

Symmetric Polynomials

Manuel Eberl

March 8, 2026

Abstract

A symmetric polynomial is a polynomial in variables X_1, \dots, 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, \dots, 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, \dots, x_n , namely $c_k = (-1)^{n-k} e_{n-k}(x_1, \dots, x_n)$.
- 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	3
1.1	Auxiliary material	3
1.2	Main proofs	4
2	Symmetric Polynomials	4
2.1	Auxiliary facts	5
2.2	Subrings and ring homomorphisms	5
2.3	Various facts about multivariate polynomials	7
2.4	Restricting a monomial to a subset of variables	12
2.5	Mapping over a polynomial	13
2.6	The leading monomial and leading coefficient	14
2.7	Turning a set of variables into a monomial	18
2.8	Permuting the variables of a polynomial	18
2.9	Symmetric polynomials	23
2.10	The elementary symmetric polynomials	25
2.11	Induction on the leading monomial	27
2.12	The fundamental theorem of symmetric polynomials	27
2.13	Uniqueness	30
2.14	A recursive characterisation of symmetry	32
2.15	Symmetric functions of roots of a univariate polynomial	33
3	Executable Operations for Symmetric Polynomials	34

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*: $X \subseteq f^{-1} A$
 shows $\text{card } (f^{-1} X \cap A) = \text{card } X$
 $\langle \text{proof} \rangle$

lemma *bij-betw-image-fixed-card-subset*:
 assumes *inj-on f A*
 shows $\text{bij-betw } (\lambda X. f^{-1} X) \{X. X \subseteq A \wedge \text{card } X = k\} \{X. X \subseteq f^{-1} A \wedge \text{card } X = k\}$
 $\langle \text{proof} \rangle$

lemma *image-image-fixed-card-subset*:
 assumes *inj-on f A*
 shows $(\lambda X. f^{-1} X)^{-1} \{X. X \subseteq A \wedge \text{card } X = k\} = \{X. X \subseteq f^{-1} A \wedge \text{card } X = k\}$
 $\langle \text{proof} \rangle$

lemma *prod-uminus*: $(\prod_{x \in A}. -f\ x :: 'a :: \text{comm-ring-1}) = (-1)^{\wedge \text{card } A} * (\prod_{x \in A}. f\ x)$
 $\langle \text{proof} \rangle$

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

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

corollary *prod-diff1*:
 fixes $f1\ f2 :: 'a \Rightarrow 'c :: \text{comm-ring-1}$
 assumes *finite*: *finite A*
 shows $(\prod_{x \in A}. f1\ x - f2\ x) = (\sum_{X \in \text{Pow } A}. (-1)^{\wedge \text{card } X} * (\prod_{x \in X}. f2\ x) * (\prod_{x \in A-X}. f1\ x))$
 $\langle \text{proof} \rangle$

corollary *prod-diff2*:

fixes $f1\ f2 :: 'a \Rightarrow 'c :: comm-ring-1$
assumes $finite: finite\ A$
shows $(\prod_{x \in A}. f1\ x - f2\ x) = (\sum_{X \in Pow\ A}. (-1)^\wedge(card\ A - card\ X) * (\prod_{x \in X}. f1\ x) * (\prod_{x \in A-X}. f2\ x))$
<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 $f :: 'a \Rightarrow 'b :: comm-ring-1$ **assumes** $fin: finite\ A$
shows $(\prod_{x \in A}. [-f\ x, 1:]) = (\sum_{X \in Pow\ A}. monom\ ((-1)^\wedge card\ X * (\prod_{x \in X}. f\ x))\ (card\ (A - X)))$
<proof>

Comparing coefficients yields Vieta's formula:

theorem *coeff-poly-from-roots*:

fixes $f :: 'a \Rightarrow 'b :: comm-ring-1$
assumes $fin: finite\ A$ **and** $k: k \leq card\ A$
shows $coeff\ (\prod_{x \in A}. [-f\ x, 1:])\ k = (-1)^\wedge(card\ A - k) * (\sum_{X \mid X \subseteq A \wedge card\ X = card\ A - k}. (\prod_{x \in X}. f\ x))$
<proof>

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

corollary *coeff-poly-from-roots'*:

fixes $f :: 'a \Rightarrow 'b :: comm-ring-1$
assumes $fin: finite\ A$ **and** $inj: inj-on\ f\ A$ **and** $k: k \leq card\ A$
shows $coeff\ (\prod_{x \in A}. [-f\ x, 1:])\ k = (-1)^\wedge(card\ A - k) * (\sum_{X \mid X \subseteq f\ 'A \wedge card\ X = card\ A - k}. \prod X)$
<proof>

end

2 Symmetric Polynomials

theory *Symmetric-Polynomials*

imports

Vieta

Polynomials.More-MPoly-Type

HOL-Combinatorics.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 ⊆ A ∧ 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 ⊆ A ∧ finite X ∧ card X = k}*
 ⟨*proof*⟩

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

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

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

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

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

2.2 Subrings and ring homomorphisms

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

lemma *minus-closed* [*simp*]: $x \in A \implies y \in A \implies x - y \in A$
<proof>

lemma *sum-closed* [*intro*]: $(\bigwedge x. x \in X \implies f x \in A) \implies \text{sum } f X \in A$
<proof>

lemma *power-closed* [*intro*]: $x \in A \implies x \wedge n \in A$
<proof>

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

lemma *prod-closed* [*intro*]: $(\bigwedge x. x \in X \implies f x \in A) \implies \text{prod } f X \in A$
<proof>

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

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

lemma *of-nat-closed* [*simp*, *intro*]: *of-nat* $n \in A$
<proof>

lemma *of-int-closed* [*simp*, *intro*]: *of-int* $n \in A$
<proof>

end

locale *ring-homomorphism* =
 fixes $f :: 'a :: \text{comm-ring-1} \Rightarrow 'b :: \text{comm-ring-1}$
 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*]: $f (x \wedge n) = f x \wedge n$
<proof>

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

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

<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*]: $(m :: \text{nat} \Rightarrow_0 \text{nat}) \geq 0$
<proof>

lemma *poly-mapping-nat-le-0* [*simp*]: $(m :: \text{nat} \Rightarrow_0 \text{nat}) \leq 0 \iff m = 0$
<proof>

lemma *of-nat-diff-poly-mapping-nat*:

assumes $m \geq n$

shows $\text{of-nat } (m - n) = (\text{of-nat } m - \text{of-nat } n :: 'a :: \text{monoid-add} \Rightarrow_0 \text{nat})$

<proof>

lemma *mpoly-coeff-transfer* [*transfer-rule*]:

$\text{rel-fun } \text{cr-mpoly} (=) \text{poly-mapping.lookup MPoly-Type.coeff}$

<proof>

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

lemma *mapping-of-eq-0-iff* [*simp*]: $\text{mapping-of } p = 0 \iff p = 0$
<proof>

lemma *Sum-any-mapping-of*: $\text{Sum-any } (\lambda x. \text{mapping-of } (f x)) = \text{mapping-of } (\text{Sum-any } f)$

<proof>

lemma *Sum-any-parametric-cr-mpoly* [*transfer-rule*]:

$(\text{rel-fun } (\text{rel-fun } (=) \text{cr-mpoly}) \text{cr-mpoly}) \text{Sum-any Sum-any}$

<proof>

lemma *lookup-mult-of-nat* [*simp*]: $\text{lookup } (\text{of-nat } n * m) k = n * \text{lookup } m k$
<proof>

lemma *mpoly-eqI*:

assumes $\bigwedge \text{mon}. \text{MPoly-Type.coeff } p \text{ mon} = \text{MPoly-Type.coeff } q \text{ mon}$

shows $p = q$
<proof>

lemma *coeff-mpoly-times*:

$MPoly\text{-Type.coeff } (p * q) \text{ mon} = \text{prod-fun } (MPoly\text{-Type.coeff } p) (MPoly\text{-Type.coeff } q) \text{ mon}$
<proof>

lemma (*in ring-closed*) *coeff-mult-closed* [intro]:

$(\bigwedge x. \text{coeff } p \ x \in A) \implies (\bigwedge x. \text{coeff } q \ x \in A) \implies \text{coeff } (p * q) \ x \in A$
<proof>

lemma *coeff-notin-vars*:

assumes $\neg(\text{keys } m \subseteq \text{vars } p)$
shows $\text{coeff } p \ m = 0$
<proof>

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

lemma *insertion-altdef*:

$\text{insertion } f \ p = \text{Sum-any } (\lambda m. \text{coeff } p \ m * \text{Prod-any } (\lambda i. f \ i \ ^\wedge \text{lookup } m \ i))$
<proof>

lemma *mpoly-coeff-uminus* [simp]: $\text{coeff } (-p) \ m = -\text{coeff } p \ m$
<proof>

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

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

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

lemma *Sum-any-diff*:

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

lemma *insertion-diff*:

$\text{insertion } f \ (p - q :: 'a :: \text{comm-ring-1 } \text{mpoly}) = \text{insertion } f \ p - \text{insertion } f \ q$
<proof>

lemma *insertion-power*: $\text{insertion } f (p \wedge n) = \text{insertion } f p \wedge n$
<proof>

lemma *insertion-sum*: $\text{insertion } f (\text{sum } g A) = (\sum x \in A. \text{insertion } f (g x))$
<proof>

lemma *insertion-prod*: $\text{insertion } f (\text{prod } g A) = (\prod x \in A. \text{insertion } f (g x))$
<proof>

lemma *coeff-Var*: $\text{coeff } (\text{Var } i) m = (1 \text{ when } m = \text{Poly-Mapping.single } i \ 1)$
<proof>

lemma *vars-Var*: $\text{vars } (\text{Var } i :: 'a :: \{\text{one,zero}\} \text{ mpoly}) = (\text{if } (0::'a) = 1 \text{ then } \{\}$
*else } i}
*<proof>**

lemma *insertion-Var [simp]*: $\text{insertion } f (\text{Var } i) = f i$
<proof>

lemma *insertion-Sum-any*:
assumes *finite* $\{x. g x \neq 0\}$
shows $\text{insertion } f (\text{Sum-any } g) = \text{Sum-any } (\lambda x. \text{insertion } f (g x))$
<proof>

lemma *keys-diff-subset*:
 $\text{keys } (f - g) \subseteq \text{keys } f \cup \text{keys } g$
<proof>

lemma *keys-empty-iff [simp]*: $\text{keys } p = \{\} \longleftrightarrow p = 0$
<proof>

lemma *mpoly-coeff-0 [simp]*: $\text{MPoly-Type.coeff } 0 m = 0$
<proof>

lemma *lookup-1*: $\text{lookup } 1 m = (\text{if } m = 0 \text{ then } 1 \text{ else } 0)$
<proof>

lemma *mpoly-coeff-1*: $\text{MPoly-Type.coeff } 1 m = (\text{if } m = 0 \text{ then } 1 \text{ else } 0)$
<proof>

lemma *lookup-Const₀*: $\text{lookup } (\text{Const}_0 c) m = (\text{if } m = 0 \text{ then } c \text{ else } 0)$
<proof>

lemma *mpoly-coeff-Const*: $\text{MPoly-Type.coeff } (\text{Const } c) m = (\text{if } m = 0 \text{ then } c \text{ else } 0)$
<proof>

lemma *coeff-smult [simp]*: $\text{coeff } (\text{smult } c p) m = (c :: 'a :: \text{mult-zero}) * \text{coeff } p m$
<proof>

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

lemma *keys-uminus* [*simp*]: $\text{keys } (-m) = \text{keys } m$
<proof>

lemma *vars-uminus* [*simp*]: $\text{vars } (-p) = \text{vars } p$
<proof>

lemma *vars-smult*: $\text{vars } (\text{smult } c p) \subseteq \text{vars } p$
<proof>

lemma *vars-0* [*simp*]: $\text{vars } 0 = \{\}$
<proof>

lemma *vars-1* [*simp*]: $\text{vars } 1 = \{\}$
<proof>

lemma *vars-sum*: $\text{vars } (\text{sum } f A) \subseteq (\bigcup x \in A. \text{vars } (f x))$
<proof>

lemma *vars-prod*: $\text{vars } (\text{prod } f A) \subseteq (\bigcup x \in A. \text{vars } (f x))$
<proof>

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

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

lemma *vars-power*: $\text{vars } (p \wedge n) \subseteq \text{vars } p$
<proof>

lemma *vars-diff*: $\text{vars } (p1 - p2) \subseteq \text{vars } p1 \cup \text{vars } p2$
<proof>

lemma *insertion-smult* [*simp*]: $\text{insertion } f (\text{smult } c p) = c * \text{insertion } f p$
<proof>

lemma *coeff-add* [*simp*]: $\text{coeff } (p + q) m = \text{coeff } p m + \text{coeff } q m$
<proof>

lemma *coeff-diff* [*simp*]: $\text{coeff } (p - q) m = \text{coeff } p m - \text{coeff } q m$
<proof>

lemma *insertion-monom* [*simp*]:
 $\text{insertion } f (\text{monom } m c) = c * \text{Prod-any } (\lambda x. f x \wedge \text{lookup } m x)$

<proof>

lemma *insertion-aux-Const₀* [simp]: *insertion-aux f (Const₀ 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-Const₀*: *keys (Const₀ 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: $\bigwedge x y. f (x + y) = f x + f y$*
 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 ^ n) 0 = coeff p 0 ^ n*
<proof>

lemma *prod-fun-max*:

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

lemma *prod-fun-gt-max-eq-zero*:

fixes $f\ g :: 'a::\{\text{linorder}, \text{ordered-cancel-comm-monoid-add}\} \Rightarrow 'b::\text{semiring-0}$
assumes $m > a + b$
assumes $\text{zero}: \bigwedge m. m > a \implies f\ m = 0 \ \bigwedge m. m > b \implies g\ m = 0$
assumes $\text{fin}: \text{finite } \{m. f\ m \neq 0\} \ \text{finite } \{m. g\ m \neq 0\}$
shows $\text{prod-fun } f\ g\ m = 0$
 $\langle \text{proof} \rangle$

2.4 Restricting a monomial to a subset of variables

lift-definition *restrictpm* :: $'a \text{ set} \Rightarrow ('a \Rightarrow_0 'b :: \text{zero}) \Rightarrow ('a \Rightarrow_0 'b) \text{ is}$

$\lambda A\ f\ x. \text{if } x \in A \text{ then } f\ x \text{ else } 0$
 $\langle \text{proof} \rangle$

lemma *lookup-restrictpm*: $\text{lookup } (\text{restrictpm } A\ m) x = (\text{if } x \in A \text{ then } \text{lookup } m\ x \text{ else } 0)$

$\langle \text{proof} \rangle$

lemma *lookup-restrictpm-in* [simp]: $x \in A \implies \text{lookup } (\text{restrictpm } A\ m) x = \text{lookup } m\ x$

and *lookup-restrict-pm-not-in* [simp]: $x \notin A \implies \text{lookup } (\text{restrictpm } A\ m) x = 0$
 $\langle \text{proof} \rangle$

lemma *keys-restrictpm* [simp]: $\text{keys } (\text{restrictpm } A\ m) = \text{keys } m \cap A$

$\langle \text{proof} \rangle$

lemma *restrictpm-add*: $\text{restrictpm } X\ (m1 + m2) = \text{restrictpm } X\ m1 + \text{restrictpm } X\ m2$

$\langle \text{proof} \rangle$

lemma *restrictpm-id* [simp]: $\text{keys } m \subseteq X \implies \text{restrictpm } X\ m = m$

$\langle \text{proof} \rangle$

lemma *restrictpm-orthogonal* [simp]: $\text{keys } m \subseteq -X \implies \text{restrictpm } X\ m = 0$

$\langle \text{proof} \rangle$

lemma *restrictpm-add-disjoint*:

$X \cap Y = \{\} \implies \text{restrictpm } X\ m + \text{restrictpm } Y\ m = \text{restrictpm } (X \cup Y)\ m$

$\langle \text{proof} \rangle$

lemma *restrictpm-add-complements*:

$\text{restrictpm } X\ m + \text{restrictpm } (-X)\ m = m \ \text{restrictpm } (-X)\ m + \text{restrictpm } X\ m$

$m = m$
 ⟨proof⟩

2.5 Mapping over a polynomial

lift-definition $map\text{-}mpoly :: ('a :: zero \Rightarrow 'b :: zero) \Rightarrow 'a\ mpoly \Rightarrow 'b\ mpoly$ is
 $\lambda(f :: 'a \Rightarrow 'b) (p :: (nat \Rightarrow_0 nat) \Rightarrow_0 'a). Poly\text{-}Mapping.map\ f\ p$ ⟨proof⟩

lift-definition $mapm\text{-}mpoly :: ((nat \Rightarrow_0 nat) \Rightarrow 'a :: zero \Rightarrow 'b :: zero) \Rightarrow 'a\ mpoly \Rightarrow 'b\ mpoly$ is
 $\lambda(f :: (nat \Rightarrow_0 nat) \Rightarrow 'a \Rightarrow 'b) (p :: (nat \Rightarrow_0 nat) \Rightarrow_0 'a).$
Poly-Mapping.mapp $f\ p$ ⟨proof⟩

lemma $poly\text{-}mapping\text{-}map\text{-}conv\text{-}mapp: Poly\text{-}Mapping.map\ f = Poly\text{-}Mapping.mapp$
 $(\lambda\cdot. f)$
 ⟨proof⟩

lemma $map\text{-}mpoly\text{-}conv\text{-}mapm\text{-}mpoly: map\text{-}mpoly\ f = mapm\text{-}mpoly\ (\lambda\cdot. f)$
 ⟨proof⟩

lemma $map\text{-}mpoly\text{-}comp: f\ 0 = 0 \implies map\text{-}mpoly\ f\ (map\text{-}mpoly\ g\ p) = map\text{-}mpoly$
 $(f\ \circ\ g)\ p$
 ⟨proof⟩

lemma $mapp\text{-}mapp:$
 $(\bigwedge x. f\ x\ 0 = 0) \implies Poly\text{-}Mapping.mapp\ f\ (Poly\text{-}Mapping.mapp\ g\ m) =$
 $Poly\text{-}Mapping.mapp\ (\lambda x\ y. f\ x\ (g\ x\ y))\ m$
 ⟨proof⟩

lemma $mapm\text{-}mpoly\text{-}comp:$
 $(\bigwedge x. f\ x\ 0 = 0) \implies mapm\text{-}mpoly\ f\ (mapm\text{-}mpoly\ g\ p) = mapm\text{-}mpoly\ (\lambda m\ c. f$
 $m\ (g\ m\ c))\ p$
 ⟨proof⟩

lemma $coeff\text{-}map\text{-}mpoly:$
 $coeff\ (map\text{-}mpoly\ f\ p)\ m = (if\ coeff\ p\ m = 0\ then\ 0\ else\ f\ (coeff\ p\ m))$
 ⟨proof⟩

lemma $coeff\text{-}map\text{-}mpoly'\ [simp]: f\ 0 = 0 \implies coeff\ (map\text{-}mpoly\ f\ p)\ m = f\ (coeff$
 $p\ m)$
 ⟨proof⟩

lemma $coeff\text{-}mapm\text{-}mpoly: coeff\ (mapm\text{-}mpoly\ f\ p)\ m = (if\ coeff\ p\ m = 0\ then\ 0$
 $else\ f\ m\ (coeff\ p\ m))$
 ⟨proof⟩

lemma $coeff\text{-}mapm\text{-}mpoly'\ [simp]: (\bigwedge m. f\ m\ 0 = 0) \implies coeff\ (mapm\text{-}mpoly\ f\ p)$
 $m = f\ m\ (coeff\ p\ m)$
 ⟨proof⟩

lemma *vars-map-mpoly-subset*: $\text{vars } (\text{map-mpoly } f \ p) \subseteq \text{vars } p$
 ⟨proof⟩

lemma *coeff-sum [simp]*: $\text{coeff } (\text{sum } f \ A) \ m = (\sum_{x \in A} \text{coeff } (f \ x) \ m)$
 ⟨proof⟩

lemma *coeff-Sum-any*: $\text{finite } \{x. f \ x \neq 0\} \implies \text{coeff } (\text{Sum-any } f) \ m = \text{Sum-any } (\lambda x. \text{coeff } (f \ x) \ m)$
 ⟨proof⟩

lemma *Sum-any-zeroI*: $(\bigwedge x. f \ x = 0) \implies \text{Sum-any } f = 0$
 ⟨proof⟩

lemma *insertion-Prod-any*:
 $\text{finite } \{x. g \ x \neq 1\} \implies \text{insertion } f \ (\text{Prod-any } g) = \text{Prod-any } (\lambda x. \text{insertion } f \ (g \ x))$
 ⟨proof⟩

lemma *insertion-insertion*:
 $\text{insertion } g \ (\text{insertion } k \ p) = \text{insertion } (\lambda x. \text{insertion } g \ (k \ x)) \ (\text{map-mpoly } (\text{insertion } g) \ p)$ (is ?lhs = ?rhs)
 ⟨proof⟩

lemma *insertion-substitute-linear*:
 $\text{insertion } (\lambda i. c \ i * f \ i) \ p = \text{insertion } f \ (\text{mapm-mpoly } (\lambda m \ d. \text{Prod-any } (\lambda i. c \ i \wedge \text{lookup } m \ i) * d) \ p)$
 ⟨proof⟩

lemma *vars-mapm-mpoly-subset*: $\text{vars } (\text{mapm-mpoly } f \ p) \subseteq \text{vars } p$
 ⟨proof⟩

lemma *map-mpoly-cong*:
assumes $\bigwedge m. f \ (\text{coeff } p \ m) = g \ (\text{coeff } p \ m) \ p = q$
shows $\text{map-mpoly } f \ p = \text{map-mpoly } g \ q$
 ⟨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 $X_1^0 \dots X_n^0$.

lift-definition *lead-monom* :: 'a :: zero mpoly \Rightarrow (nat \Rightarrow_0 nat) is
 $\lambda f :: (\text{nat } \Rightarrow_0 \text{ nat}) \Rightarrow_0 'a. \text{Max } (\text{insert } 0 \ (\text{keys } f))$ ⟨proof⟩

lemma *lead-monom-geI* [intro]:

assumes *coeff p m* $\neq 0$

shows $m \leq \text{lead-monom } p$

<proof>

lemma *coeff-gt-lead-monom-zero* [simp]:

assumes $m > \text{lead-monom } p$

shows *coeff p m* = 0

<proof>

lemma *lead-monom-nonzero-eq*:

assumes $p \neq 0$

shows $\text{lead-monom } p = \text{Max } (\text{keys } (\text{mapping-of } p))$

<proof>

lemma *lead-monom-0* [simp]: $\text{lead-monom } 0 = 0$

<proof>

lemma *lead-monom-1* [simp]: $\text{lead-monom } 1 = 0$

<proof>

lemma *lead-monom-Const* [simp]: $\text{lead-monom } (\text{Const } c) = 0$

<proof>

lemma *lead-monom-uminus* [simp]: $\text{lead-monom } (-p) = \text{lead-monom } p$

<proof>

lemma *keys-mult-const* [simp]:

fixes $c :: 'a :: \{\text{semiring-0}, \text{semiring-no-zero-divisors}\}$

assumes $c \neq 0$

shows $\text{keys } (\text{Poly-Mapping.map } ((*) c) p) = \text{keys } p$

<proof>

lemma *lead-monom-eq-0-iff*: $\text{lead-monom } p = 0 \iff \text{vars } p = \{\}$

<proof>

lemma *lead-monom-monom*: $\text{lead-monom } (\text{monom } m c) = (\text{if } c = 0 \text{ then } 0 \text{ else } m)$

<proof>

lemma *lead-monom-monom'* [simp]: $c \neq 0 \implies \text{lead-monom } (\text{monom } m c) = m$

<proof>

lemma *lead-monom-numeral* [simp]: $\text{lead-monom } (\text{numeral } n) = 0$

<proof>

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

<proof>

lemma *lead-monom-diff*: $\text{lead-monom } (p - q) \leq \max (\text{lead-monom } p) (\text{lead-monom } q)$
<proof>

definition *lead-coeff* **where** $\text{lead-coeff } p = \text{coeff } p (\text{lead-monom } p)$

lemma *vars-empty-iff*: $\text{vars } p = \{\} \longleftrightarrow p = \text{Const } (\text{lead-coeff } p)$
<proof>

lemma *lead-coeff-0* [*simp*]: $\text{lead-coeff } 0 = 0$
<proof>

lemma *lead-coeff-1* [*simp*]: $\text{lead-coeff } 1 = 1$
<proof>

lemma *lead-coeff-Const* [*simp*]: $\text{lead-coeff } (\text{Const } c) = c$
<proof>

lemma *lead-coeff-monom* [*simp*]: $\text{lead-coeff } (\text{monom } p c) = c$
<proof>

lemma *lead-coeff-nonzero* [*simp*]: $p \neq 0 \implies \text{lead-coeff } p \neq 0$
<proof>

lemma

fixes $c :: 'a :: \text{semiring-0}$

assumes $c * \text{lead-coeff } p \neq 0$

shows *lead-monom-smult* [*simp*]: $\text{lead-monom } (\text{smult } c p) = \text{lead-monom } p$

and *lead-coeff-smult* [*simp*]: $\text{lead-coeff } (\text{smult } c p) = c * \text{lead-coeff } p$

<proof>

lemma *lead-coeff-mult-aux*:

$\text{coeff } (p * q) (\text{lead-monom } p + \text{lead-monom } q) = \text{lead-coeff } p * \text{lead-coeff } q$

<proof>

lemma *lead-monom-mult-le*: $\text{lead-monom } (p * q) \leq \text{lead-monom } p + \text{lead-monom } q$

<proof>

lemma *lead-monom-mult*:

assumes $\text{lead-coeff } p * \text{lead-coeff } q \neq 0$

shows $\text{lead-monom } (p * q) = \text{lead-monom } p + \text{lead-monom } q$

<proof>

lemma *lead-coeff-mult*:

assumes $\text{lead-coeff } p * \text{lead-coeff } q \neq 0$

shows $lead-coeff (p * q) = lead-coeff p * lead-coeff q$
 ⟨proof⟩

lemma *keys-lead-monom-subset*: $keys (lead-monom p) \subseteq vars p$
 ⟨proof⟩

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

lemma *lead-monom-sum-le*: $(\bigwedge x. x \in X \implies lead-monom (h x) \leq ub) \implies lead-monom (sum h X) \leq ub$
 ⟨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* $(lead-monom \circ h) X$ **and** *finite* X **and** $X \neq \{\}$ **and** $\bigwedge x. x \in X \implies h x \neq 0$
defines $m \equiv Max ((lead-monom \circ h) ' X)$
shows $lead-monom (\sum x \in X. h x) = m$
 ⟨proof⟩

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

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

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

lemma
fixes $p :: 'a :: semidom mpoly$
assumes $p \neq 0$

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

2.7 Turning a set of variables into a monomial

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

lift-definition *monom-of-set* :: $\text{nat set} \Rightarrow (\text{nat} \Rightarrow_0 \text{nat})$ **is**
 $\lambda X x. \text{if finite } X \wedge x \in X \text{ then } 1 \text{ else } 0$
 ⟨proof⟩

lemma *lookup-monom-of-set*:
 $\text{Poly-Mapping.lookup } (\text{monom-of-set } X) i = (\text{if finite } X \wedge i \in X \text{ then } 1 \text{ else } 0)$
 ⟨proof⟩

lemma *lookup-monom-of-set-1* [simp]:
 $\text{finite } X \Longrightarrow i \in X \Longrightarrow \text{Poly-Mapping.lookup } (\text{monom-of-set } X) i = 1$
and *lookup-monom-of-set-0* [simp]:
 $i \notin X \Longrightarrow \text{Poly-Mapping.lookup } (\text{monom-of-set } X) i = 0$
 ⟨proof⟩

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

lemma *keys-monom-of-set-finite* [simp]: $\text{finite } X \Longrightarrow \text{keys } (\text{monom-of-set } X) = X$
 ⟨proof⟩

lemma *monom-of-set-eq-iff* [simp]: $\text{finite } X \Longrightarrow \text{finite } Y \Longrightarrow \text{monom-of-set } X = \text{monom-of-set } Y \longleftrightarrow X = Y$
 ⟨proof⟩

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

lemma *monom-of-set-eq-zero-iff* [simp]: $\text{monom-of-set } X = 0 \longleftrightarrow \text{infinite } X \vee X = \{\}$
 ⟨proof⟩

lemma *zero-eq-monom-of-set-iff* [simp]: $0 = \text{monom-of-set } X \longleftrightarrow \text{infinite } X \vee X = \{\}$
 ⟨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*:
 $\text{bij } f \Longrightarrow \text{permutep } f (\text{Poly-Mapping.single } a \ b) = \text{Poly-Mapping.single } (\text{inv-into UNIV } f \ a) \ b$
 ⟨*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*:
 $\text{Poly-Mapping.lookup } (\text{permutep } f \ m) \ x = (\text{if } \text{bij } f \ \text{then } \text{Poly-Mapping.lookup } m \ (f \ x) \ \text{else } \text{Poly-Mapping.lookup } m \ x)$
 ⟨*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*:
 $\text{mpoly-map-vars } f \ (\text{map-mpoly } g \ p) = \text{map-mpoly } g \ (\text{mpoly-map-vars } f \ p)$
 ⟨*proof*⟩

lemma *coeff-mpoly-map-vars*:

fixes $f :: \text{nat} \Rightarrow \text{nat}$ **and** $p :: 'a :: \text{zero mpoly}$
assumes $\text{bij } f$
shows $\text{MPoly-Type.coeff } (\text{mpoly-map-vars } f \ p) \ \text{mon} =$
 $\text{MPoly-Type.coeff } p \ (\text{permutep } f \ \text{mon})$
 $\langle \text{proof} \rangle$

lemma *permutep-monom-of-set*:
assumes $\text{bij } f$
shows $\text{permutep } f \ (\text{monom-of-set } A) = \text{monom-of-set } (f \ -' \ A)$
 $\langle \text{proof} \rangle$

lemma *permutep-comp*: $\text{bij } f \Longrightarrow \text{bij } g \Longrightarrow \text{permutep } (f \circ g) = \text{permutep } g \circ$
 $\text{permutep } f$
 $\langle \text{proof} \rangle$

lemma *permutep-comp'*: $\text{bij } f \Longrightarrow \text{bij } g \Longrightarrow \text{permutep } (f \circ g) \ \text{mon} = \text{permutep } g$
 $(\text{permutep } f \ \text{mon})$
 $\langle \text{proof} \rangle$

lemma *mpoly-map-vars-comp*:
 $\text{bij } f \Longrightarrow \text{bij } g \Longrightarrow \text{mpoly-map-vars } f \ (\text{mpoly-map-vars } g \ p) = \text{mpoly-map-vars } (f$
 $\circ g) \ p$
 $\langle \text{proof} \rangle$

lemma *permutep-id [simp]*: $\text{permutep } \text{id} \ \text{mon} = \text{mon}$
 $\langle \text{proof} \rangle$

lemma *permutep-id' [simp]*: $\text{permutep } (\lambda x. x) \ \text{mon} = \text{mon}$
 $\langle \text{proof} \rangle$

lemma *inv-permutep [simp]*:
fixes $f :: 'a \Rightarrow 'a$
assumes $\text{bij } f$
shows $\text{inv-into UNIV } (\text{permutep } f) = \text{permutep } (\text{inv-into UNIV } f)$
 $\langle \text{proof} \rangle$

lemma *mpoly-map-vars-monom*:
 $\text{bij } f \Longrightarrow \text{mpoly-map-vars } f \ (\text{monom } m \ c) = \text{monom } (\text{permutep } (\text{inv-into UNIV}$
 $f) \ m) \ c$
 $\langle \text{proof} \rangle$

lemma *vars-mpoly-map-vars*:
fixes $f :: \text{nat} \Rightarrow \text{nat}$ **and** $p :: 'a :: \text{zero mpoly}$
assumes $\text{bij } f$
shows $\text{vars } (\text{mpoly-map-vars } f \ p) = f \ ' \ \text{vars } p$
 $\langle \text{proof} \rangle$

lemma *permutep-eq-monom-of-set-iff [simp]*:
assumes $\text{bij } f$

shows $\text{permutep } f \text{ mon} = \text{monom-of-set } A \longleftrightarrow \text{mon} = \text{monom-of-set } (f \text{ ` } A)$
<proof>

lemma *permutep-monom-of-set-permutes* [simp]:

assumes π permutes A

shows $\text{permutep } \pi (\text{monom-of-set } A) = \text{monom-of-set } A$

<proof>

lemma *mpoly-map-vars-0* [simp]: $\text{mpoly-map-vars } f \ 0 = 0$

<proof>

lemma *permutep-eq-0-iff* [simp]: $\text{permutep } f \ m = 0 \longleftrightarrow m = 0$

<proof>

lemma *mpoly-map-vars-1* [simp]: $\text{mpoly-map-vars } f \ 1 = 1$

<proof>

lemma *permutep-Const₀* [simp]: $(\bigwedge x. f \ x = 0 \longleftrightarrow x = 0) \implies \text{permutep } f (\text{Const}_0 \ c) = \text{Const}_0 \ c$

<proof>

lemma *permutep-add* [simp]: $\text{permutep } f (m1 + m2) = \text{permutep } f \ m1 + \text{permutep } f \ m2$

<proof>

lemma *permutep-diff* [simp]: $\text{permutep } f (m1 - m2) = \text{permutep } f \ m1 - \text{permutep } f \ m2$

<proof>

lemma *permutep-uminus* [simp]: $\text{permutep } f (-m) = -\text{permutep } f \ m$

<proof>

lemma *permutep-mult* [simp]:

$(\bigwedge x \ y. f (x + y) = f \ x + f \ y) \implies \text{permutep } f (m1 * m2) = \text{permutep } f \ m1 * \text{permutep } f \ m2$

<proof>

lemma *mpoly-map-vars-Const* [simp]: $\text{mpoly-map-vars } f (\text{Const } c) = \text{Const } c$

<proof>

lemma *mpoly-map-vars-add* [simp]: $\text{mpoly-map-vars } f (p + q) = \text{mpoly-map-vars } f \ p + \text{mpoly-map-vars } f \ q$

<proof>

lemma *mpoly-map-vars-diff* [simp]: $\text{mpoly-map-vars } f (p - q) = \text{mpoly-map-vars } f \ p - \text{mpoly-map-vars } f \ q$

<proof>

lemma *mpoly-map-vars-uminus* [simp]: $\text{mpoly-map-vars } f (-p) = -\text{mpoly-map-vars } f \ p$

$f p$
 $\langle \text{proof} \rangle$

lemma *permutep-smult*:

$\text{permutep } (\text{permutep } f) (\text{Poly-Mapping.map } ((*) c) p) =$
 $\text{Poly-Mapping.map } ((*) c) (\text{permutep } (\text{permutep } f) p)$
 $\langle \text{proof} \rangle$

lemma *mpoly-map-vars-smult* [simp]: $\text{mpoly-map-vars } f (\text{smult } c p) = \text{smult } c$
 $(\text{mpoly-map-vars } f p)$
 $\langle \text{proof} \rangle$

lemma *mpoly-map-vars-mult* [simp]: $\text{mpoly-map-vars } f (p * q) = \text{mpoly-map-vars}$
 $f p * \text{mpoly-map-vars } f q$
 $\langle \text{proof} \rangle$

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

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

lemma *mpoly-map-vars-eq-0-iff* [simp]: $\text{mpoly-map-vars } f p = 0 \longleftrightarrow p = 0$
 $\langle \text{proof} \rangle$

lemma *permutep-eq-iff* [simp]: $\text{permutep } f p = \text{permutep } f q \longleftrightarrow p = q$
 $\langle \text{proof} \rangle$

lemma *mpoly-map-vars-Sum-any* [simp]:
 $\text{mpoly-map-vars } f (\text{Sum-any } g) = \text{Sum-any } (\lambda x. \text{mpoly-map-vars } f (g x))$
 $\langle \text{proof} \rangle$

lemma *mpoly-map-vars-power* [simp]: $\text{mpoly-map-vars } f (p \hat{=} n) = \text{mpoly-map-vars}$
 $f p \hat{=} n$
 $\langle \text{proof} \rangle$

lemma *mpoly-map-vars-monom-single* [simp]:

assumes *bij f*

shows $\text{mpoly-map-vars } f (\text{monom } (\text{Poly-Mapping.single } i n) c) =$
 $\text{monom } (\text{Poly-Mapping.single } (f i) n) c$

$\langle \text{proof} \rangle$

lemma *insertion-mpoly-map-vars*:

assumes *bij f*

shows $\text{insertion } g (\text{mpoly-map-vars } f p) = \text{insertion } (g \circ f) p$

$\langle \text{proof} \rangle$

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

lemma *mpoly-map-vars-cong*:
assumes f permutes $(-vars\ p)$ g permutes $(-vars\ q)$ $p = q$
shows $mpoly-map-vars\ f\ p = mpoly-map-vars\ g\ (q :: 'a :: zero\ mpoly)$
 $\langle proof \rangle$

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

lemma *symmetric-mpoly-empty* [*simp, intro*]: *symmetric-mpoly* $\{\}$ p
 $\langle proof \rangle$

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-mpoly* $A\ p$
 $\langle proof \rangle$

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

lemma *symmetric-mpoly-subset*:
assumes *symmetric-mpoly* $A\ p$ $B \subseteq A$
shows *symmetric-mpoly* $B\ p$
 $\langle proof \rangle$

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-mpoly* $A\ p$
shows $vars\ p \cap A = \{\} \vee A \subseteq vars\ p$
 $\langle proof \rangle$

Symmetric polynomials are closed under ring operations.

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

lemma *symmetric-mpoly-diff* [intro]:

symmetric-mpoly A $p \implies$ *symmetric-mpoly* A $q \implies$ *symmetric-mpoly* A $(p - q)$
(proof)

lemma *symmetric-mpoly-uminus* [intro]: *symmetric-mpoly* A $p \implies$ *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 $p \implies$ *symmetric-mpoly* A $(smult\ c\ p)$
(proof)

lemma *symmetric-mpoly-mult* [intro]:

symmetric-mpoly A $p \implies$ *symmetric-mpoly* A $q \implies$ *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 $p \implies$ *symmetric-mpoly* A $(p \wedge n)$
(proof)

lemma *symmetric-mpoly-sum* [intro]:

$(\bigwedge i. i \in B \implies$ *symmetric-mpoly* A $(f\ i)) \implies$ *symmetric-mpoly* A $(sum\ f\ B)$
(proof)

lemma *symmetric-mpoly-prod* [intro]:

$(\bigwedge i. i \in B \implies$ *symmetric-mpoly* A $(f\ i)) \implies$ *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 $\bigwedge x\ \pi. x \in X \implies \pi$ permutes $A \implies$ *mpoly-map-vars* π $(f\ x) = f\ (g\ x)$

shows *symmetric-mpoly* A $(\sum_{x \in X}. f\ x)$

(proof)

lemma *symmetric-mpoly-symmetric-prod*:

assumes g permutes X

assumes $\bigwedge x\ \pi. x \in X \implies \pi$ permutes $A \implies$ *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 $m \equiv \text{lead-monom } p$

assumes $i \in A \ j \in A \ i \leq j$

shows $\text{lookup } m \ i \geq \text{lookup } m \ j$

⟨*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* :: $\text{nat set} \Rightarrow \text{nat} \Rightarrow (\text{nat} \Rightarrow_0 \text{nat}) \Rightarrow_0 'a :: \{\text{zero-neq-one}\}$
is

$\lambda X k \text{ mon. if finite } X \wedge (\exists Y. Y \subseteq X \wedge \text{card } Y = k \wedge \text{mon} = \text{monom-of-set } Y)$
 then 1 else 0

⟨*proof*⟩

lemma *lookup-sym-mpoly-aux*:

Poly-Mapping.lookup (*sym-mpoly-aux* $X k$) *mon* =

(if finite $X \wedge (\exists Y. Y \subseteq X \wedge \text{card } Y = k \wedge \text{mon} = \text{monom-of-set } Y)$ then 1
 else 0)

⟨*proof*⟩

lemma *lookup-sym-mpoly-aux-monom-of-set* [*simp*]:

assumes *finite* $X \ Y \subseteq X \ \text{card } Y = k$

shows *Poly-Mapping.lookup* (*sym-mpoly-aux* $X k$) (*monom-of-set* Y) = 1

⟨*proof*⟩

lemma *keys-sym-mpoly-aux*: $m \in \text{keys } (\text{sym-mpoly-aux } A k) \implies \text{keys } m \subseteq A$

⟨*proof*⟩

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

sym-mpoly-aux ⟨*proof*⟩

lemma *vars-sym-mpoly-subset*: $\text{vars } (\text{sym-mpoly } A k) \subseteq A$

⟨*proof*⟩

lemma *coeff-sym-mpoly*:

MPoly-Type.coeff (*sym-mpoly* $X k$) *mon* =

(if finite $X \wedge (\exists Y. Y \subseteq X \wedge \text{card } Y = k \wedge \text{mon} = \text{monom-of-set } Y)$ then 1
 else 0)

<proof>

lemma *sym-mpoly-infinite*: $\neg \text{finite } A \implies \text{sym-mpoly } A \ k = 0$

<proof>

lemma *sym-mpoly-altdef*: $\text{sym-mpoly } A \ k = (\sum X \mid X \subseteq A \wedge \text{card } X = k. \text{monom } (\text{monom-of-set } X) \ 1)$

<proof>

lemma *coeff-sym-mpoly-monom-of-set* [simp]:

assumes *finite* $X \ Y \subseteq X \ \text{card } Y = k$

shows $\text{MPoly-Type.coeff } (\text{sym-mpoly } X \ k) \ (\text{monom-of-set } Y) = 1$

<proof>

lemma *coeff-sym-mpoly-0*: $\text{coeff } (\text{sym-mpoly } X \ k) \ 0 = (\text{if } \text{finite } X \wedge k = 0 \text{ then } 1 \text{ else } 0)$

<proof>

lemma *symmetric-sym-mpoly* [intro]:

assumes $A \subseteq B$

shows $\text{symmetric-mpoly } A \ (\text{sym-mpoly } B \ k :: 'a :: \text{zero-neq-one mpoly})$

<proof>

lemma *insertion-sym-mpoly*:

assumes *finite* X

shows $\text{insertion } f \ (\text{sym-mpoly } X \ k) = (\sum Y \mid Y \subseteq X \wedge \text{card } Y = k. \text{prod } f \ Y)$

<proof>

lemma *sym-mpoly-nz* [simp]:

assumes *finite* $A \ k \leq \text{card } A$

shows $\text{sym-mpoly } A \ k \neq (0 :: 'a :: \text{zero-neq-one mpoly})$

<proof>

lemma *coeff-sym-mpoly-0-or-1*: $\text{coeff } (\text{sym-mpoly } A \ k) \ m \in \{0, 1\}$

<proof>

lemma *lead-coeff-sym-mpoly* [simp]:

assumes *finite* $A \ k \leq \text{card } A$

shows $\text{lead-coeff } (\text{sym-mpoly } A \ k) = 1$

<proof>

lemma *lead-monom-sym-mpoly*:

assumes *sorted* $xs \ \text{distinct } xs \ k \leq \text{length } xs$

shows $\text{lead-monom } (\text{sym-mpoly } (\text{set } xs) \ k :: 'a :: \text{zero-neq-one mpoly}) = \text{monom-of-set } (\text{set } (\text{take } k \ xs)) \ (\text{is } \text{lead-monom } ?p = -)$

<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*

$$\text{monom-less-on } A = \{(m1, m2). m1 < m2 \wedge \text{keys } m1 \subseteq A \wedge \text{keys } m2 \subseteq A\}$$

lemma *wf-monom-less-on:*

assumes *finite A*

shows *wf (monom-less-on A :: ((nat \Rightarrow_0 'b :: {zero, wellorder}) \times -) set)*

<proof>

lemma *lead-monom-induct [consumes 2, case-names less]:*

fixes *p :: 'a :: zero mpoly*

assumes *fin: finite A and vars: vars p \subseteq A*

assumes *IH: $\bigwedge p. \text{vars } p \subseteq A \implies$*

$$(\bigwedge p'. \text{vars } p' \subseteq A \implies \text{lead-monom } p' < \text{lead-monom } p \implies P p')$$

$\implies P p$

shows *P p*

<proof>

lemma *lead-monom-induct' [case-names less]:*

fixes *p :: 'a :: zero mpoly*

assumes *IH: $\bigwedge p. (\bigwedge p'. \text{vars } p' \subseteq \text{vars } p \implies \text{lead-monom } p' < \text{lead-monom } p \implies P p') \implies P p$*

shows *P p*

<proof>

2.12 The fundamental theorem of symmetric polynomials

lemma *lead-coeff-sym-mpoly-powerprod:*

assumes *finite A $\bigwedge x. x \in X \implies f x \in \{1..card A\}$*

shows *lead-coeff ($\prod_{x \in X}. \text{sym-mpoly } A (f (x::'a)) \wedge g x) = 1$*

<proof>

context

fixes *A :: nat set and xs n f and decr :: 'a :: comm-ring-1 mpoly \Rightarrow bool*

defines *xs \equiv sorted-list-of-set A*

defines *n \equiv card A*

defines *f \equiv ($\lambda i. \text{if } i < n \text{ then } xs ! i \text{ else } 0$)*

defines *decr \equiv ($\lambda p. \forall i \in A. \forall j \in A. i \leq j \longrightarrow$*

$$\text{lookup (lead-monom } p) i \geq \text{lookup (lead-monom } p) j$$

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 \Rightarrow 'a mpoly **where**
fund-sym-step-coeff $p = \text{monom} (\text{restrictpm} (-A) (\text{lead-monom } p)) (\text{lead-coeff } p)$

definition *fund-sym-step-monom* :: 'a mpoly \Rightarrow (nat \Rightarrow_0 nat) **where**

fund-sym-step-monom $p =$ (
 let $g = (\lambda i. \text{if } i < n \text{ then lookup } (\text{lead-monom } p) (f \ i) \text{ else } 0)$
 in $(\sum_{i < n}. \text{Poly-Mapping.single } (Suc \ i) (g \ i - g \ (Suc \ i)))$)

definition *fund-sym-step-poly* :: 'a mpoly \Rightarrow 'a mpoly **where**

fund-sym-step-poly $p =$ (
 let $g = (\lambda i. \text{if } i < n \text{ then lookup } (\text{lead-monom } p) (f \ i) \text{ else } 0)$
 in $\text{fund-sym-step-coeff } p * (\prod_{i < n}. \text{sym-mpoly } A \ (Suc \ i) \ \wedge (g \ i - g \ (Suc \ i)))$)

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 \Rightarrow 'a mpoly
mpoly where

fund-sym-poly-wit $p =$
 (if $\neg \text{symmetric-mpoly } A \ p \vee \text{lead-monom } p = 0 \vee \text{vars } p \cap A = \{\}$ then *Const*
 p else
 $\text{fund-sym-poly-wit } (p - \text{fund-sym-step-poly } p) +$
 $\text{monom } (\text{fund-sym-step-monom } p) (\text{fund-sym-step-coeff } p)$)
 <proof>

lemma *coeff-fund-sym-step-coeff*: $\text{coeff } (\text{fund-sym-step-coeff } p) \ m \in \{\text{lead-coeff } p, 0\}$

<proof>

lemma *vars-fund-sym-step-coeff*: $\text{vars } (\text{fund-sym-step-coeff } p) \subseteq \text{vars } p - A$

<proof>

lemma *keys-fund-sym-step-monom*: $\text{keys } (\text{fund-sym-step-monom } p) \subseteq \{1..n\}$

<proof>

lemma *coeff-fund-sym-step-poly*:

assumes $C: \forall m. \text{coeff } p \ m \in C$ **and** *ring-closed* C

shows $\text{coeff } (\text{fund-sym-step-poly } p) \ m \in C$

<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 $\text{finite } A \implies p \neq 0 \implies \text{decr } p \implies \text{lead-monom } (\text{fund-sym-step-poly } p) = \text{lead-monom } p$
and $\text{finite } A \implies p \neq 0 \implies \text{decr } p \implies \text{lead-coeff } (\text{fund-sym-step-poly } p) = \text{lead-coeff } p$
and $\text{finite } A \implies p \neq 0 \implies \text{decr } p \implies \text{fund-sym-step-poly } p = \text{fund-sym-step-coeff } p * (\prod x. \text{sym-mpoly } A \ x \ \hat{\text{lookup}} (\text{fund-sym-step-monom } p) \ x)$
and $\text{vars } (\text{fund-sym-step-poly } p) \subseteq \text{vars } p \cup A$
and $\text{symmetric-mpoly } A \ (\text{fund-sym-step-poly } p)$
 $\langle \text{proof} \rangle$

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 $\text{finite } A$ **and** $\text{lead-monom } p \neq 0$ **and** $\text{decr } p$
shows $\text{lead-monom } (p - \text{fund-sym-step-poly } p) < \text{lead-monom } p$
 $\langle \text{proof} \rangle$

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

lemma *fund-sym-poly-wit-dom-aux*:

assumes $\text{finite } B$ $\text{vars } p \subseteq B$ $A \subseteq B$
shows $\text{fund-sym-poly-wit-dom } p$
 $\langle \text{proof} \rangle$

lemma *fund-sym-poly-wit-dom* [intro]: $\text{fund-sym-poly-wit-dom } p$
 $\langle \text{proof} \rangle$

termination *fund-sym-poly-wit*
 $\langle \text{proof} \rangle$

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

1. If the original polynomial was in $R[X_1, \dots, X_n, \dots, X_m]$ where the X_1 to X_n are the symmetric variables, then the witness is a polynomial in $R[X_{n+1}, \dots, X_m][Y_1, \dots, Y_n]$. 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 $e_i(X_1, \dots, X_n)$ 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 $\forall m. \text{coeff } p \ m \in C$ *ring-closed C*

shows $\forall m \ m'. \text{coeff } (\text{coeff } (\text{fund-sym-poly-wit } p) \ m) \ m' \in C$

<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, \dots, e_n are the elementary symmetric polynomials, $a = (a_1, \dots, a_n)$ and $b = (b_1, \dots, b_n)$ are vectors of natural numbers, then:

$$e_1^{a_1} \dots e_n^{a_n} = e_1^{b_1} \dots e_n^{b_n} \iff a = b$$

We show this now.

lemma *lead-monom-sym-mpoly-prod:*

assumes *finite A*

shows $\text{lead-monom} (\prod_{i=1..n} \text{sym-mpoly } A \ i \ \wedge \ h \ i :: 'a \ \text{mpoly}) =$
 $(\sum_{i=1..n} \text{of-nat } (h \ i) * \text{lead-monom} (\text{sym-mpoly } A \ i :: 'a \ \text{mpoly}))$
 $\langle \text{proof} \rangle$

lemma *lead-monom-sym-mpoly-prod-notin:*

assumes $\text{finite } A \ k \notin A$
shows $\text{lookup} (\text{lead-monom} (\prod_{i=1..n} \text{sym-mpoly } A \ i \ \wedge \ h \ i :: 'a \ \text{mpoly})) \ k = 0$
 $\langle \text{proof} \rangle$

lemma *lead-monom-sym-mpoly-prod-in:*

assumes $\text{finite } A \ k < n$
shows $\text{lookup} (\text{lead-monom} (\prod_{i=1..n} \text{sym-mpoly } A \ i \ \wedge \ h \ i :: 'a \ \text{mpoly})) \ (xs \ !$
 $k) =$
 $(\sum_{i=k+1..n} h \ i)$
 $\langle \text{proof} \rangle$

lemma *lead-monom-sym-poly-powerprod-inj:*

assumes $\text{lead-monom} (\prod_{i=1..n} \text{sym-mpoly } A \ i \ \wedge \ \text{lookup } m1 \ i :: 'a \ \text{mpoly}) =$
 $\text{lead-monom} (\prod_{i=1..n} \text{sym-mpoly } A \ i \ \wedge \ \text{lookup } m2 \ i :: 'a \ \text{mpoly})$
assumes $\text{finite } A \ \text{keys } m1 \subseteq \{1..n\} \ \text{keys } m2 \subseteq \{1..n\}$
shows $m1 = m2$
 $\langle \text{proof} \rangle$

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

We have

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

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 $p(e_1, \dots, e_n) = 0$, 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 $p :: 'a \ \text{mpoly} \ \text{mpoly}$
assumes $\text{finite } A \ \text{insertion} (\text{sym-mpoly } A) \ p = 0$
 $\bigwedge m. \text{vars} (\text{coeff } p \ m) \cap A = \{\} \ \text{vars } p \subseteq \{1..n\}$
shows $p = 0$
 $\langle \text{proof} \rangle$

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

theorem *sym-mpoly-representation-unique:*

fixes $p :: 'a \text{ mpoly mpoly}$
assumes $\text{finite } A$
 $\text{insertion } (\text{sym-mpoly } A) p = \text{insertion } (\text{sym-mpoly } A) q$
 $\bigwedge m. \text{vars } (\text{coeff } p m) \cap A = \{\} \wedge m. \text{vars } (\text{coeff } q m) \cap A = \{\}$
 $\text{vars } p \subseteq \{1..n\} \text{ vars } q \subseteq \{1..n\}$
shows $p = q$
 $\langle \text{proof} \rangle$

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

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) $\text{check-symmetric-mpoly}$ **where**
 $\text{check-symmetric-mpoly } p \longleftrightarrow$
 $(\text{vars } p \cap A = \{\}) \vee$
 $A \subseteq \text{vars } p \wedge \text{decr } p \wedge \text{check-symmetric-mpoly } (p - \text{fund-sym-step-poly } p)$
 $\langle \text{proof} \rangle$

lemma $\text{check-symmetric-mpoly-dom-aux}$:
assumes $\text{finite } B \text{ vars } p \subseteq B A \subseteq B$
shows $\text{check-symmetric-mpoly-dom } p$
 $\langle \text{proof} \rangle$

lemma $\text{check-symmetric-mpoly-dom [intro]}$: $\text{check-symmetric-mpoly-dom } p$
 $\langle \text{proof} \rangle$

termination $\text{check-symmetric-mpoly}$
 $\langle \text{proof} \rangle$

lemmas $[\text{simp del}] = \text{check-symmetric-mpoly.simps}$

lemma $\text{check-symmetric-mpoly-correct}$: $\text{check-symmetric-mpoly } p \longleftrightarrow \text{symmetric-mpoly } A p$
 $\langle \text{proof} \rangle$

end

2.15 Symmetric functions of roots of a univariate polynomial

Consider a factored polynomial

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

where c_n is a unit.

Then any symmetric polynomial expression $q(x_1, \dots, x_n)$ in the roots x_i can be written as a polynomial expression $q'(c_0, \dots, c_{n-1})$ 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 $C :: 'b :: \text{comm-ring-1 set}$
and $A :: \text{nat set}$
and $\text{root} :: \text{nat} \Rightarrow 'a :: \text{comm-ring-1}$
and $l :: 'a \Rightarrow 'b$
and $q :: 'b \text{ mpoly}$
and $n :: \text{nat}$
defines $n \equiv \text{card } A$
assumes $C: \text{ring-closed } C \forall m. \text{coeff } q \ m \in C$
assumes $l: \text{ring-homomorphism } l$
assumes $\text{finite}: \text{finite } A$
assumes $\text{sym}: \text{symmetric-mpoly } A \ q$ **and** $\text{vars}: \text{vars } q \subseteq A$

begin

interpretation $\text{ring-closed } C \langle \text{proof} \rangle$

interpretation $\text{ring-homomorphism } l \langle \text{proof} \rangle$

theorem $\text{symmetric-poly-of-roots-conv-poly-of-coeffs}$:

assumes $c: \text{cinv} * l \ c = 1 \ \text{cinv} \in C$
assumes $p = \text{Polynomial.smult } c \ (\prod i \in A. [:-\text{root } i, 1:])$
obtains q' **where** $\text{vars } q' \subseteq \{0..<n\}$
and $\bigwedge m. \text{coeff } q' \ m \in C$
and $\text{insertion } (l \circ \text{poly.coeff } p) \ q' = \text{insertion } (l \circ \text{root}) \ q$

$\langle \text{proof} \rangle$

corollary $\text{symmetric-poly-of-roots-conv-poly-of-coeffs-monic}$:

assumes $p = (\prod i \in A. [:-\text{root } i, 1:])$
obtains q' **where** $\text{vars } q' \subseteq \{0..<n\}$
and $\bigwedge m. \text{coeff } q' \ m \in C$
and $\text{insertion } (l \circ \text{poly.coeff } p) \ q' = \text{insertion } (l \circ \text{root}) \ q$

$\langle \text{proof} \rangle$

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 $x_1, \dots, x_n \in S$, i. e. $p(X) = c(X - x_1) \dots (X - x_n)$. Let $q \in R[X_1, \dots, X_n]$ be some symmetric polynomial expression in the roots. Then $q(x_1, \dots, x_n) \in R$.

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 again 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* $(\lambda x. l\ (root\ x))\ q \in C$

<proof>

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* $(\lambda x. l\ (root\ x))\ q \in C$

<proof>

end

end

3 Executable Operations for Symmetric Polynomials

theory *Symmetric-Polynomials-Code*

imports *Symmetric-Polynomials Polynomials.MPoly-Type-Class-Finite-Map*

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 $(\forall x \in S. P\ x) = (\forall x \in T. P\ x)$

<proof>

lemma *compute-less-pp[code]*:

$xs < (ys :: 'a :: linorder \Rightarrow_0 'b :: \{zero, linorder\}) \longleftrightarrow$

$(\exists i \in keys\ xs \cup keys\ ys. lookup\ xs\ i < lookup\ ys\ i \wedge$

$(\forall j \in keys\ xs \cup keys\ ys. j < i \longrightarrow lookup\ xs\ j = lookup\ ys\ j))$

<proof>

lemma *compute-le-pp*[code]:

$xs \leq ys \iff xs = ys \vee xs < (ys :: - \Rightarrow_0 -)$
<proof>

lemma *vars-code* [code]:

$vars (MPoly p) = (\bigcup m \in keys p. keys m)$
<proof>

lemma *mpoly-coeff-code* [code]: $coeff (MPoly p) = lookup p$

<proof>

lemma *sym-mpoly-code* [code]:

$sym-mpoly (set xs) k = (\sum X \in Set.filter (\lambda X. card X = k) (Pow (set xs)). monom (monom-of-set X) 1)$
<proof>

lemma *monom-of-set-code* [code]:

$monom-of-set (set xs) = Pm-fmap (fmap-of-list (map (\lambda x. (x, 1)) xs))$
(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.