

Symmetric Polynomials

Manuel Eberl

June 16, 2019

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	5
2	Symmetric Polynomials	7
2.1	Auxiliary facts	7
2.2	Subrings and ring homomorphisms	9
2.3	Various facts about multivariate polynomials	11
2.4	Restricting a monomial to a subset of variables	18
2.5	Mapping over a polynomial	19
2.6	The leading monomial and leading coefficient	21
2.7	Turning a set of variables into a monomial	28
2.8	Permuting the variables of a polynomial	28
2.9	Symmetric polynomials	35
2.10	The elementary symmetric polynomials	39
2.11	Induction on the leading monomial	45
2.12	The fundamental theorem of symmetric polynomials	47
2.13	Uniqueness	57
2.14	A recursive characterisation of symmetry	63
2.15	Symmetric functions of roots of a univariate polynomial	65
3	Executable Operations for Symmetric Polynomials	68

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 $\text{card } (f^{-1} X \cap A) = \text{card } X$

proof -

have $\text{card } (f^{-1} X \cap A) = \text{card } (f^{-1} (f^{-1} X \cap A))$
by (*subst card-image*) (*auto intro!*: *inj-on-subset*[*OF inj*])
also have $f^{-1} (f^{-1} X \cap A) = X$
using *assms* **by** *auto*
finally show *?thesis* .

qed

lemma *bij-betw-image-fixed-card-subset*:

assumes *inj-on* *f* *A*
shows *bij-betw* $(\lambda X. f^{-1} X) \{X. X \subseteq A \wedge \text{card } X = k\} \{X. X \subseteq f^{-1} A \wedge \text{card } X = k\}$
using *assms inj-on-subset*[*OF assms*]
by (*intro bij-betwI*[*of - - -* $\lambda X. f^{-1} X \cap A$]) (*auto simp*: *card-image card-vimage-inter*)

lemma *image-image-fixed-card-subset*:

assumes *inj-on* *f* *A*
shows $(\lambda X. f^{-1} X)^{-1} \{X. X \subseteq A \wedge \text{card } X = k\} = \{X. X \subseteq f^{-1} A \wedge \text{card } X = k\}$
using *bij-betw-imp-surj-on*[*OF bij-betw-image-fixed-card-subset*[*OF assms*, *of k*]]
.

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

by (*induction A rule*: *infinite-finite-induct*) (*auto simp*: *algebra-simps*)

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 $(\prod_{x \in A}. \sum_{y \in B x}. f x y) = (\sum_{g \in \text{PiE } A B}. \prod_{x \in A}. f x (g x))$
using *assms*

proof (*induction A rule*: *finite-induct*)

case *empty*
thus *?case* **by** *auto*

next

case (*insert* *x* *A*)
have $(\sum_{g \in \text{PiE } (\text{insert } x A) B}. \prod_{x \in \text{insert } x A}. f x (g x)) =$

$(\sum_{g \in Pi_E} (insert\ x\ A)\ B.\ f\ x\ (g\ x) * (\prod_{x' \in A} f\ x'\ (g\ x')))$
using *insert by simp*
also have $(\lambda g. \prod_{x' \in A} f\ x'\ (g\ x')) = (\lambda g. \prod_{x' \in A} f\ x'\ (if\ x' = x\ then\ undefined\ else\ g\ x'))$
using *insert by (intro ext prod.cong) auto*
also have $(\sum_{g \in Pi_E} (insert\ x\ A)\ B.\ f\ x\ (g\ x) * \dots\ g) =$
 $(\sum_{(y,g) \in B\ x \times Pi_E\ A\ B} f\ x\ y * (\prod_{x' \in A} f\ x'\ (g\ x')))$
using *insert.premis insert.hyps*
by $(intro\ sum.reindex-bij-witness[of\ -\ \lambda(y,g).\ g(x := y)\ \lambda g.\ (g\ x,\ g(x := unde-
fined))])$
 $(auto\ simp:\ PiE-def\ extensional-def)$
also have $\dots = (\sum_{y \in B\ x} \sum_{g \in Pi_E\ A\ B} f\ x\ y * (\prod_{x' \in A} f\ x'\ (g\ x')))$
by *(subst sum.cartesian-product) auto*
also have $\dots = (\sum_{y \in B\ x} f\ x\ y) * (\sum_{g \in Pi_E\ A\ B} \prod_{x' \in A} f\ x'\ (g\ x'))$
using *insert by (subst sum.swap) (simp add: sum-distrib-left sum-distrib-right)*
also have $(\sum_{g \in Pi_E\ A\ B} \prod_{x' \in A} f\ x'\ (g\ x')) = (\prod_{x \in A} \sum_{y \in B\ x} f\ x\ y)$
using *insert.premis by (intro insert.IH [symmetric]) auto*
also have $(\sum_{y \in B\ x} f\ x\ y) * \dots = (\prod_{x \in insert\ x\ A} \sum_{y \in B\ x} f\ x\ y)$
using *insert.hyps by simp*
finally show *?case ..*
qed

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 $-$
have $(\prod_{x \in A} f1\ x + f2\ x) = (\sum_{g \in A \rightarrow_E\ UNIV} \prod_{x \in A} if\ g\ x\ then\ f1\ x\ else\ f2\ x)$
using *prod-sum-PiE[of A λ -. UNIV :: bool set $\lambda x\ y.$ if y then f1 x else f2 x]*
assms
by *(simp-all add: UNIV-bool add-ac)*
also have $\dots = (\sum_{X \in Pow\ A} \prod_{x \in A} if\ x \in X\ then\ f1\ x\ else\ f2\ x)$
by *(intro sum.reindex-bij-witness*
 $[of\ -\ \lambda X\ x. if\ x \in A\ then\ x \in X\ else\ undefined\ \lambda P. \{x \in A. P\ x\}])\ auto$
also have $\dots = (\sum_{X \in Pow\ A} (\prod_{x \in X} f1\ x) * (\prod_{x \in A-X} f2\ x))$
proof *(intro sum.cong refl, goal-cases)*
case $(1\ X)$
let $?f = \lambda x. if\ x \in X\ then\ f1\ x\ else\ f2\ x$
have $prod\ f1\ X * prod\ f2\ (A - X) = prod\ ?f\ X * prod\ ?f\ (A - X)$
by *(intro arg-cong2[of - - - (*)] prod.cong) auto*
also have $\dots = prod\ ?f\ (X \cup (A - X))$
using 1 **by** *(subst prod.union-disjoint) (auto intro: finite-subset[OF - finite])*
also have $X \cup (A - X) = A$ **using** 1 **by** *auto*
finally show *?case ..*
qed
finally show *?thesis .*
qed

corollary *prod-diff1*:

fixes $f1\ f2 :: 'a \Rightarrow 'c :: \text{comm-ring-1}$
assumes *finite*: $\text{finite } A$
shows $(\prod_{x \in A}. f1\ x - f2\ x) = (\sum_{X \in \text{Pow } A}. (-1) \wedge \text{card } X * (\prod_{x \in X}. f2\ x) * (\prod_{x \in A-X}. f1\ x))$
proof –
have $(\prod_{x \in A}. f1\ x - f2\ x) = (\prod_{x \in A}. -f2\ x + f1\ x)$
by *simp*
also have $\dots = (\sum_{X \in \text{Pow } A}. (\prod_{x \in X}. -f2\ x) * \text{prod } f1\ (A - X))$
by (*rule prod-add*) *fact+*
also have $\dots = (\sum_{X \in \text{Pow } A}. (-1) \wedge \text{card } X * (\prod_{x \in X}. f2\ x) * \text{prod } f1\ (A - X))$
by (*simp add: prod-uminus*)
finally show *?thesis* .
qed

corollary *prod-diff2*:

fixes $f1\ f2 :: 'a \Rightarrow 'c :: \text{comm-ring-1}$
assumes *finite*: $\text{finite } A$
shows $(\prod_{x \in A}. f1\ x - f2\ x) = (\sum_{X \in \text{Pow } A}. (-1) \wedge (\text{card } A - \text{card } X) * (\prod_{x \in X}. f1\ x) * (\prod_{x \in A-X}. f2\ x))$
proof –
have $(\prod_{x \in A}. f1\ x - f2\ x) = (\prod_{x \in A}. f1\ x + (-f2\ x))$
by *simp*
also have $\dots = (\sum_{X \in \text{Pow } A}. (\prod_{x \in X}. f1\ x) * (\prod_{x \in A-X}. -f2\ x))$
by (*rule prod-add*) *fact+*
also have $\dots = (\sum_{X \in \text{Pow } A}. (-1) \wedge \text{card } (A - X) * (\prod_{x \in X}. f1\ x) * (\prod_{x \in A-X}. f2\ x))$
by (*simp add: prod-uminus mult-ac*)
also have $\dots = (\sum_{X \in \text{Pow } A}. (-1) \wedge (\text{card } A - \text{card } X) * (\prod_{x \in X}. f1\ x) * (\prod_{x \in A-X}. f2\ x))$
using *finite-subset[OF - assms]* **by** (*intro sum.cong refl, subst card-Diff-subset*)
auto
finally show *?thesis* .
qed

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

proof –
have $(\prod_{x \in A}. [-f x, 1:]) = (\prod_{x \in A}. [:0, 1:] - [:f x:])$
by *simp*
also have $\dots = (\sum_{X \in \text{Pow } A}. (-1) ^ \text{card } X * (\prod_{x \in X}. [:f x:]) * \text{monom } 1$
 $(\text{card } (A - X)))$
using *fin* **by** (*subst prod-diff1*) (*auto simp: monom-altdef mult-ac*)
also have $\dots = (\sum_{X \in \text{Pow } A}. \text{monom } ((-1) ^ \text{card } X * (\prod_{x \in X}. f x)) (\text{card}$
 $(A - X)))$
proof (*intro sum.cong refl, goal-cases*)
case (1 X)
have $(-1 :: 'b \text{ poly}) ^ \text{card } X = [:(-1) ^ \text{card } X:]$
by (*induction X rule: infinite-finite-induct*) (*auto simp: one-pCons algebra-simps*)
moreover have $(\prod_{x \in X}. [:f x:]) = [:\prod_{x \in X}. f x:]$
by (*induction X rule: infinite-finite-induct*) *auto*
ultimately show ?*case* **by** (*simp add: smult-monom*)
qed
finally show ?*thesis* .
qed

Comparing coefficients yields Vieta's formula:

theorem *coeff-poly-from-roots*:
fixes $f :: 'a \Rightarrow 'b :: \text{comm-ring-1}$
assumes *fin*: *finite A* **and** $k: k \leq \text{card } A$
shows $\text{coeff } (\prod_{x \in A}. [-f x, 1:]) k =$
 $(-1) ^ (\text{card } A - k) * (\sum_{X \mid X \subseteq A \wedge \text{card } X = \text{card } A - k}. (\prod_{x \in X}. f x))$
proof –
have $(\prod_{x \in A}. [-f x, 1:]) = (\sum_{X \in \text{Pow } A}. \text{monom } ((-1) ^ \text{card } X * (\prod_{x \in X}. f x)) (\text{card } (A - X)))$
by (*intro poly-from-roots fin*)
also have $\text{coeff } \dots k = (\sum_{X \mid X \subseteq A \wedge \text{card } X = \text{card } A - k}. (-1) ^ (\text{card } A - k) * (\prod_{x \in X}. f x))$
unfolding *coeff-sum coeff-monom* **using** *finite-subset[OF - fin] k card-mono[OF fin]*
by (*intro sum.mono-neutral-cong-right*) (*auto simp: card-Diff-subset*)
also have $\dots = (-1) ^ (\text{card } A - k) * (\sum_{X \mid X \subseteq A \wedge \text{card } X = \text{card } A - k}. (\prod_{x \in X}. f x))$
by (*simp add: sum-distrib-left*)
finally show ?*thesis* .
qed

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

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

```

have coeff ( $\prod x \in A. [-f x, 1:]$ )  $k =$ 
   $(-1)^{\wedge} (\text{card } A - k) * (\sum X \mid X \subseteq A \wedge \text{card } X = \text{card } A - k. (\prod x \in X. f x))$ 
by (intro coeff-poly-from-roots assms)
also have  $(\sum X \mid X \subseteq A \wedge \text{card } X = \text{card } A - k. (\prod x \in X. f x)) =$ 
   $(\sum X \mid X \subseteq A \wedge \text{card } X = \text{card } A - k. \prod (f'X))$ 
by (intro sum.cong refl, subst prod.reindex) (auto intro: inj-on-subset[OF inj])
also have  $\dots = (\sum X \in (\lambda X. f'X) \cdot \{X. X \subseteq A \wedge \text{card } X = \text{card } A - k\}. \prod X)$ 
by (subst sum.reindex) (auto intro!: inj-on-image inj-on-subset[OF inj])
also have  $(\lambda X. f'X) \cdot \{X. X \subseteq A \wedge \text{card } X = \text{card } A - k\} = \{X. X \subseteq f' A \wedge \text{card } X = \text{card } A - k\}$ 
by (intro image-image-fixed-card-subset inj)
finally show ?thesis .
qed

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* $\{X. X \subseteq A \wedge \text{infinite } X\}$

proof –

have $\forall k. \exists X. X \subseteq A \wedge \text{infinite } X \wedge \text{card } (A - X) = k$ **for** $k :: \text{nat}$

proof

fix $k :: \text{nat}$ **obtain** Y **where** *finite* Y $\text{card } Y = k$ $Y \subseteq A$

using *infinite-arbitrarily-large[of A k] assms* **by** *auto*

moreover from this have $A - (A - Y) = Y$ **by** *auto*

ultimately show $\exists X. X \subseteq A \wedge \text{infinite } X \wedge \text{card } (A - X) = k$

using *assms* **by** (*intro exI[of - A - Y]*) *auto*

qed

from *choice[OF this]* **obtain** f

where $f: \wedge k. f k \subseteq A \wedge \text{infinite } (f k) \wedge \text{card } (A - f k) = k$ **by** *blast*

have $k = l$ **if** $f k = f l$ **for** $k l$

proof (*rule ccontr*)

assume $k \neq l$

hence $\text{card } (A - f k) \neq \text{card } (A - f l)$

```

    using f[of k] f[of l] by auto
    with ⟨f k = f l⟩ show False by simp
qed
hence inj f by (auto intro: injI)
moreover have range f ⊆ {X. X ⊆ A ∧ infinite X}
  using f by auto
ultimately show ?thesis
  by (subst infinite-iff-countable-subset) auto
qed

```

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 -
  obtain B where B: B ⊆ A finite B card B = k - 1
    using infinite-arbitrarily-large[OF assms(1), of k - 1] by blast
  define f where f = (λx. insert x B)
  have f' (A - B) ⊆ {X. X ⊆ A ∧ finite X ∧ card X = k}
    using assms B by (auto simp: f-def)
  moreover have inj-on f (A - B)
    by (auto intro!: inj-onI simp: f-def)
  hence infinite (f' (A - B))
    using assms B by (subst finite-image-iff) auto
  ultimately show ?thesis
    by (rule infinite-super)
qed

```

```

lemma comp-bij-eq-iff:
  assumes bij f
  shows g ∘ f = h ∘ f ⟷ g = h
proof
  assume *: g ∘ f = h ∘ f
  show g = h
  proof
    fix x
    obtain y where [simp]: x = f y using bij-is-surj[OF assms] by auto
    have (g ∘ f) y = (h ∘ f) y by (simp only: *)
    thus g x = h x by simp
  qed
qed auto

```

```

lemma sum-list-replicate [simp]:
  sum-list (replicate n x) = of-nat n * (x :: 'a :: semiring-1)
  by (induction n) (auto simp: algebra-simps)

```

```

lemma ex-subset-of-card:
  assumes finite A card A ≥ k

```



```

shows  $\exists B. B \subseteq A \wedge \text{card } B = k$ 
using assms
proof (induction arbitrary: k rule: finite-induct)
  case empty
  thus ?case by auto
next
  case (insert x A k)
  show ?case
  proof (cases k = 0)
    case True
    thus ?thesis by (intro exI[of - {}]) auto
  next
  case False
  from insert have  $\exists B \subseteq A. \text{card } B = k - 1$  by (intro insert.IH) auto
  then obtain B where  $B \subseteq A \text{ card } B = k - 1$  by auto
  with insert have [simp]:  $x \notin B$  by auto
  have  $\text{insert } x B \subseteq \text{insert } x A$ 
    using B insert by auto
  moreover have  $\text{card } (\text{insert } x B) = k$ 
    using insert B finite-subset[of B A] False by (subst card-insert) auto
  ultimately show ?thesis by blast
qed
qed

```

```

lemma length-sorted-list-of-set [simp]:  $\text{length } (\text{sorted-list-of-set } A) = \text{card } A$ 
using distinct-card[of sorted-list-of-set A] by (cases finite A) simp-all

```

```

lemma upt-add-eq-append':  $i \leq j \implies j \leq k \implies [i..<k] = [i..<j] @ [j..<k]$ 
using upt-add-eq-append[of i j k - j] by simp

```

2.2 Subrings and ring homomorphisms

```

locale ring-closed =
  fixes  $A :: 'a :: \text{comm-ring-1 set}$ 
  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]:  $x \in A \implies y \in A \implies x - y \in A$ 
using add-closed[of x -y] uminus-closed[of y] by auto

```

```

lemma sum-closed [intro]:  $(\bigwedge x. x \in X \implies f x \in A) \implies \text{sum } f X \in A$ 
by (induction X rule: infinite-finite-induct) auto

```

```

lemma power-closed [intro]:  $x \in A \implies x ^ n \in A$ 
by (induction n) auto

```

lemma *Sum-any-closed* [intro]: $(\bigwedge x. f x \in A) \implies \text{Sum-any } f \in A$
unfolding *Sum-any.expand-set* **by** (rule *sum-closed*)

lemma *prod-closed* [intro]: $(\bigwedge x. x \in X \implies f x \in A) \implies \text{prod } f X \in A$
by (induction *X* rule: *infinite-finite-induct*) *auto*

lemma *Prod-any-closed* [intro]: $(\bigwedge x. f x \in A) \implies \text{Prod-any } f \in A$
unfolding *Prod-any.expand-set* **by** (rule *prod-closed*)

lemma *prod-fun-closed* [intro]: $(\bigwedge x. f x \in A) \implies (\bigwedge x. g x \in A) \implies \text{prod-fun } f g$
 $x \in A$
by (*auto simp: prod-fun-def when-def intro!*: *Sum-any-closed mult-closed*)

lemma *of-nat-closed* [simp, intro]: *of-nat* $n \in A$
by (induction *n*) *auto*

lemma *of-int-closed* [simp, intro]: *of-int* $n \in A$
by (induction *n*) *auto*

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]: $f (x - y) = f x - f y$
using *add*[*of* $x - y$] **by** (*simp del: add*)

lemma *power* [simp]: $f (x \wedge n) = f x \wedge n$
by (induction *n*) *auto*

lemma *sum* [simp]: $f (\text{sum } g A) = (\sum x \in A. f (g x))$
by (induction *A* rule: *infinite-finite-induct*) *auto*

lemma *prod* [simp]: $f (\text{prod } g A) = (\prod x \in A. f (g x))$
by (induction *A* rule: *infinite-finite-induct*) *auto*

end

lemma *ring-homomorphism-id* [intro]: *ring-homomorphism* *id*
by *standard auto*

lemma *ring-homomorphism-id'* [intro]: *ring-homomorphism* $(\lambda x. x)$

by *standard auto*

lemma *ring-homomorphism-of-int* [intro]: *ring-homomorphism of-int*
by *standard auto*

2.3 Various facts about multivariate polynomials

lemma *poly-mapping-nat-ge-0* [simp]: $(m :: \text{nat} \Rightarrow_0 \text{nat}) \geq 0$

proof (*cases* $m = 0$)

case *False*

hence *Poly-Mapping.lookup* $m \neq \text{Poly-Mapping.lookup } 0$ by *transfer auto*

hence $\exists k. \text{Poly-Mapping.lookup } m \ k \neq 0$ by (*auto simp: fun-eq-iff*)

from *LeastI-ex*[OF *this*] *Least-le*[of $\lambda k. \text{Poly-Mapping.lookup } m \ k \neq 0$] **show**
?thesis

by (*force simp: less-eq-poly-mapping-def less-fun-def*)

qed *auto*

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

unfolding *less-eq-poly-mapping-def poly-mapping-eq-iff less-fun-def* by *auto*

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

assumes $m \geq n$

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

by (*auto intro!: poly-mapping-eqI simp: lookup-of-nat lookup-minus when-def*)

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

rel-fun cr-mpoly (=) *poly-mapping.lookup MPoly-Type.coeff*

unfolding *MPoly-Type.coeff-def* by *transfer-prover*

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

by (*induction A rule: infinite-finite-induct*) (*auto simp: plus-mpoly.rep-eq zero-mpoly.rep-eq*)

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

by *transfer auto*

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

by (*simp add: Sum-any.expand-set mapping-of-sum*)

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

(rel-fun (rel-fun (=) cr-mpoly) Sum-any Sum-any

by (*auto simp: rel-fun-def cr-mpoly-def Sum-any-mapping-of*)

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

proof –

have $\text{of-nat } n * m = (\sum i < n. m)$ by *simp*

also have $\text{lookup } \dots \ k = (\sum i < n. \text{lookup } m \ k)$

by (*simp only: lookup-sum*)

also have $\dots = n * \text{lookup } m \ k$

by *simp*
finally show *?thesis* .
qed

lemma *mpoly-eqI*:
assumes $\bigwedge mon. MPoly-Type.coeff\ p\ mon = MPoly-Type.coeff\ q\ mon$
shows $p = q$
using *assms* **by** (*transfer*, *transfer*) (*auto simp: fun-eq-iff*)

lemma *coeff-mpoly-times*:
 $MPoly-Type.coeff\ (p * q)\ mon = prod-fun\ (MPoly-Type.coeff\ p)\ (MPoly-Type.coeff\ q)\ mon$
by (*transfer'*, *transfer'*) *auto*

lemma (**in** *ring-closed*) *coeff-mult-closed* [*intro*]:
 $(\bigwedge x. coeff\ p\ x \in A) \implies (\bigwedge x. coeff\ q\ x \in A) \implies coeff\ (p * q)\ x \in A$
by (*auto simp: coeff-mpoly-times prod-fun-closed*)

lemma *coeff-notin-vars*:
assumes $\neg(keys\ m \subseteq vars\ p)$
shows $coeff\ p\ m = 0$
using *assms* **unfolding** *vars-def* **by** *transfer'* (*auto simp: in-keys-iff*)

lemma *finite-coeff-support* [*intro*]: *finite* $\{m. coeff\ p\ m \neq 0\}$
by *transfer simp*

lemma *insertion-altdef*:
 $insertion\ f\ p = Sum-any\ (\lambda m. coeff\ p\ m * Prod-any\ (\lambda i. f\ i \wedge lookup\ m\ i))$
by (*transfer'*, *transfer'*) (*simp add: insertion-fun-def*)

lemma *mpoly-coeff-uminus* [*simp*]: $coeff\ (-p)\ m = -coeff\ p\ m$
by *transfer auto*

lemma *Sum-any-uminus*: $Sum-any\ (\lambda x. -f\ x :: 'a :: ab-group-add) = -Sum-any\ f$
by (*simp add: Sum-any.expand-set sum-negf*)

lemma *insertion-uminus* [*simp*]: $insertion\ f\ (-p :: 'a :: comm-ring-1\ mpoly) = -insertion\ f\ p$
by (*simp add: insertion-altdef Sum-any-uminus*)

lemma *Sum-any-lookup*: $finite\ \{x. g\ x \neq 0\} \implies Sum-any\ (\lambda x. lookup\ (g\ x)\ y) = lookup\ (Sum-any\ g)\ y$
by (*auto simp: Sum-any.expand-set lookup-sum intro!: sum.mono-neutral-left*)

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

g
proof –
have $\{x. f x - g x \neq 0\} \subseteq \{x. f x \neq 0\} \cup \{x. g x \neq 0\}$ **by** *auto*
moreover have *finite* $(\{x. f x \neq 0\} \cup \{x. g x \neq 0\})$
by (*subst finite-Un*) (*insert assms, auto*)
ultimately have *finite* $\{x. f x - g x \neq 0\}$
by (*rule finite-subset*)
with *assms* **show** *?thesis*
by (*simp add: algebra-simps Sum-any.distrib [symmetric]*)
qed

lemma *insertion-diff*:
 $insertion\ f\ (p - q :: 'a :: comm-ring-1\ mpoly) = insertion\ f\ p - insertion\ f\ q$
proof (*transfer, transfer*)
fix $f :: nat \Rightarrow 'a$ **and** $p\ q :: (nat \Rightarrow_0\ nat) \Rightarrow 'a$
assume *fin*: *finite* $\{x. p\ x \neq 0\}$ *finite* $\{x. q\ x \neq 0\}$
have $insertion\text{-}fun\ f\ (\lambda x. p\ x - q\ x) =$
 $(\sum m. p\ m * (\prod v. f\ v \wedge lookup\ m\ v) - q\ m * (\prod v. f\ v \wedge lookup\ m\ v))$
by (*simp add: insertion-fun-def algebra-simps Sum-any-diff*)
also have $\dots = (\sum m. p\ m * (\prod v. f\ v \wedge lookup\ m\ v)) - (\sum m. q\ m * (\prod v. f\ v \wedge lookup\ m\ v))$
by (*subst Sum-any-diff*) (*auto intro: finite-subset[OF - fin(1)] finite-subset[OF - fin(2)]*)
also have $\dots = insertion\text{-}fun\ f\ p - insertion\text{-}fun\ f\ q$
by (*simp add: insertion-fun-def*)
finally show $insertion\text{-}fun\ f\ (\lambda x. p\ x - q\ x) = \dots$
qed

lemma *insertion-power*: $insertion\ f\ (p \wedge n) = insertion\ f\ p \wedge n$
by (*induction n*) (*simp-all add: insertion-mult*)

lemma *insertion-sum*: $insertion\ f\ (sum\ g\ A) = (\sum x \in A. insertion\ f\ (g\ x))$
by (*induction A rule: infinite-finite-induct*) (*auto simp: insertion-add*)

lemma *insertion-prod*: $insertion\ f\ (prod\ g\ A) = (\prod x \in A. insertion\ f\ (g\ x))$
by (*induction A rule: infinite-finite-induct*) (*auto simp: insertion-mult*)

lemma *coeff-Var*: $coeff\ (Var\ i)\ m = (1\ when\ m = Poly\text{-}Mapping.\ single\ i\ 1)$
by *transfer'* (*auto simp: Var₀-def lookup-single when-def*)

lemma *vars-Var*: $vars\ (Var\ i :: 'a :: \{one, zero\}\ mpoly) = (if\ (0 :: 'a) = 1\ then\ \{\}\ else\ \{i\})$
unfolding *vars-def* **by** (*auto simp: Var.rep-eq Var₀-def*)

lemma *insertion-Var* [*simp*]: $insertion\ f\ (Var\ i) = f\ i$

proof –
have $insertion\ f\ (Var\ i) = (\sum m. (1\ when\ m = Poly\text{-}Mapping.\ single\ i\ 1) * (\prod i. f\ i \wedge lookup\ m\ i))$
by (*simp add: insertion-altdef coeff-Var*)

also have $\dots = (\prod j. f j \wedge \text{lookup } (\text{Poly-Mapping.single } i \ 1) \ j)$
by $(\text{subst Sum-any.expand-superset}[\text{of } \{\text{Poly-Mapping.single } i \ 1\}])$ $(\text{auto simp: when-def})$
also have $\dots = f \ i$
by $(\text{subst Prod-any.expand-superset}[\text{of } \{i\}])$ $(\text{auto simp: when-def lookup-single})$
finally show $?thesis$.
qed

lemma insertion-Sum-any:
assumes $\text{finite } \{x. g \ x \neq 0\}$
shows $\text{insertion } f \ (\text{Sum-any } g) = \text{Sum-any } (\lambda x. \text{insertion } f \ (g \ x))$
unfolding $\text{Sum-any.expand-set insertion-sum}$
by $(\text{intro sum.mono-neutral-right})$ $(\text{auto intro!: finite-subset}[OF - \text{assms}])$

lemma keys-diff-subset:
 $\text{keys } (f - g) \subseteq \text{keys } f \cup \text{keys } g$
by transfer auto

lemma keys-empty-iff [simp]: $\text{keys } p = \{\} \longleftrightarrow p = 0$
by transfer auto

lemma mpoly-coeff-0 [simp]: $\text{MPoly-Type.coeff } 0 \ m = 0$
by transfer auto

lemma lookup-1: $\text{lookup } 1 \ m = (\text{if } m = 0 \ \text{then } 1 \ \text{else } 0)$
by $\text{transfer (simp add: when-def)}$

lemma mpoly-coeff-1: $\text{MPoly-Type.coeff } 1 \ m = (\text{if } m = 0 \ \text{then } 1 \ \text{else } 0)$
by $(\text{simp add: MPoly-Type.coeff-def one-mpoly.rep-eq lookup-1})$

lemma lookup-Const₀: $\text{lookup } (\text{Const}_0 \ c) \ m = (\text{if } m = 0 \ \text{then } c \ \text{else } 0)$
unfolding $\text{Const}_0\text{-def}$ **by** $(\text{simp add: lookup-single when-def})$

lemma mpoly-coeff-Const: $\text{MPoly-Type.coeff } (\text{Const } c) \ m = (\text{if } m = 0 \ \text{then } c \ \text{else } 0)$
by $(\text{simp add: MPoly-Type.coeff-def Const.rep-eq lookup-Const}_0)$

lemma coeff-smult [simp]: $\text{coeff } (\text{smult } c \ p) \ m = (c :: 'a :: \text{mult-zero}) * \text{coeff } p \ m$
by $\text{transfer (auto simp: map-lookup)}$

lemma in-keys-mapI: $x \in \text{keys } m \implies f \ (\text{lookup } m \ x) \neq 0 \implies x \in \text{keys } (\text{Poly-Mapping.map } f \ m)$
by transfer auto

lemma keys-uminus [simp]: $\text{keys } (-m) = \text{keys } m$
by transfer auto

lemma vars-uminus [simp]: $\text{vars } (-p) = \text{vars } p$
unfolding vars-def **by** transfer' auto

lemma *vars-smult*: $\text{vars } (\text{smult } c \ p) \subseteq \text{vars } p$
unfolding *vars-def* **by** (*transfer'*, *transfer'*) *auto*

lemma *vars-0* [*simp*]: $\text{vars } 0 = \{\}$
unfolding *vars-def* **by** *transfer' simp*

lemma *vars-1* [*simp*]: $\text{vars } 1 = \{\}$
unfolding *vars-def* **by** *transfer' simp*

lemma *vars-sum*: $\text{vars } (\text{sum } f \ A) \subseteq (\bigcup x \in A. \text{vars } (f \ x))$
using *vars-add* **by** (*induction A rule: infinite-finite-induct*) *auto*

lemma *vars-prod*: $\text{vars } (\text{prod } f \ A) \subseteq (\bigcup x \in A. \text{vars } (f \ x))$
using *vars-mult* **by** (*induction A rule: infinite-finite-induct*) *auto*

lemma *vars-Sum-any*: $\text{vars } (\text{Sum-any } h) \subseteq (\bigcup i. \text{vars } (h \ i))$
unfolding *Sum-any.expand-set* **by** (*intro order.trans[OF vars-sum]*) *auto*

lemma *vars-Prod-any*: $\text{vars } (\text{Prod-any } h) \subseteq (\bigcup i. \text{vars } (h \ i))$
unfolding *Prod-any.expand-set* **by** (*intro order.trans[OF vars-prod]*) *auto*

lemma *vars-power*: $\text{vars } (p \ ^n) \subseteq \text{vars } p$
using *vars-mult* **by** (*induction n*) *auto*

lemma *vars-diff*: $\text{vars } (p1 \ - \ p2) \subseteq \text{vars } p1 \cup \text{vars } p2$
unfolding *vars-def*
proof *transfer'*
fix *p1 p2* :: (*nat* \Rightarrow_0 *nat*) \Rightarrow_0 'a
show $\bigcup (\text{keys } ' \ \text{keys } (p1 \ - \ p2)) \subseteq \bigcup (\text{keys } ' \ (\text{keys } p1)) \cup \bigcup (\text{keys } ' \ (\text{keys } p2))$
using *keys-diff-subset[of p1 p2]* **by** (*auto simp flip: not-in-keys-iff-lookup-eq-zero*)
qed

lemma *insertion-smult* [*simp*]: $\text{insertion } f \ (\text{smult } c \ p) = c * \text{insertion } f \ p$
unfolding *insertion-altdef*
by (*subst Sum-any-right-distrib*)
(auto intro: finite-subset[OF - finite-coeff-support[of p]] simp: mult.assoc)

lemma *coeff-add* [*simp*]: $\text{coeff } (p \ + \ q) \ m = \text{coeff } p \ m + \text{coeff } q \ m$
by *transfer' (simp add: lookup-add)*

lemma *coeff-diff* [*simp*]: $\text{coeff } (p \ - \ q) \ m = \text{coeff } p \ m - \text{coeff } q \ m$
by *transfer' (simp add: lookup-minus)*

lemma *insertion-monom* [*simp*]:
 $\text{insertion } f \ (\text{monom } m \ c) = c * \text{Prod-any } (\lambda x. f \ x \ ^{\text{lookup } m \ x})$
proof –
have $\text{insertion } f \ (\text{monom } m \ c) =$
 $(\sum m'. (c \ \text{when } m = m') * (\prod v. f \ v \ ^{\text{lookup } m' \ v}))$

by (*simp add: insertion-def insertion-aux-def insertion-fun-def lookup-single*)
 also have $\dots = c * (\prod v. f v \wedge \text{lookup } m v)$
 by (*subst Sum-any.expand-superset[of {m}]*) (*auto simp: when-def*)
 finally show ?thesis .
 qed

lemma *insertion-aux-Const₀ [simp]: insertion-aux f (Const₀ c) = c*
proof –
 have *insertion-aux f (Const₀ c) = ($\sum m. (c \text{ when } m = 0) * (\prod v. f v \wedge \text{lookup } m v)$)*
 by (*simp add: Const₀-def insertion-aux-def insertion-fun-def lookup-single*)
 also have $\dots = (\sum m \in \{0\}. (c \text{ when } m = 0) * (\prod v. f v \wedge \text{lookup } m v))$
 by (*intro Sum-any.expand-superset*) (*auto simp: when-def*)
 also have $\dots = c$ by *simp*
 finally show ?thesis .
 qed

lemma *insertion-Const [simp]: insertion f (Const c) = c*
 by (*simp add: insertion-def Const.rep-eq*)

lemma *coeffs-0 [simp]: coeffs 0 = {}*
 by *transfer auto*

lemma *coeffs-1 [simp]: coeffs 1 = {1}*
 by *transfer auto*

lemma *coeffs-Const: coeffs (Const c) = (if c = 0 then {} else {c})*
 unfolding *Const-def Const₀-def* by *transfer' auto*

lemma *coeffs-subset: coeffs (Const c) \subseteq {c}*
 by (*auto simp: coeffs-Const*)

lemma *keys-Const₀: keys (Const₀ c) = (if c = 0 then {} else {0})*
 unfolding *Const₀-def* by *transfer' auto*

lemma *vars-Const [simp]: vars (Const c) = {}*
 unfolding *vars-def* by *transfer' (auto simp: keys-Const₀)*

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 \circ f) (m2 \circ f) x*
proof –
 have [*simp*]: $f x = f y \iff x = y$ for $x y$
 using $\langle \text{bij } f \rangle$ by (*auto dest!: bij-is-inj inj-onD*)
 have *prod-fun (m1 \circ f) (m2 \circ f) x =*
 *Sum-any (($\lambda l. m1 l * \text{Sum-any } ((\lambda q. m2 q \text{ when } f x = l + q) \circ f)) \circ f)$*
 by (*simp add: prod-fun-def f(1) [symmetric] o-def*)
 also have $\dots = \text{Sum-any } ((\lambda l. m1 l * \text{Sum-any } ((\lambda q. m2 q \text{ when } f x = l + q))))$
 by (*simp only: Sum-any.reindex-cong[OF assms(1) refl, symmetric]*)

also have $\dots = \text{prod-fun } m1 \ m2 \ (f \ x)$
by (*simp add: prod-fun-def*)
finally show *?thesis ..*
qed

lemma *add-nat-poly-mapping-zero-iff* [*simp*]:
 $(a + b :: 'a \Rightarrow_0 \text{nat}) = 0 \longleftrightarrow a = 0 \wedge b = 0$
by *transfer (auto simp: fun-eq-iff)*

lemma *prod-fun-nat-0*:
fixes $f \ g :: ('a \Rightarrow_0 \text{nat}) \Rightarrow 'b::\text{semiring-0}$
shows $\text{prod-fun } f \ g \ 0 = f \ 0 * g \ 0$
proof –
have $\text{prod-fun } f \ g \ 0 = (\sum l. f \ l * (\sum q. g \ q \ \text{when } 0 = l + q))$
unfolding *prod-fun-def ..*
also have $(\lambda l. \sum q. g \ q \ \text{when } 0 = l + q) = (\lambda l. \sum q \in \{0\}. g \ q \ \text{when } 0 = l + q)$
by (*intro ext Sum-any.expand-superset*) (*auto simp: when-def*)
also have $(\sum l. f \ l * \dots \ l) = (\sum l \in \{0\}. f \ l * \dots \ l)$
by (*intro ext Sum-any.expand-superset*) (*auto simp: when-def*)
finally show *?thesis by simp*
qed

lemma *mpoly-coeff-times-0*: $\text{coeff } (p * q) \ 0 = \text{coeff } p \ 0 * \text{coeff } q \ 0$
by (*simp add: coeff-mpoly-times prod-fun-nat-0*)

lemma *mpoly-coeff-prod-0*: $\text{coeff } (\prod x \in A. f \ x) \ 0 = (\prod x \in A. \text{coeff } (f \ x) \ 0)$
by (*induction A rule: infinite-finite-induct*) (*auto simp: mpoly-coeff-times-0 mpoly-coeff-1*)

lemma *mpoly-coeff-power-0*: $\text{coeff } (p \ ^n) \ 0 = \text{coeff } p \ 0 \ ^n$
by (*induction n*) (*auto simp: mpoly-coeff-times-0 mpoly-coeff-1*)

lemma *prod-fun-max*:
fixes $f \ g :: 'a::\{\text{linorder, ordered-cancel-comm-monoid-add}\} \Rightarrow 'b::\text{semiring-0}$
assumes *zero*: $\bigwedge m. m > a \implies f \ m = 0 \ \bigwedge m. m > b \implies g \ m = 0$
assumes *fin*: *finite* $\{m. f \ m \neq 0\}$ *finite* $\{m. g \ m \neq 0\}$
shows $\text{prod-fun } f \ g \ (a + b) = f \ a * g \ b$
proof –
note $\text{fin}' = \text{finite-subset}[OF - \text{fin}(1)] \ \text{finite-subset}[OF - \text{fin}(2)]$
have $\text{prod-fun } f \ g \ (a + b) = (\sum l. f \ l * (\sum q. g \ q \ \text{when } a + b = l + q))$
by (*simp add: prod-fun-def Sum-any-right-distrib*)
also have $\dots = (\sum l. \sum q. f \ l * g \ q \ \text{when } a + b = l + q)$
by (*subst Sum-any-right-distrib*) (*auto intro!: Sum-any.cong fin'(2) simp: when-def*)
also {
fix $l \ q$ **assume** $lq: a + b = l + q$ ($a, b \neq (l, q)$) **and** $nz: f \ l * g \ q \neq 0$
from *nz* **and** *zero* **have** $l \leq a \leq b$ **by** (*auto intro: leI*)
moreover from *this* **and** $lq(2)$ **have** $l < a \vee q < b$ **by** *auto*
ultimately have $l + q < a + b$
by (*auto intro: add-less-le-mono add-le-less-mono*)

with $lq(1)$ **have** *False* **by** *simp*
}
hence $(\sum l. \sum q. f l * g q \text{ when } a + b = l + q) = (\sum l. \sum q. f l * g q \text{ when } (a, b) = (l, q))$
by (*intro Sum-any.cong refl*) (*auto simp: when-def*)
also have $\dots = (\sum (l,q). f l * g q \text{ when } (a, b) = (l, q))$
by (*intro Sum-any.cartesian-product[of {(a, b)}]*) *auto*
also have $\dots = (\sum (l,q) \in \{(a,b)\}. f l * g q \text{ when } (a, b) = (l, q))$
by (*intro Sum-any.expand-superset*) *auto*
also have $\dots = f a * g b$ **by** *simp*
finally show *?thesis* .
qed

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

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

proof –

note $\text{fin}' = \text{finite-subset}[OF - \text{fin}(1)] \text{ finite-subset}[OF - \text{fin}(2)]$
have $\text{prod-fun } f g m = (\sum l. f l * (\sum q. g q \text{ when } m = l + q))$
by (*simp add: prod-fun-def Sum-any-right-distrib*)
also have $\dots = (\sum l. \sum q. f l * g q \text{ when } m = l + q)$
by (*subst Sum-any-right-distrib*) (*auto intro!: Sum-any.cong fin'(2) simp: when-def*)
also {
fix $l q$ **assume** $lq: m = l + q$ **and** $nz: f l * g q \neq 0$
from *nz* **and** *zero* **have** $l \leq a \ q \leq b$ **by** (*auto intro: leI*)
hence $l + q \leq a + b$ **by** (*intro add-mono*)
also have $\dots < m$ **by** *fact*
finally have $l + q < m$.
}
hence $(\sum l. \sum q. f l * g q \text{ when } m = l + q) = (\sum l. \sum q. f l * g q \text{ when } \text{False})$
by (*intro Sum-any.cong refl*) (*auto simp: when-def*)
also have $\dots = 0$ **by** *simp*
finally show *?thesis* .
qed

2.4 Restricting a monomial to a subset of variables

lift-definition *restrictpm* $:: 'a \text{ set} \Rightarrow ('a \Rightarrow_0 'b :: \text{zero}) \Rightarrow ('a \Rightarrow_0 'b) \text{ is}$
 $\lambda A f x. \text{if } x \in A \text{ then } f x \text{ else } 0$
by (*erule finite-subset[rotated]*) *auto*

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

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]: $x \notin A \implies \text{lookup} (\text{restrictpm } A \ m) \ x = 0$
by (*simp-all add: lookup-restrictpm*)

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

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

lemma *restrictpm-id* [simp]: $\text{keys } m \subseteq X \implies \text{restrictpm } X \ m = m$
by *transfer (auto simp: fun-eq-iff)*

lemma *restrictpm-orthogonal* [simp]: $\text{keys } m \subseteq -X \implies \text{restrictpm } X \ m = 0$
by *transfer (auto simp: fun-eq-iff)*

lemma *restrictpm-add-disjoint*:
 $X \cap Y = \{\} \implies \text{restrictpm } X \ m + \text{restrictpm } Y \ m = \text{restrictpm } (X \cup Y) \ m$
by *transfer (auto simp: fun-eq-iff)*

lemma *restrictpm-add-complements*:
 $\text{restrictpm } X \ m + \text{restrictpm } (-X) \ m = m \ \text{restrictpm } (-X) \ m + \text{restrictpm } X \ m = m$
by (*subst restrictpm-add-disjoint; force*)+

2.5 Mapping over a polynomial

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

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

lemma *poly-mapping-map-conv-mapp*: $\text{Poly-Mapping.map } f = \text{Poly-Mapping.mapp} (\lambda-. \ f)$
by (*auto simp: Poly-Mapping.mapp-def Poly-Mapping.map-def map-fun-def o-def fun-eq-iff when-def in-keys-iff cong: if-cong*)

lemma *map-mpoly-conv-mapm-mpoly*: $\text{map-mpoly } f = \text{mapm-mpoly} (\lambda-. \ f)$
by *transfer' (auto simp: poly-mapping-map-conv-mapp)*

lemma *map-mpoly-comp*: $f \ 0 = 0 \implies \text{map-mpoly } f \ (\text{map-mpoly } g \ p) = \text{map-mpoly} (f \circ g) \ p$
by (*transfer', transfer'*) (*auto simp: when-def fun-eq-iff*)

lemma *mapp-mapp*:

$(\bigwedge x. f x 0 = 0) \implies \text{Poly-Mapping.mapp } f (\text{Poly-Mapping.mapp } g m) =$
 $\text{Poly-Mapping.mapp } (\lambda x y. f x (g x y)) m$
by *transfer'* (*auto simp: fun-eq-iff lookup-mapp in-keys-iff*)

lemma *mapm-mpoly-comp*:

$(\bigwedge x. f x 0 = 0) \implies \text{mapm-mpoly } f (\text{mapm-mpoly } g p) = \text{mapm-mpoly } (\lambda m c. f$
 $m (g m c)) p$
by *transfer'* (*simp add: mapp-mapp*)

lemma *coeff-map-mpoly*:

$\text{coeff } (\text{map-mpoly } f p) m = (\text{if } \text{coeff } p m = 0 \text{ then } 0 \text{ else } f (\text{coeff } p m))$
by (*transfer, transfer'*) *auto*

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

by (*subst coeff-map-mpoly*) *auto*

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

by (*transfer, transfer'*) (*auto simp: in-keys-iff*)

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

by (*subst coeff-mapm-mpoly*) *auto*

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

unfolding *vars-def* **by** (*transfer', transfer'*) (*auto simp: map-mpoly.rep-eq*)

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

by (*induction A rule: infinite-finite-induct*) *auto*

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

by (*auto simp add: Sum-any.expand-set intro!: sum.mono-neutral-right*)

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

by (*auto simp: Sum-any.expand-set*)

lemma *insertion-Prod-any*:

$\text{finite } \{x. g x \neq 1\} \implies \text{insertion } f (\text{Prod-any } g) = \text{Prod-any } (\lambda x. \text{insertion } f (g$
 $x))$

by (*auto simp: Prod-any.expand-set insertion-prod intro!: prod.mono-neutral-right*)

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 –

have $\text{insertion } g (\text{insertion } k p) =$

```

      (∑ x. insertion g (coeff p x) * insertion g (∏ i. k i ^ lookup x i))
unfolding insertion-altdef[of k p]
by (subst insertion-Sum-any)
      (auto intro: finite-subset[OF - finite-coeff-support[of p]] simp: insertion-mult)
also have ... = (∑ x. insertion g (coeff p x) * (∏ i. insertion g (k i) ^ lookup
x i))
proof (intro Sum-any.cong)
  fix x show insertion g (coeff p x) * insertion g (∏ i. k i ^ lookup x i) =
      insertion g (coeff p x) * (∏ i. insertion g (k i) ^ lookup x i)
  by (subst insertion-Prod-any)
      (auto simp: insertion-power intro!: finite-subset[OF - finite-lookup[of x]]
Nat.gr0I)
qed
also have ... = insertion (λx. insertion g (k x)) (map-mpoly (insertion g) p)
unfolding insertion-altdef[of - map-mpoly f p for f] by auto
finally show ?thesis .
qed

```

```

lemma insertion-substitute-linear:
  insertion (λi. c i * f i) p =
    insertion f (mapm-mpoly (λm d. Prod-any (λi. c i ^ lookup m i) * d) p)
unfolding insertion-altdef
proof (intro Sum-any.cong, goal-cases)
  case (1 m)
  have coeff (mapm-mpoly (λm. (*) (∏ i. c i ^ lookup m i)) p) m * (∏ i. f i ^
lookup m i) =
    MPoly-Type.coeff p m * ((∏ i. c i ^ lookup m i) * (∏ i. f i ^ lookup m i))
  by (simp add: mult-ac)
  also have (∏ i. c i ^ lookup m i) * (∏ i. f i ^ lookup m i) =
    (∏ i. (c i * f i) ^ lookup m i)
  by (subst Prod-any.distrib [symmetric])
      (auto simp: power-mult-distrib intro!: finite-subset[OF - finite-lookup[of m]]
Nat.gr0I)
  finally show ?case by simp
qed

```

```

lemma vars-mapm-mpoly-subset: vars (mapm-mpoly f p) ⊆ vars p
unfolding vars-def using keys-mapp-subset[of f] by (auto simp: mapm-mpoly.rep-eq)

```

```

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
  using assms by (intro mpoly-eqI) (auto simp: coeff-map-mpoly)

```

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

lemma $lead\text{-}monom\text{-}geI$ [intro]:
assumes $coeff\ p\ m \neq 0$
shows $m \leq lead\text{-}monom\ p$
using *assms* **by** (auto simp: $lead\text{-}monom\text{-}def\ coeff\text{-}def\ in\text{-}keys\text{-}iff$)

lemma $coeff\text{-}gt\text{-}lead\text{-}monom\text{-}zero$ [simp]:
assumes $m > lead\text{-}monom\ p$
shows $coeff\ p\ m = 0$
using $lead\text{-}monom\text{-}geI$ [of $p\ m$] *assms* **by** force

lemma $lead\text{-}monom\text{-}nonzero\text{-}eq$:
assumes $p \neq 0$
shows $lead\text{-}monom\ p = Max (keys (mapping\text{-}of\ p))$
using *assms* **by** transfer (simp add: $max\text{-}def$)

lemma $lead\text{-}monom\text{-}0$ [simp]: $lead\text{-}monom\ 0 = 0$
by (simp add: $lead\text{-}monom\text{-}def\ zero\text{-}mpoly.rep\text{-}eq$)

lemma $lead\text{-}monom\text{-}1$ [simp]: $lead\text{-}monom\ 1 = 0$
by (simp add: $lead\text{-}monom\text{-}def\ one\text{-}mpoly.rep\text{-}eq$)

lemma $lead\text{-}monom\text{-}Const$ [simp]: $lead\text{-}monom (Const\ c) = 0$
by (simp add: $lead\text{-}monom\text{-}def\ Const.rep\text{-}eq\ Const_0\text{-}def$)

lemma $lead\text{-}monom\text{-}uminus$ [simp]: $lead\text{-}monom (-p) = lead\text{-}monom\ p$
by (simp add: $lead\text{-}monom\text{-}def\ uminus\text{-}mpoly.rep\text{-}eq$)

lemma $keys\text{-}mult\text{-}const$ [simp]:
fixes $c :: 'a :: \{semiring\text{-}0, semiring\text{-}no\text{-}zero\text{-}divisors\}$
assumes $c \neq 0$
shows $keys (Poly\text{-}Mapping.map ((*)\ c)\ p) = keys\ p$
using *assms* **by** transfer auto

lemma $lead\text{-}monom\text{-}eq\text{-}0\text{-}iff$: $lead\text{-}monom\ p = 0 \iff vars\ p = \{\}$
unfolding $vars\text{-}def$ **by** transfer' (auto simp: $Max\text{-}eq\text{-}iff$)

lemma $lead\text{-}monom\text{-}monom$: $lead\text{-}monom (monom\ m\ c) = (if\ c = 0\ then\ 0\ else\ m)$
by (auto simp add: $lead\text{-}monom\text{-}def\ monom.rep\text{-}eq\ Const_0\text{-}def\ max\text{-}def$)

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

by (simp add: lead-monom-monom)

lemma lead-monom-numeral [simp]: lead-monom (numeral n) = 0
unfolding monom-numeral[symmetric] by (subst lead-monom-monom) auto

lemma lead-monom-add: lead-monom (p + q) ≤ max (lead-monom p) (lead-monom q)

proof transfer

fix p q :: (nat ⇒₀ nat) ⇒₀ 'a

show Max (insert 0 (keys (p + q))) ≤ max (Max (insert 0 (keys p))) (Max (insert 0 (keys q)))

proof (rule Max.boundedI)

fix m assume m: m ∈ insert 0 (keys (p + q))

thus m ≤ max (Max (insert 0 (keys p))) (Max (insert 0 (keys q)))

proof

assume m ∈ keys (p + q)

with keys-add[of p q] have m ∈ keys p ∨ m ∈ keys q

by (auto simp: in-keys-iff plus-poly-mapping.rep-eq)

thus ?thesis by (auto simp: le-max-iff-disj)

qed auto

qed auto

qed

lemma lead-monom-diff: lead-monom (p - q) ≤ max (lead-monom p) (lead-monom q)

proof transfer

fix p q :: (nat ⇒₀ nat) ⇒₀ 'a

show Max (insert 0 (keys (p - q))) ≤ max (Max (insert 0 (keys p))) (Max (insert 0 (keys q)))

proof (rule Max.boundedI)

fix m assume m: m ∈ insert 0 (keys (p - q))

thus m ≤ max (Max (insert 0 (keys p))) (Max (insert 0 (keys q)))

proof

assume m ∈ keys (p - q)

with keys-diff-subset[of p q] have m ∈ keys p ∨ m ∈ keys q by auto

thus ?thesis by (auto simp: le-max-iff-disj)

qed auto

qed auto

qed

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

lemma vars-empty-iff: vars p = {} ↔ p = Const (lead-coeff p)

proof

assume vars p = {}

hence [simp]: lead-monom p = 0

by (simp add: lead-monom-eq-0-iff)

have [simp]: mon ≠ 0 ↔ (mon > (0 :: nat ⇒₀ nat)) for mon

by (auto simp: order.strict-iff-order)
 thus $p = \text{Const } (\text{lead-coeff } p)$
 by (intro mpoly-eqI) (auto simp: mpoly-coeff-Const lead-coeff-def)
 next
 assume $p = \text{Const } (\text{lead-coeff } p)$
 also have $\text{vars } \dots = \{\}$ by simp
 finally show $\text{vars } p = \{\}$.
 qed

lemma *lead-coeff-0* [simp]: $\text{lead-coeff } 0 = 0$
 by (simp add: lead-coeff-def)

lemma *lead-coeff-1* [simp]: $\text{lead-coeff } 1 = 1$
 by (simp add: lead-coeff-def mpoly-coeff-1)

lemma *lead-coeff-Const* [simp]: $\text{lead-coeff } (\text{Const } c) = c$
 by (simp add: lead-coeff-def mpoly-coeff-Const)

lemma *lead-coeff-monom* [simp]: $\text{lead-coeff } (\text{monom } p \ c) = c$
 by (simp add: lead-coeff-def coeff-monom when-def lead-monom-monom)

lemma *lead-coeff-nonzero* [simp]: $p \neq 0 \implies \text{lead-coeff } p \neq 0$
unfolding *lead-coeff-def lead-monom-def*
 by (cases keys (mapping-of p) = {\}) (auto simp: coeff-def max-def)

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 –
from *assms* **have** $*$: $\text{keys } (\text{mapping-of } p) \neq \{\}$
by *auto*
from *assms* **have** *coeff* (*MPoly-Type.smult* $c \ p$) (*lead-monom* p) $\neq 0$
by (simp add: lead-coeff-def)
hence *smult-nz*: *MPoly-Type.smult* $c \ p \neq 0$ **by** (auto simp del: coeff-smult)
with *assms* **have** $**$: $\text{keys } (\text{mapping-of } (\text{smult } c \ p)) \neq \{\}$
by *simp*

have *Max* (*keys* (*mapping-of* (*smult* $c \ p$))) = *Max* (*keys* (*mapping-of* p))

proof (*safe intro!*: *antisym Max.coboundedI*)

have *lookup* (*mapping-of* p) (*Max* (*keys* (*mapping-of* p))) = *lead-coeff* p
using $*$ **by** (simp add: lead-coeff-def lead-monom-def max-def coeff-def)

with *assms* **show** *Max* (*keys* (*mapping-of* p)) \in *keys* (*mapping-of* (*smult* $c \ p$))
using $*$ **by** (auto simp: smult.rep-eq intro!: in-keys-mapI)

from *smult-nz* **have** *lead-coeff* (*smult* $c \ p$) $\neq 0$

by (*intro lead-coeff-nonzero*) *auto*

hence *coeff* p (*Max* (*keys* (*mapping-of* (*smult* $c \ p$)))) $\neq 0$

using *assms* $**$ **by** (auto simp: lead-coeff-def lead-monom-def max-def)

thus $Max (keys (mapping-of (smult c p))) \in keys (mapping-of p)$
by (*auto simp: smult.rep-eq coeff-def in-keys-iff*)
qed *auto*
with * ** **show** $lead-monom (smult c p) = lead-monom p$
by (*simp add: lead-monom-def max-def*)
thus $lead-coeff (smult c p) = c * lead-coeff p$
by (*simp add: lead-coeff-def*)
qed

lemma *lead-coeff-mult-aux*:
 $coeff (p * q) (lead-monom p + lead-monom q) = lead-coeff p * lead-coeff q$
proof (*cases p = 0 \vee q = 0*)
case *False*
define *a b* **where** $a = lead-monom p$ **and** $b = lead-monom q$
have $coeff (p * q) (a + b) = coeff p a * coeff q b$
unfolding *coeff-mpoly-times*
by (*rule prod-fun-max*) (*insert False, auto simp: a-def b-def*)
thus *?thesis* **by** (*simp add: a-def b-def lead-coeff-def*)
qed *auto*

lemma *lead-monom-mult-le*: $lead-monom (p * q) \leq lead-monom p + lead-monom q$
proof (*cases p * q = 0*)
case *False*
show *?thesis*
proof (*intro leI notI*)
assume $lead-monom p + lead-monom q < lead-monom (p * q)$
hence $lead-coeff (p * q) = 0$
unfolding *lead-coeff-def coeff-mpoly-times* **by** (*rule prod-fun-gt-max-eq-zero*)
auto
with *False* **show** *False* **by** *simp*
qed
qed *auto*

lemma *lead-monom-mult*:
assumes $lead-coeff p * lead-coeff q \neq 0$
shows $lead-monom (p * q) = lead-monom p + lead-monom q$
by (*intro antisym lead-monom-mult-le lead-monom-geI*)
(insert assms, auto simp: lead-coeff-mult-aux)

lemma *lead-coeff-mult*:
assumes $lead-coeff p * lead-coeff q \neq 0$
shows $lead-coeff (p * q) = lead-coeff p * lead-coeff q$
using *assms* **by** (*simp add: lead-monom-mult lead-coeff-mult-aux lead-coeff-def*)

lemma *keys-lead-monom-subset*: $keys (lead-monom p) \subseteq vars p$
proof (*cases p = 0*)
case *False*
hence $lead-coeff p \neq 0$ **by** *simp*

hence $\text{coeff } p (\text{lead-monom } p) \neq 0$ **unfolding** lead-coeff-def .
thus $?thesis$ **unfolding** vars-def **by** $\text{transfer}'$ ($\text{auto simp: max-def in-keys-iff}$)
qed auto

lemma

assumes $(\prod i \in A. \text{lead-coeff } (f i)) \neq 0$
shows $\text{lead-monom-prod: lead-monom } (\prod i \in A. f i) = (\sum i \in A. \text{lead-monom } (f i))$ (**is** $?th1$)
and $\text{lead-coeff-prod: lead-coeff } (\prod i \in A. f i) = (\prod i \in A. \text{lead-coeff } (f i))$ (**is** $?th2$)

proof –

have $?th1 \wedge ?th2$ **using** assms

proof ($\text{induction } A$ $\text{rule: infinite-finite-induct}$)

case ($\text{insert } x A$)

from $\text{insert have nz: lead-coeff } (f x) \neq 0 (\prod i \in A. \text{lead-coeff } (f i)) \neq 0$ **by** auto

note $IH = \text{insert.IH}[OF \text{this}(2)]$

from $\text{insert have nz': lead-coeff } (f x) * \text{lead-coeff } (\prod i \in A. f i) \neq 0$

by ($\text{subst } IH$) auto

from $\text{insert.premis insert.hyps nz nz' show ?case}$

by ($\text{auto simp: lead-monom-mult lead-coeff-mult } IH$)

qed auto

thus $?th1 ?th2$ **by** blast+

qed

lemma $\text{lead-monom-sum-le: } (\bigwedge x. x \in X \implies \text{lead-monom } (h x) \leq ub) \implies \text{lead-monom } (\text{sum } h X) \leq ub$

by ($\text{induction } X$ $\text{rule: infinite-finite-induct}$) ($\text{auto intro!: order.trans}[OF \text{lead-monom-add}]$)

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 $\text{inj-on } (\text{lead-monom } \circ h) X$ **and** $\text{finite } X$ **and** $X \neq \{\}$ **and** $\bigwedge x. x \in X \implies h x \neq 0$

defines $m \equiv \text{Max } ((\text{lead-monom } \circ h) ' X)$

shows $\text{lead-monom } (\sum x \in X. h x) = m$

proof (rule antisym)

show $\text{lead-monom } (\text{sum } h X) \leq m$ **unfolding** $m\text{-def}$ **using** assms

by ($\text{intro lead-monom-sum-le Max-ge finite-imageI}$) auto

next

from $\text{assms have } m \in (\text{lead-monom } \circ h) ' X$

unfolding $m\text{-def}$ **by** ($\text{intro Max-in finite-imageI}$) auto

then obtain x **where** $x: x \in X m = \text{lead-monom } (h x)$ **by** auto

have $\text{coeff } (\sum x \in X. h x) m = (\sum x \in X. \text{coeff } (h x) m)$

by simp

also have $\dots = (\sum x \in \{x\}. \text{coeff } (h x) m)$

proof ($\text{intro sum.mono-neutral-right ballI}$)

fix y **assume** $y: y \in X - \{x\}$

hence $(\text{lead-monom } \circ h) y \leq m$

using *assms* **unfolding** *m-def* **by** (*intro Max-ge finite-imageI*) *auto*
moreover have $(\text{lead-monom} \circ h) y \neq (\text{lead-monom} \circ h) x$
using $\langle x \in X \rangle y \text{ inj-onD}[OF \text{ assms}(1), \text{ of } x y]$ **by** *auto*
ultimately have $\text{lead-monom} (h y) < m$
using *x* **by** *auto*
thus $\text{coeff} (h y) m = 0$ **by** *simp*
qed (*insert x assms, auto*)
also have $\dots = \text{coeff} (h x) m$ **by** *simp*
also have $\dots = \text{lead-coeff} (h x)$ **using** *x* **by** (*simp add: lead-coeff-def*)
also have $\dots \neq 0$ **using** *assms x* **by** *auto*
finally show $\text{lead-monom} (\text{sum } h X) \geq m$ **by** (*intro lead-monom-geI*)
qed

lemma *lead-coeff-eq-0-iff* [*simp*]: $\text{lead-coeff } p = 0 \longleftrightarrow p = 0$
by (*cases p = 0*) *auto*

lemma
fixes *f* :: $- \Rightarrow 'a :: \text{semidom mpoly}$
assumes $\bigwedge i. i \in A \implies f i \neq 0$
shows *lead-monom-prod'* [*simp*]: $\text{lead-monom} (\prod_{i \in A}. f i) = (\sum_{i \in A}. \text{lead-monom} (f i))$ (*is ?th1*)
and *lead-coeff-prod'* [*simp*]: $\text{lead-coeff} (\prod_{i \in A}. f i) = (\prod_{i \in A}. \text{lead-coeff} (f i))$ (*is ?th2*)
proof –
from *assms* **have** $(\prod_{i \in A}. \text{lead-coeff} (f i)) \neq 0$
by (*cases finite A*) *auto*
thus *?th1 ?th2* **by** (*simp-all add: lead-monom-prod lead-coeff-prod*)
qed

lemma
fixes *p* :: $'a :: \text{comm-semiring-1 mpoly}$
assumes $\text{lead-coeff } p \neq 0$
shows *lead-monom-power*: $\text{lead-monom} (p \wedge n) = \text{of-nat } n * \text{lead-monom } p$
and *lead-coeff-power*: $\text{lead-coeff} (p \wedge n) = \text{lead-coeff } p \wedge n$
using *assms* *lead-monom-prod*[*of* $\lambda-. p \{..<n\}$] *lead-coeff-prod*[*of* $\lambda-. p \{..<n\}$]
by *simp-all*

lemma
fixes *p* :: $'a :: \text{semidom mpoly}$
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*]: $\text{lead-coeff} (p \wedge n) = \text{lead-coeff } p \wedge n$
using *assms* *lead-monom-prod'*[*of* $\{..<n\} \lambda-. p$] *lead-coeff-prod'*[*of* $\{..<n\} \lambda-. p$]
by *simp-all*

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* :: *nat set* \Rightarrow (*nat* \Rightarrow_0 *nat*) **is**
 $\lambda X x. \text{if finite } X \wedge x \in X \text{ then } 1 \text{ else } 0$
by *auto*

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

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*]:
 $i \notin X \Longrightarrow \text{Poly-Mapping.lookup } (\text{monom-of-set } X) \ i = 0$
by (*simp-all add: lookup-monom-of-set*)

lemma *keys-monom-of-set*: *keys* (*monom-of-set* X) = (*if finite* X *then* X *else* $\{\}$)
by *transfer auto*

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

lemma *monom-of-set-eq-iff* [*simp*]: *finite* $X \Longrightarrow \text{finite } Y \Longrightarrow \text{monom-of-set } X = \text{monom-of-set } Y \longleftrightarrow X = Y$
by *transfer (auto simp: fun-eq-iff)*

lemma *monom-of-set-empty* [*simp*]: *monom-of-set* $\{\} = 0$
by *transfer auto*

lemma *monom-of-set-eq-zero-iff* [*simp*]: *monom-of-set* $X = 0 \longleftrightarrow \text{infinite } X \vee X = \{\}$
by *transfer (auto simp: fun-eq-iff)*

lemma *zero-eq-monom-of-set-iff* [*simp*]: $0 = \text{monom-of-set } X \longleftrightarrow \text{infinite } X \vee X = \{\}$
by *transfer (auto simp: fun-eq-iff)*

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**
 $\lambda f p. \text{if bij } f \text{ then } p \circ f \text{ else } p$
proof –
fix $f :: 'a \Rightarrow 'a$ **and** $g :: 'a \Rightarrow 'b$
assume $*$: *finite* $\{x. g \ x \neq 0\}$

```

show finite {x. (if bij f then g ∘ f else g) x ≠ 0}
proof (cases bij f)
  case True
    with * have finite (f -' {x. g x ≠ 0})
      by (intro finite-vimageI) (auto dest: bij-is-inj)
    with True show ?thesis by auto
qed (use * in auto)
qed

lift-definition mpoly-map-vars :: (nat ⇒ nat) ⇒ 'a :: zero mpoly ⇒ 'a mpoly is
  λf p. permutep (permutep f) p .

lemma keys-permutep: bij f ⇒ keys (permutep f m) = f -' keys m
  by transfer auto

lemma permutep-id'' [simp]: permutep id = id
  by transfer' (auto simp: fun-eq-iff)

lemma permutep-id''' [simp]: permutep (λx. x) = id
  by transfer' (auto simp: fun-eq-iff)

lemma permutep-0 [simp]: permutep f 0 = 0
  by transfer auto

lemma permutep-single:
  bij f ⇒ permutep f (Poly-Mapping.single a b) = Poly-Mapping.single (inv-into
  UNIV f a) b
  by transfer (auto simp: fun-eq-iff when-def inv-f-f surj-f-inv-f bij-is-inj bij-is-surj)

lemma mpoly-map-vars-id [simp]: mpoly-map-vars id = id
  by transfer auto

lemma mpoly-map-vars-id' [simp]: mpoly-map-vars (λx. x) = id
  by transfer auto

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)
  by transfer auto

lemma inj-permutep [intro]: inj (permutep (f :: 'a ⇒ 'a) :: - ⇒ 'a ⇒0 'b :: zero)
  unfolding inj-def
proof (transfer, safe)
  fix f :: 'a ⇒ 'a and x y :: 'a ⇒ 'b
  assume eq: (if bij f then x ∘ f else x) = (if bij f then y ∘ f else y)
  show x = y
  proof (cases bij f)
    case True
      show ?thesis

```

```

proof
  fix  $t :: 'a$ 
  from  $\langle \text{bij } f \rangle$  obtain  $s$  where  $t = f s$ 
  by  $(\text{auto dest!: bij-is-surj})$ 
  with  $eq$  and  $\text{True}$  show  $x t = y t$ 
  by  $(\text{auto simp: fun-eq-iff})$ 
qed
qed  $(\text{use } eq \text{ in } \text{auto})$ 
qed

lemma  $\text{surj-permutep}$  [intro]:  $\text{surj } (\text{permutep } (f :: 'a \Rightarrow 'a) :: - \Rightarrow 'a \Rightarrow_0 'b :: \text{zero})$ 
  unfolding  $\text{surj-def}$ 
proof  $(\text{transfer, safe})$ 
  fix  $f :: 'a \Rightarrow 'a$  and  $y :: 'a \Rightarrow 'b$ 
  assume  $\text{fin: finite } \{t. y t \neq 0\}$ 
  show  $\exists x \in \{f. \text{finite } \{x. f x \neq 0\}\}. y = (\text{if } \text{bij } f \text{ then } x \circ f \text{ else } x)$ 
  proof  $(\text{cases } \text{bij } f)$ 
  case  $\text{True}$ 
  with  $\text{fin}$  have  $\text{finite } (\text{the-inv } f - ' \{t. y t \neq 0\})$ 
  by  $(\text{intro finite-vimageI})$   $(\text{auto simp: bij-is-inj bij-betw-the-inv-into})$ 
  moreover have  $y \circ \text{the-inv } f \circ f = y$ 
  using  $\text{True}$  by  $(\text{simp add: fun-eq-iff the-inv-f-f bij-is-inj})$ 
  ultimately show  $?thesis$  by  $(\text{intro } \text{bexI}[of - y \circ \text{the-inv } f])$   $(\text{auto simp: True})$ 
qed  $(\text{use } \text{fin} \text{ in } \text{auto})$ 
qed

lemma  $\text{bij-permutep}$  [intro]:  $\text{bij } (\text{permutep } f)$ 
  using  $\text{inj-permutep}[of f]$   $\text{surj-permutep}[of f]$  by  $(\text{simp add: bij-def})$ 

lemma  $\text{mpoly-map-vars-map-mpoly}$ :
   $\text{mpoly-map-vars } f (\text{map-mpoly } g p) = \text{map-mpoly } g (\text{mpoly-map-vars } f p)$ 
  by  $(\text{transfer}', \text{transfer}')$   $(\text{auto simp: fun-eq-iff})$ 

lemma  $\text{coeff-mpoly-map-vars}$ :
  fixes  $f :: \text{nat} \Rightarrow \text{nat}$  and  $p :: 'a :: \text{zero } \text{mpoly}$ 
  assumes  $\text{bij } f$ 
  shows  $\text{MPoly-Type.coeff } (\text{mpoly-map-vars } f p) \text{ mon} =$ 
   $\text{MPoly-Type.coeff } p (\text{permutep } f \text{ mon})$ 
  using  $\text{assms}$  by  $\text{transfer}'$   $(\text{simp add: lookup-permutep bij-permutep})$ 

lemma  $\text{permutep-monom-of-set}$ :
  assumes  $\text{bij } f$ 
  shows  $\text{permutep } f (\text{monom-of-set } A) = \text{monom-of-set } (f - ' A)$ 
  using  $\text{assms}$  by  $\text{transfer}$   $(\text{auto simp: fun-eq-iff bij-is-inj finite-vimage-iff})$ 

lemma  $\text{permutep-comp}$ :  $\text{bij } f \Longrightarrow \text{bij } g \Longrightarrow \text{permutep } (f \circ g) = \text{permutep } g \circ \text{permutep } f$ 
  by  $\text{transfer}'$   $(\text{auto simp: fun-eq-iff bij-comp})$ 

```

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

by *transfer (auto simp: fun-eq-iff bij-comp)*

lemma *mpoly-map-vars-comp*:

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

by *transfer (auto simp: bij-permutep permutep-comp)*

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

by *transfer auto*

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

by *transfer auto*

lemma *inv-permutep* [*simp*]:

fixes $f :: 'a \Rightarrow 'a$

assumes *bij f*

shows $\text{inv-into UNIV } (\text{permutep } f) = \text{permutep } (\text{inv-into UNIV } f)$

proof

fix $m :: 'a \Rightarrow_0 'b$

show $\text{inv-into UNIV } (\text{permutep } f) m = \text{permutep } (\text{inv-into UNIV } f) m$

using *permutep-comp'[of inv-into UNIV f f m] assms inj-iff[of f]*

by (*intro inv-f-eq*) (*auto simp: bij-imp-bij-inv bij-is-inj*)

qed

lemma *mpoly-map-vars-monom*:

$\text{bij } f \implies \text{mpoly-map-vars } f (\text{monom } m c) = \text{monom } (\text{permutep } (\text{inv-into UNIV } f) m) c$

by *transfer' (simp add: permutep-single bij-permutep)*

lemma *vars-mpoly-map-vars*:

fixes $f :: \text{nat} \Rightarrow \text{nat}$ **and** $p :: 'a :: \text{zero mpoly}$

assumes *bij f*

shows $\text{vars } (\text{mpoly-map-vars } f p) = f \text{ ' vars } p$

using *assms unfolding vars-def*

proof *transfer'*

fix $f :: \text{nat} \Rightarrow \text{nat}$ **and** $p :: (\text{nat} \Rightarrow_0 \text{nat}) \Rightarrow_0 'a$

assume $f: \text{bij } f$

have $\text{eq}: f (\text{inv-into UNIV } f x) = x$ **for** x

using f **by** (*subst surj-f-inv-f[of f]*) (*auto simp: bij-is-surj*)

show $\bigcup (\text{keys } \text{' keys } (\text{permutep } (\text{permutep } f) p)) = f \text{ ' } \bigcup (\text{keys } \text{' keys } p)$

proof *safe*

fix $m x$ **assume** $mx: m \in \text{keys } (\text{permutep } (\text{permutep } f) p) \ x \in \text{keys } m$

from mx **have** $\text{permutep } f m \in \text{keys } p$

by (*auto simp: keys-permutep bij-permutep f*)

with mx **have** $f (\text{inv-into UNIV } f x) \in f \text{ ' } (\bigcup_{m \in \text{keys } p} \text{keys } m)$

by (*intro imageI*) (*auto intro!: bexI[of - permutep f m] simp: keys-permutep f*)

eq)
 with eq show $x \in f^{-1} (\bigcup_{m \in \text{keys } p} \text{keys } m)$ by simp
 next
 fix m x assume mx: $m \in \text{keys } p \ x \in \text{keys } m$
 from mx have permuted id $m \in \text{keys } p$ by simp
 also have $\text{id} = \text{inv-into UNIV } f \circ f$ using f by (intro ext) (auto simp: bij-is-inj
 inv-f-f)
 also have permuted ... $m = \text{permuted } f (\text{permuted } (\text{inv-into UNIV } f) m)$
 by (simp add: permuted-comp f bij-imp-bij-inv)
 finally have **: $\text{permuted } f (\text{permuted } (\text{inv-into UNIV } f) m) \in \text{keys } p$.
 moreover from f mx have $f x \in \text{keys } (\text{permuted } (\text{inv-into UNIV } f) m)$
 by (auto simp: keys-permuted bij-imp-bij-inv inv-f-f bij-is-inj)
 ultimately show $f x \in \bigcup (\text{keys } ^\text{' } \text{keys } (\text{permuted } (\text{permuted } f) p))$ using f
 by (auto simp: keys-permuted bij-permuted)
 qed
 qed

lemma permuted-eq-monom-of-set-iff [simp]:
 assumes bij f
 shows $\text{permuted } f \text{ mon} = \text{monom-of-set } A \iff \text{mon} = \text{monom-of-set } (f^{-1} A)$
 proof
 assume eq: $\text{permuted } f \text{ mon} = \text{monom-of-set } A$
 have $\text{permuted } (\text{inv-into UNIV } f) (\text{permuted } f \text{ mon}) = \text{monom-of-set } (\text{inv-into UNIV } f ^{-1} A)$
 using assms by (simp add: eq bij-imp-bij-inv assms permuted-monom-of-set)
 also have $\text{inv-into UNIV } f ^{-1} A = f^{-1} A$
 using assms by (force simp: bij-is-surj image-iff inv-f-f bij-is-inj surj-f-inv-f)
 also have $\text{permuted } (\text{inv-into UNIV } f) (\text{permuted } f \text{ mon}) = \text{permuted } (f \circ \text{inv-into UNIV } f) \text{ mon}$
 using assms by (simp add: permuted-comp bij-imp-bij-inv)
 also have $f \circ \text{inv-into UNIV } f = \text{id}$
 by (subst surj-iff [symmetric]) (insert assms, auto simp: bij-is-surj)
 finally show $\text{mon} = \text{monom-of-set } (f^{-1} A)$ by simp
 qed (insert assms, auto simp: permuted-monom-of-set inj-vimage-image-eq bij-is-inj)

lemma permuted-monom-of-set-permutes [simp]:
 assumes π permutes A
 shows $\text{permuted } \pi (\text{monom-of-set } A) = \text{monom-of-set } A$
 using assms
 by transfer (auto simp: if-splits fun-eq-iff permutes-in-image)

lemma mpoly-map-vars-0 [simp]: $\text{mpoly-map-vars } f 0 = 0$
 by (transfer, transfer') (simp add: o-def)

lemma permuted-eq-0-iff [simp]: $\text{permuted } f m = 0 \iff m = 0$
 proof transfer
 fix f :: 'a \Rightarrow 'a and m :: 'a \Rightarrow 'b assume finite {x. m x \neq 0}
 show ((if bij f then $m \circ f$ else m) = ($\lambda k. 0$)) = (m = ($\lambda k. 0$))
 proof (cases bij f)

case *True*
hence $(\forall x. m (f x) = 0) \longleftrightarrow (\forall x. m x = 0)$
using *bij-iff[of f]* **by** *metis*
with *True* **show** *?thesis* **by** (*auto simp: fun-eq-iff*)
qed (*auto simp: fun-eq-iff*)
qed

lemma *mpoly-map-vars-1 [simp]*: *mpoly-map-vars f 1 = 1*
by (*transfer, transfer'*) (*auto simp: o-def fun-eq-iff when-def*)

lemma *permutep-Const₀ [simp]*: $(\bigwedge x. f x = 0 \longleftrightarrow x = 0) \implies \text{permutep } f (\text{Const}_0 c) = \text{Const}_0 c$
unfolding *Const₀-def* **by** *transfer'* (*auto simp: when-def fun-eq-iff*)

lemma *permutep-add [simp]*: $\text{permutep } f (m1 + m2) = \text{permutep } f m1 + \text{permutep } f m2$
unfolding *Const₀-def* **by** *transfer'* (*auto simp: when-def fun-eq-iff*)

lemma *permutep-diff [simp]*: $\text{permutep } f (m1 - m2) = \text{permutep } f m1 - \text{permutep } f m2$
unfolding *Const₀-def* **by** *transfer'* (*auto simp: when-def fun-eq-iff*)

lemma *permutep-uminus [simp]*: $\text{permutep } f (-m) = -\text{permutep } f m$
unfolding *Const₀-def* **by** *transfer'* (*auto simp: when-def fun-eq-iff*)

lemma *permutep-mult [simp]*:
 $(\bigwedge x y. f (x + y) = f x + f y) \implies \text{permutep } f (m1 * m2) = \text{permutep } f m1 * \text{permutep } f m2$
unfolding *Const₀-def* **by** *transfer'* (*auto simp: when-def fun-eq-iff prod-fun-compose-bij*)

lemma *mpoly-map-vars-Const [simp]*: $\text{mpoly-map-vars } f (\text{Const } c) = \text{Const } c$
by *transfer* (*auto simp: o-def fun-eq-iff when-def*)

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

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

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

lemma *permutep-smult*:
 $\text{permutep } (\text{permutep } f) (\text{Poly-Mapping.map } ((*) c) p) = \text{Poly-Mapping.map } ((*) c) (\text{permutep } (\text{permutep } f) p)$
by *transfer'* (*auto split: if-splits simp: fun-eq-iff*)

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

by *transfer (simp add: permutep-smult)*

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

by *transfer simp*

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

by (*induction A rule: infinite-finite-induct*) *auto*

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

by (*induction A rule: infinite-finite-induct*) *auto*

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

by *transfer auto*

lemma *permutep-eq-iff* [*simp*]: $permutep\ f\ p = permutep\ f\ q \longleftrightarrow p = q$

by *transfer (auto simp: comp-bij-eq-iff)*

lemma *mpoly-map-vars-Sum-any* [*simp*]:

$mpoly\text{-map-}vars\ f\ (Sum\text{-any}\ g) = Sum\text{-any}\ (\lambda x.\ mpoly\text{-map-}vars\ f\ (g\ x))$

by (*simp add: Sum-any.expand-set*)

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

by (*induction n*) *auto*

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

assumes *bij f*

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

$monom\ (Poly\text{-Mapping}\.single\ (f\ i)\ n)\ c$

using *assms* **by** (*simp add: mpoly-map-vars-monom permutep-single bij-imp-bij-inv inv-inv-eq*)

lemma *insertion-mpoly-map-vars*:

assumes *bij f*

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

proof –

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

$(\sum\ m.\ coeff\ p\ (permutep\ f\ m) * (\prod\ i.\ g\ i \wedge lookup\ m\ i))$

using *assms* **by** (*simp add: insertion-altdef coeff-mpoly-map-vars*)

also have $\dots = Sum\text{-any}\ (\lambda m.\ coeff\ p\ (permutep\ f\ m) *$

$Prod\text{-any}\ (\lambda i.\ g\ (f\ i) \wedge lookup\ m\ (f\ i)))$

by (*intro Sum-any.cong arg-cong[where ?f = $\lambda y.\ x * y$ for x]*

Prod-any.reindex-cong[OF assms]) (*auto simp: o-def*)

```

also have ... = Sum-any ( $\lambda m. \text{coeff } p \ m * (\prod i. g \ (f \ i) \ ^ \ \text{lookup } m \ i)$ )
  by (intro Sum-any.reindex-cong [OF bij-permutep[of f], symmetric])
    (auto simp: o-def lookup-permutep assms)
also have ... = insertion ( $g \circ f$ ) p
  by (simp add: insertion-altdef)
finally show ?thesis .
qed

lemma permutep-cong:
  assumes f permutes ( $-keys \ p$ ) g permutes ( $-keys \ p$ )  $p = q$ 
  shows permutep f p = permutep g q
proof (intro poly-mapping-eqI)
  fix k :: 'a
  show lookup (permutep f p) k = lookup (permutep g q) k
  proof (cases k ∈ keys p)
    case False
    with assms have  $f \ k \notin keys \ p \ g \ k \notin keys \ p$ 
    using permutes-in-image[of - keys p k] by auto
    thus ?thesis using assms by (auto simp: lookup-permutep permutes-bij in-keys-iff)
  qed (insert assms, auto simp: lookup-permutep permutes-bij permutes-not-in)
qed

```

```

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 ( $q :: 'a :: zero \ mpoly$ )
proof (intro mpoly-eqI)
  fix mon ::  $nat \Rightarrow_0 \ nat$ 
  show coeff (mpoly-map-vars f p) mon = coeff (mpoly-map-vars g q) mon
  proof (cases keys mon ⊆ vars p)
    case True
    with assms have permutep f mon = permutep g mon
    by (intro permutep-cong assms(1,2)[THEN permutes-subset]) auto
    thus ?thesis using assms by (simp add: coeff-mpoly-map-vars permutes-bij)
  next
    case False
    hence  $\neg(keys \ mon \subseteq f \ 'vars \ q) \ \neg(keys \ mon \subseteq g \ 'vars \ q)$ 
    using assms by (auto simp: subset-iff permutes-not-in)
    thus ?thesis using assms
    by (subst (1 2) coeff-notin-vars)
    (auto simp: coeff-notin-vars vars-mpoly-map-vars permutes-bij)
  qed
qed

```

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* = $(\forall \pi. \pi \text{ permutes } A \longrightarrow \text{mpoly-map-vars } \pi \ p = p)$

lemma *symmetric-mpoly-empty* [*simp, intro*]: *symmetric-mpoly* $\{\}$ *p*
by (*simp add: symmetric-mpoly-def*)

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*
unfolding *symmetric-mpoly-def*
proof *safe*
fix π **assume** π : π *permutes A*
with *assms* **have** π $x = x$ **if** $x \in \text{vars } p$ **for** x
using *that permutes-not-in[of π A x]* **by** *auto*
from *assms* **have** *mpoly-map-vars* π $p = \text{mpoly-map-vars id } p$
by (*intro mpoly-map-vars-cong permutes-subset[OF π] permutes-id*) *auto*
also **have** $\dots = p$ **by** *simp*
finally **show** *mpoly-map-vars* π $p = p$.
qed

lemma *symmetric-mpoly-monom* [*intro*]:
assumes *keys m* $\cap A = \{\}$
shows *symmetric-mpoly A (monom m c)*
using *assms vars-monom-subset[of m c]* **by** (*intro symmetric-mpoly-orthogonal*)
auto

lemma *symmetric-mpoly-subset*:
assumes *symmetric-mpoly A p B* $\subseteq A$
shows *symmetric-mpoly B p*
unfolding *symmetric-mpoly-def*
proof *safe*
fix π **assume** π *permutes B*
with *assms* **have** π *permutes A* **using** *permutes-subset* **by** *blast*
with *assms* **show** *mpoly-map-vars* π $p = p$
by (*auto simp: symmetric-mpoly-def*)
qed

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* $\cap A = \{\}$ $\vee A \subseteq \text{vars } p$
proof (*rule ccontr*)
assume $\neg(\text{vars } p \cap A = \{\} \vee A \subseteq \text{vars } p)$
then **obtain** x y **where** $x \in \text{vars } p \cap A$ $y \in A - \text{vars } p$ **by** *auto*
define π **where** $\pi = \text{Fun.swap } x$ y *id*
from xy **have** π : π *permutes A*
unfolding π -*def* **by** (*intro permutes-swap-id*) *auto*
from xy **have** $y \in \pi$ ‘ *vars p* **by** (*auto simp: π -def Fun.swap-def*)

also from π have π ‘ vars $p = \text{vars } (\text{mpoly-map-vars } \pi p)$
by (auto simp: vars-mpoly-map-vars permutes-bij)
also have $\text{mpoly-map-vars } \pi p = p$
using assms π by (simp add: symmetric-mpoly-def)
finally show False using xy by auto
qed

Symmetric polynomials are closed under ring operations.

lemma symmetric-mpoly-add [intro]:
 $\text{symmetric-mpoly } A p \implies \text{symmetric-mpoly } A q \implies \text{symmetric-mpoly } A (p + q)$
unfolding symmetric-mpoly-def by simp

lemma symmetric-mpoly-diff [intro]:
 $\text{symmetric-mpoly } A p \implies \text{symmetric-mpoly } A q \implies \text{symmetric-mpoly } A (p - q)$
unfolding symmetric-mpoly-def by simp

lemma symmetric-mpoly-uminus [intro]: $\text{symmetric-mpoly } A p \implies \text{symmetric-mpoly } A (-p)$
unfolding symmetric-mpoly-def by simp

lemma symmetric-mpoly-uminus-iff [simp]: $\text{symmetric-mpoly } A (-p) \longleftrightarrow \text{symmetric-mpoly } A p$
unfolding symmetric-mpoly-def by simp

lemma symmetric-mpoly-smult [intro]: $\text{symmetric-mpoly } A p \implies \text{symmetric-mpoly } A (smult c p)$
unfolding symmetric-mpoly-def by simp

lemma symmetric-mpoly-mult [intro]:
 $\text{symmetric-mpoly } A p \implies \text{symmetric-mpoly } A q \implies \text{symmetric-mpoly } A (p * q)$
unfolding symmetric-mpoly-def by simp

lemma symmetric-mpoly-0 [simp, intro]: $\text{symmetric-mpoly } A 0$
and symmetric-mpoly-1 [simp, intro]: $\text{symmetric-mpoly } A 1$
and symmetric-mpoly-Const [simp, intro]: $\text{symmetric-mpoly } A (\text{Const } c)$
by (simp-all add: symmetric-mpoly-def)

lemma symmetric-mpoly-power [intro]:
 $\text{symmetric-mpoly } A p \implies \text{symmetric-mpoly } A (p \wedge n)$
by (induction n) (auto intro!: symmetric-mpoly-mult)

lemma symmetric-mpoly-sum [intro]:
 $(\bigwedge i. i \in B \implies \text{symmetric-mpoly } A (f i)) \implies \text{symmetric-mpoly } A (\text{sum } f B)$
by (induction B rule: infinite-finite-induct) (auto intro!: symmetric-mpoly-add)

lemma symmetric-mpoly-prod [intro]:
 $(\bigwedge i. i \in B \implies \text{symmetric-mpoly } A (f i)) \implies \text{symmetric-mpoly } A (\text{prod } f B)$
by (induction B rule: infinite-finite-induct) (auto intro!: symmetric-mpoly-mult)

An symmetric sum or product over polynomials yields a symmetric polyno-

mial:

lemma *symmetric-mpoly-symmetric-sum:*

assumes g permutes X

assumes $\bigwedge x \pi. x \in X \implies \pi$ permutes $A \implies \text{mpoly-map-vars } \pi (f x) = f (g x)$

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

unfolding *symmetric-mpoly-def*

proof *safe*

fix π **assume** $\pi: \pi$ permutes A

have $\text{mpoly-map-vars } \pi (\text{sum } f X) = (\sum_{x \in X}. \text{mpoly-map-vars } \pi (f x))$

by *simp*

also have $\dots = (\sum_{x \in X}. f (g x))$

by (*intro sum.cong assms π refl*)

also have $\dots = (\sum_{x \in g'X}. f x)$

using *assms* **by** (*subst sum.reindex*) (*auto simp: permutes-inj-on*)

also have $g' X = X$

using *assms* **by** (*simp add: permutes-image*)

finally show $\text{mpoly-map-vars } \pi (\text{sum } f X) = \text{sum } f X$.

qed

lemma *symmetric-mpoly-symmetric-prod:*

assumes g permutes X

assumes $\bigwedge x \pi. x \in X \implies \pi$ permutes $A \implies \text{mpoly-map-vars } \pi (f x) = f (g x)$

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

unfolding *symmetric-mpoly-def*

proof *safe*

fix π **assume** $\pi: \pi$ permutes A

have $\text{mpoly-map-vars } \pi (\text{prod } f X) = (\prod_{x \in X}. \text{mpoly-map-vars } \pi (f x))$

by *simp*

also have $\dots = (\prod_{x \in X}. f (g x))$

by (*intro prod.cong assms π refl*)

also have $\dots = (\prod_{x \in g'X}. f x)$

using *assms* **by** (*subst prod.reindex*) (*auto simp: permutes-inj-on*)

also have $g' X = X$

using *assms* **by** (*simp add: permutes-image*)

finally show $\text{mpoly-map-vars } \pi (\text{prod } f X) = \text{prod } f X$.

qed

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 (*cases* $p = 0$)

case [*simp*]: *False*

show *?thesis*

proof (*intro leI notI*)

```

assume less: lookup m i < lookup m j
define  $\pi$  where  $\pi = \text{Fun.swap } i \ j \ \text{id}$ 
from assms have  $\pi$ :  $\pi$  permutes A
  unfolding  $\pi$ -def by (intro permutes-swap-id) auto
have [simp]:  $\pi \circ \pi = \text{id}$   $\pi \ i = j$   $\pi \ j = i$   $\wedge k. k \neq i \implies k \neq j \implies \pi \ k = k$ 
  by (auto simp: \pi-def Fun.swap-def fun-eq-iff)

have  $0 \neq \text{lead-coeff } p$  by simp
also have  $\text{lead-coeff } p = \text{MPoly-Type.coeff } (\text{mpoly-map-vars } \pi \ p)$  (permutep  $\pi$ 
m)
  using  $\pi$  by (simp add: lead-coeff-def m-def coeff-mpoly-map-vars
permutes-bij permutep-comp' [symmetric])
also have mpoly-map-vars  $\pi \ p = p$ 
  using  $\pi$  assms by (simp add: symmetric-mpoly-def)
finally have permutep  $\pi \ m \leq m$  by (auto simp: m-def)

moreover have lookup m i < lookup (permutep \pi m) i
  and ( $\forall k < i. \text{lookup } m \ k = \text{lookup } (\text{permutep } \pi \ m) \ k$ )
  using assms  $\pi$  less by (auto simp: lookup-permutep permutes-bij)
hence  $m < \text{permutep } \pi \ m$ 
  by (auto simp: less-poly-mapping-def less-fun-def)
ultimately show False by simp
qed
qed (auto simp: m-def)

```

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

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

proof –

fix $k :: \text{nat}$ **and** $X :: \text{nat set}$

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

($0 :: 'a$)} (**is** *finite* ? A)

proof (*cases finite X*)

case *True*

have ? $A \subseteq \text{monom-of-set } \text{' Pow } X$ **by** *auto*

moreover from *True* **have** *finite (monom-of-set ' Pow X)* **by** *simp*

ultimately show ?*thesis* **by** (*rule finite-subset*)

qed *auto*

qed

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)

by transfer' simp

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

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

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

using assms by (auto simp: lookup-sym-mpoly-aux)

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

by transfer' (auto split: if-splits simp: keys-monom-of-set)

lift-definition sym-mpoly :: nat set \Rightarrow nat \Rightarrow 'a :: {zero-neq-one} mpoly is
sym-mpoly-aux .

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

using keys-sym-mpoly-aux by (auto simp: vars-def sym-mpoly.rep-eq)

lemma coeff-sym-mpoly:

$\text{MPoly-Type.coeff } (\text{sym-mpoly } X k) \text{ mon} =$

(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)

by transfer' (simp add: lookup-sym-mpoly-aux)

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

by (transfer, transfer) auto

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 (cases finite A)

case False

hence *: infinite $\{X. X \subseteq A \wedge \text{infinite } X\}$

by (rule infinite-infinite-subsets)

have infinite $\{X. X \subseteq A \wedge \text{card } X = 0\}$

by (rule infinite-super[OF - *]) auto

moreover have **: infinite $\{X. X \subseteq A \wedge \text{finite } X \wedge \text{card } X = k\}$ if $k \neq 0$

using that infinite-card-subsets[of A k] False by auto

have infinite $\{X. X \subseteq A \wedge \text{card } X = k\}$ if $k \neq 0$

by (rule infinite-super[OF - **[OF that]]) auto

ultimately show ?thesis using False

by (cases $k = 0$) (simp-all add: sym-mpoly-infinite)

next

case True

show ?thesis

proof (intro mpoly-eqI, goal-cases)

case (1 m)

show ?case

proof (cases $\exists X. X \subseteq A \wedge \text{card } X = k \wedge m = \text{monom-of-set } X$)

case False


```

thus ?thesis by (auto simp: coeff-sym-mpoly coeff-sum coeff-monom)
next
case True
then obtain X where X:  $X \subseteq A$   $\text{card } X = k$   $m = \text{monom-of-set } X$ 
by blast
have coeff ( $\sum X \mid X \subseteq A \wedge \text{card } X = k.$ 
  monom (monom-of-set X) 1)  $m = (\sum X \in \{X\}. 1)$  unfolding coeff-sum
proof (intro sum.mono-neutral-cong-right ballI)
fix Y assume Y:  $Y \in \{X. X \subseteq A \wedge \text{card } X = k\} - \{X\}$ 
hence  $X = Y$  if monom-of-set X = monom-of-set Y
using that finite-subset[OF X(1)] finite-subset[of Y A] ⟨finite A⟩ by auto
thus coeff (monom (monom-of-set Y) 1)  $m = 0$ 
using X Y by (auto simp: coeff-monom when-def )
qed (insert X ⟨finite A⟩, auto simp: coeff-monom)
thus ?thesis using ⟨finite A⟩ by (auto simp: coeff-sym-mpoly coeff-sum
coeff-monom)
qed
qed
qed

```

```

lemma coeff-sym-mpoly-monom-of-set [simp]:
assumes finite X  $Y \subseteq X$   $\text{card } Y = k$ 
shows MPoly-Type.coeff (sym-mpoly X k) (monom-of-set Y) = 1
using assms by (auto simp: coeff-sym-mpoly)

```

```

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

```

```

proof –
consider finite X  $k = 0 \mid$  finite X  $k \neq 0 \mid$  infinite X by blast
thus ?thesis
proof cases
assume finite X  $k = 0$ 
hence coeff (sym-mpoly X k) (monom-of-set {}) = 1
by (subst coeff-sym-mpoly-monom-of-set) auto
thus ?thesis unfolding monom-of-set-empty using ⟨finite X⟩ ⟨ $k = 0$ ⟩ by simp
next
assume finite X  $k \neq 0$ 
hence  $\neg(\exists Y. \text{finite } Y \wedge Y \subseteq X \wedge \text{card } Y = k \wedge \text{monom-of-set } Y = 0)$ 
by auto
thus ?thesis using ⟨ $k \neq 0$ ⟩
by (auto simp: coeff-sym-mpoly)
next
assume infinite X
thus ?thesis by (simp add: coeff-sym-mpoly)
qed
qed

```

```

lemma symmetric-sym-mpoly [intro]:
assumes  $A \subseteq B$ 

```

shows *symmetric-mpoly A (sym-mpoly B k :: 'a :: zero-neq-one mpoly)*
unfolding *symmetric-mpoly-def*
proof (*safe intro!: mpoly-eqI*)
fix π **and** $mon :: nat \Rightarrow_0 nat$ **assume** $\pi: \pi$ *permutes A*
from π **have** π' : π *permutes B* **by** (*rule permutes-subset*) *fact*
from π **have** *MPoly-Type.coeff (mpoly-map-vars π (sym-mpoly B k :: 'a mpoly))*
 $mon =$
 $MPoly-Type.coeff (sym-mpoly B k :: 'a mpoly) (permutep \pi mon)$
by (*simp add: coeff-mpoly-map-vars permutes-bij*)
also have $\dots = 1 \longleftrightarrow MPoly-Type.coeff (sym-mpoly B k :: 'a mpoly) mon = 1$
(is ?lhs = 1 \longleftrightarrow ?rhs = 1)
proof
assume $?rhs = 1$
then obtain Y **where** *finite B and $Y: Y \subseteq B$ card $Y = k$ $mon = monom-of-set$*
 Y
by (*auto simp: coeff-sym-mpoly split: if-splits*)
with π' **have** $\pi - ' Y \subseteq B$ *card $(\pi - ' Y) = k$ $permutep \pi mon = monom-of-set$*
 $(\pi - ' Y)$
by (*auto simp: permutes-in-image card-vimage-inj permutep-monom-of-set*
permutes-bij permutes-inj permutes-surj)
thus $?lhs = 1$ **using** $\langle finite B \rangle$ **by** (*auto simp: coeff-sym-mpoly*)
next
assume $?lhs = 1$
then obtain Y **where** *finite B and $Y: Y \subseteq B$ card $Y = k$ $permutep \pi mon$*
 $= monom-of-set Y$
by (*auto simp: coeff-sym-mpoly split: if-splits*)
from $Y(1)$ **have** *inj-on πY using inj-on-subset[of $\pi UNIV Y$] π'*
by (*auto simp: permutes-inj*)
with $Y \pi'$ **have** $\pi - ' Y \subseteq B$ *card $(\pi - ' Y) = k$ $mon = monom-of-set$ $(\pi - ' Y)$*
by (*auto simp: permutes-in-image card-image permutep-monom-of-set*
permutes-bij permutes-inj permutes-surj)
thus $?rhs = 1$ **using** $\langle finite B \rangle$ **by** (*auto simp: coeff-sym-mpoly*)
qed
hence $?lhs = ?rhs$
by (*auto simp: coeff-sym-mpoly split: if-splits*)
finally show *MPoly-Type.coeff (mpoly-map-vars π (sym-mpoly B k :: 'a mpoly))*
 $mon =$
 $MPoly-Type.coeff (sym-mpoly B k :: 'a mpoly) mon .$
qed

lemma *insertion-sym-mpoly:*

assumes *finite X*
shows *insertion f (sym-mpoly X k) = $(\sum Y \mid Y \subseteq X \wedge card Y = k. prod f Y)$*
using *assms*
proof (*transfer, transfer*)
fix $f :: nat \Rightarrow 'a$ **and** $k :: nat$ **and** $X :: nat set$
assume $X: finite X$
have *insertion-fun f $(\lambda mon.$*

if finite $X \wedge (\exists Y \subseteq X. \text{card } Y = k \wedge \text{mon} = \text{monom-of-set } Y)$ then 1
 else 0) =
 ($\sum m. (\prod v. f v \wedge \text{poly-mapping.lookup } m v)$ when $(\exists Y \subseteq X. \text{card } Y = k \wedge$
 $m = \text{monom-of-set } Y)$)
 by (auto simp add: insertion-fun-def X when-def intro!: Sum-any.cong)
 also have ... = ($\sum m \mid \exists Y \in \text{Pow } X. \text{card } Y = k \wedge m = \text{monom-of-set } Y. (\prod v.$
 $f v \wedge \text{poly-mapping.lookup } m v)$ when $(\exists Y \subseteq X. \text{card } Y = k \wedge m = \text{monom-of-set}$
 $Y)$)
 by (rule Sum-any.expand-superset) (use X in auto)
 also have ... = ($\sum m \mid \exists Y \in \text{Pow } X. \text{card } Y = k \wedge m = \text{monom-of-set } Y.$
 $(\prod v. f v \wedge \text{poly-mapping.lookup } m v)$)
 by (intro sum.cong) (auto simp: when-def)
 also have ... = ($\sum Y \mid Y \subseteq X \wedge \text{card } Y = k. (\prod v. f v \wedge \text{poly-mapping.lookup}$
 $(\text{monom-of-set } Y) v)$)
 by (rule sum.reindex-bij-witness[of - monom-of-set keys]) (auto simp: finite-subset[OF
 - X])
 also have ... = ($\sum Y \mid Y \subseteq X \wedge \text{card } Y = k. \prod v \in Y. f v$)
 proof (intro sum.cong when-cong refl, goal-cases)
 case (1 Y)
 hence finite Y by (auto dest: finite-subset[OF - X])
 with 1 have $(\prod v. f v \wedge \text{poly-mapping.lookup } (\text{monom-of-set } Y) v) =$
 $(\prod v::\text{nat. if } v \in Y \text{ then } f v \text{ else } 1)$
 by (intro Prod-any.cong) (auto simp: lookup-monom-of-set)
 also have ... = $(\prod v \in Y. f v)$
 by (rule Prod-any.conditionalize [symmetric]) fact+
 finally show ?case .
 qed
 finally show insertion-fun f
 ($\lambda \text{mon. if finite } X \wedge (\exists Y \subseteq X. \text{card } Y = k \wedge \text{mon} = \text{monom-of-set}$
 $Y)$ then 1 else 0) =
 $(\sum Y \mid Y \subseteq X \wedge \text{card } Y = k. \text{prod } f Y)$.
 qed

lemma sym-mpoly-nz [simp]:
 assumes finite A $k \leq \text{card } A$
 shows sym-mpoly A $k \neq (0 :: 'a :: \text{zero-neq-one mpoly})$
proof –
 from assms obtain B where $B \subseteq A$ $\text{card } B = k$
 using ex-subset-of-card by blast
 with assms have coeff (sym-mpoly A $k :: 'a \text{ mpoly}$) (monom-of-set B) = 1
 by (intro coeff-sym-mpoly-monom-of-set)
 thus ?thesis by auto
 qed

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

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 –
from *assms* **have** $\text{lead-coeff } (\text{sym-mpoly } A \ k) \neq 0$ **by** *simp*
thus *?thesis* **using** *coeff-sym-mpoly-0-or-1*[*of A k lead-monom (sym-mpoly A k)*]
unfolding *lead-coeff-def* **by** *blast*
qed

lemma *lead-monom-sym-mpoly*:

assumes *sorted xs distinct xs k ≤ length xs*

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

proof –

let $?m = \text{lead-monom } ?p$

have *sym*: *symmetric-mpoly (set xs) (sym-mpoly (set xs) k)*

by (*intro symmetric-sym-mpoly*) *auto*

from *assms* **have** [*simp*]: $\text{card } (\text{set } xs) = \text{length } xs$

by (*subst distinct-card*) *auto*

from *assms* **have** $\text{lead-coeff } ?p = 1$

by (*subst lead-coeff-sym-mpoly*) *auto*

then obtain *X* **where** $X \subseteq \text{set } xs \ \text{card } X = k \ ?m = \text{monom-of-set } X$

unfolding *lead-coeff-def* **by** (*subst (asm) coeff-sym-mpoly*) (*auto split: if-splits*)

define *ys* **where** $ys = \text{map } (\lambda x. \text{if } x \in X \text{ then } 1 \text{ else } 0 :: \text{nat}) \ xs$

have [*simp*]: $\text{length } ys = \text{length } xs$ **by** (*simp add: ys-def*)

have *ys-altdef*: $ys = \text{map } (\text{lookup } ?m) \ xs$

unfolding *ys-def* **using** *X finite-subset[OF X(1)]*

by (*intro map-cong*) (*auto simp: lookup-monom-of-set*)

define *i* **where** $i = \text{Min } (\text{insert } (\text{length } xs) \ \{i. i < \text{length } xs \wedge ys ! i = 0\})$

have $i \leq \text{length } xs$ **by** (*auto simp: i-def*)

have *in-X*: $xs ! j \in X$ **if** $j < i$ **for** *j*

using *that* **unfolding** *i-def* **by** (*auto simp: ys-def*)

have *not-in-X*: $xs ! j \notin X$ **if** $i \leq j < \text{length } xs$ **for** *j*

proof –

have *ne*: $\{i. i < \text{length } xs \wedge ys ! i = 0\} \neq \{\}$

proof

assume [*simp*]: $\{i. i < \text{length } xs \wedge ys ! i = 0\} = \{\}$

from *that* **show** *False* **by** (*simp add: i-def*)

qed

hence $\text{Min } \{i. i < \text{length } xs \wedge ys ! i = 0\} \in \{i. i < \text{length } xs \wedge ys ! i = 0\}$

using *that* **by** (*intro Min-in*) *auto*

also have $\text{Min } \{i. i < \text{length } xs \wedge ys ! i = 0\} = i$

unfolding *i-def* **using** *ne* **by** (*subst Min-insert*) (*auto simp: min-def*)

finally have $ys ! i = 0 \ i < \text{length } xs$ **by** *simp-all*

have $\text{lookup } ?m \ (xs ! j) \leq \text{lookup } ?m \ (xs ! i)$ **using** *that* *assms*

by (*intro lookup-lead-monom-decreasing[OF sym]*)

(*auto intro!: sorted-nth-mono simp: set-conv-nth*)

also have $\dots = 0$ **using** *i* **by** (*simp add: ys-altdef*)

finally show *?thesis* using that X finite-subset[$OF X(1)$] by (auto simp: lookup-monom-of-set)

qed

from X have $k = \text{card } X$

by simp

also have $X = (\lambda i. xs ! i) \cdot \{i. i < \text{length } xs \wedge xs ! i \in X\}$

using X by (auto simp: set-conv-nth)

also have $\text{card } \dots = (\sum i \mid i < \text{length } xs \wedge xs ! i \in X. 1)$

using *assms* by (subst card-image) (auto intro!: inj-on-nth)

also have $\dots = (\sum i \mid i < \text{length } xs. \text{if } xs ! i \in X \text{ then } 1 \text{ else } 0)$

by (intro sum.mono-neutral-cong-left) auto

also have $\dots = \text{sum-list } ys$

by (auto simp: sum-list-sum-nth ys-def intro!: sum.cong)

also have $ys = \text{take } i \text{ } ys @ \text{drop } i \text{ } ys$ by simp

also have $\text{sum-list } \dots = \text{sum-list } (\text{take } i \text{ } ys) + \text{sum-list } (\text{drop } i \text{ } ys)$

by (subst sum-list-append) auto

also have $\text{take } i \text{ } ys = \text{replicate } i \text{ } 1$ using $\langle i \leq \text{length } xs \rangle$ in- X

by (intro replicate-eqI) (auto simp: ys-def set-conv-nth)

also have $\text{sum-list } \dots = i$ by simp

also have $\text{drop } i \text{ } ys = \text{replicate } (\text{length } ys - i) \text{ } 0$ using $\langle i \leq \text{length } xs \rangle$ not-in- X

by (intro replicate-eqI) (auto simp: ys-def set-conv-nth)

also have $\text{sum-list } \dots = 0$ by simp

finally have $i = k$ by simp

have $X = \text{set } (\text{filter } (\lambda x. x \in X) \text{ } xs)$

using X by auto

also have $xs = \text{take } i \text{ } xs @ \text{drop } i \text{ } xs$ by simp

also note filter-append

also have $\text{filter } (\lambda x. x \in X) (\text{take } i \text{ } xs) = \text{take } i \text{ } xs$

using in- X by (intro filter-True) (auto simp: set-conv-nth)

also have $\text{filter } (\lambda x. x \in X) (\text{drop } i \text{ } xs) = []$

using not-in- X by (intro filter-False) (auto simp: set-conv-nth)

finally have $X = \text{set } (\text{take } i \text{ } xs)$ by simp

with $\langle i = k \rangle$ and X show *?thesis* by simp

qed

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* $A :: ((\text{nat} \Rightarrow_0 'b :: \{\text{zero}, \text{wellorder}\}) \times -) \text{ set}$)

proof (*rule wf-subset*)

```

define n where n = Suc (Max (insert 0 A))
have less-n: k < n if k ∈ A for k
  using that assms by (auto simp: n-def less-Suc-eq-le Max-ge-iff)

define f :: (nat ⇒0 'b) ⇒ 'b list where f = (λm. map (lookup m) [0..n])

show wf (inv-image (lexn {(x,y). x < y} n) f)
  by (intro wf-inv-image wf-lexn wellorder-class.wf)
show monom-less-on A ⊆ inv-image (lexn {(x, y). x < y} n) f
proof safe
  fix m1 m2 :: nat ⇒0 'b assume (m1, m2) ∈ monom-less-on A
  hence m12: m1 < m2 keys m1 ⊆ A keys m2 ⊆ A
    by (auto simp: monom-less-on-def)
  then obtain k where k: lookup m1 k < lookup m2 k ∀ i < k. lookup m1 i =
lookup m2 i
    by (auto simp: less-poly-mapping-def less-fun-def)
  have ¬(lookup m1 k = 0 ∧ lookup m2 k = 0)
  proof (intro notI)
    assume lookup m1 k = 0 ∧ lookup m2 k = 0
    hence [simp]: lookup m1 k = 0 lookup m2 k = 0 by blast+
    from k(1) show False by simp
  qed
  hence k ∈ A using m12 by (auto simp: in-keys-iff)
  hence k < n by (simp add: less-n)

define as where as = map (lookup m1) [0..k]
define bs1 where bs1 = map (lookup m1) [Suc k..n]
define bs2 where bs2 = map (lookup m2) [Suc k..n]
have decomp: [0..n] = [0..k] @ [k] @ drop (Suc k) [0..n]
  using <k < n> by (simp flip: upt-conv-Cons upt-add-eq-append')
have [simp]: length as = k length bs1 = n - Suc k length bs2 = n - Suc k
  by (simp-all add: as-def bs1-def bs2-def)

have f m1 = as @ [lookup m1 k] @ bs1 unfolding f-def
  by (subst decomp) (simp add: as-def bs1-def)
moreover have f m2 = as @ [lookup m2 k] @ bs2 unfolding f-def
  using k by (subst decomp) (simp add: as-def bs2-def)
ultimately show (m1, m2) ∈ inv-image (lexn {(x,y). x < y} n) f
  using k(1) <k < n> unfolding lexn-conv by auto
qed
qed

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

```

```

using assms(2)
proof (induct  $m \equiv \text{lead-monom } p$  arbitrary: p rule: wf-induct-rule[OF wf-monom-less-on[OF
fin]])
  case (1 p)
  show ?case
  proof (rule IH)
    fix  $p' :: 'a \text{ mpoly}$  assume *:  $\text{vars } p' \subseteq A$  lead-monom  $p' < \text{lead-monom } p$ 
    show  $P p'$ 
    by (rule 1) (insert * 1.prems keys-lead-monom-subset, auto simp: monom-less-on-def)
  qed (insert 1, auto)
qed

```

```

lemma lead-monom-induct' [case-names less]:
  fixes  $p :: 'a :: \text{zero mpoly}$ 
  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 -
  have finite ( $\text{vars } p$ )  $\text{vars } p \subseteq \text{vars } p$  by (auto simp: vars-finite)
  thus ?thesis
  by (induction rule: lead-monom-induct) (use IH in blast)
qed

```

2.12 The fundamental theorem of symmetric polynomials

```

lemma lead-coeff-sym-mpoly-powerprod:
  assumes finite  $A \bigwedge x. x \in X \implies f x \in \{1.. \text{card } A\}$ 
  shows lead-coeff  $(\prod_{x \in X}. \text{sym-mpoly } A (f (x :: 'a))) \wedge g x = 1$ 
proof -
  have eq: lead-coeff  $(\text{sym-mpoly } A (f x) \wedge g x :: 'b \text{ mpoly}) = 1$  if  $x \in X$  for  $x$ 
  using that assms by (subst lead-coeff-power) (auto simp: lead-coeff-sym-mpoly
assms)
  hence  $(\prod_{x \in X}. \text{lead-coeff } (\text{sym-mpoly } A (f x) \wedge g x :: 'b \text{ mpoly})) = (\prod_{x \in X}. 1)$ 
  by (intro prod.cong eq refl)
  also have  $\dots = 1$  by simp
  finally have eq':  $(\prod_{x \in X}. \text{lead-coeff } (\text{sym-mpoly } A (f x) \wedge g x :: 'b \text{ mpoly})) = 1$ 
  .
  show ?thesis by (subst lead-coeff-prod) (auto simp: eq eq')
qed

```

```

context
  fixes  $A :: \text{nat set}$  and  $xs \text{ n } f$  and  $\text{decr} :: 'a :: \text{comm-ring-1 mpoly} \Rightarrow \text{bool}$ 
  defines  $xs \equiv \text{sorted-list-of-set } A$ 
  defines  $n \equiv \text{card } A$ 
  defines  $f \equiv (\lambda i. \text{if } i < n \text{ then } xs ! i \text{ else } 0)$ 
  defines  $\text{decr} \equiv (\lambda p. \forall i \in A. \forall j \in A. i \leq j \longrightarrow$ 
 $\text{lookup } (\text{lead-monom } p) i \geq \text{lookup } (\text{lead-monom } p) j)$ 
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 $(\sum_{i < n}. \text{Poly-Mapping.single } (\text{Suc } i) (g \ i - g (\text{Suc } i)))$)

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 $\text{fund-sym-step-coeff } p * (\prod_{i < n}. \text{sym-mpoly } A (\text{Suc } i) \wedge (g \ i - g (\text{Suc } i)))$)

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
fund-sym-poly-wit $(p - \text{fund-sym-step-poly } p) +$
monom $(\text{fund-sym-step-monom } p) (\text{fund-sym-step-coeff } p)$)
by *auto*

lemma *coeff-fund-sym-step-coeff*: *coeff* $(\text{fund-sym-step-coeff } p) \ m \in \{\text{lead-coeff } p, 0\}$
by (*auto simp: fund-sym-step-coeff-def coeff-monom when-def*)

lemma *vars-fund-sym-step-coeff*: *vars* $(\text{fund-sym-step-coeff } p) \subseteq \text{vars } p - A$
unfolding *fund-sym-step-coeff-def* **using** *keys-lead-monom-subset*[of p]
by (*intro order.trans[OF vars-monom-subset]*) *auto*

lemma *keys-fund-sym-step-monom*: *keys* $(\text{fund-sym-step-monom } p) \subseteq \{1..n\}$
unfolding *fund-sym-step-monom-def* *Let-def*
by (*intro order.trans[OF keys-sum] UN-least, subst keys-single*) *auto*

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 –
interpret *ring-closed* C **by** *fact*
have *: $\bigwedge m. \text{coeff } (p \wedge x) \ m \in C$ **if** $\bigwedge m. \text{coeff } p \ m \in C$ **for** $p \ x$
using *that* **by** (*induction* x)
(auto simp: coeff-mpoly-times mpoly-coeff-1 intro!: prod-fun-closed)
have **: $\bigwedge m. \text{coeff } (\text{prod } f \ X) \ m \in C$ **if** $\bigwedge i \ m. i \in X \implies \text{coeff } (f \ i) \ m \in C$
for X **and** $f :: \text{nat} \Rightarrow -$
using *that* **by** (*induction* X *rule: infinite-finite-induct*)
(auto simp: coeff-mpoly-times mpoly-coeff-1 intro!: prod-fun-closed)
show *?thesis* **using** C
unfolding *fund-sym-step-poly-def* *Let-def* *fund-sym-step-coeff-def* *coeff-mpoly-times*
by (*intro* *prod-fun-closed*)
*(auto simp: coeff-monom when-def lead-coeff-def coeff-sym-mpoly intro!: **)*
qed

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

proof –

define g **where** $g = (\lambda i. \text{if } i < n \text{ then } \text{lookup } (\text{lead-monom } p) \ (f \ i) \ \text{else } 0)$
define q **where** $q = (\prod i < n. \text{sym-mpoly } A \ (\text{Suc } i) \wedge (g \ i - g \ (\text{Suc } i)) :: 'a$
 $\text{mpoly})$
define c **where** $c = \text{monom } (\text{restrictpm } (-A) \ (\text{lead-monom } p)) \ (\text{lead-coeff } p)$
have [*simp*]: $\text{fund-sym-step-poly } p = c * q$
by (*simp* *add: fund-sym-step-poly-def* *fund-sym-step-coeff-def* *c-def* *q-def* *f-def* *g-def*)

have $\text{vars } (c * q) \subseteq \text{vars } p \cup A$
using *keys-lead-monom-subset*[*of* p]
 vars-monom-subset [*of* $\text{restrictpm } (-A) \ (\text{lead-monom } p) \ \text{lead-coeff } p]$
unfolding *c-def* *q-def*
by (*intro* *order.trans*[*OF* *vars-mult*] *order.trans*[*OF* *vars-prod*] *order.trans*[*OF* *vars-power*])

Un-least UN-least order.trans[OF vars-sym-mpoly-subset]) auto

thus *vars (fund-sym-step-poly p) ⊆ vars p ∪ A*
by *simp*
have *symmetric-mpoly A (c * q) unfolding c-def q-def*
by *(intro symmetric-mpoly-mult symmetric-mpoly-monom symmetric-mpoly-prod*
symmetric-mpoly-power symmetric-sym-mpoly) auto
thus *symmetric-mpoly A (fund-sym-step-poly p) by simp*

assume *finite: finite A and [simp]: p ≠ 0 and decr p*
have *set xs = A distinct xs and [simp]: length xs = n*
using *finite by (auto simp: xs-def n-def)*
have *[simp]: lead-coeff c = lead-coeff p lead-monom c = restrictpm (− A)*
(lead-monom p)
by *(simp-all add: c-def lead-monom-monom)*
hence *f-range [simp]: f i ∈ A if i < n for i*
using *that ⟨set xs = A⟩ by (auto simp: f-def set-conv-nth)*
have *sorted xs by (simp add: xs-def)*
hence *f-mono: f i ≤ f j if i ≤ j j < n for i j using that*
by *(auto simp: f-def n-def intro: sorted-nth-mono)*
hence *g-mono: g i ≥ g j if i ≤ j for i j*
unfolding *g-def using that using ⟨decr p⟩ by (auto simp: decr-def)*

have **: (∏ i < n. lead-coeff (sym-mpoly A (Suc i) ^ (g i − g (Suc i)) :: 'a mpoly))*
 $=$
 $(\prod i < \text{card } A. 1)$
using *⟨finite A⟩ by (intro prod.cong) (auto simp: n-def lead-coeff-power)*
hence *lead-coeff q = (∏ i < n. lead-coeff (sym-mpoly A (Suc i) ^ (g i − g (Suc i)) :: 'a mpoly))*
by *(simp add: lead-coeff-prod lead-coeff-power n-def q-def)*
also have $\dots = (\prod i < n. 1)$
using *⟨finite A⟩ by (intro prod.cong) (auto simp: lead-coeff-power n-def)*
finally have *[simp]: lead-coeff q = 1 by simp*

have *lead-monom q = (∑ i < n. lead-monom (sym-mpoly A (Suc i) ^ (g i − g (Suc i)) :: 'a mpoly))*
using ** by (simp add: q-def lead-monom-prod lead-coeff-power n-def)*
also have $\dots = (\sum i < n. \text{of-nat } (g i - g (Suc i)) * \text{lead-monom } (sym-mpoly A (Suc i) :: 'a mpoly))$
using *⟨finite A⟩ by (intro sum.cong) (auto simp: lead-monom-power n-def)*
also have $\dots = (\sum i < n. \text{of-nat } (g i - g (Suc i)) * \text{monom-of-set } (set (take (Suc i) xs)))$
proof *(intro sum.cong refl, goal-cases)*
case *(1 i)*
have *lead-monom (sym-mpoly A (Suc i) :: 'a mpoly) =*
 $\text{lead-monom } (sym-mpoly (set xs) (Suc i) :: 'a mpoly)$
by *(simp add: ⟨set xs = A⟩)*
also from *1 have* $\dots = \text{monom-of-set } (set (take (Suc i) xs))$
by *(subst lead-monom-sym-mpoly) (auto simp: xs-def n-def)*
finally show *?case by simp*

```

qed
finally have lead-monom-q:
  lead-monom q = (∑ i < n. of-nat (g i - g (Suc i)) * monom-of-set (set (take (Suc i) xs))) .

have lead-monom (c * q) = lead-monom c + lead-monom q
  by (simp add: lead-monom-mult)
also have ... = lead-monom p (is ?S = -)
proof (intro poly-mapping-eqI)
  fix i :: nat
  show lookup (lead-monom c + lead-monom q) i = lookup (lead-monom p) i
  proof (cases i ∈ A)
    case False
    hence lookup (lead-monom c + lead-monom q) i = lookup (lead-monom p) i
+
    (∑ j < n. (g j - g (Suc j)) * lookup (monom-of-set (set (take (Suc j) xs))) i)
    (is - = - + ?S) by (simp add: lookup-add lead-monom-q lookup-sum)
    also from False have ?S = 0
    by (intro sum.neutral) (auto simp: lookup-monom-of-set ⟨set xs = A⟩ dest!: in-set-takeD)
    finally show ?thesis by simp
next
case True
with ⟨set xs = A⟩ obtain m where m: i = xs ! m m < n
  by (auto simp: set-conv-nth)
have lookup (lead-monom c + lead-monom q) i =
  (∑ j < n. (g j - g (Suc j)) * lookup (monom-of-set (set (take (Suc j) xs))) i)
  using True by (simp add: lookup-add lookup-sum lead-monom-q)
also have ... = (∑ j | j < n ∧ i ∈ set (take (Suc j) xs). g j - g (Suc j))
  by (intro sum.mono-neutral-cong-right) auto
also have {j. j < n ∧ i ∈ set (take (Suc j) xs)} = {m..<n}
  using m ⟨distinct xs⟩ by (force simp: set-conv-nth nth-eq-iff-index-eq)
also have (∑ j ∈ ... g j - g (Suc j)) = (∑ j ∈ ... g j) - (∑ j ∈ ... g (Suc j))
  by (subst sum-subtractf-nat) (auto intro!: g-mono)
also have (∑ j ∈ {m..<n}. g (Suc j)) = (∑ j ∈ {m<..n}. g j)
  by (intro sum.reindex-bij-witness[of - λj. j - 1 Suc]) auto
also have ... = (∑ j ∈ {m<..<n}. g j)
  by (intro sum.mono-neutral-right) (auto simp: g-def)
also have (∑ j ∈ {m..<n}. g j) - ... = (∑ j ∈ {m..<n} - {m<..<n}. g j)
  by (intro sum-diff-nat [symmetric]) auto
also have {m..<n} - {m<..<n} = {m} using m by auto
also have (∑ j ∈ ... g j) = lookup (lead-monom p) i
  using m by (auto simp: g-def not-less le-Suc-eq f-def)
finally show ?thesis .
qed
qed

```

finally show $\text{lead-monom } (\text{fund-sym-step-poly } p) = \text{lead-monom } p$ **by** *simp*
show $\text{lead-coeff } (\text{fund-sym-step-poly } p) = \text{lead-coeff } p$
by (*simp add: lead-coeff-mult*)

have $*$: $\text{lookup } (\text{fund-sym-step-monom } p) k = (\text{if } k \in \{1..n\} \text{ then } g (k - 1) - g k \text{ else } 0)$ **for** k
proof –
have $\text{lookup } (\text{fund-sym-step-monom } p) k =$
 $(\sum x \in (\text{if } k \in \{1..n\} \text{ then } \{k - 1\} \text{ else } \{\}) . g (k - 1) - g k)$
unfolding *fund-sym-step-monom-def lookup-sum Let-def*
by (*intro sum.mono-neutral-cong-right*)
 $(\text{auto simp: } g\text{-def lookup-single when-def split: if-splits})$
thus *?thesis* **by** *simp*
qed
hence $(\prod x . \text{sym-mpoly } A x \wedge \text{lookup } (\text{fund-sym-step-monom } p) x :: 'a \text{ mpoly}) =$
 $(\prod x \in \{1..n\} . \text{sym-mpoly } A x \wedge \text{lookup } (\text{fund-sym-step-monom } p) x)$
by (*intro Prod-any.expand-superset*) *auto*
also have $\dots = (\prod x < n . \text{sym-mpoly } A (\text{Suc } x) \wedge \text{lookup } (\text{fund-sym-step-monom } p) (\text{Suc } x))$
by (*intro prod.reindex-bij-witness[of - Suc] i . i - 1*) *auto*
also have $\dots = q$
unfolding *q-def* **by** (*intro prod.cong*) (*auto simp: **)
finally show $\text{fund-sym-step-poly } p =$
 $\text{fund-sym-step-coeff } p * (\prod x . \text{sym-mpoly } A x \wedge \text{lookup } (\text{fund-sym-step-monom } p) x)$
by (*simp add: c-def q-def f-def g-def fund-sym-step-monom-def fund-sym-step-coeff-def*)
qed

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 ≠ 0 and decr p*
shows $\text{lead-monom } (p - \text{fund-sym-step-poly } p) < \text{lead-monom } p$
proof (*cases p = fund-sym-step-poly p*)
case *True*
thus *?thesis* **using** *assms* **by** (*auto simp: order.strict-iff-order*)
next
case *False*
from *assms* **have** [*simp*]: $p \neq 0$ **by** *auto*
let $?q = \text{fund-sym-step-poly } p$ **and** $?m = \text{lead-monom } p$
have $\text{coeff } (p - ?q) ?m = 0$
using *fund-sym-step-poly[of p] assms* **by** (*simp add: lead-coeff-def*)
moreover have $\text{lead-coeff } (p - ?q) \neq 0$ **using** *False* **by** *auto*
ultimately have $\text{lead-monom } (p - ?q) \neq ?m$
unfolding *lead-coeff-def* **by** *auto*
moreover have $\text{lead-monom } (p - ?q) \leq ?m$
using *fund-sym-step-poly[of p] assms*
by (*intro order.trans[OF lead-monom-diff] max.boundedI*) *auto*
ultimately show *?thesis* **by** (*auto simp: order.strict-iff-order*)

qed

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* $\subseteq B$ $A \subseteq B$
shows *fund-sym-poly-wit-dom p*
using *assms(1-3)*
proof (*induction p rule: lead-monom-induct*)
case (*less p*)
have [*simp*]: *finite A* **by** (*rule finite-subset[of - B]*) *fact+*
show *?case*
proof (*cases lead-monom p = 0 \vee \neg symmetric-mpoly A p*)
case *False*
hence [*simp*]: *p \neq 0* **by** *auto*
note *decr = lookup-lead-monom-decreasing[of A p]*
have *vars (p - fund-sym-step-poly p) \subseteq B*
using *fund-sym-step-poly[of p] decr False less.premis less.hyps $\langle A \subseteq B \rangle$*
by (*intro order.trans[OF vars-diff]*) *auto*
hence *fund-sym-poly-wit-dom (p - local.fund-sym-step-poly p)*
using *False less.premis less.hyps decr*
by (*intro less.IH fund-sym-step-poly symmetric-mpoly-diff*
lead-monom-fund-sym-step-poly-less) (*auto simp: decr-def*)
thus *?thesis* **using** *fund-sym-poly-wit.domintros* **by** *blast*
qed (*auto intro: fund-sym-poly-wit.domintros*)
qed

lemma *fund-sym-poly-wit-dom [intro]: fund-sym-poly-wit-dom p*
proof –
consider \neg *symmetric-mpoly A p* | *vars p* $\cap A = \{\}$ | *symmetric-mpoly A p A*
 \subseteq *vars p*
using *symmetric-mpoly-imp-orthogonal-or-subset[of A p]* **by** *blast*
thus *?thesis*
proof *cases*
assume *symmetric-mpoly A p A \subseteq vars p*
thus *?thesis* **using** *fund-sym-poly-wit-dom-aux[of vars p p]* **by** (*auto simp:*
vars-finite)
qed (*auto intro: fund-sym-poly-wit.domintros*)
qed

termination *fund-sym-poly-wit*
by (*intro allI fund-sym-poly-wit-dom*)

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* $\subseteq B$ *symmetric-mpoly A p* $A \subseteq B$
shows *vars (coeff (fund-sym-poly-wit p) m)* $\subseteq B - A$
using *assms*
proof (*induction p rule: fund-sym-poly-wit.induct*)
case (*1 p*)
show *?case*
proof (*cases lead-monom p = 0 \vee vars p \cap A = {}*)
case *False*
have *vars (p - fund-sym-step-poly p)* $\subseteq B$
using *1.premis fund-sym-step-poly[of p]* **by** (*intro order.trans[OF vars-diff]*)
auto
with *1 False* **have** *vars (coeff (fund-sym-poly-wit (p - fund-sym-step-poly p)) m)* $\subseteq B - A$
by (*intro 1 symmetric-mpoly-diff fund-sym-step-poly*) *auto*
hence *vars (coeff (fund-sym-poly-wit (p - fund-sym-step-poly p) + monom (fund-sym-step-monom p) (fund-sym-step-coeff p)) m)* $\subseteq B - A$
unfolding *coeff-add coeff-monom* **using** *vars-fund-sym-step-coeff[of p] 1.premis*
by (*intro order.trans[OF vars-add] Un-least order.trans[OF vars-monom-subset]*)
(auto simp: when-def)
thus *?thesis* **using** *1.premis False* **unfolding** *fund-sym-poly-wit.simps[of p]* **by** *simp*
qed (*insert 1.premis,*
auto simp: fund-sym-poly-wit.simps[of p] mpoly-coeff-Const lead-monom-eq-0-iff)
qed

lemma *fund-sym-poly-wit-coeffs*:
assumes *symmetric-mpoly A p*
shows *vars (coeff (fund-sym-poly-wit p) m)* \subseteq *vars p - A*
proof (*cases A \subseteq vars p*)
case *True*
with *fund-sym-poly-wit-coeffs-aux[of vars p p m]* *assms*
show *?thesis* **by** (*auto simp: vars-finite*)
next
case *False*
hence *vars p \cap A = {}*
using *symmetric-mpoly-imp-orthogonal-or-subset[OF assms]* **by** *auto*

```

thus ?thesis by (auto simp: fund-sym-poly-wit.simps[of p] mpoly-coeff-Const)
qed

lemma fund-sym-poly-wit-vars: vars (fund-sym-poly-wit p)  $\subseteq$  {1..n}
proof (cases symmetric-mpoly A p  $\wedge$  A  $\subseteq$  vars p)
  case True
    define B where B = vars p
    have finite B vars p  $\subseteq$  B symmetric-mpoly A p A  $\subseteq$  B
      using True unfolding B-def by (auto simp: vars-finite)
    thus ?thesis
    proof (induction p rule: fund-sym-poly-wit.induct)
      case (1 p)
        show ?case
        proof (cases lead-monom p = 0  $\vee$  vars p  $\cap$  A = {})
          case False
            have vars (p - fund-sym-step-poly p)  $\subseteq$  B
              using 1.prem1 fund-sym-step-poly[of p] by (intro order.trans[OF vars-diff])
            auto
            hence vars (local.fund-sym-poly-wit (p - local.fund-sym-step-poly p))  $\subseteq$  {1..n}
              using False 1.prem1
            by (intro 1 symmetric-mpoly-diff fund-sym-step-poly) (auto simp: lead-monom-eq-0-iff)
            hence vars (fund-sym-poly-wit (p - fund-sym-step-poly p) +
              monom (fund-sym-step-monom p) (local.fund-sym-step-coeff p))  $\subseteq$ 
              {1..n}
            by (intro order.trans[OF vars-add] Un-least order.trans[OF vars-monom-subset]
              keys-fund-sym-step-monom) auto
            thus ?thesis using 1.prem1 False unfolding fund-sym-poly-wit.simps[of p]
          case True
            by simp
            qed (insert 1.prem1,
              auto simp: fund-sym-poly-wit.simps[of p] mpoly-coeff-Const lead-monom-eq-0-iff)
          qed
        next
          case False
            then consider  $\neg$ symmetric-mpoly A p | symmetric-mpoly A p vars p  $\cap$  A = {}
              using symmetric-mpoly-imp-orthogonal-or-subset[of A p] by auto
            thus ?thesis
            by cases (auto simp: fund-sym-poly-wit.simps[of p])
          qed
    qed

lemma fund-sym-poly-wit-insertion-aux:
  assumes finite B vars p  $\subseteq$  B symmetric-mpoly A p A  $\subseteq$  B
  shows insertion (sym-mpoly A) (fund-sym-poly-wit p) = p
  using assms
proof (induction p rule: fund-sym-poly-wit.induct)
  case (1 p)
    from 1.prem1 have decr p
      using lookup-lead-monom-decreasing[of A p] by (auto simp: decr-def)
    show ?case
    proof (cases lead-monom p = 0  $\vee$  vars p  $\cap$  A = {})

```

```

case False
have vars ( $p - \text{fund-sym-step-poly } p$ )  $\subseteq B$ 
  using 1.prems fund-sym-step-poly[of p] by (intro order.trans[OF vars-diff])
auto
hence insertion (sym-mpoly A) (fund-sym-poly-wit ( $p - \text{fund-sym-step-poly } p$ ))
=
   $p - \text{fund-sym-step-poly } p$  using 1 False
  by (intro 1 symmetric-mpoly-diff fund-sym-step-poly) auto
moreover have fund-sym-step-poly  $p =$ 
   $\text{fund-sym-step-coeff } p * (\prod x. \text{sym-mpoly } A \ x \ ^{\text{lookup}}$ 
(fund-sym-step-monom  $p$ )  $x)$ 
  using 1.prems finite-subset[of A B] False (decr p) by (intro fund-sym-step-poly)
auto
ultimately show ?thesis
  unfolding fund-sym-poly-wit.simps[of p] by (auto simp: insertion-add)
qed (auto simp: fund-sym-poly-wit.simps[of p])
qed

```

```

lemma fund-sym-poly-wit-insertion:
assumes symmetric-mpoly A p
shows insertion (sym-mpoly A) (fund-sym-poly-wit  $p$ ) =  $p$ 
proof (cases A  $\subseteq$  vars p)
case False
hence vars  $p \cap A = \{\}$ 
  using symmetric-mpoly-imp-orthogonal-or-subset[OF assms] by auto
thus ?thesis
  by (auto simp: fund-sym-poly-wit.simps[of p])
next
case True
with fund-sym-poly-wit-insertion-aux[of vars p p] assms show ?thesis
  by (auto simp: vars-finite)
qed

```

```

lemma fund-sym-poly-wit-coeff:
assumes  $\forall m. \text{coeff } p \ m \in C$  ring-closed C
shows  $\forall m \ m'. \text{coeff} (\text{coeff} (\text{fund-sym-poly-wit } p) \ m) \ m' \in C$ 
using assms(1)
proof (induction p rule: fund-sym-poly-wit.induct)
case (1 p)
interpret ring-closed C by fact
show ?case
proof (cases  $\neg \text{symmetric-mpoly } A \ p \vee \text{lead-monom } p = 0 \vee \text{vars } p \cap A = \{\}$ )
case True
thus ?thesis using 1.prems
  by (auto simp: fund-sym-poly-wit.simps[of p] mpoly-coeff-Const)
next
case False
have  $*$ :  $\forall m \ m'. \text{coeff} (\text{coeff} (\text{fund-sym-poly-wit} (p - \text{fund-sym-step-poly } p))$ 
 $m) \ m' \in C$ 

```



```

    using False 1.premss assms coeff-fund-sym-step-poly [of p] by (intro 1) auto
  show ?thesis
  proof (intro allI, goal-cases)
    case (1 m m')
    thus ?case using * False coeff-fund-sym-step-coeff [of p m'] 1.premss
    by (auto simp: fund-sym-poly-wit.simps [of p] coeff-monom lead-coeff-def
when-def)
  qed
  qed
  qed

```

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 $\text{lead-monom } (\prod i = 1..n. \text{sym-mpoly } A \ i \ ^h \ h \ i \ :: \ 'a \ \text{mpoly}) =$
 $(\sum i = 1..n. \text{of-nat } (h \ i) * \text{lead-monom } (\text{sym-mpoly } A \ i \ :: \ 'a \ \text{mpoly}))$

proof –

have $(\prod i = 1..n. \text{lead-coeff } (\text{sym-mpoly } A \ i \ ^h \ h \ i \ :: \ 'a \ \text{mpoly})) = 1$

using *assms unfolding n-def* **by** (*intro prod.neutral allI*) (*auto simp: lead-coeff-power*)

hence $\text{lead-monom } (\prod i = 1..n. \text{sym-mpoly } A \ i \ ^h \ h \ i \ :: \ 'a \ \text{mpoly}) =$
 $(\sum i = 1..n. \text{lead-monom } (\text{sym-mpoly } A \ i \ ^h \ h \ i \ :: \ 'a \ \text{mpoly}))$

by (*subst lead-monom-prod*) *auto*

also have $\dots = (\sum i = 1..n. \text{of-nat } (h \ i) * \text{lead-monom } (\text{sym-mpoly } A \ i \ :: \ 'a \ \text{mpoly}))$

by (*intro sum.cong refl, subst lead-monom-power*)

(*auto simp: lead-coeff-power assms n-def*)

finally show *?thesis* .

qed

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

assumes *finite A k ∉ A*

shows $\text{lookup } (\text{lead-monom } (\prod i = 1..n. \text{sym-mpoly } A \ i \ ^h \ h \ i \ :: \ 'a \ \text{mpoly})) \ k =$
 0

proof –

have *xs: set xs = A distinct xs sorted xs* **and** [*simp*]: *length xs = n*

using *assms* **by** (*auto simp: xs-def n-def*)

have $\text{lead-monom } (\prod i = 1..n. \text{sym-mpoly } A i \wedge h i :: 'a \text{ mpoly}) =$
 $(\sum i = 1..n. \text{of-nat } (h i) * \text{lead-monom } (\text{sym-mpoly } (\text{set } xs) i :: 'a \text{ mpoly}))$
by $(\text{subst } \text{lead-monom-sym-mpoly-prod})$ $(\text{use } xs \text{ assms in auto})$
also have $\text{lookup } \dots k = 0$ **unfolding** lookup-sum
by $(\text{intro } \text{sum.neutral } \text{ballI}, \text{subst } \text{lead-monom-sym-mpoly})$
 $(\text{insert } xs \text{ assms}, \text{auto } \text{simp}: xs \text{ lead-monom-sym-mpoly } \text{lookup-monom-of-set}$
 $\text{set-conv-nth})$
finally show $?thesis$.
qed

lemma $\text{lead-monom-sym-mpoly-prod-in}$:

assumes $\text{finite } A$ $k < n$

shows $\text{lookup } (\text{lead-monom } (\prod i=1..n. \text{sym-mpoly } A i \wedge h i :: 'a \text{ mpoly})) (xs ! k) =$
 $(\sum i=k+1..n. h i)$

proof –

have $xs: \text{set } xs = A$ $\text{distinct } xs$ $\text{sorted } xs$ **and** $[simp]: \text{length } xs = n$

using $assms$ **by** $(\text{auto } \text{simp}: xs\text{-def } n\text{-def})$

have $\text{lead-monom } (\prod i = 1..n. \text{sym-mpoly } A i \wedge h i :: 'a \text{ mpoly}) =$
 $(\sum i = 1..n. \text{of-nat } (h i) * \text{lead-monom } (\text{sym-mpoly } (\text{set } xs) i :: 'a$
 $\text{mpoly}))$

by $(\text{subst } \text{lead-monom-sym-mpoly-prod})$ $(\text{use } xs \text{ assms in } \text{simp-all})$

also have $\dots = (\sum i=1..n. \text{of-nat } (h i) * \text{monom-of-set } (\text{set } (\text{take } i \text{ } xs)))$

using xs **by** $(\text{intro } \text{sum.cong } \text{refl}, \text{subst } \text{lead-monom-sym-mpoly})$ auto

also have $\text{lookup } \dots (xs ! k) = (\sum i \mid i \in \{1..n\} \wedge xs ! k \in \text{set } (\text{take } i \text{ } xs). h i)$

unfolding $\text{lookup-sum } \text{lookup-monom-of-set}$ **by** $(\text{intro } \text{sum.mono-neutral-cong-right})$

auto

also have $\{i. i \in \{1..n\} \wedge xs ! k \in \text{set } (\text{take } i \text{ } xs)\} = \{k+1..n\}$

proof $(\text{intro } \text{equalityI } \text{subsetI})$

fix i **assume** $i: i \in \{k+1..n\}$

hence $\text{take } i \text{ } xs ! k = xs ! k$ $k < n$ $k < i$ **using** $assms$

by auto

with i **show** $i \in \{i. i \in \{1..n\} \wedge xs ! k \in \text{set } (\text{take } i \text{ } xs)\}$

by $(\text{force } \text{simp}: \text{set-conv-nth})$

qed $(\text{insert } assms \text{ } xs, \text{auto } \text{simp}: \text{set-conv-nth } \text{Suc-le-eq } \text{nth-eq-iff-index-eq})$

finally show $?thesis$.

qed

lemma $\text{lead-monom-sym-poly-powerprod-inj}$:

assumes $\text{lead-monom } (\prod i. \text{sym-mpoly } A i \wedge \text{lookup } m1 i :: 'a \text{ mpoly}) =$

$\text{lead-monom } (\prod i. \text{sym-mpoly } A i \wedge \text{lookup } m2 i :: 'a \text{ mpoly})$

assumes $\text{finite } A$ $\text{keys } m1 \subseteq \{1..n\}$ $\text{keys } m2 \subseteq \{1..n\}$

shows $m1 = m2$

proof $(\text{rule } \text{poly-mapping-eqI})$

fix $k :: \text{nat}$

have $xs: \text{set } xs = A$ $\text{distinct } xs$ $\text{sorted } xs$ **and** $[simp]: \text{length } xs = n$

using $assms$ **by** $(\text{auto } \text{simp}: xs\text{-def } n\text{-def})$

from $assms(3,4)$ **have** $*$: $i \in \{1..n\}$ **if** $\text{lookup } m1 i \neq 0 \vee \text{lookup } m2 i \neq 0$ **for**

```

i
  using that by (auto simp: subset-iff in-keys-iff)
  have **: (∏ i. sym-mpoly A i ^ lookup m i :: 'a mpoly) =
           (∏ i=1..n. sym-mpoly A i ^ lookup m i :: 'a mpoly) if m ∈ {m1, m2}
for m
  using that * by (intro Prod-any.expand-superset subsetI *) (auto intro!:
Nat.gr0I)
  have ***: lead-monom (∏ i=1..n. sym-mpoly A i ^ lookup m1 i :: 'a mpoly) =
            lead-monom (∏ i=1..n. sym-mpoly A i ^ lookup m2 i :: 'a mpoly)
  using assms by (simp add: **)

  have sum-eq: sum (lookup m1) {Suc k..n} = sum (lookup m2) {Suc k..n} if k
< n for k
  using arg-cong[OF ***, of λm. lookup m (xs ! k)] ⟨finite A⟩ that
  by (subst (asm) (1 2) lead-monom-sym-mpoly-prod-in) auto

  show lookup m1 k = lookup m2 k
  proof (cases k ∈ {1..n})
    case False
      hence lookup m1 k = 0 lookup m2 k = 0 using assms by (auto simp:
in-keys-iff)
      thus ?thesis by simp
    next
      case True
      thus ?thesis
      proof (induction n - k arbitrary: k rule: less-induct)
        case (less l)
          have sum (lookup m1) {Suc (l - 1)..n} = sum (lookup m2) {Suc (l - 1)..n}
          using less by (intro sum-eq) auto
          also have {Suc (l - 1)..n} = insert l {Suc l..n}
          using less by auto
          also have sum (lookup m1) ... = lookup m1 l + (∑ i=Suc l..n. lookup m1
i)
          by (subst sum.insert) auto
          also have (∑ i=Suc l..n. lookup m1 i) = (∑ i=Suc l..n. lookup m2 i)
          by (intro sum.cong less) auto
          also have sum (lookup m2) (insert l {Suc l..n}) = lookup m2 l + (∑ i=Suc
l..n. lookup m2 i)
          by (subst sum.insert) auto
          finally show lookup m1 l = lookup m2 l by simp
        qed
      qed
    qed
  qed

```

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 finite A insertion (sym-mpoly A) p = 0
           $\bigwedge m. \text{vars } (\text{coeff } p \ m) \cap A = \{\} \ \text{vars } p \subseteq \{1..n\}$ 
shows  $p = 0$ 
proof (rule ccontr)
assume  $p \neq 0$ 
have  $xs$ : set xs = A distinct xs sorted xs and [simp]: length xs = n
using assms by (auto simp: xs-def n-def)
define  $h$  where  $h = (\lambda m. \text{coeff } p \ m * (\prod i. \text{sym-mpoly } A \ i \ ^ \ \text{lookup } m \ i))$ 
define  $M$  where  $M = \{m. \text{coeff } p \ m \neq 0\}$ 
define  $maxm$  where  $maxm = \text{Max } ((\text{lead-monom } \circ h) \ ` \ M)$ 
have finite M
by (auto intro!: finite-subset[OF - finite-coeff-support[of p]] simp: h-def M-def)
have keys-subset: keys m  $\subseteq$  {1..n} if coeff p m  $\neq$  0 for m
using that assms coeff-notin-vars[of m p] by blast

have lead-coeff: lead-coeff (h m) = lead-coeff (coeff p m) (is ?th1)
and lead-monom: lead-monom (h m) = lead-monom (coeff p m) +
          lead-monom ( $\prod i. \text{sym-mpoly } A \ i \ ^ \ \text{lookup } m \ i :: 'a \ \text{mpoly}$ ) (is
?th2)
if [simp]: coeff p m  $\neq$  0 for m
proof -
have  $(\prod i. \text{sym-mpoly } A \ i \ ^ \ \text{lookup } m \ i :: 'a \ \text{mpoly}) =$ 
           $(\prod i \mid \text{lookup } m \ i \neq 0. \text{sym-mpoly } A \ i \ ^ \ \text{lookup } m \ i :: 'a \ \text{mpoly})$ 
by (intro Prod-any.expand-superset (auto intro!: Nat.gr0I))
also have lead-coeff ... = 1
using assms keys-subset[of m]
by (intro lead-coeff-sym-mpoly-powerprod (auto simp: in-keys-iff subset-iff
?n-def))
finally have eq: lead-coeff ( $\prod i. \text{sym-mpoly } A \ i \ ^ \ \text{lookup } m \ i :: 'a \ \text{mpoly}$ ) = 1 .
thus ?th1 unfolding h-def using  $\langle \text{coeff } p \ m \neq 0 \rangle$  by (subst lead-coeff-mult)
auto
show ?th2 unfolding h-def by (subst lead-monom-mult) (auto simp: eq)
qed

have insertion (sym-mpoly A) p = ( $\sum m \in M. h \ m$ )
unfolding insertion-altdef h-def M-def by (intro Sum-any.expand-superset)
auto
also have lead-monom ... = maxm
unfolding maxm-def

```

```

proof (rule lead-monom-sum)
  from  $p$  obtain  $m$  where  $\text{coeff } p \ m \neq 0$ 
    using  $\text{mpoly-eqI}[of \ p \ 0]$  by  $\text{auto}$ 
  hence  $m \in M$ 
    using  $\langle \text{coeff } p \ m \neq 0 \rangle \text{lead-coeff}[of \ m]$  by ( $\text{auto simp: } M\text{-def}$ )
  thus  $M \neq \{\}$  by  $\text{auto}$ 
next
  have  $\text{restrict-lead-monom}$ :
     $\text{restrictpm } A \ (\text{lead-monom } (h \ m)) =$ 
       $\text{lead-monom } (\prod i. \text{sym-mpoly } A \ i \ ^\wedge \ \text{lookup } m \ i :: 'a \ \text{mpoly})$ 
    if  $[\text{simp}]: \text{coeff } p \ m \neq 0$  for  $m$ 
  proof -
    have  $\text{restrictpm } A \ (\text{lead-monom } (h \ m)) =$ 
       $\text{restrictpm } A \ (\text{lead-monom } (\text{coeff } p \ m)) +$ 
       $\text{restrictpm } A \ (\text{lead-monom } (\prod i. \text{sym-mpoly } A \ i \ ^\wedge \ \text{lookup } m \ i :: 'a$ 
 $\text{mpoly}))$ 
    by ( $\text{auto simp: lead-monom restrictpm-add}$ )
    also have  $\text{restrictpm } A \ (\text{lead-monom } (\text{coeff } p \ m)) = 0$ 
    using  $\text{assms}$  by ( $\text{intro restrictpm-orthogonal order.trans}[OF \ \text{keys-lead-monom-subset}]$ )
   $\text{auto}$ 
    also have  $\text{restrictpm } A \ (\text{lead-monom } (\prod i. \text{sym-mpoly } A \ i \ ^\wedge \ \text{lookup } m \ i :: 'a$ 
 $\text{mpoly})) =$ 
       $\text{lead-monom } (\prod i. \text{sym-mpoly } A \ i \ ^\wedge \ \text{lookup } m \ i :: 'a \ \text{mpoly})$ 
    by ( $\text{intro restrictpm-id order.trans}[OF \ \text{keys-lead-monom-subset}]$ 
       $\text{order.trans}[OF \ \text{vars-Prod-any}] \ \text{UN-least order.trans}[OF \ \text{vars-power}]$ 
       $\text{vars-sym-mpoly-subset}$ )
    finally show  $?thesis$  by  $\text{simp}$ 
  qed
  show  $\text{inj-on } (\text{lead-monom } \circ \ h) \ M$ 
  proof
    fix  $m1 \ m2$  assume  $m12: m1 \in M \ m2 \in M \ (\text{lead-monom } \circ \ h) \ m1 =$ 
 $(\text{lead-monom } \circ \ h) \ m2$ 
    hence  $[\text{simp}]: \text{coeff } p \ m1 \neq 0 \ \text{coeff } p \ m2 \neq 0$  by ( $\text{auto simp: } M\text{-def } h\text{-def}$ )
    have  $\text{restrictpm } A \ (\text{lead-monom } (h \ m1)) = \text{restrictpm } A \ (\text{lead-monom } (h$ 
 $m2))$ 
    using  $m12$  by  $\text{simp}$ 
    hence  $\text{lead-monom } (\prod i. \text{sym-mpoly } A \ i \ ^\wedge \ \text{lookup } m1 \ i :: 'a \ \text{mpoly}) =$ 
       $\text{lead-monom } (\prod i. \text{sym-mpoly } A \ i \ ^\wedge \ \text{lookup } m2 \ i :: 'a \ \text{mpoly})$ 
    by ( $\text{simp add: restrict-lead-monom}$ )
    thus  $m1 = m2$ 
    by ( $\text{rule lead-monom-sym-poly-powerprod-inj}$ )
      ( $\text{use } \langle \text{finite } A \rangle \ \text{keys-subset}[of \ m1] \ \text{keys-subset}[of \ m2]$  in  $\text{auto}$ )
  qed
next
  fix  $m$  assume  $m \in M$ 
  hence  $\text{lead-coeff } (h \ m) = \text{lead-coeff } (\text{coeff } p \ m)$ 
  by ( $\text{simp add: lead-coeff } M\text{-def}$ )
  with  $\langle m \in M \rangle$  show  $h \ m \neq 0$  by ( $\text{auto simp: } M\text{-def}$ )
qed fact+

```

finally have $\text{maxm} = 0$ **by** (*simp add: assms*)

have *only-zero*: $m = 0$ **if** $m \in M$ **for** m

proof –

from *that* **have** nz [*simp*]: $\text{coeff } p \ m \neq 0$ **by** (*auto simp: M-def h-def*)

from *that* **have** $(\text{lead-monom} \circ h) \ m \leq \text{maxm}$

using $\langle \text{finite } M \rangle$ **unfolding** *maxm-def* **by** (*intro Max-ge imageI finite-imageI*)

with $\langle \text{maxm} = 0 \rangle$ **have** [*simp*]: $\text{lead-monom} (h \ m) = 0$ **by** *simp*

have *lookup-nzD*: $k \in \{1..n\}$ **if** $\text{lookup } m \ k \neq 0$ **for** k

using *keys-subset*[*of m*] **that** **by** (*auto simp: in-keys-iff subset-iff*)

have $\text{lead-monom} (\text{coeff } p \ m) + 0 \leq \text{lead-monom} (h \ m)$

unfolding *lead-monom[OF nz]* **by** (*intro add-left-mono*) *auto*

also have $\dots = 0$ **by** *simp*

finally have *lead-monom-0*: $\text{lead-monom} (\text{coeff } p \ m) = 0$ **by** *simp*

have $\text{sum} (\text{lookup } m) \ \{1..n\} = 0$

proof (*rule ccontr*)

assume $\text{sum} (\text{lookup } m) \ \{1..n\} \neq 0$

hence $\text{sum} (\text{lookup } m) \ \{1..n\} > 0$ **by** *presburger*

have $0 \neq \text{lead-coeff} (\text{MPoly-Type.coeff } p \ m)$

by *auto*

also have $\text{lead-coeff} (\text{MPoly-Type.coeff } p \ m) = \text{lead-coeff} (h \ m)$

by (*simp add: lead-coeff*)

also have $\text{lead-coeff} (h \ m) = \text{coeff} (h \ m) \ 0$

by (*simp add: lead-coeff-def*)

also have $\dots = \text{coeff} (\text{coeff } p \ m) \ 0 * \text{coeff} (\prod i. \text{sym-mpoly } A \ i \ ^{\text{lookup } m \ i}) \ 0$

by (*simp add: h-def mpoly-coeff-times-0*)

also have $(\prod i. \text{sym-mpoly } A \ i \ ^{\text{lookup } m \ i}) = (\prod i=1..n. \text{sym-mpoly } A \ i \ ^{\text{lookup } m \ i})$

lookup m i

by (*intro Prod-any.expand-superset subsetI lookup-nzD*) (*auto intro!: Nat.gr0I*)

also have $\text{coeff} \ \dots \ 0 = (\prod i=1..n. \ 0 \ ^{\text{lookup } m \ i})$

unfolding *mpoly-coeff-prod-0 mpoly-coeff-power-0*

by (*intro prod.cong*) (*auto simp: coeff-sym-mpoly-0*)

also have $\dots = 0 \ ^{\sum i=1..n. \ \text{lookup } m \ i}$

by (*simp add: power-sum*)

also have $\dots = 0$

using *zero-power[OF (sum (lookup m) {1..n} > 0)]* **by** *simp*

finally show *False* **by** *auto*

qed

hence $\text{lookup } m \ k = 0$ **for** k

using *keys-subset*[*of m*] **by** (*cases k \in \{1..n\}*) (*auto simp: in-keys-iff*)

thus $m = 0$ **by** (*intro poly-mapping-eqI*) *auto*

qed

have $0 = \text{insertion} (\text{sym-mpoly } A) \ p$

using *assms* **by** *simp*

also have $\text{insertion} (\text{sym-mpoly } A) \ p = (\sum m \in M. \ h \ m)$

```

    by fact
  also have ... = (∑ m∈{0}. h m)
    using only-zero by (intro sum.mono-neutral-left) (auto simp: h-def M-def)
  also have ... = coeff p 0
    by (simp add: h-def)
  finally have 0 ∉ M by (auto simp: M-def)
  with only-zero have M = {} by auto
  hence p = 0 by (intro mpoly-eqI) (auto simp: M-def)
  with ⟨p ≠ 0⟩ show False by contradiction
qed

```

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 mpoly mpoly
  assumes finite A
    insertion (sym-mpoly A) p = insertion (sym-mpoly A) q
    ∧ m. vars (coeff p m) ∩ A = {} ∧ m. vars (coeff q m) ∩ A = {}
    vars p ⊆ {1..n} vars q ⊆ {1..n}
  shows p = q
proof -
  have p - q = 0
proof (rule sym-mpoly-representation-unique-aux)
  fix m show vars (coeff (p - q) m) ∩ A = {}
    using vars-diff[of coeff p m coeff q m] assms(3,4)[of m] by auto
qed (insert assms vars-diff[of p q], auto simp: insertion-diff)
thus ?thesis by simp
qed

```

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
    ∧ m. vars (coeff q m) ∩ A = {}
    vars q ⊆ {1..n}
  shows q = fund-sym-poly-wit p
  using fund-sym-poly-wit-insertion[of p] fund-sym-poly-wit-vars[of p]
    fund-sym-poly-wit-coeffs[of p]
  by (intro sym-mpoly-representation-unique)
    (insert assms, auto simp: fund-sym-poly-wit-insertion)

```

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**

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

lemma *check-symmetric-mpoly-dom-aux*:
assumes *finite B vars p ⊆ B A ⊆ B*
shows *check-symmetric-mpoly-dom p*
using *assms(1-3)*
proof (*induction p rule: lead-monom-induct*)
case (*less p*)
have [*simp*]: *finite A* **by** (*rule finite-subset[of - B]*) *fact+*
show *?case*
proof (*cases lead-monom p = 0 ∨ ¬decr p*)
case *False*
hence [*simp*]: *p ≠ 0* **by** *auto*
have *vars (p - fund-sym-step-poly p) ⊆ B*
using *fund-sym-step-poly[of p] False less.premis less.hyps (A ⊆ B)*
by (*intro order.trans[OF vars-diff]*) *auto*
hence *check-symmetric-mpoly-dom (p - local.fund-sym-step-poly p)*
using *False less.premis less.hyps*
by (*intro less.IH fund-sym-step-poly symmetric-mpoly-diff*
lead-monom-fund-sym-step-poly-less) (*auto simp: decr-def*)
thus *?thesis* **using** *check-symmetric-mpoly.domintros* **by** *blast*
qed (*auto intro: check-symmetric-mpoly.domintros simp: lead-monom-eq-0-iff*)
qed

lemma *check-symmetric-mpoly-dom [intro]: check-symmetric-mpoly-dom p*
proof –
show *?thesis*
proof (*cases A ⊆ vars p*)
assume *A ⊆ vars p*
thus *?thesis* **using** *check-symmetric-mpoly-dom-aux[of vars p p]* **by** (*auto simp:*
vars-finite)
qed (*auto intro: check-symmetric-mpoly.domintros*)
qed

termination *check-symmetric-mpoly*
by (*intro allI check-symmetric-mpoly-dom*)

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

lemma *check-symmetric-mpoly-correct: check-symmetric-mpoly p ⟷ symmetric-mpoly A p*
proof (*induction p rule: check-symmetric-mpoly.induct*)
case (*1 p*)
have *symmetric-mpoly A (p - fund-sym-step-poly p) ⟷ symmetric-mpoly A p*
(is ?lhs = ?rhs)
proof


```

assume ?rhs
thus ?lhs by (intro symmetric-mpoly-diff fund-sym-step-poly)
next
assume ?lhs
hence symmetric-mpoly A (p - fund-sym-step-poly p + fund-sym-step-poly p)
by (intro symmetric-mpoly-add fund-sym-step-poly)
thus ?rhs by simp
qed
moreover have decr p if symmetric-mpoly A p
using lookup-lead-monom-decreasing[of A p] that by (auto simp: decr-def)
ultimately show check-symmetric-mpoly p  $\longleftrightarrow$  symmetric-mpoly A p
using 1 symmetric-mpoly-imp-orthogonal-or-subset[of A p]
by (auto simp: Let-def check-symmetric-mpoly.simps[of p] intro: symmetric-mpoly-orthogonal)
qed

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 :: comm-ring-1 set
and A :: nat set
and root :: nat  $\Rightarrow$  'a :: comm-ring-1
and l :: 'a  $\Rightarrow$  'b
and q :: 'b mpoly
and n :: nat
defines n  $\equiv$  card A
assumes C: ring-closed C  $\forall$  m. coeff q m  $\in$  C
assumes l: ring-homomorphism l
assumes finite: finite A
assumes sym: symmetric-mpoly A q and vars: vars q  $\subseteq$  A
begin

```

interpretation ring-closed C **by** fact

interpretation ring-homomorphism l **by** fact

theorem symmetric-poly-of-roots-conv-poly-of-coeffs:

assumes c: cinv * l c = 1 cinv \in C

assumes p = Polynomial.smult c ($\prod_{i \in A} [:-root\ i, 1:]$)

obtains q' **where** $\text{vars } q' \subseteq \{0..<n\}$
and $\bigwedge m. \text{coeff } q' m \in C$
and $\text{insertion } (l \circ \text{poly.coeff } p) q' = \text{insertion } (l \circ \text{root}) q$

proof –

define q' **where** $q' = \text{fund-sym-poly-wit } A q$
define q'' **where** $q'' =$
 $\text{mapm-mpoly } (\lambda m x. (\prod i. (\text{cinv} * l (- 1) ^ i) ^ \text{lookup } m i) * \text{insertion } (\lambda .$
 $0) x) q'$
define reindex **where** $\text{reindex} = (\lambda i. \text{if } i \leq n \text{ then } n - i \text{ else } i)$
have bij reindex
by $(\text{intro } \text{bij-betwI}[\text{of reindex - - reindex}]) (\text{auto simp: reindex-def})$
have $\text{vars } q' \subseteq \{1..n\}$ **unfolding** q' -def n -def **by** $(\text{intro fund-sym-poly-wit-vars})$
hence $\text{vars } q'' \subseteq \{1..n\}$
unfolding q'' -def **using** $\text{vars-mapm-mpoly-subset}$ **by** auto

have $\text{insertion } (l \circ \text{root}) (\text{insertion } (\text{sym-mpoly } A) q') =$
 $\text{insertion } (\lambda n. \text{insertion } (l \circ \text{root}) (\text{sym-mpoly } A n))$
 $(\text{map-mpoly } (\text{insertion } (l \circ \text{root})) q')$
by $(\text{rule insertion-insertion})$
also have $\text{insertion } (\text{sym-mpoly } A) q' = q$
unfolding q' -def **by** $(\text{intro fund-sym-poly-wit-insertion sym})$
also have $\text{insertion } (\lambda i. \text{insertion } (l \circ \text{root}) (\text{sym-mpoly } A i))$
 $(\text{map-mpoly } (\text{insertion } (l \circ \text{root})) q') =$
 $\text{insertion } (\lambda i. \text{cinv} * l ((- 1) ^ i) * l (\text{poly.coeff } p (n - i)))$
 $(\text{map-mpoly } (\text{insertion } (l \circ \text{root})) q')$

proof $(\text{intro insertion-irrelevant-vars, goal-cases})$

case $(1 i)$
hence $i \in \text{vars } q'$ **using** $\text{vars-map-mpoly-subset}$ **by** auto
also have $\dots \subseteq \{1..n\}$ **unfolding** q' -def n -def
by $(\text{intro fund-sym-poly-wit-vars})$
finally have $i: i \in \{1..n\}$.
have $\text{insertion } (l \circ \text{root}) (\text{sym-mpoly } A i) =$
 $l (\sum Y \mid Y \subseteq A \wedge \text{card } Y = i. \text{prod root } Y)$
using $\langle \text{finite } A \rangle$ **by** $(\text{simp add: insertion-sym-mpoly})$
also have $\dots = \text{cinv} * l (c * (\sum Y \mid Y \subseteq A \wedge \text{card } Y = i. \text{prod root } Y))$
unfolding $\text{mult mult.assoc}[\text{symmetric}] \langle \text{cinv} * l c = 1 \rangle$ **by** simp
also have $c * (\sum Y \mid Y \subseteq A \wedge \text{card } Y = i. \text{prod root } Y) = ((-1) ^ i * \text{poly.coeff } p (n - i))$
using $\text{coeff-poly-from-roots}[\text{of } A \ n - i \ \text{root}] i \ \text{assms finite}$
by $(\text{auto simp: n-def minus-one-power-iff})$
finally show $?case$ **by** (simp add: o-def)

qed

also have $\text{map-mpoly } (\text{insertion } (l \circ \text{root})) q' = \text{map-mpoly } (\text{insertion } (\lambda . 0))$
 q'
using $\text{fund-sym-poly-wit-coeffs}[\text{OF sym}] \ \text{vars}$
by $(\text{intro map-mpoly-cong insertion-irrelevant-vars}) (\text{auto simp: } q'$ -def)
also have $\text{insertion } (\lambda i. \text{cinv} * l ((- 1) ^ i) * l (\text{poly.coeff } p (n - i))) \dots =$
 $\text{insertion } (\lambda i. l (\text{poly.coeff } p (n - i))) q''$
unfolding $\text{insertion-substitute-linear map-mpoly-conv-mapm-mpoly } q''$ -def

by (*subst mapm-mpoly-comp*) *auto*
also have $\dots = \text{insertion } (l \circ \text{poly.coeff } p) \text{ (mpoly-map-vars reindex } q'')$
using $\langle \text{bij reindex} \rangle$ **and** $\langle \text{vars } q'' \subseteq \{1..n\} \rangle$
by (*subst insertion-mpoly-map-vars*)
(auto simp: o-def reindex-def intro!: insertion-irrelevant-vars)
finally have $\text{insertion } (l \circ \text{root}) q =$
 $\text{insertion } (l \circ \text{poly.coeff } p) \text{ (mpoly-map-vars reindex } q'')$.

moreover have $\text{coeff } (\text{mpoly-map-vars reindex } q'') m \in C$ **for** m
unfolding $q''\text{-def } q'\text{-def}$ **using** $\langle \text{bij reindex} \rangle$ *fund-sym-poly-wit-coeff*[of q C A]
 C $\langle \text{cinv} \in C \rangle$
by (*auto simp: coeff-mpoly-map-vars*)
intro!: mult-closed Prod-any-closed power-closed Sum-any-closed
moreover have $\text{vars } (\text{mpoly-map-vars reindex } q'') \subseteq \{0..<n\}$
using $\langle \text{bij reindex} \rangle$ **and** $\langle \text{vars } q'' \subseteq \{1..n\} \rangle$
by (*subst vars-mpoly-map-vars*) (*auto simp: reindex-def subset-iff*)+
ultimately show *?thesis using that*[of *mpoly-map-vars reindex q''*] **by** *auto*
qed

corollary *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 $\text{insertion } (l \circ \text{poly.coeff } p) q' = \text{insertion } (l \circ \text{root}) q$

proof –

obtain q' **where** $\text{vars } q' \subseteq \{0..<n\}$
and $\bigwedge m. \text{coeff } q' m \in C$
and $\text{insertion } (l \circ \text{poly.coeff } p) q' = \text{insertion } (l \circ \text{root}) q$
by (*rule symmetric-poly-of-roots-conv-poly-of-coeffs*[of 1 1 p])
(use assms in auto)
thus *?thesis by* (*intro that*[of q']) *auto*
qed

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 $\text{cinv} * l c = 1$ $\text{cinv} \in C$

```

assumes  $p = \text{Polynomial.smult } c (\prod_{i \in A}. [-\text{root } i, 1:])$ 
assumes  $\forall i. l (\text{poly.coeff } p \ i) \in C$ 
shows  $\text{insertion } (\lambda x. l (\text{root } x)) \ q \in C$ 
proof –
  obtain  $q'$ 
    where  $q': \text{vars } q' \subseteq \{0..<n\} \wedge m. \text{coeff } q' \ m \in C$ 
       $\text{insertion } (l \circ \text{poly.coeff } p) \ q' = \text{insertion } (l \circ \text{root}) \ q$ 
    by (rule symmetric-poly-of-roots-conv-poly-of-coeffs [of cinv c p])
      (use assms in simp-all)
  have  $\text{insertion } (l \circ \text{poly.coeff } p) \ q' \in C$  using  $C$  assms unfolding insertion-altdef
    by (intro Sum-any-closed mult-closed q' Prod-any-closed power-closed) auto
  also have  $\text{insertion } (l \circ \text{poly.coeff } p) \ q' = \text{insertion } (l \circ \text{root}) \ q$  by fact
  finally show ?thesis by (simp add: o-def)
qed

```

corollary *symmetric-poly-of-roots-in-subring-monic*:

```

assumes  $p = (\prod_{i \in A}. [-\text{root } i, 1:])$ 
assumes  $\forall i. l (\text{poly.coeff } p \ i) \in C$ 
shows  $\text{insertion } (\lambda x. l (\text{root } x)) \ q \in C$ 
proof –
  interpret ring-closed C by fact
  interpret ring-homomorphism l by fact
  show ?thesis
    by (rule symmetric-poly-of-roots-in-subring [of 1 1 p]) (use assms in auto)
qed

```

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  $(\forall x \in S. P \ x) = (\forall x \in T. P \ x)$ 
using assms by auto

```

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

$xs < (ys :: 'a :: \text{linorder} \Rightarrow_0 'b :: \{\text{zero}, \text{linorder}\}) \longleftrightarrow$
 $(\exists i \in \text{keys } xs \cup \text{keys } ys. \text{lookup } xs \ i < \text{lookup } ys \ i \wedge$
 $(\forall j \in \text{keys } xs \cup \text{keys } ys. j < i \longrightarrow \text{lookup } xs \ j = \text{lookup } ys \ j))$

proof *transfer*

fix $f \ g :: 'a \Rightarrow 'b$

let $?dom = \{i. f \ i \neq 0\} \cup \{i. g \ i \neq 0\}$

have $\text{less-fun } f \ g \longleftrightarrow (\exists k. f \ k < g \ k \wedge (\forall k' < k. f \ k' = g \ k'))$

unfolding *less-fun-def ..*

also have $\dots \longleftrightarrow (\exists i. f \ i < g \ i \wedge (i \in ?dom \wedge (\forall j \in ?dom. j < i \longrightarrow f \ j = g \ j)))$

proof (*intro iff-exI conj-cong refl*)

fix k **assume** $f \ k < g \ k$

hence $k: k \in ?dom$ **by** *auto*

have $(\forall k' < k. f \ k' = g \ k') = (\forall k' \in \{..<k\}. f \ k' = g \ k')$

by *auto*

also have $\dots \longleftrightarrow (\forall j \in (\{k. f \ k \neq 0\} \cup \{k. g \ k \neq 0\}) \cap \{..<k\}. f \ j = g \ j)$

by (*intro Ball-subset-right*) *auto*

also have $\dots \longleftrightarrow (\forall j \in (\{k. f \ k \neq 0\} \cup \{k. g \ k \neq 0\}). j < k \longrightarrow f \ j = g \ j)$

by *auto*

finally show $(\forall k' < k. f \ k' = g \ k') \longleftrightarrow k \in ?dom \wedge (\forall j \in ?dom. j < k \longrightarrow f \ j = g \ j)$

using k **by** *simp*

qed

also have $\dots \longleftrightarrow (\exists i \in ?dom. f \ i < g \ i \wedge (\forall j \in ?dom. j < i \longrightarrow f \ j = g \ j))$

by (*simp add: Bex-def conj-ac*)

finally show $\text{less-fun } f \ g \longleftrightarrow (\exists i \in ?dom. f \ i < g \ i \wedge (\forall j \in ?dom. j < i \longrightarrow f \ j = g \ j))$.

qed

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

$xs \leq ys \longleftrightarrow xs = ys \vee xs < (ys :: - \Rightarrow_0 -)$

by (*auto simp: order.order-iff-strict*)

lemma *vars-code* [code]:

$\text{vars } (\text{MPoly } p) = (\bigcup m \in \text{keys } p. \text{keys } m)$

unfolding *vars-def* **by** *transfer' simp*

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

by *transfer' simp*

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

$\text{sym-mpoly } (\text{set } xs) \ k = (\sum X \in \text{Set.filter } (\lambda X. \text{card } X = k) (\text{Pow } (\text{set } xs))).$
 $\text{monom } (\text{monom-of-set } X) \ 1)$

by (*simp add: sym-mpoly-altdef Set.filter-def*)

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

$\text{monom-of-set } (\text{set } xs) = \text{Pm-fmap } (\text{fmap-of-list } (\text{map } (\lambda x. (x, 1)) \ xs))$
 $(\text{is } ?lhs = ?rhs)$

proof (*intro poly-mapping-eqI*)

```

fix k
show lookup ?lhs k = lookup ?rhs k
  by (induction xs) (auto simp: lookup-monom-of-set fmllookup-default-def)
qed

lemma restrictpm-code [code]:
  restrictpm A (Pm-fmap m) = Pm-fmap (fmrestrict-set A m)
  by (intro poly-mapping-eqI) (auto simp: lookup-restrictpm fmllookup-default-def)

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

notepad
begin
  define X Y Z :: int mpoly where X = Var 1 Y = Var 2 Z = Var 3
  define e1 e2 :: int mpoly mpoly where e1 = Var 1 e2 = Var 2
  have sym-mpoly {1, 2, 3} 2 = X * Y + X * Z + Y * Z
    unfolding X-Y-Z-def by eval
  have symmetric-mpoly {1, 2} (X ^ 3 + Y ^ 3)
    unfolding X-Y-Z-def by eval
  have fund-sym-poly-wit {1, 2} (X ^ 3 + Y ^ 3) = e1 ^ 3 - 3 * e1 * e2
    unfolding X-Y-Z-def e1-e2-def by eval
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.