The BKR Decision Procedure for Univariate Real Arithmetic

Katherine Cordwell, Yong Kiam Tan, and André Platzer

March 17, 2025

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

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

1 Kronecker Product

definition *kronecker-product* :: *'a* :: *ring mat* ⇒ *'a mat* ⇒ *'a mat* **where**
kronecker-product A B =
(*let ra* = *dim-row A*; *ca* = *dim-col A*;
rb = *dim-row B*; *cb* = *dim-col B*
in
mat (*ra*∗*rb*) (*ca*∗*cb*)
(λ(*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-kronecker*[*simp*]:
dim-row (*kronecker-product A B*) = *dim-row A* ∗ *dim-row B*
dim-col (*kronecker-product A B*) = *dim-col A* ∗ *dim-col B*
unfolding *kronecker-product-def Let-def* **by** *auto*

lemma *kronecker-inverse-index*:
assumes *r* < *dim-row A s* < *dim-col A*
assumes *v* < *dim-row B w* < *dim-col B*
shows *kronecker-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 ⟹ mat nr nc f $$ (i mod nr,j mod nc) = f (i mod nr,j mod 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*)
by (*smt* (*verit*, *best*) *div-less-iff-less-mult div-mult2-eq kronecker-inverse-index linordered-semiring-strict-class.mult-pos-pos mod-less-divisor mod-mult2-eq mult.assoc mult.commute mult-div-mod-eq*)

lemma *sum-sum-mod-div*:
(∑ *ia = 0*::*nat*..<*x*. ∑ *ja = 0*..<*y*. *f ia ja*) =
(∑ *ia = 0*..<*x*∗*y*. *f* (*ia div y*) (*ia mod y*))
proof −
have *1*: *inj-on* (λ*ia*. (*ia div y*, *ia mod y*)) {*0*..<*x * y*}
by (*smt* (*verit*, *best*) *Pair-inject div-mod-decomp inj-onI*)
have *21*: {*0*..<*x*} × {*0*..<*y*} ⊆ (λ*ia*. (*ia div y*, *ia mod y*)) ‘ {*0*..<*x * y*}
proof *clarsimp*
fix *a b*
assume ∗:*a < x b < y*
have *a * y + b* ∈ {*0*..<*x*∗*y*}
by (*metis arith * atLeastLessThan-iff le0 mult.commute*)
thus (*a*, *b*) ∈ (λ*ia*. (*ia div y*, *ia mod y*)) ‘ {*0*..<*x * y*}
using ∗ **by** (*auto simp add*: *image-iff*)

```
    (metis ‹a * y + b ∈ {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:(λia. (ia div y, ia mod y)) ` {0..<x * y} ⊆ {0..<x} × {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} × {0..<y} = (λia. (ia div y, ia mod y)) ` {0..<x * y}
    using 21 22 by auto
  have *: (∑ ia = 0::nat..<x. ∑ ja = 0..<y. f ia ja) =
        (∑ (x, y)∈{0..<x} × {0..<y}. f x y)
    by (auto simp add: sum.cartesian-product)
  show ?thesis unfolding *
    apply (intro sum.reindex-cong[of λia. (ia div y, ia 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 < dim-row A * dim-row B j < dim-col C * dim-col D
  have 1: (A * C) $$ (i div dim-row B, j div dim-col D) =
    row A (i div dim-row B) · col C (j div dim-col D)
    using ij less-mult-imp-div-less by (auto intro!: index-mult-mat)
  have 2: (B * D) $$ (i mod dim-row B, j mod dim-col D) =
    row B (i mod dim-row B) · col D (j mod 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: ⋀x. x < dim-row C * dim-row D ⟹
        A $$ (i div dim-row B, x div dim-row D) *
        B $$ (i mod dim-row B, x mod dim-row D) *
        (C $$ (x div dim-row D, j div dim-col D) *
         D $$ (x mod dim-row D, j mod dim-col D)) =
        row A (i div dim-row B) $ (x div dim-row D) *
        col C (j div dim-col D) $ (x div dim-row D) *
        (row B (i mod dim-row B) $ (x mod dim-row D) *
         col D (j mod dim-col D) $ (x mod dim-row D))
  proof −
    fix x
    assume *:x < dim-row C * dim-row D
    have 1: row A (i div dim-row B) $ (x div dim-row D) = A $$ (i div dim-row
B, x div dim-row D)
      by (simp add: * assms(1) less-mult-imp-div-less row-def)
```

have *2*: *row B* (*i mod dim-row B*) $ (*x mod dim-row D*) = *B* $$ (*i mod dim-row B*, *x mod dim-row D*)
by (*metis* ∗ *assms*(*2*) *ij*(*1*) *index-row*(*1*) *mod-less-divisor nat-0-less-mult-iff neq0-conv not-less-zero*)
have *3*: *col C* (*j div dim-col D*) $ (*x div dim-row D*) = *C* $$ (*x div dim-row D*, *j div dim-col D*)
by (*simp add*: ∗ *ij*(*2*) *less-mult-imp-div-less*)
have *4*: *col D* (*j mod dim-col D*) $ (*x mod dim-row D*) = *D* $$ (*x mod dim-row D*, *j mod dim-col D*)
by (*metis* ∗ *bot-nat-0.not-eq-extremum ij*(*2*) *index-col mod-less-divisor mult-zero-right not-less-zero*)
show *A* $$ (*i div dim-row B*, *x div dim-row D*) ∗
B $$ (*i mod dim-row B*, *x mod dim-row D*) ∗
(*C* $$ (*x div dim-row D*, *j div dim-col D*) ∗
D $$ (*x mod dim-row D*, *j mod dim-col D*)) =
row A (*i div dim-row B*) $ (*x div dim-row D*) ∗
col C (*j div dim-col D*) $ (*x div dim-row D*) ∗
(*row B* (*i mod dim-row B*) $ (*x mod dim-row D*) ∗
col D (*j mod dim-col D*) $ (*x mod dim-row D*)) **unfolding** *1 2 3 4*
by (*simp add*: *mult.assoc mult.left-commute*)
qed
have ∗: (*A* ∗ *C*) $$ (*i div dim-row B*, *j div dim-col D*) ∗
(*B* ∗ *D*) $$ (*i mod dim-row B*, *j mod dim-col D*) =
(∑ *ia* = *0*..<*dim-row C* ∗ *dim-row D*.
A $$ (*i div dim-row B*, *ia div dim-row D*) ∗
B $$ (*i mod dim-row B*, *ia mod dim-row D*) ∗
(*C* $$ (*ia div dim-row D*, *j div dim-col D*) ∗
D $$ (*ia mod dim-row D*, *j mod dim-col D*)))
unfolding *1 2 scalar-prod-def sum-product sum-sum-mod-div*
apply (*auto simp add*: *sum-product sum-sum-mod-div intro*!: *sum.cong*)
using *3* **by** *presburger*
show *vec* (*dim-col A* ∗ *dim-col B*)
(λ*j*. *A* $$ (*i div dim-row B*, *j div dim-col B*) ∗
B $$ (*i mod dim-row B*, *j mod dim-col B*)) ·
vec (*dim-row C* ∗ *dim-row D*)
(λ*i*. *C* $$ (*i div dim-row D*, *j div dim-col D*) ∗
D $$ (*i mod dim-row D*, *j mod dim-col D*)) =
(*A* ∗ *C*) $$ (*i div dim-row B*, *j div dim-col D*) ∗
(*B* ∗ *D*) $$ (*i mod dim-row B*, *j mod dim-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-row B* = *dim-row A dim-col B* = *dim-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 *Ai* **where** *Ai*: *inverts-mat A Ai inverts-mat Ai A* **using** *assms invertible-mat-def* **by** *blast*
obtain *Bi* **where** *Bi*: *inverts-mat B Bi inverts-mat Bi 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 Ai Bi*)
using *Ai Bi* **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 Ai Bi*) (*kronecker-product A B*)
using *Ai Bi* **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 ⇒ 'a mat*
where *conjugate m = mat* (*dim-row m*) (*dim-col m*) (λ(*i*,*j*). *conjugate* (*m* $$ (*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*]: $m \in$ *carrier-mat nr nc* $\Longrightarrow$ *conjugate* $m \in$ *carrier-mat nr nc*
by (*auto*)

lemma *mat-index-conjugate*[*simp*]:
shows $i <$ *dim-row* $m \Longrightarrow j <$ *dim-col* $m \Longrightarrow$ *conjugate* m \$\$ $(i,j) =$ *conjugate* (m \$\$ (i,j))
unfolding *conjugate-mat-def* **by** *auto*

lemma *row-conjugate*[*simp*]: $i <$ *dim-row* $m \Longrightarrow$ *row* (*conjugate* m) $i =$ *conjugate* (*row* m i)
by (*auto*)

lemma *col-conjugate*[*simp*]: $i <$ *dim-col* $m \Longrightarrow$ *col* (*conjugate* m) $i =$ *conjugate* (*col* m i)
by (*auto*)

lemma *rows-conjugate*: *rows* (*conjugate* m) = *map conjugate* (*rows* m)
by (*simp add*: *list-eq-iff-nth-eq*)

lemma *cols-conjugate*: *cols* (*conjugate* m) = *map conjugate* (*cols* m)
by (*simp add*: *list-eq-iff-nth-eq*)

instance
proof
fix a b :: *'a mat*
show *conjugate* (*conjugate* a) = a
unfolding *mat-eq-iff* **by** *auto*
let *?a* = *conjugate a*
let *?b* = *conjugate b*
show *conjugate a* = *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*::*conjugate mat* $\Rightarrow$ *'a mat*
where *conjugate-transpose* $A \equiv$ *conjugate* (A^T)

notation *conjugate-transpose* (‹(-^{H})› [*1000*])

lemma *transpose-conjugate*:
shows (*conjugate* A)$^T = A^H$
unfolding *conjugate-mat-def*
by *auto*

lemma *vec-module-col-helper*:
fixes A:: (*'a* :: *field*) *mat*

shows (0_v (*dim-row A*) ∈ *LinearCombinations.module.span class-ring* (|*carrier* = *carrier-vec* (*dim-row A*), *mult* = *undefined*, *one* = *undefined*, *zero* = 0_v (*dim-row A*), *add* = (+), *smult* = ($\cdot_v$)|) (*set* (*cols A*)))
proof −
have ∀ *v*. (*0*::'*a*) $\cdot_v$ *v* + *v* = *v*
by *auto*
then show 0_v (*dim-row A*) ∈ *LinearCombinations.module.span class-ring* (|*carrier* = *carrier-vec* (*dim-row A*), *mult* = *undefined*, *one* = *undefined*, *zero* = 0_v (*dim-row A*), *add* = (+), *smult* = ($\cdot_v$)|) (*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* :: *field*) *mat*
shows ⋀*a x*. *x* ∈ *LinearCombinations.module.span class-ring*
(|*carrier* = *carrier-vec* (*dim-row A*), *mult* = *undefined*, *one* = *undefined*,
zero = 0_v (*dim-row A*), *add* = (+), *smult* = ($\cdot_v$)|)
(*set* (*cols A*)) ⟹
(⋀*a b v*. (*a* + *b*) $\cdot_v$ *v* = *a* $\cdot_v$ *v* + *b* $\cdot_v$ *v*) ⟹
a $\cdot_v$ *x*
∈ *LinearCombinations.module.span class-ring*
(|*carrier* = *carrier-vec* (*dim-row A*), *mult* = *undefined*, *one* = *undefined*,
zero = 0_v (*dim-row A*), *add* = (+), *smult* = ($\cdot_v$)|)
(*set* (*cols A*))
proof −
fix *a* :: '*a* **and** *x* :: '*a vec*
assume *x* ∈ *LinearCombinations.module.span class-ring* (|*carrier* = *carrier-vec* (*dim-row A*), *mult* = *undefined*, *one* = *undefined*, *zero* = 0_v (*dim-row A*), *add* = (+), *smult* = ($\cdot_v$)|) (*set* (*cols A*))
then show *a* $\cdot_v$ *x* ∈ *LinearCombinations.module.span class-ring* (|*carrier* = *carrier-vec* (*dim-row A*), *mult* = *undefined*, *one* = *undefined*, *zero* = 0_v (*dim-row A*), *add* = (+), *smult* = ($\cdot_v$)|) (*set* (*cols A*))
by (*metis* (*full-types*) *cols-dim idom-vec.smult-in-span module-vec-def*)
qed

lemma *vec-module-col*: *module* (*class-ring* :: '*a* :: *field ring*)
(*module-vec TYPE*('*a*)
(*dim-row A*)
(|*carrier* :=
LinearCombinations.module.span
class-ring (*module-vec TYPE*('*a*) (*dim-row A*)) (*set* (*cols A*))|))
proof −
interpret *abelian-group module-vec TYPE*('*a*) (*dim-row A*)
(|*carrier* :=
LinearCombinations.module.span
class-ring (*module-vec TYPE*('*a*) (*dim-row A*)) (*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 vectorspace.span-closed*)
unfolding *Units-def* **apply** *auto*
by (*metis* (*no-types*, *opaque-lifting*) *cols-dim comm-add-vec module-vec-def partial-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 $S \subseteq$ *set ls*
obtains *ss* **where** *distinct ss* $S =$ *set ss*
using *assms finite-distinct-list infinite-super* **by** *blast*

lemma *mul-mat-of-cols*:
assumes $A \in$ *carrier-mat nr n*
assumes $\bigwedge j.\ j <$ *length cs* $\Longrightarrow$ *cs* ! $j \in$ *carrier-vec n*
shows $A *$ (*mat-of-cols n cs*) = *mat-of-cols nr* (*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* :: {*conjugatable-ring*, *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* :: {*conjugatable-ring*, *comm-ring*} *vec*
assumes $A \in$ *carrier-mat nr nc*
assumes $x \in$ *carrier-vec nr*
assumes $y \in$ *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* :: {*conjugatable-ordered-ring*,*semiring-no-zero-divisors*,*comm-ring*} *vec*
assumes $A \in$ *carrier-mat nr nc*
assumes $v \in$ *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 ... = *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*) (λj. (*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 $\bigwedge x.\ x \in set\ ls \Longrightarrow dim\text{-}vec\ x = nr$
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-of-cols nr (*permute-list f ss*) $*_v$ *vec-of-list* (*permute-list f v*) =
mat-of-cols nr ss $*_v$ *vec-of-list v*
unfolding *mat-of-cols-def mult-mat-vec-def vec-eq-iff scalar-prod-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 ((λ*ia*. *ss* ! *f ia* $ *i* ∗ *v* ! *f ia*) ∘ *f*) {..<*length ss*} **.**
also have ... = ($\sum$ *ia* = *0*..<*length v*. *ss* ! *f ia* $ *i* ∗ *v* ! *f ia*)
using *assms*(*2*) *calculation lessThan-atLeast0* **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-tags*, *lifting*) ‹$\bigwedge$*g*. *sum g* {..<*length ss*} = *sum* (*g* ∘ *f*) {..<*length ss*}› *assms*(*2*) *comp-apply lessThan-atLeast0 sum.cong*)
show ($\sum$ *ia* = *0*..<*length v*.
vec (*length ss*) (λ*j*. *permute-list f ss* ! *j* $ *i*) $ *ia* ∗
vec-of-list (*permute-list f v*) $ *ia*) =
($\sum$ *ia* = *0*..<*length v*. *vec* (*length ss*) (λ*j*. *ss* ! *j* $ *i*) $ *ia* ∗ *vec-of-list v* $ *ia*)
using *assms* ∗ **by** (*auto simp add*: *permute-list-nth vec-of-list-index*)
qed

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

lemma *subindex-permutation2*:
assumes *distinct ss set ss* ⊆ {..<*length ls*}
obtains *f* **where** *f permutes* {..<*length ls*}

```
    ls = permute-list f (map ((!) ls) (filter (λi. i ∉ set ss) [0..<length ls]) @ map
((!) ls) 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 ⊆ set ls
  obtains ids where distinct ids set ids ⊆ {..<length ls} ss = map ((!) ls) ids
proof −
  let ?ids = map (λi. @j. j < length ls ∧ 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 ⊆ {..<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 (⋀ids. distinct ids ⟹
          set ids ⊆ {..<length ls} ⟹
          ss = map ((!) ls) ids ⟹ thesis) ⟹
    thesis using 1 2 3 by blast
qed

lemma helper3:
  fixes A ::'a::comm-ring mat
  assumes A: A ∈ carrier-mat nr nc
  assumes ss:distinct ss set ss ⊆ set (cols A)
  assumes v ∈ 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 ⊆ {..<length (cols A)}
    and ss: ss = map ((!) (cols A)) ids by blast
  let ?ls =  map ((!) (cols A)) (filter (λi. i ∉ set ids) [0..<length (cols A)])
  from subindex-permutation2[OF ids] obtain f where
    f: f permutes {..<length (cols A)}
    cols A = permute-list f (?ls @ ss) using ss by blast
  have *: ⋀x. x ∈ set ?ls ⟹ 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* $*_v$ *vec-of-list* (*permute-list f ?cs1*) **using** *f*(*2*) *assms* **by** *auto*
ultimately show
(⋀*c*. *mat-of-cols nr ss* $*_v$ *v* = *A* $*_v$ *c* ⟹ *dim-vec c* = *nc* ⟹ *thesis*) ⟹ *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* :: {*conjugatable-ordered-ring*,*semiring-no-zero-divisors*,*comm-ring*} *mat*
assumes *A* ∈ *carrier-mat nr nc*
assumes *distinct ss set ss* ⊆ *set* (*cols* (*A*^{H}))
assumes *v* ∈ *carrier-vec* (*length ss*)
assumes *A* $*_v$ (*mat-of-cols nc ss* $*_v$ *v*) = *0*$_v$ *nr*
shows (*mat-of-cols nc ss* $*_v$ *v*) = *0*$_v$ *nc*
proof −
have *A*^{H} ∈ *carrier-mat nc nr* **using** *assms*(*1*) **by** *auto*
from *helper3*[*OF this assms*(*2*−*4*)]
obtain *c* **where** *c*: *mat-of-cols nc ss* $*_v$ *v* = *A*^{H} $*_v$ *c dim-vec c* = *nr* **by** *blast*
have *1*: *c* ∈ *carrier-vec nr*
using *c carrier-vec-dim-vec* **by** *blast*
have *2*: *A* $*_v$ (*A*^{H} $*_v$ *c*) = *0*$_v$ *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*$_v$ *nc* **.**
thus *?thesis* **unfolding** *c*(*1*)[*symmetric*] **.**
qed

lemma *Units-invertible*:
fixes *A*:: '*a*::*semiring-1 mat*
assumes *A* ∈ *Units* (*ring-mat TYPE*('*a*) *n 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*::*semiring-1 mat*
assumes *invertible-mat A*
shows *A* ∈ *Units* (*ring-mat TYPE*('*a*) (*dim-row A*) *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*::*field mat*
assumes *A* ∈ *carrier-mat n 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* < *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* ⊆ *carrier-vec n*
assumes *lincomb-list f ss* = 0_v *n*
assumes *i* < *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* (λ*v*. *sum f* (*set* (*find-indices v 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* (λ*v*. *sum f* (*set* (*find-indices v ss*))) (*set ss*) **by** *auto*
then have ∗:*lincomb* (λ*v*. *sum f* (*set* (*find-indices v 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 (λ*v*. *sum f* (*set* (*find-indices v ss*))) ∈ *set ss* → {*0*}
by *auto*
from *funcset-mem*[*OF this*]
have *sum f* (*set* (*find-indices* (*nth ss i*) *ss*)) ∈ {*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 $A \in$ *carrier-mat n d*
assumes $B \in$ *carrier-mat d nc*
shows *span* (*set* (*cols* (*A* ∗ *B*))) ⊆ *span* (*set* (*cols A*))
proof −
have ∗: ⋀*v*. ∃ *ca*. *lincomb-list v* (*cols* (*A* ∗ *B*)) =
lincomb-list ca (*cols A*)
proof −
fix *v*
have *lincomb-list v* (*cols* (*A* ∗ *B*)) = (*A* ∗ *B*) ∗$_v$ *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 ... = *A* ∗$_v$ (*B* ∗$_v$ *vec nc v*)
using *assms*(*1*) *assms*(*2*) **by** *auto*
also have ... = *lincomb-list* (λ*i*. (*B* ∗$_v$ *vec nc v*) $ *i*) (*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 *lincomb-list v* (*cols* (*A* ∗ *B*)) =
lincomb-list (λ*i*. (*B* ∗$_v$ *vec nc v*) $ *i*) (*cols A*) **by** *auto*
thus ∃ *ca*. *lincomb-list v* (*cols* (*A* ∗ *B*)) = *lincomb-list ca* (*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 $A \in$ *carrier-mat n d*
assumes $B \in$ *carrier-mat d nc*
shows *rank* (*A* ∗ *B*) ≤ *rank A*
proof −
have *subspace class-ring* (*local.span* (*set* (*cols* (*A*∗*B*))))
(*vs* (*local.span* (*set* (*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 *vectorspace.dim class-ring*
(*vs* (*local.span* (*set* (*cols A*))))
(⦇*carrier* := *local.span* (*set* (*cols* (*A* ∗ *B*)))⦈) ≤

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 $\bigwedge v.\ v \in set\ ls \Longrightarrow dim\text{-}vec\ v = n$
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 $\bigwedge v.\ v \in set\ s \Longrightarrow dim\text{-}vec\ v = n$
assumes $\bigwedge i.\ i \in set\ ls \Longrightarrow i < length\ s$
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* ≠ *b*
shows *ls* = [*a*,*b*] ∨ *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* ∨ *x* = *b*
using ‹*ls* = *x* # *xs*› *assms*(*2*) **by** *auto*
have *2*:*y* = *a* ∨ *y* = *b*
using ‹*ls* = *x* # *xs*› ‹*xs* = *y* # *ys*› *assms*(*2*) **by** *auto*
have *3*:*ys* = []
by (*metis* (*no-types*) ‹*ls* = *x* # *xs*› ‹*xs* = *y* # *ys*› *assms*(*1*) *assms*(*2*) *distinct.simps*(*2*) *distinct-length-2-or-more in-set-member member-rec*(*2*) *neq-Nil-conv set-ConsD*)
show *ls* = [*a*, *b*] ∨ *ls* = [*b*, *a*] **using** *ls xs 1 2 3 assms*
by *auto*
qed

lemma *filter-disj-inds*:
assumes *i* < *length ls j* < *length ls i* ≠ *j*
shows *filter* (λ*ia*. *ia* ≠ *j* ⟶ *ia* = *i*) [*0*..<*length ls*] = [*i*, *j*] ∨
filter (λ*ia*. *ia* ≠ *j* ⟶ *ia* = *i*) [*0*..<*length ls*] = [*j*,*i*]
proof −
have *1*: *distinct* (*filter* (λ*ia*. *ia* = *i* ∨ *ia* = *j*) [*0*..<*length ls*])
using *distinct-filter distinct-upt* **by** *blast*
have *2*:*set* (*filter* (λ*ia*. *ia* = *i* ∨ *ia* = *j*) [*0*..<*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 ⋀*v*. *v* ∈ *set ls* ⟹ *dim-vec v* = *n*
assumes
⋀*c*. *lincomb-list c ls* = 0_v *n* ⟹ (∀ *i*. *i* < (*length ls*) ⟶ *c i* = *0*)
shows *distinct ls*
unfolding *distinct-conv-nth*
proof *clarsimp*
fix *i j*

assume *ij*: $i < length\ ls\ j < length\ ls\ i \neq j$
assume *lsij*: $ls\ !\ i = ls\ !\ j$
have *lincomb-list* $(\lambda k.\ if\ k = i\ then\ 1\ else\ if\ k = j\ then\ -1\ else\ 0)\ ls =$
$(ls\ !\ i) - (ls\ !\ j)$
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 ‹$j < length\ ls$› *assms*(*1*)
using *carrier-vec-dim-vec nth-mem* **by** *blast*
ultimately have *lincomb-list* $(\lambda k.\ if\ k = i\ then\ 1\ else\ if\ k = j\ then\ -1\ else\ 0)$
$ls = 0_v\ n$ **by** *auto*
from *assms*(*2*)[*OF this*]
show *False*
using ‹$i < length\ ls$› **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 $A \in carrier\text{-}mat\ n\ nc$
shows $vec\text{-}space.rank\ nc\ A^H \leq rank\ (A * A^H)$
proof −
have *1*: $A^H \in carrier\text{-}mat\ nc\ n$ **using** *assms* **by** *auto*
have *2*: $A * A^H \in carrier\text{-}mat\ n\ n$ **using** *assms* **by** *auto*

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

ultimately show *finite* $S \wedge$ *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* A^H = *card S* **.**

have *s-hyp*: $S \subseteq$ *set* (*cols* A^H)
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*: $\bigwedge$*c*. *lincomb-list c* (*map* (($*_v$) *A*) *ss*) = 0_v *n* $\Longrightarrow$
$\forall$ *i*. (*i* < (*length ss*) $\longrightarrow$ *c i* = *0*)
proof *clarsimp*
fix *c i*
assume $*$: *lincomb-list c* (*map* (($*_v$) *A*) *ss*) = 0_v *n*
assume *i*: *i* < *length ss*
have $\forall$ *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* = 0_v *n*
using $*$ *rw* **by** *auto*
then have *hq*: *A* $*_v$ (*mat-of-cols nc ss* $*_v$ *vec* (*length ss*) *c*) = 0_v *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*) = 0_v *nc*
apply (*intro mat-mul-conjugate-transpose-sub-vec-eq-0*)
using *assms ss s-hyp* **by** *auto*

have *dim-hyp2*: $\forall\, w{\in}set\ ss.\ dim\text{-}vec\ w = nc$
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* $*_v$ *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* ($*_v$) *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* (($*_v$) *A* ‘ *S*)
by (*metis ss distinct-card image-set length-map*)
then have *4*: ($*_v$) *A* ‘ *S* ⊆ *set* (*cols* (*A* ∗ A^H))
using *cols-mat-mul* ‹*S* ⊆ *set* (*cols* A^H)› **by** *blast*
have *5*: *lin-indpt* (($*_v$) *A* ‘ *S*)
proof *clarsimp*
assume *ld*:*lin-dep* (($*_v$) *A* ‘ *S*)
have ∗: *finite* (($*_v$) *A* ‘ *S*)
by (*metis List.finite-set ss2*)
have ∗∗: ($*_v$) *A* ‘ *S* ⊆ *carrier-vec n*
using *2 4 cols-dim* **by** *blast*
from *finite-lin-dep*[*OF* ∗ *ld* ∗∗]
obtain *a v* **where**
a: *lincomb a* (($*_v$) *A* ‘ *S*) = 0_v *n* **and**
v: *v* ∈ ($*_v$) *A* ‘ *S a v* ≠ *0* **by** *blast*
obtain *i* **where** *i*:*v* = *map* (($*_v$) *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* (($*_v$) *A*) *ss*)⊆ *carrier-vec n* **using** ∗∗ *ss2* **by** *auto*
from *lincomb-as-lincomb-list-distinct*[*OF this distinct*] **have**
lincomb-list
(λ*i*. *a* (*map* (($*_v$) *A*) *ss* ! *i*)) (*map* (($*_v$) *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* (($*_v$) *A* ‘ *S*) ≤ *rank* (*A* ∗ A^H) **.**
thus *?thesis* **using** *rankeq 3* **by** *linarith*
qed

lemma *conjugate-transpose-rank-le*:
assumes $A \in$ *carrier-mat n nc*
shows *vec-space.rank nc* $(A^H) \leq$ *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*: $f : U \to$ *carrier-vec n*
shows *conjugate* (*finsum V f U*) = *finsum V* (*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*: $f : U \to$ *carrier-vec n f u* : *carrier-vec n* **by** *auto*
then have *cf*: *conjugate* $\circ$ *f* : $U \to$ *carrier-vec n*
(*conjugate* $\circ$ *f*) *u* : *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.finsum-closed f*(1) **apply** *blast*
by (*simp add*: *comp-def f*(1) *insert.hyps*(3))
qed

lemma *rank-conjugate-le*:
assumes *A*:$A \in$ *carrier-mat n d*
shows *rank* (*conjugate* (*A*)) $\leq$ *rank A*
proof −

let *?P* = $(\lambda T.\ T \subseteq$ *set* (*cols* (*conjugate A*)) $\wedge$ *lin-indpt T*)
have $*$:$\bigwedge A.$ *?P A* $\Longrightarrow$ *finite A* $\wedge$ *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*: *maximal S ?P* **using** $**$
by (*metis* (*no-types*, *lifting*))
have *s-hyp*: *S* $\subseteq$ *set* (*cols* (*conjugate A*)) *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* ⊆ *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* ⊆ *conjugate* ‘ *S finite T* **and**
lc:*lincomb c T* = 0_v *n* **and** *v* ∈ *T c v* ≠ *0*
let *?T* = *conjugate* ‘ *T*
let *?c* = *conjugate* ∘ *c* ∘ *conjugate*
have *1*: *finite ?T* **using** *T* **by** *auto*
have *2*: *?T* ⊆ *S* **using** *T* **by** *auto*
have *3*: *?c* ∈ *?T* → *UNIV* **by** *auto*
have *lincomb ?c ?T* = ($\bigoplus_V x \in T$. *conjugate* (*c x*) ·$_v$ *conjugate x*)
unfolding *lincomb-def*
apply (*subst finsum-reindex*)
apply *auto*
apply (*metis 2 carrier-vec-conjugate assms carrier-matD*(*1*) *cols-dim image-eqI s-hyp*(*1*) *subsetD*)
by (*meson conjugate-cancel-iff inj-onI*)
also have ... = ($\bigoplus_V x \in T$. *conjugate* (*c x* ·$_v$ *x*))
by (*simp add*: *conjugate-smult-vec*)
also have ... = *conjugate* ($\bigoplus_V x \in T$. (*c x* ·$_v$ *x*))
apply(*subst conjugate-finsum*[*of* λ*x*.(*c x* ·$_v$ *x*) *T*])
apply (*auto simp add*:*o-def*)
by (*smt* (*verit*, *ccfv-SIG*) *Matrix.carrier-vec-conjugate Pi-I' T*(*1*) *assms carrier-matD*(*1*) *cols-dim dim-row-conjugate imageE s-hyp*(*1*) *smult-carrier-vec subset-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* ∘ *c* ∘ *conjugate* ∈ *conjugate* ‘ *T* → {*0*} **.**
then have *c v* = *0*
by (*simp add*: *Pi-iff* ‹*v* ∈ *T*›)
thus *False* **using** ‹*c v* ≠ *0*› **by** *auto*
qed
from *rank-ge-card-indpt*[*OF A 1 2*]
have *3*:*card* (*conjugate* ‘ *S*) ≤ *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 $A \in$ *carrier-mat n d*
shows *rank* (*conjugate A*) = *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*::{*conjugatable-ordered-field*} *mat*
shows *vec-space.rank* (*dim-row A*) *A* = *vec-space.rank* (*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*::{*conjugatable-ordered-field*} *mat*
shows *vec-space.rank* (*dim-row A*) *A* = *vec-space.rank* (*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*::{*conjugatable-ordered-field*} *mat*
assumes $A \in$ *carrier-mat n d*
assumes $B \in$ *carrier-mat d nc*
shows *vec-space.rank n* (*A* ∗ *B*) ≤ *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 mat* ⇒ *nat list* ⇒ *'a mat*
where *take-cols A inds* = *mat-of-cols* (*dim-row A*) (*map* ((!) (*cols A*)) (*filter* ((>) (*dim-col A*)) *inds*))

definition *take-cols-var* :: *'a mat* ⇒ *nat list* ⇒ *'a mat*
where *take-cols-var A inds* = *mat-of-cols* (*dim-row A*) (*map* ((!) (*cols A*)) (*inds*))

definition *take-rows* :: *'a mat* ⇒ *nat list* ⇒ *'a mat*
where *take-rows A inds* = *mat-of-rows* (*dim-col A*) (*map* ((!) (*rows A*)) (*filter* ((>) (*dim-row A*)) *inds*))

lemma *cong1*:
x = *y* ⟹ *mat-of-cols n x* = *mat-of-cols n y*

by *auto*

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

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

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

lemma *take-cols-carrier-mat-strict*:
assumes $A \in carrier\text{-}mat\ nr\ nc$
assumes $\bigwedge i.\ i \in set\ inds \Longrightarrow i < nc$
shows $take\text{-}cols\ A\ inds \in carrier\text{-}mat\ nr\ (length\ inds)$
unfolding *take-cols-def*
using *assms* **by** *auto*

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

CD: $C = P * A$ $D = P *$ *take-cols A inds* **by** *blast*
have *P*: $P \in$ *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 $P *$ *take-cols A inds* = *take-cols* $(P * A)$ *inds* **.**
thus *?thesis* **using** *CD* **by** *blast*
qed

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

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

lemma *cols-take-cols-subset*:
shows *set* (*cols* (*take-cols A inds*)) $\subseteq$ *set* (*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 *dim-row* (*take-cols A ls*) = *dim-row A*
by (*simp add*: *take-cols-def*)

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

lemma *rows-inj*:
assumes *dim-col A* = *dim-col B*
assumes *rows A* = *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 *dim-col A* = *dim-col B*
assumes $i <$ *dim-row A* + *dim-row B*
assumes $j <$ *dim-col A*
shows (A @$_r$ B) $$ (i,j) = (*if* $i <$ *dim-row A then A* $$ (i,j) *else B* $$ ($i-$*dim-row*

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

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

lemma *append-rows-mat-mul*:
assumes *dim-col A* = *dim-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 *card* {*i*. *i* < (*n*::*nat*)} ≤ *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 *take-rows A* [*0*..<*length* (*pivot-positions A*)] @$_r$ *0*$_m$ (*dim-row A* − *length* (*pivot-positions A*)) (*dim-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-row A*, **where** *nc* = *dim-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* (λ*y*. *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* (λ*y*. *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: ⋀i j. i < dim-row A ⟹
          j < dim-col A ⟹
          ¬ i < dim-row (take-rows A [0..<length (pivot-positions A)]) ⟹
          0_m (dim-row A − length (pivot-positions A)) (dim-col A) $$
          (i − dim-row (take-rows A [0..<length (pivot-positions A)]), j) =
          A $$ (i, j)
  proof −
    fix i
    fix j
    assume lt-i: i < dim-row A
    assume lt-j: j < dim-col A
    assume not-lt: ¬ i < dim-row (take-rows A [0..<length (pivot-positions A)])
    let ?ip = i+1
    have h0: ∃f. pivot-fun A f (dim-col A) ∧ f i = ?nc
    proof −
     have half1: ∃f. pivot-fun A f (dim-col A) using assms unfolding row-echelon-form-def
        by blast
      have half2: ∀f. pivot-fun A f (dim-col A) ⟶ f i = ?nc
      proof clarsimp
        fix f
        assume is-piv: pivot-fun A f (dim-col A)
        have len-pp: length (pivot-positions A) = card {i. i < ?nr ∧ row A i ≠ 0_v
?nc} using is-piv pivot-positions[of A ?nr ?nc f]
          by auto
        have  ∀i. (i < ?nr ∧ row A i ≠ 0_v ?nc) ⟷  (i < ?nr ∧ f i ≠ ?nc)
          using is-piv pivot-fun-zero-row-iff[of A f ?nc ?nr]
          by blast
       then have len-pp-var: length (pivot-positions A) = card {i. i < ?nr ∧ f i
≠ ?nc}
          using len-pp  by auto
      have allj-hyp: ∀j < ?nr. f j = ?nc ⟶ ((Suc j) < ?nr ⟶ f (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 ≠ ?nc ⟶ (∀j. j ≤ i ⟶ f j ≠ ?nc)
        proof clarsimp
          fix j
          assume not-i: f i ≠ ?nc
          assume j-leq: j ≤ i
          assume bad-asm: f j = ?nc
          have ⋀k. k ≥ j ⟹  k < ?nr ⟹ f k = ?nc
          proof −
            fix k :: nat
            assume a1: j ≤ k
            assume a2: k < dim-row A
            have f3: ⋀n. ¬ n < dim-row A ∨ f n ≠ f j ∨ ¬ Suc n < dim-row A ∨ f
```

$(Suc\ n) = f\ j$
using *allj-hyp bad-asm* **by** *presburger*
obtain *nn* :: *nat* ⇒ *nat* ⇒ (*nat* ⇒ *bool*) ⇒ *nat* **where**
f4: ⋀*n na p nb nc*. (¬ *n* ≤ *na* ∨ *Suc n* ≤ *Suc na*) ∧ (¬ *p nb* ∨ ¬ *nc* ≤ *nb* ∨ ¬ *p* (*nn nc nb p*) ∨ *p nc*) ∧ (¬ *p nb* ∨ ¬ *nc* ≤ *nb* ∨ *p nc* ∨ *p* (*Suc* (*nn nc nb p*)))
using *inc-induct* **by** (*metis Suc-le-mono*)
then have *f5*: ⋀*p*. ¬ *p k* ∨ *p j* ∨ *p* (*Suc* (*nn j k p*))
using *a1* **by** *presburger*
have *f6*: ⋀*p*. ¬ *p k* ∨ ¬ *p* (*nn j k p*) ∨ *p j*
using *f4 a1* **by** *meson*
{ assume *nn j k* (λ*n*. *n* < *dim-row A* ∧ *f n* ≠ *dim-col A*) < *dim-row A* ∧ *f* (*nn j k* (λ*n*. *n* < *dim-row A* ∧ *f n* ≠ *dim-col A*)) ≠ *dim-col A*
moreover
{ assume (*nn j k* (λ*n*. *n* < *dim-row A* ∧ *f n* ≠ *dim-col A*) < *dim-row A* ∧ *f* (*nn j k* (λ*n*. *n* < *dim-row A* ∧ *f n* ≠ *dim-col A*)) ≠ *dim-col A*) ∧ (¬ *j* < *dim-row A* ∨ *f j* = *dim-col A*)
then have ¬ *k* < *dim-row A* ∨ *f k* = *dim-col A*
using *f6*
by (*metis* (*mono-tags*, *lifting*)) **}**
ultimately have (¬ *j* < *dim-row A* ∨ *f j* = *dim-col A*) ∧ (¬ *Suc* (*nn j k* (λ*n*. *n* < *dim-row A* ∧ *f n* ≠ *dim-col A*)) < *dim-row A* ∨ *f* (*Suc* (*nn j k* (λ*n*. *n* < *dim-row A* ∧ *f n* ≠ *dim-col A*))) = *dim-col A*) ∨ ¬ *k* < *dim-row A* ∨ *f k* = *dim-col A*
using *bad-asm*
by *blast* **}**
moreover
{ assume (¬ *j* < *dim-row A* ∨ *f j* = *dim-col A*) ∧ (¬ *Suc* (*nn j k* (λ*n*. *n* < *dim-row A* ∧ *f n* ≠ *dim-col A*)) < *dim-row A* ∨ *f* (*Suc* (*nn j k* (λ*n*. *n* < *dim-row A* ∧ *f n* ≠ *dim-col A*))) = *dim-col A*)
then have ¬ *k* < *dim-row A* ∨ *f k* = *dim-col A*
using *f5*
proof −
have ¬ (*Suc* (*nn j k* (λ*n*. *n* < *dim-row A* ∧ *f n* ≠ *dim-col A*)) < *dim-row A* ∧ *f* (*Suc* (*nn j k* (λ*n*. *n* < *dim-row A* ∧ *f n* ≠ *dim-col A*))) ≠ *dim-col A*) ∧ ¬ (*j* < *dim-row A* ∧ *f j* ≠ *dim-col A*)
using ‹(¬ *j* < *dim-row A* ∨ *f j* = *dim-col A*) ∧ (¬ *Suc* (*nn j k* (λ*n*. *n* < *dim-row A* ∧ *f n* ≠ *dim-col A*)) < *dim-row A* ∨ *f* (*Suc* (*nn j k* (λ*n*. *n* < *dim-row A* ∧ *f n* ≠ *dim-col A*))) = *dim-col A*)› **by** *linarith*
then have ¬ (*k* < *dim-row A* ∧ *f k* ≠ *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 ≠ ?nc ⟶ ({0..<?ip} ⊆ {y. y < ?nr ∧ f y ≠ dim-col A})
      proof −
        have h1: f i ≠ dim-col A ⟶ (∀ j≤i. j < ?nr ∧ f j ≠ dim-col A)
          using if-then-bad lt-i by auto
        then show ?thesis by auto
      qed
       then have gteq: f i ≠ ?nc ⟶ (card {i. i < ?nr ∧ f i ≠ dim-col A} ≥
(i+1))
         using card-lessThan[of ?ip] card-mono[where B = {i. i < ?nr ∧ f i ≠
dim-col A} , where A = {0..<?ip}]
        by auto
      then have clear: dim-row (take-rows A [0..<length (pivot-positions A)]) =
length (pivot-positions A)
        unfolding take-rows-def using dim-asm fileq by (auto)
      have i + 1 > length (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: row A i =  0v (dim-col A)
    using pivot-fun-zero-row-iff[of A - ?nc ?nr]
    using lt-i by blast
  then have h1: A $$ (i, j) = 0
    using index-row(1) lt-i lt-j by fastforce
 have h2a: i − dim-row (take-rows A [0..<length (pivot-positions A)]) < dim-row
A − length (pivot-positions A)
    using pivot-len lt-i not-lt
    by (simp add: take-rows-def)
  then have h2: 0m (dim-row A − length (pivot-positions A)) (dim-col A) $$
        (i − dim-row (take-rows A [0..<length (pivot-positions A)]), j) = 0
    unfolding zero-mat-def using pivot-len lt-i lt-j
    using index-mat(1) by blast
  then show 0m (dim-row A − length (pivot-positions A)) (dim-col A) $$
        (i − dim-row (take-rows A [0..<length (pivot-positions A)]), j) =
        A $$ (i, j) using h1 h2
    by simp
 qed
 have h3: (dim-row (take-rows A [0..<length (pivot-positions A)])::nat) + ((dim-row
A::nat) − (length (pivot-positions A)::nat)) =
   dim-row A
 proof −
    have h0: dim-row (take-rows A [0..<length (pivot-positions A)]) = (length
(pivot-positions A)::nat)
    by (simp add: take-rows-def fileq)
  then show ?thesis using add-diff-inverse-nat  pivot-len
    by linarith
 qed
```

```
  have h4: ⋀i j. i < dim-row A ⟹
        j < dim-col A ⟹
        i < dim-row (take-rows A [0..<length (pivot-positions A)]) +
          (dim-row A − length (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) ≤ dim-row A
proof −
  have 1: A ∈ carrier-mat (dim-row A) (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 < dim-row A ∧ row A i ≠ 0v (dim-col
A)} .
  also have ... ≤ card {i. i < 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 ∈ carrier-mat nr nc
  shows ⋀i j. (i,j) ∈ set (pivot-positions R) ⟹ i < nr ∧ j < nc
proof −
  obtain f where f: pivot-fun R f nc
    using assms(1) assms(2) row-echelon-form-def by blast
  have ∗: ⋀i. i < nr ⟹ f i ≤ 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 ∧ f i ≠ nc} by auto
  then have ∗∗: set (pivot-positions R) = {(i, f i) |i. i < nr ∧ f i < nc}
    using ∗
    by fastforce
  fix i j
  assume (i, j) ∈ set (pivot-positions R)
  thus i < nr ∧ 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 ⋀ i. i < nr ⟹ (⋀ k. ((k < nr ∧ i < k) ⟶ f i ≤ f k))
proof −
  fix i
  assume i < nr
  show (⋀ k. ((k < nr ∧ i < k) ⟶ f i ≤ f k))
  proof −
    fix k
    show ((k < nr ∧ i < k) ⟶ f i ≤ 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-col A ≥ dim-row A
  assumes pivot-fun A f (dim-col A)
  shows ∀ i < (length (pivot-positions A)). (i, f i) ∈ set (pivot-positions A)
proof −
  let ?nr = dim-row A
  let ?nc = dim-col A
  let ?pp = pivot-positions A
  have i-nr: ∀ i < (length ?pp). i < ?nr using rref-pivot-positions assms
    using length-pivot-positions-dim-row less-le-trans by blast
  have i-nc: ∀ i < (length ?pp). f i < ?nc
  proof clarsimp
    fix i
    assume i-lt: i < length ?pp
    have fis-nc: f i = ?nc ⟹ (∀ k > i. k < ?nr ⟶ f k = ?nc)
    proof −
      assume is-nc: f i = ?nc
      show (∀ k > i. k < ?nr ⟶ f k = ?nc)
      proof clarsimp
        fix k
        assume k-gt: k > i
        assume k-lt: k < ?nr
          have fk-lt: f k ≤ ?nc using pivot-funD(1)[of A ?nr f ?nc k] k-lt apply
(auto)
          using ‹pivot-fun A f (dim-col A)› 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 ‹pivot-fun A f (dim-col A)› by auto
    qed
  qed
  have ncimp: f i = ?nc ⟹ (∀ k ≥i. k ∉ { i. i < ?nr ∧ row A i ≠ 0v ?nc})
  proof −
    assume nchyp: f i = ?nc
    show (∀ k ≥i. k ∉ { i. i < ?nr ∧ row A i ≠ 0v ?nc})
    proof clarsimp
      fix k
      assume i-lt: i ≤ k
      assume k-lt: k < dim-row A
      show row A k = 0v (dim-col A)
        using i-lt k-lt fis-nc
        using pivot-fun-zero-row-iff[of A f ?nc ?nr]
        using ‹pivot-fun A f (dim-col A)› le-neq-implies-less nchyp by blast
    qed
  qed
  then have f i = ?nc ⟹ card { i. i < ?nr ∧ row A i ≠ 0v ?nc} ≤ i
  proof −
    assume nchyp: f i = ?nc
    have h: { i. i < ?nr ∧ row A i ≠ 0v ?nc} ⊆ {0..<i}
      using atLeast0LessThan le-less-linear nchyp ncimp by blast
    then show  card { i. i < ?nr ∧ row A i ≠ 0v ?nc} ≤ 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 ‹pivot-fun A f (dim-col A)› i-nr le-neq-implies-less not-less pivot-funD(1))

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

lemma pivot-positions-form-helper-1:
 shows (a, b) ∈ set (pivot-positions-main-gen z A nr nc i j) ⟹ i ≤ 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* ≥ *dim-row A*
shows ∀ *i* < (*length* (*pivot-positions A*)). *fst* ((*pivot-positions A*) ! *i*) = *i*
proof *clarsimp*
let *?nr* = *dim-row A*
let *?nc* = *dim-col A*
let *?pp* = *pivot-positions A* :: (*nat* × *nat*) *list*
fix *i*
assume *i-lt*: *i* < *length* (*pivot-positions A*)
have ∃*f*. *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*: ∀ *i* < (*length ?pp*). (*i*, *f i*) ∈ *set ?pp*
using *pivot-positions-contains pf*
assms
by *blast*
have *sorted-hyp*: ⋀ (*p*::*nat*) (*q*::*nat*). *p* < (*length ?pp*) ⟹ *q* < (*length ?pp*) ⟹ *p* < *q* ⟹ (*fst* (*?pp* ! *p*) < *fst* (*?pp* ! *q*))
proof −
fix *p*::*nat*
fix *q*::*nat*
assume *p-lt*: *p* < *q*
assume *p-welldef*: *p* < (*length ?pp*)
assume *q-welldef*: *q* < (*length ?pp*)
show *fst* (*?pp* ! *p*) < *fst* (*?pp* ! *q*)
using *sorted-pivot-positions p-lt p-welldef q-welldef sorted-wrt-nth-less* **by** *fastforce*
qed
have *h*: *i* < (*length ?pp*) ⟶ *fst* (*pivot-positions A* ! *i*) = *i*
proof (*induct i*)
case *0*
have ∃*j*. *fst* (*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*: *fst* (*pivot-positions A* ! *j*) = *0*
by *blast*
have *j* ≠ *0* ⟶ (*fst* (*pivot-positions A* ! *0*) > *0* ⟶ *j* ≤ *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 < length (pivot-positions A) ⟶ fst (pivot-positions A ! i) = i
      using Suc.hyps by blast
    have thesis-h: (Suc i) < length (pivot-positions A) ⟹ fst (pivot-positions A !
(Suc i)) = (Suc i)
    proof −
      assume suc-i-lt: (Suc i) < length (pivot-positions A)
      have fst-i-is: fst (pivot-positions A ! i) = i using suc-i-lt ind-h
        using Suc-lessD by blast
      have (∃j < (length ?pp). fst (pivot-positions A ! j) = (Suc i))
        by (metis suc-i-lt all-f-in fst-conv  in-set-conv-nth)
       then obtain j where jth: j < (length ?pp) ∧ fst (pivot-positions A ! j) =
(Suc i)
        by blast
      have j > i
        using sorted-hyp apply (auto)
           by (metis Suc-lessD ‹fst (pivot-positions A ! i) = i› jth less-not-refl
linorder-neqE-nat n-not-Suc-n suc-i-lt)
      have j > (Suc i) ⟹ False
      proof −
        assume j-gt: j > (Suc i)
        then have h1: fst (pivot-positions A ! (Suc i)) > i
          using fst-i-is sorted-pivot-positions
          using sorted-hyp suc-i-lt by force
        have fst (pivot-positions A ! j) > fst (pivot-positions A ! (Suc i))
          using jth j-gt sorted-hyp apply (auto)
          by fastforce
        then have h2: fst (pivot-positions A ! (Suc i)) < (Suc i)
          using jth
          by simp
        show False using h1 h2
          using not-less-eq by blast
      qed
      show fst (pivot-positions A ! (Suc i)) = (Suc i)
        using Suc-lessI ‹Suc i < j ⟹ False› ‹i < j› jth by blast
    qed
    then show ?case
      by blast
  qed
  then show fst (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*: *dim-col A* ≥ *dim-row A*
shows *take-cols A* (*map snd* (*pivot-positions A*)) =
1$_m$ (*length* (*pivot-positions A*)) @$_r$
0$_m$ (*dim-row A* − *length* (*pivot-positions A*)) (*length* (*pivot-positions A*))
proof −
let *?nr* = *dim-row A*
let *?nc* = *dim-col A*
have *h1*: *dim-col*
(*1*$_m$ (*length* (*pivot-positions A*)) @$_r$
0$_m$ (*dim-row A* − *length* (*pivot-positions A*)) (*length* (*pivot-positions A*))) =
(*length* (*pivot-positions A*))
by (*simp add*: *append-rows-def*)
have *len-pivot*: *length* (*pivot-positions A*) = *card* {*i*. *i* < *?nr* ∧ *row A i* ≠ *0*$_v$ *?nc*}
using *row-ech pivot-positions*(*4*) *row-echelon-form-def* **by** *blast*
have *pp-leq-nc*: ∀ *f*. *pivot-fun A f ?nc* ⟶ (∀ *i* < *?nr*. *f i* ≤ *?nc*) **unfolding** *pivot-fun-def*
by *meson*
have *pivot-set*: ∃ *f*. *pivot-fun A f ?nc* ∧ *set* (*pivot-positions A*) = {(*i*, *f i*) | *i*. *i* < *?nr* ∧ *f i* ≠ *?nc*}
using *row-ech row-echelon-form-def pivot-positions*(*1*)
by (*smt* (*verit*) *Collect-cong carrier-matI*)
then have *pivot-set-alt*: ∃ *f*. *pivot-fun A f ?nc* ∧ *set* (*pivot-positions A*) = {(*i*, *f i*) | *i*. *i* < *?nr* ∧ *row A i* ≠ *0*$_v$ *?nc*}
using *pivot-positions pivot-fun-zero-row-iff Collect-cong carrier-mat-triv*
by (*smt* (*verit*, *best*))
have ∃ *f*. *pivot-fun A f ?nc* ∧ *set* (*pivot-positions A*) = {(*i*, *f i*) | *i*. *f i* ≤ *?nc* ∧ *i* < *?nr* ∧ *f i* ≠ *?nc*}
using *pivot-set pp-leq-nc* **by** *auto*
then have *pivot-set-var*: ∃ *f*. *pivot-fun A f ?nc* ∧ *set* (*pivot-positions A*) = {(*i*, *f i*) | *i*. *i* < *?nr* ∧ *f i* < *?nc*}
by *auto*
have *length* (*map snd* (*pivot-positions A*)) = *card* (*set* (*map snd* (*pivot-positions A*)))
using *row-ech row-echelon-form-def pivot-positions*(*3*) *distinct-card*[**where** *xs* = *map snd* (*pivot-positions A*)]
by (*metis carrier-mat-triv*)
then have *length* (*map snd* (*pivot-positions A*)) = *card* (*set* (*pivot-positions A*))
by (*metis card-distinct distinct-card distinct-map length-map*)
then have *length* (*map snd* (*pivot-positions A*)) = *card* {*i*. *i* < *?nr* ∧ *row A i* ≠ *0*$_v$ *?nc*}
using *pivot-set-alt*
by (*simp add*: *len-pivot*)
then have *length-asm*: *length* (*map snd* (*pivot-positions A*)) = *length* (*pivot-positions A*)
using *len-pivot* **by** *linarith*
then have ∀ *a*. *List.member* (*map snd* (*pivot-positions A*)) *a* ⟶ *a* < *dim-col A*
proof *clarsimp*
fix *a*

```
    assume a-in: List.member (map snd (pivot-positions A)) a
    have ∃ v ∈ set (pivot-positions A). a = snd v
      using a-in in-set-member[where xs = (pivot-positions A)] apply (auto)
     by (metis in-set-impl-in-set-zip2 in-set-member length-map snd-conv zip-map-fst-snd)

    then show a < dim-col A
      using pivot-set-var in-set-member by auto
  qed
  then have h2b: (filter (λy. y < dim-col A) (map snd (pivot-positions A))) =
(map snd (pivot-positions A))
    by (meson filter-True in-set-member)
  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
     (1_m (length (pivot-positions A)) @_r
      0_m (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
              (1_m (length (pivot-positions A)) @_r
               0_m (dim-row A − length (pivot-positions A)) (length (pivot-positions
A))) ⟹
          take-cols A (map snd (pivot-positions A)) $$ (i, j) =
          (1_m (length (pivot-positions A)) @_r
          0_m (dim-row A − length (pivot-positions A)) (length (pivot-positions A)))
$$
          (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
              (1_m (length (pivot-positions A)) @_r
               0_m (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)
       = (1_m (length (pivot-positions A))) $$ (i, j))
   proof −

     assume i < (length ?pp)
     have ∃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 < (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 ∈ set (pivot-positions A)
         using j-nc nth-mem by blast
       then have ∃n. pivot-positions A ! j = (n, f n) ∧ n < dim-row A ∧ f n ≠
dim-col A
         using ‹⟦A ∈ carrier-mat (dim-row A) (dim-col A); pivot-fun A f (dim-col
A)⟧ ⟹ set (pivot-positions A) = {(i, f i) |i. i < dim-row A ∧ f i ≠ dim-col A}›
‹pivot-fun A f (dim-col A)› by blast
       then show ?thesis
         by (metis (no-types) ‹⋀A. ⟦row-echelon-form A; dim-row A ≤ dim-col A⟧
⟹ ∀i<length (pivot-positions A). fst (pivot-positions A ! i) = i› 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 ⟶ ((cols A) ! ?w) $ i = 1 using w-is pivot-funD(4)[of A
?nr f ?nc i]
        by (metis ‹∀a. List.member (map snd (pivot-positions A)) a ⟶ a <
dim-col A› ‹i < length (pivot-positions A)› ‹pivot-fun A f (dim-col A)› cols-length
i-lt in-set-member length-asm mat-of-cols-cols mat-of-cols-index nth-mem)
     have h1:  i ≠ j ⟶ ((cols A) ! ?w) $ i = 0 using w-is pivot-funD(5)
        by (metis ‹∀a. List.member (map snd (pivot-positions A)) a ⟶ a <
dim-col A› ‹pivot-fun A f (dim-col A)› cols-length h1 i-lt in-set-member j-lt len-lt
length-asm less-le-trans mat-of-cols-cols mat-of-cols-index nth-mem)
     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 ‹i < length (pivot-positions A)› h2 h-len index-one-mat(1) j-lt
len-lt)
```

qed
have *h1b*: $i \geq (length\ ?pp) \Longrightarrow (mat\text{-}of\text{-}cols\ (dim\text{-}row\ A)\ (map\ ((!)\ (cols\ A))\ (map\ snd\ (pivot\text{-}positions\ A)))\ \$\$\ (i,\ j)\ =\ 0)$
proof −
assume *i-gt*: $i \geq (length\ ?pp)$
have *h0a*: $((cols\ A)\ !\ ((map\ snd\ (pivot\text{-}positions\ A))\ !\ j))\ \$\ i = (row\ A\ i)\ \$\ ?w$
by (*metis ‹∀ a. List.member (map snd (pivot-positions A)) a ⟶ a < dim-col A› cols-length h1 i-lt in-set-member index-row(1) j-lt length-asm mat-of-cols-cols mat-of-cols-index nth-mem*)
have *h0b*:
$take\text{-}rows\ A\ [0..<length\ (pivot\text{-}positions\ A)]\ @_r\ 0_m\ (dim\text{-}row\ A - length\ (pivot\text{-}positions\ A))\ (dim\text{-}col\ A) = A$
using *assms row-echelon-form-zero-rows*[*of A*]
by *blast*
then have *h0c*: $(row\ A\ i) = 0_v\ (dim\text{-}col\ A)$ **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** *h0a breaking-it-down* **apply** (*auto*)
by (*metis ‹∀ a. List.member (map snd (pivot-positions A)) a ⟶ a < dim-col A› h1 in-set-member index-zero-vec(1) j-lt length-asm nth-mem*)
qed
have *h1*: $mat\text{-}of\text{-}cols\ (dim\text{-}row\ A)$
$(map\ ((!)\ (cols\ A))\ (map\ snd\ (pivot\text{-}positions\ A)))\ \$\$\ (i,\ j) =$
$(1_m\ (length\ (pivot\text{-}positions\ A))\ @_r$
$0_m\ (dim\text{-}row\ A - length\ (pivot\text{-}positions\ A))\ (length\ (pivot\text{-}positions\ A)))$
$\$\$$
$(i,\ j)$ **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 $take\text{-}cols\ A\ (map\ snd\ (pivot\text{-}positions\ A))\ \$\$\ (i,\ j) =$
$(1_m\ (length\ (pivot\text{-}positions\ A))\ @_r$
$0_m\ (dim\text{-}row\ A - length\ (pivot\text{-}positions\ A))\ (length\ (pivot\text{-}positions\ A)))$
$\$\$$
$(i,\ j)$
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 $dim\text{-}col\ A \geq dim\text{-}row\ A$

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

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

lemma *lin-indpt-id*:
shows *lin-indpt (set (cols (1*$_m$ *n)::′a vec list))*
proof −
have ∗: *set (cols (1*$_m$ *n)) = set (rows (1*$_m$ *n))*
by (*metis cols-transpose transpose-one*)
have *det (1*$_m$ *n) ≠ 0* **using** *det-one* **by** *auto*
from *det-not-0-imp-lin-indpt-rows*[*OF - this*]
have *lin-indpt (set (rows (1*$_m$ *n)))*
using *one-carrier-mat* **by** *blast*
thus *?thesis*
by (*simp add*: ∗)
qed

lemma *lin-indpt-take-cols-id*:
shows *lin-indpt (set (cols (take-cols (1*$_m$ *n) inds)))*
proof −
have *subset-h*: *set (cols (take-cols (1*$_m$ *n) inds)) ⊆ set (cols (1*$_m$ *n)::′a vec list)*
using *cols-take-cols-subset* **by** *blast*
then show *?thesis* **using** *lin-indpt-id subset-li-is-li* **by** *auto*
qed

lemma *cols-id-unit-vecs*:
shows *cols (1*$_m$ *d) = unit-vecs d*
unfolding *unit-vecs-def list-eq-iff-nth-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 $\bigwedge j.\ j \in set\ inds \Longrightarrow j < length\ ls$
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 rank B = rank (C* A * B) using * by auto
  also have ... ≤ 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 mat
  assumes invertible-mat A
  assumes A ∈ carrier-mat n n
  assumes distinct inds
  shows rank (take-cols A inds) = length (filter ((>) n) inds)
proof −
  have  A = A * 1_m n using assms(2) by auto
  then have take-cols A inds = A * take-cols (1_m n) inds
    by (metis assms(2) one-carrier-mat take-cols-mat-mul)
  then have rank (take-cols A inds) = rank (take-cols (1_m n) 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 ∈ carrier-mat n nc
  shows rank (take-cols R ls) ≤ rank R
proof −
  from take-cols-mat-mul[OF R]
  have take-cols R ls =  R * take-cols (1_m nc) 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 ∈ carrier-mat n nc
  assumes t:take-cols R ls ∈ carrier-mat n r
  assumes B:B ∈ carrier-mat r nc
  assumes R = (take-cols R ls) * B
  shows rank (take-cols R ls) ≥ 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 rank (take-cols R (map snd (pivot-positions R))) = rank R
proof −
  let ?B = take-rows R [0..<length (pivot-positions R)]
```

have *equa*: R = *take-cols* R (*map snd* (*pivot-positions* R)) $*$ *?B* **using** *assms rref-right-mul*
by (*metis carrier-matD*(*1*) *carrier-matD*(*2*))
have *ex-r*: $\exists r$. *take-cols* R (*map snd* (*pivot-positions* R)) $\in$ *carrier-mat* n r $\wedge$ *?B* $\in$ *carrier-mat* r *nc*
proof −
have *h1*:
take-cols R (*map snd* (*pivot-positions* R)) $\in$ *carrier-mat* n (*length* (*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. *pivot-fun* R f *nc* **using** *row-ech* **unfolding** *row-echelon-form-def* **using** *dims*
by *blast*
then have *length* (*pivot-positions* R) = *card* $\{i.\ i < n \wedge row\ R\ i \neq 0_v\ nc\}$
using *pivot-positions*[*of* R n *nc*]
using *dims* **by** *auto*
then have *nc* $\geq$ *length* (*pivot-positions* R) **using** *order*
using *carrier-matD*(*1*) *dims dual-order.trans length-pivot-positions-dim-row row-ech* **by** *blast*
then have *dim-col* $R \geq$ *length* (*pivot-positions* R) **using** *dims* **by** *auto*
then have *h2*: *?B* $\in$ *carrier-mat* (*length* (*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*
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 *rank* R $\leq$ *rank* (*take-cols* R (*map snd* (*pivot-positions* R)))
using *dims ex-r rank-take-cols-geq*[**where** $R = R$, **where** B = *?B*, **where** *ls* = (*map snd* (*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:$A \in$ *carrier-mat* n *nc*
assumes *order*: *nc* $\geq n$
shows (*take-cols* A (*map snd* (*pivot-positions* (*gauss-jordan-single* A)))) =
(*take-cols-var* A (*map snd* (*pivot-positions* (*gauss-jordan-single* A))))
proof −
let *?gjs* = (*gauss-jordan-single* A)

have $\forall x.$ *List.member* (*map snd* (*pivot-positions* (*gauss-jordan-single A*))) $x \longrightarrow$
$x \leq$ *dim-col A*
using *rref-pivot-positions gauss-jordan-single*(*3*) *carrier-matD*(*2*) *gauss-jordan-single*(*2*)
in-set-impl-in-set-zip2 in-set-member length-map less-irrefl less-trans not-le-imp-less
zip-map-fst-snd
by (*metis* (*no-types*, *lifting*) *carrier-mat-triv*)
then have (*filter* ($\lambda y.$ $y <$ *dim-col A*) (*map snd* (*pivot-positions* (*gauss-jordan-single*
A)))) =
(*map snd* (*pivot-positions* (*gauss-jordan-single A*)))
by (*metis* (*no-types*, *lifting*) *A 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* **unfolding** *take-cols-def take-cols-var-def*
by *simp*
qed

lemma *gauss-jordan-single-rank*:
fixes *A*::*'a mat*
assumes *A*:*A* ∈ *carrier-mat n nc*
assumes *order*: *nc* ≥ *n*
shows *rank* (*take-cols A* (*map snd* (*pivot-positions* (*gauss-jordan-single A*)))) =
rank A
proof −
let *?R* = *gauss-jordan-single A*
obtain *P* **where** *P*:*P*∈*Units* (*ring-mat TYPE*(*'a*) *n undefined*) **and**
i: *?R* = *P* ∗ *A* **using** *gauss-jordan-transform*[*OF A*]
by (*metis A carrier-matD*(*1*) *fst-eqD gauss-jordan-single-def surj-pair zero-carrier-mat*)
have *pcarrier*: *P* ∈ *carrier-mat n n* **using** *P* **unfolding** *Units-def*
by (*auto simp add*: *ring-mat-def*)
have *invertible-mat P* **using** *P* **unfolding** *invertible-mat-def Units-def inverts-mat-def*
apply *auto*
apply (*simp add*: *ring-mat-simps*(*5*))
by (*metis index-mult-mat*(*2*) *index-one-mat*(*2*) *ring-mat-simps*(*1*) *ring-mat-simps*(*3*))
then
obtain *Pi* **where** *Pi*: *invertible-mat Pi Pi* ∗ *P* = 1_m *n*
proof −
assume *a1*: $\bigwedge$*Pi*. ⟦*invertible-mat Pi*; *Pi* ∗ *P* = 1_m *n*⟧ ⟹ *thesis*
have *dim-row P* = *n*
by (*metis* (*no-types*) *A assms*(*1*) *carrier-matD*(*1*) *gauss-jordan-single*(*2*) *i*
index-mult-mat(*2*))
then show *?thesis*
using *a1* **by** (*metis* (*no-types*) ‹*invertible-mat P*› *index-mult-mat*(*3*) *in-*
dex-one-mat(*3*) *invertible-mat-def inverts-mat-def square-mat.simps*)
qed
then have *pi-carrier*:*Pi* ∈ *carrier-mat n n*
by (*metis carrier-mat-triv index-mult-mat*(*2*) *index-one-mat*(*2*) *invertible-mat-def*
square-mat.simps)
have *R1*:*row-echelon-form ?R*
using *assms*(*2*) *gauss-jordan-single*(*3*) **by** *blast*
have *R2*: *?R* ∈ *carrier-mat n nc*

```
    using A assms(2) gauss-jordan-single(2) by auto
  have Rcm: take-cols ?R (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 A^T))
    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* ⊆ *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*) (λ*T*. *T* ⊆ *set* (*B*) ∧ *lin-indpt T*) **using** *h1*
by (*simp add*: *maximal-def subset-antisym*)
then have *h2*: *maximal* (*set B*) (λ*T*. *T* ⊆ *set* (*cols* (*mat-of-cols n B*)) ∧ *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* ⇒ *nat list* ⇒ *'a list*
where *retrieve-polys qss index-list* = (*map* (*nth qss*) *index-list*)

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

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

5 Base Case

definition *base-case-info*:: (*rat mat* × (*nat list list* × *rat list list*))
where *base-case-info* =
((*mat-of-rows-list 2* [[*1*,*1*], [*1*,−*1*]]), ([[],[*0*]], [[*1*],[−*1*]]))

definition *base-case-solve-for-lhs*:: *real poly* ⇒ *real poly* ⇒ *rat vec*
where *base-case-solve-for-lhs p q* = (*mult-mat-vec* (*mat-of-rows-list 2* [[*1*/*2*,

1/2], [*1/2*, −*1/2*]]) (*construct-rhs-vector p* [*q*] [[], [*0*]]))

thm *gauss-jordan-compute-inverse*

primrec *matr-option*:: *nat* ⇒ *'a*::{*one*, *zero*} *mat option* ⇒ *'a mat*
where *matr-option dimen None* = 1_m *dimen*
| *matr-option dimen* (*Some c*) = *c*

definition *mat-equal*:: *'a*:: *field mat* ⇒ *'a* :: *field mat* ⇒ *bool*
where *mat-equal A B* = (*dim-row A* = *dim-row B* ∧ *dim-col A* = *dim-col B* ∧
(*mat-to-list A*) = (*mat-to-list B*))

definition *mat-inverse-var* :: *'a* :: *field mat* ⇒ *'a mat option* **where**
mat-inverse-var A = (*if dim-row A* = *dim-col A then*
let one = 1_m (*dim-row A*) *in*
(*case gauss-jordan A one of*
(*B*, *C*) ⇒ *if* (*mat-equal B one*) *then Some C else None*) *else None*)

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

6 Smashing

definition *subsets-smash*::*nat* ⇒ *nat list list* ⇒ *nat list list* ⇒ *nat list list*
where *subsets-smash n s1 s2* = *concat* (*map* (λ*l1*. *map* (λ *l2*. *l1* @ (*map* ((+) *n*) *l2*)) *s2*) *s1*)

definition *signs-smash*::*'a list list* ⇒ *'a list list* ⇒ *'a list list*
where *signs-smash s1 s2* = *concat* (*map* (λ*l1*. *map* (λ *l2*. *l1* @ *l2*) *s2*) *s1*)

definition *smash-systems*:: *real poly* ⇒ *real poly list* ⇒ *real poly list* ⇒ *nat list list* ⇒ *nat list list* ⇒
rat list list ⇒ *rat list list* ⇒ *rat mat* ⇒ *rat mat* ⇒
real poly list × (*rat mat* × (*nat list list* × *rat list list*))
where *smash-systems p qs1 qs2 subsets1 subsets2 signs1 signs2 mat1 mat2* =
(*qs1*@*qs2*, (*kronecker-product mat1 mat2*, (*subsets-smash* (*length qs1*) *subsets1 subsets2*, *signs-smash signs1 signs2*)))

fun *combine-systems*:: *real poly* ⇒ (*real poly list* × (*rat mat* × (*nat list list* × *rat list list*))) ⇒ (*real poly list* × (*rat mat* × (*nat list list* × *rat list list*))) ⇒ (*real poly list* × (*rat mat* × (*nat list list* × *rat list list*)))
where *combine-systems p* (*qs1*, *m1*, *sub1*, *sgn1*) (*qs2*, *m2*, *sub2*, *sgn2*) =
(*smash-systems p qs1 qs2 sub1 sub2 sgn1 sgn2 m1 m2*)

7 Reduction

definition *find-nonzeros-from-input-vec*:: *rat vec* ⇒ *nat list*
where *find-nonzeros-from-input-vec lhs-vec* = *filter* (λ*i*. *lhs-vec* $ *i* ≠ *0*) [*0*..< *dim-vec lhs-vec*]

definition *take-indices*:: *'a list* ⇒ *nat list* ⇒ *'a list*
where *take-indices subsets indices* = *map* ((!) *subsets*) *indices*

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

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

fun *reduce-mat-cols*:: *'a mat* ⇒ *rat vec* ⇒ *'a mat*
where *reduce-mat-cols A lhs-vec* = *take-cols-from-matrix A* (*find-nonzeros-from-input-vec lhs-vec*)

definition *rows-to-keep*:: (*'a*::*field*) *mat* ⇒ *nat list* **where**
rows-to-keep A = *map snd* (*pivot-positions* (*gauss-jordan-single* (A^T)))

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

fun *reduce-system*:: *real poly* ⇒ (*real poly list* × (*rat mat* × (*nat list list* × *rat list list*))) ⇒ (*rat mat* × (*nat list list* × *rat list list*))
where *reduce-system p* (*qs*,*m*,*subs*,*signs*) =
reduction-step m signs subs (*solve-for-lhs p qs subs m*)

8 Overall algorithm

fun *calculate-data*:: *real poly* ⇒ *real poly list* ⇒ (*rat mat* × (*nat list list* × *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*}

+ *card* {*x*. *poly p x* = 0 ∧ *poly q x* < 0} **using** *3* **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*:: *real*
assumes *finite A*
shows *k*∗*card A* = *sum* (λ*x*. *k*) *A*
proof −
have *plus* ∘ (λ-. *Suc 0*) = (λ-. *Suc*)
by (*simp add*: *fun-eq-iff*)
then have *Finite-Set.fold* (*plus* ∘ (λ-. *Suc 0*)) = *Finite-Set.fold* (λ-. *Suc*)
by (*rule arg-cong*)
then have *Finite-Set.fold* (*plus* ∘ (λ-. *Suc 0*)) *0 A* = *Finite-Set.fold* (λ-. *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*::*real poly*
assumes *coprime p q*
assumes *p* ≠ *0*
assumes *q* ≠ *0*
shows *changes-R-smods p* ((*pderiv p*) ∗ *q*) = *card* {*x*. *poly p x* = *0* ∧ *poly q x* > *0*} − *int*(*card* {*x*. *poly p x* = *0* ∧ *poly q x* < *0*})
proof −
have *3*: *taq* {*x*. *poly p x*=*0*} *q* ≡ ∑ *y*∈{*x*. *poly p x*=*0*}. *Sturm-Tarski.sign* (*poly q y*) **by** (*simp add*: *taq-def*)
have *4*: {*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 *5*: {*x*. *poly p x*=*0*} = {*x*. *poly p x* = *0* ∧ *poly q x* > *0*} ∪ {*x*. *poly p x* = *0* ∧ *poly q x* < *0*} **using** *assms*(*1*) *coprime-poly-0* **by** *auto*
then have *6*: ∑ *y*∈{*x*. *poly p x*=*0*}. *Sturm-Tarski.sign* (*poly q y*) ≡ ∑ *y*∈{*x*. *poly p x* = *0* ∧ *poly q x* > *0*} ∪ {*x*. *poly p x* = *0* ∧ *poly q x* < *0*}. *Sturm-Tarski.sign* (*poly q y*) **by** *presburger*
then have *12*: *taq* {*x*. *poly p x*=*0*} *q* ≡ ∑ *y*∈{*x*. *poly p x* = *0* ∧ *poly q x* > *0*} ∪ {*x*. *poly p x* = *0* ∧ *poly q x* < *0*}. *Sturm-Tarski.sign* (*poly q y*)
by (*simp add*: *3*)
have *7*: {*x*. *poly p x* = *0* ∧ *poly q x* > *0*} ∩ {*x*. *poly p x* = *0* ∧ *poly q x* < *0*} = {} **by** *auto*
then have *8*: ∑ *y*∈{*x*. *poly p x* = *0* ∧ *poly q x* > *0*} ∪ {*x*. *poly p x* = *0* ∧ *poly q x* < *0*}. *Sturm-Tarski.sign* (*poly q y*) ≡ (∑ *y*∈{*x*. *poly p x* = *0* ∧ *poly q x* > *0*}.*Sturm-Tarski.sign* (*poly q y*)) + (∑ *y*∈{*x*. *poly p x* = *0* ∧ *poly q x* < *0*}.*Sturm-Tarski.sign*(*poly q y*)) **by** (*simp add*: *assms*(*2*) *poly-roots-finite sum.union-disjoint*)

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

lemma *restate-tarski2*:
fixes *q*::*real poly*
assumes $p \neq 0$
shows *changes-R-smods p* ((*pderiv p*) $*$ *q*) =
int(*card* $\{x.\ poly\ p\ x = 0 \wedge poly\ q\ x > 0\}$) −
int(*card* $\{x.\ poly\ p\ x = 0 \wedge poly\ q\ x < 0\}$)
unfolding *sturm-tarski-R*[*symmetric*] *taq-def*
proof −
let *?all* = $\{x.\ poly\ p\ x{=}0\}$
let *?lt* = $\{x.\ poly\ p\ x{=}0 \wedge poly\ q\ x < 0\}$
let *?gt* = $\{x.\ poly\ p\ x{=}0 \wedge poly\ q\ x > 0\}$
let *?eq* = $\{x.\ poly\ p\ x{=}0 \wedge poly\ q\ x = 0\}$
have *eq*: *?all* = *?lt* ∪ *?gt* ∪ *?eq* **by** *force*
from *poly-roots-finite*[*OF assms*] **have** *fin*: *finite ?all* **.**
show $(\sum x \mid poly\ p\ x = 0.\ Sturm\text{-}Tarski.sign\ (poly\ q\ x)) = int\ (card\ ?gt) - int\ (card\ ?lt)$
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*:: *real poly set*
shows *finite I* $\Longrightarrow ((\forall\ q \in I.\ (coprime\ p\ q)) \longrightarrow (coprime\ p\ (\prod\ I)))$
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* $\Longrightarrow$ ((∀ *q* ∈ *I*. *q* ≠ *0*) ⟶ $\prod$ *I* ≠ *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* ∈ *insert x F*
by *simp*
have *hq*: (∀ *q* ∈ (*insert x F*). *q* ≠ *0*) ⟶ *x* ≠ *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* ⇒ *real list*
where *characterize-root-list-p p* ≡ *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*≠*0*
shows *construct-NofI p I* =
rat-of-int (*int* (*card* {*x*. *poly p x* = *0* ∧ *poly* (*prod-list I*) *x* > *0*}) −
int (*card* {*x*. *poly p x* = *0* ∧ *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* ⇒ *real poly list list* ⇒ *rat vec*
where *construct-s-vector p Is* = *vec-of-list* (*map* (λ *I*.(*construct-NofI p I*)) *Is*)

definition *squash*::*'a*::*linordered-field* ⇒ *rat*
where *squash x* = (*if x* > *0 then 1*
else if x < *0 then* −*1*
else 0)

definition *signs-at*::*real poly list* ⇒ *real* ⇒ *rat list*
where *signs-at qs x* ≡
map (*squash* ∘ (λ*q*. *poly q x*)) *qs*

definition *characterize-consistent-signs-at-roots*:: *real poly* ⇒ *real poly list* ⇒ *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* ⇒ *real* ⇒ *rat list*
where *consistent-sign-vec-copr qs x* ≡
map (λ *q*. *if* (*poly q x* > *0*) *then* (*1*::*rat*) *else* (−*1*::*rat*)) *qs*

definition *characterize-consistent-signs-at-roots-copr*:: *real poly* ⇒ *real poly list* ⇒ *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*≠*0*
assumes *pairwise-rel-prime*: ∀ *q*. ((*List.member qs q*) ⟶ (*coprime p q*))
shows *characterize-consistent-signs-at-roots p qs* = *characterize-consistent-signs-at-roots-copr p qs*
proof −
have ∀ *q* ∈ *set*(*qs*). ∀ *x* ∈ *set* (*characterize-root-list-p p*). *poly q x* ≠ *0*
using *pairwise-rel-prime*
using *coprime-poly-0 in-set-member nonzero poly-roots-finite characterize-root-list-p-def*
by *fastforce*
then have *h*: ∀ *q* ∈ *set*(*qs*). ∀ *x* ∈ *set* (*characterize-root-list-p p*). *squash* (*poly q x*) = (*if* (*poly q x* > *0*) *then* (*1*::*rat*) *else* (−*1*::*rat*))
by (*simp add*: *squash-def*)
have *map* (λ*r*. *map* (λ*p*. *if 0* < *poly p r then 1 else* − *1*) *qs*) (*characterize-root-list-p p*) = *map* (λ*r*. *map* (*squash* ∘ (λ*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-d signs-at-def consistent-sign-vec-copr-def*
by *presburger*
qed

definition *list-constr*:: *nat list* ⇒ *nat* ⇒ *bool*
where *list-constr L n* ≡ *list-all* (λ*x*. *x* < *n*) *L*

definition *all-list-constr*:: *nat list list* ⇒ *nat* ⇒ *bool*
where *all-list-constr L n* ≡ (∀ *x*. *List.member L x* ⟶ *list-constr x n*)

definition *z*:: *nat list* ⇒ *rat list* ⇒ *rat*
where *z index-list sign-asg* ≡ (*prod-list* (*map* (*nth sign-asg*) *index-list*))

definition *mtx-row*:: *rat list list* ⇒ *nat list* ⇒ *rat list*
where *mtx-row sign-list index-list* ≡ (*map* ((*z index-list*)) *sign-list*)

definition *matrix-A*:: *rat list list* ⇒ *nat list list* ⇒ *rat mat*
where *matrix-A sign-list subset-list* =
(*mat-of-rows-list* (*length sign-list*) (*map* (λ*i* .(*mtx-row sign-list i*)) *subset-list*))

definition *alt-matrix-A*:: *rat list list* ⇒ *nat list list* ⇒ *rat mat*
where *alt-matrix-A signs subsets* = (*mat* (*length subsets*) (*length signs*)
(λ(*i*, *j*). *z* (*subsets* ! *i*) (*signs* ! *j*)))

lemma *alt-matrix-char*: *alt-matrix-A signs subsets* = *matrix-A signs subsets*
proof −
have *h0*: (⋀*i j*. *i* < *length subsets* ⟹
j < *length signs* ⟹
map (λ*index-list*. *map* (*z index-list*) *signs*) *subsets* ! *i* ! *j* = *z* (*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 index-list*) *signs*) *subsets*) ! *i*) ! *j* = *z* (*subsets* ! *i*) (*signs* ! *j*)
proof −
have *h0*: (*map* (λ*index-list*. *map* (*z index-list*) *signs*) *subsets*) ! *i* = *map* (*z* (*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* (*subsets* ! *i*) (*signs* ! *j*)) =
mat (*length subsets*) (*length signs*) (λ(*i*, *y*). *map* (λ*index-list*. *map* (*z index-list*) *signs*) *subsets* ! *i* ! *y*)
using *h0 eq-matI*[**where** *A* = *mat* (*length subsets*) (*length signs*) (λ(*i*, *j*). *z* (*subsets* ! *i*) (*signs* ! *j*)),
where *B* = *mat* (*length subsets*) (*length signs*) (λ(*i*, *y*). *map* (λ*index-list*.

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

lemma *subsets-are-rows*: ∀ *i* < (*length subsets*). *row* (*alt-matrix-A signs subsets*) *i* = *vec* (*length signs*) (λ*j*. *z* (*subsets* ! *i*) (*signs* ! *j*))
unfolding *row-def* **unfolding** *alt-matrix-A-def* **by** *auto*

lemma *signs-are-cols*: ∀ *i* < (*length signs*). *col* (*alt-matrix-A signs subsets*) *i* = *vec* (*length subsets*) (λ*j*. *z* (*subsets* ! *j*) (*signs* ! *i*))
unfolding *col-def* **unfolding** *alt-matrix-A-def*
by *auto*

definition *construct-lhs-vector*:: *real poly* ⇒ *real poly list* ⇒ *rat list list* ⇒ *rat vec*
where *construct-lhs-vector p qs signs* ≡
vec-of-list (*map* (λ*w*. *rat-of-int* (*int* (*length* (*filter* (λ*v*. *v* = *w*) (*map* (*consistent-sign-vec-copr qs*) (*characterize-root-list-p p*)))))) *signs*)

definition *satisfy-equation*:: *real poly* ⇒ *real poly list* ⇒ *nat list list* ⇒ *rat list list* ⇒ *bool*
where *satisfy-equation p qs subset-list sign-list* =
(*mult-mat-vec* (*matrix-A sign-list subset-list*) (*construct-lhs-vector p qs sign-list*) = (*construct-rhs-vector p qs subset-list*))

11 Setting up the construction: Proofs

lemma *row-mat-of-rows-list*:
assumes *list-all* (λ*r*. *length r* = *nc*) *rs*
assumes *i* < *length rs*
shows *row* (*mat-of-rows-list nc rs*) *i* = *vec-of-list* (*nth 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 *length ls* = *nc*
assumes *list-all* (λ*r*. *length r* = *nc*) *rs*
shows *mat-of-rows-list nc rs* ∗$_v$ *vec-of-list ls* =
vec-of-list (*map* (λ*r*. *vec-of-list r* ∙ *vec-of-list ls*) *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 (λ*r*. *length r* = *length 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* ≠ *0*
assumes *i* < *length signs*
shows (*construct-lhs-vector p qs signs*) $ *i* =
card {*x*. *poly p x* = *0* ∧ ((*consistent-sign-vec-copr qs x*) = (*nth signs i*))}
proof −
from *poly-roots-finite*[*OF assms*(*1*)] **have** *finite* {*x*. *poly p x* = *0*} **.**
then have *eq*: (*Collect*
((λ*v*. *v* = *signs* ! *i*) ∘
consistent-sign-vec-copr qs) ∩
set (*sorted-list-of-set*
{*x*. *poly p x* = *0*})) =
{*x*. *poly p x* = *0* ∧ *consistent-sign-vec-copr qs x* = *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* ≠ *0*
shows (*construct-lhs-vector p qs signs*) =
vec-of-list (*map* (λ*s*. *rat-of-int* (*card* {*x*. *poly p x* = *0* ∧ ((*consistent-sign-vec-copr qs x*) = *s*)})) *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* (λ*i*. *i* < *length signs*) *I*
assumes *list-all* (λ*s*. *s* = *1* ∨ *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* (λ*i*. *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* (λ*s*. *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 characterize-root-list-p-def* **using** *sign-fix*
  by *blast*
qed

lemma *nonzero-product*:
 fixes *p*:: *real poly*
 assumes *nonzero*: $p \neq 0$
 fixes *qs*:: *real poly list*
 assumes *pairwise-rel-prime-1*: $\forall q.\ ((List.member\ qs\ q) \longrightarrow (coprime\ p\ q))$
 fixes *I*:: *nat list*
 fixes *x*:: *real*
 assumes *root-p*: $x \in \{x.\ poly\ p\ x = 0\}$
 assumes *welldefined*: *list-constr I* (*length qs*)
 shows (*poly* (*prod-list* (*retrieve-polys qs I*)) $x > 0$) $\vee$ (*poly* (*prod-list* (*retrieve-polys qs I*)) $x < 0$)
proof −
 have $\bigwedge x.\ x \in set\ (retrieve\text{-}polys\ qs\ I) \Longrightarrow coprime\ p\ x$
  unfolding *retrieve-polys-def*
  by (*smt in-set-conv-nth in-set-member length-map list-all-length list-constr-def nth-map pairwise-rel-prime-1 welldefined*)
 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*: $p \neq 0$
 fixes *qs*:: *real poly list*
 assumes *pairwise-rel-prime-1*: $\forall q.\ ((List.member\ qs\ q) \longrightarrow (coprime\ p\ q))$
 fixes *I*:: *nat list*
 fixes *sign*:: *rat list*
 fixes *x*:: *real*
 assumes *root-p*: $x \in \{x.\ poly\ p\ x = 0\}$
 assumes *sign-fix*: *sign* = *consistent-sign-vec-copr qs x*
 shows (*list-constr I* (*length qs*)) $\longrightarrow$ (*poly* (*prod-list* (*retrieve-polys qs I*)) $x > 0$) $\longleftrightarrow$ (*z I sign* = 1)
proof (*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 in-set-member 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≠0
 fixes qs:: real poly list
 assumes pairwise-rel-prime-1: ∀ q. ((List.member qs q) ⟶ (coprime p q))
 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 (poly (prod-list (retrieve-polys qs I)) x > 0) ⟷ (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≠0
 fixes qs:: real poly list
 assumes pairwise-rel-prime-1: ∀ q. ((List.member qs q) ⟶ (coprime p q))
 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 (poly (prod-list (retrieve-polys qs I)) x < 0) ⟷ (z I sign = −1)
proof −
 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 poly-hyp: (poly (prod-list (retrieve-polys qs I)) x > 0) ∨ (poly (prod-list
(retrieve-polys qs I)) x < 0)
   using nonzero-product
   using pairwise-rel-prime-1 nonzero root-p
   using welldefined by blast
 have pos-hyp: (poly (prod-list (retrieve-polys qs I)) x > 0) ⟷ (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* (λ*s*. (*z I s*) ∗ *rat-of-int* (*card* {*x*. *poly p x* = *0* ∧ *consistent-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* (λ*s*. *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* (λ*s*. *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* ⇒ *int*
assumes *distinct xs distinct ys*
assumes *set ys* ⊆ *set xs*
assumes ⋀*x*. *x* ∈ *set xs* − *set ys* ⟹ *f x* = *0*
shows *sum-list* (*map f xs*) = *sum-list* (*map f ys*)
by (*metis List.finite-set assms*(*1*) *assms*(*2*) *assms*(*3*) *assms*(*4*) *sum.mono-neutral-cong-left sum-list-distinct-conv-sum-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*≠*0*
assumes *distinct-signs*: *distinct signs*
assumes *all-info*: *set* (*characterize-consistent-signs-at-roots-copr p qs*) ⊆ *set*(*signs*)
assumes *welldefined*: *list-constr I* (*length qs*)
shows
(*vec-of-list* (*mtx-row signs I*) ⋅ (*construct-lhs-vector p qs signs*)) =
(*vec-of-list* (*mtx-row* (*characterize-consistent-signs-at-roots-copr p qs*) *I*) ⋅
(*construct-lhs-vector p qs* (*characterize-consistent-signs-at-roots-copr p qs*)))
proof −
have *h0*: ∀ *sgn*. *sgn* ∈ *set signs* ∧ *sgn* ∉ *consistent-sign-vec-copr qs* ‘ *set* (*characterize-root-list-p p*) ∧ *0* < *rat-of-nat* (*card*
{*xa*. *poly p xa* = *0* ∧ *consistent-sign-vec-copr qs xa* = *sgn*}) ⟶ *z I sgn* = *0*
proof −
have ∀ *sgn*. *sgn* ∈ *set signs* ∧ *sgn* ∉ *consistent-sign-vec-copr qs* ‘ *set* (*characterize-root-list-p p*) ∧ *0* < *rat-of-int* (*card*
{*xa*. *poly p xa* = *0* ∧ *consistent-sign-vec-copr qs xa* = *sgn*}) ⟶ {*xa*. *poly p xa* = *0* ∧ *consistent-sign-vec-copr qs xa* = *sgn*} ≠ {}
by *fastforce*
then show *?thesis*
using *characterize-consistent-signs-at-roots-copr-def in-set nonzero welldefined*
by *auto*
qed
then have ∀ *sgn*. *sgn* ∈ *set signs* ∧ *sgn* ∉ *consistent-sign-vec-copr qs* ‘ *set* (*characterize-root-list-p p*) ⟶ ((*0* = *rat-of-nat* (*card*
{*xa*. *poly p xa* = *0* ∧ *consistent-sign-vec-copr qs xa* = *sgn*}) ∨ *z I sgn* = *0*))
by *auto*
then have *hyp*: ∀ *s*. *s* ∈ *set signs* ∧ *s* ∉ *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*}) = *0*)
by *auto*
then have (∑ *s*∈ *set*(*signs*). *z I s* ∗ *rat-of-nat* (*card* {*x*. *poly p x* = *0* ∧ *consistent-sign-vec-copr qs x* = *s*})) =

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

proof −

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

by *blast*

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

by *auto*

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

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

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

using *hyp*

by (*simp add*: *hyp*)

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

qed

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

using *all-info*

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

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

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

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

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* ∧ *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*}))
using *set-sum-eq hyp1 hyp2*
using ‹(∑ *s*∈*set signs*. *z I s* ∗ *rat-of-nat* (*card* {*x*. *poly p x* = *0* ∧ *consistent-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* ∧ *consistent-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 characterize-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*: $p \neq 0$
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*)) ∩
{*s*. *z I s* = *1*})
let *?lt* = (*set* (*map* (*consistent-sign-vec-copr qs*) (*characterize-root-list-p p*)) ∩
{*s*. *z I s* = −*1*})
have *eq*: *set* (*map* (*consistent-sign-vec-copr qs*) (*characterize-root-list-p p*)) =
?gt ∪ *?lt*
apply *auto*
by (*metis characterize-root-list-p-def horiz-vector-helper-neg horiz-vector-helper-pos-ind nonzero nonzero-product pairwise-rel-prime-1 poly-roots-finite sorted-list-of-set*(*1*) *welldefined*)

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 ... =
($\sum$ *s*←*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 ... =
($\sum$ *s*∈*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 ... =
($\sum$ *s*∈*?gt*. *z I s* ∗ *rat-of-int* (*card* {*x*. *poly p x* = *0* $\wedge$ *consistent-sign-vec-copr qs x* = *s*})) +
($\sum$ *s*∈*?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 $*$: $\bigwedge s.\ \{x.\ poly\ p\ x = 0 \wedge consistent\text{-}sign\text{-}vec\text{-}copr\ qs\ x = s\} =$
$\{x \in \{x.\ poly\ p\ x = 0\}.\ consistent\text{-}sign\text{-}vec\text{-}copr\ qs\ x = s\}$
by *auto*
have *lem-e1*: $\bigwedge x.\ x \in \{x.\ poly\ p\ x = 0\} \Longrightarrow$
card
$\{s \in consistent\text{-}sign\text{-}vec\text{-}copr\ \ qs\ `\ \{x.\ poly\ p\ x = 0\} \cap \{s.\ z\ I\ s = 1\}.$
$consistent\text{-}sign\text{-}vec\text{-}copr\ qs\ x = s\} =$
$(if\ 0 < poly\ (prod\text{-}list\ (retrieve\text{-}polys\ qs\ I))\ x\ then\ 1\ else\ 0)$
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-pos*[*OF assms*(*1*) *assms*(*5*) *rt*]
have *2*: $... = \{s.\ (0 < poly\ (prod\text{-}list\ (retrieve\text{-}polys\ qs\ I))\ x) \wedge consistent\text{-}sign\text{-}vec\text{-}copr\ qs\ x = s\}$
using *welldefined* **by** *blast*
have *3*: $... = (if\ (0 < poly\ (prod\text{-}list\ (retrieve\text{-}polys\ qs\ I))\ x)\ then\ \{consistent\text{-}sign\text{-}vec\text{-}copr\ qs\ x\}\ else\ \{\})$
by *auto*
thus $card\ \{s \in consistent\text{-}sign\text{-}vec\text{-}copr\ qs\ `\ \{x.\ poly\ p\ x = 0\} \cap \{s.\ z\ I\ s = 1\}.\ consistent\text{-}sign\text{-}vec\text{-}copr\ qs\ x = s\} =$
$(if\ 0 < poly\ (prod\text{-}list\ (retrieve\text{-}polys\ qs\ I))\ x\ then\ 1\ else\ 0)$ **using** *1 2 3* **by** *auto*
qed
have *e1*: $(\sum s \in consistent\text{-}sign\text{-}vec\text{-}copr\ qs\ `\ \{x.\ poly\ p\ x = 0\} \cap \{s.\ z\ I\ s = 1\}.$
$card\ \{x.\ poly\ p\ x = 0 \wedge consistent\text{-}sign\text{-}vec\text{-}copr\ qs\ x = s\}) =$
$(sum\ (\lambda x.\ if\ (poly\ (prod\text{-}list\ (retrieve\text{-}polys\ qs\ I))\ x) > 0\ then\ 1\ else\ 0)\ \{x.\ poly\ p\ x = 0\})$
unfolding $*$ **apply** (*rule sum-multicount-gen*)
using ‹*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*: $(\sum s \in ?gt.\ z\ I\ s * rat\text{-}of\text{-}int\ (card\ \{x.\ poly\ p\ x = 0 \wedge consistent\text{-}sign\text{-}vec\text{-}copr\ qs\ x = s\})) =$
$rat\text{-}of\text{-}int\ (card\ \{x.\ poly\ p\ x = 0 \wedge 0 < poly\ (prod\text{-}list\ (retrieve\text{-}polys\ qs\ I))\ x\})$
apply (*auto simp add*: *setroots*)
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*: $\bigwedge x.\ x \in \{x.\ poly\ p\ x = 0\} \Longrightarrow$
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 poly (prod-list (retrieve-polys qs I)) x < 0 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-neg[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 poly (prod-list (retrieve-polys qs I)) x < 0 then 1 else 0) using 1 2 3 by
auto
  qed
  have e2:  (∑ s∈consistent-sign-vec-copr qs ‘ {x. poly p x = 0} ∩ {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-e2 by auto
  have ltchr: (∑ s∈?lt. z I s ∗ rat-of-int (card {x. poly p x = 0 ∧ consistent-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 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 ‹(∑ s∈set (map (consistent-sign-vec-copr qs) (characterize-root-list-p p)).
z I s ∗ rat-of-int (card {x. poly p x = 0 ∧ consistent-sign-vec-copr qs x = s})) =
(∑ s∈set (map (consistent-sign-vec-copr qs) (characterize-root-list-p p)) ∩ {s. z I
s = 1}. z I s ∗ rat-of-int (card {x. poly p x = 0 ∧ consistent-sign-vec-copr qs
```

$x = s\}))$ + $(\sum s{\in}set\ (map\ (consistent\text{-}sign\text{-}vec\text{-}copr\ qs)\ (characterize\text{-}root\text{-}list\text{-}p\ p)) \cap \{s.\ z\ I\ s = -1\}.\ z\ I\ s * rat\text{-}of\text{-}int\ (card\ \{x.\ poly\ p\ x = 0 \wedge consistent\text{-}sign\text{-}vec\text{-}copr\ qs\ x = s\}))$› ‹$(\sum s{\leftarrow}characterize\text{-}consistent\text{-}signs\text{-}at\text{-}roots\text{-}copr\ p\ qs.\ z\ I\ s * rat\text{-}of\text{-}int\ (card\ \{x.\ poly\ p\ x = 0 \wedge consistent\text{-}sign\text{-}vec\text{-}copr\ qs\ x = s\}))$ = $(\sum s{\in}set\ (map\ (consistent\text{-}sign\text{-}vec\text{-}copr\ qs)\ (characterize\text{-}root\text{-}list\text{-}p\ p)).\ z\ I\ s * rat\text{-}of\text{-}int\ (card\ \{x.\ poly\ p\ x = 0 \wedge consistent\text{-}sign\text{-}vec\text{-}copr\ qs\ x = s\}))$› ‹$vec\text{-}of\text{-}list\ (mtx\text{-}row\ (characterize\text{-}consistent\text{-}signs\text{-}at\text{-}roots\text{-}copr\ p\ qs)\ I) \cdot construct\text{-}lhs\text{-}vector\ p\ qs\ (characterize\text{-}consistent\text{-}signs\text{-}at\text{-}roots\text{-}copr\ p\ qs) = (\sum s{\leftarrow}characterize\text{-}consistent\text{-}signs\text{-}at\text{-}roots\text{-}copr\ p\ qs.\ z\ I\ s * rat\text{-}of\text{-}int\ (card\ \{x.\ poly\ p\ x = 0 \wedge consistent\text{-}sign\text{-}vec\text{-}copr\ qs\ x = s\}))$› ‹$vec\text{-}of\text{-}list\ (mtx\text{-}row\ signs\ I) \cdot construct\text{-}lhs\text{-}vector\ p\ qs\ signs = vec\text{-}of\text{-}list\ (mtx\text{-}row\ (characterize\text{-}consistent\text{-}signs\text{-}at\text{-}roots\text{-}copr\ p\ qs)\ I) \cdot construct\text{-}lhs\text{-}vector\ p\ qs\ (characterize\text{-}consistent\text{-}signs\text{-}at\text{-}roots\text{-}copr\ p\ qs)$›
  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*: $set\ (characterize\text{-}consistent\text{-}signs\text{-}at\text{-}roots\text{-}copr\ p\ qs) \subseteq set(signs)$
  assumes *welldefined*: *list-constr I* (*length qs*)
  assumes *pairwise-rel-prime-1*: $\forall\, q.\ ((List.member\ qs\ q) \longrightarrow (coprime\ p\ q))$
  shows $(vec\text{-}of\text{-}list\ (mtx\text{-}row\ signs\ I) \cdot (construct\text{-}lhs\text{-}vector\ p\ qs\ signs)) =$
    construct-NofI p (*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 *map f xs* = *map g ys*
  shows *map-vec f* (*vec-of-list xs*) = *map-vec g* (*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*: $set\ (characterize\text{-}consistent\text{-}signs\text{-}at\text{-}roots\text{-}copr\ p\ qs) \subseteq set(signs)$
  assumes *pairwise-rel-prime*: $\forall\, q.\ ((List.member\ qs\ q) \longrightarrow (coprime\ p\ q))$
  assumes *welldefined*: *all-list-constr* (*subsets*) (*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 in-set-member welldefined* **by** *fastforce*

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

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

definition *sgn-vec*::*real poly list* ⇒ *real* ⇒ *rat list*
where *sgn-vec qs x* ≡ *map* (*sgn* ∘ (λ*q*. *poly q x*)) *qs*

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

lemma *consistent-signs-at-roots-eq*:
assumes *p* ≠ *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* ⇒ *real poly list* ⇒ *rat list list* ⇒ *rat vec*
where *w-vec* ≡ *construct-lhs-vector*

abbreviation *v-vec*:: *real poly* ⇒ *real poly list* ⇒ *nat list list* ⇒ *rat vec*
where *v-vec* ≡ *construct-rhs-vector*

abbreviation *M-mat*:: *rat list list* ⇒ *nat list list* ⇒ *rat mat*
where *M-mat* ≡ *matrix-A*

theorem *matrix-equation-pretty*:
assumes *p*≠*0*
assumes ⋀*q*. *q* ∈ *set qs* ⟹ *coprime p q*
assumes *distinct signs*

assumes *consistent-signs-at-roots p qs* $\subseteq$ *set signs*
assumes $\bigwedge l\ i.\ l \in set\ subsets \Longrightarrow i \in set\ l \Longrightarrow i < length\ qs$
shows *M-mat signs subsets* $*_v$ *w-vec p qs signs = v-vec p qs subsets*
unfolding *satisfy-equation-def*[*symmetric*]
apply (*rule matrix-equation*[*OF assms*(*1*) *assms*(*3*)])
apply (*metis assms*(*1*) *assms*(*2*) *assms*(*4*) *consistent-signs-at-roots-eq csa-list-copr-rel member-def*)
apply (*simp add*: *assms*(*2*) *in-set-member*)
using *Ball-set all-list-constr-def assms*(*5*) *list-constr-def member-def* **by** *fastforce*

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

begin

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

definition *satisfies-properties*:: *real poly* $\Rightarrow$ *real poly list* $\Rightarrow$ *nat list list* $\Rightarrow$ *rat list list* $\Rightarrow$ *rat mat* $\Rightarrow$ *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*]] = (*mat-of-rows-list 2* [[*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*: $\forall q.\ (List.member\ qs\ q) \longrightarrow coprime\ p\ q$
shows *set* (*characterize-consistent-signs-at-roots-copr p qs*) $\subseteq \{[1], [-1]\}$
proof −
have *ex-q*: $\exists (q::real\ poly).\ qs = [q]$
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*: $\forall qa.$ *List.member* [*q*] *qa* $\longrightarrow$ *coprime p qa* **using** *rel-prime*
by (*simp add*: *member-rec*(*1*) *member-rec*(*2*))
have *h3*: *all-list-constr* [[], [*0*]] (*Suc 0*) **unfolding** *all-list-constr-def*
by (*simp add*: *list-constr-def member-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*: $\forall q.\ (List.member\ qs\ q) \longrightarrow coprime\ p\ q$
shows *satisfy-equation p qs* [[],[*0*]] [[*1*],[−*1*]]
proof −
have *ex-q*: $\exists (q::real\ poly).\ qs = [q]$
using *len1*
using *length-Suc-conv*[*of qs 0*] **by** *auto*
then show *?thesis* **using** *base-case-satisfy-equation nonzero rel-prime*
apply (*auto*)
by (*simp add*: *member-rec*(*1*))
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 $j < Suc\ (Suc\ 0) \longleftrightarrow j = 0 \vee j = 1$ **by** *auto*

lemma *inverse-mat-base-case*:
shows *inverts-mat* (*mat-of-rows-list 2* [[*1/2*, *1/2*], [*1/2*, −(*1/2*)]]::*rat mat*) (*mat-of-rows-list 2* [[*1*, *1*], [*1*, −*1*]]::*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*]]::*rat mat*) (*mat-of-rows-list 2* [[*1/2*, *1/2*], [*1/2*, −(*1/2*)]]::*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 ∀ (*elem* :: *'a list*). (*elem* ∈ (*set* (*signs-smash signs1 signs2*)) ⟶
(∃ *n m* :: *nat*.
elem = ((*nth signs1 n*)@(*nth signs2 m*))))
proof *clarsimp*
fix *elem*
assume *assum*: *elem* ∈ *set* (*signs-smash signs1 signs2*)
show ∃ *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*)) $\longrightarrow$
(∃ (*elem1*::*'a list*). ∃ (*elem2*::*'a list*).
(*elem1* $\in$ *set*(*signs1*) $\wedge$ *elem2* $\in$ *set*(*signs2*) $\wedge$ *elem* = (*elem1*@*elem2*))))
proof *clarsimp*
fix *elem*
assume *assum*: *elem* $\in$ *set* (*signs-smash signs1 signs2*)
show ∃ *elem1*. *elem1* $\in$ *set signs1* $\wedge$ (∃ *elem2*. *elem2* $\in$ *set signs2* $\wedge$ *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 $\forall$ (*elem* :: *nat list*). (*List.member* (*subsets-smash n subsets1 subsets2*) *elem*) $\longrightarrow$
(∃ (*elem1*::*nat list*). ∃ (*elem2*::*nat list*).
(*elem1* $\in$ *set*(*subsets1*) $\wedge$ *elem2* $\in$ *set*(*subsets2*) $\wedge$ *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*
by (*smt* (*verit*) *imageE in-set-member set-map signs-smash-property-set*)
qed

13.2 Well-defined subsets preserved when smashing

lemma *list-constr-append*:
list-constr a n $\wedge$ *list-constr b n* $\longrightarrow$ *list-constr* (*a*@*b*) *n*
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*: $\forall$ *elem*.
List.member (*subsets-smash* (*length qs1*) *subsets1 subsets2*) *elem* $\longrightarrow$
(∃ *elem1 elem2*. *elem1* $\in$ *set subsets1* $\wedge$ *elem2* $\in$ *set subsets2* $\wedge$ *elem* = *elem1* @ *map* ((+) (*length qs1*)) *elem2*)
using *subsets-smash-property* **by** *blast*
have *h2*: $\forall$ *elem1 elem2*. (*elem1* $\in$ *set subsets1* $\wedge$ *elem2* $\in$ *set subsets2*) $\longrightarrow$

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 add*: *in-set-member*)
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 add*: *in-set-member*)
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*) ⟷ *x=a ∧ 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* ⊆ *set a2*
assumes *sub2*: *set b1* ⊆ *set b2*
shows *set* (*signs-smash a1 b1*) ⊆ *set* (*signs-smash a2 b2*)
proof −
have *h1*: ∀ *x*∈*set* (*signs-smash a1 b1*). *x*∈*set* (*signs-smash a2 b2*)
proof *clarsimp*
fix *x*
assume *x-in*: *x* ∈ *set* (*signs-smash a1 b1*)
have *h1*: ∃ *n m* :: *nat*. *x* = (*nth a1 n*)@(*nth b1 m*)
using *signs-smash-property*[*of a1 b1*] *x-in*
by *blast*
then have *h2*: ∃ *p q*::*nat*. *x* = (*nth a2 p*)@(*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* ∈ *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 ∀ *x* ∈ *set* (*characterize-consistent-signs-at-roots-copr p* (*qs1* @ *qs2*)).
∃ *x1* ∈ *set* (*characterize-consistent-signs-at-roots-copr p qs1*).
∃ *x2* ∈ *set* (*characterize-consistent-signs-at-roots-copr p qs2*).
x = *x1*@*x2*
proof *clarsimp*
fix *x*
assume *x-in*: *x* ∈ *set* (*characterize-consistent-signs-at-roots-copr p* (*qs1* @ *qs2*))

have *x-in-csv*: *x* ∈ *set*(*map* (*consistent-sign-vec-copr* (*qs1* @ *qs2*)) (*characterize-root-list-p p*))
using *x-in* **unfolding** *characterize-consistent-signs-at-roots-copr-def* **by** *simp*
have *csv-hyp*: ∀ *r*. *consistent-sign-vec-copr* (*qs1* @ *qs2*) *r* = (*consistent-sign-vec-copr qs1 r*)@(*consistent-sign-vec-copr qs2 r*)
unfolding *consistent-sign-vec-copr-def* **by** *auto*
have *exr-iff*: (∃ *r* ∈ *set* (*characterize-root-list-p p*). *x* = *consistent-sign-vec-copr* (*qs1* @ *qs2*) *r*) ⟷ (∃ *r* ∈ *set* (*characterize-root-list-p p*). *x* = (*consistent-sign-vec-copr qs1 r*)@(*consistent-sign-vec-copr qs2 r*))
using *csv-hyp* **by** *auto*
have *exr*: *x* ∈ *set*(*map* (*consistent-sign-vec-copr* (*qs1* @ *qs2*)) (*characterize-root-list-p p*)) ⟶ (∃ *r* ∈ *set* (*characterize-root-list-p p*). *x* = *consistent-sign-vec-copr* (*qs1* @ *qs2*) *r*)
by *auto*
have *mem-list1*: ∀ *r* ∈ *set* (*characterize-root-list-p p*). (*consistent-sign-vec-copr qs1 r*) ∈ *set*(*map* (*consistent-sign-vec-copr qs1*) (*characterize-root-list-p p*))
by *simp*
have *mem-list2*: ∀ *r* ∈ *set* (*characterize-root-list-p p*). (*consistent-sign-vec-copr qs2 r*) ∈ *set*(*map* (*consistent-sign-vec-copr qs2*) (*characterize-root-list-p p*))
by *simp*
then show ∃ *x1*∈*set* (*characterize-consistent-signs-at-roots-copr p qs1*).
∃ *x2*∈*set* (*characterize-consistent-signs-at-roots-copr p qs2*). *x* = *x1* @ *x2*
using *x-in-csv exr mem-list1 mem-list2 characterize-consistent-signs-at-roots-copr-def exr-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*: *set* (*characterize-consistent-signs-at-roots-copr p qs1*) ⊆ *set* (*signs1*)
assumes *csa2*: *set* (*characterize-consistent-signs-at-roots-copr p qs2*) ⊆ *set* (*signs2*)

shows *set* (*characterize-consistent-signs-at-roots-copr p*
(*qs1* @ *qs2*))

$\subseteq$ *set* (*signs-smash signs1 signs2*)
proof −
have *h0*: ∀ *x* ∈ *set* (*characterize-consistent-signs-at-roots-copr p* (*qs1* @ *qs2*)). ∃ *x1* ∈ *set* (*characterize-consistent-signs-at-roots-copr p qs1*). ∃ *x2* ∈ *set* (*characterize-consistent-signs-at-roots-c p qs2*).
(*x* = *x1*@*x2*) **using** *subset-helper*[*of p qs1 qs2*]
by *blast*
then have ∀ *x* ∈ *set* (*characterize-consistent-signs-at-roots-copr p* (*qs1* @ *qs2*)).
x ∈ *set* (*signs-smash* (*characterize-consistent-signs-at-roots-copr p qs1*) (*characterize-consistent-signs-at-roots-copr p qs2*))
unfolding *signs-smash-def* **apply** (*auto*)
by *fastforce*
then have ∀ *x* ∈ *set* (*characterize-consistent-signs-at-roots-copr p* (*qs1* @ *qs2*)). *x* ∈ *set* (*signs-smash signs1 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 *dim-row* (*mat-of-rows-list nr ls*) = *length ls*
unfolding *mat-of-rows-list-def* **by** *auto*

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

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

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

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

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

```
lemma length-eq-concat:
  assumes ⋀x. x ∈ set ls ⟹ length x = n
  assumes i < n * length ls
  shows concat ls ! i = ls ! (i div n) ! (i mod n)
  using assms
proof (induct ls arbitrary: i)
  case Nil
  then show ?case
    by simp
next
  from assms have ‹n > 0›
    by (cases ‹n = 0›) simp-all
  case (Cons a ls)
  show ?case
  proof (cases ‹n ≤ i›)
    case False
    with Cons show ?thesis
      by (simp add: nth-append)
  next
    case True
    moreover define j where ‹j = i - n›
    ultimately have ‹i = n + j›
      by simp
    with Cons ‹n > 0› show ?thesis
      by (simp add: nth-append div-add-self1)
  qed
qed

lemma z-append:
  assumes ⋀i. i ∈ set xs ⟹ i < length as
  shows z (xs @ (map ((+) (length as)) ys)) (as @ bs) = z xs as * z ys bs
proof -
  have 1: map ((!) (as @ bs)) xs = map ((!) as) xs
    unfolding list-eq-iff-nth-eq
    using assms by (auto simp add:nth-append)
  have 2: map ((!) (as @ bs) ∘ (+) (length as)) ys = 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 $\bigwedge l\ i.\ l \in set\ subs1 \Longrightarrow i \in set\ l \Longrightarrow i < n1$
assumes $\bigwedge j.\ j \in set\ signs1 \Longrightarrow length\ j = n1$
shows
(*matrix-A* (*signs-smash signs1 signs2*) (*subsets-smash n1 subs1 subs2*)) =
kronecker-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-kronecker
proof *safe*
fix *i j*
assume *i*: $i < length\ subs1 * length\ subs2$
and *j*: $j < length\ signs1 * length\ signs2$
have *ld*: $i\ div\ length\ subs2 < length\ subs1$
$j\ div\ length\ signs2 < length\ signs1$
using *i j less-mult-imp-div-less* **by** *auto*
have *lm*: $i\ mod\ length\ subs2 < length\ subs2$
$j\ mod\ length\ signs2 < length\ signs2$
using *i j less-mult-imp-mod-less* **by** *auto*

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

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

(*signs2* ! (*j mod length signs2*))
unfolding *kronecker-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*) $$ (*i*, *j*) =
kronecker-product (*matrix-A signs1 subs1*) (*matrix-A signs2 subs2*) $$ (*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*: ∀ *q*. ((*List.member qs1 q*) ⟶ (*coprime p q*))
assumes *pairwise-rel-prime2*: ∀ *q*. ((*List.member qs2 q*) ⟶ (*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 subsets2*) (*signs-smash signs1 signs2*)
∧ *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-kronecker-product*
kronecker-invertible invertibleMtx1 invertibleMtx2
welldefined-subsets1 welldefined-subsets2 **unfolding** *all-list-constr-def list-constr-def*
by (*smt* (*verit*) *Ball-set member-def well-def-signs-def welldefined-signs1*)
have *h2a*: *distinct* (*signs-smash signs1 signs2*)
using *distinct-signs1 distinct-signs2 welldefined-signs1 welldefined-signs2 nontriv1 nontriv2*
distinct-step **by** *auto*
have *h2ba*: ∀ *q*. *List.member* (*qs1* @ *qs2*) *q* ⟶ (*List.member qs1 q* ∨ *List.member*

qs2 q)
unfolding *member-def*
by *auto*
have *h2b*: ∀ *q*. ((*List.member* (*qs1*@*qs2*) *q*) ⟶ (*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*)) ⊆ *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 subsets2*) (*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* × (*nat list list* × *rat list list*)) ⇒ *rat mat*
where *get-matrix data* = *fst*(*data*)

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

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

definition *reduction-signs*:: *real poly* ⇒ *real poly list* ⇒ *rat list list* ⇒ *nat list list* ⇒ *rat mat* ⇒ *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* ⇒ *real poly list* ⇒ *rat list list* ⇒ *nat list list* ⇒ *rat mat* ⇒ *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 ∀ *n* < (*dim-vec input-vec*). ((*input-vec* $ *n* ≠ *0*) ⟷
List.member (*find-nonzeros-from-input-vec input-vec*) *n*)
proof −
have *h1*: ∀ *n* < (*dim-vec input-vec*). ((*input-vec* $ *n* ≠ *0*) ⟶ *List.member* (*find-nonzeros-from-input-vec input-vec*) *n*)
unfolding *find-nonzeros-from-input-vec-def List.member-def* **by** *auto*
have *h2*: ∀ *n* < (*dim-vec input-vec*). (*List.member* (*find-nonzeros-from-input-vec input-vec*) *n* ⟶ (*input-vec* $ *n* ≠ *0*))
unfolding *find-nonzeros-from-input-vec-def List.member-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* < *length* (*take-indices arb-list index-list*)
assumes *well-def*: ∀ *q*.(*List.member index-list q* ⟶ *q* < *length arb-list*)
shows ∃ *k*<*length arb-list*.
(*take-indices arb-list index-list*) ! *n* = *arb-list* ! *k*
using *lest well-def* **unfolding** *take-indices-def* **apply** (*auto*)
by (*metis in-set-member nth-mem*)

lemma *invertible-means-mult-id*:
fixes *A*:: *'a*::*field mat*
assumes *assm*: *invertible-mat A*
shows *matr-option* (*dim-row A*)
(*mat-inverse* (*A*))∗*A* = 1_m (*dim-row A*)
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*
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 *assm*: *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 *dim-vec* (*construct-lhs-vector p qs signs*) = *length signs*
unfolding *construct-lhs-vector-def*
by *simp*

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

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

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

lemma *mat-inverse-same*: *mat-inverse-var A* = *mat-inverse A*
unfolding *mat-inverse-var-def mat-inverse-def mat-equal-def*
by (*smt* (*verit*) *mat-equal-list-lem split-cong*)

lemma *construct-lhs-matches-solve-for-lhs*:
fixes *p*:: *real poly*

fixes *qs* :: *real poly list*
fixes *subsets* :: *nat list list*
fixes *signs* :: *rat list list*
assumes *match*: *satisfy-equation p qs subsets signs*
assumes *invertible-mat*: *invertible-mat* (*matrix-A signs subsets*)
shows (*construct-lhs-vector p qs signs*) = *solve-for-lhs p qs subsets* (*matrix-A signs subsets*)
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* ⊆ *set*(*B*)
assumes *not-in-hyp*: ∀ *n* < *dim-vec C*. *C* $ *n* = *0* ⟶ (*nth B n*) ∉ *A*
shows *A* ⊆ *set* (*take-indices B*
(*find-nonzeros-from-input-vec C*))
proof −
have *unfold*: ⋀ *x*. (*x* ∈ *A* ⟹ *x* ∈ *set* (*take-indices B*
(*find-nonzeros-from-input-vec C*)))
proof −
fix *x*
assume *in-a*: *x* ∈ *A*
have *x* ∈ *set* (*B*)
using *sub in-a* **by** *blast*
then have ∃ *n* < *length B*. *nth B n* = *x*
by (*simp add*: *in-set-conv-nth*)
then have ∃ *n* < *length B*. (*nth B n* = *x* ∧ (*List.member* (*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* ∈ *set* (*take-indices B*
(*find-nonzeros-from-input-vec C*))
unfolding *take-indices-def*
using *member-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 ⊆ set(B)
  assumes not-in-hyp: ∀ n < dim-vec C. C $ n ≠ 0 ⟶ (nth B n) ∈ A
  shows set (take-indices B
          (find-nonzeros-from-input-vec C)) ⊆ A
proof −
  have unfold: ⋀ x. (x ∈ (set (take-indices B
          (find-nonzeros-from-input-vec C))) ⟹ x ∈ A)
  proof −
    fix x
    assume in-set: x ∈ set (take-indices B (find-nonzeros-from-input-vec C))
    have h: ∃ n< dim-vec C. (C $ n ≠ 0 ∧ (nth B n) = x)
    proof −
      have h1: ∃ n < length B.(nth B n) = x
        using in-set unfolding take-indices-def
          find-nonzeros-from-input-vec-def eq-len by auto
    have h2: ∀ n< length B. (nth B n = x ⟶ List.member (find-nonzeros-from-input-vec
C) n)
      proof clarsimp
        fix n
        assume leq-hyp: n < length B
        assume x-eq: x = B ! n
        have h: (B !n) ∈ set (map ((!) B) (find-nonzeros-from-input-vec C))
          using x-eq in-set
          by (simp add: take-indices-def)
       then have h2: List.member (map ((!) B) (find-nonzeros-from-input-vec C))
(B ! n)
          using in-set
          by (meson in-set-member)
        then have h3: ∃ k< length B. (List.member (find-nonzeros-from-input-vec
C) k ∧ ((B ! k) = (B ! n)))
         by (smt (verit) atLeastLessThan-iff eq-len find-nonzeros-from-input-vec-def
imageE in-set-member mem-Collect-eq set-filter set-map set-upt)
        have h4: ∀ v< length B. ((B ! v) = (B ! n) ⟶ v = n)
          using dist by (metis find-first-unique leq-hyp)
        then show List.member (find-nonzeros-from-input-vec C) n
          using h2 h3 h4 by auto
      qed
      then have ∀ n<length B. (nth B n = x ⟶ C $ n ≠ 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 ∈ 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* (λ*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*) ‹*construct-lhs-vector p qs signs = solve-for-lhs p qs subsets* (*matrix-A signs subsets*)› *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*

$= \textit{consistent-sign-vec-copr qs } x\} = 0$
assume *zero-asm*: $\textit{poly } p\ x = 0$
have *card-hyp*: $\{xa.\ \textit{poly } p\ xa = 0 \wedge \textit{consistent-sign-vec-copr qs } xa = \textit{consistent-sign-vec-copr qs } x\} = \{\}$
using *card-eq-0-iff*
using *card-asm nonzero poly-roots-finite* **by** *fastforce*
have $x \in \{xa.\ \textit{poly } p\ xa = 0 \wedge \textit{consistent-sign-vec-copr qs } xa = \textit{consistent-sign-vec-copr qs } x\}$
using *zero-asm* **by** *auto*
thus *False* **using** *card-hyp*
by *blast*
qed
have *h2*: $\bigwedge w.\ (\textit{rat-of-nat } (\textit{card } \{x.\ \textit{poly } p\ x = 0 \wedge \textit{consistent-sign-vec-copr qs } x = w\}) = 0 \Longrightarrow$
$(\neg \textit{List.member } (\textit{characterize-consistent-signs-at-roots-copr } p\ qs)\ w))$
by (*smt* (*verit*, *best*) *characterize-consistent-signs-at-roots-copr-def characterize-root-list-p-def h1 imageE in-set-member mem-Collect-eq nonzero poly-roots-finite set-map set-remdups sorted-list-of-set*(*1*))
then have *h3*: $\forall\ w.\ \textit{rat-of-nat } (\textit{card } \{x.\ \textit{poly } p\ x = 0 \wedge \textit{consistent-sign-vec-copr qs } x = w\}) = 0 \longrightarrow$
$w \notin \textit{set } (\textit{characterize-consistent-signs-at-roots-copr } p\ qs)$
by (*simp add*: *in-set-member*)
show *?thesis* **using** *h0 h3*
by *blast*
qed
then have $\textit{set } (\textit{characterize-consistent-signs-at-roots-copr } p\ qs) \subseteq \textit{set } (\textit{take-indices signs}$
$(\textit{find-nonzeros-from-input-vec } (\textit{solve-for-lhs } p\ qs\ \textit{subsets } (\textit{matrix-A signs subsets}))))$
using *all-info*
reduction-signs-set-helper-lemma[**where** $A = \textit{set } (\textit{characterize-consistent-signs-at-roots-copr } p\ qs)$, **where** $B = \textit{signs}$,
where $C = (\textit{solve-for-lhs } p\ qs\ \textit{subsets } (\textit{matrix-A signs subsets}))$]
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*: $p \neq 0$
assumes *welldefined-signs1*: *well-def-signs* (*length qs*) *signs*
assumes *distinct-signs*: *distinct signs*
assumes *all-info*: $\textit{set } (\textit{characterize-consistent-signs-at-roots-copr } p\ qs) \subseteq \textit{set}(\textit{signs})$
assumes *match*: *satisfy-equation p qs subsets signs*
assumes *invertible-mat*: *invertible-mat* (*matrix-A signs subsets*)
shows $\textit{set } (\textit{characterize-consistent-signs-at-roots-copr } p\ qs) = \textit{set}(\textit{reduction-signs}$

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*)) ∈ *carrier-mat* (*length signs*) (*length signs*)
using *invertible-mat dim-invertible*
using *same-size* **by** *fastforce*
have *supset*: *set* (*characterize-consistent-signs-at-roots-copr p qs*) ⊇ *set*(*reduction-signs p qs signs subsets* (*matrix-A signs subsets*))
proof −
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* (λ*s*. *card* {*x*. *poly p x* = *0* ∧ *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 ∀ *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*) ⟶
(*nth signs n*) ∈ *set* (*characterize-consistent-signs-at-roots-copr p qs*))
proof −
have *h0*: ∀ *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*) ⟶
rat-of-nat (*card* {*x*. *poly p x* = *0* ∧ *consistent-sign-vec-copr qs x* = (*nth signs n*)}) = *0*)
by (*metis* (*mono-tags*, *lifting*) ‹*construct-lhs-vector p qs signs* = *solve-for-lhs p qs subsets* (*matrix-A signs subsets*)› *construct-lhs-vector-clean nonzero of-nat-0-eq-iff of-rat-of-nat-eq size-of-lhs*)
have *h1*: ∀ *w*. (*rat-of-nat* (*card* {*x*. *poly p x* = *0* ∧ *consistent-sign-vec-copr qs x* = *w*}) ≠ *0* ⟶
(∃ *x*. *poly p x* = *0* ∧ *consistent-sign-vec-copr qs x* = *w*))
proof *clarsimp*
fix *w*
assume *card-asm*: *0* < *card* {*x*. *poly p x* = *0* ∧ *consistent-sign-vec-copr qs x* = *w*}
show ∃ *x*. *poly p x* = *0* ∧ *consistent-sign-vec-copr qs x* = *w*
by (*metis* (*mono-tags*, *lifting*) *Collect-empty-eq card-asm card-eq-0-iff gr-implies-not0*)
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*))
proof *clarsimp*
fix *w*
assume *card-asm*: *0* < *card* {*x*. *poly p x* = *0* ∧ *consistent-sign-vec-copr qs x* = *w*}

```
      have h0: ∃x. poly p x = 0 ∧ consistent-sign-vec-copr qs x = w
        using card-asm
        by (simp add: h1)
      then show List.member (characterize-consistent-signs-at-roots-copr p qs) w

        unfolding characterize-consistent-signs-at-roots-copr-def
         using in-set-member nonzero poly-roots-finite characterize-root-list-p-def
by fastforce
    qed
   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 add: in-set-member)
     show ?thesis using h0 h3
         by (metis (no-types, lifting) ‹solve-for-lhs p qs subsets (matrix-A signs
subsets) = vec-of-list (map rat-of-nat (map (λs. card {x. poly p x = 0 ∧ consis-
tent-sign-vec-copr qs x = s}) signs))› dim-vec-of-list length-map nth-map vec-of-list-index)
   qed
   then have set (take-indices signs
                (find-nonzeros-from-input-vec (solve-for-lhs p qs subsets (matrix-A
signs subsets)))) ⊆
            set (characterize-consistent-signs-at-roots-copr p qs)
     using all-info
     reduction-signs-set-helper-lemma2[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 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 subset: set (characterize-consistent-signs-at-roots-copr p qs) ⊆ set(reduction-signs
p qs signs subsets (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:: 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 (get-signs (reduce-system p (qs, ((matrix-A signs subsets), (subsets,
```

signs))))) = *set* (*characterize-consistent-signs-at-roots-copr p 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 ⋀*ell*. *ell* ∈ *set* (*rows-to-keep A*) ⟹ *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* ≠ *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*) ⊆ *set*(*signs*)
assumes *match*: *satisfy-equation p qs subsets signs*
assumes *invertible-mat*: *invertible-mat* (*matrix-A signs subsets*)
assumes *pairwise-rel-prime*: ∀ *q*. ((*List.member qs q*) ⟶ (*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* ≠ *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*: ∀ *i* < *length* (*take-indices signs ?sym*). ∀ *j* < *length*(*take-indices signs ?sym*).
i ≠ *j* ⟶ *nth* (*take-indices signs ?sym*) *i* ≠ *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* < *length*
(*find-nonzeros-from-input-vec* (*solve-for-lhs p qs subsets* (*matrix-A signs subsets*)))
assume *neq-hyp*: *i* ≠ *j*
assume *signs* ! (*find-nonzeros-from-input-vec* (*solve-for-lhs p qs subsets* (*matrix-A signs subsets*)) ! *i*) = *signs* ! (*find-nonzeros-from-input-vec* (*solve-for-lhs p qs subsets* (*matrix-A signs subsets*)) ! *j*)
have *h1*: *find-nonzeros-from-input-vec* (*solve-for-lhs p qs subsets* (*matrix-A signs subsets*)) ! *i* ≠ *find-nonzeros-from-input-vec* (*solve-for-lhs p qs subsets* (*matrix-A signs subsets*)) ! *j*
unfolding *find-nonzeros-from-input-vec-def* **using** *neq-hyp*
by (*metis* ‹*i* < *length* (*find-nonzeros-from-input-vec* (*solve-for-lhs p qs subsets* (*matrix-A signs subsets*)))› ‹*j* < *length* (*find-nonzeros-from-input-vec* (*solve-for-lhs p qs subsets* (*matrix-A signs subsets*)))› *distinct-conv-nth distinct-filter distinct-upt find-nonzeros-from-input-vec-def*)
then show *False* **using** *distinct-signs*
proof −
have *f1*: ∀ *p ns n*. ((*n*::*nat*) ∈ {*n* ∈ *set ns*. *p n*}) = (*n* ∈ *set ns* ∧ *n* ∈ *Collect p*)
by *simp*
then have *f2*: *filter* (λ*n*. *solve-for-lhs p qs subsets* (*matrix-A signs subsets*) $ *n* ≠ *0*) [*0*..<*dim-vec* (*solve-for-lhs p qs subsets* (*matrix-A signs subsets*))] ! *i* ∈ *set* [*0*..<*length signs*]
by (*metis* (*full-types*) ‹*i* < *length* (*find-nonzeros-from-input-vec* (*solve-for-lhs p qs subsets* (*matrix-A signs subsets*)))› *construct-lhs-matches-solve-for-lhs find-nonzeros-from-input-vec-def invertible-mat match nth-mem set-filter size-of-lhs*)
have *filter* (λ*n*. *solve-for-lhs p qs subsets* (*matrix-A signs subsets*) $ *n* ≠ *0*) [*0*..<*dim-vec* (*solve-for-lhs p qs subsets* (*matrix-A signs subsets*))] ! *j* ∈ *set* [*0*..<*length signs*]
using *f1* **by** (*metis* (*full-types*) ‹*j* < *length* (*find-nonzeros-from-input-vec* (*solve-for-lhs p qs subsets* (*matrix-A signs subsets*)))› *construct-lhs-matches-solve-for-lhs find-nonzeros-from-input-vec-def invertible-mat match nth-mem set-filter size-of-lhs*)
then show *?thesis*
using *f2* **by** (*metis* ‹*signs* ! (*find-nonzeros-from-input-vec* (*solve-for-lhs p qs subsets* (*matrix-A signs subsets*)) ! *i*) = *signs* ! (*find-nonzeros-from-input-vec* (*solve-for-lhs p qs subsets* (*matrix-A signs subsets*)) ! *j*)› *atLeastLessThan-iff distinct-conv-nth distinct-signs find-nonzeros-from-input-vec-def h1 set-upt*)
qed
qed
then have *distinct* (*take-indices signs* (*find-nonzeros-from-input-vec* (*solve-for-lhs p qs subsets* (*matrix-A signs subsets*))))
using *distinct-conv-nth* **by** *blast*

then show *?thesis*
using *get-signs-def reduction-signs-def reduction-signs-is-get-signs* **by** *auto*
qed
have *all-info-hyp*: *set* (*characterize-consistent-signs-at-roots-copr p qs*) ⊆ *set*(*snd* (*snd* (*reduce-system p* (*qs*, ((*matrix-A signs subsets*), (*subsets*, *signs*))))))
using *reduce-system-sign-conditions assms* **unfolding** *get-signs-def* **by** *auto*
have *pairwise-rel-prime-hyp*: ∀ *q*. ((*List.member qs q*) ⟶ (*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.member-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*: ∀ *x*. *List.member S x* ⟶ *x* < *length signs*
assumes *nonzero*: *p* ≠ *0*
shows *alt-matrix-A* (*take-indices signs S*) *subsets* = *take-cols-from-matrix* (*alt-matrix-A signs subsets*) *S*
proof −
have *h0a*: *dim-col* (*take-cols-from-matrix* (*alt-matrix-A signs subsets*) *S*) = *length* (*take-indices signs S*)
unfolding *take-cols-from-matrix-def* **apply** (*auto*) **unfolding** *take-indices-def* **by** *auto*
have *h0*: ∀ *i* < *length* (*take-indices signs S*). (*col* (*alt-matrix-A* (*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* < *length* (*take-indices signs S*)
have *i-lt*: *i* < *length* (*map* ((!) (*cols* (*alt-matrix-A signs subsets*))) *S*) **using** *asm*
apply (*auto*) **unfolding** *take-indices-def* **by** *auto*
have *h0*: *vec* (*length subsets*) (λ*j*. *z* (*subsets* ! *j*) (*map* ((!) *signs*) *S* ! *i*)) =
vec (*length subsets*) (λ*j*. *z* (*subsets* ! *j*) (*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* ∈ *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* < *length* (*signs*) **using** *well-def-h*
using *i-lt in-set-member* **by** *fastforce*
have
map-eq: (*map* ((!) (*cols* (*alt-matrix-A signs subsets*))) *S*) ! *i* = *nth* (*cols* (*alt-matrix-A signs subsets*)) (*S* ! *i*)
using *i-lt* **by** *auto*
have *nth* (*cols* (*alt-matrix-A signs subsets*)) (*S* ! *i*) ∈ *carrier-vec* (*dim-row* (*alt-matrix-A signs subsets*))
using *col-dim* **unfolding** *cols-def* **using** *nth-map well-d*
by (*simp add*: ‹*dim-col* (*alt-matrix-A signs subsets*) = *length signs*›)
then show *?thesis* **using** *map-eq* **unfolding** *alt-matrix-A-def* **by** *auto*
qed
have *h1*: *col* (*take-cols-from-matrix* (*alt-matrix-A signs subsets*) *S*) *i* =
col (*mat-of-cols* (*dim-row* (*alt-matrix-A signs subsets*)) (*map* ((!) (*cols* (*alt-matrix-A 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 signs subsets*)) (*map* ((!) (*cols* (*alt-matrix-A signs subsets*))) *S*)) *i*
= *nth* (*map* ((!) (*cols* (*alt-matrix-A signs subsets*))) *S*) *i*
using *dim i-lt asm col-mat-of-cols*[**where** *j* = *i*, **where** *n* = (*dim-row* (*alt-matrix-A signs subsets*)),
where *vs* = (*map* ((!) (*cols* (*alt-matrix-A signs subsets*))) *S*)]
by *blast*
have *h3*: *col* (*take-cols-from-matrix* (*alt-matrix-A signs subsets*) *S*) *i* = (*col* (*alt-matrix-A signs subsets*) (*S* !*i*))
using *h1 h2* **apply** (*auto*)
by (*metis alt-matrix-char asm cols-nth dim-col-mat*(*1*) *in-set-member length-map mat-of-rows-list-def matrix-A-def nth-map nth-mem take-indices-def well-def-h*)
have *vec* (*length subsets*) (λ*j*. *z* (*subsets* ! *j*) (*signs* ! (*S* ! *i*))) = (*col* (*alt-matrix-A signs subsets*) (*S* !*i*))
by (*metis asm in-set-member length-map nth-mem signs-are-cols take-indices-def well-def-h*)
then have *vec* (*length subsets*) (λ*j*. *z* (*subsets* ! *j*) (*take-indices signs S* ! *i*)) =
col (*take-cols-from-matrix* (*alt-matrix-A signs subsets*) *S*) *i*
using *h0 h3*
by (*simp add*: *take-indices-def*)
then show *col* (*alt-matrix-A* (*take-indices signs S*) *subsets*) *i* =
col (*take-cols-from-matrix* (*alt-matrix-A signs subsets*) *S*) *i*
using *asm signs-are-cols*[**where** *signs* = (*take-indices signs S*), **where** *subsets* = *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*:: *real poly*
fixes *qs* :: *real poly list*
fixes *subsets* :: *nat list list*
fixes *signs* :: *rat list list*
fixes *S*:: *nat list*
assumes *well-def-h*: $\forall x$. *List.member S x* $\longrightarrow$ $x <$ *length signs*
assumes *nonzero*: $p \neq 0$
shows *matrix-A* (*take-indices signs S*) *subsets* = *take-cols-from-matrix* (*matrix-A signs subsets*) *S*
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*: *length subsets* $\geq$ *length signs*
assumes *well-def-h*: $\forall x$. *List.member S x* $\longrightarrow$ $x <$ *length subsets*
assumes *nonzero*: $p \neq 0$
shows *alt-matrix-A signs* (*take-indices subsets S*) = *take-rows-from-matrix* (*alt-matrix-A signs subsets*) *S*
proof −
have *h0a*: *dim-row* (*take-rows-from-matrix* (*alt-matrix-A signs subsets*) *S*) = *length* (*take-indices subsets S*)
unfolding *take-rows-from-matrix-def* **apply** (*auto*) **unfolding** *take-indices-def*
by *auto*
have *h0*: $\forall i <$ *length* (*take-indices subsets S*). (*row* (*alt-matrix-A signs* (*take-indices subsets S*)) *i* =
row (*take-rows-from-matrix* (*alt-matrix-A signs subsets*) *S*) *i*)
proof *clarsimp*
fix *i*
assume *asm*: $i <$ *length* (*take-indices subsets S*)
have *i-lt*: $i <$ *length* (*map* ((!) (*rows* (*alt-matrix-A signs subsets*))) *S*) **using** *asm*
apply (*auto*) **unfolding** *take-indices-def* **by** *auto*
have *h0*: *row* (*take-rows-from-matrix* (*alt-matrix-A signs subsets*) *S*) *i* =
row (*mat-of-rows* (*dim-col* (*alt-matrix-A signs subsets*)) (*map* ((!) (*rows* (*alt-matrix-A signs subsets*))) *S*)) *i*
unfolding *take-rows-from-matrix-def take-indices-def* **by** *auto*
have *dim*: (*map* ((!) (*rows* (*alt-matrix-A signs subsets*))) *S*) ! *i* $\in$ *carrier-vec* (*dim-col* (*alt-matrix-A signs subsets*))

proof −
have *dim-col* (*alt-matrix-A signs subsets*) = *length* (*signs*)
by (*simp add*: *alt-matrix-A-def*)
then have *lenh*: *dim-col* (*alt-matrix-A signs subsets*) ≤ *length* (*subsets*)
using *assms*
by *auto*
have *well-d*: *S* ! *i* < *length* (*subsets*) **using** *well-def-h*
using *i-lt in-set-member* **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 in-set-member 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 (*meson in-set-member nth-mem*)
then show *?thesis*
using *f1* **by** (*simp add*: ‹⋀*subsets signs*. ∀ *i*<*length subsets*. *row* (*alt-matrix-A signs subsets*) *i* = *vec* (*length signs*) (λ*j*. *z* (*subsets* ! *i*) (*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*: *length subsets* $\geq$ *length signs*
assumes *well-def-h*: $\forall x$. *List.member S x* $\longrightarrow$ $x <$ *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*: *set* (*characterize-consistent-signs-at-roots-copr p qs*) $\subseteq$ *set*(*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*: ∀ *x*. *List.member* (*find-nonzeros-from-input-vec ?lhs-vec*) *x* ⟶*x* < *length* (*signs*)
using *h30 size-of-lhs* **unfolding** *find-nonzeros-from-input-vec-def* **apply** (*auto*)
by (*metis atLeastLessThan-iff filter-is-subset member-def set-upt subset-eq*)
have *h3b-a*: ∀ *x*. *List.member* (*find-nonzeros-from-input-vec ?lhs-vec*) *x* ⟶*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* (λ*i*. *solve-for-lhs p qs subsets* (*matrix-A signs subsets*) $ *i* ≠ *0*)
[*0*..<*length subsets*])
≤ *length subsets* **using** *length-filter-le*[**where** *P* = (λ*i*. *solve-for-lhs p qs subsets* (*matrix-A signs subsets*) $ *i* ≠ *0*),
where *xs* = [*0*..<*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* (λ*i*. *solve-for-lhs p qs subsets* (*matrix-A signs subsets*) $ *i* ≠ *0*)
[*0*..<*dim-vec* (*solve-for-lhs p qs subsets* (*matrix-A signs subsets*))])
≤ *length subsets* **using** *h3ba*
by *auto*
then have *h3b*: *length subsets* ≥ *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*: ∀ *x*. *List.member* (*rows-to-keep* (*reduce-mat-cols ?A ?lhs-vec*)) *x* ⟶ *x* < *length* (*subsets*)
proof *clarsimp*
fix *x*
assume *x-mem*: *List.member* (*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*))))) *x*
obtain *nn* :: *rat list list* ⇒ *nat list* ⇒ *nat* **where**
∀ *x2 x3*. (∃ *v4*. *v4* ∈ *set x3* ∧ ¬ *v4* < *length x2*) = (*nn x2 x3* ∈ *set x3* ∧ ¬ *nn x2 x3* < *length x2*)
by *moura*
then have *f4*: *nn signs* (*find-nonzeros-from-input-vec* (*solve-for-lhs p qs subsets* (*matrix-A signs subsets*))) ∈ *set* (*find-nonzeros-from-input-vec* (*solve-for-lhs p qs subsets* (*matrix-A signs subsets*))) ∧ ¬ *nn signs* (*find-nonzeros-from-input-vec* (*solve-for-lhs p qs subsets* (*matrix-A signs subsets*))) < *length signs* ∨ *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*))) ∧ *x* ∈ *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*
by (*metis h3a in-set-member rows-to-keep-def x-mem*)
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 using in-set-member 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*) ⊆ *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-assm*: *dim-col ?step1-mat*^{T} ≥ *dim-row ?step1-mat*^{T}
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-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* ∈ *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-assm 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-mat*^{T}))) ≤ (*length* (*find-nonzeros-from-input-vec ?lhs*))
using *obv length-pivot-positions-dim-row*[**where** *A* = (*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* (*pivot-positions* (*gauss-jordan-single* (*?step1-mat*^{T}))) ≤ *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 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*)))))
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 subsets* (*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 subsets*)) (*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*:: *real poly*
fixes *qs* :: *real poly list*
fixes *subsets* :: *nat list list*
fixes *signs* :: *rat list list*
assumes *length-init* : ∀ *x*∈ *set*(*signs*). *length x* = *length qs*
assumes *sat-eq*: *satisfy-equation p qs subsets signs*
assumes *inv-mat*: *invertible-mat* (*matrix-A signs subsets*)
shows ∀ *x* ∈ *set*(*reduction-signs p qs signs subsets* (*matrix-A signs subsets*)).
length x = *length qs*
proof *clarsimp*
fix *x*
assume *x-in-set*: *x* ∈ *set* (*reduction-signs p qs signs subsets* (*matrix-A signs subsets*))
have *List.member* (*reduction-signs p qs signs subsets* (*matrix-A signs subsets*)) *x*
using *x-in-set* **by** (*simp add*: *in-set-member*)
then have *take-ind*: *List.member* (*take-indices signs*
(*find-nonzeros-from-input-vec* (*solve-for-lhs p qs subsets* (*matrix-A signs subsets*)))) *x*
unfolding *reduction-signs-def* **by** *auto*
have *find-nz-len*: ∀ *y*. *List.member* (*find-nonzeros-from-input-vec* (*solve-for-lhs p qs subsets* (*matrix-A signs subsets*))) *y* ⟶ *y* < *length signs*
using *size-of-lhs sat-eq inv-mat construct-lhs-matches-solve-for-lhs*[*of p qs subsets signs*] **unfolding** *find-nonzeros-from-input-vec-def*
by (*metis atLeastLessThan-iff filter-is-subset in-set-member set-upt subset-code*(*1*))

then have ∃ *n* < *length signs*. *x* = *signs* ! *n*
using *take-ind*
by (*metis in-set-conv-nth reduction-signs-def take-indices-lem x-in-set*)
then show *length x* = *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*: *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 *h1*: *distinct* (*find-nonzeros-from-input-vec* (*solve-for-lhs p qs subsets* (*matrix-A signs subsets*)))
unfolding *find-nonzeros-from-input-vec-def*
using *distinct-filter*
using *distinct-upt* **by** *blast*
have *h2*: ($\bigwedge$*j*. *j* ∈ *set* (*find-nonzeros-from-input-vec* (*solve-for-lhs p qs subsets* (*matrix-A signs subsets*))) ⟹
j < *length signs*)
proof −
fix *j*
assume *j* ∈ *set* (*find-nonzeros-from-input-vec* (*solve-for-lhs p qs subsets* (*matrix-A 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 subsets*)) (*length qs*)
unfolding *all-list-constr-def*
proof *clarsimp*
fix *x*
assume *in-red-subsets*: *List.member* (*reduction-subsets p qs signs subsets* (*matrix-A signs subsets*)) *x*
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*: $y \in set\ (rows\text{-}to\text{-}keep$
$(take\text{-}cols\text{-}from\text{-}matrix\ (matrix\text{-}A\ signs\ subsets)\ (find\text{-}nonzeros\text{-}from\text{-}input\text{-}vec$
$(solve\text{-}for\text{-}lhs\ p\ qs\ subsets\ (matrix\text{-}A\ signs\ subsets)))))$
have *in-set-2*: $y \in set\ (rows\text{-}to\text{-}keep$
$(take\text{-}cols\text{-}from\text{-}matrix\ (matrix\text{-}A\ signs\ subsets)\ (find\text{-}nonzeros\text{-}from\text{-}input\text{-}vec$
$(construct\text{-}lhs\text{-}vector\ p\ qs\ signs))))$
using *in-set solve-construct* **by** *simp*
let *?lhs-vec* = $(solve\text{-}for\text{-}lhs\ p\ qs\ subsets\ (matrix\text{-}A\ signs\ subsets))$
have *h30*: $(construct\text{-}lhs\text{-}vector\ p\ qs\ signs) = ?lhs\text{-}vec$
using *assms construct-lhs-matches-solve-for-lhs*
by *simp*
have *h3a*: $\forall x.\ List.member\ (find\text{-}nonzeros\text{-}from\text{-}input\text{-}vec\ ?lhs\text{-}vec)\ x \longrightarrow x <$
$length\ (signs)$
using *h30 size-of-lhs* **unfolding** *find-nonzeros-from-input-vec-def* **apply** $(auto)$
by $(metis\ atLeastLessThan\text{-}iff\ filter\text{-}is\text{-}subset\ member\text{-}def\ set\text{-}upt\ subset\text{-}eq)$
have *h3c*: $\forall x.\ List.member\ (rows\text{-}to\text{-}keep\ (reduce\text{-}mat\text{-}cols\ (matrix\text{-}A\ signs$
$subsets)\ (solve\text{-}for\text{-}lhs\ p\ qs\ subsets\ (matrix\text{-}A\ signs\ subsets))))\ x \longrightarrow x < length$
$(subsets)$
proof *clarsimp*
fix x
assume *x-mem*: *List.member* $(rows\text{-}to\text{-}keep$
$(take\text{-}cols\text{-}from\text{-}matrix\ (matrix\text{-}A\ signs\ subsets)$
$(find\text{-}nonzeros\text{-}from\text{-}input\text{-}vec\ (solve\text{-}for\text{-}lhs\ p\ qs\ subsets\ (matrix\text{-}A\ signs$
$subsets)))))\ x$
obtain *nn* :: *rat list list* $\Rightarrow$ *nat list* $\Rightarrow$ *nat* **where**
$\forall x2\ x3.\ (\exists v4.\ v4 \in set\ x3 \wedge \neg\ v4 < length\ x2) = (nn\ x2\ x3 \in set\ x3 \wedge \neg$
$nn\ x2\ x3 < length\ x2)$
by *moura*
then have *f4*: *nn signs* $(find\text{-}nonzeros\text{-}from\text{-}input\text{-}vec\ (solve\text{-}for\text{-}lhs\ p\ qs\ sub\text{-}$
$sets\ (matrix\text{-}A\ signs\ subsets))) \in set\ (find\text{-}nonzeros\text{-}from\text{-}input\text{-}vec\ (solve\text{-}for\text{-}lhs\ p$
$qs\ subsets\ (matrix\text{-}A\ signs\ subsets))) \wedge \neg\ nn\ signs\ (find\text{-}nonzeros\text{-}from\text{-}input\text{-}vec$
$(solve\text{-}for\text{-}lhs\ p\ qs\ subsets\ (matrix\text{-}A\ signs\ subsets))) < length\ signs \vee matrix\text{-}A$
$(take\text{-}indices\ signs\ (find\text{-}nonzeros\text{-}from\text{-}input\text{-}vec\ (solve\text{-}for\text{-}lhs\ p\ qs\ subsets\ (matrix\text{-}A$
$signs\ subsets)))) subsets = take\text{-}cols\text{-}from\text{-}matrix\ (matrix\text{-}A\ signs\ subsets)\ (find\text{-}nonzeros\text{-}from\text{-}input\text{-}vec$
$(solve\text{-}for\text{-}lhs\ p\ qs\ subsets\ (matrix\text{-}A\ signs\ subsets)))$
using *h3a nonzero reduce-system-matrix-signs-helper* **by** *auto*
then have *matrix-A* $(take\text{-}indices\ signs\ (find\text{-}nonzeros\text{-}from\text{-}input\text{-}vec\ (solve\text{-}for\text{-}lhs$
$p\ qs\ subsets\ (matrix\text{-}A\ signs\ subsets)))) subsets = take\text{-}cols\text{-}from\text{-}matrix\ (matrix\text{-}A$
$signs\ subsets)\ (find\text{-}nonzeros\text{-}from\text{-}input\text{-}vec\ (solve\text{-}for\text{-}lhs\ p\ qs\ subsets\ (matrix\text{-}A$
$signs\ subsets))) \wedge x \in set\ (map\ snd\ (pivot\text{-}positions\ (gauss\text{-}jordan\text{-}single\ (take\text{-}cols\text{-}from\text{-}matrix$
$(matrix\text{-}A\ signs\ subsets)\ (find\text{-}nonzeros\text{-}from\text{-}input\text{-}vec\ (solve\text{-}for\text{-}lhs\ p\ qs\ subsets$
$(matrix\text{-}A\ signs\ subsets))))^T)))$
by $(metis\ h3a\ in\text{-}set\text{-}member\ rows\text{-}to\text{-}keep\text{-}def\ x\text{-}mem)$
thus $x < length\ (subsets)$ **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* < *length subsets* **using** *in-set-member* **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*
by (*metis all-list-constr-def in-set-conv-nth in-set-member length-init*)
qed

15 Overall Lemmas

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

lemma *getter-functions*: *calculate-data p qs* = (*get-matrix* (*calculate-data p qs*), (*get-subsets* (*calculate-data p qs*), *get-signs* (*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* ≠ *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 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* = (*get-subsets* (*snd* (*combine-systems p* (*qs1*, *matrix1*, *subsets1*, *signs1*)
(*qs2*, *matrix2*, *subsets2*, *signs2*))))
let *?signs* = (*get-signs* (*snd* (*combine-systems p* (*qs1*, *matrix1*, *subsets1*, *signs1*) (*qs2*, *matrix2*, *subsets2*, *signs2*))))
let *?matrix* = (*get-matrix* (*snd* (*combine-systems p* (*qs1*, *matrix1*, *subsets1*, *signs1*) (*qs2*, *matrix2*, *subsets2*, *signs2*))))
have *h1*: *all-list-constr ?subsets* (*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 satisfies-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 all-list-constr-def in-set-member list-constr-def satisfies-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* **by** *auto*
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*: $p \neq 0$
assumes *nontriv1*: *length qs1* > 0
assumes *pairwise-rel-prime1*: $\forall q.$ ((*List.member qs1 q*) $\longrightarrow$ (*coprime p q*))
assumes *nontriv2*: *length qs2* > 0
assumes *pairwise-rel-prime2*: $\forall q.$ ((*List.member qs2 q*) $\longrightarrow$ (*coprime p q*))
assumes *satisfies-properties-sys1*: *satisfies-properties p qs1* (*get-subsets* (*calculate-data p qs1*)) (*get-signs* (*calculate-data p qs1*)) (*get-matrix* (*calculate-data p qs1*))
assumes *satisfies-properties-sys2*: *satisfies-properties p qs2* (*get-subsets* (*calculate-data p qs2*)) (*get-signs* (*calculate-data p qs2*)) (*get-matrix* (*calculate-data p qs2*))
shows *satisfies-properties p* (*qs1*@*qs2*) (*get-subsets* (*snd* ((*combine-systems p* (*qs1*,*calculate-data p qs1*) (*qs2*,*calculate-data p qs2*)))))
(*get-signs* (*snd* ((*combine-systems p* (*qs1*,*calculate-data p qs1*) (*qs2*,*calculate-data p qs2*)))))
(*get-matrix* (*snd* ((*combine-systems p* (*qs1*,*calculate-data p qs1*) (*qs2*,*calculate-data p qs2*)))))
using *combining-sys-satisfies-properties-helper*
by (*metis getter-functions nontriv1 nontriv2 nonzero pairwise-rel-prime1 pairwise-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 qs q*) $\longrightarrow$ (*coprime p q*))
assumes *satisfies-properties-sys*: *satisfies-properties p qs subsets signs matrix*
shows *satisfies-properties p qs* (*get-subsets* (*reduce-system p* (*qs*,*matrix*,*subsets*,*signs*)))
(*get-signs* (*reduce-system p* (*qs*,*matrix*,*subsets*,*signs*)))
(*get-matrix* (*reduce-system p* (*qs*,*matrix*,*subsets*,*signs*)))
proof −
have *h1*: *all-list-constr* (*get-subsets* (*reduce-system p* (*qs*, *matrix*, *subsets*, *signs*))) (*length qs*)
using *reduction-doesnt-break-subsets assms reduction-subsets-is-get-subsets satisfies-properties-def satisfies-properties-sys* **by** *auto*
have *h2*: *well-def-signs* (*length qs*) (*get-signs* (*reduce-system p* (*qs*, *matrix*, *subsets*, *signs*)))
using *reduction-doesnt-break-length-signs*[*of signs qs p subsets*] *assms reduction-signs-is-get-signs satisfies-properties-def well-def-signs-def* **by** *auto*
have *h3*: *distinct* (*get-signs* (*reduce-system p* (*qs*, *matrix*, *subsets*, *signs*)))
using *reduction-signs-are-distinct*[*of p qs subsets signs*] *assms reduction-signs-is-get-signs satisfies-properties-def* **by** *auto*
have *h4*: *satisfy-equation p qs* (*get-subsets* (*reduce-system p* (*qs*, *matrix*, *subsets*,

signs)))
(*get-signs* (*reduce-system p* (*qs*, *matrix*, *subsets*, *signs*)))
using *reduce-system-matrix-equation-preserved*[*of p qs signs subsets*] *assms satisfies-properties-def* **by** *auto*
have *h5*: *invertible-mat* (*get-matrix* (*reduce-system p* (*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* (*qs*, *matrix*, *subsets*, *signs*)) =
matrix-A (*get-signs* (*reduce-system p* (*qs*, *matrix*, *subsets*, *signs*)))
(*get-subsets* (*reduce-system p* (*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*) ⊆ *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* ≠ *0*
assumes *len1*: *length qs* = *1*
assumes *pairwise-rel-prime*: ∀ *q*. ((*List.member qs q*) ⟶ (*coprime p q*))
shows *satisfies-properties p qs* (*get-subsets* (*calculate-data p qs*)) (*get-signs* (*calculate-data p qs*)) (*get-matrix* (*calculate-data p qs*))
proof −
have *h1*: *all-list-constr* [[],[*0*]] (*length qs*)
using *len1* **unfolding** *all-list-constr-def list-constr-def* **apply** (*auto*)
by (*metis* (*full-types*) *length-Cons less-Suc0 list.size*(*3*) *list-all-length list-all-simps*(*2*) *member-rec*(*1*) *member-rec*(*2*) *nth-Cons-0*)
have *h2*: *well-def-signs* (*length qs*) [[*1*],[−*1*]]
unfolding *well-def-signs-def* **using** *len1 in-set-member*
by *auto*
have *h3*: *distinct* ([[*1*],[−*1*]]::*rat list list*)
unfolding *distinct-def* **using** *in-set-member* **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*) ⊆ *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* ∧ *List.member l2 m*) ⟶ (*distinct* (*l1*@*l2*) = *False*)
proof −
have *h1*: *List.member l1 m* ⟶ (∃ *n*. *n* < *length l1* ∧ (*nth l1 n*) = *m*)
using *member-def nth-find-first*
by (*simp add*: *member-def in-set-conv-nth*)
have *h2*: ∀ *n*. *n* < *length l1* ∧ (*nth l1 n*) = *m* ⟶ (*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* ⟶ (∃ *n*. *n* < *length l1* ∧ (*nth* (*l1*@*l2*) *n*) = *m*)
using *h1 h2* **by** *auto*
have *h4*: *List.member l2 m* ⟶ (∃ *n*. (*nth l2 n*) = *m*)
by (*meson member-def nth-find-first*)
have *h5*: ∀ *n*. (*nth l2 n*) = *m* ⟶ (*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 ⟶ (∃ n. (nth (l1@l2) (n + length l1)) = m)
    using h4 h5
    by blast
  show ?thesis using h3 h6
    by (metis distinct-append equalityI insert-disjoint(1) insert-subset member-def
order-refl)
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 ≠ 0 ∧ (length qs > 0) ∧ (∀ q. ((List.member qs q) ⟶ (coprime p q)))
)
   ⟶ satisfies-properties p qs (get-subsets (calculate-data p qs)) (get-signs (calculate-data
p qs)) (get-matrix (calculate-data p qs))
proof (induction length qs arbitrary: qs rule: less-induct)
  case less
  have len1-h: 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))
    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 ⟶ (length ?q1 > 0) by auto
  have h-q2-len: length qs > 1 ⟶ (length ?q2 > 0) by auto
 have h-q1-copr: (∀ q. ((List.member qs q) ⟶ (coprime p q))) ⟶ (∀ q. ((List.member
?q1 q) ⟶ (coprime p q)))
  proof −
    have mem-hyp: (∀ q. ((List.member ?q1 q) ⟶ (List.member qs q)))
```

by (*meson in-set-member in-set-takeD*)
then show *?thesis*
by *blast*
qed
have *h-q2-copr*: (∀ *q*. ((*List.member qs q*) ⟶ (*coprime p q*))) ⟶ (∀ *q*. ((*List.member ?q2 q*) ⟶ (*coprime p q*)))
proof −
have *mem-hyp*: (∀ *q*. ((*List.member ?q2 q*) ⟶ (*List.member qs q*)))
by (*meson in-set-dropD in-set-member*)
then show *?thesis*
by *blast*
qed
have *q1-sat-props*: *length qs* > *1* ⟶ (*p* ≠ *0* ∧ (*length qs* > *0*) ∧ (∀ *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 member-rec*(*1*))
have *match*: *satisfy-equation p qs ?subsets ?signs*
using *ex-q base-case-satisfy-equation nonzero pairwise-rel-prime*
apply (*auto*)
by (*simp add*: *member-rec*(*1*))
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-copr 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*: ($\bigwedge j.\ j \in$ *set* (*get-signs* (*calculate-data p qs1*)) $\Longrightarrow$ *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*: $(\bigwedge l\ i.\ l \in set\ (get\text{-}subsets(calculate\text{-}data\ p\ qs1)) \Longrightarrow i \in set\ l \Longrightarrow i < (length\ qs1))$
unfolding *all-list-constr-def list-constr-def* **using** *in-set-member*
by (*metis 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 $(p \neq 0 \wedge length\ qs > 0 \wedge$
$(\forall q.\ ((List.member\ qs\ q) \longrightarrow (coprime\ p\ q)))) \longrightarrow$
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. List.member qsa q ⟶ 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. List.member qs q ⟶ 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 ⟶ length ?q1 > 0
      by auto
    have qs-more-q1: length qs>1 ⟶ length qs > length ?q1
      by auto
    have pairwise-rel-prime-q1: ∀ q. ((List.member ?q1 q) ⟶ (coprime p q))
    proof clarsimp
      fix q
      assume mem: List.member (take (length qs div 2) qs) q
      have List.member qs q using mem set-take-subset[where n = length qs div
2]
      proof −
        show ?thesis
          by (meson ‹List.member (take (length qs div 2) qs) q› in-set-member
in-set-takeD)
      qed
      then show coprime p q
        using copr by blast
    qed
    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 pairwise-rel-prime-q2: ∀ q. ((List.member ?q2 q) ⟶ (coprime p q))
    proof clarsimp
      fix q
      assume mem: List.member (drop (length qs div 2) qs) q
      have List.member qs q using mem set-take-subset[where n = length qs div
2]
      proof −
        show ?thesis
          by (meson ‹List.member (drop (length qs div 2) qs) q› in-set-dropD
```

in-set-member)
qed
then show *coprime p q*
using *copr* by *blast*
qed
have *key-h*: *set* (*snd* (*snd* (*if ?len* ≤ *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* ⟶ *set* (*snd* (*snd* (*if ?len* ≤ *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* ⟶ *set* (*snd* (*snd* (*if ?len* ≤ *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* ⟶ *set* (*snd* (*snd* (*reduce-system p* (*combine-systems*
p (*?q1*, *?left*) (*?q2*, *?right*))))) =
set (*characterize-consistent-signs-at-roots-copr p qs*)
proof −
have *h1*: *?len* > *1* ⟶ *set*(*snd*(*snd*(*?left*))) = *set* (*characterize-consistent-signs-at-roots-copr*
p ?q1)
using *nontriv-q1 pairwise-rel-prime-q1 ind-hyp*[*of ?q1*] *qs-more-q1*
by (*metis find-consistent-signs-at-roots-thm less-numeral-extra*(*3*) *list.size*(*3*))
have *h2*: *?len* > *1* ⟶ *set*(*snd*(*snd*(*?right*))) = *set* (*characterize-consistent-signs-at-roots-copr*
p ?q2)
using *nontriv-q2 pairwise-rel-prime-q2 ind-hyp*[*of ?q2*] *qs-more-q2*
by (*metis* (*full-types*) *find-consistent-signs-at-roots-thm list.size*(*3*)
not-less-zero)
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* ⟶ *set* (*snd* (*snd* (*Let* (*combine-systems p*
(*?q1*, *?left*) (*?q2*, *?right*))
(*reduce-system p*)))) =
set (*characterize-consistent-signs-at-roots-copr 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 p qs) = set (characterize-consistent-signs-at-roots-copr
p 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 ≠ 0
  shows set(find-consistent-signs-at-roots p []) =
        set(characterize-consistent-signs-at-roots-copr p [])
proof −
  obtain a b c where abc: reduce-system p ([1], 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]) = set (characterize-consistent-signs-at-roots-copr
p [1])
    apply (subst find-consistent-signs-at-roots-1 )
    using assms apply auto
    by (simp add: member-rec(1) member-rec(2))
 have set(characterize-consistent-signs-at-roots-copr p []) = drop 1 ‘ set(characterize-consistent-signs-at-roots-c
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 ≠ 0
  assumes ⋀q. q ∈ set qs ⟹ coprime p q
 shows set(find-consistent-signs-at-roots p qs) = set(characterize-consistent-signs-at-roots-copr
p qs)
   by (metis assms find-consistent-signs-at-roots-0 find-consistent-signs-at-roots-1
in-set-member length-greater-0-conv)

lemma find-consistent-signs-at-roots:
  fixes p:: real poly
  fixes qs :: real poly list
  assumes p ≠ 0
  assumes ⋀q. q ∈ set qs ⟹ coprime p q
```

shows *set*(*find-consistent-signs-at-roots p qs*) = *set*(*characterize-consistent-signs-at-roots p qs*)
by (*metis assms csa-list-copr-rel find-consistent-signs-at-roots-copr in-set-member*)

theorem *find-consistent-signs-at-roots-alt*:
assumes $p \neq 0$
assumes $\bigwedge q.\ q \in set\ qs \Longrightarrow coprime\ p\ q$
shows *set* (*find-consistent-signs-at-roots p qs*) = *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* ⇒ (*'a*::*linordered-field list*) ⇒ *bool*
where
lookup-sem (*And l r*) *ls* = (*lookup-sem l ls* ∧ *lookup-sem r ls*)
| *lookup-sem* (*Or l r*) *ls* = (*lookup-sem l ls* ∨ *lookup-sem r ls*)
| *lookup-sem* (*Gt p*) *ls* = (*ls* ! *p* > *0*)
| *lookup-sem* (*Geq p*) *ls* = (*ls* ! *p* ≥ *0*)
| *lookup-sem* (*Lt p*) *ls* = (*ls* ! *p* < *0*)
| *lookup-sem* (*Leq p*) *ls* = (*ls* ! *p* ≤ *0*)
| *lookup-sem* (*Eq p*) *ls* = (*ls* ! *p* = *0*)
| *lookup-sem* (*Neq p*) *ls* = (*ls* ! *p* ≠ *0*)

```
primrec poly-list :: 'a fml ⇒ '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 ⇒ 'a ⇒ nat ⇒ 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 ⇒ 'a ⇒ nat where
  index-of xs y = index-of-aux xs y 0

definition convert :: 'a fml ⇒ (nat fml × '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 ⇒ (rat × (rat poly × nat) list)
  where
    factorize-rat-poly-monic p = (
    let (c,fs) =  factorize-rat-poly p ;
        lcs = prod-list (map (λ(f,i). (lead-coeff f) ^ i) fs) ;
        fs = map (λ(f,i). (normalize f, i)) fs
    in
    (c * lcs,fs)
  )


definition factorize-polys :: rat poly list ⇒ (rat poly list × (rat × (nat × 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 (λ(c,fs). (c, map (λ(f,pow). (index-of factors f, pow) ) fs))
fact-ps
    in
```

```
    (factors,data)
  )

definition undo-factorize :: rat × (nat × nat) list ⇒ rat list ⇒ rat
  where
    undo-factorize cfs signs =
   squash
    (case cfs of (c,fs) ⇒
    (c * prod-list (map (λ(f,pow). (signs ! f) ^ pow) fs)))

definition undo-factorize-polys :: (rat × (nat × nat) list) list ⇒ rat list ⇒ rat list
  where
    undo-factorize-polys ls signs = map (λl. undo-factorize l signs) ls
```

16.3 Auxiliary Polynomial

```
definition crb:: real poly ⇒ 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 ⇒ 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 ⇒ 'a  ⇒ 'a list ⇒ 'a list where
  insertAt n x ls = take n ls @ x # (drop n ls)

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

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

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

16.5 Deciding Univariate Problems

```
definition decide-universal :: rat poly fml ⇒ 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 ⇒ bool
  where [code]:
    decide-existential fml = (
    let (fml-struct,polys) = convert fml;
        conds = find-consistent-signs polys
    in
      find (lookup-sem fml-struct) conds ≠ None
  )
```

17 Proofs

17.1 Parsing and Semantics

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

primrec *fml-sem* :: *rat poly fml* ⇒ *real* ⇒ *bool*
where
fml-sem (*And l r*) *x* = (*fml-sem l x* ∧ *fml-sem r x*)
| *fml-sem* (*Or l r*) *x* = (*fml-sem l x* ∨ *fml-sem r x*)
| *fml-sem* (*Gt p*) *x* = (*rpoly p x* > *0*)
| *fml-sem* (*Geq p*) *x* = (*rpoly p x* ≥ *0*)
| *fml-sem* (*Lt p*) *x* = (*rpoly p x* < *0*)
| *fml-sem* (*Leq p*) *x* = (*rpoly p x* ≤ *0*)
| *fml-sem* (*Eq p*) *x* = (*rpoly p x* = *0*)
| *fml-sem* (*Neq p*) *x* = (*rpoly p x* ≠ *0*)

lemma *poly-list-set-fml*:
shows *set* (*poly-list fml*) = *set-fml fml*
apply (*induction*) **by** *auto*

lemma *convert-semantics-lem*:
assumes ⋀*p*. *p* ∈ *set* (*poly-list fml*) ⟹
ls ! (*index-of ps p*) = *rpoly p x*
shows *fml-sem fml x* = *lookup-sem* (*map-fml* (*index-of ps*) *fml*) *ls*
using *assms* **apply** (*induct fml*)
by *auto*

lemma *index-of-aux-more*:
shows *index-of-aux ls p n* ≥ *n*
apply (*induct ls arbitrary*: *n*)
apply *auto*
using *Suc-leD* **by** *blast*

lemma *index-of-aux-lookup*:
assumes *p* ∈ *set ls*
shows (*index-of-aux ls p n*) − *n* < *length ls*
ls ! ((*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* ∈ *set ls*
shows *index-of ls p* < *length ls*
ls ! (*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-semantics*:
shows *fml-sem fml x* = *lookup-sem* (*fst* (*convert fml*)) (*map* (λ*p*. *rpoly p x*) (*snd* (*convert fml*)))

unfolding *convert-def Let-def* **apply** *simp*
apply (*intro convert-semantics-lem*)
by (*simp add*: *index-of-lookup*(*1*) *index-of-lookup*(*2*))

lemma *convert-closed*:
shows $\bigwedge i.\ i \in$ *set-fml* (*fst* (*convert fml*)) $\Longrightarrow i <$ *length* (*snd* (*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* ⇒ *real* ⇒ *rat list*
where *sign-vec qs x* ≡
map (*squash* ∘ (λ*p*. *rpoly p x*)) *qs*

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

lemma *sign-vec-semantics*:
assumes $\bigwedge i.\ i \in$ *set-fml fml* $\Longrightarrow i <$ *length ls*
shows *lookup-sem fml* (*map* (λ*p*. *rpoly p x*) *ls*) = *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 $\bigwedge i.\ i \in$ *set-fml fml* $\Longrightarrow i <$ *length qs*
assumes *set signs* = *consistent-sign-vectors qs UNIV*
shows (∀ *x*::*real*. *lookup-sem fml* (*map* (λ*p*. *rpoly p x*) *qs*)) ⟷
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 $\bigwedge i.\ i \in$ *set-fml fml* $\Longrightarrow i <$ *length qs*
assumes *set signs* = *consistent-sign-vectors qs UNIV*
shows (∃ *x*::*real*. *lookup-sem fml* (*map* (λ*p*. *rpoly p x*) *qs*)) ⟷
find (*lookup-sem fml*) *signs* ≠ *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 *poly* (*real-of-rat-poly* ($\prod$ (*a*,*b*)∈*s*. *f a b*)) *y* = ($\prod$ (*a*,*b*)∈*s*. *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* ∈ *set ftrs*
shows *sign-vec ftrs x* ! (*index-of ftrs f*) = *squash* (*rpoly f x*)
by (*simp add*: *assms index-of-lookup*(*1*) *index-of-lookup*(*2*) *sign-vec-def*)

lemma *squash-idem*:
shows *squash* (*squash x*) = *squash x*
unfolding *squash-def* **by** *auto*

lemma *squash-mult*:
shows *squash* ((*a*::*real*) ∗ *b*) = *squash a* ∗ *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 *squash* (*prod-list* (*ls*::*real list*)) = *prod-list* (*map 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 *squash* ((*x*::*real*) ^ (*y*::*nat*)) = (*squash x*) ^ *y*
unfolding *squash-def* **apply** *auto*
by (*auto simp add*: *zero-less-power-eq*)

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

lemma *factorize-rat-poly-monic-irreducible-monic*:
assumes *factorize-rat-poly-monic f* = (*c*,*fs*)
assumes (*fi*,*i*) ∈ *set fs*
shows *irreducible fi* ∧ *monic fi*
proof −
obtain *c*′ *fs*′ **where** *cfs*: *factorize-rat-poly f* = (*c*′,*fs*′)
by (*meson surj-pair*)
then have *fs*: *fs* = *map* (λ(*f*,*i*). (*normalize f*, *i*)) *fs*′
using *factorize-rat-poly-monic-def assms* **by** *auto*
obtain *fi*′ **where** (*fi*′,*i*) ∈ *set fs*′ *fi* = *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* = *Polynomial.smult* (*unit-factor* (*lead-coeff a*)) (*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 (∏ (*x*,*y*)∈*set fs*. *Polynomial.smult* ((*f x y*)::*rat*) (*g x y*)) =
Polynomial.smult (∏ (*x*,*y*)∈*set fs*. *f x y*) (∏ (*x*,*y*)∈*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*: $fs = map\ (\lambda(f,i).\ (normalize\ f,\ i))\ fs'$
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*: $lcs = prod\text{-}list\ (map\ (\lambda(f,i).\ lead\text{-}coeff\ f\ \hat{}\ i)\ fs')$ **by** *force*
have *c*: $c = c' * lcs$ **using** *assms* **unfolding** *factorize-rat-poly-monic-def cfs Let-def lcs* **by** *auto*
show $f = 0 \Longrightarrow c = 0$ **using** *c cfs* **by** *auto*
show $f = 0 \Longrightarrow fs = []$ **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*: $i < length\ fs'\ j < length\ fs'\ i \neq j$
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*: $c = c' * (\prod (a,\ i) \in set\ fs'.\ (lead\text{-}coeff\ a)\ \hat{}\ i)$ **using** *c lcs*
by (*simp add*: *dist prod.distinct-set-conv-list*)
have *fseq*: $(\prod (a,\ i) \in set\ fs.\ a\ \hat{}\ i) = (\prod (a,\ i) \in set\ fs'.\ (normalize\ a)\ \hat{}\ i)$
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 $f = Polynomial.smult\ c'\ (\prod (a,\ i) \in set\ fs'.\ a\ \hat{}\ i)$ **using** *sq square-free-factorizationD*(*1*) **by** *blast*
moreover have ... = $Polynomial.smult\ c'\ (\prod (a,\ i) \in set\ fs'.\ (Polynomial.smult\ ((unit\text{-}factor\ (lead\text{-}coeff\ a)))\ (normalize\ a))\ \hat{}\ i)$
apply (*subst undo-normalize*[*symmetric*]) **by** *auto*

moreover have ... = *Polynomial.smult c'*
(∏(*a*, *i*)∈*set fs'*. (*Polynomial.smult* ((*lead-coeff a*) ^ *i*) ((*normalize a*) ^ *i*)))
apply (*subst smult-power*) **by** *auto*
moreover have ... = *Polynomial.smult c'*
(*Polynomial.smult* (∏(*a*, *i*)∈*set fs'*. ((*lead-coeff a*) ^ *i*))
(∏(*a*, *i*)∈*set fs'*. (*normalize a*) ^ *i*))
apply (*subst finite-smult-distr*) **by** (*auto simp add*: *dist*)
moreover have ... = *Polynomial.smult* (*c'* ∗ (∏(*a*, *i*)∈*set fs'*. (*lead-coeff a*) ^ *i*))
(∏(*a*, *i*)∈*set fs'*. (*normalize a*) ^ *i*)
using *smult-smult* **by** *blast*
moreover have ... = *Polynomial.smult c* (∏(*a*, *i*)∈*set fs*. *a* ^ *i*)
unfolding *ceq fseq* **by** *auto*
ultimately show *f* = *Polynomial.smult c* (∏(*a*, *i*)∈*set fs*. *a* ^ *i*) **by** *auto*
fix *a i*
assume *ai*: (*a*,*i*) ∈ *set fs*
obtain *a'* **where** *a'*: (*a'*,*i*) ∈ *set fs'* *a* = *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* < *degree a* *0* < *i* **using** *degree-normalize a'*
using *sq square-free-factorizationD'*(*3*) **by** *fastforce*+
fix *b j*
assume *bj*: (*b*,*j*) ∈ *set fs* (*a*,*i*) ≠ (*b*,*j*)
obtain *b'* **where** *b'*: (*b'*,*j*) ∈ *set fs'* *b* = *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 ⋀*f p*. (*f*,*p*) ∈ *set fs* ⟹ *f* ∈ *set ftrs*
shows *undo-factorize* (*c*,*map* (λ(*f*,*pow*). (*index-of ftrs f*, *pow*)) *fs*) (*sign-vec ftrs x*) = *squash* (*rpoly p x*)
proof −
have *p*: *p* = *smult c* (∏(*a*, *i*)∈ *set fs*. *a* ^ *i*)
using *assms*(*1*) *factorize-rat-poly-monic-square-free-factorization square-free-factorizationD*(*1*) **by** *blast*
have *fs*: *distinct fs*
using *assms*(*1*) *factorize-rat-poly-monic-square-free-factorization square-free-factorizationD*(*5*) **by** *blast*
have *rpoly p x* = ((*real-of-rat c*) ∗ *rpoly* (∏(*a*, *i*)∈ *set fs*. *a* ^ *i*) *x*)
using *p* **by** (*simp add*: *of-rat-hom.map-poly-hom-smult*)
moreover have ... = ((*real-of-rat c*) ∗ *rpoly* (∏*ai*∈ *set fs*. *case ai of* (*a*,*i*) ⇒ *a* ^ *i*) *x*)
by *blast*
moreover have ... = ((*real-of-rat c*) ∗ (∏*ai*∈ *set fs*. *case ai of* (*a*,*i*) ⇒ *rpoly* (*a* ^ *i*) *x*))

by (*simp add*: *finite-prod-map-of-rat-poly-hom*)
moreover have ... = ((*real-of-rat c*) * (∏ *ai*∈ *set fs*. *case ai of* (*a*,*i*) ⇒ (*rpoly a x*) ^ *i*))
by (*metis* (*mono-tags*, *lifting*) *of-rat-poly-hom.hom-power poly-hom.hom-power split-cong*)
moreover have ... = ((*real-of-rat c*) * (*prod-list* (*map* (λ*ai*. *case ai of* (*a*,*i*) ⇒ (*rpoly a x*) ^ *i*) *fs*)))
by (*simp add*: *fs prod.distinct-set-conv-list*)
ultimately have *rpoly p x* = ((*real-of-rat c*) * (*prod-list* (*map* (λ*ai*. *case ai of* (*a*,*i*) ⇒ (*rpoly a x*) ^ *i*) *fs*))) **by** *auto*

then have *squash* (*rpoly p x*) = *squash c* * *prod-list* (*map squash* (*map* (λ*ai*. *case ai of* (*a*,*i*) ⇒ (*rpoly a x*) ^ *i*) *fs*))
by (*auto simp add*: *squash-mult squash-prod-list o-def*)
moreover have ... = *squash c* * *prod-list* (*map* (λ*ai*. *case ai of* (*a*,*i*) ⇒ *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* (λ*ai*. *case ai of* (*a*,*i*) ⇒ *squash* (*rpoly a x*) ^ *i*) *fs*)
using *squash-pow*
by *presburger*
have *undo-factorize*
(*c*, *map* (λ(*f*, *pow*).(*index-of ftrs f*, *pow*)) *fs*) (*sign-vec ftrs x*) =
squash
(*c* * (∏ *xa*←*fs*. *case xa of* (*f*, *y*) ⇒ *sign-vec ftrs x* ! *index-of ftrs f* ^ *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* * (∏ *xa*←*fs*. *case xa of* (*f*, *y*) ⇒ (*squash* (*rpoly f x*)) ^ *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* ∈ *set ps*
assumes *factorize-rat-poly-monic p* = (*c*,*fs*)
shows *set* (*map fst fs*) ⊆ *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 *fsts*:*set* (*map fst fs*) ⊆ *set ftrs*
using *assms factorize-polys-has-factors il leq nth-mem* **by** *fastforce*
have ∗:*data* ! *i* = (*c*,*map* (λ(*f*,*pow*). (*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*) **unfolding** ∗
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* ⋀*f*. *f* ∈ *set fs* ⟹ *irreducible f* ∧ *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 ⋀*f*. *f* ∈ *set fs* ⟹ *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* ≠ *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 ⋀*f g*. *f* ∈ *set fs* ⟹ *g* ∈ *set fs* ⟹ *f* ≠ *g* ⟹ *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-coprimereal-poly*:
shows *coprime p* (*q*::*rat poly*) ⟷ *coprime* (*real-of-rat-poly p*) ((*real-of-rat-poly q*)::*real poly*)
proof −
have *forward*: *coprime p* (*q*::*rat poly*) ⟶ *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*) ⟹ *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* ⟹ *False*
proof −
assume *deg*: *degree* (*gcd p* (*q*::*rat poly*)) > *0*
then have ∃ *y*. *y dvd p* ∧ *y dvd q* ∧ *degree y* > *0*
by *blast*
then obtain *y* **where** *yprop*: *y dvd p* ∧ *y dvd q* ∧ *degree y* > *0*
by *auto*
then have (*real-of-rat-poly y*) *dvd* (*real-of-rat-poly p*) ∧
(*real-of-rat-poly y*) *dvd* (*real-of-rat-poly q*) ∧ *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 ⋀*f*. *f* ∈ *set fss* ⟹ *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 ⋀*f g*. *f* ∈ *set fss* ⟹ *g* ∈ *set fss* ⟹ *f* ≠ *g* ⟹ *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 ⋀*p*. *p* ∈ *set ps* ⟹ *p* ≠ *0*
assumes *coprime* (*prod-list ps*) (*q*::*real poly*)
shows ⋀*p*. *p* ∈ *set ps* ⟹ *coprime p q*
proof −
fix *p*
assume *p* ∈ *set ps*
then obtain *r* **where** *r*: *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*: ⋀*p q*. *p* ∈ *set fs* ⟹ *q* ∈ *set fs* ⟹ *p* ≠ *q* ⟹ *coprime p q* **.**
from *factorize-polys-square-free*[*OF assms*]
have *sq*: ⋀*p*. *p* ∈ *set fs* ⟹ *square-free p* **.**
thus *prod-list fs* ≠ *0* **unfolding** *prod-list-zero-iff*
using *square-free-def* **by** *blast*
fix *q*
assume *degree q* > *0 q* ∗ *q dvd prod-list fs*
from *irreducible*$_d$*-factor*[*OF this*(*1*)] *this*(*2*) **obtain** *q* **where**
irr: *irreducible q* **and** *dvd*: *q* ∗ *q dvd prod-list fs* **unfolding** *dvd-def* **by** *auto*

hence *dvd'*: *q dvd prod-list fs* **unfolding** *dvd-def* **by** *auto*
from *irreducible-dvd-prod-list*[*OF irr dvd'*] **obtain** *b* **where**
mem: *b* ∈ *set fs* **and** *dvd1*: *q 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* ≠ *0* **and** *dq*: *degree q* > *0* **unfolding** *irreducible*$_d$*-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*: ¬ *q dvd k* **by** *simp*
from *dvd*[*unfolded bs b*] **have** *q* ∗ *q dvd q* ∗ (*k* ∗ *prod-list* (*bs1* @ *bs2*))
by (*auto simp*: *ac-simps*)
with *q0* **have** *q dvd k* ∗ *prod-list* (*bs1* @ *bs2*) **by** *auto*
with *irr qk* **have** *q dvd 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 ⋀*f*. *f* ∈ *set fss* ⟹ *coprime f* (*pderiv* (*prod-list fss*))
proof −
fix *f*
assume *f*: *f* ∈ *set fss*
from *factorize-polys-map-square-free*[*OF assms*]
have *sq*: ⋀*p*. *p* ∈ *set fss* ⟹ *square-free p* **.**
have *z*: ⋀*p*. *p* ∈ *set fss* ⟹ *p* ≠ *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* ⇒ *bool*
where *pairwise-coprime-list qs* =
(∀ *m* < *length qs*. ∀ *n* < *length qs*.
m ≠ *n* ⟶ *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 (∀ *m* < *length ?fs*. ∀ *n* < *length ?fs*.
m ≠ *n* ⟶ *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 degree-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 ∀ *x* . *poly* (*prod-list* (*qs*:: *rat poly list*)) *x* = *0* ⟷ (∃ *q* ∈ *set* (*qs*). *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 ⋀*f*. *f* ∈ *set fss* ⟹ *coprime f* (*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*: ∀ *x*. *poly f x* = *0* ⟶ *poly* (*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*: ∀ *x*. *poly ?inf-p x* = *0* ⟷ *x* = *crb* (*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-pos*[**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 ?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*: ∀ *x0*. − (*x0*::*real*) = − *1* ∗ *x0*
by *simp*
have (*1*::*real*) ≠ *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: ∀ x. poly f x = 0 ⟶ poly (prod-list fss) x = 0
      using f-in apply (auto)
      using poly-prod-list-zero-iff by blast
    have zero-prop: ∀ x. poly ?ninf-p x = 0 ⟷ x = −crb (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*: $fs \neq []$
assumes *fss-is*: $fss = map\ real\text{-}of\text{-}rat\text{-}poly\ fs$
shows $(coprime\text{-}r\ fss) \neq 0$
proof −
have *nonempty-fss*: $fss \neq []$ **using** *nonempty-fs fss-is* **by** *auto*
have *deg-f*: $\forall f \in set\ (fs).\ degree\ f > 0$
using *factorize-polys-irreducible-monic*
apply (*auto*)
using *assms*(*1*) *irreducible-degree-field* **by** *blast*
then have *deg-fss*: $\forall f \in set\ (fss).\ degree\ f > 0$
using *fss-is* **by** *simp*
then have *fss-nonz*: $\forall f \in set\ (fss).\ f \neq 0$
by *auto*
have $fss \neq [] \longrightarrow ((\forall f \in set\ (fss).\ (degree\ f > 0 \wedge f \neq 0)) \longrightarrow degree\ (prod\text{-}list\ fss) > 0)$
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*: $a \neq 0$
assume *fnonz*: $\forall f{\in}set\ fss.\ 0 < degree\ f \wedge f \neq 0$
have *h*: $degree\ (a * prod\text{-}list\ fss) = degree\ a + degree\ (prod\text{-}list\ fss)$
using *degree-mult-eq*[**where** $p = a$, **where** $q = prod\text{-}list\ fss$] *anonz fnonz*
by *auto*
then show $0 < degree\ (a * prod\text{-}list\ fss)$
using *z-lt Cons.hyps* **by** *auto*
qed
qed
then have $degree\ (prod\text{-}list\ fss) > 0$
using *nonempty-fss deg-fss fss-nonz* **by** *auto*
then have *pderiv-nonzero*: $pderiv\ (prod\text{-}list\ fss) \neq 0$
by (*simp add*: *pderiv-eq-0-iff*)
have $(([:-(crb\ (prod\text{-}list\ fss)),1:]) * ([:(crb\ (prod\text{-}list\ fss)),1:])) \neq 0$
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 $(x1 < x2 \wedge poly\ q\ x1 = 0 \wedge poly\ q\ x2 = 0) \longrightarrow (\exists w.\ x1 < w \wedge w < x2 \wedge poly\ (pderiv\ q)\ w = 0)$

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* ∈ (*consistent-sign-vectors qs UNIV*)
assumes *p1p2*: *p1*≠*p2* ∧ *p1* < *length csa* ∧ *p2* < *length csa* ∧ *csa* ! *p1* = *0* ∧ *csa* ! *p2* = *0*
shows ¬ *algebraic-semidom-class.coprime* (*nth qs p1*) (*nth qs p2*)
proof −
have *isx*: ∃(*x*::*real*). *csa* = (*sign-vec qs x*)
using *csa-in* **unfolding** *consistent-sign-vectors-def* **by** *auto*
then obtain *x* **where** *havex*: *csa* = (*sign-vec qs x*) **by** *auto*
then have *expolys*: *poly* (*real-of-rat-poly* (*nth qs p1*)) *x* = *0* ∧ *poly* (*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 ¬ *coprime* (*real-of-rat-poly* (*nth qs p1*)) ((*real-of-rat-poly* (*nth qs p2*))::*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 (∀ *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-lt*: *x1* < *x2*
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*: $q \in set\ qs$ **by** *auto*
have *poss1*: $squash\ (rpoly\ q\ x1) = -1 \wedge squash\ (rpoly\ q\ x2) = 1 \longrightarrow (\exists\, r{>}x1.\ r \leq x2 \wedge (\exists\, q{\in}set\ qs.\ rpoly\ q\ r = 0))$
using *poly-IVT-pos*[*of x1 x2*] **using** *x1-lt* **unfolding** *squash-def* **apply** (*auto*)
using *q-prop* **by** *fastforce*
have *poss2*: $squash\ (rpoly\ q\ x1) = 1 \wedge squash\ (rpoly\ q\ x2) = -1 \longrightarrow (\exists\, r{>}x1.\ r \leq x2 \wedge (\exists\, q{\in}set\ qs.\ rpoly\ q\ r = 0))$
using *poly-IVT-neg*[*of x1 x2*] **using** *x1-lt* **unfolding** *squash-def* **apply** (*auto*)
using *q-prop* **by** *fastforce*
have *poss3*: $squash\ (rpoly\ q\ x2) = 0 \longrightarrow (\exists\, r{>}x1.\ r \leq x2 \wedge (\exists\, q{\in}set\ qs.\ rpoly\ q\ r = 0))$
using *x1-lt* **unfolding** *squash-def* **apply** (*auto*)
using *q-prop* **by** *blast*
have $(q \in set\ qs \wedge rpoly\ q\ x1 = 0) \longrightarrow \neg has\text{-}no\text{-}zeros\ (sign\text{-}vec\ qs\ x1)$
unfolding *has-no-zeros-def sign-vec-def*
by (*smt* (*verit*) *image-eqI list.set-map o-apply squash-def*)
have *not-poss4*: $squash\ (rpoly\ q\ x1) \neq 0$
using *hnz q-in* **unfolding** *squash-def*
using ‹$q \in set\ qs \wedge rpoly\ q\ x1 = 0 \longrightarrow \neg\ has\text{-}no\text{-}zeros\ (sign\text{-}vec\ qs\ x1)$› **by** *auto*
then show $\exists\, r{>}x1.\ r \leq x2 \wedge (\exists\, q{\in}set\ qs.\ rpoly\ q\ r = 0)$
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 $\forall\, x2 < x1.\ ((sign\text{-}vec\ qs\ x1) \neq (sign\text{-}vec\ qs\ x2)) \longrightarrow (\exists\, (r{::}real) < x1.\ (r \geq x2 \wedge (\exists\ \ q \in set(qs).\ rpoly\ q\ r = 0)))$
proof *clarsimp*
fix *x2*
assume *x1-gt*: $x2 < x1$
assume *diff-sign-vec*: $sign\text{-}vec\ qs\ x1 \neq sign\text{-}vec\ qs\ x2$
then have $\exists\, q \in set\ qs.\ squash\ (rpoly\ q\ x1) \neq squash\ (rpoly\ q\ x2)$
unfolding *sign-vec-def*
by *simp*
then obtain *q* **where** *q-prop*: $q \in set\ qs \wedge squash\ (rpoly\ q\ x1) \neq squash\ (rpoly\ q\ x2)$
by *auto*
then have *q-in*: $q \in set\ qs$ **by** *auto*
have *poss1*: $squash\ (rpoly\ q\ x1) = -1 \wedge squash\ (rpoly\ q\ x2) = 1 \longrightarrow (\exists\, r{<}x1.\ (r \geq x2 \wedge (\exists\ \ q \in set(qs).\ rpoly\ q\ r = 0)))$
using *poly-IVT-neg*[*of x2 x1*] **using** *x1-gt* **unfolding** *squash-def* **apply** (*auto*)
using *q-prop* **by** *fastforce*
have *poss2*: $squash\ (rpoly\ q\ x1) = 1 \wedge squash\ (rpoly\ q\ x2) = -1 \longrightarrow (\exists\, r{<}x1.\ (r \geq x2 \wedge (\exists\ \ q \in set(qs).\ rpoly\ q\ r = 0)))$

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 (∃ *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 *sorted-list-lemma*:
fixes *l*:: *real list*
fixes *a b*:: *real*
fixes *n*:: *nat*
assumes *a* < *b*
assumes (*n* + *1*) < *length l*
assumes *strict-sort*: *sorted-wrt* (<) *l*
assumes *lt-a*: (*l* ! *n*) < *a*
assumes *b-lt*: *b* < (*l* ! (*n*+*1*))
shows ¬(∃ (*x*::*real*). (*List.member l x* ∧ *a* ≤ *x* ∧ *x* ≤ *b*))
proof −
have *sorted-hyp-var*: ∀ *q1* < *length l*. ∀ *q2* < *length l*. (*q1* < *q2* ⟶
(*l* ! *q1*) < (*l* ! *q2*))
using *strict-sort* **by** (*auto simp*: *sorted-wrt-iff-nth-less*)
then have *sorted-hyp-var2*: ∀ *q1* < *length l*. ∀ *q2* < *length l*. ((*l* ! *q1*) < (*l* ! *q2*)) ⟶ *q1* < *q2*
using *linorder-neqE-nat*
by (*metis Groups.add-ac*(*2*) *add-mono-thms-linordered-field*(*5*) *less-irrefl*)
have (∃ (*x*::*real*). (*List.member l x* ∧ *a* ≤ *x* ∧ *x* ≤ *b*)) ⟹ *False*
proof −
assume (∃ (*x*::*real*). (*List.member l x* ∧ *a* ≤ *x* ∧ *x* ≤ *b*))
then obtain *x* **where** *x-prop*: *List.member l x* ∧ *a* ≤ *x* ∧ *x* ≤ *b* **by** *auto*
then have *l-prop*: *List.member l x* ∧ (*l* ! *n*) < *x* ∧ *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 ∃ *k*. *k* < *length l* ∧ *nth l k* = *x* **using** *l-prop*
by (*meson in-set-member index-of-lookup*(*1*) *index-of-lookup*(*2*))
then obtain *k* **where** *k-prop*: *k* < *length l* ∧ *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: ⋀q. q ∈ set qs ⟹ 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::real). (∃ q ∈ set(qs). (rpoly
q x = 0))}
  shows zer-list ≠ [] ⟶ ((x1 < (zer-list ! 0)) ∨ (x1 > (zer-list ! (length zer-list
− 1)) ∨
   (∃ n < (length zer-list − 1). x1 > (zer-list ! n) ∧ x1 < (zer-list ! (n+1)))))
proof −
  let ?zer-list = sorted-list-of-set {(x::real). (∃ q ∈ set(qs). (rpoly q x = 0))} :: real
list
  show ?thesis
  proof −
   have ((∀ q. (List.member qs q) ⟶ q ≠ 0) ∧ has-no-zeros (sign-vec qs x1)) ⟹
¬ List.member ?zer-list x1
    proof (induct qs)
     case Nil
     then show ?case  apply (auto)
      by (simp add: member-rec(2))
    next
     case (Cons a qs)
     then show ?case
     proof clarsimp
      assume imp: ((∀ q. List.member qs q ⟶ q ≠ 0) ∧
    has-no-zeros (sign-vec qs x1) ⟹
    ¬ List.member (sorted-list-of-set {x. ∃ q∈set qs. rpoly q x = 0})
       x1)
      assume nonz: ∀ q. List.member (a # qs) q ⟶ q ≠ 0
      assume hnz:  has-no-zeros (sign-vec (a # qs) x1)
      assume mem-list: List.member
    (sorted-list-of-set {x. rpoly a x = 0 ∨ (∃ q∈set qs. rpoly q x = 0)})
    x1
      have has-no-zeros (sign-vec (a # qs) x1) ⟹ has-no-zeros (sign-vec qs x1)
```

proof −
assume *hnz*: *has-no-zeros* (*sign-vec* (*a* # *qs*) *x1*)
have *same-vec*: *sign-vec* (*a* # *qs*) *x1* = ((*if rpoly a x1* > *0 then 1 else if rpoly a x1* = *0 then 0 else* −*1*) # *sign-vec* (*qs*) *x1*)
unfolding *sign-vec-def squash-def* **by** *auto*
have *has-no-zeros* ((*if rpoly a x1* > *0 then 1 else if rpoly a x1* = *0 then 0 else* −*1*) # *sign-vec* (*qs*) *x1*)
⟹ *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*: ¬ *List.member* (*sorted-list-of-set* {*x*. ∃ *q*∈*set qs*. *rpoly q x* = *0*}) *x1*
using *hnz nonz imp* **apply** (*auto*)
by (*simp add*: *member-rec*(*1*))
have ∀ *q* ∈*set qs*. *q* ≠ *0*
using *nonz* **using** *in-set-member* **apply** (*auto*) **by** *fastforce*
then have ∀ *q* ∈*set qs*. *finite* {*x*. *rpoly q x* = *0*}
by (*simp add*: *poly-roots-finite*)
then have *fin-set*: *finite* {*x*. ∃ *q*∈*set qs*. *rpoly q x* = *0*}
by *auto*
have *not-in*: *x1* ∉ {*x*. ∃ *q*∈*set qs*. *rpoly q x* = *0*} **using** *fin-set nmem set-sorted-list-of-set*
all-squarefree
apply (*auto*)
by (*simp add*: *List.member-def* ‹*finite* {*x*. ∃ *q*∈*set qs*. *rpoly q x* = *0*}›)
have *x1-in*: *x1* ∈ {*x*. *rpoly a x* = *0* ∨ (∃ *q*∈*set qs*. *rpoly q x* = *0*)}
using *mem-list sorted-list-of-set*
proof −
have *f1*: ∀ *r R*. ((*r*::*real*) ∈ *R* ∨ ¬ *List.member* (*sorted-list-of-set R*) *r*) ∨ *infinite R*
by (*metis in-set-member set-sorted-list-of-set*)
have *finite* {*r*. *rpoly a* (*r*::*real*) = *0*}
by (*metis* (*full-types*) *List.finite-set member-rec*(*1*) *nonz real-roots-of-rat-poly*(*1*))
then show *?thesis*
using *f1* ‹*finite* {*x*. ∃ *q*∈*set qs*. *rpoly q x* = *0*}› *mem-list* **by** *fastforce*
qed
have *rpoly a x1* ≠ *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
qed
then have *non-mem*: ¬ *List.member ?zer-list x1*
using *all-squarefree* **unfolding** *rsquarefree-def hnz* **apply** (*auto*)
using *hnz x1-prop*
by (*simp add*: *in-set-member*)
have *?zer-list* ≠ [] ⟹ ((*x1* ≥ (*?zer-list* ! *0*)) ∧ (*x1* ≤ (*?zer-list* ! (*length ?zer-list* − *1*))))

$\Longrightarrow$ ($\exists$ n < (*length ?zer-list* − *1*). *x1* > (*?zer-list* ! *n*) ∧ *x1* < (*?zer-list* ! (*n*+*1*)))
proof −
assume *nonempty*: *?zer-list* ≠ []
assume *x1-asm*: (*x1* ≥ (*?zer-list* ! *0*)) ∧ (*x1* ≤ (*?zer-list* ! (*length ?zer-list* − *1*)))
have *nm1*: *x1* ≠ *?zer-list* ! *0* **using** *non-mem*
using ‹*sorted-list-of-set* {*x*. ∃ *q*∈*set qs*. *rpoly q x* = *0*} ≠ []› *in-set-member*
by (*metis* (*no-types*, *lifting*) *in-set-conv-nth length-greater-0-conv*)
have *nm2*: *x1* ≠ *?zer-list* ! (*length ?zer-list* −*1*) **using** *non-mem*
by (*metis* (*no-types*, *lifting*) *One-nat-def* ‹*sorted-list-of-set* {*x*. ∃ *q*∈*set qs*. *rpoly q x* = *0*} ≠ []› *diff-Suc-less in-set-member length-greater-0-conv nth-mem*)
then have *x-asm-var*: *x1* > (*?zer-list* ! *0*) ∧ *x1* < *?zer-list* ! (*length ?zer-list* −*1*)
using *x1-asm nm1 nm2* **by** *auto*
have (∀ *n*. (*n* < (*length ?zer-list* − *1*) ∧ *x1* ≥ (*?zer-list* ! *n*) ⟶ *x1* ≥ (*?zer-list* ! (*n*+*1*)))) $\Longrightarrow$ *False*
proof −
assume *assump*: (∀ *n*. (*n* < (*length ?zer-list* − *1*) ∧ *x1* ≥ (*?zer-list* ! *n*) ⟶ *x1* ≥ (*?zer-list* ! (*n*+*1*))))
have *zer-case*: *x1* ≥ *?zer-list* ! *0* **using** *x-asm-var* **by** *auto*
have *all-n*: ⋀ *n*. (*n* < (*length ?zer-list* − *1*) ⟶ *x1* ≥ *?zer-list* ! *n*)
proof −
fix *n*
show *n-lt*: (*n* < (*length ?zer-list* − *1*) ⟶ *x1* ≥ *?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*) ≤ *length ?zer-list* −*1*
using *diff-le-mono2 one-le-numeral* **by** *blast*
have *x1* ≥ *?zer-list* ! (*length ?zer-list* − *1*)
proof −
have *h1*: *length ?zer-list* = *1* ⟶ *x1* ≥ *?zer-list* ! (*length ?zer-list* − *1*)
using *assump zer-case* **by** *auto*
have *h2*: *length ?zer-list* > *1* ⟶ *x1* ≥ *?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*
by (*smt* (*verit*) *One-nat-def Suc-pred nonempty in-set-member length-greater-0-conv less-SucI non-mem nth-mem*)
qed
then have *h1*: (*?zer-list* ≠ [] ∧ (*x1* ≥ (*?zer-list* ! *0*)) ∧ (*x1* ≤ (*?zer-list* ! (*length ?zer-list* − *1*))) ⟹
(∃ *n* < (*length ?zer-list* − *1*). *x1* > (*?zer-list* ! *n*) ∧ *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*: ⋀*q*. *q* ∈ *set qs* ⟹ *rsquarefree q*
assumes *ex-x1*: *sga* = *sign-vec qs x1*
assumes *hnz*: *has-no-zeros sga*
shows (∃ *w* ∈ (*roots-of-coprime-r* (*cast-rat-list qs*)). *sga* = (*sign-vec qs w*))
proof −
obtain *x1* **where** *x1-prop*: *sga* = (*sign-vec qs x1*) **using** *ex-x1* **by** *auto*
let *?zer-list* = *sorted-list-of-set* {(*x*::*real*). (∃ *q* ∈ *set*(*qs*). (*rpoly q x* = *0*))} :: *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*: ∀ *q* < *length ?zer-list*. (*q* + *1* < *length ?zer-list*) ⟶
(*?zer-list* ! *q*) < (*?zer-list* ! (*q* +*1*))
using *strict-sorted-h* **by** (*auto simp*: *sorted-wrt-iff-nth-less*)
then have *sorted-hyp-var*: ∀ *q1* < *length ?zer-list*. ∀ *q2* < *length ?zer-list*. (*q1* < *q2* ⟶
(*?zer-list* ! *q1*) < (*?zer-list* ! *q2*))
using *sorted-wrt-iff-nth-less strict-sorted-h* **by** *blast*
then have *sorted-hyp-var2*: ∀ *q1* < *length ?zer-list*. ((*?zer-list* ! *q1*)::*real*) ≤ (*?zer-list* ! (*length ?zer-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*: ∀ *q* ∈*set qs*. *q* ≠ *0*
using *all-squarefree* **unfolding** *rsquarefree-def* **using** *in-set-member* **by** *auto*
then have ∀ *q* ∈*set qs*. *finite* {*x*. *rpoly q x* = *0*}
by (*simp add*: *poly-roots-finite*)
then have *fin-set*: *finite* {*x*. ∃ *q*∈*set qs*. *rpoly q x* = *0*}
by *auto*
have *x1-and-roots*: *?zer-list* ≠ [] ⟶ ((*x1* < (*?zer-list* ! *0*)) ∨ (*x1* > (*?zer-list* ! (*length ?zer-list* − *1*)) ∨

$(\exists\ n < (\mathit{length}\ \mathit{?zer\text{-}list} - 1).\ \mathit{x1} > (\mathit{?zer\text{-}list}\ !\ n) \wedge \mathit{x1} < (\mathit{?zer\text{-}list}\ !\ (n{+}1)))))$

using *roots-of-coprime-r-location-property x1-prop assms* **by** *auto*
have *x2gt*: $\forall\, \mathit{x2}{>}\mathit{x1}.\ \mathit{sign\text{-}vec\ qs\ x1} \neq \mathit{sign\text{-}vec\ qs\ x2} \longrightarrow (\exists\, r{>}\mathit{x1}.\ r \leq \mathit{x2} \wedge (\exists\, q{\in}\mathit{set\ qs}.\ \mathit{rpoly\ q\ r} = 0))$
using *hnz x1-prop zero-above*[*of qs x1*] **by** *auto*
have *x2lt*: $\forall\, \mathit{x2}{<}\mathit{x1}.\ \mathit{sign\text{-}vec\ qs\ x1} \neq \mathit{sign\text{-}vec\ qs\ x2} \longrightarrow (\exists\, r{<}\mathit{x1}.\ \mathit{x2} \leq r \wedge (\exists\, q{\in}\mathit{set\ qs}.\ \mathit{rpoly\ q\ r} = 0))$
using *hnz x1-prop zero-below*[*of qs x1*] **by** (*auto*)
have *triv-neg-inf-h*: $\mathit{?zer\text{-}list} = [] \Longrightarrow \mathit{sga} = (\mathit{sign\text{-}vec\ qs}\ (-\mathit{crb}\ (\mathit{prod\text{-}list}\ (\mathit{cast\text{-}rat\text{-}list\ qs}))))$
proof −
assume *empty-zer*: $(\mathit{?zer\text{-}list}{::}\ \mathit{real\ list}) = []$
let *?zer-set* = $\{x.\ \exists\, q{\in}\mathit{set\ qs}.\ \mathit{rpoly\ q\ x} = 0\}{::}\ \mathit{real\ set}$
have *fin-zer*: *finite ?zer-set* **using** *fin-set* **by** *auto*
have *finite ?zer-set* $\Longrightarrow$ $(\mathit{sorted\text{-}list\text{-}of\text{-}set\ ?zer\text{-}set} = []) = (\mathit{?zer\text{-}set} = \{\})$
using *fin-zer sorted-list-of-set-eq-Nil-iff*[**where** A = *?zer-set*] **by** *auto*
then have $(\mathit{sorted\text{-}list\text{-}of\text{-}set\ ?zer\text{-}set} = []) = (\mathit{?zer\text{-}set} = \{\})$
using *fin-zer* **by** *auto*
then have *nozers*: *?zer-set* = {}
using *empty-zer* **by** *auto*
then have $\neg(\exists\, (r{::}\mathit{real}).\ (\exists\, (q{::}\mathit{rat\ poly}){\in}\mathit{set\ qs}.\ \mathit{rpoly\ q\ r} = 0))$
using *nozers* **by** *auto*
then have $\forall\, y.\ \mathit{sign\text{-}vec\ qs\ x1} = \mathit{sign\text{-}vec\ qs\ y}$
proof −
fix *y*
have *gt-prop*: $\mathit{x1} > y \longrightarrow \mathit{sign\text{-}vec\ qs\ x1} = \mathit{sign\text{-}vec\ qs\ y}$
using *hnz x1-prop zero-below*[*of qs x1*] **apply** (*auto*)
using ‹$\nexists\, r.\ \exists\, q{\in}\mathit{set\ qs}.\ \mathit{rpoly\ q\ r} = 0$› **by** *blast*
have *lt-prop*: $\mathit{x1} < y \longrightarrow \mathit{sign\text{-}vec\ qs\ x1} = \mathit{sign\text{-}vec\ qs\ y}$
using *zero-above*[*of qs x1*] **apply** (*auto*)
using ‹$\nexists\, r.\ \exists\, q{\in}\mathit{set\ qs}.\ \mathit{rpoly\ q\ r} = 0$› *x2gt* **by** *blast*
show *?thesis* **using** *gt-prop lt-prop* **apply** (*auto*)
apply (*metis* ‹$\nexists\, r.\ \exists\, q{\in}\mathit{set\ qs}.\ \mathit{rpoly\ q\ r} = 0$› *linorder-neqE-linordered-idom x2gt x2lt*)
using *x2gt x2lt* **apply** (*auto*)
apply (*metis* ‹$\nexists\, r.\ \exists\, q{\in}\mathit{set\ qs}.\ \mathit{rpoly\ q\ r} = 0$› *linorder-neqE-linordered-idom*)
apply (*metis* ‹$\nexists\, r.\ \exists\, q{\in}\mathit{set\ qs}.\ \mathit{rpoly\ q\ r} = 0$› *linorder-neqE-linordered-idom*)
by (*metis* ‹$\nexists\, r.\ \exists\, q{\in}\mathit{set\ qs}.\ \mathit{rpoly\ q\ r} = 0$› *linorder-neqE-linordered-idom*)
qed
then show *?thesis*
by (*simp add*: *x1-prop*)
qed
have *neg-inf-h*: $\mathit{?zer\text{-}list} \neq [] \Longrightarrow (\mathit{x1} < (\mathit{?zer\text{-}list}\ !\ 0) \Longrightarrow \mathit{sga} = (\mathit{sign\text{-}vec\ qs}\ (-\mathit{crb}\ (\mathit{prod\text{-}list}\ (\mathit{cast\text{-}rat\text{-}list\ qs})))))$
proof −
let *?neg-crb* = $-\mathit{crb}\ (\mathit{prod\text{-}list}\ (\mathit{cast\text{-}rat\text{-}list\ qs}))$
assume *len-nontriv*: $\mathit{?zer\text{-}list} \neq []$
assume *x1-lt*: $\mathit{x1} < \mathit{?zer\text{-}list}\ !\ 0$

have *r-gt*: $\forall r.\ (\exists q{\in}set\ qs.\ rpoly\ q\ r = 0) \longrightarrow r \geq (?zer\text{-}list\ !\ 0)$
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* $\{x.\ \exists q{\in}set\ qs.\ rpoly\ q\ x = 0\}$) *r*
using *in-set-member set-sorted-list-of-set fin-set* **by** (*smt* (*verit*, *best*))
then show *sorted-list-of-set* $\{x.\ \exists q{\in}set\ qs.\ rpoly\ q\ x = 0\}$! *0* $\leq r$
using *sorted-hyp-var*
by (*metis* (*no-types*, *lifting*) *gr-implies-not0 in-set-conv-nth in-set-member not-less sorted-iff-nth-mono sorted-sorted-list-of-set*)
qed
have *prod-zer*: $\forall x.\ (\exists q{\in}set\ qs.\ rpoly\ q\ x = 0) \longrightarrow (poly\ (prod\text{-}list\ (cast\text{-}rat\text{-}list\ qs))\ x) = 0$
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* $\neq[] \longrightarrow$ *List.member* (*sorted-list-of-set* $\{x.\ \exists q{\in}set\ qs.\ rpoly\ q\ x = 0\}$) (*?zer-list* ! *0*)
using *nth-Cons-0* **apply** (*auto*)
by (*meson gr0I in-set-member length-0-conv nth-mem*)
then have *?zer-list* $\neq[] \longrightarrow$ (*?zer-list* ! *0*) $\in \{x.\ \exists q{\in}set\ qs.\ rpoly\ q\ x = 0\}$
using *in-set-member*[**where** *x* = (*sorted-list-of-set* $\{x.\ \exists q{\in}set\ qs.\ rpoly\ q\ x = 0\}$! *0*),
where *xs* = *sorted-list-of-set* $\{x.\ \exists q{\in}set\ qs.\ rpoly\ q\ x = 0\}$]
set-sorted-list-of-set fin-set
by *blast*
then have *?zer-list* $\neq[] \longrightarrow (\exists q{\in}set\ qs.\ rpoly\ q\ (?zer\text{-}list\ !\ 0) = 0)$
by *blast*
then have *poly-zer*: *?zer-list* $\neq[] \longrightarrow$ (*poly* (*prod-list* (*cast-rat-list qs*)) (*?zer-list* ! *0*)) = *0*
using *prod-zer* **by** *auto*
have $\forall q.$ *List.member* (*cast-rat-list qs*) $q \longrightarrow q \neq 0$ **using** *nonz-q*
unfolding *cast-rat-list-def* **using** *in-set-member imageE image-set map-poly-zero of-rat-eq-0-iff*
by (*smt* (*verit*, *best*))
then have (*prod-list* (*cast-rat-list qs*)) $\neq 0$
using *prod-list-zero-iff in-set-member* **by** *fastforce*
then have *crb-lt*: *?zer-list* $\neq[] \longrightarrow$ *?neg-crb* $<$ *?zer-list* ! *0*
using *crb-lem-neg*[**where** *p* = (*prod-list* (*cast-rat-list qs*)), **where** *x* = *sorted-list-of-set* $\{x.\ \exists q{\in}set\ qs.\ rpoly\ q\ x = 0\}$! *0*] **apply** (*auto*)
using *poly-zer*
by *blast*
have *crb-gt-x1*: *?zer-list* $\neq[] \longrightarrow$ (*?neg-crb* $>$ *x1* $\longrightarrow$ (*sga* $\neq$ (*sign-vec qs ?neg-crb*)) $\longrightarrow$ $(\exists r{>}x1.\ r \leq$ *?neg-crb* $\wedge$ $(\exists q{\in}set\ qs.\ rpoly\ q\ r = 0)))$
using *x2gt crb-lt r-gt x1-prop* **by** *fastforce*
have *crb-lt-x1*: *?neg-crb* $<$ *x1* $\longrightarrow$ (*sga* $\neq$ (*sign-vec qs ?neg-crb*)) $\longrightarrow$ $(\exists r{<}x1.$

?neg-crb ≤ *r* ∧ (∃ *q*∈*set qs*. *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* ≠ [] ⟹ (*x1* > (*?zer-list* ! (*length ?zer-list* − *1*)) ⟹ *sga* = (*sign-vec qs* (*crb* (*prod-list* (*cast-rat-list qs*)))))
proof −
let *?pos-crb* = *crb* (*prod-list* (*cast-rat-list qs*))
assume *len-nontriv*: *?zer-list* ≠[]
assume *x1-lt*: *x1* > *?zer-list* ! (*length ?zer-list* − *1*)
have *r-gt*: ⋀*r*. (∃ *q*∈*set qs*. *rpoly q r* = *0*) ⟹ *r* ≤ (*?zer-list* ! (*length ?zer-list* − *1*))
proof −
fix *r*:: *real*
assume *q-in*: (∃ *q*∈*set qs*. *rpoly q r* = *0*)
then obtain *q*::*rat poly* **where** *q-prop*: *q* ∈ *set qs* ∧ *rpoly q r* = *0* **by** *auto*
then have *r* ∈ {*x*. ∃ *q*∈*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*
using *in-set-member set-sorted-list-of-set fin-set* **by** (*smt* (*verit*))
then have ∃ *n* < (*length ?zer-list*). *r* = *?zer-list* ! *n*
by (*metis* (*no-types*, *lifting*) *in-set-conv-nth in-set-member*)
then obtain *n* **where** *n-prop*: *n* < *length ?zer-list* ∧ *r* = *?zer-list* ! *n*
by *auto*
then show *r* ≤ (*?zer-list* ! (*length ?zer-list* − *1*))
proof −
have ∀ *q1*. *q1* < *length ?zer-list* ⟶ (*?zer-list* ! *q1*) ≤ (*?zer-list* ! (*length ?zer-list* − *1*:: *nat*))
using *sorted-hyp-var2* **by** *auto*
then have (*?zer-list* ! *n*) ≤ (*?zer-list* ! (*length ?zer-list* − *1*))
using *n-prop* **by** *auto*
then show *?thesis* **using** *n-prop* **by** *auto*
qed
qed
have *prod-zer*: ∀ *x*. (∃ *q*∈*set qs*. *rpoly q x* = *0*) ⟶ (*poly* (*prod-list* (*cast-rat-list qs*)) *x*) = *0*
using *prod-list-zero-iff*[**where** *xs* = (*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 *?zer-list* ≠[] ⟶ *List.member* (*sorted-list-of-set* {*x*. ∃ *q*∈*set qs*. *rpoly q x* = *0*})
(*?zer-list* ! (*length ?zer-list* −*1*))
using *nth-Cons-0* **apply** (*auto*)
by (*metis* (*no-types*, *lifting*) *diff-less in-set-conv-nth in-set-member length-greater-0-conv length-sorted-list-of-set zero-less-Suc*)
then have *?zer-list* ≠[] ⟶ (*?zer-list* ! (*length ?zer-list* −*1*))
∈ {*x*. ∃ *q*∈*set qs*. *rpoly q x* = *0*}

using *in-set-member*[**where** $x = (sorted\text{-}list\text{-}of\text{-}set\ \{x.\ \exists\, q{\in}set\ qs.\ rpoly\ q\ x = 0\}\ !\ (length\ ?zer\text{-}list - 1))$,
where $xs = sorted\text{-}list\text{-}of\text{-}set\ \{x.\ \exists\, q{\in}set\ qs.\ rpoly\ q\ x = 0\}$]
set-sorted-list-of-set fin-set
by *blast*
then have $?zer\text{-}list \neq [] \longrightarrow (\exists\, q{\in}set\ qs.\ rpoly\ q\ (?zer\text{-}list\ !\ (length\ ?zer\text{-}list - 1)) = 0)$
by *blast*
then have *poly-zer*: $?zer\text{-}list \neq [] \longrightarrow (poly\ (prod\text{-}list\ (cast\text{-}rat\text{-}list\ qs))\ (?zer\text{-}list\ !\ (length\ ?zer\text{-}list - 1))) = 0$
using *prod-zer* **by** *auto*
have $\forall\, q.\ List.member\ (cast\text{-}rat\text{-}list\ qs)\ q \longrightarrow q \neq 0$ **using** *nonz-q*
unfolding *cast-rat-list-def* **using** *in-set-member imageE image-set map-poly-zero of-rat-eq-0-iff*
by (*smt* (*verit*))
then have $(prod\text{-}list\ (cast\text{-}rat\text{-}list\ qs)) \neq 0$
using *prod-list-zero-iff in-set-member* **apply** (*auto*)
by *fastforce*
then have *crb-lt*: $?zer\text{-}list \neq [] \longrightarrow ?pos\text{-}crb > ?zer\text{-}list\ !\ (length\ ?zer\text{-}list - 1)$
using *crb-lem-pos*[**where** $p = (prod\text{-}list\ (cast\text{-}rat\text{-}list\ qs))$, **where** $x = sorted\text{-}list\text{-}of\text{-}set\ \{x.\ \exists\, q{\in}set\ qs.\ rpoly\ q\ x = 0\}\ !\ (length\ ?zer\text{-}list - 1)$] **apply** (*auto*)
using *poly-zer*
by *simp*
have *crb-gt-x1*: $?zer\text{-}list \neq [] \longrightarrow ((?pos\text{-}crb{::}real) > (x1{::}real) \longrightarrow (sga \neq (sign\text{-}vec\ (qs{::}rat\ poly\ list)\ (?pos\text{-}crb{::}real))) \longrightarrow (\exists\, (r{::}real){<}x1.\ r \geq ?pos\text{-}crb \wedge (\exists\, (q{::}rat\ poly){\in}set\ qs.\ rpoly\ q\ r = 0)))$
using *x2gt* **apply** (*auto*)
using *crb-lt r-gt x1-prop*
using *x1-lt* **by** *fastforce*
have *crb-lt-x1*: $?pos\text{-}crb < x1 \longrightarrow (sga \neq (sign\text{-}vec\ qs\ ?pos\text{-}crb)) \longrightarrow (\exists\, r{>}x1.\ ?pos\text{-}crb \geq r \wedge (\exists\, q{\in}set\ qs.\ poly\ (real\text{-}of\text{-}rat\text{-}poly\ q)\ r = 0))$
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*: $(\exists\ n < (length\ ?zer\text{-}list - 1).\ x1 > (?zer\text{-}list\ !\ n) \wedge x1 < (?zer\text{-}list\ !\ (n{+}1))) \Longrightarrow (\exists\, w \in (roots\text{-}of\text{-}coprime\text{-}r\ (cast\text{-}rat\text{-}list\ qs)).\ sga = (sign\text{-}vec\ qs\ w))$
proof −
assume $(\exists\ n < (length\ ?zer\text{-}list - 1).\ x1 > (?zer\text{-}list\ !\ n) \wedge x1 < (?zer\text{-}list\ !\ (n{+}1)))$
then obtain n **where** *n-prop*: $n < (length\ ?zer\text{-}list - 1) \wedge x1 > (?zer\text{-}list\ !\ n) \wedge x1 < (?zer\text{-}list\ !\ (n{+}1))$
by *auto*
have $\forall\, q1\ q2.\ (q1 \neq q2 \wedge (List.member\ (cast\text{-}rat\text{-}list\ qs)\ q1) \wedge (List.member$

(cast-rat-list qs) q2))⟶ coprime q1 q2
using *pairwise-rel-prime coprime-rat-poly-iff-coprimereal-poly*
unfolding *pairwise-coprime-list-def*
by (*smt* (*verit*) *cast-rat-list-def imageE image-set in-set-conv-nth in-set-member*)
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 (∃ *q*. (*?zer-list* ! *n*) < *q* ∧ *q* < (*?zer-list* ! (*n*+*1*)) ∧ *poly* (*coprime-r* (*cast-rat-list qs*)) *q* = *0*)
using *all-prop n-prop2* **by** *auto*
then have ∃ *w* ∈ (*roots-of-coprime-r* (*cast-rat-list qs*)). (*?zer-list* ! *n*) < *w* ∧ *w* < (*?zer-list* ! (*n*+*1*))
apply (*auto*)
using *roots-of-coprime-r-def* **by** *auto*
then obtain *w* **where** *w-prop*: *w* ∈ (*roots-of-coprime-r* (*cast-rat-list qs*)) ∧ (*?zer-list* ! *n*) < *w* ∧ *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* ⟶ ¬(∃ (*x*::*real*). (*List.member ?zer-list x* ∧ *x1* ≤ *x* ∧ *x* ≤ *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* ⟹ ¬(∃ *r*>*x1*. *r* ≤ *w* ∧ (∃ *q*∈*set qs*. *rpoly q r* = *0*))
proof −
assume *w* > *x1*
then have *nex*: ¬(∃ (*x*::*real*). (*List.member ?zer-list x* ∧ *x1* ≤ *x* ∧ *x* ≤ *w*))
using *helper* **by** *auto*
have (∃ *r*>*x1*. *r* ≤ *w* ∧ (∃ *q*∈*set qs*. *rpoly q r* = *0*)) ⟹ *False*
proof −
assume (∃ *r*>*x1*. *r* ≤ *w* ∧ (∃ *q*∈*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 (*smt* (*verit*, *best*) *fin-set in-set-member mem-Collect-eq set-sorted-list-of-set*)
then show *?thesis* **using** *nex r-prop*
by *blast*
qed
then show *?thesis* **by** *auto*
qed
have *helper2*: $w < x1 \longrightarrow \neg(\exists\, (x{::}real).\ (List.member\ ?zer\text{-}list\ x \wedge w \leq x \wedge x \leq x1))$
using *n-lt2 n-prop sorted-list-lemma strict-sorted-h w-prop* **by** *blast*
have *no-root2*: $w < x1 \Longrightarrow \neg(\exists\, r{<}x1.\ w \leq r \wedge (\exists\, q{\in}set\ qs.\ rpoly\ q\ r = 0))$
proof −
assume $w < x1$
then have *nex*: $\neg(\exists\, (x{::}real).\ (List.member\ ?zer\text{-}list\ x \wedge\ \ w \leq x \wedge x \leq x1))$
using *helper2* **by** *auto*
have $(\exists\, r{<}x1.\ w \leq r \wedge (\exists\, q{\in}set\ qs.\ rpoly\ q\ r = 0)) \Longrightarrow False$
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 (*smt* (*verit*, *best*) *fin-set in-set-member mem-Collect-eq set-sorted-list-of-set*)

then show *?thesis* **using** *nex r-prop*
by *blast*
qed
then show *?thesis* **by** *auto*
qed
then have *w-gt*: $w > x1 \longrightarrow (sign\text{-}vec\ qs\ x1) = (sign\text{-}vec\ qs\ w)$
using *no-root1 n-prop x2gt* **by** *auto*
have *w-lt*: $w < x1 \longrightarrow (sign\text{-}vec\ qs\ x1)\ = (sign\text{-}vec\ qs\ w)$
using *no-root2 n-prop x2lt* **by** *auto*
then have $sga = (sign\text{-}vec\ qs\ w)$ **using** *w-gt w-lt x1-prop* **by** *auto*
then show $(\exists\, w \in (roots\text{-}of\text{-}coprime\text{-}r\ (cast\text{-}rat\text{-}list\ qs)).\ sga = (sign\text{-}vec\ qs\ w))$
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 $(csa \in (consistent\text{-}sign\text{-}vectors\ fs\ UNIV) \wedge has\text{-}no\text{-}zeros\ csa) \longleftrightarrow$

```
  csa ∈ 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 ≠ 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: (⋀q. q ∈ set ?new-l ⟹
      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 ∈ set (find-consistent-signs-at-roots ?copr ?new-l)
      ⟷ csa ∈ set(characterize-consistent-signs-at-roots ?copr ?new-l) by auto
  have forward: (csa ∈ (consistent-sign-vectors fs UNIV) ∧ 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: ⋀q. q ∈ set fs ⟹ 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∈roots-of-coprime-r (cast-rat-list fs) csa = 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 ∈ 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* ((λ*x*. *if* 0 < *x* *then* 1 *else* *if* *x* < 0 *then* − 1 *else* 0) ∘ (λ*p*. *rpoly* *p* *w*)) *fs* =
map ((λ*x*. *if* 0 < *x* *then* 1 *else* − 1) ∘ (λ*p*. *rpoly* *p* *w*)) *fs*
proof −
have ¬(∃ *x* ∈ *set* *fs*. *rpoly* *x* *w* = 0)
proof *clarsimp*
fix *x*
assume *x-in*: *x* ∈ *set* *fs*
assume *x-zer*: *rpoly* *x* *w* = 0
then have ∃ *k* < *length* *fs*. *nth* *fs* *k* = *x*
using *x-in*
by (*simp* *add*: *in-set-conv-nth*)
then obtain *k* **where** *k-prop*: *k* < *length* *fs* ∧ *fs* ! *k* = *x*
by *auto*
then have (*sign-vec* *fs* *w*) ! *k* = 0 **using** *x-zer* **apply** (*auto*)
unfolding *sign-vec-def* *squash-def* **by** *auto*
then have ¬ (*has-no-zeros* (*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* ((λ*x*. *if* 0 < *x* *then* 1 *else* *if* *x* < 0 *then* − 1 *else* 0) ∘ (λ*p*. *rpoly* *p* *w*)) *fs* =
map (λ*q*. *if* 0 < *poly* *q* *w* *then* 1 *else* − 1)
(*cast-rat-list* *fs*)
unfolding *cast-rat-list-def* **by** *auto*
then have *csa* = *map* (λ*q*. *if* 0 < *poly* *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* *pairwise-cp* *set-fsga* **by** *auto*
have ∀ (*x*::*real*). ∀ (*l*::*real* *poly* *list*). *rec-list* *True* (λ*h* *T*. *If* (*h* = 0) *False*)
(*map* (λ*q*. *if* 0 < *poly* *q* *x* *then* (1::*rat*) *else* (−1::*rat*))
l)

```
    proof clarsimp
      fix x::real
      fix l::real poly list
      show  rec-list True (λh T. If (h = 0) False)
            (map (λq. if 0 < poly 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 < poly 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 ∃ w ∈ set(characterize-root-list-p ?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 ∈ set (characterize-root-list-p ?copr) ∧ 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: ∀ p ∈ set(?new-l). poly p w ≠ 0
      using w-prop coprime-r-coprime-prop apply (auto)
      by (meson coprime-poly-0 in-set-member 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 ∈ (consistent-sign-vectors fs UNIV) ∧ has-no-zeros csa)
    ⟷ csa ∈ 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 ⟶ range a = range b*
by *simp*

lemma *removeAt-distinct*:
shows *distinct fss ⟹ distinct (removeAt i fss)*
unfolding *removeAt-def*
by (*simp add*: *set-take-disj-set-drop-if-distinct*)

lemma *consistent-signs-atw*:
assumes *⋀p. p ∈ set fs ⟹ poly p x ≠ 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 ≠ 0*
assumes *copr*: *⋀q. q ∈ set fs ⟹ coprime p q*
shows *set (characterize-consistent-signs-at-roots p fs) =*
consistent-sign-vectors-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 ≤ length qs*
shows *insertAt i (squash (poly p x)) (signs-at qs x) =*
signs-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 ≤ 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-at qs x*) = *length qs*
unfolding *signs-at-def* **by** *auto*

lemma *find-csa-at-index*:
assumes *i* < *length fss*
assumes ⋀*p q*. *p* ∈ *set fss* ⟹ *q* ∈ *set fss* ⟹ *p* ≠ *q* ⟹ *coprime p q*
assumes ⋀*p*. *p* ∈ *set fss* ⟹ *p* ≠ *0*
assumes *distinct fss*
shows
set (*map* (λ*v*. *insertAt i 0 v*) (*find-consistent-signs-at-roots* (*fss* ! *i*) (*removeAt i fss*))) =
{*v* ∈ *consistent-sign-vectors-r fss UNIV*. *v* ! *i* = *0*}
proof −
from *removeAt-distinct*[*OF assms*(*4*)]
have *neq*: *fss* ! *i* ≠ *0* **using** *assms* **by** *simp*
have *cop*: ⋀*q*. *q* ∈ *set* (*removeAt i fss*) ⟹ *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* < *length ls*
assumes *x* ∈ *set* (*f* (*ls* ! *i*))
shows *x* ∈ *set* (*concat* (*map f ls*))
using *assms* **by** *auto*

lemma *find-csas-lemma-onezero-gen*:
fixes *qs*:: *rat poly list*
assumes *fs*: *factorize-polys qs* = (*fs*,*data*)
assumes *fss*: *fss* = *cast-rat-list fs*
shows (*csa* ∈ (*consistent-sign-vectors-r fss UNIV*) ∧ ¬(*has-no-zeros csa*))
⟷ *csa* ∈ *set* (*find-sgas-aux fss*)
proof −
have *a*:(⋀*p q*. *p* ∈ *set fss* ⟹
q ∈ *set fss* ⟹
p ≠ *q* ⟹ *coprime p*
q)
using *cast-rat-list-def factorize-polys-map-coprime fs fss* **by** *blast*
have *b*:(⋀*p*. *p* ∈ *set fss* ⟹ *p* ≠ *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*:*distinct fss*
using *factorize-polys-map-distinct*[*OF assms*(*1*)] *assms*(*2*) **unfolding** *cast-rat-list-def*
by *auto*
have *forwards*: *csa* ∈ (*consistent-sign-vectors-r fss UNIV*) ⟹
¬ (*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* (λ*v*. *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* (λ*i*. *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* (λ*v*. *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* (λ*v*. *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 add*: *List.member-def*)

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*)))) = *consistent-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*:∀ *q*. (*List.member* (*fst*(*factorize-polys qs*)) *q*) ⟶ (*rsquarefree q*)
using *factorize-polys-square-free*
by (*metis in-set-member prod.collapse square-free-rsquarefree*)
have *allnonzero*: ∀ *q*. (*List.member ?ftrs q*) ⟶ *q* ≠ *0*
using *all-squarefree* **apply** (*auto*)
using *rsquarefree-def* **by** *blast*
have *h1*: ∀ *csa*. (*csa* ∈ (*consistent-sign-vectors ?ftrs UNIV*) ∧ ¬ (*has-no-zeros csa*))
⟷ *csa* ∈ *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*: ∀ *csa*. (*csa* ∈ (*consistent-sign-vectors ?ftrs UNIV*) ∧ *has-no-zeros csa*)
⟷
List.member (*find-consistent-signs-at-roots* (*coprime-r* (*cast-rat-list ?ftrs*)) (*cast-rat-list ?ftrs*)) *csa*
using *lenh find-csas-lemma-nozeros pairwise-rel-prime allnonzero*
by (*metis in-set-member length-greater-0-conv prod.collapse*)

have *h3*: ∀ *csa*. *List.member* (*find-sgas* (*map* (*map-poly of-rat*) *?ftrs*)) *csa* ⟷
((*List.member* (*find-sgas-aux* (*cast-rat-list ?ftrs*)) *csa*) ∨ (*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*: ∀ *csa*. *List.member* (*find-sgas* (*map* (*map-poly of-rat*) *?ftrs*)) *csa* ⟷
((*csa* ∈ (*consistent-sign-vectors ?ftrs UNIV*) ∧ *has-no-zeros csa*) ∨ (*csa* ∈ (*consistent-sign-vectors ?ftrs UNIV*) ∧ ¬ (*has-no-zeros csa*)))
using *h1 h2 h3* **apply** (*auto*) **apply** (*simp add*: *in-set-member*) **by** (*simp add*: *in-set-member*)
have *h5*: ∀ *csa*. (*csa* ∈ (*consistent-sign-vectors ?ftrs UNIV*) ∧ *has-no-zeros csa*) ∨ (*csa* ∈ (*consistent-sign-vectors ?ftrs UNIV*) ∧ ¬ (*has-no-zeros csa*))
⟷ *csa* ∈ (*consistent-sign-vectors ?ftrs UNIV*)
by *auto*
then have ∀ *csa*. *List.member* (*find-sgas* (*map* (*map-poly of-rat*) *?ftrs*)) *csa* ⟷ *csa* ∈ (*consistent-sign-vectors ?ftrs UNIV*)
using *h4*
by *blast*
then show *?thesis* **using** *in-set-member* **apply** (*auto*)
apply (*simp add*: *in-set-member*)
by (*simp add*: *in-set-member*)
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*: ∀ *x*. *undo-factorize-polys data* (*sign-vec ftrs x*) = *sign-vec ps x*
using *factorize-polys-undo-factorize-polys*
by (*simp add*: *assms*(*1*))
have *h1*: ∀ *x*. *sign-vec ps x* ∈ *set* (*map* (*undo-factorize-polys data*) *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* ⊆ *set* (*map* (*undo-factorize-polys data*) *sgas*)
unfolding *consistent-sign-vectors-def* **by** *auto*
have *supset-h*: *consistent-sign-vectors ps UNIV* ⊇ *set* (*map* (*undo-factorize-polys data*) *sgas*)
proof −
have ∀ *ele*. *ele* ∈ *set* (*map* (*undo-factorize-polys data*) *sgas*) ⟶
(∃ *n* < *length sgas*. *ele* = (*undo-factorize-polys data* (*nth sgas n*)))
using *index-of-lookup*(*1*) *index-of-lookup*(*2*) **by** *fastforce*
then have ∀ *ele*. *ele* ∈ *set* (*map* (*undo-factorize-polys data*) *sgas*) ⟶
(∃ *x*. *ele* = (*undo-factorize-polys data* (*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) = {(map (λx. if poly x 0 < 0 then −1
else if poly x 0 = 0 then 0 else 1) 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) = [] ⟹ (∀ q ∈set(qs). 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 (∀ *x*::*real*. *lookup-sem fml-struct* (*map* (λ*p*. *rpoly p x*) *polys*)) ⟷
(*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 (∃ *x*::*real*. *lookup-sem fml-struct* (*map* (λ*p*. *rpoly p x*) *polys*)) ⟷
(*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*::*real*. *fml-sem fml x*) $\longleftrightarrow$ (*decide-universal fml*)
$\exists$ *x*::*real*. *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* (λ*x*. *x*^*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* (λ(*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*])]))

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* ⇒ (*nat list*∗*nat list*) *list* ⇒ (*nat list*∗*nat list*) *list* ⇒ (*nat list*∗*nat list*) *list*
where *subsets-smash-R n s1 s2* = *concat* (*map* (λ*l1*. *map* (λ *l2*. (((*fst l1*) @ (*map* ((+) *n*) (*fst l2*))), (*snd l1*) @ (*map* ((+) *n*) (*snd l2*)))) *s2*) *s1*)

definition *smash-systems-R*:: ('*a*::*zero*) *poly* ⇒ ('*a*::*zero*) *poly list* ⇒ ('*a*::*zero*) *poly list* ⇒ (*nat list* ∗ *nat list*) *list* ⇒ (*nat list* ∗ *nat list*) *list* ⇒
rat list list ⇒ *rat list list* ⇒ *rat mat* ⇒ *rat mat* ⇒
('*a*::*zero*) *poly list* × (*rat mat* × ((*nat list* ∗ *nat list*) *list* × *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* ⇒ (('*a*::*zero*) *poly list* × (*rat mat* × ((*nat list* ∗ *nat list*) *list* × *rat list list*))) ⇒ (('*a*::*zero*) *poly list* × (*rat mat* × ((*nat list* ∗ *nat list*) *list* × *rat list list*)))
⇒ (('*a*::*zero*) *poly list* × (*rat mat* × ((*nat list* ∗ *nat list*) *list* × *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* ⇒ *rat list list* ⇒ (*nat list*∗*nat list*) *list* ⇒ *rat vec* ⇒ *rat mat* × ((*nat list*∗*nat list*) *list* × *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* ⇒ (*real poly list* × (*rat mat* × ((*nat list*∗*nat list*) *list* × *rat list list*))) ⇒ (*rat mat* × ((*nat list*∗*nat list*) *list* × *rat list list*))
where *reduce-system-R p* (*qs*,*m*,*subs*,*signs*) =
reduction-step-R m signs subs (*solve-for-lhs-R p qs subs m*)

21 Overall algorithm

fun *calculate-data-R*:: *real poly* ⇒ *real poly list* ⇒ (*rat mat* × ((*nat list*∗*nat list*) *list* × *rat list list*))
where
calculate-data-R p qs =
(*let len* = *length qs in*
if len = *0 then*
(λ(*a*,*b*,*c*).(*a*,*b*,*map* (*drop 1*) *c*)) (*reduce-system-R p* ([*1*],*base-case-info-R*))
else if len ≤ *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* ⇒ *real poly list* ⇒ *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* (λ*x*. *x^2*) (*p # I1*))) *x = 0 ∧ poly* (*prod-list I2*) *x > 0*}) −
int (*card* {*x*. *poly* (*sum-list* (*map* (λ*x*. *x^2*) (*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* (λ*x*. *x^2*) *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* (λ*r*. (*1* − *r*^*2*)) *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* (λ*i* .(*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 index-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 subsets*) *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* ≡
map (λ *q*. *if* (*poly q x* > *0*) *then* (*1*::*rat*) *else* (*if* (*poly q x* = *0*) *then* (*0*::*rat*) *else* (−*1*::*rat*))) *qs*

definition *construct-lhs-vector-R*:: *real poly* ⇒ *real poly list* ⇒ *rat list list* ⇒ *rat vec*
where *construct-lhs-vector-R p qs signs* ≡
vec-of-list (*map* (λ*w*. *rat-of-int* (*int* (*length* (*filter* (λ*v*. *v* = *w*) (*map* (*consistent-sign-vec qs*) (*characterize-root-list-p p*)))))) *signs*)

definition *satisfy-equation-R*:: *real poly* ⇒ *real poly list* ⇒ (*nat list*∗*nat list*) *list* ⇒ *rat list list* ⇒ *bool*
where *satisfy-equation-R p qs subset-list sign-list* =
(*mult-mat-vec* (*matrix-A-R sign-list subset-list*) (*construct-lhs-vector-R p qs sign-list*) = (*construct-rhs-vector-R p qs subset-list*))

lemma *construct-lhs-vector-clean-R*:
assumes *p* ≠ *0*
assumes *i* < *length signs*
shows (*construct-lhs-vector-R p qs signs*) $ *i* =
card {*x*. *poly p x* = *0* ∧ ((*consistent-sign-vec qs x*) = (*nth signs i*))}
proof −
from *poly-roots-finite*[*OF assms*(*1*)] **have** *finite* {*x*. *poly p x* = *0*} .
then have *eq*: (*Collect*

```
      ((λv. v = signs ! i) ∘
       consistent-sign-vec qs) ∩
      set (sorted-list-of-set
           {x. poly p x = 0})) =
    {x. poly p x = 0 ∧ 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 ≠ 0
  shows (construct-lhs-vector-R p qs signs) =
   vec-of-list (map (λs. rat-of-int (card {x. poly p x = 0 ∧ ((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 (λi. i < length signs) I
  assumes la:  list-all (λs. s = 1 ∨ s = 0 ∨ s = −1) signs
  shows (prod-list (map (nth signs) I)) = 1 ∨
  (prod-list (map (nth signs) I)) = 0 ∨
  (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 ∨ signs ! a = 0 ∨ signs ! a = −1
    using assms
    by (smt (verit, del-insts) calculation(2) list-all-length list-all-simps(1))
 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 (λi. i < length signs) I
  assumes la:  list-all (λs. s = 1 ∨ s = 0 ∨ s = −1) signs
  shows (prod-list (map (nth (map-sgas signs)) I)) = 1 ∨
(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 ∨ signs ! a = 0 ∨ signs ! a = −1
    using assms
    by (smt (verit, best) calculation(2) list-all-length list-all-simps(1))
  then have eo: 1 − (signs ! a)^2 = 1 ∨ 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* (λ*i*. *i* < *length sign*) (*fst I*))
using *welldefined1*
by (*auto simp add*: *list-constr-def characterize-consistent-signs-at-roots-def consistent-sign-vec-copr-def*)
thus (*list-all* (λ*i*. *i* < *length sign*) (*snd I*))
using *same welldefined2*
by (*auto simp add*: *list-constr-def characterize-consistent-signs-at-roots-def consistent-sign-vec-copr-def*)
show *list-all* (λ*s*. *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* ∈ {*x*. *poly p x* = *0*}
assumes *sign-fix*: *sign* = *consistent-sign-vec qs x*
shows *sign* ∈ *set* (*characterize-consistent-signs-at-roots p qs*)
proof −
have *h1*: *consistent-sign-vec qs x* ∈
set (*remdups* (*map* (*signs-at qs*) (*sorted-list-of-set* {*x*. *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*≠*0*
fixes *qs*:: *real poly 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*
shows *list-all* (λ*s*. *s* = *1* ∨ *s* = *0* ∨ *s* = −*1*) *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≠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 qs x*
assumes *asm*: *list-constr I* (*length qs*)
shows (*prod-list* (*map* (*nth* (*map-sgas sign*)) *I*)) = *1* ⟷
(∀ *p* ∈ *set* (*retrieve-polys qs I*). *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* ⟷
(∀ *p* ∈ *set* (*retrieve-polys qs xa*). *poly p x* = *0*) **unfolding** *list-constr-def* **by** *auto*
have *welldef*: *a* < *length qs* **using** *Cons.prems 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* (λ*s*. *s* = *1* ∨ *s* = *0* ∨ *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* ∨ (*prod-list* (*map* ((!) (*map-sgas sign*)) (*xa*))) = *1*
using *z-signs-R1 assms Cons.prems consistent-sign-vec-def length-map list-all-simps*(*1*) *length-map list-all-length list-constr-def*
by (*smt* (*verit*, *best*))
have (*sign* ! *a*)^*2* = *1* ∨ (*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* ⟷
((*sign* ! *a*)^*2* = *0* ∧ (*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* ⟷ (*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 ⟷
   ((poly (qs ! a) x = 0) ∧ (prod-list (map ((!) (map-sgas sign)) (xa))) = 1) using
s1 by auto
  have set (retrieve-polys qs (a#xa)) = {(qs ! a)} ∪ 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:(∀ p ∈ set (retrieve-polys qs (a#xa)). poly p x = 0) = ((poly
(qs ! a) x = 0)
      ∧ (∀ p ∈ set (retrieve-polys qs (xa)). 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≠0
  fixes qs:: real poly 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
  shows length sign = length qs
  using assms unfolding consistent-sign-vec-def by auto

lemma horiz-vector-helper-zer-ind-R2:
  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 qs x
  assumes asm: list-constr I (length qs)
  shows (prod-list (map (nth sign) I)) = 0 ⟷
    (poly (prod-list (retrieve-polys qs I)) x = 0)
  using asm
proof (induction I)
  case Nil
 then show ?case unfolding map-sgas-def apply (auto) unfolding retrieve-polys-def
    by auto next
  case (Cons a xa)
   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 show ?case using Cons.prems
```

by (*smt* (*z3*) *Cons.IH class-field.neg-1-not-0 class-field.zero-not-one consistent-sign-vec-def list.simps*(*9*) *list-all-simps*(*1*) *list-constr-def mult-eq-0-iff nth-map prod-list.Cons sign-fix*)
qed

lemma *horiz-vector-helper-pos-ind-R2*:
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 qs x*
assumes *asm*: *list-constr I* (*length qs*)
shows (*prod-list* (*map* (*nth sign*) *I*)) = *1* ⟷
(*poly* (*prod-list* (*retrieve-polys qs I*)) *x* > *0*)
using *asm*
proof (*induction I*)
case *Nil*
then show *?case* **unfolding** *map-sgas-def* **apply** (*auto*) **unfolding** *retrieve-polys-def*
by *auto* **next**
case (*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*) ⟷
(((*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*))
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.prems* **unfolding** *list-constr-def* **by** *auto*
have *sagt*: *sign* ! *a* = *1* ⟷ (*poly* (*qs* ! *a*) *x*) > *0* **using** *assms* **unfolding** *consistent-sign-vec-def*
apply (*auto*)
apply (*smt* (*verit*, *best*) *Cons.prems list-all-simps*(*1*) *list-constr-def neg-equal-zero nth-map zero-neq-one*)
by (*smt* (*verit*, *ccfv-threshold*) *Cons.prems list-all-simps*(*1*) *list-constr-def nth-map*)
have *salt*: *sign* ! *a* = −*1* ⟷ (*poly* (*qs* ! *a*) *x*) < *0* **using** *assms* **unfolding** *consistent-sign-vec-def*
apply (*auto*)
apply (*smt* (*verit*, *ccfv-SIG*) *Cons.prems less-minus-one-simps*(*1*) *less-minus-one-simps*(*3*)

list-all-simps(*1*) *list-constr-def neg-0-less-iff-less nth-map*)
by (*smt* (*verit*, *best*) *Cons.prems list-all-simps*(*1*) *list-constr-def nth-map*)
have *h1*: ((*poly* (*qs* ! *a*) *x*) > *0* ∧ *poly* (*prod-list* (*retrieve-polys qs* (*xa*))) *x* > *0*)
⟶
(*prod-list* (*map* (*nth sign*) (*a*#*xa*))) = *1*
using *prodsame sagt ih* **by** *auto*
have *eo*: (*prod-list* (*map* ((!) *sign*) *xa*) = *1*) ∨ (*prod-list* (*map* ((!) *sign*) *xa*) =
0) ∨
(*prod-list* (*map* ((!) *sign*) *xa*) = −*1*)
using *Cons.prems 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*) ⟹ (*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 *prodsame salt ih assms Cons.prems class-field.neg-1-not-0 equal-neg-zero
horiz-vector-helper-zer-ind-R2 linorder-neqE-linordered-idom list-all-simps*(*1*) *list-constr-def*
apply (*auto*)
apply (*smt* (*verit*, *ccfv-threshold*) *class-field.neg-1-not-0 list.set-map list-all-length
semidom-class.prod-list-zero-iff*)
by (*smt* (*verit*, *ccfv-threshold*) *class-field.neg-1-not-0 list.set-map list-all-length
semidom-class.prod-list-zero-iff*)
qed
have *d2*: (*0* > *poly* (*prod-list* (*retrieve-polys qs xa*)) *x*) ⟶ (*prod-list* (*map* ((!)
sign) *xa*) = −*1*)
using *eo assms horiz-vector-helper-zer-ind-R2*[**where** *p* = *p*, **where** *x* = *x*,
where *sign* = *sign*, **where** *I* =*I*]
apply (*auto*)
using *ih* **apply** *force*
by (*metis* (*full-types*, *lifting*) *Cons.prems class-field.neg-1-not-0 horiz-vector-helper-zer-ind-R2
ih imageI list.set-map list-all-simps*(*1*) *list-constr-def mem-Collect-eq neg-equal-0-iff-equal
semidom-class.prod-list-zero-iff*)
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.prems* ‹*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.prems* **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*: $x \in \{x.\ poly\ p\ x = 0\}$
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 $((\forall p \in set\ (retrieve\text{-}polys\ qs\ (fst\ I)).\ poly\ p\ x = 0) \wedge (poly\ (prod\text{-}list\ (retrieve\text{-}polys\ qs\ (snd\ I)))\ x > 0)) \longleftrightarrow (z\text{-}R\ I\ sign = 1)$
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*: $x \in \{x.\ poly\ p\ x = 0\}$
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 $((\forall p \in set\ (retrieve\text{-}polys\ qs\ (fst\ I)).\ poly\ p\ x = 0) \wedge (poly\ (prod\text{-}list\ (retrieve\text{-}polys\ qs\ (snd\ I)))\ x < 0)) \longleftrightarrow (z\text{-}R\ I\ sign = -1)$
proof −
have *set-hyp*: *sign* ∈ *set* (*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*: $((z\text{-}R\ I\ sign = 1) \vee (z\text{-}R\ I\ sign = 0) \vee (z\text{-}R\ I\ sign = -1))$
using *welldefined1 welldefined2 set-hyp z-lemma-R*[**where** *sign*=*sign*, **where** $I = I$, **where** *p*=*p*, **where** *qs*=*qs*] **by** *blast*
have *d1*: $((\forall p \in set\ (retrieve\text{-}polys\ qs\ (fst\ I)).\ poly\ p\ x = 0) \wedge (poly\ (prod\text{-}list\ (retrieve\text{-}polys\ qs\ (snd\ I)))\ x < 0)) \Longrightarrow (z\text{-}R\ I\ sign = -1)$
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-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*))
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-neq-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*≠*0*
shows
(*vec-of-list* (*mtx-row-R signs I*) ⋅ (*construct-lhs-vector-R p qs signs*)) =
sum-list (*map* (λ*s*. (*z-R I s*) ∗ *rat-of-int* (*card* {*x*. *poly p x* = *0* ∧ *consistent-sign-vec qs x* = *s*})) *signs*)
proof −
have *p* ≠ *0* **using** *nonzero* **by** *auto*
from *construct-lhs-vector-cleaner*[*OF this*]
have *rhseq*: *construct-lhs-vector-R p qs signs* =
vec-of-list
(*map* (λ*s*. *rat-of-int* (*card* {*x*. *poly p x* = *0* ∧ *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* (λ*s*. *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*}) ⟶ *z-R I sgn* = 0
proof −
have ∀ *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*}) ⟶ {*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 ∃ *is*. *is* = *iis* ∧ {*r*. *poly p r* = 0 ∧ *consistent-sign-vec qs r* = *is*} ≠ {}
then have ∃ *is*. *consistent-sign-vec qs* (*rr* (λ*r*. *poly p r* = 0 ∧ *consistent-sign-vec qs r* = *is*)) = *iis* ∧ {*r*. *poly p r* = 0 ∧ *consistent-sign-vec qs r* = *is*} ≠ {}
using *ff2*
by (*metis* (*mono-tags*, *lifting*))
then have ∃ *r*. *poly p r* = 0 ∧ *consistent-sign-vec qs r* = *iis*
using *ff2*
by *smt*
then have *iis* ∈ *consistent-sign-vec qs* ‘ *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* ∉ *set signs* ∨ *iis* ∈ *consistent-sign-vec qs* ‘ *set* (*characterize-root-list-p p*) ∨ ¬ 0 < *rat-of-int* (*int* (*card* {*r*. *poly p r* = 0 ∧ *consistent-sign-vec qs r* = *iis*}))
by (*metis* (*no-types*) ‹∀ *sgn*. *sgn* ∈ *set signs* ∧ *sgn* ∉ *consistent-sign-vec qs* ‘ *set* (*characterize-root-list-p p*) ∧ 0 < *rat-of-int* (*int* (*card* {*xa*. *poly p xa* = 0 ∧ *consistent-sign-vec qs xa* = *sgn*})) ⟶ {*xa*. *poly p xa* = 0 ∧ *consistent-sign-vec qs xa* = *sgn*} ≠ {}› *characterize-root-list-p-def*) **}**
then show *?thesis*
by *fastforce*

qed
qed
then have $\forall$ *sgn. sgn* $\in$ *set signs* $\wedge$ *sgn* $\notin$ *consistent-sign-vec qs* ‘ *set* (*characterize-root-list-p p*) $\longrightarrow$ ((*0* = *rat-of-nat* (*card* {*xa. poly p xa* = *0* $\wedge$ *consistent-sign-vec qs xa* = *sgn*}) $\vee$ *z-R I sgn* = *0*))
by *auto*
then have *hyp*: $\forall$ *s. s* $\in$ *set signs* $\wedge$ *s* $\notin$ *consistent-sign-vec qs* ‘ *set* (*characterize-root-list-p p*) $\longrightarrow$ (*z-R I s* $*$ *rat-of-nat* (*card* {*x. poly p x* = *0* $\wedge$ *consistent-sign-vec qs x* = *s*}) = *0*)
by *auto*
then have ($\sum$ *s*$\in$ *set*(*signs*). *z-R I s* $*$ *rat-of-nat* (*card* {*x. poly p x* = *0* $\wedge$ *consistent-sign-vec qs x* = *s*})) =
($\sum$ *s*$\in$(*set* (*signs*) $\cap$ (*consistent-sign-vec qs* ‘ *set* (*characterize-root-list-p p*))). *z-R I s* $*$ *rat-of-nat* (*card* {*x. poly p x* = *0* $\wedge$ *consistent-sign-vec qs x* = *s*}))
proof −
have *set*(*signs*) =(*set* (*signs*) $\cap$ (*consistent-sign-vec qs* ‘ *set* (*characterize-root-list-p p*))) $\cup$
(*set*(*signs*)−(*consistent-sign-vec qs* ‘ *set* (*characterize-root-list-p p*)))
by *blast*
then have *sum-rewrite*: ($\sum$ *s*$\in$ *set*(*signs*). *z-R I s* $*$ *rat-of-nat* (*card* {*x. poly p x* = *0* $\wedge$ *consistent-sign-vec qs x* = *s*})) =
($\sum$ *s*$\in$ (*set* (*signs*) $\cap$ (*consistent-sign-vec qs* ‘ *set* (*characterize-root-list-p p*))) $\cup$
(*set*(*signs*)−(*consistent-sign-vec qs* ‘ *set* (*characterize-root-list-p p*))). *z-R I s* $*$ *rat-of-nat* (*card* {*x. poly p x* = *0* $\wedge$ *consistent-sign-vec qs x* = *s*}))
by *auto*
then have *sum-split*: ($\sum$ *s*$\in$ (*set* (*signs*) $\cap$ (*consistent-sign-vec qs* ‘ *set* (*characterize-root-list-p p*))) $\cup$
(*set*(*signs*)−(*consistent-sign-vec qs* ‘ *set* (*characterize-root-list-p p*))). *z-R I s* $*$ *rat-of-nat* (*card* {*x. poly p x* = *0* $\wedge$ *consistent-sign-vec qs x* = *s*})) =
($\sum$ *s*$\in$ (*set* (*signs*) $\cap$ (*consistent-sign-vec qs* ‘ *set* (*characterize-root-list-p p*))). *z-R I s* $*$ *rat-of-nat* (*card* {*x. poly p x* = *0* $\wedge$ *consistent-sign-vec qs x* = *s*})) + ($\sum$ *s*$\in$ (*set*(*signs*)−(*consistent-sign-vec qs* ‘ *set* (*characterize-root-list-p p*))). *z-R I s* $*$ *rat-of-nat* (*card* {*x. poly p x* = *0* $\wedge$ *consistent-sign-vec qs x* = *s*}))
by (*metis* (*no-types*, *lifting*) *List.finite-set sum.Int-Diff*)
have *sum-zero*: ($\sum$ *s*$\in$ (*set*(*signs*)−(*consistent-sign-vec qs* ‘ *set* (*characterize-root-list-p p*))). *z-R I s* $*$ *rat-of-nat* (*card* {*x. poly p x* = *0* $\wedge$ *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*: *set* (*remdups*
(*map* (*consistent-sign-vec qs*)
(*characterize-root-list-p p*))) = *set* (*signs*) $\cap$ (*consistent-sign-vec qs* ‘ *set* (*characterize-root-list-p p*))
using *all-info* **apply** (*simp add*: *characterize-consistent-signs-at-roots-def sub-*

set-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*←*signs*. *z-R I s* ∗ *rat-of-nat* (*card* {*x*. *poly p x* = *0* ∧ *consistent-sign-vec qs x* = *s*})) =
($\sum$ *s*∈*set* (*signs*). *z-R I s* ∗ *rat-of-nat* (*card* {*x*. *poly p x* = *0* ∧ *consistent-sign-vec qs x* = *s*}))
using *distinct-signs sum-list-distinct-conv-sum-set* **by** *blast*
have *hyp2*: ($\sum$ *s*←*remdups*
(*map* (*consistent-sign-vec qs*)
(*characterize-root-list-p p*)). *z-R I s* ∗ *rat-of-nat* (*card* {*x*. *poly p x* = *0* ∧ *consistent-sign-vec qs x* = *s*}))
= ($\sum$ *s*∈ *set* (*remdups*
(*map* (*consistent-sign-vec qs*)
(*characterize-root-list-p p*))). *z-R I s* ∗ *rat-of-nat* (*card* {*x*. *poly p x* = *0* ∧ *consistent-sign-vec qs x* = *s*}))
using *sum-list-distinct-conv-sum-set* **by** *blast*
have *set-sum-eq*: ($\sum$ *s*∈(*set* (*signs*) ∩ (*consistent-sign-vec qs* ‘ *set* (*characterize-root-list-p p*))). *z-R I s* ∗ *rat-of-nat* (*card* {*x*. *poly p x* = *0* ∧ *consistent-sign-vec qs x* = *s*}))
=
($\sum$ *s*∈ *set* (*remdups*
(*map* (*consistent-sign-vec qs*)
(*characterize-root-list-p p*))). *z-R I s* ∗ *rat-of-nat* (*card* {*x*. *poly p x* = *0* ∧ *consistent-sign-vec qs x* = *s*}))
using *set-eq* **by** *auto*
then have ($\sum$ *s*←*signs*. *z-R I s* ∗ *rat-of-nat* (*card* {*x*. *poly p x* = *0* ∧ *consistent-sign-vec qs x* = *s*})) =
($\sum$ *s*←*remdups*
(*map* (*consistent-sign-vec qs*)
(*characterize-root-list-p p*)). *z-R I s* ∗ *rat-of-nat* (*card* {*x*. *poly p x* = *0* ∧ *consistent-sign-vec qs x* = *s*}))
using *set-sum-eq hyp1 hyp2*
using ‹($\sum$ *s*∈*set signs*. *z-R I s* ∗ *rat-of-nat* (*card* {*x*. *poly p x* = *0* ∧ *consistent-sign-vec qs x* = *s*})) = ($\sum$ *s*∈*set signs* ∩ *consistent-sign-vec qs* ‘ *set* (*characterize-root-list-p p*). *z-R I s* ∗ *rat-of-nat* (*card* {*x*. *poly p x* = *0* ∧ *consistent-sign-vec qs x* = *s*}))›
by *linarith*
then have *consistent-sign-vec qs* ‘ *set* (*characterize-root-list-p p*) ⊆ *set signs* ⟹
(⋀*p qss*.
characterize-consistent-signs-at-roots p qss =
remdups (*map* (*consistent-sign-vec qss*) (*characterize-root-list-p p*))) ⟹
($\sum$ *s*←*signs*. *z-R I s* ∗ *rat-of-nat* (*card* {*x*. *poly p x* = *0* ∧ *consistent-sign-vec qs x* = *s*})) =
($\sum$ *s*←*remdups*
(*map* (*consistent-sign-vec qs*)
(*characterize-root-list-p p*)). *z-R I s* ∗ *rat-of-nat* (*card* {*x*. *poly p x* = *0* ∧ *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 consis-*

tent-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*≠*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*)) =
rat-of-int (*card* {*x*. *poly p x* = *0* ∧ (∀ *p* ∈ *set* (*retrieve-polys qs* (*fst I*)). *poly p x* = *0*) ∧ *poly* (*prod-list* (*retrieve-polys qs* (*snd I*))) *x* > *0*}) −
rat-of-int (*card* {*x*. *poly p x* = *0* ∧ (∀ *p* ∈ *set* (*retrieve-polys qs* (*fst I*)). *poly p x* = *0*) ∧ *poly* (*prod-list* (*retrieve-polys qs* (*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*)) ∩ {*s*. *z-R I s* = *1*})
let *?lt* = (*set* (*map* (*consistent-sign-vec qs*) (*characterize-root-list-p p*)) ∩ {*s*. *z-R I s* = −*1*})
let *?zer* = (*set* (*map* (*consistent-sign-vec qs*) (*characterize-root-list-p p*)) ∩ {*s*. *z-R I s* = *0*})
have *eq*: *set* (*map* (*consistent-sign-vec qs*) (*characterize-root-list-p p*)) = (*?gt* ∪ *?lt*) ∪ *?zer*
proof *safe*
fix *x*
assume *h*:*x* ∈ *set* (*map* (*consistent-sign-vec qs*) (*characterize-root-list-p p*))
z-R I x ≠ *0 z-R I x* ≠ − *1*
then have *x* ∈ *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*: (∑ *s*∈(*?gt*∪*?lt*). *z-R I s* ∗ *rat-of-int* (*card* {*x*. *poly p x* = *0* ∧ *consistent-sign-vec qs x* = *s*}))
= (∑ *s*∈*?gt*. *z-R I s* ∗ *rat-of-int* (*card* {*x*. *poly p x* = *0* ∧ *consistent-sign-vec qs x* = *s*})) +
(∑ *s*∈*?lt*. *z-R I s* ∗ *rat-of-int* (*card* {*x*. *poly p x* = *0* ∧ *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*) · *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*) **.**

from *lhs-dot-rewrite*[*OF assms*(*1*)]
moreover have ... =
(∑ *s*←*characterize-consistent-signs-at-roots p qs*.
z-R I s ∗ *rat-of-int* (*card* {*x*. *poly p x* = *0* ∧ *consistent-sign-vec qs x* = *s*})) **.**
moreover have ... =
(∑ *s*∈*set* (*map* (*consistent-sign-vec qs*) (*characterize-root-list-p p*)).
z-R I s ∗ *rat-of-int* (*card* {*x*. *poly p x* = *0* ∧ *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*:... =
(∑ *s*∈(*?gt*∪*?lt*). *z-R I s* ∗ *rat-of-int* (*card* {*x*. *poly p x* = *0* ∧ *consistent-sign-vec qs x* = *s*})) +
(∑ *s*∈*?zer*. *z-R I s* ∗ *rat-of-int* (*card* {*x*. *poly p x* = *0* ∧ *consistent-sign-vec qs x* = *s*}))
apply (*subst eq*)
apply (*rule sum.union-disjoint*) **by** *auto*
ultimately have *setc*: ... =
(∑ *s*∈*?gt*. *z-R I s* ∗ *rat-of-int* (*card* {*x*. *poly p x* = *0* ∧ *consistent-sign-vec qs x* = *s*})) +
(∑ *s*∈*?lt*. *z-R I s* ∗ *rat-of-int* (*card* {*x*. *poly p x* = *0* ∧ *consistent-sign-vec qs x* = *s*})) +
(∑ *s*∈*?zer*. *z-R I s* ∗ *rat-of-int* (*card* {*x*. *poly p x* = *0* ∧ *consistent-sign-vec qs x* = *s*}))
using *sumeq* **by** *auto*
have ∀ *s* ∈ *?zer*. (*z-R I s* ∗ *rat-of-int* (*card* {*x*. *poly p x* = *0* ∧ *consistent-sign-vec qs x* = *s*})) = *0*
by *auto*
then have *obvzer*: (∑ *s*∈*?zer*. *z-R I s* ∗ *rat-of-int* (*card* {*x*. *poly p x* = *0* ∧ *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 ∗: ⋀*s*. {*x*. *poly p x* = *0* ∧ *consistent-sign-vec qs x* = *s*} =
{*x* ∈{*x*. *poly p x* = *0*}. *consistent-sign-vec qs x* = *s*}
by *auto*
have *e1*: (∑ *s*∈*consistent-sign-vec qs* ‘ {*x*. *poly p x* = *0*} ∩ {*s*. *z-R I s* = *1*}.
card {*x*. *poly p x* = *0* ∧ *consistent-sign-vec qs x* = *s*}) =
(*sum* (λ*x*. *if* (∀ *p* ∈ *set* (*retrieve-polys qs* (*fst I*)). *poly p x* = *0*) ∧ (*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* ∈ *consistent-sign-vec qs* ' {*x. poly p x = 0*} ∩ {*s. z-R I s = 1*}. *consistent-sign-vec qs x = s*} =
{*s. z-R I s = 1* ∧ *consistent-sign-vec qs x = s*}
by *auto*
have *2*: ... = {*s.* (∀ *p* ∈ *set* (*retrieve-polys qs* (*fst I*)). *poly p x = 0*) ∧ (*0 < poly* (*prod-list* (*retrieve-polys qs* (*snd I*))) *x*) ∧ *consistent-sign-vec qs x = s*}
using *horiz-vector-helper-pos-R assms welldefined1 welldefined2 rt* **by** *blast*
have *3*: ... = (*if* (∀ *p* ∈ *set* (*retrieve-polys qs* (*fst I*)). *poly p x = 0*) ∧ (*0 < poly* (*prod-list* (*retrieve-polys qs* (*snd I*))) *x*) *then* {*consistent-sign-vec qs x*} *else* {})
by *auto*
then have *card* {*s* ∈ *consistent-sign-vec qs* ' {*x. poly p x = 0*} ∩ {*s. z-R I s = 1*}. *consistent-sign-vec qs x = s*} =
(*if* ((∀ *p* ∈ *set* (*retrieve-polys qs* (*fst I*)). *poly p x = 0*) ∧ *0 < poly* (*prod-list* (*retrieve-polys qs* (*snd I*))) *x*)
then 1 else 0) **using** *1 2 3* **by** *auto*
thus *card*
{*s* ∈ *consistent-sign-vec qs* ' {*x. poly p x = 0*} ∩ {*s. z-R I s = 1*}.
consistent-sign-vec qs x = s} =
(*if* (∀ *p*∈*set* (*retrieve-polys qs* (*fst I*)). *poly p x = 0*) ∧
0 < poly (*prod-list* (*retrieve-polys qs* (*snd I*))) *x*
then 1 else 0)
by *auto*
qed

have *gtchr*: (∑ *s*∈*?gt. z-R I s* ∗ *rat-of-int* (*card* {*x. poly p x = 0* ∧ *consistent-sign-vec qs x = s*})) =
rat-of-int (*card* {*x. poly p x = 0* ∧ (∀ *p*∈*set* (*retrieve-polys qs* (*fst I*)). *poly p x = 0*) ∧ *0 < poly* (*prod-list* (*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*: (∑ *s*∈*consistent-sign-vec qs* ' {*x. poly p x = 0*} ∩ {*s. z-R I s = − 1*}.
card {*x. poly p x = 0* ∧ *consistent-sign-vec qs x = s*}) =
(*sum* (λ*x. if* (∀ *p*∈*set* (*retrieve-polys qs* (*fst I*)). *poly p x = 0*) ∧ (*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* ∈ *consistent-sign-vec qs* ' {*x*. *poly p x = 0*} ∩ {*s*. *z-R I s = −1*}. *consistent-sign-vec qs x = s*} =
{*s*. *z-R I s = −1* ∧ *consistent-sign-vec qs x = s*}
by *auto*
have *2*: ... = {*s*. ((∀ *p*∈*set* (*retrieve-polys qs* (*fst I*)). *poly p x = 0*) ∧ *0 > poly* (*prod-list* (*retrieve-polys qs* (*snd I*))) *x*) ∧ *consistent-sign-vec qs x = s*}
using *horiz-vector-helper-neg-R assms rt welldefined1 welldefined2* **by** *blast*
have *3*: ... = (*if* (∀ *p*∈*set* (*retrieve-polys qs* (*fst I*)). *poly p x = 0*) ∧ (*0 > poly* (*prod-list* (*retrieve-polys qs* (*snd I*))) *x*) *then* {*consistent-sign-vec qs x*} *else* {})
by *auto*
thus *card* {*s* ∈ *consistent-sign-vec qs* ' {*x*. *poly p x = 0*} ∩ {*s*. *z-R I s = −1*}. *consistent-sign-vec qs x = s*} =
(*if* (∀ *p*∈*set* (*retrieve-polys qs* (*fst I*)). *poly p x = 0*) ∧ *0 > poly*
(*prod-list*
(*retrieve-polys qs* (*snd I*)))
x
then 1 else 0) **using** *1 2 3* **by** *auto*
qed
have *ltchr*: (∑ *s*∈*?lt*. *z-R I s* ∗ *rat-of-int* (*card* {*x*. *poly p x = 0* ∧ *consistent-sign-vec qs x = s*})) =
− *rat-of-int* (*card* {*x*. *poly p x = 0* ∧ (∀ *p*∈*set* (*retrieve-polys qs* (*fst I*)). *poly p x = 0*) ∧ *0 > poly* (*prod-list* (*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-roo*
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 (λ*r*. *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*≠*0*
assumes *distinct-signs*: *distinct signs*
assumes *all-info*: *set* (*characterize-consistent-signs-at-roots p qs*) ⊆ *set*(*signs*)
assumes *welldefined*: *all-list-constr-R* (*subsets*) (*length qs*)
shows *satisfy-equation-R p qs subsets 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 in-set-member welldefined* **by** *fastforce*

lemma *consistent-signs-at-roots-eq*:
assumes *p* ≠ *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-R*:: *real poly* ⇒ *real poly list* ⇒ *rat list list* ⇒ *rat vec*
where *w-vec-R* ≡ *construct-lhs-vector-R*

abbreviation *v-vec-R*:: *real poly* ⇒ *real poly list* ⇒ (*nat list*∗*nat list*) *list* ⇒ *rat vec*
where *v-vec-R* ≡ *construct-rhs-vector-R*

abbreviation *M-mat-R*:: *rat list list* ⇒ (*nat list*∗*nat list*) *list* ⇒ *rat mat*
where *M-mat-R* ≡ *matrix-A-R*

theorem *matrix-equation-pretty*:
fixes *subsets*:: (*nat list*∗*nat list*) *list*
assumes *p*≠*0*
assumes *distinct signs*
assumes *consistent-signs-at-roots p qs* ⊆ *set signs*
assumes ⋀*a b i*. (*a*, *b*) ∈ *set* (*subsets*) ⟹ (*i* ∈ *set a* ∨ *i* ∈ *set b*) ⟹ *i* < *length qs*
shows *M-mat-R signs subsets* ∗$_v$ *w-vec-R p qs signs* = *v-vec-R p qs 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 in-set-member*)

24 Base Case

definition *satisfies-properties-R*:: *real poly* ⇒ *real poly list* ⇒ (*nat list*∗*nat list*) *list* ⇒ *rat list list* ⇒ *rat mat* ⇒ *bool*
where *satisfies-properties-R p qs subsets signs matrix* =
(*all-list-constr-R subsets* (*length qs*) ∧ *well-def-signs* (*length qs*) *signs* ∧ *distinct signs*
∧ *satisfy-equation-R p qs subsets signs* ∧ *invertible-mat matrix* ∧ *matrix* = *matrix-A-R signs subsets*
∧ *set* (*characterize-consistent-signs-at-roots p qs*) ⊆ *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 mtx-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*]) ⊆ {[*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*) ⊆ {[*1*], [*0*], [− *1*]}
proof −
have *ex-q*: ∃(*q*::*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*]) ⊆ {[*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 member-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*: ∃(*q*::*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*::*nat*) < *Suc* (*Suc* (*Suc 0*)) ⟷ *n* = *0* ∨ *n* = *1* ∨ *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*]]::*rat mat*) (*mat-of-rows-list 3* [[*1*,*1*,*1*], [*0*,*1*,*0*], [*1*,*0*,−*1*]]::*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*]]::*rat mat*) (*mat-of-rows-list 3* [[*1*/*2*, −*1*/*2*, *1*/*2*], [*0*, *1*, *0*], [*1*/*2*, −*1*/*2*, −*1*/*2*]]:: *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* ⇒ *nat list list*
where *subsets-first-component-list I* = *map* (λ*I*. (*fst I*)) *I*

definition *subsets-second-component-list*::(*nat list*∗*nat list*) *list* ⇒ *nat list list*
where *subsets-second-component-list I* = *map* (λ*I*. (*snd I*)) *I*

definition *smash-list-list*::('*a list*∗'*a list*) *list* ⇒ ('*a list*∗'*a list*) *list* ⇒ ('*a list*∗'*a list*) *list*
where *smash-list-list s1 s2* = *concat* (*map* (λ*l1*. *map* (λ*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 ∀ (*elem* :: (*'a list*∗*'a list*)). (*elem* ∈ (*set* (*smash-list-list l1 l2*)) ⟶
(∃ (*elem1*:: (*'a list*∗*'a list*)). ∃ (*elem2*:: (*'a list*∗*'a list*)).
(*elem1* ∈ *set*(*l1*) ∧ *elem2* ∈ *set*(*l2*) ∧ *elem* = (*fst elem1*@ *fst elem2*, *snd elem1* @ *snd elem2*))))
proof *clarsimp*
fix *a b*
assume *assum*: (*a*, *b*) ∈ *set* (*smash-list-list l1 l2*)
show ∃ *aa ba*. (*aa*, *ba*) ∈ *set l1* ∧ (∃ *ab bb*. (*ab*, *bb*) ∈ *set l2* ∧ *a* = *aa* @ *ab* ∧ *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 ∀ (*elem* :: *nat list*∗*nat list*). (*List.member* (*subsets-smash-R n subsets1 subsets2*) *elem*) ⟶
(∃ (*elem1*::*nat list*∗*nat list*). ∃ (*elem2*::*nat list*∗*nat list*).
(*elem1* ∈ *set*(*subsets1*) ∧ *elem2* ∈ *set*(*subsets2*) ∧ *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* (λ*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 in-set-member 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*: ∀ *elem*.
List.member (*subsets-smash-R ?n subsets1 subsets2*) *elem* ⟶
(∃ (*elem1*::*nat list*∗*nat list*). ∃ (*elem2*::*nat list*∗*nat list*).
(*elem1* ∈ *set*(*subsets1*) ∧ *elem2* ∈ *set*(*subsets2*) ∧ *elem* = ((*fst elem1*) @ (*map* ((+) *?n*) (*fst elem2*)), (*snd elem1*) @ (*map* ((+) *?n*) (*snd elem2*)))))
using *subsets-smash-property-R* **by** *blast*
have *h2*: ∀ *elem1 elem2*. (*elem1* ∈ *set subsets1* ∧ *elem2* ∈ *set subsets2*) ⟶ *list-constr* ((*fst elem1*) @ *map* ((+) (*length qs1*)) (*fst elem2*)) (*length* (*qs1* @ *qs2*))
proof *clarsimp*
fix *elem1 b elem2 ba*
assume *e1*: (*elem1*, *b*) ∈ *set subsets1*
assume *e2*: (*elem2*, *ba*) ∈ *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*
apply (*auto*)
by (*simp add*: *in-set-member*)
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 add*: *in-set-member*)
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*: ∀ *elem1 elem2*. (*elem1* ∈ *set subsets1* ∧ *elem2* ∈ *set subsets2*) ⟶ *list-constr* ((*snd elem1*) @ *map* ((+) (*length qs1*)) (*snd elem2*)) (*length* (*qs1* @ *qs2*))
proof *clarsimp*
fix *elem1 b elem2 ba*
assume *e1*: (*b*, *elem1*) ∈ *set subsets1*
assume *e2*: (*ba*, *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-R-def*
apply (*auto*)
by (*simp add*: *in-set-member*)

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 add*: *in-set-member*)
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 ∀ *x* ∈ *set* (*characterize-consistent-signs-at-roots p* (*qs1* @ *qs2*)).
∃ *x1* ∈ *set* (*characterize-consistent-signs-at-roots p qs1*).
∃ *x2* ∈ *set* (*characterize-consistent-signs-at-roots p qs2*).
x = *x1*@*x2*
proof *clarsimp*
fix *x*
assume *x-in*: *x* ∈ *set* (*characterize-consistent-signs-at-roots p* (*qs1* @ *qs2*))
have *x-in-csv*: *x* ∈ *set*(*map* (*consistent-sign-vec* (*qs1* @ *qs2*)) (*characterize-root-list-p 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*: ∀ *r*. *consistent-sign-vec* (*qs1* @ *qs2*) *r* = (*consistent-sign-vec qs1 r*)@(*consistent-sign-vec qs2 r*)
unfolding *consistent-sign-vec-def* **by** *auto*
have *exr-iff*: (∃ *r* ∈ *set* (*characterize-root-list-p p*). *x* = *consistent-sign-vec* (*qs1* @ *qs2*) *r*) ⟷ (∃ *r* ∈ *set* (*characterize-root-list-p p*). *x* = (*consistent-sign-vec qs1 r*)@(*consistent-sign-vec qs2 r*))
using *csv-hyp* **by** *auto*
have *exr*: *x* ∈ *set*(*map* (*consistent-sign-vec* (*qs1* @ *qs2*)) (*characterize-root-list-p p*)) ⟶ (∃ *r* ∈ *set* (*characterize-root-list-p p*). *x* = *consistent-sign-vec* (*qs1* @ *qs2*) *r*)

by *auto*
have *mem-list1*: ∀ *r* ∈ *set* (*characterize-root-list-p p*). (*consistent-sign-vec qs1 r*) ∈ *set*(*map* (*consistent-sign-vec qs1*) (*characterize-root-list-p p*))
by *simp*
have *mem-list2*: ∀ *r* ∈ *set* (*characterize-root-list-p p*). (*consistent-sign-vec qs2 r*) ∈ *set*(*map* (*consistent-sign-vec qs2*) (*characterize-root-list-p p*))
by *simp*
then show ∃ *x1*∈*set* (*characterize-consistent-signs-at-roots p qs1*).
∃ *x2*∈*set* (*characterize-consistent-signs-at-roots p qs2*). *x* = *x1* @ *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*: *set* (*characterize-consistent-signs-at-roots p qs1*) ⊆ *set* (*signs1*)
assumes *csa2*: *set* (*characterize-consistent-signs-at-roots p qs2*) ⊆ *set* (*signs2*)
shows *set* (*characterize-consistent-signs-at-roots p*
(*qs1* @ *qs2*))
⊆ *set* (*signs-smash signs1 signs2*)
proof −
have *h0*: ∀ *x* ∈ *set* (*characterize-consistent-signs-at-roots p* (*qs1* @ *qs2*)). ∃ *x1* ∈ *set* (*characterize-consistent-signs-at-roots p qs1*). ∃ *x2* ∈ *set* (*characterize-consistent-signs-at-roots p qs2*).
(*x* = *x1*@*x2*) **using** *subset-helper-R*[*of p qs1 qs2*]
by *blast*
then have ∀ *x* ∈ *set* (*characterize-consistent-signs-at-roots p* (*qs1* @ *qs2*)).
x ∈ *set* (*signs-smash* (*characterize-consistent-signs-at-roots p qs1*)
(*characterize-consistent-signs-at-roots p qs2*))
unfolding *signs-smash-def* **apply** (*auto*)
by *fastforce*
then have ∀ *x* ∈ *set* (*characterize-consistent-signs-at-roots p*
(*qs1* @ *qs2*)). *x* ∈ *set* (*signs-smash signs1 signs2*)
using *csa1 csa2 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 ⋀*i*. *i* ∈ *set* (*fst xs*) ⟹ *i* < *length as*
assumes ⋀*i*. *i* ∈ *set* (*snd xs*) ⟹ *i* < *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*) ∘ (+) (*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*) ∘ (+) (*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 ⋀*l i*. *l* ∈ *set subs1* ⟹ (*i* ∈ *set* (*fst l*) ∨ *i* ∈ *set* (*snd l*)) ⟹ *i* < *n1*
assumes ⋀*j*. *j* ∈ *set signs1* ⟹ *length j* = *n1*
shows (*matrix-A-R* (*signs-smash signs1 signs2*) (*subsets-smash-R n1 subs1 subs2*))
=
kronecker-product (*matrix-A-R signs1 subs1*) (*matrix-A-R signs2 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 < length subs1 * length subs2
  assume j: j < length signs1 * length signs2

  have ld: i div length subs2 < length subs1
    j div length signs2 < length signs1
    using i j less-mult-imp-div-less by auto
  have lm: i mod length subs2 < length subs2
    j mod length signs2 < length signs2
    using i j less-mult-imp-mod-less by auto

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

   have 1: matrix-A-R (signs-smash signs1 signs2) (subsets-smash-R n1 subs1
subs2) $$ (i, j) =
    z-R (subsets-smash-R n1 subs1 subs2 ! i) (signs-smash signs1 signs2 ! j)
    unfolding mat-of-rows-list-def matrix-A-R-def mtx-row-R-def
    using i j by (auto simp add: length-signs-smash length-subsets-smash-R)
  have 2: ... = z-R ((fst (subs1 ! (i div length subs2)) @ map ((+) n1) (fst(subs2
! (i mod length subs2)))),
  (snd (subs1 ! (i div length subs2)) @ map ((+) n1) (snd (subs2 ! (i mod length
subs2)))))
     (signs1 ! (j div length signs2) @ signs2 ! (j mod length signs2))
    unfolding signs-smash-def subsets-smash-R-def
    apply (subst length-eq-concat)
    using i apply (auto simp add: mult.commute)
    apply (subst length-eq-concat)
    using j apply (auto simp add: mult.commute)
    using ld lm by auto
  have 3: ... =
  z-R (subs1 ! (i div length subs2)) (signs1 ! (j div length signs2)) *
  z-R (subs2 ! (i mod length subs2)) (signs2 ! (j mod length signs2))
    unfolding n1
    apply (subst z-append-R)
    apply (auto simp add: n1[symmetric])
    using assms(1) ld(1) nth-mem
    apply blast
    using assms(1) ld(1) nth-mem by blast
  have 4: kronecker-product (matrix-A-R signs1 subs1) (matrix-A-R signs2 subs2)
$$ (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*: $p \neq 0$
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 in-set-member well-def-signs-def welldefined-signs1 in-set-conv-nth list-all-length*
apply (*auto*)

by (*smt* (*z3*) *Ball-set kronecker-invertible member-def*)
have *h2a*: *distinct* (*signs-smash signs1 signs2*)
using *distinct-signs1 distinct-signs2 welldefined-signs1 welldefined-signs2 nontriv1 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*)) ⊆ *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* × ((*nat list*∗*nat list*) *list* × *rat list list*)) ⇒ *rat mat*
where *get-matrix-R data* = *fst*(*data*)

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

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

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

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

lemma *reduction-signs-is-get-signs-R*: *reduction-signs-R p qs signs subsets m* = *get-signs-R* (*reduce-system-R p* (*qs*, (*m*, (*subsets*, *signs*))))
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* < *length* (*take-indices arb-list index-list*)
assumes *well-def*: ∀ *q*.(*List.member index-list q* ⟶ *q* < *length arb-list*)
shows ∃ *k*<*length arb-list*.
(*take-indices arb-list index-list*) ! *n* = *arb-list* ! *k*
using *lest well-def* **unfolding** *take-indices-def* **apply** (*auto*)
by (*metis member-def 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*) ∈ *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*))
= (1_m (*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*: $p \neq 0$
assumes *welldefined-signs1*: *well-def-signs* (*length qs*) *signs*
assumes *distinct-signs*: *distinct signs*
assumes *all-info*: *set* (*characterize-consistent-signs-at-roots p qs*) ⊆ *set*(*signs*)
assumes *match*: *satisfy-equation-R p qs subsets signs*
assumes *invertible-mat*: *invertible-mat* (*matrix-A-R signs subsets*)
shows *set* (*characterize-consistent-signs-at-roots p qs*) ⊆ *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*)) ∈ *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* (λ*s*. *card* {*x*. *poly p x* = *0* ∧ *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 ∀ *n* < (*dim-vec* (*solve-for-lhs-R p qs subsets* (*matrix-A-R signs subsets*))).
(((*solve-for-lhs-R p qs subsets* (*matrix-A-R signs subsets*)) $ *n* = *0*) ⟶
(*nth signs n*) ∉ *set* (*characterize-consistent-signs-at-roots p qs*))
proof −
have *h0*: ∀ *n* < (*dim-vec* (*solve-for-lhs-R p qs subsets* (*matrix-A-R signs subsets*))).
(((*solve-for-lhs-R p qs subsets* (*matrix-A-R signs subsets*)) $ *n* = *0*) ⟶
rat-of-nat (*card* {*x*. *poly p x* = *0* ∧ *consistent-sign-vec qs x* = (*nth signs n*)})
= *0*)
by (*metis* (*mono-tags*, *lifting*) ‹*construct-lhs-vector-R p qs signs* = *solve-for-lhs-R p qs subsets* (*matrix-A-R signs subsets*)› *construct-lhs-vector-clean-R nonzero of-nat-0-eq-iff of-rat-of-nat-eq size-of-lhs-R*)
have *h1*: ∀ *w*. (*rat-of-nat* (*card* {*x*. *poly p x* = *0* ∧ *consistent-sign-vec qs x* = *w*}) = *0* ⟶
(∄ *x*. *poly p x* = *0* ∧ *consistent-sign-vec qs x* = *w*))
proof *clarsimp*
fix *x*
assume *card-asm*: *card* {*xa*. *poly p xa* = *0* ∧ *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* ∧ *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* ∈ {*xa*. *poly p xa* = *0* ∧ *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*: ⋀ *w*. (*rat-of-nat* (*card* {*x*. *poly p x* = *0* ∧ *consistent-sign-vec qs x* = *w*}) = *0* ⟹
(¬*List.member* (*characterize-consistent-signs-at-roots p qs*) *w*))
proof *clarsimp*
fix *w*
assume *card-asm*: *card* {*x*. *poly p x* = *0* ∧ *consistent-sign-vec qs x* = *w*} = *0*
assume *mem-asm*: *List.member* (*characterize-consistent-signs-at-roots p qs*) *w*
have *h0*: ∄ *x*. *poly p x* = *0* ∧ *consistent-sign-vec qs x* = *w* **using** *h1 card-asm*
by (*simp add*: *h1*)
have *h1*: ∃ *x*. *poly p x* = *0* ∧ *consistent-sign-vec qs x* = *w* **using** *mem-asm*
unfolding *characterize-consistent-signs-at-roots-def characterize-root-list-p-def*
proof −
have *w* ∈ *Collect* (*List.member* (*remdups* (*map* (*consistent-sign-vec qs*)

(*sorted-list-of-set* {*r*. *poly p r* = *0*}))))
using *characterize-consistent-signs-at-roots-def mem-asm characterize-root-list-p-def*
by (*smt* (*verit*, *ccfv-SIG*) *consistent-sign-vec-def h0 imageE in-set-member list.set-map map-cong mem-Collect-eq nonzero o-apply poly-roots-finite set-remdups set-sorted-list-of-set signs-at-def squash-def*)
then have *f1*: *w* ∈ *set* (*map* (*consistent-sign-vec qs*) (*sorted-list-of-set* {*r*. *poly p r* = *0*}))
by (*metis* (*no-types*) *in-set-member mem-Collect-eq set-remdups*)
have *finite* {*r*. *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*: ∀ *w*. *rat-of-nat* (*card* {*x*. *poly p x* = *0* ∧ *consistent-sign-vec qs x* = *w*}) = *0* ⟶
w ∉ *set* (*characterize-consistent-signs-at-roots p qs*)
by (*simp add*: *in-set-member*)
show *?thesis* **using** *h0 h3*
by *blast*
qed
then have *set* (*characterize-consistent-signs-at-roots p qs*) ⊆ *set* (*take-indices signs*
(*find-nonzeros-from-input-vec* (*solve-for-lhs-R p qs subsets* (*matrix-A-R signs subsets*))))
using *all-info*
reduction-signs-set-helper-lemma[**where** *A* = *set* (*characterize-consistent-signs-at-roots p qs*), **where** *B* = *signs*,
where *C* = (*solve-for-lhs-R p qs subsets* (*matrix-A-R signs 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* ≠ *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*) = *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*)) ∈ *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*) ⊇ *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* (λ*s*. *card* {*x*. *poly p x* = *0* ∧ *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 ∀ *n* < (*dim-vec* (*solve-for-lhs-R p qs subsets* (*matrix-A-R signs subsets*))).
(((*solve-for-lhs-R p qs subsets* (*matrix-A-R signs subsets*)) $ *n* ≠ *0*) ⟶
(*nth signs n*) ∈ *set* (*characterize-consistent-signs-at-roots p qs*))
proof −
have *h0*: ∀ *n* < (*dim-vec* (*solve-for-lhs-R p qs subsets* (*matrix-A-R signs subsets*))).
(((*solve-for-lhs-R p qs subsets* (*matrix-A-R signs subsets*)) $ *n* = *0*) ⟶
rat-of-nat (*card* {*x*. *poly p x* = *0* ∧ *consistent-sign-vec qs x* = (*nth signs n*)}) = *0*)
by (*simp add*: ‹*solve-for-lhs-R p qs subsets* (*M-mat-R signs subsets*) = *vec-of-list* (*map rat-of-nat* (*map* (λ*s*. *card* {*x*. *poly p x* = *0* ∧ *consistent-sign-vec qs x* = *s*}) *signs*))› *vec-of-list-index*)
have *h1*: ∀ *w*. (*rat-of-nat* (*card* {*x*. *poly p x* = *0* ∧ *consistent-sign-vec qs x* = *w*}) ≠ *0* ⟶
(∃ *x*. *poly p x* = *0* ∧ *consistent-sign-vec qs x* = *w*))
proof *clarsimp*
fix *w*
assume *card-asm*: *0* < *card* {*x*. *poly p x* = *0* ∧ *consistent-sign-vec qs x* = *w*}
show ∃ *x*. *poly p x* = *0* ∧ *consistent-sign-vec qs x* = *w*
by (*metis* (*mono-tags*, *lifting*) *Collect-empty-eq card-asm card-eq-0-iff gr-implies-not0*)
qed
have *h2*: ⋀ *w*. (*rat-of-nat* (*card* {*x*. *poly p x* = *0* ∧ *consistent-sign-vec qs x* = *w*}) ≠ *0* ⟹
(*List.member* (*characterize-consistent-signs-at-roots p qs*) *w*))
proof *clarsimp*
fix *w*
assume *card-asm*: *0* < *card* {*x*. *poly p x* = *0* ∧ *consistent-sign-vec qs x* = *w*}
have *h0*: ∃ *x*. *poly p x* = *0* ∧ *consistent-sign-vec qs x* = *w*
using *card-asm*

by (*simp add*: *h1*)
then show *List.member* (*characterize-consistent-signs-at-roots p qs*) *w*
unfolding *characterize-consistent-signs-at-roots-def*
using *in-set-member 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*: ∀ *w*. *rat-of-nat* (*card* {*x*. *poly p x* = *0* ∧ *consistent-sign-vec qs x* = *w*}) ≠ *0* ⟶
w ∈ *set* (*characterize-consistent-signs-at-roots p qs*)
by (*simp add*: *in-set-member*)
show *?thesis* **using** *h0 h3*
by (*metis* (*no-types*, *lifting*) ‹*solve-for-lhs-R p qs subsets* (*matrix-A-R signs subsets*) = *vec-of-list* (*map rat-of-nat* (*map* (λ*s*. *card* {*x*. *poly p x* = *0* ∧ *consistent-sign-vec qs x* = *s*}) *signs*))› *dim-vec-of-list length-map nth-map vec-of-list-index*)
qed
then have *set* (*take-indices signs*
(*find-nonzeros-from-input-vec* (*solve-for-lhs-R p qs subsets* (*matrix-A-R signs subsets*)))) ⊆
set (*characterize-consistent-signs-at-roots p qs*)
using *all-info*
reduction-signs-set-helper-lemma2[**where** *A* = *set* (*characterize-consistent-signs-at-roots p qs*), **where** *B* = *signs*,
where *C* = (*solve-for-lhs-R p qs subsets* (*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 *subset*: *set* (*characterize-consistent-signs-at-roots p qs*) ⊆ *set*(*reduction-signs-R p qs signs subsets* (*matrix-A-R signs subsets*))
using *reduction-doesnt-break-things-signs-R*[*of p qs 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* ≠ *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* (*get-signs-R* (*reduce-system-R p* (*qs*, ((*matrix-A-R signs subsets*), (*subsets*, *signs*))))) = *set* (*characterize-consistent-signs-at-roots p qs*)
unfolding *get-signs-R-def*

using *reduction-deletes-bad-sign-conds-R*[*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 snd-conv*
apply (*metis reduction-signs-R-def*)
by (*metis all-info distinct-signs invertible-mat match nonzero reduction-signs-R-def snd-conv welldefined-signs1*)

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*) ⊆ *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*: ∀ *i* < *length* (*take-indices signs ?sym*). ∀ *j* < *length*(*take-indices signs ?sym*).
i ≠ *j* ⟶ *nth* (*take-indices signs ?sym*) *i* ≠ *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-R p qs subsets* (*matrix-A-R signs subsets*)))
assume *j* < *length*
(*find-nonzeros-from-input-vec* (*solve-for-lhs-R p qs subsets* (*matrix-A-R signs subsets*)))
assume *neq-hyp*: *i* ≠ *j*
assume *signs* ! (*find-nonzeros-from-input-vec* (*solve-for-lhs-R p qs subsets*
(*matrix-A-R signs subsets*)) ! *i*) =

signs ! (*find-nonzeros-from-input-vec* (*solve-for-lhs-R p qs subsets* (*matrix-A-R signs subsets*)) ! *j*)
have *h1*: *find-nonzeros-from-input-vec* (*solve-for-lhs-R p qs subsets* (*matrix-A-R signs subsets*)) ! *i* ≠ *find-nonzeros-from-input-vec* (*solve-for-lhs-R p qs subsets* (*matrix-A-R signs subsets*)) ! *j*
unfolding *find-nonzeros-from-input-vec-def* **using** *neq-hyp*
by (*metis* ‹*i* < *length* (*find-nonzeros-from-input-vec* (*solve-for-lhs-R p qs subsets* (*matrix-A-R signs subsets*)))› ‹*j* < *length* (*find-nonzeros-from-input-vec* (*solve-for-lhs-R p qs subsets* (*matrix-A-R signs subsets*)))› *distinct-conv-nth distinct-filter distinct-upt find-nonzeros-from-input-vec-def*)
then show *False* **using** *distinct-signs*
proof −
have *f1*: ∀ *p ns n*. ((*n*::*nat*) ∈ {*n* ∈ *set ns*. *p n*}) = (*n* ∈ *set ns* ∧ *n* ∈ *Collect p*)
by *simp*
then have *f2*: *filter* (λ*n*. *solve-for-lhs-R p qs subsets* (*matrix-A-R signs subsets*) $ *n* ≠ *0*) [*0*..<*dim-vec* (*solve-for-lhs-R p qs subsets* (*matrix-A-R signs subsets*))] ! *i* ∈ *set* [*0*..<*length signs*]
by (*metis* (*full-types*) ‹*i* < *length* (*find-nonzeros-from-input-vec* (*solve-for-lhs-R p qs subsets* (*matrix-A-R signs subsets*)))› *construct-lhs-matches-solve-for-lhs-R find-nonzeros-from-input-vec-def invertible-mat match nth-mem set-filter size-of-lhs-R*)
have *filter* (λ*n*. *solve-for-lhs-R p qs subsets* (*matrix-A-R signs subsets*) $ *n* ≠ *0*) [*0*..<*dim-vec* (*solve-for-lhs-R p qs subsets* (*matrix-A-R signs subsets*))] ! *j* ∈ *set* [*0*..<*length signs*]
using *f1* **by** (*metis* (*full-types*) ‹*j* < *length* (*find-nonzeros-from-input-vec* (*solve-for-lhs-R p qs subsets* (*matrix-A-R signs subsets*)))› *construct-lhs-matches-solve-for-lhs-R find-nonzeros-from-input-vec-def invertible-mat match nth-mem set-filter size-of-lhs-R*)
then show *?thesis*
using *f2* **by** (*metis* ‹*signs* ! (*find-nonzeros-from-input-vec* (*solve-for-lhs-R p qs subsets* (*matrix-A-R signs subsets*)) ! *i*) = *signs* ! (*find-nonzeros-from-input-vec* (*solve-for-lhs-R p qs subsets* (*matrix-A-R signs subsets*)) ! *j*)› *atLeastLessThan-iff distinct-conv-nth distinct-signs find-nonzeros-from-input-vec-def h1 set-upt*)
qed
qed
then have *distinct* (*take-indices signs* (*find-nonzeros-from-input-vec* (*solve-for-lhs-R p qs subsets* (*matrix-A-R signs subsets*))))
using *distinct-conv-nth* **by** *blast*
then show *?thesis*
using *get-signs-R-def reduction-signs-R-def reduction-signs-is-get-signs-R* **by** *auto*
qed
have *all-info-hyp*: *set* (*characterize-consistent-signs-at-roots p qs*) ⊆ *set*(*snd* (*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.member-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*: ∀ *x*. *List.member S x* ⟶ *x* < *length signs*
assumes *nonzero*: *p* ≠ *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*: ∀ *i* < *length* (*take-indices signs S*). (*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* < *length* (*take-indices signs S*)
have *i-lt*: *i* < *length* (*map* ((!) (*cols* (*alt-matrix-A-R signs subsets*))) *S*) **using** *asm*
apply (*auto*) **unfolding** *take-indices-def* **by** *auto*
have *h0*: *vec* (*length subsets*) (λ*j*. *z-R* (*subsets* ! *j*) (*map* ((!) *signs*) *S* ! *i*)) =
vec (*length subsets*) (λ*j*. *z-R* (*subsets* ! *j*) (*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-R signs subsets*))) *S*) ! *i* ∈ *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 in-set-member* **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* **apply** (*auto*)
by (*metis alt-matrix-char-R asm cols-nth dim-col-mat*(*1*) *in-set-member length-map mat-of-rows-list-def matrix-A-R-def nth-map nth-mem take-indices-def well-def-h*)
have *vec* (*length subsets*) (λ*j*. *z-R* (*subsets* ! *j*) (*signs* ! (*S* ! *i*))) = (*col* (*alt-matrix-A-R signs subsets*) (*S* !*i*))
by (*metis asm in-set-member length-map nth-mem signs-are-cols-R take-indices-def well-def-h*)
then have *vec* (*length subsets*) (λ*j*. *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*: ∀ *x*. *List.member S x* ⟶ *x* < *length signs*
assumes *nonzero*: *p* ≠ *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* ≥ *length signs*
assumes *well-def-h*: ∀ *x*. *List.member S x* ⟶ *x* < *length subsets*
assumes *nonzero*: *p* ≠ *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-row* (*take-rows-from-matrix* (*alt-matrix-A-R signs subsets*) *S*) = *length* (*take-indices subsets S*)
unfolding *take-rows-from-matrix-def* **apply** (*auto*) **unfolding** *take-indices-def* **by** *auto*
have *h0*: ∀ *i* < *length* (*take-indices subsets S*). (*row* (*alt-matrix-A-R signs* (*take-indices subsets S*)) *i* = *row* (*take-rows-from-matrix* (*alt-matrix-A-R signs subsets*) *S*) *i*)
proof *clarsimp*
fix *i*
assume *asm*: *i* < *length* (*take-indices subsets S*)
have *i-lt*: *i* < *length* (*map* ((!) (*rows* (*alt-matrix-A-R signs subsets*))) *S*) **using** *asm*
apply (*auto*) **unfolding** *take-indices-def* **by** *auto*
have *h0*: *row* (*take-rows-from-matrix* (*alt-matrix-A-R signs subsets*) *S*) *i* = *row* (*mat-of-rows* (*dim-col* (*alt-matrix-A-R signs subsets*)) (*map* ((!) (*rows* (*alt-matrix-A-R signs subsets*))) *S*)) *i*
unfolding *take-rows-from-matrix-def take-indices-def* **by** *auto*
have *dim*: (*map* ((!) (*rows* (*alt-matrix-A-R signs subsets*))) *S*) ! *i* ∈ *carrier-vec* (*dim-col* (*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*)
then have *lenh*: *dim-col* (*alt-matrix-A-R signs subsets*) ≤ *length* (*subsets*)
using *assms*
by *auto*
have *well-d*: *S* ! *i* < *length* (*subsets*) **using** *well-def-h*

```
      using i-lt in-set-member by fastforce
    have
     map-eq: (map ((!) (rows (alt-matrix-A-R signs subsets))) S) ! i = nth (rows
(alt-matrix-A-R signs subsets)) (S ! i)
      using i-lt by auto
   have nth (rows (alt-matrix-A-R signs subsets)) (S ! i) ∈ carrier-vec (dim-col
(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: row (mat-of-rows (dim-col (alt-matrix-A-R signs subsets)) (map ((!)
(rows (alt-matrix-A-R signs subsets))) S)) i
    = (row (alt-matrix-A-R signs subsets) (S ! i))
   using dim i-lt mat-of-rows-row[where i = i, where n = (dim-col (alt-matrix-A-R
signs subsets)),
       where vs = (map ((!) (rows (alt-matrix-A-R signs subsets))) S)]
    using alt-matrix-char-R in-set-member nth-mem well-def-h
    by fastforce
 have row (alt-matrix-A-R signs (take-indices subsets S) ) i = (row (alt-matrix-A-R
signs subsets) (S ! i))
    using subsets-are-rows-R
  proof −
    have f1: i < length S
      by (metis (no-types) asm length-map take-indices-def)
    then have List.member S (S ! i)
      by (meson in-set-member nth-mem)
    then show ?thesis
      using f1
     by (simp add: ‹⋀subsets signs. ∀ i<length subsets. row (alt-matrix-A-R signs
subsets) i = vec (length signs) (λj. z-R (subsets ! i) (signs ! j))› take-indices-def
well-def-h)
  qed
  then show (row (alt-matrix-A-R signs (take-indices subsets S) ) i =
    row (take-rows-from-matrix (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 ap-
ply (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*: *length subsets* ≥ *length signs*
assumes *well-def-h*: ∀ *x*. *List.member S x* ⟶ *x* < *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* (*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 *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-vec* = (*solve-for-lhs-R p qs subsets* (*matrix-A-R signs subsets*))
let *?red-mtx* = (*take-rows-from-matrix* (*reduce-mat-cols* (*matrix-A-R signs subsets*) *?lhs-vec*) (*rows-to-keep* (*reduce-mat-cols* (*matrix-A-R signs subsets*) *?lhs-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-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-vec*)) (*take-indices subsets* (*rows-to-keep* (*reduce-mat-cols ?A ?lhs-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*: ∀ *x*. *List.member* (*find-nonzeros-from-input-vec ?lhs-vec*) *x* ⟶*x* < *length* (*signs*)
using *h30 size-of-lhs-R*[*of p qs signs*]
unfolding *find-nonzeros-from-input-vec-def* **apply** (*auto*)
using *in-set-member* **by** *force*
have *h3b-a*: ∀ *x*. *List.member* (*find-nonzeros-from-input-vec ?lhs-vec*) *x* ⟶*x* < *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* (λ*i*. *solve-for-lhs-R p qs subsets* (*matrix-A-R signs subsets*) $ *i* ≠ *0*)
[*0*..<*length subsets*])
≤ *length subsets* **using** *length-filter-le*[**where** *P* = (λ*i*. *solve-for-lhs-R p qs subsets* (*matrix-A-R signs subsets*) $ *i* ≠ *0*),
where *xs* = [*0*..<*length subsets*]] *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* (λ*i*. *solve-for-lhs-R p qs subsets* (*matrix-A-R signs subsets*) $ *i* ≠ *0*)
[*0*..<*dim-vec* (*solve-for-lhs-R p qs subsets* (*matrix-A-R signs subsets*))])
≤ *length subsets* **using** *h3ba*
by *auto*
then have *h3b*: *length subsets* ≥ *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*: ∀ *x*. *List.member* (*rows-to-keep* (*reduce-mat-cols ?A ?lhs-vec*)) *x* ⟶ *x* < *length* (*subsets*)
proof *clarsimp*
fix *x*
assume *x-mem*: *List.member* (*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*))))) *x*
obtain *nn* :: *rat list list* ⇒ *nat list* ⇒ *nat* **where**
∀ *x2 x3*. (∃ *v4*. *v4* ∈ *set x3* ∧ ¬ *v4* < *length x2*) = (*nn x2 x3* ∈ *set x3* ∧ ¬ *nn x2 x3* < *length x2*)
by *moura*
then have *f4*: *nn signs* (*find-nonzeros-from-input-vec* (*solve-for-lhs-R p qs subsets* (*matrix-A-R signs subsets*))) ∈ *set* (*find-nonzeros-from-input-vec* (*solve-for-lhs-R p qs subsets* (*matrix-A-R signs subsets*))) ∧ ¬ *nn signs* (*find-nonzeros-from-input-vec* (*solve-for-lhs-R p qs subsets* (*matrix-A-R signs subsets*))) < *length signs* ∨ *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*)))
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*))) ∧ *x* ∈ *set* (*map snd* (*pivot-positions* (*gauss-jordan-single* (*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*)))^T)))
using *f4*
by (*metis h3a in-set-member rows-to-keep-def x-mem*)
thus *x* < *length* (*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* ≠ *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 (⋀*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*)))

proof −
fix *i*
fix *j*
assume *j-in*: $j \in$ *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*: $p \neq 0$
assumes *welldefined-signs1*: *well-def-signs* (*length qs*) *signs*
assumes *distinct-signs*: *distinct signs*
assumes *all-info*: *set* (*characterize-consistent-signs-at-roots p qs*) ⊆ *set*(*signs*)
assumes *match*: *satisfy-equation-R p qs subsets signs*
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*. $j \in$ *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*. $x \in$ *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* **using** *in-set-member* **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*: (⋀*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 *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*))))
⊆ *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*= *matrix-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* ≠ *0*
assumes *welldefined-signs1*: *well-def-signs* (*length qs*) *signs*
assumes *distinct-signs*: *distinct signs*

assumes *all-info*: *set* (*characterize-consistent-signs-at-roots p qs*) ⊆ *set*(*signs*)
assumes *match*: *satisfy-equation-R p qs subsets signs*
shows *invertible-mat* (*get-matrix-R* (*reduce-system-R p* (*qs*, ((*matrix-A-R signs subsets*), (*subsets*, *signs*)))))
proof −
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* ∈ *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-R diff-zero find-nonzeros-from-input-vec-def inv length-filter-le length-upt match size-of-lhs-R*)
then have *gt-eq-assm*: *dim-col ?step1-mat*^{T} ≥ *dim-row ?step1-mat*^{T}
by *simp*
have *det-h*: *det ?og-mat* ≠ *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* ∈ *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-assm 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*
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-assm obv rechar-take-cols reduce-mat-cols.simps*)
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* **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 *length* (*pivot-positions* (*gauss-jordan-single* (*?step1-mat*^{T}))) ≤ (*length* (*find-nonzeros-from-input-vec ?lhs*))
using *obv length-pivot-positions-dim-row*[**where** *A* = (*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* (*pivot-positions* (*gauss-jordan-single* (*?step1-mat*^{T}))) ≤ *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-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* ((!) *signs*) (*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*: ∀ *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* : ∀ *x*∈ *set*(*signs*). *length x* = *length qs*
assumes *sat-eq*: *satisfy-equation-R p qs subsets signs*
assumes *inv-mat*: *invertible-mat* (*matrix-A-R signs subsets*)
shows ∀ *x* ∈ *set*(*reduction-signs-R p qs signs subsets* (*matrix-A-R signs subsets*)).

length x = *length qs*
unfolding *reduction-signs-def* **using** *take-indices-lem*
by (*smt* (*verit*) *atLeastLessThan-iff construct-lhs-matches-solve-for-lhs-R filter-is-subset find-nonzeros-from-input-vec-def in-set-conv-nth in-set-member inv-mat length-init reduction-signs-R-def sat-eq set-upt size-of-lhs-R subset-code*(*1*))

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*: (⋀*j*. *j* ∈ *set* (*find-nonzeros-from-input-vec* (*solve-for-lhs-R p qs subsets* (*matrix-A-R signs subsets*))) ⟹
j < *length signs*)
proof −
fix *j*
assume *j* ∈ *set* (*find-nonzeros-from-input-vec* (*solve-for-lhs-R p qs subsets* (*matrix-A-R signs subsets*)))
show *j* < *length signs* **using** *solve-construct size-of-lhs-R*
by (*metis* ‹*j* ∈ *set* (*find-nonzeros-from-input-vec* (*solve-for-lhs-R p qs 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*: *List.member* (*reduction-subsets-R p qs signs subsets* (*matrix-A-R signs subsets*)) (*a*, *b*)
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 member-def set-upt subset-eq* **apply** (*auto*)
by (*smt* (*verit*, *best*) *atLeastLessThan-iff in-set-member mem-Collect-eq set-filter set-upt*)
have *h3c*: $\forall x$. *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*)))) $x \longrightarrow x <$ *length* (*subsets*)
proof *clarsimp*
fix x
assume *x-mem*: *List.member* (*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*))))) x
show $x <$ *length* (*subsets*) **using** *x-mem* **unfolding** *rows-to-keep-def* **using** *pivot-positions*
using *dim-row-matrix-A h3a in-set-member nonzero reduce-system-matrix-signs-helper-R rows-to-keep-def rows-to-keep-lem*
apply (*auto*)
by (*smt* (*verit*, *best*) *List.member-def dim-row-matrix-A-R rows-to-keep-def rows-to-keep-lem*)
qed
then show $y <$ *length subsets* **using** *in-set-member* **apply** (*auto*)
using *in-set-2 solve-construct* **by** *fastforce*
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 in-set-member length-init*
by (*metis fst-conv reduction-subsets-R-def snd-conv*)
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* ≠ *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*
unfolding *get-matrix-R-def combine-systems-R.simps smash-systems-R-def get-signs-R-def get-subsets-R-def*
apply *simp*
apply (*subst matrix-construction-is-kronecker-product-R*[*of subsets1 - signs1 signs2 subsets2*])

apply (*metis Ball-set all-list-constr-R-def in-set-member 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* **by** *auto*
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* ≠ *0*
assumes *nontriv*: *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*))) (*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* (*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*) ⊆ *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* **apply** (*auto*)
apply (*smt* (*verit*, *best*) *Ball-set in-set-member member-rec*(*1*) *member-rec*(*2*) *prod.inject*)
by (*smt* (*verit*, *ccfv-threshold*) *Ball-set in-set-member member-rec*(*1*) *member-rec*(*2*) *prod.inject*)
have *h2*: *well-def-signs* (*length qs*) [[*1*],[−*1*]]
unfolding *well-def-signs-def* **using** *len1 in-set-member*
by *auto*
have *h3*: *distinct* ([[*1*],[*0*],[−*1*]]::*rat list list*)
unfolding *distinct-def* **using** *in-set-member* **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*) ⊆ *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*)) ⟶ *satisfies-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* ⟶ (*p* ≠ *0* ∧ (*length qs* > *0*)) ⟶ (*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* ⟶ (*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*))
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* ≠ *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*: ∃ (*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*) ⊆ *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** *invertible-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*)) ⊆ *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*: (⋀*j*. *j* ∈ *set* (*get-signs-R* (*calculate-data-R p qs1*)) ⟹ *length j* = (*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*: (⋀*l i*. *l* ∈ *set* (*get-subsets-R*(*calculate-data-R p qs1*)) ⟹ (*i* ∈ *set* (*fst l*) ⟶ *i* < (*length qs1*)) ∧ (*i* ∈ *set* (*snd l*) ⟶ *i* < (*length qs1*)))
unfolding *all-list-constr-R-def list-constr-def*
using *in-set-member*
by (*metis in-set-conv-nth list-all-length*)
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 $p \neq 0$
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 (λ*r*. []) ‘ *set* (*characterize-root-list-p p*) = (λ*r*. *drop* (*Suc 0*) [*1*::*rat*]) ‘ *set* (*characterize-root-list-p p*)
by *force*
then show (λ*r*. []) ‘ *set* (*characterize-root-list-p p*) = *drop* (*Suc 0*) ‘ (λ*r*. [*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 $p \neq 0$
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 $q \in set\ qs$
assumes $q \neq 0$
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*: $q = a \vee q \in set\ qs$ **by** *auto*
have *c1*: $q = a \Longrightarrow q\ dvd\ prod\text{-}list\text{-}var\ (a\#qs)$
proof −
assume $q = a$
then have $prod\text{-}list\text{-}var\ (a\#qs) = q*(prod\text{-}list\text{-}var\ qs)$ **using** *Cons.prems*
unfolding *prod-list-var-def* **by** *auto*
then show *?thesis* **using** *prod-list-var-nonzero*[*of qs*] **by** *auto*
qed
have *c2*: $q \in set\ qs \longrightarrow q\ dvd\ prod\text{-}list\text{-}var\ qs$
using *Cons.prems 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 $check\text{-}all\text{-}const\text{-}deg\ l = True \longleftrightarrow (\forall\, p \in set(l).\ degree\ p = 0)$
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 $(poly\text{-}f\ qs) \neq 0$
proof −
have *eo*: $(\forall\, p \in set\ qs.\ degree\ p = 0) \vee (\exists\, p \in set\ qs.\ degree\ p > 0)$
by *auto*
have *c1*: $(\forall\, p \in set\ qs.\ degree\ p = 0) \longrightarrow (poly\text{-}f\ qs) \neq 0$

```
  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 ∈ set qs
  assume deg-q: 0 < degree q
  assume contrad: poly-f qs = 0
 have nonconst: check-all-const-deg qs = False using deg-q check-all-const-deg-prop
     q-in by auto
  have h1: prod-list-var qs ≠ 0 using prod-list-var-nonzero by auto
  then have degree (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 ≠ 0 using Cons.prems by auto
   have q-ins: q ∈ set (a # qs)  using Cons.prems by auto
   then have q = a ∨ q ∈ set qs by auto
   then have eo:  q = a ∨ List.member qs q using in-set-member[of q qs]
    by auto
   have degq: degree q > 0 using Cons.prems by auto
   have h2: (prod-list (a # qs)) = a* (prod-list qs)
    by auto
   have isa: q = a ⟶ 0 < degree (prod-list-var (a # qs))
    using h2 degree-mult-eq-0[where p = q, where q = prod-list-var qs]
     Cons.prems by auto
   have inl: List.member qs q ⟶ 0 < degree (prod-list-var (a # qs))
   proof −
    have nonzprod: prod-list-var (a # qs) ≠ 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 = prod-list-var
(a # qs)] nonzprod degq
     by auto
   qed
   then show ?case using eo isa by auto
  qed
  then have h2: pderiv (prod-list-var qs) ≠ 0 using pderiv-eq-0-iff[where p =
prod-list-var qs]
   by auto
  then have pderiv (prod-list-var qs) * prod-list-var qs ≠ 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 $\forall x1.\ \forall x2.\ ((x1 < x2 \wedge (\exists q1 \in set\ (qs).\ q1 \neq 0 \wedge (poly\ q1\ x1) = 0) \wedge (\exists q2 \in set(qs).\ q2 \neq 0 \wedge (poly\ q2\ x2) = 0)) \longrightarrow (\exists q.\ x1 < q \wedge q < x2 \wedge poly\ (poly\text{-}f\ qs)\ q = 0))$
proof *clarsimp*
fix *x1*:: *real*
fix *x2*:: *real*
fix *q1*:: *real poly*
fix *q2*:: *real poly*
assume *x1 < x2*
assume *q1-in*: *q1 ∈ set qs*
assume *q1-0*: *poly q1 x1 = 0*
assume *q1-nonz*: *q1 ≠ 0*
assume *q2-in*: *q2 ∈ set qs*
assume *q2-0*: *poly q2 x2 = 0*
assume *q2-nonz*: *q2 ≠ 0*
have *prod-z-x1*: *poly (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*: *poly (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 $\exists w{>}x1.\ w < x2 \wedge poly\ (pderiv\ (prod\text{-}list\text{-}var\ qs))\ w = 0$
using *Rolle-pderiv*[**where** *q = prod-list-var qs*] *prod-z-x1 prod-z-x2*
using ‹*x1 < x2*› **by** *blast*
then obtain *w* **where** *w-def*: $w > x1 \wedge w < x2 \wedge poly\ (pderiv\ (prod\text{-}list\text{-}var\ qs))\ w = 0$
by *auto*
then have *poly (poly-f qs) w = 0*
unfolding *poly-f-def* **using** *non-const*
by *simp*
then show $\exists q{>}x1.\ q < x2 \wedge poly\ (poly\text{-}f\ qs)\ q = 0$
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 *set (find-consistent-signs-R qs) = consistent-sign-vectors-R qs UNIV*
proof −
have *poly-f-is*: *poly-f qs = [:0, 1:]* **unfolding** *poly-f-def* **using** *assms*

by *auto*
have *same*: *set (find-consistent-signs-at-roots-R [:0, 1:] qs) = set (characterize-consistent-signs-at-roots [:0, 1:] qs)* **using** *find-consistent-signs-at-roots-R*[*of [:0, 1:] qs*]

by *auto*
have *rech*: (*sorted-list-of-set* {*x*. *poly* ([:*0*, *1*:]::*real poly*) *x* = *0*}) = [*0*] **by** *auto*
have *alldeg0*: (∀ *p* ∈ *set qs*. *degree p* = *0*) **using** *non-const check-all-const-deg-prop*
by *auto*
then have *allconst*: ∀ *p* ∈ *set qs*. (∃ (*k*::*real*). *p* = [:*k*:])
apply (*auto*)
by (*meson degree-eq-zeroE*)
then have *allconstvar*: ∀ *p* ∈ *set qs*. ∀ (*x*::*real*). ∀ (*y*::*real*). *poly p x* = *poly p y*
by *fastforce*
have *e1*: *set* (*remdups* (*map* (*signs-at qs*) [*0*])) ⊆
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*: *consistent-sign-vectors-R qs UNIV* ⊆ *set* (*remdups* (*map* (*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 *set* (*remdups* (*map* (*signs-at qs*) [*0*])) =
consistent-sign-vectors-R qs UNIV
using *e1 e2* **by** *auto*
then have *set* (*characterize-consistent-signs-at-roots* [:*0*, *1*:] *qs*) = *consistent-sign-vectors-R qs 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*: ¬ (*List.member l x*)
assumes *lt-a*: *x* > (*l* ! *0*)
assumes *b-lt*: *x* < (*l* ! (*length l* − *1*))
shows (∃ *n*. *n* < *length l* − *1* ∧ *x* > *l* ! *n* ∧ *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* ≥ *1* **using** *Cons.prems*(*1*)
by (*metis One-nat-def Suc-eq-plus1 list.size*(*4*) *not-le not-less-eq*)

have *len-one*: *length l = 1* $\Longrightarrow$ ($\exists$ *n. n < length* (*a#l*) *− 1 ∧ x > (a#l) ! n ∧ x < (a#l) !(n+1)*)
proof −
assume *len-is*: *length l = 1*
then have *x > (a#l) ! 0 ∧ x < (a#l) !1* **using** *Cons.prems(4) Cons.prems(5)*
by *auto*
then show ($\exists$ *n. n < length* (*a#l*) *− 1 ∧ x > (a#l) ! n ∧ x < (a#l) !(n+1)*)
using *len-is* **by** *auto*
qed
have *len-gt*: *length l > 1* $\Longrightarrow$ ($\exists$ *n. n < length* (*a#l*) *− 1 ∧ x > (a#l) ! n ∧ x < (a#l) !(n+1)*)
proof −
assume *len-gt-one*: *length l > 1*
have *eo*: *x ≠ l ! 0* **using** *Cons.prems(3)* **apply** (*auto*)
by (*metis One-nat-def Suc-lessD in-set-member len-gt-one member-rec(1) nth-mem*)
have *c1*: *x < l ! 0* $\Longrightarrow$ ($\exists$ *n. n < length* (*a#l*) *− 1 ∧ x > (a#l) ! n ∧ 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* $\Longrightarrow$ ($\exists$ *n. n < length* (*a#l*) *− 1 ∧ x > (a#l) ! n ∧ x < (a#l) !(n+1)*)
proof −
assume *asm*: *x > l ! 0*
have *xlt-1*: *x < l ! (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 *¬ List.member l x* **using** *Cons.prems(3)*
by (*meson member-rec(1)*)
then have *∃ n<length l − 1. l ! n < x ∧ 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 < length* (*a#l*) *− 1 ∧ x > (a#l) ! n ∧ 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* = (*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*:])))
shows *consistent-sign-vectors-R qs UNIV* = *consistent-sign-vectors-R qs* (*set S*)
proof −
let *?zer-list* = *sorted-list-of-set* {(*x*::*real*). (∃ *q* ∈ *set*(*qs*). (*q* ≠ *0* ∧ *poly q x* = *0*))} :: *real list*
have *strict-sorted-h*: *sorted-wrt* (<) *?zer-list* **using** *sorted-sorted-list-of-set strict-sorted-iff* **by** *auto*
have *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*:])
unfolding *poly-f-def* **using** *is-not-const* **by** *auto*
then have *set-S-char*: *set S* = ({*x*. *poly* (*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* ⊆ *consistent-sign-vectors-R qs* (*set S*)
proof *clarsimp*
fix *x*
assume *x* ∈ *consistent-sign-vectors-R qs UNIV*
then have ∃ *y*. *x* = (*consistent-sign-vec qs y*) **unfolding** *consistent-sign-vectors-R-def* **by** *auto*
then obtain *y* **where** *y-prop*: *x* = *consistent-sign-vec qs y* **by** *auto*
then have ∃ *k* ∈ (*set S*). *consistent-sign-vec qs k* = *consistent-sign-vec qs y*
proof −
have *c1*: (∃ *q* ∈ (*set qs*). *q* ≠ *0* ∧ *poly q y* = *0*) ⟹ (∃ *k* ∈ (*set S*). *consistent-sign-vec qs k* = *consistent-sign-vec qs y*)
proof −
assume (∃ *q* ∈ (*set qs*). *q* ≠ *0* ∧ *poly q y* = *0*)
then obtain *q* **where** *q* ∈ (*set qs*) ∧ *q* ≠ *0* ∧ *poly q y* = *0* **by** *auto*
then have *poly* (*prod-list-var qs*) *y* = *0*
using *q-dvd-prod-list-var-prop*[*of q qs*] **by** *auto*
then have *poly* (*pderiv* (*prod-list-var qs*) ∗ (*prod-list-var qs*)∗([:−(*crb* (*prod-list-var qs*)),*1*:]) ∗ ([:(*crb* (*prod-list-var qs*)),*1*:])) *y* = *0*
by *auto*
then have *y* ∈ (*set S*)
using *s-is* **unfolding** *characterize-root-list-p-def*
proof −
have *y* ∈ {*r*. *poly* (*pderiv* (*prod-list-var qs*) ∗ (*prod-list-var qs*)∗([:−(*crb* (*prod-list-var qs*)),*1*:]) ∗ ([:(*crb* (*prod-list-var qs*)),*1*:])) *r* = *0*}

using ‹*poly* (*pderiv* (*prod-list-var qs*) ∗ (*prod-list-var qs*)∗([:−(*crb* (*prod-list-var qs*)),1:]) ∗ ([:(*crb* (*prod-list-var qs*)),1:])) *y* = *0*› **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 ∃ *k* ∈ (*set S*). *consistent-sign-vec qs k* = *consistent-sign-vec qs y*
by *auto*
qed
have *len-gtz-prop*: *length ?zer-list* > *0* ⟶
((∃ *w*. *w* < *length ?zer-list* ∧ *y* = *?zer-list* ! *w*) ∨
(*y* < *?zer-list* ! *0*) ∨
(*y* > *?zer-list* ! (*length ?zer-list* − *1*)) ∨
(∃ *k* < (*length ?zer-list* − *1*). *y* > *?zer-list* ! *k* ∧ *y* < *?zer-list* ! (*k*+*1*)))
proof −
let *?c* = (∃ *w*. *w* < *length ?zer-list* ∧ *y* = *?zer-list* ! *w*) ∨
(*y* < *?zer-list* ! *0*) ∨
(*y* > *?zer-list* ! (*length ?zer-list* − *1*)) ∨
(∃ *k* < (*length ?zer-list* − *1*). *y* > *?zer-list* ! *k* ∧ *y* < *?zer-list* ! (*k*+*1*))
have *lis1*: *length ?zer-list* = *1* ⟹ *?c*
by *auto*
have *h1*: ¬(∃ *w*. *w* < *length ?zer-list* ∧ *y* = *?zer-list* ! *w*) ⟹ ¬ (*List.member ?zer-list y*)
by (*metis* (*no-types*, *lifting*) *in-set-conv-nth in-set-member*)
have *h2*: (*length ?zer-list* > *0* ∧ ¬(∃ *w*. *w* < *length ?zer-list* ∧ *y* = *?zer-list* ! *w*) ∧ ¬ (*y* < *?zer-list* ! *0*)) ⟹ *y* > *?zer-list* ! *0*
by *auto*
have *h3*: (*length ?zer-list* > *1* ∧ ¬(∃ *w*. *w* < *length ?zer-list* ∧ *y* = *?zer-list* ! *w*) ∧ ¬ (*y* > *?zer-list* ! (*length ?zer-list* − *1*))) ⟹
y < *?zer-list* ! (*length ?zer-list* − *1*)
apply (*auto*)
by (*smt* (*z3*) *diff-Suc-less gr-implies-not0 not-gr-zero*)
have *length ?zer-list* > *1* ∧ ¬(∃ *w*. *w* < *length ?zer-list* ∧ *y* = *?zer-list* ! *w*) ∧ ¬ (*y* < *?zer-list* ! *0*) ∧ ¬ (*y* > *?zer-list* ! (*length ?zer-list* − *1*))
⟹ (∃ *k* < (*length ?zer-list* − *1*). *y* > *?zer-list* ! *k* ∧ *y* < *?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*: *length ?zer-list* > *1* ⟹ *?c*
by *auto*
then show *?thesis* **using** *lis1 lgt1*
by (*smt* (*z3*) *diff-is-0-eq′ not-less*)
qed
have *neg-crb-in*: (− *crb* (*prod-list-var qs*)) ∈ *set S*
using *set-S-char poly-f-is* **by** *auto*
have *pos-crb-in*: (*crb* (*prod-list-var qs*)) ∈ *set S*
using *set-S-char poly-f-is* **by** *auto*
have *set-S-nonempty*: *set S* ≠ {} **using** *neg-crb-in* **by** *auto*

have *finset*: *finite* $\{x.\ \exists\, q{\in}set\ qs.\ q \neq 0 \wedge poly\ q\ x = 0\}$
proof −
have $\forall\, q \in set\ qs.\ \ q{\neq}\ 0 \longrightarrow finite\ \{x.\ poly\ q\ x = 0\}$
using *poly-roots-finite* **by** *auto*
then show *?thesis* **by** *auto*
qed
have *c2*: $\neg(\exists\, q \in (set\ qs).\ q \neq 0 \wedge poly\ q\ y = 0) \Longrightarrow \exists\ k \in (set\ S).$ *consistent-sign-vec qs k = consistent-sign-vec qs y*
proof −
assume $\neg(\exists\, q \in (set\ qs).\ q \neq 0 \wedge poly\ q\ y = 0)$
have *c-c1*: *length ?zer-list = 0* $\Longrightarrow\ \exists\ k \in (set\ S).$ *consistent-sign-vec qs k = consistent-sign-vec qs y*
proof −
assume *length ?zer-list = 0*
then have $\forall\, q \in set\ (qs).\ \forall\ (x{::}\ real).\ \forall\, (y{::}real).$ *squash* $(poly\ q\ x) =$ *squash* $(poly\ q\ y)$
proof *clarsimp*
fix *q x y*
assume *czer*: *card* $\{x.\ \exists\, q{\in}set\ qs.\ q \neq 0 \wedge poly\ q\ x = 0\} = 0$
assume *qin*: $q \in set\ qs$
have *fin-means-empty*: $\{x.\ \exists\, q{\in}set\ qs.\ q \neq 0 \wedge poly\ q\ x = 0\} = \{\}$
using *finset czer*
by *auto*
have *qzer*: $q = 0 \Longrightarrow$ *squash* $(poly\ q\ x) =$ *squash* $(poly\ q\ y)$ **by** *auto*
have *qnonz*: $q \neq 0 \Longrightarrow$ *squash* $(poly\ q\ x) =$ *squash* $(poly\ q\ y)$
proof −
assume *qnonz*: $q \neq 0$
then have *noroots*: $\{x.\ poly\ q\ x = 0\} = \{\}$ **using** *qin finset*
using *Collect-empty-eq fin-means-empty* **by** *auto*
have *nonzsq1*: *squash* $(poly\ q\ x) \neq 0$ **using** *fin-means-empty qnonz czer qin*
unfolding *squash-def* **by** *auto*
then have *eo*: $(poly\ q\ x) > 0 \vee (poly\ q\ x) < 0$ **unfolding** *squash-def*
apply (*auto*)
by *presburger*
have *eo1*: $poly\ q\ x > 0 \Longrightarrow poly\ q\ y > 0$
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*: $poly\ q\ x < 0 \Longrightarrow poly\ q\ y < 0$
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 *squash* $(poly\ q\ x) =$ *squash* $(poly\ q\ y)$
using *eo eo1 eo2* **unfolding** *squash-def* **by** *auto*
qed
show *squash* $(poly\ q\ x) =$ *squash* $(poly\ q\ y)$
using *qzer qnonz*
by *blast*
qed

```
          then have ∀ q ∈ set (qs). squash (poly q y) = squash (poly q (− crb
(prod-list-var qs)))
           by auto
          then show ∃ k ∈ (set S). consistent-sign-vec qs k = 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: length ?zer-list > 0 ⟹ ∃ k ∈ (set S). consistent-sign-vec qs k
= consistent-sign-vec qs y
        proof −
          assume lengt: length ?zer-list > 0
          let ?t = ∃ k ∈ (set S). consistent-sign-vec qs k = consistent-sign-vec qs y
          have sg1: (∃ w. w < length ?zer-list ∧ y = ?zer-list ! w) ⟹ ?t
          proof −
            assume (∃ w. w < length ?zer-list ∧ y = ?zer-list ! w)
            then obtain w where w-prop: w < length ?zer-list ∧ y = ?zer-list ! w
by auto
            then have  y ∈ {x. ∃ q∈set qs. q ≠ 0 ∧ poly q x = 0}
              using finset set-sorted-list-of-set[of {x. ∃ q∈set qs. q ≠ 0 ∧ poly q x =
0}]
              by (smt (verit, best) nth-mem)
            then have y ∈ {x. poly (poly-f qs) x = 0} using poly-f-is
              using ‹¬ (∃ q∈set qs. q ≠ 0 ∧ poly q y = 0)› by blast
            then show ?thesis using set-S-char
              by blast
          qed
          have sg2: (y < ?zer-list ! 0) ⟹ ?t
          proof −
            assume ylt: y < ?zer-list ! 0
            have ynonzat-some-qs: ∀ q ∈ (set qs). q ≠ 0 ⟶ poly q y ≠ 0
            proof clarsimp
              fix q
              assume q-in: q ∈ set qs
              assume qnonz: q ≠ 0
              assume poly q y = 0
              then have y ∈ {x. ∃ q∈set qs. q ≠ 0 ∧ poly q x = 0}
                using q-in qnonz by auto
              then have List.member ?zer-list y
            by (smt (verit, best) finset in-set-member mem-Collect-eq set-sorted-list-of-set)

              then have y ≥ ?zer-list ! 0 using strict-sorted-h
                 using ‹¬ (∃ q∈set qs. q ≠ 0 ∧ poly q y = 0)› ‹poly q y = 0› q-in
qnonz by blast
              then show False using ylt
                by auto
            qed
```

let *?ncrb* = (− *crb* (*prod-list-var qs*))
have ∀ *x* ∈ {*x*. ∃ *q*∈*set qs*. *q* ≠ *0* ∧ *poly q x* = *0*}. *poly* (*prod-list-var qs*) *x* = *0*
using *q-dvd-prod-list-var-prop*
by *fastforce*
then have *poly* (*prod-list-var qs*) (*sorted-list-of-set* {*x*. ∃ *q*∈*set qs*. *q* ≠ *0* ∧ *poly q x* = *0*} ! *0*) = *0*
using *finset set-sorted-list-of-set*
by (*metis* (*no-types*, *lifting*) *lengt nth-mem*)
then have *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*: ∀ *q* ∈ (*set qs*). *q* = *0* ⟶ *squash* (*poly q ?ncrb*) = *squash* (*poly q y*)
by *auto*
have ∀ *q* ∈ (*set qs*). *q* ≠ *0* ⟶ *squash* (*poly q ?ncrb*) = *squash* (*poly q y*)
proof *clarsimp*
fix *q*
assume *q-in*: *q* ∈ *set qs*
assume *qnonz*: *q* ≠ *0*
have *nonzylt*:¬(∃ *x* ≤ *y*. *poly q x* = *0*)
proof *clarsimp*
fix *x*
assume *xlt*: *x* ≤ *y*
assume *poly q x* = *0*
then have *x* ∈ {*x*. ∃ *q*∈*set qs*. *q* ≠ *0* ∧ *poly q x* = *0*}
using *q-in qnonz* **by** *auto*
then have *List.member ?zer-list x*
by (*smt* (*verit*, *best*) *finset in-set-member mem-Collect-eq set-sorted-list-of-set*)
then have *x* ≥ *?zer-list* ! *0* **using** *strict-sorted-h*
by (*metis* (*no-types*, *lifting*) *gr-implies-not0 in-set-conv-nth in-set-member not-less sorted-iff-nth-mono sorted-list-of-set*(*2*))
then show *False* **using** *xlt ylt*
by *auto*
qed
have *nonzncrb*:¬(∃ *x* ≤ (*real-of-int ?ncrb*). *poly q x* = *0*)
proof *clarsimp*
fix *x*
assume *xlt*: *x* ≤ − *real-of-int* (*crb* (*prod-list-var qs*))
assume *poly q x* = *0*
then have *x* ∈ {*x*. ∃ *q*∈*set qs*. *q* ≠ *0* ∧ *poly q x* = *0*}
using *q-in qnonz* **by** *auto*
then have *List.member ?zer-list x*
by (*smt* (*verit*, *best*) *finset in-set-member mem-Collect-eq set-sorted-list-of-set*)
then have *x* ≥ *?zer-list* ! *0* **using** *strict-sorted-h*
by (*metis* (*no-types*, *lifting*) *gr-implies-not0 in-set-conv-nth in-set-member not-less sorted-iff-nth-mono sorted-list-of-set*(*2*))

```
            then show False using xlt ncrblt
              by auto
          qed
          have c1: (poly q ?ncrb) > 0 ⟹ (poly q y) > 0
          proof −
            assume qncrbgt: (poly q ?ncrb) > 0
            then have eq: ?ncrb = y ⟹ poly q y > 0 by auto
                 have gt:  ?ncrb > y ⟹ poly q y > 0 using qncrbgt qnonz
poly-IVT-pos[of y ?ncrb q] poly-IVT-neg[of ?ncrb y q] nonzncrb nonzylt
              apply (auto)
              by (meson less-eq-real-def linorder-neqE-linordered-idom)
            have lt: ?ncrb < y ⟹ poly q y > 0 using qncrbgt
              using qnonz poly-IVT-pos[of y ?ncrb q] poly-IVT-neg[of ?ncrb y q]
nonzncrb nonzylt
              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 ⟹ (poly q y) < 0
           using poly-IVT-pos[of ?ncrb y q] poly-IVT-neg[of y ?ncrb q] nonzncrb
nonzylt
            apply (auto)
            by (metis less-eq-real-def linorder-neqE-linordered-idom)
          have eo: (poly q ?ncrb) > 0 ∨ (poly q ?ncrb) < 0
            using nonzncrb
            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 ∀ q ∈ (set qs). squash (poly q ?ncrb) = squash (poly q 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)) ⟹ ?t
      proof −
        assume ygt: y > ?zer-list ! (length ?zer-list − 1)
        have ynonzat-some-qs: ∀ q ∈ (set qs). q ≠ 0 ⟶ poly q y ≠ 0
        proof clarsimp
          fix q
          assume q-in: q ∈ set qs
          assume qnonz: q ≠ 0
          assume poly q y = 0
          then have y ∈ {x. ∃ q∈set qs. q ≠ 0 ∧ poly q x = 0}
```

using *q-in qnonz* **by** *auto*
then have *List.member ?zer-list y*
by (*smt* (*verit*, *best*) *finset in-set-member mem-Collect-eq set-sorted-list-of-set*)

then have $y \leq$ *?zer-list* ! (*length ?zer-list* − *1*) **using** *strict-sorted-h*
using ‹¬ (∃ *q*∈*set qs*. *q* ≠ *0* ∧ *poly q y* = *0*)› ‹*poly q y* = *0*› *q-in qnonz* **by** *blast*
then show *False* **using** *ygt*
by *auto*
qed
let *?crb* = *crb* (*prod-list-var qs*)
have ∀ *x* ∈ {*x*. ∃ *q*∈*set qs*. *q* ≠ *0* ∧ *poly q x* = *0*}. *poly* (*prod-list-var qs*) *x* = *0*
using *q-dvd-prod-list-var-prop*
by *fastforce*
then have *poly* (*prod-list-var qs*) (*sorted-list-of-set* {*x*. ∃ *q*∈*set qs*. *q* ≠ *0* ∧ *poly q x* = *0*} ! *0*) = *0*
using *finset set-sorted-list-of-set*
by (*metis* (*no-types*, *lifting*) *lengt nth-mem*)
then have *crbgt*: *?crb* > *?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*) ‹∀ *x*∈{*x*. ∃ *q*∈*set qs*. *q* ≠ *0* ∧ *poly q x* = *0*}. *poly* (*prod-list-var qs*) *x* = *0*› *diff-less finset lengt less-numeral-extra*(*1*) *nth-mem set-sorted-list-of-set*)
have *qzerh*: ∀ *q* ∈ (*set qs*). *q* = *0* ⟶ *squash* (*poly q ?crb*) = *squash* (*poly q y*)
by *auto*
have ∀ *q* ∈ (*set qs*). *q* ≠ *0* ⟶ *squash* (*poly q ?crb*) = *squash* (*poly q y*)
proof *clarsimp*
fix *q*
assume *q-in*: *q* ∈ *set qs*
assume *qnonz*: *q* ≠ *0*
have *nonzylt*:¬(∃ *x* ≥ *y*. *poly q x* = *0*)
proof *clarsimp*
fix *x*
assume *xgt*: *x* ≥ *y*
assume *poly q x* = *0*
then have *x* ∈ {*x*. ∃ *q*∈*set qs*. *q* ≠ *0* ∧ *poly q x* = *0*}
using *q-in qnonz* **by** *auto*
then have *List.member ?zer-list x*
by (*smt* (*verit*, *best*) *finset in-set-member mem-Collect-eq set-sorted-list-of-set*)
then have *x* ≤ *?zer-list* ! (*length ?zer-list* − *1*) **using** *strict-sorted-h*
by (*metis* (*no-types*, *lifting*) *One-nat-def Suc-leI Suc-pred diff-Suc-less in-set-conv-nth in-set-member lengt not-less sorted-iff-nth-mono sorted-list-of-set*(*2*))

then show *False* **using** *xgt ygt*
by *auto*

qed
have *nonzcrb*:¬(∃ *x* ≥ (*real-of-int ?crb*). *poly q x* = *0*)
proof *clarsimp*
fix *x*
assume *xgt*: *x* ≥ *real-of-int* (*crb* (*prod-list-var qs*))
assume *poly q x* = *0*
then have *x* ∈ {*x*. ∃ *q*∈*set qs*. *q* ≠ *0* ∧ *poly q x* = *0*}
using *q-in qnonz* **by** *auto*
then have *List.member ?zer-list x*
by (*smt* (*verit*, *best*) *finset in-set-member mem-Collect-eq set-sorted-list-of-set*)
then have *x* ≤ *?zer-list* ! (*length ?zer-list* − *1*) **using** *strict-sorted-h*
by (*meson* ‹∀ *x*∈{*x*. ∃ *q*∈*set qs*. *q* ≠ *0* ∧ *poly q x* = *0*}. *poly* (*prod-list-var qs*) *x* = *0*› ‹*x* ∈ {*x*. ∃ *q*∈*set qs*. *q* ≠ *0* ∧ *poly q x* = *0*}› *crb-lem-pos not-less prod-list-var-nonzero xgt*)
then show *False* **using** *xgt crbgt*
by *auto*
qed
have *c1*: (*poly q ?crb*) > *0* ⟹ (*poly q y*) > *0*
proof −
assume *qcrbgt*: (*poly q ?crb*) > *0*
then have *eq*: *?crb* = *y* ⟹ *poly q y* > *0* **by** *auto*
have *gt*: *?crb* > *y* ⟹ *poly q y* > *0* **using** *qcrbgt qnonz poly-IVT-pos*[*of y ?crb q*] *poly-IVT-neg*[*of ?crb y q*] *nonzcrb nonzylt*
apply (*auto*)
by (*meson less-eq-real-def linorder-neqE-linordered-idom*)
have *lt*: *?crb* < *y* ⟹ *poly q y* > *0* **using** *qcrbgt*
using *qnonz poly-IVT-pos*[*of y ?crb q*] *poly-IVT-neg*[*of ?crb y q*] *nonzcrb nonzylt*
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 ?crb*) < *0* ⟹ (*poly q y*) < *0*
using *poly-IVT-pos*[*of ?crb y q*] *poly-IVT-neg*[*of y ?crb q*] *nonzcrb nonzylt*
apply (*auto*)
by (*metis less-eq-real-def linorder-neqE-linordered-idom*)
have *eo*: (*poly q ?crb*) > *0* ∨ (*poly q ?crb*) < *0*
using *nonzcrb*
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 ∀ *q* ∈ (*set qs*). *squash* (*poly q ?crb*) = *squash* (*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 (*metis* (*mono-tags*, *lifting*) *finset in-set-member mem-Collect-eq set-sorted-list-of-set*)
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 ⟹ squash (poly q s) = squash (poly q y)
            proof −
              assume slt: s < y
              then have ∄x. List.member ?zer-list x ∧ s ≤ x ∧ x ≤ y
              using sorted-list-lemma[of s y 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 ∧ s ≤ x ∧ x ≤ y using q-in qnonz
                by (metis (mono-tags, lifting) finset in-set-member mem-Collect-eq
set-sorted-list-of-set)
              then  have c1: poly q s ≠ 0 using s-prop q-in qnonz
                by (metis (mono-tags, lifting) ‹s < y› less-eq-real-def )
              have c2: poly q s > 0 ⟹ poly q y > 0
                using poly-IVT-pos poly-IVT-neg nox
                by (meson ‹s < y› 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 ‹s < y› less-eq-real-def linorder-neqE-linordered-idom)
              show ?thesis using c1 c2 c3 unfolding squash-def
                by auto
            qed
            have s = y ⟹ squash (poly q s) = squash (poly q y)
              by auto
            then show squash (poly q s) = squash (poly q y)
              using sgt slt
              by (meson linorder-neqE-linordered-idom)
          qed
           have ∀q ∈ set qs. q= 0 ⟶ squash (poly q s) = squash (poly q y) by
auto
          then have ∀q ∈ set qs. squash (poly q s) = squash (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 $\exists\ k \in (set\ S).\ consistent\text{-}sign\text{-}vec\ qs\ k = consistent\text{-}sign\text{-}vec\ qs\ y$
using *c-c1 c-c2* **by** *auto*
qed
show *?thesis*
using *c1 c2* **by** *auto*
qed
then show $x \in consistent\text{-}sign\text{-}vectors\text{-}R\ qs\ (set\ S)$
using *y-prop* **unfolding** *consistent-sign-vectors-R-def*
by (*metis imageI*)
qed
have *easy-direction*: $consistent\text{-}sign\text{-}vectors\text{-}R\ qs\ (set\ S) \subseteq consistent\text{-}sign\text{-}vectors\text{-}R\ qs\ UNIV$
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 $\bigwedge i.\ i \in$ *set-fml fml* $\Longrightarrow i <$ *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 $\bigwedge i.\ i \in$ *set-fml fml* $\Longrightarrow i <$ *length qs*
assumes *set signs* = *consistent-sign-vectors-R qs UNIV*
shows ($\forall$ *x*::*real*. *lookup-sem fml* (*map* ($\lambda p.$ *poly p x*) *qs*)) $\longleftrightarrow$
list-all (*lookup-sem fml*) *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 $\bigwedge i.\ i \in$ *set-fml fml* $\Longrightarrow i <$ *length qs*
assumes *set signs* = *consistent-sign-vectors-R qs UNIV*
shows ($\exists$ *x*::*real*. *lookup-sem fml* (*map* ($\lambda p.$ *poly p x*) *qs*)) $\longleftrightarrow$
find (*lookup-sem fml*) *signs* $\neq$ *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 (*fml-struct*,*polys*) = *convert fml*
shows ($\forall$ *x*::*real*. *lookup-sem fml-struct* (*map* ($\lambda p.$ *poly p x*) *polys*)) $\longleftrightarrow$ (*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 (*fml-struct*,*polys*) = *convert fml*
shows ($\exists$ *x*::*real*. *lookup-sem fml-struct* (*map* ($\lambda p.$ *poly p x*) *polys*)) $\longleftrightarrow$ (*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 $\bigwedge p.\ p \in$ *set* (*poly-list fml*) $\Longrightarrow$
ls ! (*index-of ps p*) = *poly p x*
shows *real-sem fml x* = *lookup-sem* (*map-fml* (*index-of ps*) *fml*) *ls*
using *assms* **apply** (*induct fml*)

by *auto*

lemma *convert-semantics-R*:
shows *real-sem fml x = lookup-sem* (*fst* (*convert fml*)) (*map* (λ*p*. *poly p x*) (*snd* (*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 (∀ *x*::*real*. *real-sem fml x*) ⟷ (*decide-universal-R fml*)
∃ *x*::*real*. *real-sem fml x* ⟷ (*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.