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*⟩

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*
⟨*proof*⟩

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*⟩

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*
⟨*proof*⟩

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
⟨*proof*⟩

lemma *index-mat-mod*[*simp*]: $nr > 0$ & $nc > 0 \Longrightarrow$ *mat nr nc f* \$\$ (*i mod nr*,*j mod nc*) = *f* (*i mod nr*,*j mod nc*)
⟨*proof*⟩

lemma *kronecker-assoc*:
shows *kronecker-product A* (*kronecker-product B C*) = *kronecker-product* (*kronecker-product A B*) *C*
⟨*proof*⟩

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

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*)
⟨*proof*⟩

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*
⟨*proof*⟩

lemma *less-mult-imp-mod-less*:
m mod i < *i* **if** *m* < *n* ∗ *i* **for** *m n i* :: *nat*
⟨*proof*⟩

lemma *kronecker-one*:
shows *kronecker-product* ((1_m *x*)::*'a* :: *ring-1 mat*) (1_m *y*) = 1_m (*x*∗*y*)
⟨*proof*⟩

lemma *kronecker-invertible*:
assumes *invertible-mat* (*A* :: *'a* :: *comm-ring-1 mat*) *invertible-mat B*
shows *invertible-mat* (*kronecker-product A B*)
⟨*proof*⟩

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*
⟨*proof*⟩

lemma *dim-col-conjugate*[*simp*]: *dim-col* (*conjugate m*) = *dim-col m*
⟨*proof*⟩

lemma *carrier-vec-conjugate*[*simp*]: *m* ∈ *carrier-mat nr nc* ⟹ *conjugate m* ∈ *carrier-mat nr nc*
⟨*proof*⟩

lemma *mat-index-conjugate*[*simp*]:
shows *i* < *dim-row m* ⟹ *j* < *dim-col m* ⟹ *conjugate m* $$ (*i*,*j*) = *conjugate* (*m* $$ (*i*,*j*))
⟨*proof*⟩

lemma *row-conjugate*[*simp*]: *i* < *dim-row m* ⟹ *row* (*conjugate m*) *i* = *conjugate* (*row m i*)
⟨*proof*⟩

lemma *col-conjugate*[*simp*]: *i* < *dim-col m* ⟹ *col* (*conjugate m*) *i* = *conjugate* (*col m i*)
⟨*proof*⟩

lemma *rows-conjugate*: *rows* (*conjugate m*) = *map conjugate* (*rows m*)
⟨*proof*⟩

lemma *cols-conjugate*: *cols* (*conjugate m*) = *map conjugate* (*cols m*)
⟨*proof*⟩

instance
⟨*proof*⟩

end

abbreviation *conjugate-transpose* :: *'a*::*conjugate mat* ⇒ *'a mat*
where *conjugate-transpose A* ≡ *conjugate* (A^T)

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

lemma *transpose-conjugate*:
shows (*conjugate A*)^T = A^H
⟨*proof*⟩

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*⟩

lemma *vec-module-col-helper2*:
fixes *A*:: ('*a* :: *field*) *mat*
shows $\bigwedge$*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*)) $\Longrightarrow$
($\bigwedge$*a b v*. (*a* + *b*) $\cdot_v$ *v* = *a* $\cdot_v$ *v* + *b* $\cdot_v$ *v*) $\Longrightarrow$
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*⟩

lemma *vec-module-col*: *module* (*class-ring* :: '*a* :: *field ring*)
(*module-vec TYPE*('*a*)
(*dim-row A*)
⦇*carrier* :=
LinearCombinations.module.span
class-ring (*module-vec TYPE*('*a*) (*dim-row A*)) (*set* (*cols A*))⦈)
⟨*proof*⟩

lemma *vec-vs-col*: *vectorspace* (*class-ring* :: '*a* :: *field ring*)
(*module-vec TYPE*('*a*) (*dim-row A*)
⦇*carrier* :=
LinearCombinations.module.span
class-ring
(*module-vec TYPE*('*a*)
(*dim-row A*))
(*set* (*cols A*))⦈)
⟨*proof*⟩

lemma *cols-mat-mul-map*:
shows *cols* (*A* ∗ *B*) = *map* (($*_v$) *A*) (*cols B*)
⟨*proof*⟩

lemma *cols-mat-mul*:
shows *set* (*cols* (*A* ∗ *B*)) = ($*_v$) *A* ‘ *set* (*cols B*)
⟨*proof*⟩

lemma *set-obtain-sublist*:
assumes *S* ⊆ *set ls*
obtains *ss* **where** *distinct ss S* = *set ss*
⟨*proof*⟩

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*)
  ⟨*proof*⟩

lemma *helper*:
  fixes $x\ y\ z$ ::$'a$ :: {*conjugatable-ring*, *comm-ring*}
  shows $x * (y * z) = y * x * z$
  ⟨*proof*⟩

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 $x \bullet c\ (A *_v y) = (A^H *_v x) \bullet c\ y$
  ⟨*proof*⟩

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*⟩

lemma *row-mat-of-cols*:
  assumes $i < nr$
  shows *row* (*mat-of-cols nr ls*) i = *vec* (*length ls*) ($\lambda j.$ (*ls* ! j) \$$i$)
  ⟨*proof*⟩

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* $*_v\ v$
  ⟨*proof*⟩

lemma *smult-vec-zero*:
  fixes v ::$'a$::*ring vec*
  shows $0 \cdot_v v = 0_v$ (*dim-vec v*)
  ⟨*proof*⟩

lemma *helper2*:

fixes A ::$'a$::*comm-ring mat*
fixes v ::$'a$ *vec*
assumes $v \in$ *carrier-vec* (*length ss*)
assumes $\bigwedge x.\ x \in$ *set ls* $\Longrightarrow$ *dim-vec* $x = nr$
shows
mat-of-cols nr ss $*_v$ v =
mat-of-cols nr (*ls* @ *ss*) $*_v$ (0_v (*length ls*) $@_v$ v)
⟨*proof*⟩

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*
⟨*proof*⟩

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

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

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*⟩

lemma *helper3*:
fixes A ::$'a$::*comm-ring mat*
assumes A: $A \in$ *carrier-mat nr nc*
assumes *ss*:*distinct ss set ss* ⊆ *set* (*cols A*)
assumes $v \in$ *carrier-vec* (*length ss*)
obtains *c* **where** *mat-of-cols nr ss* $*_v$ v = A $*_v$ *c dim-vec c* = *nc*
⟨*proof*⟩

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

assumes $A \in$ *carrier-mat nr nc*
assumes *distinct ss set ss* $\subseteq$ *set* (*cols* (A^H))
assumes $v \in$ *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*⟩

lemma *Units-invertible*:
fixes *A*:: *'a::semiring-1 mat*
assumes $A \in$ *Units* (*ring-mat TYPE('a) n b*)
shows *invertible-mat A*
⟨*proof*⟩

lemma *invertible-Units*:
fixes *A*:: *'a::semiring-1 mat*
assumes *invertible-mat A*
shows $A \in$ *Units* (*ring-mat TYPE('a)* (*dim-row A*) *b*)
⟨*proof*⟩

lemma *invertible-det*:
fixes *A*:: *'a::field mat*
assumes $A \in$ *carrier-mat n n*
shows *invertible-mat A* $\longleftrightarrow$ *det A* $\neq 0$
⟨*proof*⟩

context *vec-space* **begin**

lemma *find-indices-distinct*:
assumes *distinct ss*
assumes $i <$ *length ss*
shows *find-indices* (*ss* ! *i*) *ss* = [*i*]
⟨*proof*⟩

lemma *lin-indpt-lin-comb-list*:
assumes *distinct ss*
assumes *lin-indpt* (*set ss*)
assumes *set ss* $\subseteq$ *carrier-vec n*
assumes *lincomb-list f ss* $= 0_v$ *n*
assumes $i <$ *length ss*
shows $f\ i = 0$
⟨*proof*⟩

lemma *span-mat-mul-subset*:
assumes $A \in$ *carrier-mat n d*
assumes $B \in$ *carrier-mat d nc*
shows *span* (*set* (*cols* $(A * B)$)) $\subseteq$ *span* (*set* (*cols A*))
⟨*proof*⟩

lemma *rank-mat-mul-right*:
  assumes $A \in$ *carrier-mat n d*
  assumes $B \in$ *carrier-mat d nc*
  shows *rank* $(A * B) \leq$ *rank A*
⟨*proof*⟩

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*)
  ⟨*proof*⟩

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*])
  ⟨*proof*⟩

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*))
  ⟨*proof*⟩

lemma *two-set*:
  assumes *distinct ls*
  assumes *set ls* = *set* [*a*,*b*]
  assumes $a \neq b$
  shows *ls* = [*a*,*b*] ∨ *ls* = [*b*,*a*]
  ⟨*proof*⟩

lemma *filter-disj-inds*:
  assumes $i <$ *length ls* $j <$ *length ls* $i \neq j$
  shows *filter* ($\lambda ia.\ ia \neq j \longrightarrow ia = i$) [*0*..<*length ls*] = [*i*, *j*] ∨
  filter ($\lambda ia.\ ia \neq j \longrightarrow ia = i$) [*0*..<*length ls*] = [*j*,*i*]
⟨*proof*⟩

lemma *lincomb-list-indpt-distinct*:
  assumes $\bigwedge v.\ v \in set\ ls \Longrightarrow dim\text{-}vec\ v = n$
  assumes
    $\bigwedge c.\ lincomb\text{-}list\ c\ ls = 0_v\ n \Longrightarrow (\forall i.\ i < (length\ ls) \longrightarrow c\ i = 0)$
  shows *distinct ls*
  ⟨*proof*⟩

end

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

$f\text{-}ty$::$'a$::*conjugatable-ordered-field itself*
and n
begin

lemma *transpose-rank-mul-conjugate-transpose*:
fixes A :: $'a$ *mat*
assumes $A \in$ *carrier-mat n nc*
shows *vec-space.rank nc* $A^H \leq$ *rank* $(A * A^H)$
⟨*proof*⟩

lemma *conjugate-transpose-rank-le*:
assumes $A \in$ *carrier-mat n nc*
shows *vec-space.rank nc* $(A^H) \leq$ *rank* A
⟨*proof*⟩

lemma *conjugate-finsum*:
assumes f: f : $U \rightarrow$ *carrier-vec n*
shows *conjugate* (*finsum* V f U) = *finsum* V (*conjugate* $\circ$ f) U
⟨*proof*⟩

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

lemma *rank-conjugate*:
assumes $A \in$ *carrier-mat n d*
shows *rank* (*conjugate* A) = *rank* A
⟨*proof*⟩

end

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

lemma *transpose-rank*:
fixes A::$'a$::{*conjugatable-ordered-field*} *mat*
shows *vec-space.rank* (*dim-row* A) A = *vec-space.rank* (*dim-col* A) (A^T)
⟨*proof*⟩

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* $(A * B) \leq$ *vec-space.rank d* B
⟨*proof*⟩

3 Results on Invertibility

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

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

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

lemma *cong1*:
$x = y \Longrightarrow$ *mat-of-cols n x* = *mat-of-cols n y*
⟨*proof*⟩

lemma *nth-filter*:
assumes $j <$ *length* (*filter P ls*)
shows *P* ((*filter P ls*) ! *j*)
⟨*proof*⟩

lemma *take-cols-mat-mul*:
assumes $A \in$ *carrier-mat nr n*
assumes $B \in$ *carrier-mat n nc*
shows $A *$ *take-cols B inds* = *take-cols* ($A * B$) *inds*
⟨*proof*⟩

lemma *take-cols-carrier-mat*:
assumes $A \in$ *carrier-mat nr nc*
obtains *n* **where** *take-cols A inds* $\in$ *carrier-mat nr n*
⟨*proof*⟩

lemma *take-cols-carrier-mat-strict*:
assumes $A \in$ *carrier-mat nr nc*
assumes $\bigwedge i.\ i \in$ *set inds* $\Longrightarrow i < nc$
shows *take-cols A inds* $\in$ *carrier-mat nr* (*length inds*)
⟨*proof*⟩

lemma *gauss-jordan-take-cols*:
assumes *gauss-jordan A* (*take-cols A inds*) = (*C*,*D*)
shows *D* = *take-cols C inds*
⟨*proof*⟩

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*
⟨*proof*⟩

lemma *dim-col-take-rows*[*simp*]:
  shows *dim-col* (*take-rows* A *inds*) = *dim-col* A
  ⟨*proof*⟩

lemma *cols-take-cols-subset*:
  shows *set* (*cols* (*take-cols* A *inds*)) $\subseteq$ *set* (*cols* A)
  ⟨*proof*⟩

lemma *dim-row-take-cols*[*simp*]:
  shows *dim-row* (*take-cols* A *ls*) = *dim-row* A
  ⟨*proof*⟩

lemma *dim-row-append-rows*[*simp*]:
  shows *dim-row* (A $@_r$ B) = *dim-row* A + *dim-row* B
  ⟨*proof*⟩

lemma *rows-inj*:
  assumes *dim-col* A = *dim-col* B
  assumes *rows* A = *rows* B
  shows $A = B$
  ⟨*proof*⟩

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))
  ⟨*proof*⟩

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))
  ⟨*proof*⟩

lemma *append-rows-mat-mul*:
  assumes *dim-col* A = *dim-col* B
  shows (A $@_r$ B) $*$ C = $A * C$ $@_r$ $B * C$
  ⟨*proof*⟩

lemma *cardlt*:
  shows *card* $\{i.\ i < (n{::}nat)\} \leq n$
  ⟨*proof*⟩

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

lemma *length-pivot-positions-dim-row*:
assumes *row-echelon-form A*
shows *length* (*pivot-positions A*) ≤ *dim-row A*
⟨*proof*⟩

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*⟩

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*⟩

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*⟩

lemma *pivot-positions-form-helper-1*:
shows (*a*, *b*) ∈ *set* (*pivot-positions-main-gen z A nr nc i j*) ⟹ *i* ≤ *a*
⟨*proof*⟩

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

lemma *sorted-pivot-positions*:
shows *sorted-wrt* (<) (*map fst* (*pivot-positions A*))
⟨*proof*⟩

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*⟩

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*⟩

lemma *rref-right-mul*:
assumes *row-echelon-form A*
assumes *dim-col A* ≥ *dim-row A*
shows
take-cols A (*map snd* (*pivot-positions A*)) ∗ *take-rows A* [*0*..<*length* (*pivot-positions A*)] = *A*
⟨*proof*⟩

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

lemma *lin-indpt-id*:
shows *lin-indpt* (*set* (*cols* (*1*$_m$ *n*)::′*a vec list*))
⟨*proof*⟩

lemma *lin-indpt-take-cols-id*:
shows *lin-indpt* (*set* (*cols* (*take-cols* (*1*$_m$ *n*) *inds*)))
⟨*proof*⟩

lemma *cols-id-unit-vecs*:
shows *cols* (*1*$_m$ *d*) = *unit-vecs d*
⟨*proof*⟩

lemma *distinct-cols-id*:
shows *distinct* (*cols* (*1*$_m$ *d*)::′*a vec list*)
⟨*proof*⟩

lemma *distinct-map-nth*:
assumes *distinct ls*
assumes *distinct inds*
assumes ⋀*j*. *j* ∈ *set inds* ⟹ *j* < *length ls*
shows *distinct* (*map* ((!) *ls*) *inds*)
⟨*proof*⟩

lemma *distinct-take-cols-id*:
assumes *distinct inds*
shows *distinct* (*cols* (*take-cols* (*1*$_m$ *n*) *inds*) :: ′*a vec list*)
⟨*proof*⟩

lemma *rank-take-cols*:
assumes *distinct inds*
shows *rank* (*take-cols* (*1*$_m$ *n*) *inds*) = *length* (*filter* ((>) *n*) *inds*)
⟨*proof*⟩

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

fixes A::$'a$ *mat*
assumes *invertible-mat* A
assumes $A \in$ *carrier-mat* n n
assumes $B \in$ *carrier-mat* n *nc*
shows *rank* $(A * B) =$ *rank* B
⟨*proof*⟩

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

lemma *rank-take-cols-leq*:
assumes R:$R \in$ *carrier-mat* n *nc*
shows *rank* (*take-cols* R *ls*) $\leq$ *rank* R
⟨*proof*⟩

lemma *rank-take-cols-geq*:
assumes R:$R \in$ *carrier-mat* n *nc*
assumes t:*take-cols* R *ls* $\in$ *carrier-mat* n r
assumes B:$B \in$ *carrier-mat* r *nc*
assumes $R =$ (*take-cols* R *ls*) $*$ B
shows *rank* (*take-cols* R *ls*) $\geq$ *rank* R
⟨*proof*⟩

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

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*⟩

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

⟨*proof*⟩

lemma *lin-indpt-subset-cols*:
  fixes *A*:: *'a mat*
  fixes *B*:: *'a vec set*
  assumes $A \in$ *carrier-mat n n*
  assumes *inv*: *invertible-mat A*
  assumes $B \subseteq$ *set* (*cols A*)
  shows *lin-indpt B*
⟨*proof*⟩

lemma *rank-invertible-subset-cols*:
  fixes *A*:: *'a mat*
  fixes *B*:: *'a vec list*
  assumes *inv*: *invertible-mat A*
  assumes *A-square*: $A \in$ *carrier-mat n n*
  assumes *set-sub*: *set* (*B*) $\subseteq$ *set* (*cols A*)
  assumes *dist-B*: *distinct B*
  shows *rank* (*mat-of-cols n B*) = *length B*
⟨*proof*⟩

end

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

begin

4 Setup

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

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

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

5 Base Case

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

((*mat-of-rows-list 2* [[*1*,*1*], [*1*,−*1*]]), ([[],[*0*]], [[*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* ($\lambda i.$ *lhs-vec* \$ $i \neq 0$) [*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*))
  ⟨*proof*⟩

end
theory *Matrix-Equation-Construction*

imports *BKR-Algorithm*
begin

9 Results with Sturm's Theorem

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

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

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

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

lemma *restate-tarski2*:
fixes q::*real poly*
assumes $p \neq 0$
shows $changes\text{-}R\text{-}smods\ p\ ((pderiv\ p) * q) =$
$int(card\ \{x.\ poly\ p\ x = 0 \wedge poly\ q\ x > 0\}) -$
$int(card\ \{x.\ poly\ p\ x = 0 \wedge poly\ q\ x < 0\})$
⟨*proof*⟩

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*⟩

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

10 Setting up the construction: Definitions

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

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

definition *construct-s-vector*:: *real poly* ⇒ *real poly list list* ⇒ *rat vec*
where $construct\text{-}s\text{-}vector\ p\ Is = vec\text{-}of\text{-}list\ (map\ (\lambda\ I.(construct\text{-}NofI\ p\ I))\ Is)$

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*⟩

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*⟩

lemma *subsets-are-rows*: ∀ *i* < (*length subsets*). *row* (*alt-matrix-A signs subsets*) *i* = *vec* (*length signs*) (λ*j*. *z* (*subsets* ! *i*) (*signs* ! *j*))
  ⟨*proof*⟩

lemma *signs-are-cols*: ∀ *i* < (*length signs*). *col* (*alt-matrix-A signs subsets*) *i* = *vec* (*length subsets*) (λ*j*. *z* (*subsets* ! *j*) (*signs* ! *i*))
  ⟨*proof*⟩

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*)
  ⟨*proof*⟩

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*)
  ⟨*proof*⟩

lemma *mtx-row-length*:

list-all (λ*r*. *length r* = *length signs*) (*map* (*mtx-row signs*) *ls*)
⟨*proof*⟩

thm *construct-lhs-vector-def*
thm *poly-roots-finite*

lemma *construct-lhs-vector-clean*:
assumes $p \neq 0$
assumes $i <$ *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*⟩

lemma *construct-lhs-vector-cleaner*:
assumes $p \neq 0$
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*)
⟨*proof*⟩

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*) ⟨*proof*⟩

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*⟩

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*⟩

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

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

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

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

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

lemma *vec-of-list-dot-rewrite*:
assumes *length xs* = *length ys*
shows *vec-of-list xs* · *vec-of-list ys* =
sum-list (*map2* (∗) *xs ys*)
⟨*proof*⟩

lemma *lhs-dot-rewrite*:
fixes *p*:: *real poly*
fixes *qs*:: *real poly list*
fixes *I*:: *nat list*
fixes *signs*:: *rat list list*
assumes *nonzero*: $p \neq 0$
shows
(*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*⟩

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*)
⟨*proof*⟩

lemma *construct-lhs-vector-drop-consistent*:
fixes *p*:: *real poly*
fixes *qs*:: *real poly list*
fixes *I*:: *nat list*
fixes *signs*:: *rat list list*
assumes *nonzero*: $p \neq 0$
assumes *distinct-signs*: *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*⟩

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*) ⊆ *set*(*signs*)
assumes *welldefined*: *list-constr I* (*length qs*)
assumes *pairwise-rel-prime-1*: ∀ *q*. ((*List.member qs q*) ⟶ (*coprime p q*))
shows (*vec-of-list* (*mtx-row signs I*) · (*construct-lhs-vector p qs signs*)) =
rat-of-int (*card* {*x*. *poly p x* = *0* ∧ *poly* (*prod-list* (*retrieve-polys qs I*)) *x* > *0*})
−
rat-of-int (*card* {*x*. *poly p x* = *0* ∧ *poly* (*prod-list* (*retrieve-polys qs I*)) *x* < *0*})
⟨*proof*⟩

lemma *matrix-equation-main-step*:
fixes *p*:: *real poly*
fixes *qs*:: *real poly list*
fixes *I*:: *nat list*
fixes *signs*:: *rat list list*
assumes *nonzero*: $p \neq 0$
assumes *distinct-signs*: *distinct signs*
assumes *all-info*: *set* (*characterize-consistent-signs-at-roots-copr p qs*) ⊆ *set*(*signs*)
assumes *welldefined*: *list-constr I* (*length qs*)
assumes *pairwise-rel-prime-1*: ∀ *q*. ((*List.member qs q*) ⟶ (*coprime p q*))
shows (*vec-of-list* (*mtx-row signs I*) · (*construct-lhs-vector p qs signs*)) =
construct-NofI p (*retrieve-polys qs I*)
⟨*proof*⟩

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*)
⟨*proof*⟩

theorem *matrix-equation*:
fixes *p*:: *real poly*
fixes *qs*:: *real poly list*
fixes *subsets*:: *nat list list*
fixes *signs*:: *rat list list*
assumes *nonzero*: $p \neq 0$
assumes *distinct-signs*: *distinct signs*
assumes *all-info*: *set* (*characterize-consistent-signs-at-roots-copr p qs*) ⊆ *set*(*signs*)
assumes *pairwise-rel-prime*: ∀ *q*. ((*List.member qs q*) ⟶ (*coprime p q*))

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

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*)
⟨*proof*⟩

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* ⊆ *set signs*
assumes ⋀*l i*. *l* ∈ *set subsets* ⟹ *i* ∈ *set l* ⟹ *i* < *length qs*
shows *M-mat signs subsets* $*_v$ *w-vec p qs signs* = *v-vec p qs subsets*
⟨*proof*⟩

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

begin

definition *well-def-signs*:: *nat* => *rat list list* ⇒ *bool*
where *well-def-signs num-polys sign-conds* ≡ ∀ *signs* ∈ *set*(*sign-conds*). (*length signs* = *num-polys*)

definition *satisfies-properties*:: *real poly* ⇒ *real poly list* ⇒ *nat list list* ⇒ *rat list list* ⇒ *rat mat* ⇒ *bool*
where *satisfies-properties p qs subsets signs matrix* =
(*all-list-constr subsets* (*length qs*) ∧ *well-def-signs* (*length qs*) *signs* ∧ *distinct signs*
∧ *satisfy-equation p qs subsets signs* ∧ *invertible-mat matrix* ∧ *matrix* = *matrix-A signs subsets*
∧ *set* (*characterize-consistent-signs-at-roots-copr p qs*) ⊆ *set*(*signs*)
)

12 Base Case

lemma *mat-base-case*:
shows *matrix-A* [[*1*],[−*1*]] [[],[*0*]] = (*mat-of-rows-list 2* [[*1*, *1*], [*1*, −*1*]])
⟨*proof*⟩

lemma *base-case-sgas*:
fixes *q p*:: *real poly*
assumes *nonzero*: *p* ≠ *0*
assumes *rel-prime*: *coprime p q*
shows *set* (*characterize-consistent-signs-at-roots-copr p* [*q*]) ⊆ {[*1*], [− *1*]}
⟨*proof*⟩

lemma *base-case-sgas-alt*:
fixes *p*:: *real poly*
fixes *qs*:: *real poly list*
assumes *len1*: *length qs* = *1*
assumes *nonzero*: *p* ≠ *0*
assumes *rel-prime*: ∀ *q*. (*List.member qs q*) ⟶ *coprime p q*
shows *set* (*characterize-consistent-signs-at-roots-copr p qs*) ⊆ {[*1*], [− *1*]}
⟨*proof*⟩

lemma *base-case-satisfy-equation*:
fixes *q p*:: *real poly*
assumes *nonzero*: *p* ≠ *0*
assumes *rel-prime*: *coprime p q*
shows *satisfy-equation p* [*q*] [[],[*0*]] [[*1*],[−*1*]]
⟨*proof*⟩

lemma *base-case-satisfy-equation-alt*:
fixes *p*:: *real poly*
fixes *qs*:: *real poly list*
assumes *len1*: *length qs* = *1*
assumes *nonzero*: *p* ≠ *0*

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

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

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

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

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

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

13 Inductive Step

13.1 Lemmas on smashing subsets and signs

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

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

lemma *subsets-smash-property*:
fixes *subsets1 subsets2* :: *nat list list*
fixes *n*:: *nat*
shows ∀ (*elem* :: *nat list*). (*List.member* (*subsets-smash n subsets1 subsets2*) *elem*) ⟶
(∃ (*elem1*::*nat list*). ∃ (*elem2*::*nat list*).
(*elem1* ∈ *set*(*subsets1*) ∧ *elem2* ∈ *set*(*subsets2*) ∧ *elem* = (*elem1* @ ((*map* ((+) *n*) *elem2*)))))
⟨*proof*⟩

13.2 Well-defined subsets preserved when smashing

lemma *list-constr-append*:
list-constr a n ∧ *list-constr b n* ⟶ *list-constr* (*a*@*b*) *n*
⟨*proof*⟩

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*⟩

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*)
⟨*proof*⟩

13.4 Distinct signs preserved when smashing

lemma *distinct-map-append*:
assumes *distinct ls*
shows *distinct* (*map* ((@) *xs*) *ls*)
⟨*proof*⟩

lemma *length-eq-append*:
assumes *length y* = *length b*
shows (*x* @ *y* = *a* @ *b*) ⟷ *x*=*a* ∧ *y* = *b*
⟨*proof*⟩

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

13.5 Consistent sign assignments preserved when smashing

lemma *subset-smash-signs*:
  fixes *a1 b1 a2 b2*:: *rat list list*
  assumes *sub1*: *set a1* ⊆ *set a2*
  assumes *sub2*: *set b1* ⊆ *set b2*
  shows *set* (*signs-smash a1 b1*) ⊆ *set* (*signs-smash a2 b2*)
⟨*proof*⟩

lemma *subset-helper*:
  fixes *p*:: *real poly*
  fixes *qs1 qs2* :: *real poly list*
  fixes *signs1 signs2* :: *rat list list*
  shows ∀ *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*⟩

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*))
    ⊆ *set* (*signs-smash signs1 signs2*)
⟨*proof*⟩

13.6 Main Results

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

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

lemma *dim-row-matrix-A*[*simp*]:
  shows *dim-row* (*matrix-A signs subsets*) = *length subsets*
  ⟨*proof*⟩

lemma *dim-col-matrix-A*[*simp*]:
  shows *dim-col* (*matrix-A signs subsets*) = *length signs*
  ⟨*proof*⟩

lemma *length-signs-smash*:
  shows
    length (*signs-smash signs1 signs2*) = *length signs1* ∗ *length signs2*
  ⟨*proof*⟩

lemma *length-subsets-smash*:
  shows
    length (*subsets-smash n subs1 subs2*) = *length subs1* ∗ *length subs2*
  ⟨*proof*⟩

lemma *length-eq-concat*:
  assumes $\bigwedge x.\ x \in \mathit{set}\ \mathit{ls} \Longrightarrow \mathit{length}\ x = n$
  assumes $i < n * \mathit{length}\ \mathit{ls}$
  shows *concat ls* ! *i* = *ls* ! (*i div n*) ! (*i mod n*)
  ⟨*proof*⟩

lemma *z-append*:
  assumes $\bigwedge i.\ i \in \mathit{set}\ \mathit{xs} \Longrightarrow i < \mathit{length}\ \mathit{as}$
  shows *z* (*xs* @ (*map* ((+) (*length as*)) *ys*)) (*as* @ *bs*) = *z xs as* ∗ *z ys bs*
⟨*proof*⟩

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

  assumes $\bigwedge l\ i.\ l \in \mathit{set}\ \mathit{subs1} \Longrightarrow i \in \mathit{set}\ l \Longrightarrow i < \mathit{n1}$
  assumes $\bigwedge j.\ j \in \mathit{set}\ \mathit{signs1} \Longrightarrow \mathit{length}\ j = \mathit{n1}$
  shows
    (*matrix-A* (*signs-smash signs1 signs2*) (*subsets-smash n1 subs1 subs2*)) =
    kronecker-product (*matrix-A signs1 subs1*) (*matrix-A signs2 subs2*)
  ⟨*proof*⟩

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*: $p \neq 0$
assumes *nontriv1*: *length qs1* > 0
assumes *nontriv2*: *length qs2* > 0
assumes *pairwise-rel-prime1*: $\forall$ *q*. ((*List.member qs1 q*) $\longrightarrow$ (*coprime p q*))
assumes *pairwise-rel-prime2*: $\forall$ *q*. ((*List.member qs2 q*) $\longrightarrow$ (*coprime p q*))
assumes *welldefined-signs1*: *well-def-signs* (*length qs1*) *signs1*
assumes *welldefined-signs2*: *well-def-signs* (*length qs2*) *signs2*
assumes *distinct-signs1*: *distinct signs1*
assumes *distinct-signs2*: *distinct signs2*
assumes *all-info1*: *set* (*characterize-consistent-signs-at-roots-copr p qs1*) $\subseteq$ *set*(*signs1*)
assumes *all-info2*: *set* (*characterize-consistent-signs-at-roots-copr p qs2*) $\subseteq$ *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*)
$\wedge$ *invertible-mat* (*matrix-A* (*signs-smash signs1 signs2*) (*subsets-smash* (*length qs1*) *subsets1 subsets2*))
⟨*proof*⟩

14 Reduction Step Proofs

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

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

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

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

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

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

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

lemma *find-zeros-from-vec-prop*:
fixes *input-vec* :: *rat vec*
shows ∀ *n* < (*dim-vec input-vec*). ((*input-vec* $ *n* ≠ *0*) ⟷
List.member (*find-nonzeros-from-input-vec input-vec*) *n*)
⟨*proof*⟩

14.1 Showing sign conditions preserved when reducing

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

lemma *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*⟩

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*⟩

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*
⟨*proof*⟩

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*)
  ⟨*proof*⟩

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*
  ⟨*proof*⟩

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*
  ⟨*proof*⟩

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*
  ⟨*proof*⟩

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*⟩

lemma *mat-inverse-same*: *mat-inverse-var A* = *mat-inverse A*
  ⟨*proof*⟩

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

signs subsets)
⟨*proof*⟩

lemma *reduction-signs-set-helper-lemma*:
fixes *A*:: *rat list set*
fixes *C*:: *rat vec*
fixes *B*:: *rat list list*
assumes *dim-vec* C = *length* B
assumes *sub*: $A \subseteq$ *set*(B)
assumes *not-in-hyp*: $\forall$ $n <$ *dim-vec* C. C \$ $n = 0 \longrightarrow$ (*nth* B n) $\notin A$
shows $A \subseteq$ *set* (*take-indices* B
(*find-nonzeros-from-input-vec* C))
⟨*proof*⟩

lemma *reduction-signs-set-helper-lemma2*:
fixes *A*:: *rat list set*
fixes *C*:: *rat vec*
fixes *B*:: *rat list list*
assumes *dist*: *distinct B*
assumes *eq-len*: *dim-vec* C = *length* B
assumes *sub*: $A \subseteq$ *set*(B)
assumes *not-in-hyp*: $\forall$ $n <$ *dim-vec* C. C \$ $n \neq 0 \longrightarrow$ (*nth* B n) $\in A$
shows *set* (*take-indices* B
(*find-nonzeros-from-input-vec* C)) $\subseteq A$
⟨*proof*⟩

lemma *reduction-doesnt-break-things-signs*:
fixes *p*:: *real poly*
fixes *qs* :: *real poly list*
fixes *subsets* :: *nat list list*
fixes *signs* :: *rat list list*
assumes *nonzero*: $p \neq 0$
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*⟩

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*: *set* (*characterize-consistent-signs-at-roots-copr p qs*) $\subseteq$ *set*(*signs*)
assumes *match*: *satisfy-equation p qs subsets signs*
assumes *invertible-mat*: *invertible-mat* (*matrix-A signs subsets*)
shows *set* (*characterize-consistent-signs-at-roots-copr p qs*) = *set*(*reduction-signs p qs signs subsets* (*matrix-A signs subsets*))
⟨*proof*⟩

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

14.2 Showing matrix equation preserved when reducing

lemma *rows-to-keep-lem*:
fixes *A*:: (*'a*::*field*) *mat*
shows $\bigwedge$*ell*. *ell* $\in$ *set* (*rows-to-keep A*) $\Longrightarrow$ *ell* < *dim-row A*
⟨*proof*⟩

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

14.3 Showing matrix preserved

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

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

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

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

shows *matrix-A signs* (*take-indices subsets S*) = *take-rows-from-matrix* (*matrix-A signs subsets*) *S*
⟨*proof*⟩

lemma *reduce-system-matrix-match*:
fixes *p*:: *real poly*
fixes *qs* :: *real poly list*
fixes *subsets* :: *nat list list*
fixes *signs* :: *rat list list*
assumes *nonzero*: $p \neq 0$
assumes *welldefined-signs1*: *well-def-signs* (*length qs*) *signs*
assumes *distinct-signs*: *distinct signs*
assumes *all-info*: *set* (*characterize-consistent-signs-at-roots-copr p qs*) ⊆ *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*⟩

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*: $p \neq 0$
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*) ⊆ *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*⟩

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*: $p \neq 0$

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*) ⊆ *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*⟩

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

lemma *rechar-take-cols*: *take-cols-var A B* = *take-cols-from-matrix A B*
⟨*proof*⟩

lemma *rows-and-cols-transpose*: *rows M* = *cols* M^T
⟨*proof*⟩

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

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

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

14.5 Well def signs preserved when reducing

lemma *reduction-doesnt-break-length-signs*:
fixes *p*:: *real poly*
fixes *qs* :: *real poly list*
fixes *subsets* :: *nat list list*
fixes *signs* :: *rat list list*
assumes *length-init* : ∀ *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*⟩

14.6 Distinct signs preserved when reducing

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

14.7 Well def subsets preserved when reducing

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

15 Overall Lemmas

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

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*)))
⟨*proof*⟩

15.1 Key properties preserved

15.1.1 Properties preserved when combining and reducing systems

lemma *combining-sys-satisfies-properties-helper*:
fixes *p*:: *real poly*
fixes *qs1* :: *real poly list*
fixes *qs2* :: *real poly list*
fixes *subsets1 subsets2* :: *nat list list*
fixes *signs1 signs2* :: *rat list list*
fixes *matrix1 matrix2*:: *rat mat*
assumes *nonzero*: $p \neq 0$
assumes *nontriv1*: *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*⟩

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*: ∀ *q*. ((*List.member qs1 q*) ⟶ (*coprime p q*))
assumes *nontriv2*: *length qs2* > *0*
assumes *pairwise-rel-prime2*: ∀ *q*. ((*List.member qs2 q*) ⟶ (*coprime p q*))
assumes *satisfies-properties-sys1*: *satisfies-properties p qs1* (*get-subsets* (*calculate-data*

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

lemma *reducing-sys-satisfies-properties*:
fixes *p*:: *real poly*
fixes *qs* :: *real poly list*
fixes *subsets* :: *nat list list*
fixes *signs* :: *rat list list*
fixes *matrix*:: *rat mat*
assumes *nonzero*: $p \neq 0$
assumes *nontriv*: $length\ qs > 0$
assumes *pairwise-rel-prime*: ∀ *q*. ((*List.member qs q*) ⟶ (*coprime p q*))
assumes *satisfies-properties-sys*: *satisfies-properties p qs subsets signs matrix*
shows *satisfies-properties p qs* (*get-subsets* (*reduce-system p* (*qs*,*matrix*,*subsets*,*signs*)))
(*get-signs* (*reduce-system p* (*qs*,*matrix*,*subsets*,*signs*)))
(*get-matrix* (*reduce-system p* (*qs*,*matrix*,*subsets*,*signs*)))
⟨*proof*⟩

15.1.2 For length 1 qs

lemma *length-1-calculate-data-satisfies-properties*:
fixes *p*:: *real poly*
fixes *qs* :: *real poly list*
fixes *subsets* :: *nat list list*
fixes *signs* :: *rat list list*
assumes *nonzero*: $p \neq 0$
assumes *len1*: $length\ qs = 1$
assumes *pairwise-rel-prime*: ∀ *q*. ((*List.member qs q*) ⟶ (*coprime p q*))
shows *satisfies-properties p qs* (*get-subsets* (*calculate-data p qs*)) (*get-signs* (*calculate-data p qs*)) (*get-matrix* (*calculate-data p qs*))
⟨*proof*⟩

15.1.3 For arbitrary qs

lemma *append-not-distinct-helper*: (*List.member l1 m* ∧ *List.member l2 m*) ⟶ (*distinct* (*l1*@*l2*) = *False*)
⟨*proof*⟩

lemma *calculate-data-satisfies-properties*:
fixes *p*:: *real poly*
fixes *qs* :: *real poly list*
fixes *subsets* :: *nat list list*

fixes *signs* :: *rat list list*
shows $(p \neq 0 \wedge (length\ qs > 0) \wedge (\forall\, q.\ ((List.member\ qs\ q) \longrightarrow (coprime\ p\ q)))$
)
$\longrightarrow$ *satisfies-properties p qs* (*get-subsets* (*calculate-data p qs*)) (*get-signs* (*calculate-data p qs*)) (*get-matrix* (*calculate-data p qs*))
⟨*proof*⟩

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*: $p \neq 0$
assumes *len1*: $length\ qs = 1$
assumes *pairwise-rel-prime*: $\forall\, q.\ ((List.member\ qs\ q) \longrightarrow (coprime\ p\ q))$
shows *set* (*find-consistent-signs-at-roots p qs*) = *set* (*characterize-consistent-signs-at-roots-copr p qs*)
⟨*proof*⟩

lemma *smaller-sys-are-good*:
fixes *p*:: *real poly*
fixes *qs1* :: *real poly list*
fixes *qs2* :: *real poly list*
fixes *subsets* :: *nat list list*
fixes *signs* :: *rat list list*
assumes *nonzero*: $p \neq 0$
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 *set*(*find-consistent-signs-at-roots p qs1*) = *set*(*characterize-consistent-signs-at-roots-copr p qs1*)
assumes *set*(*find-consistent-signs-at-roots p qs2*) = *set*(*characterize-consistent-signs-at-roots-copr p qs2*)
shows *set*(*snd*(*snd*(*reduce-system p* (*combine-systems p* (*qs1*,*calculate-data p qs1*) (*qs2*,*calculate-data p qs2*)))))
= *set*(*characterize-consistent-signs-at-roots-copr p* (*qs1*@*qs2*))
⟨*proof*⟩

lemma *find-consistent-signs-at-roots-1*:
fixes *p*:: *real poly*
fixes *qs* :: *real poly list*
shows $(p \neq 0 \wedge length\ qs > 0 \wedge$
$(\forall\, q.\ ((List.member\ qs\ q) \longrightarrow (coprime\ p\ q)))) \longrightarrow$
set(*find-consistent-signs-at-roots p qs*) = *set*(*characterize-consistent-signs-at-roots-copr p qs*)

⟨*proof*⟩

lemma *find-consistent-signs-at-roots-0*:
fixes *p*:: *real poly*
assumes $p \neq 0$
shows *set*(*find-consistent-signs-at-roots p* []) =
set(*characterize-consistent-signs-at-roots-copr p* [])
⟨*proof*⟩

lemma *find-consistent-signs-at-roots-copr*:
fixes *p*:: *real poly*
fixes *qs* :: *real poly list*
assumes $p \neq 0$
assumes $\bigwedge q.\ q \in set\ qs \Longrightarrow coprime\ p\ q$
shows *set*(*find-consistent-signs-at-roots p qs*) = *set*(*characterize-consistent-signs-at-roots-copr p qs*)
⟨*proof*⟩

lemma *find-consistent-signs-at-roots*:
fixes *p*:: *real poly*
fixes *qs* :: *real poly list*
assumes $p \neq 0$
assumes $\bigwedge q.\ q \in set\ qs \Longrightarrow coprime\ p\ q$
shows *set*(*find-consistent-signs-at-roots p qs*) = *set*(*characterize-consistent-signs-at-roots p qs*)
⟨*proof*⟩

theorem *find-consistent-signs-at-roots-alt*:
assumes $p \neq 0$
assumes $\bigwedge q.\ q \in set\ qs \Longrightarrow coprime\ p\ q$
shows *set* (*find-consistent-signs-at-roots p qs*) = *consistent-signs-at-roots p qs*
⟨*proof*⟩

end
theory *BKR-Decision*
imports *BKR-Algorithm*
Berlekamp-Zassenhaus.Factorize-Rat-Poly
Algebraic-Numbers.Real-Roots
BKR-Proofs
HOL.Deriv
begin

16 Algorithm

16.1 Parsing

datatype *'a fml* =

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


primrec lookup-sem :: nat fml ⇒ ('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* (λ*l*. *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* (λ*i*.
map (λ*v*. *insertAt i 0 v*) (*find-consistent-signs-at-roots* (*in-list* ! *i*) (*removeAt i in-list*))
) [*0*..<*length in-list*])

definition *find-sgas* :: *real poly list* ⇒ *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* (λ*x*. *if poly x 0* < *0 then* −*1 else if poly x 0* = *0 then 0 else 1*) *ps*)] *else rsgas*)
)

16.5 Deciding Univariate Problems

definition *decide-universal* :: *rat poly fml* ⇒ *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*
⟨*proof*⟩

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*
⟨*proof*⟩

lemma *index-of-aux-more*:
shows *index-of-aux ls p n* ≥ *n*
⟨*proof*⟩

lemma *index-of-aux-lookup*:
assumes *p* ∈ *set ls*

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

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

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

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

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

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

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

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

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

17.2 Factoring Lemmas

interpretation *of-rat-poly-hom*: *map-poly-comm-semiring-hom of-rat*⟨*proof*⟩
interpretation *of-rat-poly-hom*: *map-poly-comm-ring-hom of-rat*⟨*proof*⟩
interpretation *of-rat-poly-hom*: *map-poly-idom-hom of-rat*⟨*proof*⟩

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*)
⟨*proof*⟩

lemma *sign-vec-index-of*:
assumes *f* ∈ *set ftrs*
shows *sign-vec ftrs x* ! (*index-of ftrs f*) = *squash* (*rpoly f x*)
⟨*proof*⟩

lemma *squash-idem*:
shows *squash* (*squash x*) = *squash x*
⟨*proof*⟩

lemma *squash-mult*:
shows *squash* ((*a*::*real*) ∗ *b*) = *squash a* ∗ *squash b*
⟨*proof*⟩

lemma *squash-prod-list*:
shows *squash* (*prod-list* (*ls*::*real list*)) = *prod-list* (*map squash ls*)
⟨*proof*⟩

lemma *squash-pow*:
shows *squash* ((*x*::*real*) ^ (*y*::*nat*)) = (*squash x*) ^ *y*
⟨*proof*⟩

lemma *squash-real-of-rat*[*simp*]:
shows *squash* (*real-of-rat x*) = *squash x*
⟨*proof*⟩

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*⟩

lemma *square-free-normalize*:
assumes *square-free p*
shows *square-free* (*normalize p*)
⟨*proof*⟩

lemma *coprime-normalize*:
assumes *coprime a b*
shows *coprime* (*normalize a*) *b*

⟨*proof*⟩

lemma *undo-normalize*:
shows $a = Polynomial.smult\ (unit\text{-}factor\ (lead\text{-}coeff\ a))\ (normalize\ a)$
⟨*proof*⟩

lemma *finite-smult-distr*:
assumes *distinct fs*
shows $(\prod (x,y) \in set\ fs.\ Polynomial.smult\ ((f\ x\ y)::rat)\ (g\ x\ y)) =$
$Polynomial.smult\ (\prod (x,y) \in set\ fs.\ f\ x\ y)\ (\prod (x,y) \in set\ fs.\ g\ x\ y)$
⟨*proof*⟩

lemma *normalize-coprime-degree*:
assumes $normalize\ (f::rat\ poly) = normalize\ g$
assumes *coprime f g*
shows $degree\ f = 0$
⟨*proof*⟩

lemma *factorize-rat-poly-monic-square-free-factorization*:
assumes *res*: $factorize\text{-}rat\text{-}poly\text{-}monic\ f = (c,fs)$
shows *square-free-factorization f (c,fs)*
⟨*proof*⟩

lemma *undo-factorize-correct*:
assumes $factorize\text{-}rat\text{-}poly\text{-}monic\ p = (c,fs)$
assumes $\bigwedge f\ p.\ (f,p) \in set\ fs \Longrightarrow f \in set\ ftrs$
shows $undo\text{-}factorize\ (c,map\ (\lambda(f,pow).\ (index\text{-}of\ ftrs\ f,\ pow))\ fs)\ (sign\text{-}vec\ ftrs\ x) = squash\ (rpoly\ p\ x)$
⟨*proof*⟩

lemma *length-sign-vec*[*simp*]:
shows $length\ (sign\text{-}vec\ ps\ x) = length\ ps$ ⟨*proof*⟩

lemma *factorize-polys-has-factors*:
assumes $factorize\text{-}polys\ ps = (ftrs,data)$
assumes $p \in set\ ps$
assumes $factorize\text{-}rat\text{-}poly\text{-}monic\ p = (c,fs)$
shows $set\ (map\ fst\ fs) \subseteq set\ ftrs$
⟨*proof*⟩

lemma *factorize-polys-undo-factorize-polys*:
assumes $factorize\text{-}polys\ ps = (ftrs,data)$
shows $undo\text{-}factorize\text{-}polys\ data\ (sign\text{-}vec\ ftrs\ x) = sign\text{-}vec\ ps\ x$
⟨*proof*⟩

lemma *factorize-polys-irreducible-monic*:
assumes $factorize\text{-}polys\ ps = (fs,data)$
shows $distinct\ fs\ \bigwedge f.\ f \in set\ fs \Longrightarrow irreducible\ f \wedge monic\ f$
⟨*proof*⟩

lemma *factorize-polys-square-free*:
  assumes *factorize-polys ps* = (*fs*,*data*)
  shows $\bigwedge f.\ f \in$ *set fs* $\Longrightarrow$ *square-free f*
  ⟨*proof*⟩

lemma *irreducible-monic-coprime*:
  assumes *f*: *monic f irreducible* (*f*::*rat poly*)
  assumes *g*: *monic g irreducible* (*g*::*rat poly*)
  assumes $f \neq g$
  shows *coprime f g*
  ⟨*proof*⟩

lemma *factorize-polys-coprime*:
  assumes *factorize-polys ps* = (*fs*,*data*)
  shows $\bigwedge f\ g.\ f \in$ *set fs* $\Longrightarrow g \in$ *set fs* $\Longrightarrow f \neq g \Longrightarrow$ *coprime f g*
  ⟨*proof*⟩

lemma *coprime-rat-poly-real-poly*:
  assumes *coprime p* (*q*::*rat poly*)
  shows *coprime* (*real-of-rat-poly p*) ((*real-of-rat-poly q*)::*real poly*)
  ⟨*proof*⟩

lemma *coprime-rat-poly-iff-coprimereal-poly*:
  shows *coprime p* (*q*::*rat poly*) $\longleftrightarrow$ *coprime* (*real-of-rat-poly p*) ((*real-of-rat-poly q*)::*real poly*)
⟨*proof*⟩

lemma *factorize-polys-map-distinct*:
  assumes *factorize-polys ps* = (*fs*,*data*)
  assumes *fss* = *map real-of-rat-poly fs*
  shows *distinct fss*
  ⟨*proof*⟩

lemma *factorize-polys-map-square-free*:
  assumes *factorize-polys ps* = (*fs*,*data*)
  assumes *fss* = *map real-of-rat-poly fs*
  shows $\bigwedge f.\ f \in$ *set fss* $\Longrightarrow$ *square-free f*
  ⟨*proof*⟩

lemma *factorize-polys-map-coprime*:
  assumes *factorize-polys ps* = (*fs*,*data*)
  assumes *fss* = *map real-of-rat-poly fs*
  shows $\bigwedge f\ g.\ f \in$ *set fss* $\Longrightarrow g \in$ *set fss* $\Longrightarrow f \neq g \Longrightarrow$ *coprime f g*
  ⟨*proof*⟩

lemma *coprime-prod-list*:
  assumes $\bigwedge p.\ p \in$ *set ps* $\Longrightarrow p \neq 0$
  assumes *coprime* (*prod-list ps*) (*q*::*real poly*)

shows $\bigwedge p.\ p \in set\ ps \Longrightarrow coprime\ p\ q$
$\langle proof \rangle$

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

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*)
$\langle proof \rangle$

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

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

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

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

17.3 Auxiliary Polynomial Lemmas

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

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

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*
  ⟨*proof*⟩

lemma *prod-zero*:
  shows ∀ *x* . *poly* (*prod-list* (*qs*:: *rat poly list*)) *x* = *0* ⟷ (∃ *q* ∈ *set* (*qs*). *poly q x* = *0*)
  ⟨*proof*⟩

lemma *coprime-r-zero1*: *poly* (*coprime-r qs*) (*crb* (*prod-list qs*)) = *0*
  ⟨*proof*⟩

lemma *coprime-r-zero2*: *poly* (*coprime-r qs*) (−*crb* (*prod-list qs*)) = *0*
  ⟨*proof*⟩

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*)
  ⟨*proof*⟩

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*⟩

lemma *coprime-r-nonzero*:
  fixes *ps*:: *rat poly list*
  assumes *factorize-polys ps* = (*fs*,*data*)
  assumes *nonempty-fs*: *fs* ≠ []
  assumes *fss-is*: *fss* = *map real-of-rat-poly fs*
  shows (*coprime-r fss*) ≠ *0*
⟨*proof*⟩

lemma *Rolle-pderiv*:
  fixes *q*:: *real poly*
  fixes *x1 x2*:: *real*

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

lemma *coprime-r-roots-prop*:
fixes *qs*:: *real poly list*
assumes *pairwise-rel-prime*: $\forall q1\ q2.\ (q1 \neq q2 \wedge (List.member\ qs\ q1) \wedge (List.member\ qs\ q2)) \longrightarrow coprime\ q1\ q2$
shows $\forall x1.\ \forall x2.\ ((x1 < x2 \wedge (\exists q1 \in set\ (qs).\ (poly\ q1\ x1) = 0) \wedge (\exists q2 \in set(qs).\ (poly\ q2\ x2) = 0)) \longrightarrow (\exists q.\ x1 < q \wedge q < x2 \wedge poly\ (coprime\text{-}r\ qs)\ q = 0))$
⟨*proof*⟩

17.4 Setting Up the Procedure: Lemmas

definition *has-no-zeros*::*rat list* ⇒ *bool*
where $has\text{-}no\text{-}zeros\ l = (0 \notin set\ l)$

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

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*
⟨*proof*⟩

lemma *coprime-over-reals-coprime-over-rats*:
fixes *qs*:: *rat poly list*
assumes *csa-in*: *csa* ∈ (*consistent-sign-vectors qs UNIV*)
assumes *p1p2*: $p1 \neq p2 \wedge p1 < length\ csa \wedge p2 < length\ csa \wedge csa\ !\ p1 = 0 \wedge csa\ !\ p2 = 0$
shows ¬ *algebraic-semidom-class.coprime* (*nth qs p1*) (*nth qs p2*)
⟨*proof*⟩

lemma *zero-above*:
fixes *qs*:: *rat poly list*
fixes *x1*:: *real*
assumes *hnz*: *has-no-zeros* (*sign-vec qs x1*)
shows $(\forall\ x2 > x1.\ ((sign\text{-}vec\ qs\ x1) \neq (sign\text{-}vec\ qs\ x2)) \longrightarrow (\exists (r::real) > x1.\ (r \leq x2 \wedge (\exists\ q \in set(qs).\ rpoly\ q\ r = 0))))$
⟨*proof*⟩

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-vec qs x1*) $\neq$ (*sign-vec qs x2*)) $\longrightarrow$
($\exists$ (*r*::*real*) < *x1*. (*r* $\geq$ *x2* $\wedge$ ($\exists$ *q* $\in$ *set*(*qs*). *rpoly q r* = *0*)))
⟨*proof*⟩

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 $\neg$($\exists$ (*x*::*real*). (*List.member l x* $\wedge$ *a* $\leq$ *x* $\wedge$ *x* $\leq$ *b*))
⟨*proof*⟩

lemma *roots-of-coprime-r-location-property*:
fixes *qs*:: *rat poly list*
fixes *sga*:: *rat list*
fixes *zer-list*
assumes *pairwise-rel-prime*: *pairwise-coprime-list qs*
assumes *all-squarefree*: $\bigwedge$*q*. *q* $\in$ *set qs* $\Longrightarrow$ *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*). ($\exists$ *q* $\in$ *set*(*qs*). (*rpoly q x* = *0*))}
shows *zer-list* $\neq$ [] $\longrightarrow$ ((*x1* < (*zer-list* ! *0*)) $\vee$ (*x1* > (*zer-list* ! (*length zer-list* $-$ *1*)) $\vee$
($\exists$ *n* < (*length zer-list* $-$ *1*). *x1* > (*zer-list* ! *n*) $\wedge$ *x1* < (*zer-list* ! (*n*+*1*)))))
⟨*proof*⟩

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

lemma *find-csas-lemma-nozeros*:

fixes *qs*:: *rat poly list*
assumes *fs*: *factorize-polys qs* = (*fs*,*data*)
assumes *fs* ≠ []
shows (*csa* ∈ (*consistent-sign-vectors fs UNIV*) ∧ *has-no-zeros csa*) ⟷
csa ∈ *set* (*find-consistent-signs-at-roots* (*coprime-r* (*cast-rat-list fs*)) (*cast-rat-list fs*))
⟨*proof*⟩

lemma *range-eq*:
fixes *a*
fixes *b*
shows *a* = *b* ⟶ *range a* = *range b*
⟨*proof*⟩

lemma *removeAt-distinct*:
shows *distinct fss* ⟹ *distinct* (*removeAt i fss*)
⟨*proof*⟩

lemma *consistent-signs-atw*:
assumes ⋀*p*. *p* ∈ *set fs* ⟹ *poly p x* ≠ *0*
shows *consistent-sign-vec-copr fs x* = *signs-at fs x*
⟨*proof*⟩

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*})
⟨*proof*⟩

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*
⟨*proof*⟩

lemma *length-removeAt*:
assumes *i* < *length ls*
shows *length* (*removeAt i ls*) = *length ls* − *1*
⟨*proof*⟩

lemma *insertAt-removeAt*:
assumes *i* < *length ls*
shows *insertAt i* (*ls*!*i*) (*removeAt i ls*) = *ls*
⟨*proof*⟩

lemma *insertAt-nth*:
assumes *i* ≤ *length ls*

shows *insertAt i n ls* ! *i* = *n*
⟨*proof*⟩

lemma *length-signs-at*[*simp*]:
shows *length* (*signs-at qs x*) = *length qs*
⟨*proof*⟩

lemma *find-csa-at-index*:
assumes *i* < *length fss*
assumes $\bigwedge$*p q*. *p* ∈ *set fss* ⟹ *q* ∈ *set fss* ⟹ *p* ≠ *q* ⟹ *coprime p q*
assumes $\bigwedge$*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*⟩

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

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*⟩

lemma *mem-append*: *List.member* (*l1*@*l2*) *m* ⟷ (*List.member l1 m* ∨ *List.member l2 m*)
⟨*proof*⟩

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*⟩

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*⟩

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*⟩

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*⟩

lemma *main-step*:
  fixes *qs*:: *rat poly list*
  shows *set* (*find-consistent-signs qs*) = *consistent-sign-vectors qs UNIV*
  ⟨*proof*⟩

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*)
  ⟨*proof*⟩

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*)
  ⟨*proof*⟩

theorem *decision-procedure*:
  shows (∀ *x*::*real*. *fml-sem fml x*) ⟷ (*decide-universal fml*)
    ∃ *x*::*real*. *fml-sem fml x* ⟷ (*decide-existential fml*)
  ⟨*proof*⟩

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

definition *construct-NofI-R*:: *real poly* ⇒ *real poly list* ⇒ *real poly list* ⇒ *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* ⇒ *real poly list* ⇒ (*nat list* ∗ *nat list*) *list* ⇒ *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* × ((*nat list* ∗ *nat list*) *list* × *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* ⇒ *real poly* ⇒ *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* ⇒ *real poly list* ⇒ (*nat list* ∗ *nat list*) *list* ⇒ *rat mat* ⇒ *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*))
  ⟨*proof*⟩

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*
  ⟨*proof*⟩

lemma *sum-list-map-power2*:
  shows *sum-list* (*map power2 ls*) ≥ (*0*::*real poly*)
  ⟨*proof*⟩

lemma *sum-list-map-power2-poly*:
  shows *poly* (*sum-list* (*map power2* (*ls*::*real poly list*))) *x* ≥ (*0*::*real*)
  ⟨*proof*⟩

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*⟩

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*⟩

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*}))
  ⟨*proof*⟩

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*⟩

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*))
⟨*proof*⟩

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*))
⟨*proof*⟩

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 $p \neq 0$
  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*⟩

lemma *construct-lhs-vector-cleaner-R*:
  assumes $p \neq 0$
  shows (*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*)
  ⟨*proof*⟩

lemma *z-signs-R2*:
  fixes *I*:: *nat list*
  fixes *signs*:: *rat list*
  assumes *lf*: *list-all* (λ*i*. *i* < *length signs*) *I*
  assumes *la*: *list-all* (λ*s*. *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* ⟨*proof*⟩

lemma *z-signs-R1*:
  fixes *I*:: *nat list*
  fixes *signs*:: *rat list*
  assumes *lf*: *list-all* (λ*i*. *i* < *length signs*) *I*
  assumes *la*: *list-all* (λ*s*. *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* ⟨*proof*⟩

lemma *z-signs-R*:
  fixes *I*:: (*nat list* ∗ *nat list*)
  fixes *signs*:: *rat list*
  assumes *lf*: *list-all* (λ*i*. *i* < *length signs*) (*fst*(*I*))
  assumes *ls*: *list-all* (λ*i*. *i* < *length signs*) (*snd*(*I*))
  assumes *la*: *list-all* (λ*s*. *s* = *1* ∨ *s* = *0* ∨ *s* = −*1*) *signs*
  shows (*z-R I signs* = *1*) ∨ (*z-R I signs* = *0*) ∨ (*z-R I signs* = −*1*)
  ⟨*proof*⟩

lemma *z-lemma-R*:
fixes *I*:: *nat list* $*$ *nat list*
fixes *sign*:: *rat list*
assumes *consistent*: *sign* $\in$ *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*) $\vee$ (*z-R I sign* = *0*) $\vee$ (*z-R I sign* = $-$*1*)
⟨*proof*⟩

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

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

lemma *horiz-vector-helper-pos-ind-R1*:
fixes *p*:: *real poly*
assumes *nonzero*: *p*$\neq$*0*
fixes *qs*:: *real poly list*
fixes *I*:: *nat list*
fixes *sign*:: *rat list*
fixes *x*:: *real*
assumes *root-p*: *x* $\in$ {*x*. *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* $\longleftrightarrow$
($\forall$ *p* $\in$ *set* (*retrieve-polys qs I*). *poly p x* = *0*)
⟨*proof*⟩

lemma *csv-length-same-as-qlist*:
fixes *p*:: *real poly*

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

lemma *horiz-vector-helper-zer-ind-R2*:
fixes *p*:: *real poly*
assumes *nonzero*: $p \neq 0$
fixes *qs*:: *real poly list*
fixes *I*:: *nat list*
fixes *sign*:: *rat list*
fixes *x*:: *real*
assumes *root-p*: $x \in \{x.\ poly\ p\ x = 0\}$
assumes *sign-fix*: *sign = consistent-sign-vec qs x*
assumes *asm*: *list-constr I (length qs)*
shows $(prod\text{-}list\ (map\ (nth\ sign)\ I)) = 0 \longleftrightarrow$
$(poly\ (prod\text{-}list\ (retrieve\text{-}polys\ qs\ I))\ x = 0)$
⟨*proof*⟩

lemma *horiz-vector-helper-pos-ind-R2*:
fixes *p*:: *real poly*
assumes *nonzero*: $p \neq 0$
fixes *qs*:: *real poly list*
fixes *I*:: *nat list*
fixes *sign*:: *rat list*
fixes *x*:: *real*
assumes *root-p*: $x \in \{x.\ poly\ p\ x = 0\}$
assumes *sign-fix*: *sign = consistent-sign-vec qs x*
assumes *asm*: *list-constr I (length qs)*
shows $(prod\text{-}list\ (map\ (nth\ sign)\ I)) = 1 \longleftrightarrow$
$(poly\ (prod\text{-}list\ (retrieve\text{-}polys\ qs\ I))\ x > 0)$
⟨*proof*⟩

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

$(retrieve\text{-}polys\ qs\ (snd\ I)))\ x > 0)) \longleftrightarrow (z\text{-}R\ I\ sign = 1)$
⟨*proof*⟩

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

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

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

lemma *construct-lhs-vector-drop-consistent-R*:
fixes *p*:: *real poly*
fixes *qs*:: *real poly list*

fixes *I*:: *nat list∗nat list*
fixes *signs*:: *rat list list*
assumes *nonzero*: *p≠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*⟩

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*⟩

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*⟩

lemma *mtx-row-length-R*:
list-all (λ*r*. *length r* = *length signs*) (*map* (*mtx-row-R signs*) *ls*)
⟨*proof*⟩

theorem *matrix-equation-R*:
  fixes *p*:: *real poly*
  fixes *qs*:: *real poly list*
  fixes *subsets*:: (*nat list*∗*nat list*) *list*
  fixes *signs*:: *rat list list*
  assumes *nonzero*: *p*≠*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*
  ⟨*proof*⟩

lemma *consistent-signs-at-roots-eq*:
  assumes *p* ≠ *0*
  shows *consistent-signs-at-roots p qs* =
    set (*characterize-consistent-signs-at-roots p qs*)
  ⟨*proof*⟩

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*
  ⟨*proof*⟩

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
  $\wedge$ *set* (*characterize-consistent-signs-at-roots p qs*) $\subseteq$ *set*(*signs*)
)

lemma *mat-base-case-R*:
  shows *matrix-A-R* [[*1*],[*0*],[*−1*]] [([], []),([*0*], []),([], [*0*])] = (*mat-of-rows-list 3* [[*1*,*1*,*1*], [*0*,*1*,*0*], [*1*,*0*,*−1*]])
  ⟨*proof*⟩

lemma *base-case-sgas-R*:
  fixes *q p*:: *real poly*
  assumes *nonzero*: $p \neq 0$
  shows *set* (*characterize-consistent-signs-at-roots p* [*q*]) $\subseteq$ {[*1*],[*0*], [− *1*]}
  ⟨*proof*⟩

lemma *base-case-sgas-alt-R*:
  fixes *p*:: *real poly*
  fixes *qs*:: *real poly list*
  assumes *len1*: *length qs* = *1*
  assumes *nonzero*: $p \neq 0$
  shows *set* (*characterize-consistent-signs-at-roots p qs*) $\subseteq$ {[*1*], [*0*], [− *1*]}
⟨*proof*⟩

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*⟩

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*⟩

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*])]))
  ⟨*proof*⟩

lemma *less-three*: (*n*::*nat*) < *Suc* (*Suc* (*Suc 0*)) $\longleftrightarrow$ *n* = *0* $\vee$ *n* = *1* $\vee$ *n* = *2*
  ⟨*proof*⟩

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*)
⟨*proof*⟩

lemma *inverse-mat-base-case-2-R*:
shows *inverts-mat* (*mat-of-rows-list 3* [[*1*,*1*,*1*], [*0*,*1*,*0*], [*1*,*0*,−*1*]]::*rat mat*) (*mat-of-rows-list 3* [[*1*/*2*, −*1*/*2*, *1*/*2*], [*0*, *1*, *0*], [*1*/*2*, −*1*/*2*, −*1*/*2*]]:: *rat mat*)
⟨*proof*⟩

lemma *base-case-invertible-mat-R*:
shows *invertible-mat* (*matrix-A-R* [[*1*],[*0*], [− *1*]] [([], []),([*0*], []),([], [*0*])])
⟨*proof*⟩

25 Inductive Step

25.1 Lemmas on smashing subsets

definition *subsets-first-component-list*::(*nat list*∗*nat list*) *list* ⇒ *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*⟩

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*⟩

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*⟩

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*⟩

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*⟩

25.4 Main Results

lemma *dim-row-matrix-A-R*[*simp*]:
  shows *dim-row* (*matrix-A-R signs subsets*) = *length subsets*
  ⟨*proof*⟩

lemma *dim-col-matrix-A-R*[*simp*]:
  shows *dim-col* (*matrix-A-R signs subsets*) = *length signs*
  ⟨*proof*⟩

lemma *length-subsets-smash-R*:
  shows
    length (*subsets-smash-R n subs1 subs2*) = *length subs1* ∗ *length subs2*
  ⟨*proof*⟩

lemma *z-append-R*:
fixes *xs*:: (*nat list* ∗ *nat list*)
assumes $\bigwedge i.\ i \in set\ (fst\ xs) \Longrightarrow i\ < length\ as$
assumes $\bigwedge i.\ i \in set\ (snd\ xs) \Longrightarrow i\ < length\ as$
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*⟩

lemma *matrix-construction-is-kronecker-product-R*:
fixes *qs1* :: *real poly list*
fixes *subs1 subs2* :: (*nat list*∗*nat list*) *list*
fixes *signs1 signs2* :: *rat list list*

assumes $\bigwedge l\ i.\ l \in set\ subs1 \Longrightarrow (i \in set\ (fst\ l) \vee i \in set\ (snd\ l)) \Longrightarrow i < n1$
assumes $\bigwedge j.\ j \in set\ signs1 \Longrightarrow length\ j = n1$
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*)
⟨*proof*⟩

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*⟩

26 Reduction Step Proofs

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

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

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

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

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

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

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

26.1 Showing sign conditions preserved when reducing

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

lemma *size-of-mat-R*:

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

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

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

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

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

lemma *reduction-doesnt-break-things-signs-R*:
fixes *p*:: *real poly*
fixes *qs* :: *real poly list*
fixes *subsets* :: (*nat list*∗*nat list*) *list*
fixes *signs* :: *rat list list*
assumes *nonzero*: *p* ≠ *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*⟩

lemma *reduction-deletes-bad-sign-conds-R*:
fixes *p*:: *real poly*
fixes *qs* :: *real poly list*
fixes *subsets* :: (*nat list*∗*nat list*) *list*
fixes *signs* :: *rat list list*
assumes *nonzero*: $p \neq 0$
assumes *welldefined-signs1*: *well-def-signs* (*length qs*) *signs*
assumes *distinct-signs*: *distinct signs*
assumes *all-info*: *set* (*characterize-consistent-signs-at-roots p qs*) ⊆ *set*(*signs*)
assumes *match*: *satisfy-equation-R p qs subsets signs*
assumes *invertible-mat*: *invertible-mat* (*matrix-A-R signs subsets*)
shows *set* (*characterize-consistent-signs-at-roots p qs*) = *set*(*reduction-signs-R p qs signs subsets* (*matrix-A-R signs subsets*))
⟨*proof*⟩

theorem *reduce-system-sign-conditions-R*:
fixes *p*:: *real poly*
fixes *qs* :: *real poly list*
fixes *subsets* :: (*nat list*∗*nat list*) *list*
fixes *signs* :: *rat list list*
assumes *nonzero*: $p \neq 0$
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* (*get-signs-R* (*reduce-system-R p* (*qs*, ((*matrix-A-R signs subsets*), (*subsets*, *signs*))))) = *set* (*characterize-consistent-signs-at-roots p qs*)
⟨*proof*⟩

26.2 Showing matrix equation preserved when reducing

lemma *reduce-system-matrix-equation-preserved-R*:
fixes *p*:: *real poly*
fixes *qs* :: *real poly list*
fixes *subsets* :: (*nat list*∗*nat list*) *list*
fixes *signs* :: *rat list list*
assumes *nonzero*: $p \neq 0$
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*⟩

26.3 Showing matrix preserved

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

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

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

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*
⟨*proof*⟩

lemma *reduce-system-matrix-match-R*:
fixes *p*:: *real poly*
fixes *qs* :: *real poly list*
fixes *subsets* :: (*nat list*∗*nat list*) *list*
fixes *signs* :: *rat list list*
assumes *nonzero*: $p \neq 0$
assumes *welldefined-signs1*: *well-def-signs* (*length qs*) *signs*
assumes *distinct-signs*: *distinct signs*
assumes *all-info*: *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*⟩

26.4 Showing invertibility preserved when reducing

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

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

lemma *well-def-find-zeros-from-lhs-vec-R*:
fixes *p*:: *real poly*
fixes *qs* :: *real poly list*
fixes *subsets* :: (*nat list*∗*nat list*) *list*
fixes *signs* :: *rat list list*
assumes *len-eq*: *length subsets* = *length signs*
assumes *inv*: *invertible-mat* (*matrix-A-R signs subsets*)
assumes *nonzero*: $p \neq 0$
assumes *welldefined-signs1*: *well-def-signs* (*length qs*) *signs*
assumes *distinct-signs*: *distinct signs*
assumes *all-info*: *set* (*characterize-consistent-signs-at-roots p qs*) ⊆ *set*(*signs*)
assumes *match*: *satisfy-equation-R p qs subsets signs*

shows $(\bigwedge j.\ j \in set$ (*find-nonzeros-from-input-vec*
(*solve-for-lhs-R p qs subsets* (*matrix-A-R signs subsets*))) $\Longrightarrow$
$j <$ *length* (*cols* (*matrix-A-R signs subsets*)))
⟨*proof*⟩

lemma *take-cols-subsets-og-cols-R*:
fixes *p*:: *real poly*
fixes *qs* :: *real poly list*
fixes *subsets* :: (*nat list*∗*nat list*) *list*
fixes *signs* :: *rat list list*
assumes *len-eq*: *length subsets* = *length signs*
assumes *inv*: *invertible-mat* (*matrix-A-R signs subsets*)
assumes *nonzero*: $p \neq 0$
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*⟩

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*: $p \neq 0$
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 *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*⟩

lemma *reduction-doesnt-break-things-invertibility-R*:
fixes *p*:: *real poly*
fixes *qs* :: *real poly list*
fixes *subsets* :: (*nat list*∗*nat list*) *list*
fixes *signs* :: *rat list list*

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

26.5 Well def signs preserved when reducing

lemma *reduction-doesnt-break-length-signs-R*:
fixes *p*:: *real poly*
fixes *qs* :: *real poly list*
fixes *subsets* :: (*nat list*∗*nat list*) *list*
fixes *signs* :: *rat list list*
assumes *length-init* : ∀ *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*
⟨*proof*⟩

26.6 Distinct signs preserved when reducing

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

26.7 Well def subsets preserved when reducing

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

shows *all-list-constr-R* (*reduction-subsets-R p qs signs subsets* (*matrix-A-R signs subsets*)) (*length qs*)
⟨*proof*⟩

27 Overall Lemmas

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

lemma *getter-functions-R*: *calculate-data-R p qs* = (*get-matrix-R* (*calculate-data-R p qs*), (*get-subsets-R* (*calculate-data-R p qs*), *get-signs-R* (*calculate-data-R p qs*)))
⟨*proof*⟩

27.1 Key properties preserved

27.1.1 Properties preserved when combining and reducing systems

lemma *combining-sys-satisfies-properties-helper-R*:
fixes *p*:: *real poly*
fixes *qs1* :: *real poly list*
fixes *qs2* :: *real poly list*
fixes *subsets1 subsets2* :: (*nat list* ∗ *nat list*) *list*
fixes *signs1 signs2* :: *rat list list*
fixes *matrix1 matrix2*:: *rat mat*
assumes *nonzero*: $p \neq 0$
assumes *nontriv1*: *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*⟩

lemma *combining-sys-satisfies-properties-R*:
fixes *p*:: *real poly*
fixes *qs1* :: *real poly list*
fixes *qs2* :: *real poly list*
assumes *nonzero*: $p \neq 0$
assumes *nontriv1*: *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*)))))
⟨*proof*⟩

lemma *reducing-sys-satisfies-properties-R*:
fixes *p*:: *real poly*
fixes *qs* :: *real poly list*
fixes *subsets* :: (*nat list*∗*nat list*) *list*
fixes *signs* :: *rat list list*
fixes *matrix*:: *rat mat*
assumes *nonzero*: $p \neq 0$
assumes *nontriv*: *length qs* > *0*
assumes *satisfies-properties-sys*: *satisfies-properties-R p qs subsets signs matrix*
shows *satisfies-properties-R p qs* (*get-subsets-R* (*reduce-system-R p* (*qs*,*matrix*,*subsets*,*signs*)))
(*get-signs-R* (*reduce-system-R p* (*qs*,*matrix*,*subsets*,*signs*)))
(*get-matrix-R* (*reduce-system-R p* (*qs*,*matrix*,*subsets*,*signs*)))
⟨*proof*⟩

27.1.2 For length 1 qs

lemma *length-1-calculate-data-satisfies-properties-R*:
fixes *p*:: *real poly*
fixes *qs* :: *real poly list*
fixes *subsets* :: (*nat list*∗*nat list*) *list*
fixes *signs* :: *rat list list*
assumes *nonzero*: $p \neq 0$
assumes *len1*: *length qs* = *1*
shows *satisfies-properties-R p qs* (*get-subsets-R* (*calculate-data-R p qs*)) (*get-signs-R* (*calculate-data-R p qs*)) (*get-matrix-R* (*calculate-data-R p qs*))
⟨*proof*⟩

27.1.3 For arbitrary qs

lemma *calculate-data-satisfies-properties-R*:
fixes *p*:: *real poly*
fixes *qs* :: *real poly list*
fixes *subsets* :: (*nat list*∗*nat list*) *list*
fixes *signs* :: *rat list list*
shows ($p \neq 0$ ∧ (*length qs* > *0*))

$\longrightarrow$ *satisfies-properties-R p qs* (*get-subsets-R* (*calculate-data-R p qs*)) (*get-signs-R* (*calculate-data-R p qs*)) (*get-matrix-R* (*calculate-data-R p qs*))
⟨*proof*⟩

27.2 Some key results on consistent sign assignments

lemma *find-consistent-signs-at-roots-len1-R*:
fixes *p*:: *real poly*
fixes *qs* :: *real poly list*
fixes *subsets* :: (*nat list*∗*nat list*) *list*
fixes *signs* :: *rat list list*
assumes *nonzero*: $p \neq 0$
assumes *len1*: *length qs* = *1*
shows *set* (*find-consistent-signs-at-roots-R p qs*) = *set* (*characterize-consistent-signs-at-roots p qs*)
⟨*proof*⟩

lemma *smaller-sys-are-good-R*:
fixes *p*:: *real poly*
fixes *qs1* :: *real poly list*
fixes *qs2* :: *real poly list*
fixes *subsets* :: (*nat list*∗*nat list*) *list*
fixes *signs* :: *rat list list*
assumes *nonzero*: $p \neq 0$
assumes *nontriv1*: *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*⟩

lemma *find-consistent-signs-at-roots-1-R*:
fixes *p*:: *real poly*
fixes *qs* :: *real poly list*
shows ($p \neq 0$ ∧ *length qs* > *0*) $\longrightarrow$
set(*find-consistent-signs-at-roots-R p qs*) = *set*(*characterize-consistent-signs-at-roots p qs*)
⟨*proof*⟩

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*⟩

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*)
  ⟨*proof*⟩

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*⟩

lemma *q-dvd-prod-list-var-prop*:
assumes *q* ∈ *set qs*
assumes *q* ≠ *0*
shows *q dvd prod-list-var qs* ⟨*proof*⟩

lemma *check-all-const-deg-prop*:
shows *check-all-const-deg l* = *True* ⟷ (∀ *p* ∈ *set*(*l*). *degree p* = *0*)
⟨*proof*⟩

lemma *poly-f-nonzero*:
fixes *qs* :: *real poly list*
shows (*poly-f qs*) ≠ *0*
⟨*proof*⟩

lemma *poly-f-roots-prop-1*:
fixes *qs*:: *real poly list*
assumes *non-const*: *check-all-const-deg qs* = *False*
shows ∀ *x1*. ∀ *x2*. ((*x1* < *x2* ∧ (∃ *q1* ∈ *set* (*qs*). *q1* ≠ *0* ∧ (*poly q1 x1*) = *0*) ∧ (∃ *q2*∈ *set*(*qs*). *q2* ≠ *0* ∧ (*poly q2 x2*) = *0*)) ⟶ (∃ *q*. *x1* < *q* ∧ *q* < *x2* ∧ *poly* (*poly-f qs*) *q* = *0*))
⟨*proof*⟩

lemma *main-step-aux1-R*:
fixes *qs*:: *real poly list*
assumes *non-const*: *check-all-const-deg qs* = *True*
shows *set* (*find-consistent-signs-R qs*) = *consistent-sign-vectors-R qs UNIV*
⟨*proof*⟩

lemma *sorted-list-lemma-var*:

fixes l:: *real list*
fixes x:: *real*
assumes *length l* $>$ *1*
assumes *strict-sort*: *sorted-wrt* $(<)$ l
assumes *x-not-in*: $\neg$ (*List.member l x*)
assumes *lt-a*: $x > (l \ ! \ 0)$
assumes *b-lt*: $x < (l \ ! \ (length \ l - 1))$
shows $(\exists\, n.\ n < length\ l - 1 \wedge x > l \ ! \ n \wedge x < l \ !(n+1))$ $\langle proof \rangle$

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*)
$\langle proof \rangle$

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*
$\langle proof \rangle$

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

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*)
$\langle proof \rangle$

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*
$\langle proof \rangle$

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*
$\langle proof \rangle$

lemma *decide-univ-lem-helper-R*:
fixes *fml*:: *real poly fml*
assumes (*fml-struct*,*polys*) = *convert fml*
shows (∀ *x*::*real*. *lookup-sem fml-struct* (*map* (λ*p*. *poly p x*) *polys*)) ⟷ (*decide-universal-R fml*)
⟨*proof*⟩

lemma *decide-exis-lem-helper-R*:
fixes *fml*:: *real poly fml*
assumes (*fml-struct*,*polys*) = *convert fml*
shows (∃ *x*::*real*. *lookup-sem fml-struct* (*map* (λ*p*. *poly p x*) *polys*)) ⟷ (*decide-existential-R fml*)
⟨*proof*⟩

lemma *convert-semantics-lem-R*:
assumes ⋀*p*. *p* ∈ *set* (*poly-list fml*) ⟹
ls ! (*index-of ps p*) = *poly p x*
shows *real-sem fml x* = *lookup-sem* (*map-fml* (*index-of ps*) *fml*) *ls*
⟨*proof*⟩

lemma *convert-semantics-R*:
shows *real-sem fml x* = *lookup-sem* (*fst* (*convert fml*)) (*map* (λ*p*. *poly p x*) (*snd* (*convert fml*)))
⟨*proof*⟩

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*)
⟨*proof*⟩

end

Acknowledgements

This material is based upon work supported by the National Science Foundation Graduate Research Fellowship under Grants Nos. DGE1252522 and DGE1745016. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the National Science Foundation. This research was also sponsored by the National Science Foundation under Grant No. CNS-1739629, the AFOSR under grant number FA9550-16-1-0288, and A*STAR, Singapore. The views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of any sponsoring institution, the U.S. government or any other entity.

References

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

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