Symmetric Polynomials

Manuel Eberl

June 16, 2019

Abstract

A symmetric polynomial is a polynomial in variables $X_1, \ldots, X_n$ that does not discriminate between its variables, i. e. it is invariant under any permutation of them. These polynomials are important in the study of the relationship between the coefficients of a univariate polynomial and its roots in its algebraic closure.

This article provides a definition of symmetric polynomials and the elementary symmetric polynomials $e_1, \ldots, e_n$ and proofs of their basic properties, including three notable ones:

- Vieta's formula, which gives an explicit expression for the k-th coefficient of a univariate monic polynomial in terms of its roots $x_1, \ldots, x_n$, namely $c_k = (-1)^{n-k} e_{n-k}(x_1, \ldots, x_n)$.
- Second, the Fundamental Theorem of Symmetric Polynomials, which states that any symmetric polynomial is itself a uniquely determined polynomial combination of the elementary symmetric polynomials.
- Third, as a corollary of the previous two, that given a polynomial over some ring R, any symmetric polynomial combination of its roots is also in R even when the roots are not.

Both the symmetry property itself and the witness for the Fundamental Theorem are executable.

Contents

1 Vieta's Formulas

theory *Vieta*
imports
  HOL−Library.FuncSet
  HOL−Computational-Algebra.Computational-Algebra
begin

1.1 Auxiliary material

lemma *card-vimage-inter*:
  assumes *inj*: *inj-on f A* **and** *subset*: $X \subseteq f$ ‘ A
  shows *card* (f −‘ $X \cap A$) = *card* X
⟨*proof*⟩

lemma *bij-betw-image-fixed-card-subset*:
  assumes *inj-on f A*
  shows *bij-betw* (λX. f ‘ X) $\{X.\ X \subseteq A \wedge$ *card* $X = k\}$ $\{X.\ X \subseteq f$ ‘ $A \wedge$ *card* $X = k\}$
  ⟨*proof*⟩

lemma *image-image-fixed-card-subset*:
  assumes *inj-on f A*
  shows (λX. f ‘ X) ‘ $\{X.\ X \subseteq A \wedge$ *card* $X = k\}$ = $\{X.\ X \subseteq f$ ‘ $A \wedge$ *card* $X = k\}$
  ⟨*proof*⟩

lemma *prod-uminus*: ($\prod x{\in}A.\ -f\ x$ :: $'a$ :: *comm-ring-1*) = (-1) ^ *card* A ∗ ($\prod x{\in}A.\ f\ x$)
  ⟨*proof*⟩

theorem *prod-sum-PiE*:
  fixes f :: $'a \Rightarrow 'b \Rightarrow 'c$ :: *comm-semiring-1*
  assumes *finite*: *finite A* **and** *finite*: $\bigwedge x.\ x \in A \Longrightarrow$ *finite* ($B\ x$)
  shows ($\prod x{\in}A.\ \sum y{\in}B\ x.\ f\ x\ y$) = ($\sum g{\in}PiE\ A\ B.\ \prod x{\in}A.\ f\ x\ (g\ x)$)
  ⟨*proof*⟩

corollary *prod-add*:
  fixes *f1 f2* :: $'a \Rightarrow 'c$ :: *comm-semiring-1*
  assumes *finite*: *finite A*
  shows ($\prod x{\in}A.\ f1\ x + f2\ x$) = ($\sum X{\in}Pow\ A.\ (\prod x{\in}X.\ f1\ x) * (\prod x{\in}A{-}X.\ f2\ x)$)
⟨*proof*⟩

corollary *prod-diff1*:
  fixes *f1 f2* :: $'a \Rightarrow 'c$ :: *comm-ring-1*
  assumes *finite*: *finite A*
  shows ($\prod x{\in}A.\ f1\ x - f2\ x$) = ($\sum X{\in}Pow\ A.\ (-1)$ ^ *card* $X * (\prod x{\in}X.\ f2\ x) * (\prod x{\in}A{-}X.\ f1\ x)$)
⟨*proof*⟩

corollary *prod-diff2*:
fixes *f1 f2* :: *'a ⇒ 'c* :: *comm-ring-1*
assumes *finite*: *finite A*
shows $(\prod x{\in}A.\ f1\ x - f2\ x) = (\sum X{\in}Pow\ A.\ (-1)\ \hat{}\ (card\ A - card\ X) * (\prod x{\in}X.\ f1\ x) * (\prod x{\in}A{-}X.\ f2\ x))$
⟨*proof*⟩

1.2 Main proofs

Our goal is to determine the coefficients of some fully factored polynomial $p(X) = c(X - x_1) \dots (X - x_n)$ in terms of the x_i. It is clear that it is sufficient to consider monic polynomials (i.e. $c = 1$), since the general case follows easily from this one.

We start off by expanding the product over the linear factors:

lemma *poly-from-roots*:
fixes *f* :: *'a ⇒ 'b* :: *comm-ring-1* **assumes** *fin*: *finite A*
shows $(\prod x{\in}A.\ [{:}-f\ x,\ 1{:}]) = (\sum X{\in}Pow\ A.\ monom\ ((-1)\ \hat{}\ card\ X * (\prod x{\in}X.\ f\ x))\ (card\ (A - X)))$
⟨*proof*⟩

Comparing coefficients yields Vieta's formula:

theorem *coeff-poly-from-roots*:
fixes *f* :: *'a ⇒ 'b* :: *comm-ring-1*
assumes *fin*: *finite A* **and** *k*: $k \le card\ A$
shows $coeff\ (\prod x{\in}A.\ [{:}-f\ x,\ 1{:}])\ k = (-1)\ \hat{}\ (card\ A - k) * (\sum X \mid X \subseteq A \wedge card\ X = card\ A - k.\ (\prod x{\in}X.\ f\ x))$
⟨*proof*⟩

If the roots are all distinct, we can get the following alternative representation:

corollary *coeff-poly-from-roots'*:
fixes *f* :: *'a ⇒ 'b* :: *comm-ring-1*
assumes *fin*: *finite A* **and** *inj*: *inj-on f A* **and** *k*: $k \le card\ A$
shows $coeff\ (\prod x{\in}A.\ [{:}-f\ x,\ 1{:}])\ k = (-1)\ \hat{}\ (card\ A - k) * (\sum X \mid X \subseteq f\ `\ A \wedge card\ X = card\ A - k.\ \prod X)$
⟨*proof*⟩

end

2 Symmetric Polynomials

theory *Symmetric-Polynomials*
imports
Vieta
Polynomials.More-MPoly-Type

HOL−Library.Permutations
begin

2.1 Auxiliary facts

An infinite set has infinitely many infinite subsets.

lemma *infinite-infinite-subsets*:
assumes *infinite A*
shows $infinite\ \{X.\ X \subseteq A \wedge infinite\ X\}$
⟨*proof*⟩

An infinite set contains infinitely many finite subsets of any fixed nonzero cardinality.

lemma *infinite-card-subsets*:
assumes *infinite A* $k > 0$
shows $infinite\ \{X.\ X \subseteq A \wedge finite\ X \wedge card\ X = k\}$
⟨*proof*⟩

lemma *comp-bij-eq-iff*:
assumes *bij f*
shows $g \circ f = h \circ f \longleftrightarrow g = h$
⟨*proof*⟩

lemma *sum-list-replicate* [*simp*]:
$sum\text{-}list\ (replicate\ n\ x) = of\text{-}nat\ n * (x :: 'a :: semiring\text{-}1)$
⟨*proof*⟩

lemma *ex-subset-of-card*:
assumes *finite A* $card\ A \geq k$
shows $\exists B.\ B \subseteq A \wedge card\ B = k$
⟨*proof*⟩

lemma *length-sorted-list-of-set* [*simp*]: $length\ (sorted\text{-}list\text{-}of\text{-}set\ A) = card\ A$
⟨*proof*⟩

lemma *upt-add-eq-append'*: $i \leq j \Longrightarrow j \leq k \Longrightarrow [i..<k] = [i..<j]\ @\ [j..<k]$
⟨*proof*⟩

2.2 Subrings and ring homomorphisms

locale *ring-closed* =
fixes $A :: 'a :: comm\text{-}ring\text{-}1\ set$
assumes *zero-closed* [*simp*]: $0 \in A$
assumes *one-closed* [*simp*]: $1 \in A$
assumes *add-closed* [*simp*]: $x \in A \Longrightarrow y \in A \Longrightarrow (x + y) \in A$
assumes *mult-closed* [*simp*]: $x \in A \Longrightarrow y \in A \Longrightarrow (x * y) \in A$
assumes *uminus-closed* [*simp*]: $x \in A \Longrightarrow -x \in A$
begin

lemma *minus-closed* [*simp*]: $x \in A \Longrightarrow y \in A \Longrightarrow x - y \in A$
⟨*proof*⟩

lemma *sum-closed* [*intro*]: $(\bigwedge x.\ x \in X \Longrightarrow f\ x \in A) \Longrightarrow sum\ f\ X \in A$
⟨*proof*⟩

lemma *power-closed* [*intro*]: $x \in A \Longrightarrow x\ \hat{}\ n \in A$
⟨*proof*⟩

lemma *Sum-any-closed* [*intro*]: $(\bigwedge x.\ f\ x \in A) \Longrightarrow Sum\text{-}any\ f \in A$
⟨*proof*⟩

lemma *prod-closed* [*intro*]: $(\bigwedge x.\ x \in X \Longrightarrow f\ x \in A) \Longrightarrow prod\ f\ X \in A$
⟨*proof*⟩

lemma *Prod-any-closed* [*intro*]: $(\bigwedge x.\ f\ x \in A) \Longrightarrow Prod\text{-}any\ f \in A$
⟨*proof*⟩

lemma *prod-fun-closed* [*intro*]: $(\bigwedge x.\ f\ x \in A) \Longrightarrow (\bigwedge x.\ g\ x \in A) \Longrightarrow prod\text{-}fun\ f\ g\ x \in A$
⟨*proof*⟩

lemma *of-nat-closed* [*simp*, *intro*]: $of\text{-}nat\ n \in A$
⟨*proof*⟩

lemma *of-int-closed* [*simp*, *intro*]: $of\text{-}int\ n \in A$
⟨*proof*⟩

end

locale *ring-homomorphism* =
  fixes $f :: 'a :: comm\text{-}ring\text{-}1 \Rightarrow 'b :: comm\text{-}ring\text{-}1$
  assumes *add*[*simp*]: $f\ (x + y) = f\ x + f\ y$
  assumes *uminus*[*simp*]: $f\ (-x) = -f\ x$
  assumes *mult*[*simp*]: $f\ (x * y) = f\ x * f\ y$
  assumes *zero*[*simp*]: $f\ 0 = 0$
  assumes *one* [*simp*]: $f\ 1 = 1$
begin

lemma *diff* [*simp*]: $f\ (x - y) = f\ x - f\ y$
⟨*proof*⟩

lemma *power* [*simp*]: $f\ (x\ \hat{}\ n) = f\ x\ \hat{}\ n$
⟨*proof*⟩

lemma *sum* [*simp*]: $f\ (sum\ g\ A) = (\sum x{\in}A.\ f\ (g\ x))$
⟨*proof*⟩

lemma *prod* [*simp*]: $f\ (prod\ g\ A) = (\prod x{\in}A.\ f\ (g\ x))$

⟨*proof*⟩

end

lemma *ring-homomorphism-id* [*intro*]: *ring-homomorphism id*
⟨*proof*⟩

lemma *ring-homomorphism-id′* [*intro*]: *ring-homomorphism* ($\lambda x.\ x$)
⟨*proof*⟩

lemma *ring-homomorphism-of-int* [*intro*]: *ring-homomorphism of-int*
⟨*proof*⟩

2.3 Various facts about multivariate polynomials

lemma *poly-mapping-nat-ge-0* [*simp*]: (m :: *nat* $\Rightarrow_0$ *nat*) $\geq$ *0*
⟨*proof*⟩

lemma *poly-mapping-nat-le-0* [*simp*]: (m :: *nat* $\Rightarrow_0$ *nat*) $\leq$ *0* $\longleftrightarrow$ m = *0*
⟨*proof*⟩

lemma *of-nat-diff-poly-mapping-nat*:
assumes $m \geq n$
shows *of-nat* ($m - n$) = (*of-nat* m − *of-nat* n :: ′*a* :: *monoid-add* $\Rightarrow_0$ *nat*)
⟨*proof*⟩

lemma *mpoly-coeff-transfer* [*transfer-rule*]:
rel-fun cr-mpoly (=) *poly-mapping.lookup MPoly-Type.coeff*
⟨*proof*⟩

lemma *mapping-of-sum*: ($\sum x{\in}A.$ *mapping-of* ($f\ x$)) = *mapping-of* (*sum* f A)
⟨*proof*⟩

lemma *mapping-of-eq-0-iff* [*simp*]: *mapping-of* p = *0* $\longleftrightarrow$ p = *0*
⟨*proof*⟩

lemma *Sum-any-mapping-of*: *Sum-any* ($\lambda x.$ *mapping-of* ($f\ x$)) = *mapping-of* (*Sum-any* f)
⟨*proof*⟩

lemma *Sum-any-parametric-cr-mpoly* [*transfer-rule*]:
(*rel-fun* (*rel-fun* (=) *cr-mpoly*) *cr-mpoly*) *Sum-any Sum-any*
⟨*proof*⟩

lemma *lookup-mult-of-nat* [*simp*]: *lookup* (*of-nat* $n * m$) k = n ∗ *lookup* m k
⟨*proof*⟩

lemma *mpoly-eqI*:
assumes $\bigwedge$*mon*. *MPoly-Type.coeff* p *mon* = *MPoly-Type.coeff* q *mon*

shows $p = q$
⟨*proof*⟩

lemma *coeff-mpoly-times*:
MPoly-Type.coeff $(p * q)$ *mon* = *prod-fun* (*MPoly-Type.coeff p*) (*MPoly-Type.coeff q*) *mon*
⟨*proof*⟩

lemma (**in** *ring-closed*) *coeff-mult-closed* [*intro*]:
$(\bigwedge x.\ coeff\ p\ x \in A) \Longrightarrow (\bigwedge x.\ coeff\ q\ x \in A) \Longrightarrow coeff\ (p * q)\ x \in A$
⟨*proof*⟩

lemma *coeff-notin-vars*:
assumes $\neg(keys\ m \subseteq vars\ p)$
shows $coeff\ p\ m = 0$
⟨*proof*⟩

lemma *finite-coeff-support* [*intro*]: $finite\ \{m.\ coeff\ p\ m \neq 0\}$
⟨*proof*⟩

lemma *insertion-altdef*:
$insertion\ f\ p = Sum\text{-}any\ (\lambda m.\ coeff\ p\ m * Prod\text{-}any\ (\lambda i.\ f\ i\ \hat{}\ lookup\ m\ i))$
⟨*proof*⟩

lemma *mpoly-coeff-uminus* [*simp*]: $coeff\ (-p)\ m = -coeff\ p\ m$
⟨*proof*⟩

lemma *Sum-any-uminus*: $Sum\text{-}any\ (\lambda x.\ -f\ x :: 'a :: ab\text{-}group\text{-}add) = -Sum\text{-}any\ f$
⟨*proof*⟩

lemma *insertion-uminus* [*simp*]: $insertion\ f\ (-p :: 'a :: comm\text{-}ring\text{-}1\ mpoly) = -insertion\ f\ p$
⟨*proof*⟩

lemma *Sum-any-lookup*: $finite\ \{x.\ g\ x \neq 0\} \Longrightarrow Sum\text{-}any\ (\lambda x.\ lookup\ (g\ x)\ y) = lookup\ (Sum\text{-}any\ g)\ y$
⟨*proof*⟩

lemma *Sum-any-diff*:
assumes $finite\ \{x.\ f\ x \neq 0\}$
assumes $finite\ \{x.\ g\ x \neq 0\}$
shows $Sum\text{-}any\ (\lambda x.\ f\ x - g\ x :: 'a :: ab\text{-}group\text{-}add) = Sum\text{-}any\ f - Sum\text{-}any\ g$
⟨*proof*⟩

lemma *insertion-diff*:
$insertion\ f\ (p - q :: 'a :: comm\text{-}ring\text{-}1\ mpoly) = insertion\ f\ p - insertion\ f\ q$
⟨*proof*⟩

lemma *insertion-power*: *insertion f* (*p* ^ *n*) = *insertion f p* ^ *n*
⟨*proof*⟩

lemma *insertion-sum*: *insertion f* (*sum g A*) = (∑*x*∈*A*. *insertion f* (*g x*))
⟨*proof*⟩

lemma *insertion-prod*: *insertion f* (*prod g A*) = (∏*x*∈*A*. *insertion f* (*g x*))
⟨*proof*⟩

lemma *coeff-Var*: *coeff* (*Var i*) *m* = (*1 when m* = *Poly-Mapping.single i 1*)
⟨*proof*⟩

lemma *vars-Var*: *vars* (*Var i* :: ′*a* :: {*one*,*zero*} *mpoly*) = (*if* (*0*::′*a*) = *1 then* {} *else* {*i*})
⟨*proof*⟩

lemma *insertion-Var* [*simp*]: *insertion f* (*Var i*) = *f i*
⟨*proof*⟩

lemma *insertion-Sum-any*:
assumes *finite* {*x*. *g x* ≠ *0*}
shows *insertion f* (*Sum-any g*) = *Sum-any* (λ*x*. *insertion f* (*g x*))
⟨*proof*⟩

lemma *keys-diff-subset*:
keys (*f* − *g*) ⊆ *keys f* ∪ *keys g*
⟨*proof*⟩

lemma *keys-empty-iff* [*simp*]: *keys p* = {} ⟷ *p* = *0*
⟨*proof*⟩

lemma *mpoly-coeff-0* [*simp*]: *MPoly-Type.coeff 0 m* = *0*
⟨*proof*⟩

lemma *lookup-1*: *lookup 1 m* = (*if m* = *0 then 1 else 0*)
⟨*proof*⟩

lemma *mpoly-coeff-1*: *MPoly-Type.coeff 1 m* = (*if m* = *0 then 1 else 0*)
⟨*proof*⟩

lemma *lookup-Const*$_0$: *lookup* (*Const*$_0$ *c*) *m* = (*if m* = *0 then c else 0*)
⟨*proof*⟩

lemma *mpoly-coeff-Const*: *MPoly-Type.coeff* (*Const c*) *m* = (*if m* = *0 then c else 0*)
⟨*proof*⟩

lemma *coeff-smult* [*simp*]: *coeff* (*smult c p*) *m* = (*c* :: ′*a* :: *mult-zero*) ∗ *coeff p m*

⟨*proof*⟩

lemma *in-keys-mapI*: $x \in keys\ m \Longrightarrow f\ (lookup\ m\ x) \neq 0 \Longrightarrow x \in keys\ (Poly\text{-}Mapping.map\ f\ m)$
⟨*proof*⟩

lemma *keys-uminus* [*simp*]: $keys\ (-m) = keys\ m$
⟨*proof*⟩

lemma *vars-uminus* [*simp*]: $vars\ (-p) = vars\ p$
⟨*proof*⟩

lemma *vars-smult*: $vars\ (smult\ c\ p) \subseteq vars\ p$
⟨*proof*⟩

lemma *vars-0* [*simp*]: $vars\ 0 = \{\}$
⟨*proof*⟩

lemma *vars-1* [*simp*]: $vars\ 1 = \{\}$
⟨*proof*⟩

lemma *vars-sum*: $vars\ (sum\ f\ A) \subseteq (\bigcup x\in A.\ vars\ (f\ x))$
⟨*proof*⟩

lemma *vars-prod*: $vars\ (prod\ f\ A) \subseteq (\bigcup x\in A.\ vars\ (f\ x))$
⟨*proof*⟩

lemma *vars-Sum-any*: $vars\ (Sum\text{-}any\ h) \subseteq (\bigcup i.\ vars\ (h\ i))$
⟨*proof*⟩

lemma *vars-Prod-any*: $vars\ (Prod\text{-}any\ h) \subseteq (\bigcup i.\ vars\ (h\ i))$
⟨*proof*⟩

lemma *vars-power*: $vars\ (p\ \hat{}\ n) \subseteq vars\ p$
⟨*proof*⟩

lemma *vars-diff*: $vars\ (p1 - p2) \subseteq vars\ p1 \cup vars\ p2$
⟨*proof*⟩

lemma *insertion-smult* [*simp*]: $insertion\ f\ (smult\ c\ p) = c * insertion\ f\ p$
⟨*proof*⟩

lemma *coeff-add* [*simp*]: $coeff\ (p + q)\ m = coeff\ p\ m + coeff\ q\ m$
⟨*proof*⟩

lemma *coeff-diff* [*simp*]: $coeff\ (p - q)\ m = coeff\ p\ m - coeff\ q\ m$
⟨*proof*⟩

lemma *insertion-monom* [*simp*]:

insertion f (*monom m c*) = *c* ∗ *Prod-any* (λ*x*. *f x* ˆ *lookup m x*)
⟨*proof*⟩

lemma *insertion-aux-Const*$_0$ [*simp*]: *insertion-aux f* (*Const*$_0$ *c*) = *c*
⟨*proof*⟩

lemma *insertion-Const* [*simp*]: *insertion f* (*Const c*) = *c*
⟨*proof*⟩

lemma *coeffs-0* [*simp*]: *coeffs 0* = {}
⟨*proof*⟩

lemma *coeffs-1* [*simp*]: *coeffs 1* = {*1*}
⟨*proof*⟩

lemma *coeffs-Const*: *coeffs* (*Const c*) = (*if c* = *0 then* {} *else* {*c*})
⟨*proof*⟩

lemma *coeffs-subset*: *coeffs* (*Const c*) ⊆ {*c*}
⟨*proof*⟩

lemma *keys-Const*$_0$: *keys* (*Const*$_0$ *c*) = (*if c* = *0 then* {} *else* {*0*})
⟨*proof*⟩

lemma *vars-Const* [*simp*]: *vars* (*Const c*) = {}
⟨*proof*⟩

lemma *prod-fun-compose-bij*:
assumes *bij f* **and** *f*: ⋀*x y*. *f* (*x* + *y*) = *f x* + *f y*
shows *prod-fun m1 m2* (*f x*) = *prod-fun* (*m1* ∘ *f*) (*m2* ∘ *f*) *x*
⟨*proof*⟩

lemma *add-nat-poly-mapping-zero-iff* [*simp*]:
(*a* + *b* :: ′*a* ⇒$_0$ *nat*) = *0* ⟷ *a* = *0* ∧ *b* = *0*
⟨*proof*⟩

lemma *prod-fun-nat-0*:
fixes *f g* :: (′*a* ⇒$_0$ *nat*) ⇒ ′*b*::*semiring-0*
shows *prod-fun f g 0* = *f 0* ∗ *g 0*
⟨*proof*⟩

lemma *mpoly-coeff-times-0*: *coeff* (*p* ∗ *q*) *0* = *coeff p 0* ∗ *coeff q 0*
⟨*proof*⟩

lemma *mpoly-coeff-prod-0*: *coeff* (∏*x*∈*A*. *f x*) *0* = (∏*x*∈*A*. *coeff* (*f x*) *0*)
⟨*proof*⟩

lemma *mpoly-coeff-power-0*: *coeff* (*p* ˆ *n*) *0* = *coeff p 0* ˆ *n*
⟨*proof*⟩

lemma *prod-fun-max*:
  fixes $f\ g$ :: $'a$::{*linorder*, *ordered-cancel-comm-monoid-add*} $\Rightarrow$ $'b$::*semiring-0*
  assumes *zero*: $\bigwedge m.\ m > a \Longrightarrow f\ m = 0$ $\bigwedge m.\ m > b \Longrightarrow g\ m = 0$
  assumes *fin*: *finite* $\{m.\ f\ m \neq 0\}$ *finite* $\{m.\ g\ m \neq 0\}$
  shows *prod-fun* $f\ g\ (a + b) = f\ a * g\ b$
⟨*proof*⟩

lemma *prod-fun-gt-max-eq-zero*:
  fixes $f\ g$ :: $'a$::{*linorder*, *ordered-cancel-comm-monoid-add*} $\Rightarrow$ $'b$::*semiring-0*
  assumes $m > a + b$
  assumes *zero*: $\bigwedge m.\ m > a \Longrightarrow f\ m = 0$ $\bigwedge m.\ m > b \Longrightarrow g\ m = 0$
  assumes *fin*: *finite* $\{m.\ f\ m \neq 0\}$ *finite* $\{m.\ g\ m \neq 0\}$
  shows *prod-fun* $f\ g\ m = 0$
⟨*proof*⟩

2.4 Restricting a monomial to a subset of variables

lift-definition *restrictpm* :: $'a$ *set* $\Rightarrow$ ($'a \Rightarrow_0 'b$ :: *zero*) $\Rightarrow$ ($'a \Rightarrow_0 'b$) **is**
  $\lambda A\ f\ x.$ *if* $x \in A$ *then* $f\ x$ *else* 0
  ⟨*proof*⟩

lemma *lookup-restrictpm*: *lookup* (*restrictpm* A m) x = (*if* $x \in A$ *then* *lookup* m x *else* 0)
  ⟨*proof*⟩

lemma *lookup-restrictpm-in* [*simp*]: $x \in A \Longrightarrow$ *lookup* (*restrictpm* A m) x = *lookup* m x
  and *lookup-restrict-pm-not-in* [*simp*]: $x \notin A \Longrightarrow$ *lookup* (*restrictpm* A m) $x = 0$
  ⟨*proof*⟩

lemma *keys-restrictpm* [*simp*]: *keys* (*restrictpm* A m) = *keys* $m \cap A$
  ⟨*proof*⟩

lemma *restrictpm-add*: *restrictpm* X ($m1 + m2$) = *restrictpm* X $m1$ + *restrictpm* X $m2$
  ⟨*proof*⟩

lemma *restrictpm-id* [*simp*]: *keys* $m \subseteq X \Longrightarrow$ *restrictpm* X $m = m$
  ⟨*proof*⟩

lemma *restrictpm-orthogonal* [*simp*]: *keys* $m \subseteq -X \Longrightarrow$ *restrictpm* X $m = 0$
  ⟨*proof*⟩

lemma *restrictpm-add-disjoint*:
  $X \cap Y = \{\} \Longrightarrow$ *restrictpm* X m + *restrictpm* Y m = *restrictpm* ($X \cup Y$) m
  ⟨*proof*⟩

lemma *restrictpm-add-complements*:

restrictpm X m + *restrictpm* (−*X*) *m* = *m restrictpm* (−*X*) *m* + *restrictpm X m* = *m*
⟨*proof*⟩

2.5 Mapping over a polynomial

lift-definition *map-mpoly* :: (′*a* :: *zero* ⇒ ′*b* :: *zero*) ⇒ ′*a mpoly* ⇒ ′*b mpoly* **is**
λ(*f* :: ′*a* ⇒ ′*b*) (*p* :: (*nat* $\Rightarrow_0$ *nat*) $\Rightarrow_0$ ′*a*). *Poly-Mapping.map f p* ⟨*proof*⟩

lift-definition *mapm-mpoly* :: ((*nat* $\Rightarrow_0$ *nat*) ⇒ ′*a* :: *zero* ⇒ ′*b* :: *zero*) ⇒ ′*a mpoly* ⇒ ′*b mpoly* **is**
λ(*f* :: (*nat* $\Rightarrow_0$ *nat*) ⇒ ′*a* ⇒ ′*b*) (*p* :: (*nat* $\Rightarrow_0$ *nat*) $\Rightarrow_0$ ′*a*).
Poly-Mapping.mapp f p ⟨*proof*⟩

lemma *poly-mapping-map-conv-mapp*: *Poly-Mapping.map f* = *Poly-Mapping.mapp* (λ-. *f*)
⟨*proof*⟩

lemma *map-mpoly-conv-mapm-mpoly*: *map-mpoly f* = *mapm-mpoly* (λ-. *f*)
⟨*proof*⟩

lemma *map-mpoly-comp*: *f 0* = *0* ⟹ *map-mpoly f* (*map-mpoly g p*) = *map-mpoly* (*f* ∘ *g*) *p*
⟨*proof*⟩

lemma *mapp-mapp*:
(⋀*x*. *f x 0* = *0*) ⟹ *Poly-Mapping.mapp f* (*Poly-Mapping.mapp g m*) =
Poly-Mapping.mapp (λ*x y*. *f x* (*g x y*)) *m*
⟨*proof*⟩

lemma *mapm-mpoly-comp*:
(⋀*x*. *f x 0* = *0*) ⟹ *mapm-mpoly f* (*mapm-mpoly g p*) = *mapm-mpoly* (λ*m c*. *f m* (*g m c*)) *p*
⟨*proof*⟩

lemma *coeff-map-mpoly*:
coeff (*map-mpoly f p*) *m* = (*if coeff p m* = *0 then 0 else f* (*coeff p m*))
⟨*proof*⟩

lemma *coeff-map-mpoly′* [*simp*]: *f 0* = *0* ⟹ *coeff* (*map-mpoly f p*) *m* = *f* (*coeff p m*)
⟨*proof*⟩

lemma *coeff-mapm-mpoly*: *coeff* (*mapm-mpoly f p*) *m* = (*if coeff p m* = *0 then 0 else f m* (*coeff p m*))
⟨*proof*⟩

lemma *coeff-mapm-mpoly′* [*simp*]: (⋀*m*. *f m 0* = *0*) ⟹ *coeff* (*mapm-mpoly f p*) *m* = *f m* (*coeff p m*)

⟨*proof*⟩

lemma *vars-map-mpoly-subset*: *vars* (*map-mpoly f p*) ⊆ *vars p*
⟨*proof*⟩

lemma *coeff-sum* [*simp*]: *coeff* (*sum f A*) *m* = (∑ *x*∈*A*. *coeff* (*f x*) *m*)
⟨*proof*⟩

lemma *coeff-Sum-any*: *finite* {*x*. *f x* ≠ *0*} ⟹ *coeff* (*Sum-any f*) *m* = *Sum-any* (λ*x*. *coeff* (*f x*) *m*)
⟨*proof*⟩

lemma *Sum-any-zeroI*: (⋀*x*. *f x* = *0*) ⟹ *Sum-any f* = *0*
⟨*proof*⟩

lemma *insertion-Prod-any*:
finite {*x*. *g x* ≠ *1*} ⟹ *insertion f* (*Prod-any g*) = *Prod-any* (λ*x*. *insertion f* (*g x*))
⟨*proof*⟩

lemma *insertion-insertion*:
insertion g (*insertion k p*) =
insertion (λ*x*. *insertion g* (*k x*)) (*map-mpoly* (*insertion g*) *p*) (**is** *?lhs* = *?rhs*)
⟨*proof*⟩

lemma *insertion-substitute-linear*:
insertion (λ*i*. *c i* ∗ *f i*) *p* =
insertion f (*mapm-mpoly* (λ*m d*. *Prod-any* (λ*i*. *c i* ^ *lookup m i*) ∗ *d*) *p*)
⟨*proof*⟩

lemma *vars-mapm-mpoly-subset*: *vars* (*mapm-mpoly f p*) ⊆ *vars p*
⟨*proof*⟩

lemma *map-mpoly-cong*:
assumes ⋀*m*. *f* (*coeff p m*) = *g* (*coeff p m*) *p* = *q*
shows *map-mpoly f p* = *map-mpoly g q*
⟨*proof*⟩

2.6 The leading monomial and leading coefficient

The leading monomial of a multivariate polynomial is the one with the largest monomial w. r. t. the monomial ordering induced by the standard variable ordering. The leading coefficient is the coefficient of the leading monomial.

As a convention, the leading monomial of the zero polynomial is defined to be the same as that of any non-constant zero polynomial, i. e. the monomial $X_1^0 \ldots X_n^0$.

lift-definition *lead-monom* :: *'a* :: *zero mpoly* ⇒ (*nat* $\Rightarrow_0$ *nat*) **is**

λf :: (*nat* $\Rightarrow_0$ *nat*) $\Rightarrow_0$ *'a*. *Max* (*insert 0* (*keys f*)) ⟨*proof*⟩

lemma *lead-monom-geI* [*intro*]:
assumes *coeff p m* $\neq$ *0*
shows *m* $\leq$ *lead-monom p*
⟨*proof*⟩

lemma *coeff-gt-lead-monom-zero* [*simp*]:
assumes *m* > *lead-monom p*
shows *coeff p m* = *0*
⟨*proof*⟩

lemma *lead-monom-nonzero-eq*:
assumes *p* $\neq$ *0*
shows *lead-monom p* = *Max* (*keys* (*mapping-of p*))
⟨*proof*⟩

lemma *lead-monom-0* [*simp*]: *lead-monom 0* = *0*
⟨*proof*⟩

lemma *lead-monom-1* [*simp*]: *lead-monom 1* = *0*
⟨*proof*⟩

lemma *lead-monom-Const* [*simp*]: *lead-monom* (*Const c*) = *0*
⟨*proof*⟩

lemma *lead-monom-uminus* [*simp*]: *lead-monom* (−*p*) = *lead-monom p*
⟨*proof*⟩

lemma *keys-mult-const* [*simp*]:
fixes *c* :: *'a* :: {*semiring-0*, *semiring-no-zero-divisors*}
assumes *c* $\neq$ *0*
shows *keys* (*Poly-Mapping.map* ((∗) *c*) *p*) = *keys p*
⟨*proof*⟩

lemma *lead-monom-eq-0-iff*: *lead-monom p* = *0* $\longleftrightarrow$ *vars p* = {}
⟨*proof*⟩

lemma *lead-monom-monom*: *lead-monom* (*monom m c*) = (*if c* = *0 then 0 else m*)
⟨*proof*⟩

lemma *lead-monom-monom'* [*simp*]: *c* $\neq$ *0* $\Longrightarrow$ *lead-monom* (*monom m c*) = *m*
⟨*proof*⟩

lemma *lead-monom-numeral* [*simp*]: *lead-monom* (*numeral n*) = *0*
⟨*proof*⟩

lemma *lead-monom-add*: *lead-monom* (*p* + *q*) $\leq$ *max* (*lead-monom p*) (*lead-monom*

q)
⟨*proof*⟩

lemma *lead-monom-diff*: *lead-monom* (*p* − *q*) ≤ *max* (*lead-monom p*) (*lead-monom q*)
⟨*proof*⟩

definition *lead-coeff* **where** *lead-coeff p* = *coeff p* (*lead-monom p*)

lemma *vars-empty-iff*: *vars p* = {} ⟷ *p* = *Const* (*lead-coeff p*)
⟨*proof*⟩

lemma *lead-coeff-0* [*simp*]: *lead-coeff 0* = *0*
⟨*proof*⟩

lemma *lead-coeff-1* [*simp*]: *lead-coeff 1* = *1*
⟨*proof*⟩

lemma *lead-coeff-Const* [*simp*]: *lead-coeff* (*Const c*) = *c*
⟨*proof*⟩

lemma *lead-coeff-monom* [*simp*]: *lead-coeff* (*monom p c*) = *c*
⟨*proof*⟩

lemma *lead-coeff-nonzero* [*simp*]: *p* ≠ *0* ⟹ *lead-coeff p* ≠ *0*
⟨*proof*⟩

lemma
fixes *c* :: *'a* :: *semiring-0*
assumes *c* ∗ *lead-coeff p* ≠ *0*
shows *lead-monom-smult* [*simp*]: *lead-monom* (*smult c p*) = *lead-monom p*
and *lead-coeff-smult* [*simp*]: *lead-coeff* (*smult c p*) = *c* ∗ *lead-coeff p*
⟨*proof*⟩

lemma *lead-coeff-mult-aux*:
coeff (*p* ∗ *q*) (*lead-monom p* + *lead-monom q*) = *lead-coeff p* ∗ *lead-coeff q*
⟨*proof*⟩

lemma *lead-monom-mult-le*: *lead-monom* (*p* ∗ *q*) ≤ *lead-monom p* + *lead-monom q*
⟨*proof*⟩

lemma *lead-monom-mult*:
assumes *lead-coeff p* ∗ *lead-coeff q* ≠ *0*
shows *lead-monom* (*p* ∗ *q*) = *lead-monom p* + *lead-monom q*
⟨*proof*⟩

lemma *lead-coeff-mult*:

assumes *lead-coeff p* ∗ *lead-coeff q* ≠ *0*
shows *lead-coeff* (*p* ∗ *q*) = *lead-coeff p* ∗ *lead-coeff q*
⟨*proof*⟩

lemma *keys-lead-monom-subset*: *keys* (*lead-monom p*) ⊆ *vars p*
⟨*proof*⟩

lemma
assumes ($\prod$ *i*∈*A*. *lead-coeff* (*f i*)) ≠ *0*
shows *lead-monom-prod*: *lead-monom* ($\prod$ *i*∈*A*. *f i*) = ($\sum$ *i*∈*A*. *lead-monom* (*f i*)) (**is** *?th1*)
and *lead-coeff-prod*: *lead-coeff* ($\prod$ *i*∈*A*. *f i*) = ($\prod$ *i*∈*A*. *lead-coeff* (*f i*)) (**is** *?th2*)
⟨*proof*⟩

lemma *lead-monom-sum-le*: ($\bigwedge$*x*. *x* ∈ *X* ⟹ *lead-monom* (*h x*) ≤ *ub*) ⟹ *lead-monom* (*sum h X*) ≤ *ub*
⟨*proof*⟩

The leading monomial of a sum where the leading monomial the summands are distinct is simply the maximum of the leading monomials.

lemma *lead-monom-sum*:
assumes *inj-on* (*lead-monom* ∘ *h*) *X* **and** *finite X* **and** *X* ≠ {} **and** $\bigwedge$*x*. *x* ∈ *X* ⟹ *h x* ≠ *0*
defines *m* ≡ *Max* ((*lead-monom* ∘ *h*) ‘ *X*)
shows *lead-monom* ($\sum$ *x*∈*X*. *h x*) = *m*
⟨*proof*⟩

lemma *lead-coeff-eq-0-iff* [*simp*]: *lead-coeff p* = *0* ⟷ *p* = *0*
⟨*proof*⟩

lemma
fixes *f* :: *-* ⇒ *'a* :: *semidom mpoly*
assumes $\bigwedge$*i*. *i* ∈ *A* ⟹ *f i* ≠ *0*
shows *lead-monom-prod'* [*simp*]: *lead-monom* ($\prod$ *i*∈*A*. *f i*) = ($\sum$ *i*∈*A*. *lead-monom* (*f i*)) (**is** *?th1*)
and *lead-coeff-prod'* [*simp*]: *lead-coeff* ($\prod$ *i*∈*A*. *f i*) = ($\prod$ *i*∈*A*. *lead-coeff* (*f i*)) (**is** *?th2*)
⟨*proof*⟩

lemma
fixes *p* :: *'a* :: *comm-semiring-1 mpoly*
assumes *lead-coeff p* ^ *n* ≠ *0*
shows *lead-monom-power*: *lead-monom* (*p* ^ *n*) = *of-nat n* ∗ *lead-monom p*
and *lead-coeff-power*: *lead-coeff* (*p* ^ *n*) = *lead-coeff p* ^ *n*
⟨*proof*⟩

lemma
fixes *p* :: *'a* :: *semidom mpoly*

assumes $p \neq 0$
shows *lead-monom-power'* [*simp*]: *lead-monom* $(p \hat{} n)$ = *of-nat* n $*$ *lead-monom* p
and *lead-coeff-power'* [*simp*]: *lead-coeff* $(p \hat{} n)$ = *lead-coeff* $p \hat{} n$
⟨*proof*⟩

2.7 Turning a set of variables into a monomial

Given a finite set $\{X_1, \ldots, X_n\}$ of variables, the following is the monomial $X_1 \ldots X_n$:

lift-definition *monom-of-set* :: *nat set* ⇒ (*nat* $\Rightarrow_0$ *nat*) **is**
λ*X x*. *if finite X* ∧ *x* ∈ *X then 1 else 0*
⟨*proof*⟩

lemma *lookup-monom-of-set*:
Poly-Mapping.lookup (*monom-of-set X*) *i* = (*if finite X* ∧ *i* ∈ *X then 1 else 0*)
⟨*proof*⟩

lemma *lookup-monom-of-set-1* [*simp*]:
finite X ⟹ *i* ∈ *X* ⟹ *Poly-Mapping.lookup* (*monom-of-set X*) *i* = *1*
and *lookup-monom-of-set-0* [*simp*]:
i ∉ *X* ⟹ *Poly-Mapping.lookup* (*monom-of-set X*) *i* = *0*
⟨*proof*⟩

lemma *keys-monom-of-set*: *keys* (*monom-of-set X*) = (*if finite X then X else* {})
⟨*proof*⟩

lemma *keys-monom-of-set-finite* [*simp*]: *finite X* ⟹ *keys* (*monom-of-set X*) = *X*
⟨*proof*⟩

lemma *monom-of-set-eq-iff* [*simp*]: *finite X* ⟹ *finite Y* ⟹ *monom-of-set X* = *monom-of-set Y* ⟷ *X* = *Y*
⟨*proof*⟩

lemma *monom-of-set-empty* [*simp*]: *monom-of-set* {} = *0*
⟨*proof*⟩

lemma *monom-of-set-eq-zero-iff* [*simp*]: *monom-of-set X* = *0* ⟷ *infinite X* ∨ *X* = {}
⟨*proof*⟩

lemma *zero-eq-monom-of-set-iff* [*simp*]: *0* = *monom-of-set X* ⟷ *infinite X* ∨ *X* = {}
⟨*proof*⟩

2.8 Permuting the variables of a polynomial

Next, we define the operation of permuting the variables of a monomial and polynomial.

lift-definition *permutep* :: $('a \Rightarrow 'a) \Rightarrow ('a \Rightarrow_0 'b) \Rightarrow ('a \Rightarrow_0 'b :: zero)$ **is**
$\lambda f\ p.$ *if bij f then p* $\circ$ *f else p*
⟨*proof*⟩

lift-definition *mpoly-map-vars* :: $(nat \Rightarrow nat) \Rightarrow 'a :: zero\ mpoly \Rightarrow 'a\ mpoly$ **is**
$\lambda f\ p.$ *permutep* (*permutep f*) *p* ⟨*proof*⟩

lemma *keys-permutep*: *bij f* $\Longrightarrow$ *keys* (*permutep f m*) = *f* −‘ *keys m*
⟨*proof*⟩

lemma *permutep-id''* [*simp*]: *permutep id* = *id*
⟨*proof*⟩

lemma *permutep-id'''* [*simp*]: *permutep* ($\lambda x.\ x$) = *id*
⟨*proof*⟩

lemma *permutep-0* [*simp*]: *permutep f 0* = *0*
⟨*proof*⟩

lemma *permutep-single*:
bij f $\Longrightarrow$ *permutep f* (*Poly-Mapping.single a b*) = *Poly-Mapping.single* (*inv-into UNIV f a*) *b*
⟨*proof*⟩

lemma *mpoly-map-vars-id* [*simp*]: *mpoly-map-vars id* = *id*
⟨*proof*⟩

lemma *mpoly-map-vars-id'* [*simp*]: *mpoly-map-vars* ($\lambda x.\ x$) = *id*
⟨*proof*⟩

lemma *lookup-permutep*:
Poly-Mapping.lookup (*permutep f m*) *x* = (*if bij f then Poly-Mapping.lookup m* (*f x*) *else Poly-Mapping.lookup m x*)
⟨*proof*⟩

lemma *inj-permutep* [*intro*]: *inj* (*permutep* ($f :: 'a \Rightarrow 'a$) :: - $\Rightarrow 'a \Rightarrow_0 'b :: zero$)
⟨*proof*⟩

lemma *surj-permutep* [*intro*]: *surj* (*permutep* ($f :: 'a \Rightarrow 'a$) :: - $\Rightarrow 'a \Rightarrow_0 'b :: zero$)
⟨*proof*⟩

lemma *bij-permutep* [*intro*]: *bij* (*permutep f*)
⟨*proof*⟩

lemma *mpoly-map-vars-map-mpoly*:
mpoly-map-vars f (*map-mpoly g p*) = *map-mpoly g* (*mpoly-map-vars f p*)
⟨*proof*⟩

lemma *coeff-mpoly-map-vars*:
fixes *f* :: *nat* ⇒ *nat* **and** *p* :: *'a* :: *zero mpoly*
assumes *bij f*
shows *MPoly-Type.coeff* (*mpoly-map-vars f p*) *mon* =
MPoly-Type.coeff p (*permutep f mon*)
⟨*proof*⟩

lemma *permutep-monom-of-set*:
assumes *bij f*
shows *permutep f* (*monom-of-set A*) = *monom-of-set* (*f* −' *A*)
⟨*proof*⟩

lemma *permutep-comp*: *bij f* ⟹ *bij g* ⟹ *permutep* (*f* ∘ *g*) = *permutep g* ∘ *permutep f*
⟨*proof*⟩

lemma *permutep-comp'*: *bij f* ⟹ *bij g* ⟹ *permutep* (*f* ∘ *g*) *mon* = *permutep g* (*permutep f mon*)
⟨*proof*⟩

lemma *mpoly-map-vars-comp*:
bij f ⟹ *bij g* ⟹ *mpoly-map-vars f* (*mpoly-map-vars g p*) = *mpoly-map-vars* (*f* ∘ *g*) *p*
⟨*proof*⟩

lemma *permutep-id* [*simp*]: *permutep id mon* = *mon*
⟨*proof*⟩

lemma *permutep-id'* [*simp*]: *permutep* (λ*x*. *x*) *mon* = *mon*
⟨*proof*⟩

lemma *inv-permutep* [*simp*]:
fixes *f* :: *'a* ⇒ *'a*
assumes *bij f*
shows *inv-into UNIV* (*permutep f*) = *permutep* (*inv-into UNIV f*)
⟨*proof*⟩

lemma *mpoly-map-vars-monom*:
bij f ⟹ *mpoly-map-vars f* (*monom m c*) = *monom* (*permutep* (*inv-into UNIV f*) *m*) *c*
⟨*proof*⟩

lemma *vars-mpoly-map-vars*:
fixes *f* :: *nat* ⇒ *nat* **and** *p* :: *'a* :: *zero mpoly*
assumes *bij f*

shows *vars* (*mpoly-map-vars f p*) = *f* ‘ *vars p*
⟨*proof*⟩

lemma *permutep-eq-monom-of-set-iff* [*simp*]:
assumes *bij f*
shows *permutep f mon* = *monom-of-set A* ⟷ *mon* = *monom-of-set* (*f* ‘ *A*)
⟨*proof*⟩

lemma *permutep-monom-of-set-permutes* [*simp*]:
assumes π *permutes A*
shows *permutep* π (*monom-of-set A*) = *monom-of-set A*
⟨*proof*⟩

lemma *mpoly-map-vars-0* [*simp*]: *mpoly-map-vars f 0* = *0*
⟨*proof*⟩

lemma *permutep-eq-0-iff* [*simp*]: *permutep f m* = *0* ⟷ *m* = *0*
⟨*proof*⟩

lemma *mpoly-map-vars-1* [*simp*]: *mpoly-map-vars f 1* = *1*
⟨*proof*⟩

lemma *permutep-Const*$_0$ [*simp*]: (⋀*x*. *f x* = *0* ⟷ *x* = *0*) ⟹ *permutep f* (*Const*$_0$ *c*) = *Const*$_0$ *c*
⟨*proof*⟩

lemma *permutep-add* [*simp*]: *permutep f* (*m1* + *m2*) = *permutep f m1* + *permutep f m2*
⟨*proof*⟩

lemma *permutep-diff* [*simp*]: *permutep f* (*m1* − *m2*) = *permutep f m1* − *permutep f m2*
⟨*proof*⟩

lemma *permutep-uminus* [*simp*]: *permutep f* (−*m*) = −*permutep f m*
⟨*proof*⟩

lemma *permutep-mult* [*simp*]:
(⋀*x y*. *f* (*x* + *y*) = *f x* + *f y*) ⟹ *permutep f* (*m1* ∗ *m2*) = *permutep f m1* ∗ *permutep f m2*
⟨*proof*⟩

lemma *mpoly-map-vars-Const* [*simp*]: *mpoly-map-vars f* (*Const c*) = *Const c*
⟨*proof*⟩

lemma *mpoly-map-vars-add* [*simp*]: *mpoly-map-vars f* (*p* + *q*) = *mpoly-map-vars f p* + *mpoly-map-vars f q*
⟨*proof*⟩

lemma *mpoly-map-vars-diff* [*simp*]: *mpoly-map-vars f* $(p - q)$ = *mpoly-map-vars f p* $-$ *mpoly-map-vars f q*
⟨*proof*⟩

lemma *mpoly-map-vars-uminus* [*simp*]: *mpoly-map-vars f* $(-p)$ = $-$*mpoly-map-vars f p*
⟨*proof*⟩

lemma *permutep-smult*:
permutep (*permutep f*) (*Poly-Mapping.map* ((∗) *c*) *p*) =
Poly-Mapping.map ((∗) *c*) (*permutep* (*permutep f*) *p*)
⟨*proof*⟩

lemma *mpoly-map-vars-smult* [*simp*]: *mpoly-map-vars f* (*smult c p*) = *smult c* (*mpoly-map-vars f p*)
⟨*proof*⟩

lemma *mpoly-map-vars-mult* [*simp*]: *mpoly-map-vars f* $(p * q)$ = *mpoly-map-vars f p* ∗ *mpoly-map-vars f q*
⟨*proof*⟩

lemma *mpoly-map-vars-sum* [*simp*]: *mpoly-map-vars f* (*sum g A*) = ($\sum x{\in}A$. *mpoly-map-vars f* (*g x*))
⟨*proof*⟩

lemma *mpoly-map-vars-prod* [*simp*]: *mpoly-map-vars f* (*prod g A*) = ($\prod x{\in}A$. *mpoly-map-vars f* (*g x*))
⟨*proof*⟩

lemma *mpoly-map-vars-eq-0-iff* [*simp*]: *mpoly-map-vars f p* = *0* ⟷ *p* = *0*
⟨*proof*⟩

lemma *permutep-eq-iff* [*simp*]: *permutep f p* = *permutep f q* ⟷ *p* = *q*
⟨*proof*⟩

lemma *mpoly-map-vars-Sum-any* [*simp*]:
mpoly-map-vars f (*Sum-any g*) = *Sum-any* (λ*x*. *mpoly-map-vars f* (*g x*))
⟨*proof*⟩

lemma *mpoly-map-vars-power* [*simp*]: *mpoly-map-vars f* (*p* ^ *n*) = *mpoly-map-vars f p* ^ *n*
⟨*proof*⟩

lemma *mpoly-map-vars-monom-single* [*simp*]:
assumes *bij f*
shows *mpoly-map-vars f* (*monom* (*Poly-Mapping.single i n*) *c*) =
monom (*Poly-Mapping.single* (*f i*) *n*) *c*
⟨*proof*⟩

lemma *insertion-mpoly-map-vars*:
assumes *bij f*
shows $insertion\ g\ (mpoly\text{-}map\text{-}vars\ f\ p) = insertion\ (g \circ f)\ p$
⟨*proof*⟩

lemma *permutep-cong*:
assumes $f\ permutes\ (-keys\ p)\ g\ permutes\ (-keys\ p)\ p = q$
shows $permutep\ f\ p = permutep\ g\ q$
⟨*proof*⟩

lemma *mpoly-map-vars-cong*:
assumes $f\ permutes\ (-vars\ p)\ g\ permutes\ (-vars\ q)\ p = q$
shows $mpoly\text{-}map\text{-}vars\ f\ p = mpoly\text{-}map\text{-}vars\ g\ (q :: 'a :: zero\ mpoly)$
⟨*proof*⟩

2.9 Symmetric polynomials

A polynomial is symmetric on a set of variables if it is invariant under any permutation of that set.

definition $symmetric\text{-}mpoly :: nat\ set \Rightarrow 'a :: zero\ mpoly \Rightarrow bool$ **where**
$symmetric\text{-}mpoly\ A\ p = (\forall \pi.\ \pi\ permutes\ A \longrightarrow mpoly\text{-}map\text{-}vars\ \pi\ p = p)$

lemma *symmetric-mpoly-empty* [*simp*, *intro*]: $symmetric\text{-}mpoly\ \{\}\ p$
⟨*proof*⟩

A polynomial is trivially symmetric on any set of variables that do not occur in it.

lemma *symmetric-mpoly-orthogonal*:
assumes $vars\ p \cap A = \{\}$
shows $symmetric\text{-}mpoly\ A\ p$
⟨*proof*⟩

lemma *symmetric-mpoly-monom* [*intro*]:
assumes $keys\ m \cap A = \{\}$
shows $symmetric\text{-}mpoly\ A\ (monom\ m\ c)$
⟨*proof*⟩

lemma *symmetric-mpoly-subset*:
assumes $symmetric\text{-}mpoly\ A\ p\ B \subseteq A$
shows $symmetric\text{-}mpoly\ B\ p$
⟨*proof*⟩

If a polynomial is symmetric over some set of variables, that set must either be a subset of the variables occurring in the polynomial or disjoint from it.

lemma *symmetric-mpoly-imp-orthogonal-or-subset*:
assumes $symmetric\text{-}mpoly\ A\ p$
shows $vars\ p \cap A = \{\} \lor A \subseteq vars\ p$
⟨*proof*⟩

Symmetric polynomials are closed under ring operations.

lemma *symmetric-mpoly-add* [*intro*]:
symmetric-mpoly A p $\Longrightarrow$ *symmetric-mpoly A q* $\Longrightarrow$ *symmetric-mpoly A* (*p* + *q*)
⟨*proof*⟩

lemma *symmetric-mpoly-diff* [*intro*]:
symmetric-mpoly A p $\Longrightarrow$ *symmetric-mpoly A q* $\Longrightarrow$ *symmetric-mpoly A* (*p* − *q*)
⟨*proof*⟩

lemma *symmetric-mpoly-uminus* [*intro*]: *symmetric-mpoly A p* $\Longrightarrow$ *symmetric-mpoly A* (−*p*)
⟨*proof*⟩

lemma *symmetric-mpoly-uminus-iff* [*simp*]: *symmetric-mpoly A* (−*p*) $\longleftrightarrow$ *symmetric-mpoly A p*
⟨*proof*⟩

lemma *symmetric-mpoly-smult* [*intro*]: *symmetric-mpoly A p* $\Longrightarrow$ *symmetric-mpoly A* (*smult c p*)
⟨*proof*⟩

lemma *symmetric-mpoly-mult* [*intro*]:
symmetric-mpoly A p $\Longrightarrow$ *symmetric-mpoly A q* $\Longrightarrow$ *symmetric-mpoly A* (*p* ∗ *q*)
⟨*proof*⟩

lemma *symmetric-mpoly-0* [*simp*, *intro*]: *symmetric-mpoly A 0*
and *symmetric-mpoly-1* [*simp*, *intro*]: *symmetric-mpoly A 1*
and *symmetric-mpoly-Const* [*simp*, *intro*]: *symmetric-mpoly A* (*Const c*)
⟨*proof*⟩

lemma *symmetric-mpoly-power* [*intro*]:
symmetric-mpoly A p $\Longrightarrow$ *symmetric-mpoly A* (*p* ^ *n*)
⟨*proof*⟩

lemma *symmetric-mpoly-sum* [*intro*]:
($\bigwedge$*i*. *i* ∈ *B* $\Longrightarrow$ *symmetric-mpoly A* (*f i*)) $\Longrightarrow$ *symmetric-mpoly A* (*sum f B*)
⟨*proof*⟩

lemma *symmetric-mpoly-prod* [*intro*]:
($\bigwedge$*i*. *i* ∈ *B* $\Longrightarrow$ *symmetric-mpoly A* (*f i*)) $\Longrightarrow$ *symmetric-mpoly A* (*prod f B*)
⟨*proof*⟩

An symmetric sum or product over polynomials yields a symmetric polynomial:

lemma *symmetric-mpoly-symmetric-sum*:
assumes *g permutes X*
assumes $\bigwedge$*x* π. *x* ∈ *X* $\Longrightarrow$ π *permutes A* $\Longrightarrow$ *mpoly-map-vars* π (*f x*) = *f* (*g x*)
shows *symmetric-mpoly A* ($\sum$*x*∈*X*. *f x*)
⟨*proof*⟩

lemma *symmetric-mpoly-symmetric-prod*:
assumes *g permutes X*
assumes $\bigwedge x\ \pi.\ x \in X \Longrightarrow \pi$ *permutes A* $\Longrightarrow$ *mpoly-map-vars* π $(f\ x) = f\ (g\ x)$
shows *symmetric-mpoly A* $(\prod x{\in}X.\ f\ x)$
⟨*proof*⟩

If p is a polynomial that is symmetric on some subset of variables A, then for the leading monomial of p, the exponents of these variables are decreasing w. r. t. the variable ordering.

theorem *lookup-lead-monom-decreasing*:
assumes *symmetric-mpoly A p*
defines $m \equiv$ *lead-monom p*
assumes $i \in A$ $j \in A$ $i \le j$
shows *lookup m i* $\ge$ *lookup m j*
⟨*proof*⟩

2.10 The elementary symmetric polynomials

The k-th elementary symmetric polynomial for a finite set of variables A, with k ranging between 1 and $|A|$, is the sum of the product of all subsets of A with cardinality k:

lift-definition *sym-mpoly-aux* :: *nat set* $\Rightarrow$ *nat* $\Rightarrow$ (*nat* $\Rightarrow_0$ *nat*) $\Rightarrow_0$ *'a* :: {*zero-neq-one*} **is**
$\lambda X\ k\ mon.$ *if finite X* $\wedge$ ($\exists\ Y.\ Y \subseteq X \wedge$ *card Y* $= k \wedge$ *mon* = *monom-of-set Y*) *then 1 else 0*
⟨*proof*⟩

lemma *lookup-sym-mpoly-aux*:
Poly-Mapping.lookup (*sym-mpoly-aux X k*) *mon* =
(*if finite X* $\wedge$ ($\exists\ Y.\ Y \subseteq X \wedge$ *card Y* $= k \wedge$ *mon* = *monom-of-set Y*) *then 1 else 0*)
⟨*proof*⟩

lemma *lookup-sym-mpoly-aux-monom-of-set* [*simp*]:
assumes *finite X* $Y \subseteq X$ *card Y* $= k$
shows *Poly-Mapping.lookup* (*sym-mpoly-aux X k*) (*monom-of-set Y*) = *1*
⟨*proof*⟩

lemma *keys-sym-mpoly-aux*: $m \in$ *keys* (*sym-mpoly-aux A k*) $\Longrightarrow$ *keys m* $\subseteq A$
⟨*proof*⟩

lift-definition *sym-mpoly* :: *nat set* $\Rightarrow$ *nat* $\Rightarrow$ *'a* :: {*zero-neq-one*} *mpoly* **is**
sym-mpoly-aux ⟨*proof*⟩

lemma *vars-sym-mpoly-subset*: *vars* (*sym-mpoly A k*) $\subseteq A$
⟨*proof*⟩

lemma *coeff-sym-mpoly*:
MPoly-Type.coeff (*sym-mpoly X k*) *mon* =
(*if finite X* ∧ (∃ *Y*. *Y* ⊆ *X* ∧ *card Y* = *k* ∧ *mon* = *monom-of-set Y*) *then 1 else 0*)
⟨*proof*⟩

lemma *sym-mpoly-infinite*: ¬*finite A* ⟹ *sym-mpoly A k* = *0*
⟨*proof*⟩

lemma *sym-mpoly-altdef*: *sym-mpoly A k* = (∑ *X* | *X* ⊆ *A* ∧ *card X* = *k*. *monom* (*monom-of-set X*) *1*)
⟨*proof*⟩

lemma *coeff-sym-mpoly-monom-of-set* [*simp*]:
assumes *finite X Y* ⊆ *X card Y* = *k*
shows *MPoly-Type.coeff* (*sym-mpoly X k*) (*monom-of-set Y*) = *1*
⟨*proof*⟩

lemma *coeff-sym-mpoly-0*: *coeff* (*sym-mpoly X k*) *0* = (*if finite X* ∧ *k* = *0 then 1 else 0*)
⟨*proof*⟩

lemma *symmetric-sym-mpoly* [*intro*]:
assumes *A* ⊆ *B*
shows *symmetric-mpoly A* (*sym-mpoly B k* :: ′*a* :: *zero-neq-one mpoly*)
⟨*proof*⟩

lemma *insertion-sym-mpoly*:
assumes *finite X*
shows *insertion f* (*sym-mpoly X k*) = (∑ *Y* | *Y* ⊆ *X* ∧ *card Y* = *k*. *prod f Y*)
⟨*proof*⟩

lemma *sym-mpoly-nz* [*simp*]:
assumes *finite A k* ≤ *card A*
shows *sym-mpoly A k* ≠ (*0* :: ′*a* :: *zero-neq-one mpoly*)
⟨*proof*⟩

lemma *coeff-sym-mpoly-0-or-1*: *coeff* (*sym-mpoly A k*) *m* ∈ {*0*, *1*}
⟨*proof*⟩

lemma *lead-coeff-sym-mpoly* [*simp*]:
assumes *finite A k* ≤ *card A*
shows *lead-coeff* (*sym-mpoly A k*) = *1*
⟨*proof*⟩

lemma *lead-monom-sym-mpoly*:
assumes *sorted xs distinct xs k* ≤ *length xs*
shows *lead-monom* (*sym-mpoly* (*set xs*) *k* :: ′*a* :: *zero-neq-one mpoly*) =

monom-of-set (set (take k xs)) (**is** *lead-monom ?p* = -)
⟨*proof*⟩

2.11 Induction on the leading monomial

We show that the monomial ordering for a fixed set of variables is well-founded, so we can perform induction on the leading monomial of a polynomial.

definition *monom-less-on* **where**
monom-less-on A = {(*m1*, *m2*). *m1* < *m2* ∧ *keys m1* ⊆ *A* ∧ *keys m2* ⊆ *A*}

lemma *wf-monom-less-on*:
assumes *finite A*
shows *wf* (*monom-less-on A* :: ((*nat* $\Rightarrow_0$ *'b* :: {*zero*, *wellorder*}) × -) *set*)
⟨*proof*⟩

lemma *lead-monom-induct* [*consumes 2*, *case-names less*]:
fixes *p* :: *'a* :: *zero mpoly*
assumes *fin*: *finite A* **and** *vars*: *vars p* ⊆ *A*
assumes *IH*: ⋀*p*. *vars p* ⊆ *A* ⟹
(⋀*p'*. *vars p'* ⊆ *A* ⟹ *lead-monom p'* < *lead-monom p* ⟹ *P p'*)
⟹ *P p*
shows *P p*
⟨*proof*⟩

lemma *lead-monom-induct'* [*case-names less*]:
fixes *p* :: *'a* :: *zero mpoly*
assumes *IH*: ⋀*p*. (⋀*p'*. *vars p'* ⊆ *vars p* ⟹ *lead-monom p'* < *lead-monom p* ⟹ *P p'*) ⟹ *P p*
shows *P p*
⟨*proof*⟩

2.12 The fundamental theorem of symmetric polynomials

lemma *lead-coeff-sym-mpoly-powerprod*:
assumes *finite A* ⋀*x*. *x* ∈ *X* ⟹ *f x* ∈ {*1*..*card A*}
shows *lead-coeff* (∏ *x*∈*X*. *sym-mpoly A* (*f* (*x*::*'a*)) ^ *g x*) = *1*
⟨*proof*⟩

context
fixes *A* :: *nat set* **and** *xs n f* **and** *decr* :: *'a* :: *comm-ring-1 mpoly* ⇒ *bool*
defines *xs* ≡ *sorted-list-of-set A*
defines *n* ≡ *card A*
defines *f* ≡ (λ*i*. *if i* < *n then xs* ! *i else 0*)
defines *decr* ≡ (λ*p*. ∀ *i*∈*A*. ∀ *j*∈*A*. *i* ≤ *j* ⟶
lookup (*lead-monom p*) *i* ≥ *lookup* (*lead-monom p*) *j*)
begin

The computation of the witness for the fundamental theorem works like this:

Given some polynomial p (that is assumed to be symmetric in the variables in A), we inspect its leading monomial, which is of the form $cX_1^{i_1} \dots X_n i_n$ where the $A = \{X_1, \dots, X_n\}$, c contains only variables not in A, and the sequence i_j is decreasing. The latter holds because p is symmetric.

Now, we form the polynomial $q := ce_1^{i_1 - i_2} e_2^{i_2 - i_3} \dots e_n^{i_n}$, which has the same leading term as p. Then $p - q$ has a smaller leading monomial, so by induction, we can assume it to be of the required form and obtain a witness for $p - q$.

Now, we only need to add $cY_1^{i_1 - i_2} \dots Y_n^{i_n}$ to that witness and we obtain a witness for p.

definition *fund-sym-step-coeff* :: *'a mpoly* ⇒ *'a mpoly* **where**
fund-sym-step-coeff p = *monom* (*restrictpm* (−*A*) (*lead-monom p*)) (*lead-coeff p*)

definition *fund-sym-step-monom* :: *'a mpoly* ⇒ (*nat* ⇒₀ *nat*) **where**
fund-sym-step-monom p = (
let g = (λ*i. if i* < *n then lookup* (*lead-monom p*) (*f i*) *else 0*)
in (∑ *i*<*n. Poly-Mapping.single* (*Suc i*) (*g i* − *g* (*Suc i*))))

definition *fund-sym-step-poly* :: *'a mpoly* ⇒ *'a mpoly* **where**
fund-sym-step-poly p = (
let g = (λ*i. if i* < *n then lookup* (*lead-monom p*) (*f i*) *else 0*)
in fund-sym-step-coeff p ∗ (∏ *i*<*n. sym-mpoly A* (*Suc i*) ^ (*g i* − *g* (*Suc i*))))

The following function computes the witness, with the convention that it returns a constant polynomial if the input was not symmetric:

function (*domintros*) *fund-sym-poly-wit* :: *'a* :: *comm-ring-1 mpoly* ⇒ *'a mpoly mpoly* **where**
fund-sym-poly-wit p =
(*if* ¬*symmetric-mpoly A p* ∨ *lead-monom p* = *0* ∨ *vars p* ∩ *A* = {} *then Const p else*
fund-sym-poly-wit (*p* − *fund-sym-step-poly p*) +
monom (*fund-sym-step-monom p*) (*fund-sym-step-coeff p*))
⟨*proof*⟩

lemma *coeff-fund-sym-step-coeff*: *coeff* (*fund-sym-step-coeff p*) *m* ∈ {*lead-coeff p*, *0*}
⟨*proof*⟩

lemma *vars-fund-sym-step-coeff*: *vars* (*fund-sym-step-coeff p*) ⊆ *vars p* − *A*
⟨*proof*⟩

lemma *keys-fund-sym-step-monom*: *keys* (*fund-sym-step-monom p*) ⊆ {*1..n*}
⟨*proof*⟩

lemma *coeff-fund-sym-step-poly*:
assumes *C*: ∀ *m. coeff p m* ∈ *C* **and** *ring-closed C*
shows *coeff* (*fund-sym-step-poly p*) *m* ∈ *C*

⟨*proof*⟩

We now show various relevant properties of the subtracted polynomial:

1. Its leading term is the same as that of the input polynomial.
2. It contains now new variables.
3. It is symmetric in the variables in A.

lemma *fund-sym-step-poly*:
shows *finite A* $\Longrightarrow$ *p* $\neq$ *0* $\Longrightarrow$ *decr p* $\Longrightarrow$ *lead-monom* (*fund-sym-step-poly p*) = *lead-monom p*
and *finite A* $\Longrightarrow$ *p* $\neq$ *0* $\Longrightarrow$ *decr p* $\Longrightarrow$ *lead-coeff* (*fund-sym-step-poly p*) = *lead-coeff p*
and *finite A* $\Longrightarrow$ *p* $\neq$ *0* $\Longrightarrow$ *decr p* $\Longrightarrow$ *fund-sym-step-poly p* =
fund-sym-step-coeff p ∗ ($\prod$ *x*. *sym-mpoly A x* ˆ *lookup* (*fund-sym-step-monom p*) *x*)
and *vars* (*fund-sym-step-poly p*) ⊆ *vars p* ∪ *A*
and *symmetric-mpoly A* (*fund-sym-step-poly p*)
⟨*proof*⟩

If the input is well-formed, a single step of the procedure always decreases the leading monomial.

lemma *lead-monom-fund-sym-step-poly-less*:
assumes *finite A* **and** *lead-monom p* $\neq$ *0* **and** *decr p*
shows *lead-monom* (*p* − *fund-sym-step-poly p*) < *lead-monom p*
⟨*proof*⟩

Finally, we prove that the witness is indeed well-defined for all inputs.

lemma *fund-sym-poly-wit-dom-aux*:
assumes *finite B vars p* ⊆ *B A* ⊆ *B*
shows *fund-sym-poly-wit-dom p*
⟨*proof*⟩

lemma *fund-sym-poly-wit-dom* [*intro*]: *fund-sym-poly-wit-dom p*
⟨*proof*⟩

termination *fund-sym-poly-wit*
⟨*proof*⟩

Next, we prove that our witness indeed fulfils all the properties stated by the fundamental theorem:

1. If the original polynomial was in $R[X_1, \ldots, X_n, \ldots, X_m]$ where the X_1 to X_n are the symmetric variables, then the witness is a polynomial in $R[X_{n+1}, \ldots, X_m][Y_1, \ldots, Y_n]$. This means that its coefficients are polynomials in the variables of the original polynomial, minus the symmetric ones, and the (new and independent) variables of the witness polynomial range from 1 to n.

2. Substituting the i-th symmetric polynomial $e_i(X_1, \ldots, X_n)$ for the Y_i variable for every i yields the original polynomial.

3. The coefficient ring R need not be the entire type; if the coefficients of the original polynomial are in some subring, then the coefficients of the coefficients of the witness also do.

lemma *fund-sym-poly-wit-coeffs-aux*:
  assumes *finite B vars p* ⊆ *B symmetric-mpoly A p A* ⊆ *B*
  shows *vars* (*coeff* (*fund-sym-poly-wit p*) *m*) ⊆ *B* − *A*
  ⟨*proof*⟩

lemma *fund-sym-poly-wit-coeffs*:
  assumes *symmetric-mpoly A p*
  shows *vars* (*coeff* (*fund-sym-poly-wit p*) *m*) ⊆ *vars p* − *A*
⟨*proof*⟩

lemma *fund-sym-poly-wit-vars*: *vars* (*fund-sym-poly-wit p*) ⊆ {*1..n*}
⟨*proof*⟩

lemma *fund-sym-poly-wit-insertion-aux*:
  assumes *finite B vars p* ⊆ *B symmetric-mpoly A p A* ⊆ *B*
  shows *insertion* (*sym-mpoly A*) (*fund-sym-poly-wit p*) = *p*
  ⟨*proof*⟩

lemma *fund-sym-poly-wit-insertion*:
  assumes *symmetric-mpoly A p*
  shows *insertion* (*sym-mpoly A*) (*fund-sym-poly-wit p*) = *p*
⟨*proof*⟩

lemma *fund-sym-poly-wit-coeff*:
  assumes ∀ *m*. *coeff p m* ∈ *C ring-closed C*
  shows ∀ *m m′*. *coeff* (*coeff* (*fund-sym-poly-wit p*) *m*) *m′* ∈ *C*
  ⟨*proof*⟩

2.13 Uniqueness

Next, we show that the polynomial representation of a symmetric polynomial in terms of the elementary symmetric polynomials not only exists, but is unique.

The key property here is that products of powers of elementary symmetric polynomials uniquely determine the exponent vectors, i. e. if $e_1, \ldots, e_n$ are the elementary symmetric polynomials, $a = (a_1, \ldots, a_n)$ and $b = (b_1, \ldots, b_n)$ are vectors of natural numbers, then:

$$e_1^{a_1} \ldots e_n^{a_n} = e_1^{b_1} \ldots e_n^{b_n} \longleftrightarrow a = b$$

We show this now.

lemma *lead-monom-sym-mpoly-prod*:
assumes *finite A*
shows $\textit{lead-monom}\ (\prod i = 1..n.\ \textit{sym-mpoly}\ A\ i\ \hat{}\ h\ i :: {}'a\ \textit{mpoly}) =$
$(\sum i = 1..n.\ \textit{of-nat}\ (h\ i) * \textit{lead-monom}\ (\textit{sym-mpoly}\ A\ i :: {}'a\ \textit{mpoly}))$
⟨*proof*⟩

lemma *lead-monom-sym-mpoly-prod-notin*:
assumes *finite A* $k \notin A$
shows $\textit{lookup}\ (\textit{lead-monom}\ (\prod i{=}1..n.\ \textit{sym-mpoly}\ A\ i\ \hat{}\ h\ i :: {}'a\ \textit{mpoly}))\ k = 0$
⟨*proof*⟩

lemma *lead-monom-sym-mpoly-prod-in*:
assumes *finite A* $k < n$
shows $\textit{lookup}\ (\textit{lead-monom}\ (\prod i{=}1..n.\ \textit{sym-mpoly}\ A\ i\ \hat{}\ h\ i :: {}'a\ \textit{mpoly}))\ (xs\ !\ k) =$
$(\sum i{=}k{+}1..n.\ h\ i)$
⟨*proof*⟩

lemma *lead-monom-sym-poly-powerprod-inj*:
assumes $\textit{lead-monom}\ (\prod i.\ \textit{sym-mpoly}\ A\ i\ \hat{}\ \textit{lookup}\ m1\ i :: {}'a\ \textit{mpoly}) =$
$\textit{lead-monom}\ (\prod i.\ \textit{sym-mpoly}\ A\ i\ \hat{}\ \textit{lookup}\ m2\ i :: {}'a\ \textit{mpoly})$
assumes *finite A* $\textit{keys}\ m1 \subseteq \{1..n\}$ $\textit{keys}\ m2 \subseteq \{1..n\}$
shows $m1 = m2$
⟨*proof*⟩

We now show uniqueness by first showing that the zero polynomial has a unique representation. We fix some polynomial p with $p(e_1, \ldots, e_n) = 0$ and then show, by contradiction, that $p = 0$.

We have

$$p(e_1, \ldots, e_n) = \sum c_{a_1,\ldots,a_n} e_1^{a_1} \ldots e_n^{a_n}$$

and due to the injectivity of products of powers of elementary symmetric polynomials, the leading term of that sum is precisely the leading term of the summand with the biggest leading monomial, since summands cannot cancel each other.

However, we also know that $p(e_1, \ldots, e_n) = 0$, so it follows that all summands must have leading term 0, and it is then easy to see that they must all be identically 0.

lemma *sym-mpoly-representation-unique-aux*:
fixes $p :: {}'a\ \textit{mpoly}\ \textit{mpoly}$
assumes *finite A* $\textit{insertion}\ (\textit{sym-mpoly}\ A)\ p = 0$
$\bigwedge m.\ \textit{vars}\ (\textit{coeff}\ p\ m) \cap A = \{\}$ $\textit{vars}\ p \subseteq \{1..n\}$
shows $p = 0$
⟨*proof*⟩

The general uniqueness theorem now follows easily. This essentially shows that the substitution $Y_i \mapsto e_i(X_1, \ldots, X_n)$ is an isomorphism between the ring $R[Y_1, \ldots, Y_n]$ and the ring $R[X_1, \ldots, X_n]^{S_n}$ of symmetric polynomials.

theorem *sym-mpoly-representation-unique*:
fixes *p* :: *'a mpoly mpoly*
assumes *finite A*
insertion (sym-mpoly A) p = insertion (sym-mpoly A) q
$\bigwedge$*m. vars (coeff p m) ∩ A = {}* $\bigwedge$*m. vars (coeff q m) ∩ A = {}*
vars p ⊆ {1..n} vars q ⊆ {1..n}
shows *p = q*
⟨*proof*⟩

theorem *eq-fund-sym-poly-witI*:
fixes *p* :: *'a mpoly* **and** *q* :: *'a mpoly mpoly*
assumes *finite A symmetric-mpoly A p*
insertion (sym-mpoly A) q = p
$\bigwedge$*m. vars (coeff q m) ∩ A = {}*
vars q ⊆ {1..n}
shows *q = fund-sym-poly-wit p*
⟨*proof*⟩

2.14 A recursive characterisation of symmetry

In a similar spirit to the proof of the fundamental theorem, we obtain a nice recursive and executable characterisation of symmetry.

function (*domintros*) *check-symmetric-mpoly* **where**
check-symmetric-mpoly p ⟷
(*vars p ∩ A = {}* ∨
A ⊆ vars p ∧ decr p ∧ check-symmetric-mpoly (p − fund-sym-step-poly p))
⟨*proof*⟩

lemma *check-symmetric-mpoly-dom-aux*:
assumes *finite B vars p ⊆ B A ⊆ B*
shows *check-symmetric-mpoly-dom p*
⟨*proof*⟩

lemma *check-symmetric-mpoly-dom* [*intro*]: *check-symmetric-mpoly-dom p*
⟨*proof*⟩

termination *check-symmetric-mpoly*
⟨*proof*⟩

lemmas [*simp del*] = *check-symmetric-mpoly.simps*

lemma *check-symmetric-mpoly-correct*: *check-symmetric-mpoly p* ⟷ *symmetric-mpoly A p*
⟨*proof*⟩

end

2.15 Symmetric functions of roots of a univariate polynomial

Consider a factored polynomial

$$p(X) = c_n X^n + c_{n-1} X^{n-1} + \ldots + c_1 X + c_0 = (X - x_1) \ldots (X - x_n) \ .$$

where c_n is a unit.

Then any symmetric polynomial expression $q(x_1, \ldots, x_n)$ in the roots x_i can be written as a polynomial expression $q'(c_0, \ldots, c_{n-1})$ in the c_i.

Moreover, if the coefficients of q and the inverse of c_n all lie in some subring, the coefficients of q' do as well.

context
  fixes *C* :: *'b* :: *comm-ring-1 set*
    and *A* :: *nat set*
    and *root* :: *nat* ⇒ *'a* :: *comm-ring-1*
    and *l* :: *'a* ⇒ *'b*
    and *q* :: *'b mpoly*
    and *n* :: *nat*
  defines *n* ≡ *card A*
  assumes *C*: *ring-closed C* ∀ *m. coeff q m* ∈ *C*
  assumes *l*: *ring-homomorphism l*
  assumes *finite*: *finite A*
  assumes *sym*: *symmetric-mpoly A q* **and** *vars*: *vars q* ⊆ *A*
begin

interpretation *ring-closed C* ⟨*proof*⟩
interpretation *ring-homomorphism l* ⟨*proof*⟩

theorem *symmetric-poly-of-roots-conv-poly-of-coeffs*:
  assumes *c*: *cinv* ∗ *l c* = *1 cinv* ∈ *C*
  assumes *p* = *Polynomial.smult c* (∏ *i*∈*A*. [:−*root i*, *1*:])
  obtains *q'* **where** *vars q'* ⊆ {*0*..<*n*}
      and ⋀*m. coeff q' m* ∈ *C*
      and *insertion* (*l* ∘ *poly.coeff p*) *q'* = *insertion* (*l* ∘ *root*) *q*
⟨*proof*⟩

corollary *symmetric-poly-of-roots-conv-poly-of-coeffs-monic*:
  assumes *p* = (∏ *i*∈*A*. [:−*root i*, *1*:])
  obtains *q'* **where** *vars q'* ⊆ {*0*..<*n*}
      and ⋀*m. coeff q' m* ∈ *C*
      and *insertion* (*l* ∘ *poly.coeff p*) *q'* = *insertion* (*l* ∘ *root*) *q*
⟨*proof*⟩

As a corollary, we obtain the following: Let R, S be rings with $R \subseteq S$. Consider a polynomial $p \in R[X]$ whose leading coefficient c is a unit in R and that has a full set of roots $x_1, \ldots, x_n \in S$, i. e. $p(X) = c(X - x_1) \ldots (X - x_n)$. Let $q \in R[X_1, \ldots, X_n]$ be some symmetric polynomial expression in the roots. Then $q(x_1, \ldots, x_n) \in R$.

A typical use case is $R = \mathbb{Q}$ and $S = \mathbb{C}$, i. e. any symmetric polynomial expression with rational coefficients in the roots of a rational polynomial is again rational. Similarly, any symmetric polynomial expression with integer coefficients in the roots of a monic integer polynomial is agan an integer.

This is remarkable, since the roots themselves are usually not rational (possibly not even real). This particular fact is a key ingredient used in the standard proof that π is transcendental.

corollary *symmetric-poly-of-roots-in-subring*:
  assumes $cinv * l\ c = 1\ cinv \in C$
  assumes $p = Polynomial.smult\ c\ (\prod i{\in}A.\ [:-root\ i,\ 1:])$
  assumes $\forall i.\ l\ (poly.coeff\ p\ i) \in C$
  shows $insertion\ (\lambda x.\ l\ (root\ x))\ q \in C$
$\langle proof \rangle$

corollary *symmetric-poly-of-roots-in-subring-monic*:
  assumes $p = (\prod i{\in}A.\ [:-root\ i,\ 1:])$
  assumes $\forall i.\ l\ (poly.coeff\ p\ i) \in C$
  shows $insertion\ (\lambda x.\ l\ (root\ x))\ q \in C$
$\langle proof \rangle$

end

end

3 Executable Operations for Symmetric Polynomials

theory *Symmetric-Polynomials-Code*
  imports *Symmetric-Polynomials Polynomials.MPoly-Type-Class-FMap*
begin

Lastly, we shall provide some code equations to get executable code for operations related to symmetric polynomials, including, most notably, the fundamental theorem of symmetric polynomials and the recursive symmetry check.

lemma *Ball-subset-right*:
  assumes $T \subseteq S\ \forall x{\in}S-T.\ P\ x$
  shows $(\forall x{\in}S.\ P\ x) = (\forall x{\in}T.\ P\ x)$
  $\langle proof \rangle$

lemma *compute-less-pp*[*code*]:
  $xs < (ys :: 'a :: linorder \Rightarrow_0 'b :: \{zero, linorder\}) \longleftrightarrow$
    $(\exists i{\in}keys\ xs \cup keys\ ys.\ lookup\ xs\ i < lookup\ ys\ i\ \wedge$
    $(\forall j{\in}keys\ xs \cup keys\ ys.\ j < i \longrightarrow lookup\ xs\ j = lookup\ ys\ j))$
$\langle proof \rangle$

lemma *compute-le-pp*[*code*]:
$xs \leq ys \longleftrightarrow xs = ys \lor xs < (ys :: \text{-} \Rightarrow_0 \text{-})$
⟨*proof*⟩

lemma *vars-code* [*code*]:
vars (*MPoly p*) = ($\bigcup$*m*∈*keys p*. *keys m*)
⟨*proof*⟩

lemma *mpoly-coeff-code* [*code*]: *coeff* (*MPoly p*) = *lookup p*
⟨*proof*⟩

lemma *sym-mpoly-code* [*code*]:
sym-mpoly (*set xs*) *k* = ($\sum$*X*∈*Set.filter* (λ*X*. *card X* = *k*) (*Pow* (*set xs*)). *monom* (*monom-of-set X*) *1*)
⟨*proof*⟩

lemma *monom-of-set-code* [*code*]:
monom-of-set (*set xs*) = *Pm-fmap* (*fmap-of-list* (*map* (λ*x*. (*x*, *1*)) *xs*))
(**is** *?lhs* = *?rhs*)
⟨*proof*⟩

lemma *restrictpm-code* [*code*]:
restrictpm A (*Pm-fmap m*) = *Pm-fmap* (*fmrestrict-set A m*)
⟨*proof*⟩

lemmas [*code*] = *check-symmetric-mpoly-correct* [*symmetric*]

notepad
begin
⟨*proof*⟩
end

end

References

[1] B. Blum-Smith and S. Coskey. The fundamental theorem on symmetric polynomials: History's first whiff of Galois theory. 48, 01 2013.