))
    using well-def-step-R[of subsets1 qs1 subsets2 qs2] assms
    by (simp add: nontriv2 get-subsets-R-def satisfies-properties-R-def smash-systems-R-def)

  have h2: well-def-signs (length (qs1 @ qs2)) ?signs
    using well-def-signs-step[of qs1 qs2 signs1 signs2]
    using get-signs-R-def nontriv1 nontriv2 satisfies-properties-R-def satisfies-properties-sys1 satisfies-properties-sys2 smash-systems-R-def
    by (metis combining-to-smash-R snd-conv)
  have h3: distinct ?signs
    using distinct-step[of - signs1 - signs2] assms
    using combine-systems.simps get-signs-R-def satisfies-properties-R-def smash-systems-R-def snd-conv
    by (metis combining-to-smash-R)
  have h4: satisfy-equation-R p (qs1 @ qs2) ?subsets ?signs
    using assms inductive-step-R[of p qs1 qs2 signs1 signs2 subsets1 subsets2]
    using get-signs-R-def get-subsets-R-def satisfies-properties-R-def smash-systems-R-def

```

```

    by (metis (no-types, opaque-lifting) combining-to-smash-R h1 h3 matrix-equation-R
        snd-conv subset-step-R)
    have h5: invertible-mat ?matrix
      using assms inductive-step-R[of p qs1 qs2 signs1 signs2 subsets1 subsets2]
      by (metis combining-to-smash-R fst-conv get-matrix-R-def kronecker-invertible
        satisfies-properties-R-def smash-systems-R-def snd-conv)
    have h6: ?matrix = matrix-A-R ?signs ?subsets
      apply (auto simp add: get-matrix-R-def smash-systems-R-def get-signs-R-def
        get-subsets-R-def)
      apply (subst matrix-construction-is-kronecker-product-R [of subsets1 - signs1
        signs2 subsets2])
      apply (metis Ball-set List.member-iff all-list-constr-R-def list-constr-def
        satisfies-properties-R-def satisfies-properties-sys1)
    using satisfies-properties-R-def satisfies-properties-sys1 well-def-signs-def apply
    blast
    using satisfies-properties-R-def satisfies-properties-sys1 satisfies-properties-sys2
    apply presburger
    done
    have h7: set (characterize-consistent-signs-at-roots p (qs1 @ qs2))
      ⊆ set (?signs)
      using subset-step-R[of p qs1 signs1 qs2 signs2] assms
    by (simp add: nonzero get-signs-R-def satisfies-properties-R-def smash-systems-R-def)

    then show ?thesis unfolding satisfies-properties-R-def using h1 h2 h3 h4 h5
    h6 h7 by blast
  qed

```

lemma *combining-sys-satisfies-properties-R:*

```

  fixes p:: real poly
  fixes qs1 :: real poly list
  fixes qs2 :: real poly list
  assumes nonzero: p ≠ 0
  assumes nontriv1: length qs1 > 0
  assumes nontriv2: 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))))))
  using combining-sys-satisfies-properties-helper-R[of p qs1 qs2]
  by (metis getter-functions-R nontriv1 nontriv2 nonzero satisfies-properties-sys1
    satisfies-properties-sys2)

```

```

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 –
  have h1: all-list-constr-R (get-subsets-R (reduce-system-R p (qs, matrix, subsets,
signs))) ( $\text{length } qs$ )
    using reduction-doesnt-break-subsets-R assms reduction-subsets-is-get-subsets-R
satisfies-properties-R-def satisfies-properties-sys by auto
  have h2: well-def-signs ( $\text{length } qs$ ) (get-signs-R (reduce-system-R p (qs, matrix,
subsets, signs)))
    using reduction-doesnt-break-length-signs-R[of signs qs p subsets] assms reduction-signs-is-get-signs-R
satisfies-properties-R-def well-def-signs-def by auto
  have h3: distinct (get-signs-R (reduce-system-R p (qs, matrix, subsets, signs)))
    using reduction-signs-are-distinct-R[of p qs subsets signs] assms reduction-signs-is-get-signs-R
satisfies-properties-R-def by auto
  have h4: satisfy-equation-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)))
    using reduce-system-matrix-equation-preserved-R[of p qs signs subsets] assms
satisfies-properties-R-def by auto
  have h5: invertible-mat (get-matrix-R (reduce-system-R p (qs, matrix, subsets,
signs)))
    using reduction-doesnt-break-things-invertibility-R assms same-size-R satisfies-properties-R-def
by auto
  have h6: get-matrix-R (reduce-system-R p (qs, matrix, subsets, signs)) =
  matrix-A-R (get-signs-R (reduce-system-R p (qs, matrix, subsets, signs)))
  (get-subsets-R (reduce-system-R p (qs, matrix, subsets, signs)))
    using reduce-system-matrix-match-R[of p qs signs subsets] assms satisfies-properties-R-def
by auto
  have h7: set (characterize-consistent-signs-at-roots p qs)  $\subseteq$  set (get-signs-R (reduce-system-R
p (qs, matrix, subsets, signs)))
    using reduction-doesnt-break-things-signs-R[of p qs signs subsets] assms reduction-signs-is-get-signs-R
satisfies-properties-R-def by auto
  then show ?thesis unfolding satisfies-properties-R-def using h1 h2 h3 h4 h5
h6 h7
  by blast
qed

```

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 $len1$: $length\ qs = 1$
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 –
have $h1$: *all-list-constr-R* $[(\ [], \ []), ([0], \ []), (\ [], [0])]$ ($length\ qs$)
using $len1$ **unfolding** *all-list-constr-R-def list-constr-def* **by** *auto*
have $h2$: *well-def-signs* ($length\ qs$) $[[1], [-1]]$
unfolding *well-def-signs-def* **using** $len1$
by *auto*
have $h3$: *distinct* $([[1], [0], [-1]]::rat\ list\ list)$
by *auto*
have $h4$: *satisfy-equation-R* $p\ qs$ $[(\ [], \ []), ([0], \ []), (\ [], [0])]$ $[[1], [0], [-1]]$
using *assms base-case-satisfy-equation-alt-R*[of $qs\ p$] **by** *auto*
have $h6$: (*mat-of-rows-list* 3 $[[1, 1, 1], [0, 1, 0], [1, 0, -1]]::rat\ mat$) = (*matrix-A-R* $([[1], [0], [-1]])$ $([(\ [], \ []), ([0], \ []), (\ [], [0])]$) :: *rat mat*)
using *mat-base-case-R* **by** *auto*
then have $h5$: *invertible-mat* (*mat-of-rows-list* 3 $[[1, 1, 1], [0, 1, 0], [1, 0, -1]]::rat\ mat$)
using *base-case-invertible-mat-R*
by *simp*
have $h7$: *set* (*characterize-consistent-signs-at-roots* $p\ qs$) \subseteq *set* $([[1], [0], [-1]])$
using *assms base-case-sgas-alt-R*[of $qs\ p$]
by *simp*
have *base-case-hyp: satisfies-properties-R* $p\ qs$ $[(\ [], \ []), ([0], \ []), (\ [], [0])]$ $[[1], [0], [-1]]$
(*mat-of-rows-list* 3 $[[1, 1, 1], [0, 1, 0], [1, 0, -1]]::rat\ mat$)
using $h1\ h2\ h3\ h4\ h5\ h6\ h7$
using *satisfies-properties-R-def* **apply** (*auto*)
by (*simp add: well-def-signs-def*)
then have *key-hyp: satisfies-properties-R* $p\ qs$ (*get-subsets-R* (*reduce-system-R* p ($qs, base-case-info-R$))) (*get-signs-R* (*reduce-system-R* p ($qs, base-case-info-R$))) (*get-matrix-R* (*reduce-system-R* p ($qs, base-case-info-R$)))
using *reducing-sys-satisfies-properties-R*
by (*metis base-case-info-R-def len1 nonzero nonzero zero-less-one-class.zero-less-one*)

show *?thesis*
by (*simp add: key-hyp len1*)
qed

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 ≠ 0 ∧ (length qs > 0))
  → 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 (induction length qs arbitrary: qs rule: less-induct)
  case less
  have len1-h: length qs = 1 → (p ≠ 0 ∧ (length qs > 0)) → 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))
    using length-1-calculate-data-satisfies-properties-R
  by blast
  let ?len = length qs
  let ?q1 = take (?len div 2) qs
  let ?left = calculate-data-R p ?q1
  let ?q2 = drop (?len div 2) qs
  let ?right = calculate-data-R p ?q2
  let ?comb = combine-systems-R p (?q1, ?left) (?q2, ?right)
  let ?red = reduce-system-R p ?comb
  have h-q1-len: length qs > 1 → (length ?q1 > 0) by auto
  have h-q2-len: length qs > 1 → (length ?q2 > 0) by auto
  have q1-sat-props: length qs > 1 → (p ≠ 0 ∧ (length qs > 0)) → satis-
fies-properties-R p ?q1 (get-subsets-R (calculate-data-R p ?q1)) (get-signs-R (calculate-data-R
p ?q1)) (get-matrix-R (calculate-data-R p ?q1))
    using less.hyps[of ?q1] h-q1-len
  by (metis div-le-dividend div-less-dividend length-take min.absorb2 one-less-numeral-iff
semiring-norm(76))
  have q2-help: length qs > 1 → length (drop (length qs div 2) qs) < length qs
    using h-q1-len by auto
  then have q2-sat-props: length qs > 1 → (p ≠ 0 ∧ (length qs > 0)) →
satisfies-properties-R p ?q2 (get-subsets-R (calculate-data-R p ?q2)) (get-signs-R
(calculate-data-R p ?q2)) (get-matrix-R (calculate-data-R p ?q2))
    using less.hyps[of ?q2] h-q2-len
  by blast
  have put-tog: ?q1@?q2 = qs
    using append-take-drop-id by blast
  then have comb-sat-props: length qs > 1 → (p ≠ 0 ∧ (length qs > 0)) →
(satisfies-properties-R p (qs) (get-subsets-R (snd ((combine-systems-R p (?q1, calculate-data-R
p ?q1) (?q2, calculate-data-R p ?q2))))))
  (get-signs-R (snd ((combine-systems-R p (?q1, calculate-data-R p ?q1) (?q2, calculate-data-R
p ?q2))))))
  (get-matrix-R (snd ((combine-systems-R p (?q1, calculate-data-R p ?q1) (?q2, calculate-data-R
p ?q2))))))
    using q1-sat-props q2-sat-props combining-sys-satisfies-properties-R
  using h-q1-len h-q2-len put-tog
  by metis
  then have comb-sat: length qs > 1 → (p ≠ 0 ∧ (length qs > 0)) →
(satisfies-properties-R p (qs) (get-subsets-R (snd ?comb)) (get-signs-R (snd

```

```

?comb)) (get-matrix-R (snd ?comb)))
  by blast
  have red-char: ?red = (reduce-system-R p (qs,(get-matrix-R (snd ?comb)),(get-subsets-R
(snd ?comb)),(get-signs-R (snd ?comb))))
    using getter-functions
    by (smt (z3) combine-systems-R.simps find-consistent-signs-at-roots-R-def find-consistent-signs-at-roots-thm
fst-conv get-matrix-R-def get-signs-R-def get-subsets-R-def prod.collapse put-tog smash-systems-R-def)
  then have length qs > 1  $\longrightarrow$  ( $p \neq 0 \wedge$  (length qs > 0))  $\longrightarrow$  (satisfies-properties-R
p qs (get-subsets-R ?red) (get-signs-R ?red) (get-matrix-R ?red))
    using reducing-sys-satisfies-properties-R comb-sat
    by presburger
  then have len-gt1: length qs > 1  $\longrightarrow$  ( $p \neq 0 \wedge$  (length qs > 0))  $\longrightarrow$  satis-
fies-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))
    apply (auto)
    by (smt (z3) div-le-dividend min.absorb2)
  then show ?case using len1-h len-gt1
    by (metis One-nat-def Suc-lessI)
qed

```

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: length qs = 1
  shows set (find-consistent-signs-at-roots-R p qs) = set (characterize-consistent-signs-at-roots
p qs)
  proof -
    let ?signs = [[1],[0],[-1]]::rat list list
    let ?subsets = [([]), ([])], ([0], ([])), ([], [0])]::(nat list * nat list) list
    let ?mat = (mat-of-rows-list 3 [[1,1,1], [0,1,0], [1,0,-1]])
    have mat-help: matrix-A-R ?signs ?subsets = (mat-of-rows-list 3 [[1,1,1], [0,1,0],
[1,0,-1]])
      using mat-base-case-R by auto
    have well-def-signs: well-def-signs (length qs) ?signs unfolding well-def-signs-def

    using len1 by auto
    have distinct-signs: distinct ?signs
      unfolding distinct-def by auto
    have ex-q:  $\exists$  (q::real poly). qs = [q]
      using len1
      using length-Suc-conv[of qs 0] by auto
    then have all-info: set (characterize-consistent-signs-at-roots p qs)  $\subseteq$  set(?signs)
      using assms base-case-sgas-R by auto
    have match: satisfy-equation-R p qs ?subsets ?signs

```

```

    using ex-q base-case-satisfy-equation-R nonzero
    by auto
    have invertible-mat: invertible-mat (matrix-A-R ?signs ?subsets)
      using inverse-mat-base-case-R inverse-mat-base-case-2-R unfolding invert-
ible-mat-def using mat-base-case-R
    by auto
    have h: set (get-signs-R (reduce-system-R p (qs, ((matrix-A-R ?signs ?subsets),
(?subsets, ?signs)))))) =
      set (characterize-consistent-signs-at-roots p qs)
    using nonzero nonzero well-def-signs distinct-signs all-info match invertible-mat
      reduce-system-sign-conditions-R[where p = p, where qs = qs, where signs
= ?signs, where subsets = ?subsets]
    by blast
    then have set (snd (snd (reduce-system-R p (qs, (?mat, (?subsets, ?signs))))))
=
      set (characterize-consistent-signs-at-roots p qs)
    unfolding get-signs-R-def using mat-help by auto
    then have set (snd (snd (reduce-system-R p (qs, base-case-info-R)))) = set
(characterize-consistent-signs-at-roots p qs)
    unfolding base-case-info-R-def
    by auto
    then show ?thesis using len1
      by (simp add: find-consistent-signs-at-roots-thm-R)
qed

```

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 ≠ 0
  assumes nontriv1: length qs1 > 0
  assumes nontriv2: length qs2 > 0
  assumes set(find-consistent-signs-at-roots-R p qs1) = set(characterize-consistent-signs-at-roots
p qs1)
  assumes set(find-consistent-signs-at-roots-R p qs2) = set(characterize-consistent-signs-at-roots
p qs2)
  shows set(snd(snd(reduce-system-R p (combine-systems-R p (qs1, calculate-data-R
p qs1) (qs2, calculate-data-R p qs2))))))
    = set(characterize-consistent-signs-at-roots p (qs1 @ qs2))
proof -
  let ?signs = (get-signs-R (snd ((combine-systems-R p (qs1, calculate-data-R p
qs1) (qs2, calculate-data-R p qs2))))))
  let ?subsets = (get-subsets-R (snd ((combine-systems-R p (qs1, calculate-data-R
p qs1) (qs2, calculate-data-R p qs2))))))
  have h0: satisfies-properties-R p (qs1 @ qs2) ?subsets ?signs
    (get-matrix-R (snd ((combine-systems-R p (qs1, calculate-data-R p qs1) (qs2, calculate-data-R
p qs2))))))

```

```

using calculate-data-satisfies-properties-R combining-sys-satisfies-properties-R
using nontriv1 nontriv2 nonzero nonzero
by simp
then have h1: set(characterize-consistent-signs-at-roots p (qs1@qs2))  $\subseteq$  set
?signs
  unfolding satisfies-properties-R-def
  by linarith
have h2: well-def-signs (length (qs1@qs2)) ?signs
  using calculate-data-satisfies-properties-R
  using h0 satisfies-properties-R-def by blast
have h3: distinct ?signs
  using calculate-data-satisfies-properties-R
  using h0 satisfies-properties-R-def by blast
have h4: satisfy-equation-R p (qs1@qs2) ?subsets ?signs
  using calculate-data-satisfies-properties-R
  using h0 satisfies-properties-R-def by blast
have h5: invertible-mat (matrix-A-R ?signs ?subsets)
  using calculate-data-satisfies-properties-R
  using h0 satisfies-properties-R-def
  by auto
have h: set (take-indices ?signs
  (find-nonzeros-from-input-vec (solve-for-lhs-R p (qs1@qs2) ?subsets
(matrix-A-R ?signs ?subsets))))
  = set(characterize-consistent-signs-at-roots p (qs1@qs2))
  using h1 h2 h3 h4 h5 reduction-deletes-bad-sign-conds-R
  using nonzero reduction-signs-R-def by auto
then have h: set (characterize-consistent-signs-at-roots p (qs1@qs2)) =
  set (reduction-signs-R p (qs1@qs2) ?signs ?subsets (matrix-A-R ?signs ?subsets
))
  unfolding reduction-signs-R-def get-signs-R-def
  by blast
have help-h: reduction-signs-R p (qs1@qs2) ?signs ?subsets (matrix-A-R ?signs
?subsets)
  = (take-indices ?signs (find-nonzeros-from-input-vec (solve-for-lhs-R p (qs1@qs2)
?subsets (matrix-A-R ?signs ?subsets))))
  unfolding reduction-signs-R-def by auto
have clear-signs: (signs-smash (get-signs-R (calculate-data-R p qs1)) (get-signs-R
(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))))))
  using combining-to-smash get-signs-R-def getter-functions-R smash-systems-R-def
snd-conv
proof –
  have combine-systems-R p (qs1, calculate-data-R p qs1) (qs2, calculate-data-R
p qs2) = (qs1 @ qs2, kronecker-product (get-matrix-R (calculate-data-R p qs1))
(get-matrix-R (calculate-data-R p qs2)), subsets-smash-R (length qs1) (get-subsets-R
(calculate-data-R p qs1)) (get-subsets-R (calculate-data-R p qs2)), signs-smash (snd
(snd (calculate-data-R p qs1))) (snd (snd (calculate-data-R p qs2))))
  by (metis (no-types) combine-systems-R.simps get-signs-R-def getter-functions-R
smash-systems-R-def)

```

```

then show ?thesis
  by (simp add: get-signs-R-def)
qed
have clear-subsets: (subsets-smash-R (length qs1) (get-subsets-R (calculate-data-R
p qs1)) (get-subsets-R (calculate-data-R p qs2))) = (get-subsets-R (snd ((combine-systems-R
p (qs1, calculate-data-R p qs1) (qs2, calculate-data-R p qs2))))))
  using Pair-inject combining-to-smash get-subsets-R-def prod.collapse smash-systems-R-def
  by (smt (z3) combine-systems-R.simps)
have well-def-signs (length qs1) (get-signs-R (calculate-data-R p qs1))
  using calculate-data-satisfies-properties-R
  using nontriv1 nonzero nonzero satisfies-properties-R-def
  by auto
then have well-def-signs1: ( $\bigwedge j. j \in \text{set } (get\text{-signs}\text{-R } (calculate\text{-data}\text{-R } p \text{ } qs1))$ 
 $\implies \text{length } j = (\text{length } qs1)$ )
  using well-def-signs-def by blast
have all-list-constr-R (get-subsets-R (calculate-data-R p qs1)) (length qs1)
  using calculate-data-satisfies-properties-R
  using nontriv1 nonzero nonzero satisfies-properties-R-def
  by auto
then have well-def-subsets1: ( $\bigwedge l i. l \in \text{set } (get\text{-subsets}\text{-R } (calculate\text{-data}\text{-R } p$ 
 $qs1)) \implies (i \in \text{set } (fst \text{ } l) \longrightarrow i < (\text{length } qs1)) \wedge (i \in \text{set } (snd \text{ } l) \longrightarrow i < (\text{length}$ 
 $qs1))$ )
  apply (simp add: all-list-constr-R-def list-constr-def)
  apply (smt (verit, ccfv-threshold) Ball-set prod.collapse)
  done
have extra-matrix-same: matrix-A-R (signs-smash (get-signs-R (calculate-data-R
p qs1)) (get-signs-R (calculate-data-R p qs2)))
  (subsets-smash-R (length qs1) (get-subsets-R (calculate-data-R p qs1))
(get-subsets-R (calculate-data-R p qs2)))
  = kronecker-product (get-matrix-R (calculate-data-R p qs1)) (get-matrix-R
(calculate-data-R p qs2))
  using well-def-signs1 well-def-subsets1
  using matrix-construction-is-kronecker-product-R
  using calculate-data-satisfies-properties-R nontriv1 nontriv2 nonzero nonzero
satisfies-properties-R-def
  by fastforce
then have matrix-same: matrix-A-R ?signs ?subsets = kronecker-product (get-matrix-R
(calculate-data-R p qs1)) (get-matrix-R (calculate-data-R p qs2))
  using clear-signs clear-subsets
  by simp
have comb-sys-h: snd(snd(reduce-system-R p (combine-systems-R p (qs1, calculate-data-R
p qs1) (qs2, calculate-data-R p qs2)))) =
  snd(snd(reduce-system-R p (qs1@qs2, (matrix-A-R ?signs ?subsets, (?subsets,
?signs))))))
  unfolding get-signs-R-def get-subsets-R-def using matrix-same
  by (metis (full-types) clear-signs clear-subsets combine-systems-R.simps get-signs-R-def
get-subsets-R-def getter-functions-R smash-systems-R-def)
then have extra-h: snd(snd(reduce-system-R p (qs1@qs2, (matrix-A-R ?signs
?subsets, (?subsets, ?signs)))))) =

```

```

      snd(snd(reduction-step-R (matrix-A-R ?signs ?subsets) ?signs ?subsets (solve-for-lhs-R
p (qs1@qs2) ?subsets (matrix-A-R ?signs ?subsets))))
    by simp
  then have same-h: set(snd(snd(reduce-system-R p (combine-systems-R p (qs1,calculate-data-R
p qs1) (qs2,calculate-data-R p qs2))))))
    = set (reduction-signs-R p (qs1@qs2) ?signs ?subsets (matrix-A-R ?signs
?subsets ))
    using comb-sys-h unfolding reduction-signs-R-def
    by (metis get-signs-R-def help-h reduction-signs-is-get-signs-R)
  then show ?thesis using h
    by blast
qed

```

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

```

  fixes p:: real poly
  fixes qs :: real poly list
  shows (p ≠ 0 ∧ length qs > 0) ⟶
    set(find-consistent-signs-at-roots-R p qs) = set(characterize-consistent-signs-at-roots
p qs)
proof (induction length qs arbitrary: qs rule: less-induct)
  case less
  then show ?case
  proof clarsimp
    assume ind-hyp: (∧ qsa.
      length qsa < length qs ⟹ qsa ≠ [] ⟶
      set (find-consistent-signs-at-roots-R p qsa) =
      set (characterize-consistent-signs-at-roots p qsa))
    assume nonzero: p ≠ 0
    assume nontriv: qs ≠ []
    let ?len = length qs
    let ?q1 = take ((?len) div 2) qs
    let ?left = calculate-data-R p ?q1
    let ?q2 = drop ((?len) div 2) qs
    let ?right = calculate-data-R p ?q2
    have nontriv-q1: length qs > 1 ⟶ length ?q1 > 0
      by auto
    have qs-more-q1: length qs > 1 ⟶ length qs > length ?q1
      by auto
    have nontriv-q2: length qs > 1 ⟶ length ?q2 > 0
      by auto
    have qs-more-q2: length qs > 1 ⟶ length qs > length ?q2
      by auto
    have key-h: set (snd (snd (if ?len ≤ Suc 0 then reduce-system-R p (qs,
base-case-info-R)
      else Let (combine-systems-R p (?q1, ?left) (?q2, ?right))
      (reduce-system-R p)))) =
      set (characterize-consistent-signs-at-roots p qs)
  proof -
  have h-len1 : ?len = 1 ⟶ set (snd (snd (if ?len ≤ Suc 0 then reduce-system-R

```

```

p (qs, base-case-info-R)
  else Let (combine-systems-R p (?q1, ?left) (?q2, ?right))
        (reduce-system-R p))) =
set (characterize-consistent-signs-at-roots p qs)
using find-consistent-signs-at-roots-len1-R[of p qs] nonzero nontriv
by (simp add: find-consistent-signs-at-roots-thm-R)
have h-len-gt1 : ?len > 1 → set (snd (snd (if ?len ≤ Suc 0 then re-
duce-system-R p (qs, base-case-info-R)
  else Let (combine-systems-R p (?q1, ?left) (?q2, ?right))
        (reduce-system-R p)))) =
set (characterize-consistent-signs-at-roots p qs)
proof –
have h-imp-a: ?len > 1 → set (snd (snd (reduce-system-R p (combine-systems-R
p (?q1, ?left) (?q2, ?right)))))) =
set (characterize-consistent-signs-at-roots p qs)
proof –
have h1: ?len > 1 → set(snd(snd(?left))) = set (characterize-consistent-signs-at-roots
p ?q1)
using nontriv-q1 ind-hyp[of ?q1] qs-more-q1
by (metis find-consistent-signs-at-roots-thm-R less-numeral-extra(3)
list.size(3))
have h2: ?len > 1 → set(snd(snd(?right))) = set (characterize-consistent-signs-at-roots
p ?q2)
using nontriv-q2 ind-hyp[of ?q2] qs-more-q2
by (metis (full-types) find-consistent-signs-at-roots-thm-R list.size(3)
not-less-zero)
show ?thesis using nonzero nontriv-q1 nontriv-q2 h1 h2 smaller-sys-are-good-R

by (metis append-take-drop-id find-consistent-signs-at-roots-thm-R)
qed
then have h-imp: ?len > 1 → set (snd (snd (Let (combine-systems-R p
(?q1, ?left) (?q2, ?right))
        (reduce-system-R p)))) =
set (characterize-consistent-signs-at-roots p qs)
by auto
then show ?thesis by auto
qed
show ?thesis using h-len1 h-len-gt1
by (meson ⟨qs ≠ []⟩ length-0-conv less-one nat-neq-iff)
qed
then show set (find-consistent-signs-at-roots-R p qs) = set (characterize-consistent-signs-at-roots
p qs)
using One-nat-def calculate-data.simps find-consistent-signs-at-roots-thm length-0-conv
nontriv
by (smt (z3) calculate-data-R.simps find-consistent-signs-at-roots-thm-R)
qed
qed

```

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

```

fixes p:: real poly
assumes p ≠ 0
shows set(find-consistent-signs-at-roots-R p []) =
        set(characterize-consistent-signs-at-roots p [])
proof –
  obtain a b c where abc: reduce-system-R p ([1], base-case-info-R) = (a,b,c)
    using prod-cases3 by blast
  have find-consistent-signs-at-roots-R p [1] = c using abc
    by (simp add: find-consistent-signs-at-roots-thm-R)
  have *: set (find-consistent-signs-at-roots-R p [1]) = set (characterize-consistent-signs-at-roots
p [1])
    apply (subst find-consistent-signs-at-roots-1-R)
    using assms by auto
  have set(characterize-consistent-signs-at-roots p []) = drop 1 ‘ set(characterize-consistent-signs-at-roots
p [1])
    unfolding characterize-consistent-signs-at-roots-def consistent-sign-vec-def signs-at-def
squash-def apply simp
    using drop0 drop-Suc-Cons image-cong image-image
  proof –
    have (λr. []) ‘ set (characterize-root-list-p p) = (λr. drop (Suc 0) [1::rat]) ‘ set
(characterize-root-list-p p)
      by force
    then show (λr. []) ‘ set (characterize-root-list-p p) = drop (Suc 0) ‘ (λr.
[1::rat]) ‘ set (characterize-root-list-p p)
      by blast
  qed
thus ?thesis using abc *
  apply (auto) apply (simp add: find-consistent-signs-at-roots-thm-R)
  by (simp add: find-consistent-signs-at-roots-thm-R)
qed

```

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

```

fixes p:: real poly
fixes qs :: real poly list
assumes p ≠ 0
shows set(find-consistent-signs-at-roots-R p qs) = set(characterize-consistent-signs-at-roots
p qs)
  by (metis assms find-consistent-signs-at-roots-0-R find-consistent-signs-at-roots-1-R
length-greater-0-conv)

```

end

```

theory Renegar-Decision
imports Renegar-Proofs
        BKR-Decision

```

begin

28 Algorithm

definition consistent-sign-vectors-R::real poly list ⇒ real set ⇒ rat list set

where *consistent-sign-vectors-R* *qs S* = (*consistent-sign-vec qs*) ‘ *S*

primrec *prod-list-var*:: ('*a*::*idom*) *list* ⇒ ('*a*::*idom*)
where *prod-list-var* [] = 1
| *prod-list-var* (*h*#*T*) = (if *h* = 0 then (*prod-list-var T*) else (*h** *prod-list-var T*))

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

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

definition *find-consistent-signs-R* :: *real poly list* ⇒ *rat list list*
where
find-consistent-signs-R ps = *find-consistent-signs-at-roots-R (poly-f ps) ps*

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

definition *decide-existential-R* :: *real poly fml* ⇒ *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 ≠ *None*
)

28.1 Proofs

definition *roots-of-poly-f*:: *real poly list* ⇒ *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* ≠ 0

proof (*induct qs*)
case *Nil*

```

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

```

```

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

```

```

lemma check-all-const-deg-prop:
  shows check-all-const-deg l = True ⟷ (∀ p ∈ set(l). degree p = 0)
proof (induct l)
  case Nil
  then show ?case by auto
next
  case (Cons a l)
  then show ?case by auto
qed

```

```

lemma poly-f-nonzero:
  fixes qs :: real poly list
  shows (poly-f qs) ≠ 0
proof -
  have eo: (∀ p ∈ set qs. degree p = 0) ∨ (∃ p ∈ set qs. degree p > 0)
    by auto
  have c1: (∀ p ∈ set qs. degree p = 0) ⟹ (poly-f qs) ≠ 0
    unfolding poly-f-def using check-all-const-deg-prop by auto
  have c2: (∃ p ∈ set qs. degree p > 0) ⟹ (poly-f qs) ≠ 0

```

```

proof clarsimp
  fix q
  assume q-in:  $q \in \text{set } qs$ 
  assume deg-q:  $0 < \text{degree } q$ 
  assume contrad:  $\text{poly-f } qs = 0$ 
  have nonconst:  $\text{check-all-const-deg } qs = \text{False}$  using deg-q check-all-const-deg-prop
    q-in by auto
  have h1:  $\text{prod-list-var } qs \neq 0$  using prod-list-var-nonzero by auto
  then have  $\text{degree } (\text{prod-list-var } qs) > 0$  using q-in deg-q h1
  proof (induct qs)
    case Nil
    then show ?case by auto
  next
  case (Cons a qs)
  have q-nonz:  $q \neq 0$  using Cons.prems by auto
  have q-ins:  $q \in \text{set } (a \# qs)$  using Cons.prems by auto
  then have  $q = a \vee q \in \text{set } qs$  by auto
  then have eo:  $q = a \vee \text{List.member } qs \ q$ 
    by auto
  have degq:  $\text{degree } q > 0$  using Cons.prems by auto
  have h2:  $(\text{prod-list } (a \# qs)) = a * (\text{prod-list } qs)$ 
    by auto
  have isa:  $q = a \longrightarrow 0 < \text{degree } (\text{prod-list-var } (a \# qs))$ 
    using h2 degree-mult-eq-0 [where  $p = q$ , where  $q = \text{prod-list-var } qs$ ]
    Cons.prems by auto
  have inl:  $\text{List.member } qs \ q \longrightarrow 0 < \text{degree } (\text{prod-list-var } (a \# qs))$ 
  proof -
    have nonzprod:  $\text{prod-list-var } (a \# qs) \neq 0$  using prod-list-var-nonzero by
auto
    have q dvd prod-list-var (a # qs)
      using q-dvd-prod-list-var-prop [where  $q = q$ , where  $qs = (a \# qs)$ ] q-nonz
q-ins
      by auto
    then show ?thesis using divides-degree [where  $p = q$ , where  $q = \text{prod-list-var } (a \# qs)$ ]
      nonzprod degq
      by auto
    qed
    then show ?case using eo isa by auto
  qed
  then have h2:  $\text{pderiv } (\text{prod-list-var } qs) \neq 0$  using pderiv-eq-0-iff [where  $p =$ 
prod-list-var qs]
    by auto
  then have  $\text{pderiv } (\text{prod-list-var } qs) * \text{prod-list-var } qs \neq 0$ 
    using prod-list-var-nonzero h2 by auto
  then show False using contrad nonconst unfolding poly-f-def deg-q
    by (smt (z3) mult-eq-0-iff pCons-eq-0-iff)
  qed
  show ?thesis using eo c1 c2 by auto
qed

```

lemma *poly-f-roots-prop-1*:

fixes *qs*:: *real poly list*

assumes *non-const*: *check-all-const-deg qs = False*

shows $\forall x1. \forall x2. ((x1 < x2 \wedge (\exists q1 \in \text{set } (qs). q1 \neq 0 \wedge (\text{poly } q1 x1) = 0) \wedge (\exists q2 \in \text{set}(qs). q2 \neq 0 \wedge (\text{poly } q2 x2) = 0)) \longrightarrow (\exists q. x1 < q \wedge q < x2 \wedge \text{poly } (\text{poly-f } qs) q = 0))$

proof *clarsimp*

fix *x1*:: *real*

fix *x2*:: *real*

fix *q1*:: *real poly*

fix *q2*:: *real poly*

assume $x1 < x2$

assume *q1-in*: $q1 \in \text{set } qs$

assume *q1-0*: $\text{poly } q1 x1 = 0$

assume *q1-nonz*: $q1 \neq 0$

assume *q2-in*: $q2 \in \text{set } qs$

assume *q2-0*: $\text{poly } q2 x2 = 0$

assume *q2-nonz*: $q2 \neq 0$

have *prod-z-x1*: $\text{poly } (\text{prod-list-var } qs) x1 = 0$ **using** *q1-in q1-0*

using *q1-nonz q-dvd-prod-list-var-prop*[of *q1 qs*] **by** *auto*

have *prod-z-x2*: $\text{poly } (\text{prod-list-var } qs) x2 = 0$ **using** *q2-in q2-0*

using *q2-nonz q-dvd-prod-list-var-prop*[of *q2 qs*] **by** *auto*

have $\exists w > x1. w < x2 \wedge \text{poly } (\text{pderiv } (\text{prod-list-var } qs)) w = 0$

using *Rolle-pderiv*[**where** $q = \text{prod-list-var } qs$] *prod-z-x1 prod-z-x2*

using $\langle x1 < x2 \rangle$ **by** *blast*

then obtain *w* **where** *w-def*: $w > x1 \wedge w < x2 \wedge \text{poly } (\text{pderiv } (\text{prod-list-var } qs)) w = 0$

w = 0

by *auto*

then have $\text{poly } (\text{poly-f } qs) w = 0$

unfolding *poly-f-def* **using** *non-const*

by *simp*

then show $\exists q > x1. q < x2 \wedge \text{poly } (\text{poly-f } qs) q = 0$

using *w-def* **by** *blast*

qed

lemma *main-step-aux1-R*:

fixes *qs*:: *real poly list*

assumes *non-const*: *check-all-const-deg qs = True*

shows $\text{set } (\text{find-consistent-signs-R } qs) = \text{consistent-sign-vectors-R } qs \text{ UNIV}$

proof –

have *poly-f-is*: $\text{poly-f } qs = [:0, 1:]$ **unfolding** *poly-f-def* **using** *assms*

by *auto*

have *same*: $\text{set } (\text{find-consistent-signs-at-roots-R } [:0, 1:] qs) = \text{set } (\text{characterize-consistent-signs-at-roots } [:0, 1:] qs)$ **using** *find-consistent-signs-at-roots-R*[of $[:0, 1:] qs$]

by *auto*

have *rech*: $(\text{sorted-list-of-set } \{x. \text{poly } ([:0, 1:]::\text{real poly}) x = 0\}) = [0]$ **by** *auto*

```

have alldeg0: ( $\forall p \in \text{set } qs. \text{degree } p = 0$ ) using non-const check-all-const-deg-prop
by auto
then have allconst:  $\forall p \in \text{set } qs. (\exists (k::\text{real}). p = [:k:])$ 
apply (auto)
by (meson degree-eq-zeroE)
then have allconstvar:  $\forall p \in \text{set } qs. \forall (x::\text{real}). \forall (y::\text{real}). \text{poly } p \ x = \text{poly } p \ y$ 
by fastforce
have e1:  $\text{set } (\text{remdups } (\text{map } (\text{signs-at } qs) [0])) \subseteq$ 
consistent-sign-vectors-R qs UNIV
unfolding signs-at-def squash-def consistent-sign-vectors-R-def consistent-sign-vec-def
apply (simp)
by (smt (verit, best) class-ring.ring-simprules(2) comp-def image-iff length-map
map-nth-eq-conv)
have e2:  $\text{consistent-sign-vectors-R } qs \text{ UNIV} \subseteq \text{set } (\text{remdups } (\text{map } (\text{signs-at } qs)$ 
 $[0]))$ 
unfolding signs-at-def squash-def consistent-sign-vectors-R-def consistent-sign-vec-def
apply (simp)
using allconstvar
by (smt (verit, best) comp-apply image-iff insert-iff map-eq-conv subsetI)
have  $\text{set } (\text{remdups } (\text{map } (\text{signs-at } qs) [0])) =$ 
consistent-sign-vectors-R qs UNIV
using e1 e2 by auto
then have  $\text{set } (\text{characterize-consistent-signs-at-roots } [:0, 1:] qs) = \text{consistent-sign-vectors-R}$ 
 $qs \text{ UNIV}$ 
unfolding characterize-consistent-signs-at-roots-def characterize-root-list-p-def

using rech by auto
then show ?thesis using same poly-f-is unfolding find-consistent-signs-R-def
by auto
qed

```

lemma *sorted-list-lemma-var*:

```

fixes l:: real list
fixes x:: real
assumes length l > 1
assumes strict-sort: sorted-wrt (<) l
assumes x-not-in:  $\neg (\text{List.member } l \ x)$ 
assumes lt-a:  $x > (l ! 0)$ 
assumes b-lt:  $x < (l ! (\text{length } l - 1))$ 
shows ( $\exists n. n < \text{length } l - 1 \wedge x > l ! n \wedge x < l !(n+1)$ ) using assms
proof (induct l)
case Nil
then show ?case by auto
next
case (Cons a l)
have len-gteq: length l  $\geq 1$  using Cons.prem1
by (metis One-nat-def Suc-eq-plus1 list.size(4) not-le not-less-eq)
have len-one: length l = 1  $\implies (\exists n. n < \text{length } (a\#l) - 1 \wedge x > (a\#l) ! n \wedge x$ 
 $< (a\#l) !(n+1))$ 

```

```

proof –
  assume len-is:  $\text{length } l = 1$ 
  then have  $x > (a\#l)! 0 \wedge x < (a\#l)! 1$  using Cons.prems(4) Cons.prems(5)
    by auto
  then show  $(\exists n. n < \text{length } (a\#l) - 1 \wedge x > (a\#l)! n \wedge x < (a\#l)!(n+1))$ 

    using len-is by auto
  qed
  have len-gt:  $\text{length } l > 1 \implies (\exists n. n < \text{length } (a\#l) - 1 \wedge x > (a\#l)! n \wedge x < (a\#l)!(n+1))$ 
  proof –
    assume len-gt-one:  $\text{length } l > 1$ 
    have eo:  $x \neq l! 0$  using Cons.prems(3) apply auto
      by (metis One-nat-def Suc-lessD len-gt-one nth-mem)
    have c1:  $x < l! 0 \implies (\exists n. n < \text{length } (a\#l) - 1 \wedge x > (a\#l)! n \wedge x < (a\#l)!(n+1))$ 
    proof –
      assume xlt:  $x < l! 0$ 
      then have  $x < (a\#l)! 1$ 
        by simp
      then show ?thesis using Cons.prems(4) len-gt-one apply (auto)
        using Cons.prems(4) Suc-lessD by blast
    qed
    have c2:  $x > l! 0 \implies (\exists n. n < \text{length } (a\#l) - 1 \wedge x > (a\#l)! n \wedge x < (a\#l)!(n+1))$ 
    proof –
      assume asm:  $x > l! 0$ 
      have xlt-1:  $x < l! (\text{length } l - 1)$ 
        using Cons.prems(5)
        by (metis Cons.prems(1) One-nat-def add-diff-cancel-right' list.size(4) nth-Cons-pos zero-less-diff)
      have ssl: sorted-wrt ( $<$ )  $l$  using Cons.prems(2)
        using sorted-wrt.simps(2) by auto
      have  $\neg \text{List.member } l x$  using Cons.prems(3)
        by simp
      then have  $\exists n < \text{length } l - 1. l! n < x \wedge x < l! (n + 1)$ 
        using asm xlt-1 len-gt-one ssl Cons.hyps
        by auto
      then show ?thesis
        by (metis One-nat-def Suc-eq-plus1 diff-Suc-1 less-diff-conv list.size(4) nth-Cons-Suc)
    qed
    show  $(\exists n. n < \text{length } (a\#l) - 1 \wedge x > (a\#l)! n \wedge x < (a\#l)!(n+1))$ 
      using eo c1 c2
      by (meson linorder-neqE-linordered-idom)
  qed
  then show ?case
    using len-gteq len-one len-gt
    apply (auto)

```

by (metis One-nat-def less-numeral-extra(1) linorder-neqE-nat not-less nth-Cons-0)

qed

lemma *all-sample-points-prop*:

assumes *is-not-const*: *check-all-const-deg qs = False*

assumes *s-is*: $S = (\text{characterize-root-list-p } (\text{pderiv } (\text{prod-list-var } qs) * (\text{prod-list-var } qs)) * ([:-(\text{crb } (\text{prod-list-var } qs)),1:]]) * ([:(\text{crb } (\text{prod-list-var } qs)),1:]])$

shows *consistent-sign-vectors-R qs UNIV = consistent-sign-vectors-R qs (set S)*

proof –

let *?zer-list = sorted-list-of-set* $\{(x::\text{real}). (\exists q \in \text{set}(qs). (q \neq 0 \wedge \text{poly } q x = 0))\}$ *:: real list*

have *strict-sorted-h*: *sorted-wrt* ($<$) *?zer-list* **using** *sorted-sorted-list-of-set strict-sorted-iff* **by** *auto*

have *poly-f-is*: $\text{poly-f } qs = (\text{pderiv } (\text{prod-list-var } qs) * \text{prod-list-var } qs) * ([:-(\text{crb } (\text{prod-list-var } qs)),1:]]) * ([:(\text{crb } (\text{prod-list-var } qs)),1:]])$

unfolding *poly-f-def* **using** *is-not-const* **by** *auto*

then have *set-S-char*: *set S =* $\{x. \text{poly } (\text{poly-f } qs) x = 0\}$ *:: real set*

using *poly-roots-finite*[*of poly-f qs*] *set-sorted-list-of-set poly-f-nonzero*[*of qs*]

using *s-is* **unfolding** *characterize-root-list-p-def* **by** *auto*

have *difficult-direction*: *consistent-sign-vectors-R qs UNIV* \subseteq *consistent-sign-vectors-R qs (set S)*

proof *clarsimp*

fix *x*

assume $x \in \text{consistent-sign-vectors-R } qs \text{ UNIV}$

then have $\exists y. x = (\text{consistent-sign-vec } qs y)$ **unfolding** *consistent-sign-vectors-R-def* **by** *auto*

then obtain *y* **where** *y-prop*: $x = \text{consistent-sign-vec } qs y$ **by** *auto*

then have $\exists k \in (\text{set } S). \text{consistent-sign-vec } qs k = \text{consistent-sign-vec } qs y$

proof –

have *c1*: $(\exists q \in (\text{set } qs). q \neq 0 \wedge \text{poly } q y = 0) \implies (\exists k \in (\text{set } S). \text{consistent-sign-vec } qs k = \text{consistent-sign-vec } qs y)$

proof –

assume $(\exists q \in (\text{set } qs). q \neq 0 \wedge \text{poly } q y = 0)$

then obtain *q* **where** $q \in (\text{set } qs) \wedge q \neq 0 \wedge \text{poly } q y = 0$ **by** *auto*

then have $\text{poly } (\text{prod-list-var } qs) y = 0$

using *q-dvd-prod-list-var-prop*[*of q qs*] **by** *auto*

then have $\text{poly } (\text{pderiv } (\text{prod-list-var } qs) * (\text{prod-list-var } qs) * ([:-(\text{crb } (\text{prod-list-var } qs)),1:]]) * ([:(\text{crb } (\text{prod-list-var } qs)),1:]]) y = 0$

by *auto*

then have $y \in (\text{set } S)$

using *s-is* **unfolding** *characterize-root-list-p-def*

proof –

have $y \in \{r. \text{poly } (\text{pderiv } (\text{prod-list-var } qs) * (\text{prod-list-var } qs) * ([:-(\text{crb } (\text{prod-list-var } qs)),1:]]) * ([:(\text{crb } (\text{prod-list-var } qs)),1:]]) r = 0\}$

using $\langle \text{poly } (\text{pderiv } (\text{prod-list-var } qs) * (\text{prod-list-var } qs) * ([:-(\text{crb } (\text{prod-list-var } qs)),1:]]) * ([:(\text{crb } (\text{prod-list-var } qs)),1:]]) y = 0 \rangle$ **by** *force*

then show *?thesis*

```

    by (metis characterize-root-list-p-def is-not-const poly-f-def poly-f-nonzero
poly-roots-finite s-is set-sorted-list-of-set)
  qed
  then show  $\exists k \in (\text{set } S). \text{consistent-sign-vec } qs \ k = \text{consistent-sign-vec } qs$ 
y
    by auto
  qed
  have len-gtz-prop:  $\text{length } ?\text{zer-list} > 0 \longrightarrow$ 
    ( $\exists w. w < \text{length } ?\text{zer-list} \wedge y = ?\text{zer-list } ! \ w$ )  $\vee$ 
    ( $y < ?\text{zer-list } ! \ 0$ )  $\vee$ 
    ( $y > ?\text{zer-list } ! \ (\text{length } ?\text{zer-list} - 1)$ )  $\vee$ 
    ( $\exists k < (\text{length } ?\text{zer-list} - 1). y > ?\text{zer-list } ! \ k \wedge y < ?\text{zer-list } ! \ (k+1)$ )
  proof -
    let ?c = ( $\exists w. w < \text{length } ?\text{zer-list} \wedge y = ?\text{zer-list } ! \ w$ )  $\vee$ 
      ( $y < ?\text{zer-list } ! \ 0$ )  $\vee$ 
      ( $y > ?\text{zer-list } ! \ (\text{length } ?\text{zer-list} - 1)$ )  $\vee$ 
      ( $\exists k < (\text{length } ?\text{zer-list} - 1). y > ?\text{zer-list } ! \ k \wedge y < ?\text{zer-list } ! \ (k+1)$ )
    have lis1:  $\text{length } ?\text{zer-list} = 1 \implies ?c$ 
      by auto
    have h1:  $\neg(\exists w. w < \text{length } ?\text{zer-list} \wedge y = ?\text{zer-list } ! \ w) \implies \neg(\text{List.member } ?\text{zer-list } y)$ 
      by (auto simp add: in-set-conv-nth)
    have h2: ( $\text{length } ?\text{zer-list} > 0 \wedge \neg(\exists w. w < \text{length } ?\text{zer-list} \wedge y = ?\text{zer-list } ! \ w) \wedge \neg(y < ?\text{zer-list } ! \ 0)$ )  $\implies y > ?\text{zer-list } ! \ 0$ 
      by auto
    have h3: ( $\text{length } ?\text{zer-list} > 1 \wedge \neg(\exists w. w < \text{length } ?\text{zer-list} \wedge y = ?\text{zer-list } ! \ w) \wedge \neg(y > ?\text{zer-list } ! \ (\text{length } ?\text{zer-list} - 1))$ )  $\implies$ 
       $y < ?\text{zer-list } ! \ (\text{length } ?\text{zer-list} - 1)$ 
      apply (auto)
      by (smt (z3) diff-Suc-less gr-implies-not0 not-gr-zero)
    have  $\text{length } ?\text{zer-list} > 1 \wedge \neg(\exists w. w < \text{length } ?\text{zer-list} \wedge y = ?\text{zer-list } ! \ w) \wedge \neg(y < ?\text{zer-list } ! \ 0) \wedge \neg(y > ?\text{zer-list } ! \ (\text{length } ?\text{zer-list} - 1))$ 
       $\implies (\exists k < (\text{length } ?\text{zer-list} - 1). y > ?\text{zer-list } ! \ k \wedge y < ?\text{zer-list } ! \ (k+1))$ 
      using h1 h2 h3 strict-sorted-h sorted-list-lemma-var[of ?zer-list y]
    using One-nat-def Suc-lessD by presburger
    then have lgt1:  $\text{length } ?\text{zer-list} > 1 \implies ?c$ 
      by auto
    then show ?thesis using lis1 lgt1
      by (smt (z3) diff-is-0-eq' not-less)
  qed
  have neg-crb-in:  $(- \text{crb } (\text{prod-list-var } qs)) \in \text{set } S$ 
    using set-S-char poly-f-is by auto
  have pos-crb-in:  $(\text{crb } (\text{prod-list-var } qs)) \in \text{set } S$ 
    using set-S-char poly-f-is by auto
  have set-S-nonempty:  $\text{set } S \neq \{\}$  using neg-crb-in by auto
  have finset:  $\text{finite } \{x. \exists q \in \text{set } qs. q \neq 0 \wedge \text{poly } q \ x = 0\}$ 
  proof -
    have  $\forall q \in \text{set } qs. q \neq 0 \longrightarrow \text{finite } \{x. \text{poly } q \ x = 0\}$ 

```

```

    using poly-roots-finite by auto
  then show ?thesis by auto
qed
  have c2:  $\neg(\exists q \in (\text{set } qs). q \neq 0 \wedge \text{poly } q y = 0) \implies \exists k \in (\text{set } S).$ 
  consistent-sign-vec qs k = consistent-sign-vec qs y
  proof -
    assume  $\neg(\exists q \in (\text{set } qs). q \neq 0 \wedge \text{poly } q y = 0)$ 
    have c-c1:  $\text{length } ?\text{zer-list} = 0 \implies \exists k \in (\text{set } S).$  consistent-sign-vec qs k
    = consistent-sign-vec qs y
    proof -
      assume  $\text{length } ?\text{zer-list} = 0$ 
      then have  $\forall q \in \text{set } (qs). \forall (x::\text{real}). \forall (y::\text{real}). \text{squash } (\text{poly } q x) =$ 
       $\text{squash } (\text{poly } q y)$ 
      proof clarsimp
        fix q x y
        assume czer:  $\text{card } \{x. \exists q \in \text{set } qs. q \neq 0 \wedge \text{poly } q x = 0\} = 0$ 
        assume qin:  $q \in \text{set } qs$ 
        have fin-means-empty:  $\{x. \exists q \in \text{set } qs. q \neq 0 \wedge \text{poly } q x = 0\} = \{\}$ 
          using finset czer
          by auto
        have qzer:  $q = 0 \implies \text{squash } (\text{poly } q x) = \text{squash } (\text{poly } q y)$  by auto
        have qnonz:  $q \neq 0 \implies \text{squash } (\text{poly } q x) = \text{squash } (\text{poly } q y)$ 
        proof -
          assume qnonz:  $q \neq 0$ 
          then have noroots:  $\{x. \text{poly } q x = 0\} = \{\}$  using qin finset
            using Collect-empty-eq fin-means-empty by auto
          have nonzsq1:  $\text{squash } (\text{poly } q x) \neq 0$  using fin-means-empty qnonz
          czer qin
          unfolding squash-def by auto
        then have eo:  $(\text{poly } q x) > 0 \vee (\text{poly } q x) < 0$  unfolding squash-def
          apply (auto)
          by presburger
        have eo1:  $\text{poly } q x > 0 \implies \text{poly } q y > 0$ 
          using noroots poly-IVT-pos[of y x q] poly-IVT-neg[of x y q]
          apply (auto)
          by (metis linorder-neqE-linordered-idom)
        have eo2:  $\text{poly } q x < 0 \implies \text{poly } q y < 0$ 
          using noroots poly-IVT-pos[of x y q] poly-IVT-neg[of y x q]
          apply (auto) by (metis linorder-neqE-linordered-idom)
        then show  $\text{squash } (\text{poly } q x) = \text{squash } (\text{poly } q y)$ 
          using eo eo1 eo2 unfolding squash-def by auto
        qed
      show  $\text{squash } (\text{poly } q x) = \text{squash } (\text{poly } q y)$ 
        using qzer qnonz
        by blast
      qed
    then have  $\forall q \in \text{set } (qs). \text{squash } (\text{poly } q y) = \text{squash } (\text{poly } q (-\text{crb}$ 
     $(\text{prod-list-var } qs)))$ 
    by auto

```

```

    then show  $\exists k \in (\text{set } S). \text{consistent-sign-vec } qs \ k = \text{consistent-sign-vec}$ 
     $qs \ y$ 
      using neg-crb-in unfolding consistent-sign-vec-def squash-def
      apply (auto)
      by (metis (no-types, opaque-lifting) antisym-conv3 class-field.neg-1-not-0
equal-neg-zero less-irrefl of-int-minus)
    qed
    have c-c2:  $\text{length } ?zer\text{-list} > 0 \implies \exists k \in (\text{set } S). \text{consistent-sign-vec } qs \ k$ 
    =  $\text{consistent-sign-vec } qs \ y$ 
    proof -
      assume lengt:  $\text{length } ?zer\text{-list} > 0$ 
      let ?t =  $\exists k \in (\text{set } S). \text{consistent-sign-vec } qs \ k = \text{consistent-sign-vec } qs \ y$ 
      have sg1:  $(\exists w. w < \text{length } ?zer\text{-list} \wedge y = ?zer\text{-list } ! \ w) \implies ?t$ 
      proof -
        assume  $(\exists w. w < \text{length } ?zer\text{-list} \wedge y = ?zer\text{-list } ! \ w)$ 
        then obtain w where w-prop:  $w < \text{length } ?zer\text{-list} \wedge y = ?zer\text{-list } ! \ w$ 
      by auto
      then have  $y \in \{x. \exists q \in \text{set } qs. q \neq 0 \wedge \text{poly } q \ x = 0\}$ 
      using finset set-sorted-list-of-set[of  $\{x. \exists q \in \text{set } qs. q \neq 0 \wedge \text{poly } q \ x =$ 
    0}]
      by (smt (verit, best) nth-mem)
      then have  $y \in \{x. \text{poly } (\text{poly-f } qs) \ x = 0\}$  using poly-f-is
      using  $\langle \neg (\exists q \in \text{set } qs. q \neq 0 \wedge \text{poly } q \ y = 0) \rangle$  by blast
      then show ?thesis using set-S-char
      by blast
    qed
    have sg2:  $(y < ?zer\text{-list } ! \ 0) \implies ?t$ 
    proof -
      assume ylt:  $y < ?zer\text{-list } ! \ 0$ 
      have ynonzat-some-qs:  $\forall q \in (\text{set } qs). q \neq 0 \longrightarrow \text{poly } q \ y \neq 0$ 
      proof clarsimp
        fix q
        assume q-in:  $q \in \text{set } qs$ 
        assume qnonz:  $q \neq 0$ 
        assume  $\text{poly } q \ y = 0$ 
        then have  $y \in \{x. \exists q \in \text{set } qs. q \neq 0 \wedge \text{poly } q \ x = 0\}$ 
        using q-in qnonz by auto
        then have List.member  $?zer\text{-list } y$ 
        by (auto simp add: finset)
        then have  $y \geq ?zer\text{-list } ! \ 0$  using strict-sorted-h
        using  $\langle \neg (\exists q \in \text{set } qs. q \neq 0 \wedge \text{poly } q \ y = 0) \rangle \langle \text{poly } q \ y = 0 \rangle$  q-in
      qnonz by blast
      then show False using ylt
      by auto
    qed
    let ?ncrb =  $(- \text{crb } (\text{prod-list-var } qs))$ 
    have  $\forall x \in \{x. \exists q \in \text{set } qs. q \neq 0 \wedge \text{poly } q \ x = 0\}. \text{poly } (\text{prod-list-var } qs)$ 
     $x = 0$ 
      using q-dvd-prod-list-var-prop

```

```

    by fastforce
  then have poly (prod-list-var qs) (sorted-list-of-set {x.  $\exists q \in \text{set } qs. q \neq 0$ 
 $\wedge \text{poly } q \ x = 0$ } ! 0) = 0
    using finset set-sorted-list-of-set
    by (metis (no-types, lifting) lengt nth-mem)
    then have ncrblt: ?ncrb < ?zer-list ! 0 using prod-list-var-nonzero
    crb-lem-neg[of prod-list-var qs ?zer-list ! 0]
    by auto
  have qzerh:  $\forall q \in (\text{set } qs). q = 0 \longrightarrow \text{squash } (\text{poly } q \ ?ncrb) = \text{squash}$ 
  (poly q y)
    by auto
  have  $\forall q \in (\text{set } qs). q \neq 0 \longrightarrow \text{squash } (\text{poly } q \ ?ncrb) = \text{squash } (\text{poly } q \ y)$ 
  proof clarsimp
    fix q
    assume q-in:  $q \in \text{set } qs$ 
    assume qnonz:  $q \neq 0$ 
    have nonzylt:  $\neg(\exists x \leq y. \text{poly } q \ x = 0)$ 
    proof clarsimp
      fix x
      assume xlt:  $x \leq y$ 
      assume poly q x = 0
      then have  $x \in \{x. \exists q \in \text{set } qs. q \neq 0 \wedge \text{poly } q \ x = 0\}$ 
        using q-in qnonz by auto
      then have List.member ?zer-list x
        by (auto simp add: finset)
      then have  $x \geq ?zer-list ! 0$ 
        using strict-sorted-h by auto
      (metis (lifting) bot-nat-0.extremum in-set-conv-nth sorted-iff-nth-mono
        sorted-list-of-set.sorted-sorted-key-list-of-set)
      then show False using xlt ylt
        by auto
    qed
  have nonzncrb:  $\neg(\exists x \leq (\text{real-of-int } ?ncrb). \text{poly } q \ x = 0)$ 
  proof clarsimp
    fix x
    assume xlt:  $x \leq - \text{real-of-int } (\text{crb } (\text{prod-list-var } qs))$ 
    assume poly q x = 0
    then have  $x \in \{x. \exists q \in \text{set } qs. q \neq 0 \wedge \text{poly } q \ x = 0\}$ 
      using q-in qnonz by auto
    then have List.member ?zer-list x
      by (auto simp add: finset)
    then have  $x \geq ?zer-list ! 0$ 
      using strict-sorted-h by (auto simp add: finset)
    (metis (lifting)  $\langle \forall x \in \{x. \exists q \in \text{set } qs. q \neq 0 \wedge \text{poly } q \ x = 0\}. \text{poly}$ 
    (prod-list-var qs)  $x = 0 \rangle$ 
       $\langle x \in \{x. \exists q \in \text{set } qs. q \neq 0 \wedge \text{poly } q \ x = 0\} \rangle$  crb-lem-neg
    linorder-not-le of-int-minus
      prod-list-var-nonzero xlt)
    then show False using xlt ncrblt

```

```

    by auto
  qed
  have c1: (poly q ?ncrb) > 0  $\implies$  (poly q y) > 0
  proof -
    assume qncrbgt: (poly q ?ncrb) > 0
    then have eq: ?ncrb = y  $\implies$  poly q y > 0 by auto
      have gt: ?ncrb > y  $\implies$  poly q y > 0 using qncrbgt qnonz
poly-IVT-pos[of y ?ncrb q] poly-IVT-neg[of ?ncrb y q] nonznocrb nonzyllt
      apply (auto)
      by (meson less-eq-real-def linorder-neqE-linordered-idom)
    have lt: ?ncrb < y  $\implies$  poly q y > 0 using qncrbgt
      using qnonz poly-IVT-pos[of y ?ncrb q] poly-IVT-neg[of ?ncrb y q]
nonznocrb nonzyllt
      apply (auto)
      by (meson less-eq-real-def linorder-neqE-linordered-idom)
    then show ?thesis using eq gt lt apply (auto)
      by (meson linorder-neqE-linordered-idom)
  qed
  have c2: (poly q ?ncrb) < 0  $\implies$  (poly q y) < 0
    using poly-IVT-pos[of ?ncrb y q] poly-IVT-neg[of y ?ncrb q] nonznocrb
nonzyllt
    apply (auto)
    by (metis less-eq-real-def linorder-neqE-linordered-idom)
  have eo: (poly q ?ncrb) > 0  $\vee$  (poly q ?ncrb) < 0
    using nonznocrb
    by auto
  then show squash (poly q (- real-of-int (crb (prod-list-var qs)))) =
squash (poly q y)
    using c1 c2
    by (smt (verit, ccfv-SIG) of-int-minus squash-def)
  qed
  then have  $\forall q \in (\text{set } qs). \text{squash } (\text{poly } q \text{ ?ncrb}) = \text{squash } (\text{poly } q \text{ y})$ 
    using qzerh by auto
  then have consistent-sign-vec qs ?ncrb = consistent-sign-vec qs y
    unfolding consistent-sign-vec-def squash-def
    by (smt (z3) map-eq-conv)
  then show ?thesis using neg-crb-in by auto
  qed
  have sg3: (y > ?zer-list ! (length ?zer-list - 1))  $\implies$  ?t
  proof -
    assume ygt: y > ?zer-list ! (length ?zer-list - 1)
    have ynonzat-some-qs:  $\forall q \in (\text{set } qs). q \neq 0 \longrightarrow \text{poly } q \text{ y} \neq 0$ 
  proof clarsimp
    fix q
    assume q-in: q  $\in$  set qs
    assume qnonz: q  $\neq$  0
    assume poly q y = 0
    then have y  $\in$  {x.  $\exists q \in \text{set } qs. q \neq 0 \wedge \text{poly } q \text{ x} = 0$ }
      using q-in qnonz by auto
  end
end

```

```

then have List.member ?zer-list y
  by (auto simp add: finset)

then have  $y \leq ?zer-list ! (length ?zer-list - 1)$  using strict-sorted-h
  using  $\langle \neg (\exists q \in set\ qs. q \neq 0 \wedge poly\ q\ y = 0) \rangle \langle poly\ q\ y = 0 \rangle q\text{-in}$ 
qnonz by blast
then show False using ygt
  by auto
qed
let ?crb = crb (prod-list-var qs)
have  $\forall x \in \{x. \exists q \in set\ qs. q \neq 0 \wedge poly\ q\ x = 0\}. poly\ (prod-list-var\ qs)$ 
 $x = 0$ 
  using q-dvd-prod-list-var-prop
  by fastforce
then have  $poly\ (prod-list-var\ qs)\ (sorted-list-of-set\ \{x. \exists q \in set\ qs. q \neq 0$ 
 $\wedge poly\ q\ x = 0\} ! 0) = 0$ 
  using finset set-sorted-list-of-set
  by (metis (no-types, lifting) lengt nth-mem)
then have crbgt:  $?crb > ?zer-list ! (length ?zer-list - 1)$  using
prod-list-var-nonzero crb-lem-pos[of prod-list-var qs ?zer-list ! (length ?zer-list -
1)]
  by (metis (no-types, lifting)  $\langle \forall x \in \{x. \exists q \in set\ qs. q \neq 0 \wedge poly\ q\ x = 0\}. poly\ (prod-list-var\ qs)\ x = 0 \rangle$ 
diff-less finset lengt less-numeral-extra(1) nth-mem
set-sorted-list-of-set)
have qzerh:  $\forall q \in (set\ qs). q = 0 \longrightarrow squash\ (poly\ q\ ?crb) = squash\ (poly$ 
 $q\ y)$ 
  by auto
have  $\forall q \in (set\ qs). q \neq 0 \longrightarrow squash\ (poly\ q\ ?crb) = squash\ (poly\ q\ y)$ 
proof clarsimp
fix q
assume q-in:  $q \in set\ qs$ 
assume qnonz:  $q \neq 0$ 
have nonzylt:  $\neg(\exists x \geq y. poly\ q\ x = 0)$ 
proof clarsimp
fix x
assume xgt:  $x \geq y$ 
assume poly q x = 0
then have  $x \in \{x. \exists q \in set\ qs. q \neq 0 \wedge poly\ q\ x = 0\}$ 
  using q-in qnonz by auto
then have List.member ?zer-list x
  by (auto simp add: finset)
then have  $x \leq ?zer-list ! (length ?zer-list - 1)$ 
  using strict-sorted-h lengt
  by (auto simp add: in-set-conv-nth sorted-nth-mono)
then show False using xgt ygt
  by auto
qed
have nonzcrb:  $\neg(\exists x \geq (real-of-int\ ?crb). poly\ q\ x = 0)$ 
proof clarsimp

```

```

fix x
assume xgt:  $x \geq \text{real-of-int } (\text{crb } (\text{prod-list-var } \text{qs}))$ 
assume poly q x = 0
then have  $x \in \{x. \exists q \in \text{set } \text{qs}. q \neq 0 \wedge \text{poly } q \ x = 0\}$ 
  using q-in qnonz by auto
then have List.member ?zer-list x
  by (auto simp add: finset)
then have  $x \leq \text{?zer-list ! } (\text{length } \text{?zer-list} - 1)$  using strict-sorted-h
  by (meson  $\langle \forall x \in \{x. \exists q \in \text{set } \text{qs}. q \neq 0 \wedge \text{poly } q \ x = 0\}. \text{poly}$ 
  (prod-list-var qs)  $x = 0 \rangle \langle x \in \{x. \exists q \in \text{set } \text{qs}. q \neq 0 \wedge \text{poly } q \ x = 0\} \rangle$  crb-lem-pos
  not-less prod-list-var-nonzero xgt)
  then show False using xgt crbgt
  by auto
qed
have c1:  $(\text{poly } q \ \text{?crb}) > 0 \implies (\text{poly } q \ y) > 0$ 
proof -
  assume qcrbgt:  $(\text{poly } q \ \text{?crb}) > 0$ 
  then have eq:  $\text{?crb} = y \implies \text{poly } q \ y > 0$  by auto
  have gt:  $\text{?crb} > y \implies \text{poly } q \ y > 0$  using qcrbgt qnonz poly-IVT-pos[of
  y ?crb q] poly-IVT-neg[of ?crb y q] nonzcrb nonzyllt
  apply (auto)
  by (meson less-eq-real-def linorder-neqE-linordered-idom)
  have lt:  $\text{?crb} < y \implies \text{poly } q \ y > 0$  using qcrbgt
  using qnonz poly-IVT-pos[of y ?crb q] poly-IVT-neg[of ?crb y q]
  nonzcrb nonzyllt
  apply (auto)
  by (meson less-eq-real-def linorder-neqE-linordered-idom)
  then show ?thesis using eq gt lt apply (auto)
  by (meson linorder-neqE-linordered-idom)
qed
have c2:  $(\text{poly } q \ \text{?crb}) < 0 \implies (\text{poly } q \ y) < 0$ 
  using poly-IVT-pos[of ?crb y q] poly-IVT-neg[of y ?crb q] nonzcrb
  nonzyllt
  apply (auto)
  by (metis less-eq-real-def linorder-neqE-linordered-idom)
  have eo:  $(\text{poly } q \ \text{?crb}) > 0 \vee (\text{poly } q \ \text{?crb}) < 0$ 
  using nonzcrb
  by auto
  then show squash  $(\text{poly } q \ (\text{real-of-int } (\text{crb } (\text{prod-list-var } \text{qs})))) = \text{squash}$ 
  (poly q y)
  using c1 c2
  by (smt (verit, ccfv-SIG) of-int-minus squash-def)
qed
then have  $\forall q \in (\text{set } \text{qs}). \text{squash } (\text{poly } q \ \text{?crb}) = \text{squash } (\text{poly } q \ y)$ 
  using qzerh by auto
then have consistent-sign-vec qs ?crb = consistent-sign-vec qs y
  unfolding consistent-sign-vec-def squash-def
  by (smt (z3) map-eq-conv)
then show ?thesis using pos-crb-in by auto

```

```

qed
have sg4: (∃ k < (length ?zer-list - 1). y > ?zer-list ! k ∧ y < ?zer-list
! (k+1)) ⇒ ?t
proof -
  assume (∃ k < (length ?zer-list - 1). y > ?zer-list ! k ∧ y < ?zer-list
! (k+1))
  then obtain k where k-prop: k < (length ?zer-list - 1) ∧ y > ?zer-list
! k ∧ y < ?zer-list ! (k+1)
  by auto
  have ltk: (?zer-list ! k) < (?zer-list ! (k+1))
  using strict-sorted-h
  using k-prop by linarith
  have q1e: (∃ q1 ∈ set qs. q1 ≠ 0 ∧ poly q1 (?zer-list ! k) = 0)
  by (smt (z3) One-nat-def Suc-lessD add.right-neutral add-Suc-right
finset k-prop less-diff-conv mem-Collect-eq nth-mem set-sorted-list-of-set)
  have q2e: (∃ q2 ∈ set qs. q2 ≠ 0 ∧ poly q2 (?zer-list ! (k + 1)) = 0)
  by (smt (verit, del-insts) finset k-prop less-diff-conv mem-Collect-eq
nth-mem set-sorted-list-of-set)
  then have (∃ q > (?zer-list ! k). q < (?zer-list ! (k + 1)) ∧ poly (poly-f
qs) q = 0)
  using poly-f-roots-prop-1[of qs] q1e q2e ltk is-not-const
  by auto
  then have ∃ s ∈ set S. s > ?zer-list ! k ∧ s < ?zer-list ! (k+1)
  using poly-f-is
  by (smt (z3) k-prop mem-Collect-eq set-S-char)
  then obtain s where s-prop: s ∈ set S ∧ s > ?zer-list ! k ∧ s < ?zer-list
! (k+1) by auto
  have qnon: ∀ q ∈ set qs. q ≠ 0 → squash (poly q s) = squash (poly q y)
  proof clarsimp
  fix q
  assume q-in: q ∈ set qs
  assume qnonz: q ≠ 0
  have sgt: s > y ⇒ squash (poly q s) = squash (poly q y)
  proof -
    assume s > y
    then have ∄ x. List.member ?zer-list x ∧ y ≤ x ∧ x ≤ s
    using sorted-list-lemma[of y s k ?zer-list] k-prop strict-sorted-h s-prop
y-prop
    using less-diff-conv by blast
    then have nox: ∄ x. poly q x = 0 ∧ y ≤ x ∧ x ≤ s using q-in qnonz
by (auto simp add: finset)
    then have c1: poly q s ≠ 0 using s-prop q-in qnonz
by (metis (mono-tags, lifting) ⟨y < s⟩ less-eq-real-def )
    have c2: poly q s > 0 ⇒ poly q y > 0
    using poly-IVT-pos poly-IVT-neg nox
by (meson ⟨y < s⟩ less-eq-real-def linorder-neqE-linordered-idom)
    have c3: poly q s < 0 ⇒ poly q y < 0 using poly-IVT-pos
poly-IVT-neg nox
by (meson ⟨y < s⟩ less-eq-real-def linorder-neqE-linordered-idom)

```

```

    show ?thesis using c1 c2 c3 unfolding squash-def
      by auto
  qed
  have slt:  $s < y \implies \text{squash } (\text{poly } q \ s) = \text{squash } (\text{poly } q \ y)$ 
  proof -
    assume slt:  $s < y$ 
    then have  $\nexists x. \text{List.member } ?\text{zer-list } x \wedge s \leq x \wedge x \leq y$ 
    using sorted-list-lemma[of  $s \ y \ k \ ?\text{zer-list}$ ] k-prop strict-sorted-h s-prop
  y-prop
    using less-diff-conv by blast
  then have nox:  $\nexists x. \text{poly } q \ x = 0 \wedge s \leq x \wedge x \leq y$  using q-in qnonz
    by (auto simp add: finset)
  then have c1:  $\text{poly } q \ s \neq 0$  using s-prop q-in qnonz
    by (metis (mono-tags, lifting)  $\langle s < y \rangle$  less-eq-real-def )
  have c2:  $\text{poly } q \ s > 0 \implies \text{poly } q \ y > 0$ 
    using poly-IVT-pos poly-IVT-neg nox
    by (meson  $\langle s < y \rangle$  less-eq-real-def linorder-neqE-linordered-idom)
  have c3:  $\text{poly } q \ s < 0 \implies \text{poly } q \ y < 0$  using poly-IVT-pos
  poly-IVT-neg nox
    by (meson  $\langle s < y \rangle$  less-eq-real-def linorder-neqE-linordered-idom)
  show ?thesis using c1 c2 c3 unfolding squash-def
    by auto
  qed
  have  $s = y \implies \text{squash } (\text{poly } q \ s) = \text{squash } (\text{poly } q \ y)$ 
    by auto
  then show  $\text{squash } (\text{poly } q \ s) = \text{squash } (\text{poly } q \ y)$ 
    using sgt slt
    by (meson linorder-neqE-linordered-idom)
  qed
  have  $\forall q \in \text{set } qs. q = 0 \longrightarrow \text{squash } (\text{poly } q \ s) = \text{squash } (\text{poly } q \ y)$  by
  auto
  then have  $\forall q \in \text{set } qs. \text{squash } (\text{poly } q \ s) = \text{squash } (\text{poly } q \ y)$ 
    using qnon
    by fastforce
  then show ?thesis
    using s-prop unfolding squash-def consistent-sign-vec-def apply (auto)
    by (metis (no-types, opaque-lifting) class-field.neg-1-not-0 equal-neg-zero
  less-irrefl linorder-neqE-linordered-idom)
  qed
  show ?thesis
    using lengt sg1 sg2 sg3 sg4 len-gtz-prop is-not-const
    by fastforce
  qed
  show  $\exists k \in (\text{set } S). \text{consistent-sign-vec } qs \ k = \text{consistent-sign-vec } qs \ y$ 
    using c-c1 c-c2 by auto
  qed
  show ?thesis
    using c1 c2 by auto
  qed

```

```

    then show  $x \in \text{consistent-sign-vectors-R } qs \text{ (set } S)$ 
      using y-prop unfolding consistent-sign-vectors-R-def
      by (metis imageI)
  qed
  have easy-direction: consistent-sign-vectors-R } qs \text{ (set } S) \subseteq \text{consistent-sign-vectors-R } qs \text{ UNIV}
    using consistent-sign-vectors-R-def by auto
  then show ?thesis using difficult-direction easy-direction by auto
  qed

lemma main-step-aux2-R:
  fixes qs:: real poly list
  assumes is-not-const: check-all-const-deg } qs = False
  shows set (find-consistent-signs-R } qs) = consistent-sign-vectors-R } qs UNIV
proof -
  have poly-f-is: poly-f } qs = (pderiv (prod-list-var } qs)) * (prod-list-var } qs)* ([:-(crb (prod-list-var } qs)),1:]) * ([:(crb (prod-list-var } qs)),1:])
    using is-not-const unfolding poly-f-def by auto
  let ?p = (pderiv (prod-list-var } qs)) * (prod-list-var } qs)* ([:-(crb (prod-list-var } qs)),1:]) * ([:(crb (prod-list-var } qs)),1:])
  let ?S = characterize-root-list-p (pderiv (prod-list-var } qs) * (prod-list-var } qs) * ([:-(crb (prod-list-var } qs)),1:]) * ([:(crb (prod-list-var } qs)),1:])))
  have set (remdups (map (signs-at } qs) ?S)) = consistent-sign-vectors-R } qs (set ?S)
    unfolding signs-at-def squash-def consistent-sign-vectors-R-def consistent-sign-vec-def
    by (smt (verit, best) comp-apply map-eq-conv set-map set-remdups)
  then have set (characterize-consistent-signs-at-roots ?p } qs) = consistent-sign-vectors-R } qs UNIV
    unfolding characterize-consistent-signs-at-roots-def using assms all-sample-points-prop[of } qs]
    by auto
  then show ?thesis
    unfolding find-consistent-signs-R-def using find-consistent-signs-at-roots-R poly-f-is poly-f-nonzero[of } qs]
    by auto
  qed

lemma main-step-R:
  fixes qs:: real poly list
  shows set (find-consistent-signs-R } qs) = consistent-sign-vectors-R } qs UNIV
  using main-step-aux1-R main-step-aux2-R by auto

lemma consistent-sign-vec-semantics-R:
  assumes  $\bigwedge i. i \in \text{set-fml } fml \implies i < \text{length } ls$ 
  shows lookup-sem } fml (map (\lambda p. poly p } x) } ls) = lookup-sem } fml (consistent-sign-vec } ls } x)
  using assms apply (induction)

```

by (auto simp add: consistent-sign-vec-def)

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 $(\forall x::\text{real}. \text{lookup-sem } fml (\text{map } (\lambda p. \text{poly } p \ x) \ qs)) \longleftrightarrow$

$\text{list-all } (\text{lookup-sem } fml) \ \text{signs}$

using *assms(2)* **unfolding** *consistent-sign-vectors-R-def list-all-iff*

by (*simp add: assms(1) consistent-sign-vec-semantics-R*)

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 $(\exists x::\text{real}. \text{lookup-sem } fml (\text{map } (\lambda p. \text{poly } p \ x) \ qs)) \longleftrightarrow$

$\text{find } (\text{lookup-sem } fml) \ \text{signs} \neq \text{None}$

using *assms(2)* **unfolding** *consistent-sign-vectors-R-def find-None-iff*

by (*simp add: assms(1) consistent-sign-vec-semantics-R*)

lemma *decide-univ-lem-helper-R:*

fixes *fml:: real poly fml*

assumes $(\text{fml-struct}, \text{polys}) = \text{convert } fml$

shows $(\forall x::\text{real}. \text{lookup-sem } \text{fml-struct} (\text{map } (\lambda p. \text{poly } p \ x) \ \text{polys})) \longleftrightarrow (\text{decide-universal-R } fml)$

using *assms universal-lookup-sem-R main-step-R* **unfolding** *decide-universal-R-def*

apply (*auto*)

apply (*metis assms convert-closed fst-conv snd-conv*)

by (*metis (full-types) assms convert-closed fst-conv snd-conv*)

lemma *decide-exis-lem-helper-R:*

fixes *fml:: real poly fml*

assumes $(\text{fml-struct}, \text{polys}) = \text{convert } fml$

shows $(\exists x::\text{real}. \text{lookup-sem } \text{fml-struct} (\text{map } (\lambda p. \text{poly } p \ x) \ \text{polys})) \longleftrightarrow (\text{decide-existential-R } fml)$

using *assms existential-lookup-sem-R main-step-R* **unfolding** *decide-existential-R-def*

apply (*auto*)

apply (*metis assms convert-closed fst-conv snd-conv*)

by (*metis (full-types) assms convert-closed fst-conv snd-conv*)

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 $\text{real-sem } fml \ x = \text{lookup-sem } (\text{map-fml } (\text{index-of } ps) \ fml) \ ls$

using *assms* **apply** (*induct fml*)

by *auto*

lemma *convert-semantics-R:*

shows $\text{real-sem } fml \ x = \text{lookup-sem } (\text{fst } (\text{convert } fml)) (\text{map } (\lambda p. \text{poly } p \ x) (\text{snd } (\text{convert } fml)))$

unfolding *convert-def Let-def* **apply** *simp*

apply (*intro convert-semantics-lem-R*)
by (*simp add: index-of-lookup(1) index-of-lookup(2)*)

theorem *decision-procedure-R*:

shows $(\forall x::\text{real. real-sem fml } x) \longleftrightarrow (\text{decide-universal-R fml})$
 $\exists x::\text{real. real-sem fml } x \longleftrightarrow (\text{decide-existential-R fml})$
using *convert-semantics-lem-R decide-univ-lem-helper-R* **apply** (*auto*)
apply (*simp add: convert-semantics-R*)
apply (*metis convert-def convert-semantics-R fst-conv snd-conv*)
using *convert-semantics-lem-R*
by (*metis convert-def convert-semantics-R decide-exis-lem-helper-R fst-conv snd-conv*)

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.