

# Power Sum Polynomials and the Girard–Newton Theorem

Manuel Eberl

February 6, 2026

## Abstract

This article provides a formalisation of the symmetric multivariate polynomials known as *power sum polynomials*. These are of the form  $p_n(X_1, \dots, X_k) = X_1^n + \dots + X_k^n$ . A formal proof of the Girard–Newton Theorem is also given. This theorem relates the power sum polynomials to the elementary symmetric polynomials  $s_k$  in the form of a recurrence relation  $(-1)^k k s_k = \sum_{i=0}^{k-1} (-1)^i s_i p_{k-i}$ .

As an application, this is then used to solve a generalised form of a puzzle given as an exercise in Dummit and Foote’s *Abstract Algebra*: For  $k$  complex unknowns  $x_1, \dots, x_k$ , define  $p_j := x_1^j + \dots + x_k^j$ . Then for each vector  $a \in \mathbb{C}^k$ , show that there is exactly one solution to the system  $p_1 = a_1, \dots, p_k = a_k$  up to permutation of the  $x_i$  and determine the value of  $p_i$  for  $i > k$ .

## Contents

<b>1</b>	<b>Auxiliary material</b>	<b>2</b>
1.1	Miscellaneous . . . . .	2
1.2	The set of roots of a univariate polynomial . . . . .	7
<b>2</b>	<b>Power sum polynomials</b>	<b>13</b>
2.1	Definition . . . . .	13
2.2	The Girard–Newton Theorem . . . . .	16
<b>3</b>	<b>Power sum puzzles</b>	<b>22</b>
3.1	General setting and results . . . . .	22
3.2	Existence of solutions . . . . .	25
3.3	A specific puzzle . . . . .	28

# 1 Auxiliary material

**theory** *Power-Sum-Polynomials-Library*

**imports**

*Polynomial-Factorization.Fundamental-Theorem-Algebra-Factorized*

*Symmetric-Polynomials.Symmetric-Polynomials*

*HOL-Computational-Algebra.Computational-Algebra*

**begin**

**unbundle** *multiset.lifting*

## 1.1 Miscellaneous

**lemma** *atLeastAtMost-nat-numeral:*

*atLeastAtMost m (numeral k :: nat) =*  
*(if m ≤ numeral k then insert (numeral k) (atLeastAtMost m (pred-numeral*  
*k))*  
*else {})*

**by** (*simp add: numeral-eq-Suc atLeastAtMostSuc-conv*)

**lemma** *sum-in-Rats [intro]:*  $(\bigwedge x. x \in A \implies f x \in \mathbf{Q}) \implies \text{sum } f A \in \mathbf{Q}$

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

**lemma** (**in** *monoid-mult*) *prod-list-distinct-conv-prod-set:*

*distinct xs  $\implies$  prod-list (map f xs) = prod f (set xs)*

**by** (*induct xs*) *simp-all*

**lemma** (**in** *monoid-mult*) *interv-prod-list-conv-prod-set-nat:*

*prod-list (map f [m.. $<n$ ]) = prod f (set [m.. $<n$ ])*

**by** (*simp add: prod-list-distinct-conv-prod-set*)

**lemma** (**in** *monoid-mult*) *prod-list-prod-nth:*

*prod-list xs = ( $\prod i = 0..< \text{length } xs. xs ! i$ )*

**using** *interv-prod-list-conv-prod-set-nat [of (!) xs 0 length xs]* **by** (*simp add: map-nth*)

**lemma** *gcd-poly-code-aux-reduce:*

*gcd-poly-code-aux p 0 = normalize p*

*q  $\neq$  0  $\implies$  gcd-poly-code-aux p q = gcd-poly-code-aux q (primitive-part (pseudo-mod p q))*

**by** (*subst gcd-poly-code-aux.simps; simp*)**+**

**lemma** *coprimeI-primes:*

**fixes** *a b :: 'a :: factorial-semiring*

**assumes** *a  $\neq$  0  $\vee$  b  $\neq$  0*

**assumes**  $\bigwedge p. \text{prime } p \implies p \text{ dvd } a \implies p \text{ dvd } b \implies \text{False}$

**shows** *coprime a b*

**proof** (*rule coprimeI*)

```

fix d assume d: d dvd a d dvd b
with assms(1) have [simp]: d ≠ 0 by auto
show is-unit d
proof (rule ccontr)
  assume ¬is-unit d
  then obtain p where p: prime p p dvd d
  using prime-divisor-exists[of d] by auto
  from assms(2)[of p] and p and d show False
  using dvd-trans by auto
qed
qed

lemma coprime-pderiv-imp-squarefree:
  assumes coprime p (pderiv p)
  shows squarefree p
proof (rule squarefreeI)
  fix d assume d: d ^ 2 dvd p
  then obtain q where q: p = d ^ 2 * q
  by (elim dvdE)
  hence d dvd p d dvd pderiv p
  by (auto simp: pderiv-mult pderiv-power-Suc numeral-2-eq-2)
  with assms show is-unit d
  using not-coprimeI by blast
qed

lemma squarefree-field-poly-iff:
  fixes p :: 'a :: {field-char-0, euclidean-ring-gcd, semiring-gcd-mult-normalize} poly
  assumes [simp]: p ≠ 0
  shows squarefree p ↔ coprime p (pderiv p)
proof
  assume squarefree p
  show coprime p (pderiv p)
  proof (rule coprimeI-primess)
    fix d assume d: d dvd p d dvd pderiv p prime d
    from d(1) obtain q where q: p = d * q
    by (elim dvdE)
    from d(2) and q have d dvd q * pderiv d
    by (simp add: pderiv-mult dvd-add-right-iff)
    with ⟨prime d⟩ have d dvd q ∨ d dvd pderiv d
    using prime-dvd-mult-iff by blast
    thus False
  proof
    assume d dvd q
    hence d ^ 2 dvd p
    by (auto simp: q power2-eq-square)
    with ⟨squarefree p⟩ show False
    using d(3) not-prime-unit squarefreeD by blast
  next
    assume d dvd pderiv d

```

**hence** *Polynomial.degree*  $d = 0$  **by** *simp*  
**moreover** **have**  $d \neq 0$  **using**  $d$  **by** *auto*  
**ultimately show** *False*  
**using**  $d(3)$  *is-unit-iff-degree-not-prime-unit* **by** *blast*  
**qed**  
**qed** *auto*  
**qed** (*use coprime-pderiv-imp-squarefree*[*of p*] **in** *auto*)

**lemma** *coprime-pderiv-imp-rsquarefree*:  
**assumes** *coprime* ( $p :: 'a :: \text{field-char-0 poly}$ ) (*pderiv*  $p$ )  
**shows** *rsquarefree*  $p$   
**unfolding** *rsquarefree-roots*  
**proof** *safe*  
**fix**  $x$  **assume** *poly*  $p\ x = 0$  *poly* (*pderiv*  $p$ )  $x = 0$   
**hence**  $[-x, 1:]$  *dvd*  $p$   $[-x, 1:]$  *dvd* *pderiv*  $p$   
**by** (*auto simp: poly-eq-0-iff-dvd*)  
**with** *assms* **have** *is-unit*  $[-x, 1:]$   
**using** *not-coprimeI* **by** *blast*  
**thus** *False* **by** *auto*  
**qed**

**lemma** *poly-of-nat* [*simp*]: *poly* (*of-nat*  $n$ )  $x = \text{of-nat } n$   
**by** (*induction*  $n$ ) *auto*

**lemma** *poly-of-int* [*simp*]: *poly* (*of-int*  $n$ )  $x = \text{of-int } n$   
**by** (*cases*  $n$ ) *auto*

**lemma** *order-eq-0-iff*:  $p \neq 0 \implies \text{order } x\ p = 0 \iff \text{poly } p\ x \neq 0$   
**by** (*auto simp: order-root*)

**lemma** *order-pos-iff*:  $p \neq 0 \implies \text{order } x\ p > 0 \iff \text{poly } p\ x = 0$   
**by** (*auto simp: order-root*)

**lemma** *order-prod*:  
**assumes**  $\bigwedge x. x \in A \implies f\ x \neq 0$   
**shows**  $\text{order } x\ (\prod_{y \in A} f\ y) = (\sum_{y \in A} \text{order } x\ (f\ y))$   
**using** *assms* **by** (*induction*  $A$  *rule: infinite-finite-induct*) (*auto simp: order-mult*)

**lemma** *order-prod-mset*:  
**assumes**  $0 \notin \# A$   
**shows**  $\text{order } x\ (\text{prod-mset } A) = \text{sum-mset } (\text{image-mset } (\text{order } x)\ A)$   
**using** *assms* **by** (*induction*  $A$ ) (*auto simp: order-mult*)

**lemma** *order-prod-list*:  
**assumes**  $0 \notin \text{set } xs$   
**shows**  $\text{order } x\ (\text{prod-list } xs) = \text{sum-list } (\text{map } (\text{order } x)\ xs)$   
**using** *assms* **by** (*induction*  $xs$ ) (*auto simp: order-mult*)

**lemma** *order-power*:  $p \neq 0 \implies \text{order } x\ (p \wedge n) = n * \text{order } x\ p$

by (induction n) (auto simp: order-mult)

**lemma** *smult-0-right* [simp]:  $MPoly\text{-Type.smult } p \ 0 = 0$   
by (transfer, transfer) auto

**lemma** *mult-smult-right* [simp]:  
fixes  $c :: 'a :: comm\text{-semiring-0}$   
shows  $p * MPoly\text{-Type.smult } c \ q = MPoly\text{-Type.smult } c \ (p * q)$   
by (simp add: smult-conv-mult mult-ac)

**lemma** *mapping-single-eq-iff* [simp]:  
 $Poly\text{-Mapping.single } a \ b = Poly\text{-Mapping.single } c \ d \longleftrightarrow b = 0 \wedge d = 0 \vee a = c \wedge b = d$   
by transfer (unfold fun-eq-iff when-def, metis)

**lemma** *monom-of-set-plus-monom-of-set*:  
assumes  $A \cap B = \{\}$  finite  $A$  finite  $B$   
shows  $monom\text{-of-set } A + monom\text{-of-set } B = monom\text{-of-set } (A \cup B)$   
using assms by transfer (auto simp: fun-eq-iff)

**lemma** *mpoly-monom-0-eq-Const*:  $monom \ 0 \ c = Const \ c$   
by (intro mpoly-eqI) (auto simp: coeff-monom when-def mpoly-coeff-Const)

**lemma** *mpoly-Const-0* [simp]:  $Const \ 0 = 0$   
by (intro mpoly-eqI) (auto simp: mpoly-coeff-Const mpoly-coeff-0)

**lemma** *mpoly-Const-1* [simp]:  $Const \ 1 = 1$   
by (intro mpoly-eqI) (auto simp: mpoly-coeff-Const mpoly-coeff-1)

**lemma** *mpoly-Const-uminus*:  $Const \ (-a) = -Const \ a$   
by (intro mpoly-eqI) (auto simp: mpoly-coeff-Const)

**lemma** *mpoly-Const-add*:  $Const \ (a + b) = Const \ a + Const \ b$   
by (intro mpoly-eqI) (auto simp: mpoly-coeff-Const)

**lemma** *mpoly-Const-mult*:  $Const \ (a * b) = Const \ a * Const \ b$   
unfolding *mpoly-monom-0-eq-Const* [symmetric] *mult-monom* by simp

**lemma** *mpoly-Const-power*:  $Const \ (a \wedge n) = Const \ a \wedge n$   
by (induction n) (auto simp: mpoly-Const-mult)

**lemma** *of-nat-mpoly-eq*:  $of\text{-nat } n = Const \ (of\text{-nat } n)$

**proof** (induction n)

case 0

have  $0 = (Const \ 0 :: 'a \ mpoly)$

by (intro mpoly-eqI) (auto simp: mpoly-coeff-Const)

thus ?case

by simp

```

next
  case (Suc n)
  have 1 + Const (of-nat n) = Const (1 + of-nat n)
    by (intro mpoly-eqI) (auto simp: mpoly-coeff-Const mpoly-coeff-1)
  thus ?case
    using Suc by auto
qed

```

```

lemma insertion-of-nat [simp]: insertion f (of-nat n) = of-nat n
  by (simp add: of-nat-mpoly-eq)

```

```

lemma insertion-monom-of-set [simp]:
  insertion f (monom (monom-of-set X) c) = c * (∏ i∈X. f i)
proof (cases finite X)
  case [simp]: True
  have insertion f (monom (monom-of-set X) c) = c * (∏ a. f a ^ (if a ∈ X then 1 else 0))
    by (auto simp: lookup-monom-of-set)
  also have (∏ a. f a ^ (if a ∈ X then 1 else 0)) = (∏ i∈X. f i ^ (if i ∈ X then 1 else 0))
    by (intro Prod-any.expand-superset) auto
  also have ... = (∏ i∈X. f i)
    by (intro prod.cong) auto
  finally show ?thesis .
qed (auto simp: lookup-monom-of-set)

```

```

lemma symmetric-mpoly-symmetric-sum:
  assumes  $\bigwedge \pi. \pi \text{ permutes } A \implies g \pi \text{ permutes } X$ 
  assumes  $\bigwedge x \pi. x \in X \implies \pi \text{ permutes } A \implies \text{mpoly-map-vars } \pi (f x) = f (g \pi x)$ 
  shows symmetric-mpoly A ( $\sum x \in X. f x$ )
  unfolding symmetric-mpoly-def
proof safe
  fix  $\pi$  assume  $\pi: \pi \text{ permutes } A$ 
  have mpoly-map-vars  $\pi$  (sum f X) = ( $\sum x \in X. \text{mpoly-map-vars } \pi (f x)$ )
    by simp
  also have ... = ( $\sum x \in X. f (g \pi x)$ )
    by (intro sum.cong assms  $\pi$  refl)
  also have ... = ( $\sum x \in g \pi' X. f x$ )
    using assms(1)[OF  $\pi$ ] by (subst sum.reindex) (auto simp: permutes-inj-on)
  also have  $g \pi' X = X$ 
    using assms(1)[OF  $\pi$ ] by (simp add: permutes-image)
  finally show mpoly-map-vars  $\pi$  (sum f X) = sum f X .
qed

```

```

lemma sym-mpoly-0 [simp]:
  assumes finite A

```

**shows**  $\text{sym-mpoly } A \ 0 = 1$   
**using** *assms* **by** (*transfer*, *transfer*) (*auto simp: fun-eq-iff when-def*)

**lemma** *sym-mpoly-eq-0* [*simp*]:

**assumes**  $k > \text{card } A$

**shows**  $\text{sym-mpoly } A \ k = 0$

**proof** (*transfer fixing: A k, transfer fixing: A k, intro ext*)

**fix** *mon*

**have**  $\neg(\text{finite } A \wedge (\exists Y \subseteq A. \text{card } Y = k \wedge \text{mon} = \text{monom-of-set } Y))$

**proof** *safe*

**fix** *Y* **assume**  $Y: \text{finite } A \ Y \subseteq A \ k = \text{card } Y \ \text{mon} = \text{monom-of-set } Y$

**hence**  $\text{card } Y \leq \text{card } A$  **by** (*intro card-mono*) *auto*

**with** *Y* **and** *assms* **show** *False* **by** *simp*

**qed**

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

**by** *auto*

**qed**

**lemma** *coeff-sym-mpoly-monom-of-set-eq-0*:

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

**shows**  $\text{MPoly-Type.coeff } (\text{sym-mpoly } X \ k) (\text{monom-of-set } Y) = 0$

**using** *assms* *finite-subset[of - X]* **by** (*auto simp: coeff-sym-mpoly*)

**lemma** *coeff-sym-mpoly-monom-of-set-eq-0'*:

**assumes**  $\text{finite } X \ \neg Y \subseteq X \ \text{finite } Y$

**shows**  $\text{MPoly-Type.coeff } (\text{sym-mpoly } X \ k) (\text{monom-of-set } Y) = 0$

**using** *assms* *finite-subset[of - X]* **by** (*auto simp: coeff-sym-mpoly*)

## 1.2 The set of roots of a univariate polynomial

**lift-definition** *poly-roots* ::  $'a :: \text{idom poly} \Rightarrow 'a \ \text{multiset}$  **is**

$\lambda p \ x. \ \text{if } p = 0 \ \text{then } 0 \ \text{else } \text{order } x \ p$

**proof** –

**fix**  $p :: 'a \ \text{poly}$

**show**  $\text{finite } \{x. \ 0 < (\text{if } p = 0 \ \text{then } 0 \ \text{else } \text{order } x \ p)\}$

**by** (*cases p = 0*) (*auto simp: order-pos-iff poly-roots-finite*)

**qed**

**lemma** *poly-roots-0* [*simp*]:  $\text{poly-roots } 0 = \{\#\}$

**by** *transfer auto*

**lemma** *poly-roots-1* [*simp*]:  $\text{poly-roots } 1 = \{\#\}$

**by** *transfer auto*

**lemma** *count-poly-roots* [*simp*]:

**assumes**  $p \neq 0$

**shows**  $\text{count } (\text{poly-roots } p) \ x = \text{order } x \ p$

**using** *assms* **by** *transfer auto*

**lemma** *in-poly-roots-iff* [simp]:  $p \neq 0 \implies x \in \# \text{ poly-roots } p \iff \text{poly } p \ x = 0$   
**by** (*subst count-greater-zero-iff* [symmetric], *subst count-poly-roots*) (*auto simp: order-pos-iff*)

**lemma** *set-mset-poly-roots*:  $p \neq 0 \implies \text{set-mset } (\text{poly-roots } p) = \{x. \text{poly } p \ x = 0\}$   
**using** *in-poly-roots-iff*[of *p*] **by** *blast*

**lemma** *count-poly-roots'*:  $\text{count } (\text{poly-roots } p) \ x = (\text{if } p = 0 \text{ then } 0 \text{ else } \text{order } x \ p)$   
**by** *transfer' auto*

**lemma** *poly-roots-const* [simp]:  $\text{poly-roots } [:c:] = \{\#\}$   
**by** (*intro multiset-eqI*) (*auto simp: count-poly-roots' order-eq-0-iff*)

**lemma** *poly-roots-linear* [simp]:  $\text{poly-roots } [-x, 1:] = \{\#x\#\}$   
**by** (*intro multiset-eqI*) (*auto simp: count-poly-roots' order-eq-0-iff*)

**lemma** *poly-roots-monom* [simp]:  $c \neq 0 \implies \text{poly-roots } (\text{Polynomial.monom } c \ n) = \text{replicate-mset } n \ 0$   
**by** (*intro multiset-eqI*) (*auto simp: count-poly-roots' order-eq-0-iff poly-monom*)

**lemma** *poly-roots-smult* [simp]:  $c \neq 0 \implies \text{poly-roots } (\text{Polynomial.smult } c \ p) = \text{poly-roots } p$   
**by** (*intro multiset-eqI*) (*auto simp: count-poly-roots' order-smult*)

**lemma** *poly-roots-mult*:  $p \neq 0 \implies q \neq 0 \implies \text{poly-roots } (p * q) = \text{poly-roots } p + \text{poly-roots } q$   
**by** (*intro multiset-eqI*) (*auto simp: count-poly-roots' order-mult*)

**lemma** *poly-roots-prod*:  
**assumes**  $\bigwedge x. x \in A \implies f \ x \neq 0$   
**shows**  $\text{poly-roots } (\text{prod } f \ A) = (\sum x \in A. \text{poly-roots } (f \ x))$   
**using** *assms* **by** (*induction A* *rule: infinite-finite-induct*) (*auto simp: poly-roots-mult*)

**lemma** *poly-roots-prod-mset*:  
**assumes**  $0 \notin \# A$   
**shows**  $\text{poly-roots } (\text{prod-mset } A) = \text{sum-mset } (\text{image-mset } \text{poly-roots } A)$   
**using** *assms* **by** (*induction A*) (*auto simp: poly-roots-mult*)

**lemma** *poly-roots-prod-list*:  
**assumes**  $0 \notin \text{set } xs$   
**shows**  $\text{poly-roots } (\text{prod-list } xs) = \text{sum-list } (\text{map } \text{poly-roots } xs)$   
**using** *assms* **by** (*induction xs*) (*auto simp: poly-roots-mult*)

**lemma** *poly-roots-power*:  $p \neq 0 \implies \text{poly-roots } (p \ ^n) = \text{repeat-mset } n \ (\text{poly-roots } p)$   
**by** (*induction n*) (*auto simp: poly-roots-mult*)

**lemma** *rsquarefree-poly-roots-eq*:

```

assumes rsquarefree p
shows poly-roots p = mset-set {x. poly p x = 0}
proof (rule multiset-eqI)
  fix x :: 'a
  from assms show count (poly-roots p) x = count (mset-set {x. poly p x = 0}) x
    by (cases poly p x = 0) (auto simp: poly-roots-finite order-eq-0-iff rsquare-free-def)
qed

```

```

lemma rsquarefree-imp-distinct-roots:
  assumes rsquarefree p and mset xs = poly-roots p
  shows distinct xs
proof (cases p = 0)
  case [simp]: False
  have *: mset xs = mset-set {x. poly p x = 0}
    using assms by (simp add: rsquarefree-poly-roots-eq)
  hence set-mset (mset xs) = set-mset (mset-set {x. poly p x = 0})
    by (simp only:)
  hence [simp]: set xs = {x. poly p x = 0}
    by (simp add: poly-roots-finite)
  from * show ?thesis
  by (subst distinct-count-atmost-1) (auto simp: poly-roots-finite)
qed (use assms in auto)

```

```

lemma poly-roots-factorization:
  fixes p c A
  assumes [simp]: c ≠ 0
  defines p ≡ Polynomial.smult c (prod-mset (image-mset ( $\lambda x$ . [ $-x$ , 1:] ) A))
  shows poly-roots p = A
proof –
  have poly-roots p = poly-roots ( $\prod_{x \in \#A}$  [ $-x$ , 1:])
    by (auto simp: p-def)
  also have ... = A
    by (subst poly-roots-prod-mset) (auto simp: image-mset.compositionality o-def)
  finally show ?thesis .
qed

```

```

lemma fundamental-theorem-algebra-factorized':
  fixes p :: complex poly
  shows p = Polynomial.smult (Polynomial.lead-coeff p)
    (prod-mset (image-mset ( $\lambda x$ . [ $-x$ , 1:] ) (poly-roots p)))
proof (cases p = 0)
  case [simp]: False
  obtain xs where
    xs: Polynomial.smult (Polynomial.lead-coeff p) ( $\prod_{x \leftarrow xs}$  [ $-x$ , 1:]) = p
    length xs = Polynomial.degree p
  using fundamental-theorem-algebra-factorized[of p] by auto
  define A where A = mset xs

```

**note**  $xs(1)$   
**also have**  $(\prod x \leftarrow xs. [-x, 1:]) = \text{prod-mset } (\text{image-mset } (\lambda x. [-x, 1:]) A)$   
**unfolding**  $A\text{-def}$  **by**  $(\text{induction } xs)$  **auto**  
**finally have**  $*$ :  $\text{Polynomial.smult } (\text{Polynomial.lead-coeff } p) (\prod x \in \#A. [-x, 1:])$   
 $= p$  .  
**also have**  $A = \text{poly-roots } p$   
**using**  $\text{poly-roots-factorization}$ [of  $\text{Polynomial.lead-coeff } p A$ ]  
**by**  $(\text{subst } * [\text{symmetric}])$  **auto**  
**finally show**  $?thesis$  ..  
**qed**  $\text{auto}$

**lemma**  $\text{poly-roots-eq-imp-eq}$ :  
**fixes**  $p q :: \text{complex poly}$   
**assumes**  $\text{Polynomial.lead-coeff } p = \text{Polynomial.lead-coeff } q$   
**assumes**  $\text{poly-roots } p = \text{poly-roots } q$   
**shows**  $p = q$   
**proof**  $(\text{cases } p = 0 \vee q = 0)$   
**case**  $\text{False}$   
**hence**  $[\text{simp}]$ :  $p \neq 0 \ q \neq 0$   
**by**  $\text{auto}$   
**have**  $p = \text{Polynomial.smult } (\text{Polynomial.lead-coeff } p)$   
 $(\text{prod-mset } (\text{image-mset } (\lambda x. [-x, 1:]) (\text{poly-roots } p)))$   
**by**  $(\text{rule fundamental-theorem-algebra-factorized}' )$   
**also have**  $\dots = \text{Polynomial.smult } (\text{Polynomial.lead-coeff } q)$   
 $(\text{prod-mset } (\text{image-mset } (\lambda x. [-x, 1:]) (\text{poly-roots } q)))$   
**by**  $(\text{simp add: assms})$   
**also have**  $\dots = q$   
**by**  $(\text{rule fundamental-theorem-algebra-factorized}' [\text{symmetric}])$   
**finally show**  $?thesis$  .  
**qed**  $(\text{use assms in auto})$

**lemma**  $\text{Sum-any-zeroI}'$ :  $(\bigwedge x. P x \implies f x = 0) \implies \text{Sum-any } (\lambda x. f x \text{ when } P x)$   
 $= 0$   
**by**  $(\text{auto simp: Sum-any.expand-set})$

**lemma**  $\text{sym-mpoly-insert}$ :  
**assumes**  $\text{finite } X \ x \notin X$   
**shows**  $(\text{sym-mpoly } (\text{insert } x X) (\text{Suc } k) :: 'a :: \text{semiring-1 mpoly}) =$   
 $\text{monom } (\text{monom-of-set } \{x\}) 1 * \text{sym-mpoly } X k + \text{sym-mpoly } X (\text{Suc } k)$  **(is**  $?lhs = ?A + ?B)$   
**proof**  $(\text{rule mpoly-eqI})$   
**fix**  $\text{mon}$   
**show**  $\text{coeff } ?lhs \ \text{mon} = \text{coeff } (?A + ?B) \ \text{mon}$   
**proof**  $(\text{cases } \forall i. \text{lookup } \text{mon } i \leq 1 \wedge (i \notin \text{insert } x X \longrightarrow \text{lookup } \text{mon } i = 0))$   
**case**  $\text{False}$   
**then obtain**  $i$  **where**  $i: \text{lookup } \text{mon } i > 1 \vee i \notin \text{insert } x X \wedge \text{lookup } \text{mon } i >$   
 $0$   
**by**  $(\text{auto simp: not-le})$



```

    qed
  qed
  finally have coeff ?A mon = 0 .
  moreover from False have coeff ?lhs mon = 0
    by (subst coeff-sym-mpoly) (auto simp: lookup-monom-of-set split: if-splits)
  moreover from False have coeff (sym-mpoly X (Suc k)) mon = 0
    by (subst coeff-sym-mpoly) (auto simp: lookup-monom-of-set split: if-splits)
  ultimately show ?thesis
    by auto
next
case True
define A where A = keys mon
have A: A ⊆ insert x X
  using True by (auto simp: A-def)
have [simp]: mon = monom-of-set A
  unfolding A-def using True by transfer (force simp: fun-eq-iff le-Suc-eq)
have finite A
  using finite-subset A assms by blast
show ?thesis
proof (cases x ∈ A)
case False
  have coeff ?A mon = prod-fun (coeff (monom (monom-of-set {x}) 1))
    (coeff (sym-mpoly X k)) (monom-of-set A)
    by (simp add: coeff-mpoly-times)
  also have ... = (∑ l. ∑ q. coeff (monom (monom-of-set {x}) 1) l * coeff
    (sym-mpoly X k) q
    when monom-of-set A = l + q)
    unfolding prod-fun-def
    by (intro Sum-any.cong, subst Sum-any-right-distrib, force)
    (auto simp: Sum-any-right-distrib when-def intro!: Sum-any.cong)
  also have ... = 0
proof (rule Sum-any-zeroI, rule Sum-any-zeroI')
  fix ma mb assume *: monom-of-set A = ma + mb
  hence keys ma ⊆ A
    using ⟨finite A⟩ by transfer (auto simp: fun-eq-iff split: if-splits)
  thus coeff (monom (monom-of-set {x}) (1::'a)) ma * coeff (sym-mpoly X
k) mb = 0
    using ⟨x ∉ A⟩ by (auto simp: coeff-monom when-def)
  qed
  finally show ?thesis
    using False A assms finite-subset[of - insert x X] finite-subset[of - X]
    by (auto simp: coeff-sym-mpoly)
next
case True
  have mon = monom-of-set {x} + monom-of-set (A - {x})
    using ⟨x ∈ A⟩ ⟨finite A⟩ by (auto simp: monom-of-set-plus-monom-of-set)
  also have coeff ?A ... = coeff (sym-mpoly X k) (monom-of-set (A - {x}))
    by (subst coeff-monom-mult) auto
  also have ... = (if card A = Suc k then 1 else 0)

```

```

proof (cases card A = Suc k)
  case True
  thus ?thesis
    using assms ⟨finite A⟩ ⟨x ∈ A⟩ A
    by (subst coeff-sym-mpoly-monom-of-set) auto
next
  case False
  thus ?thesis
    using assms ⟨x ∈ A⟩ A ⟨finite A⟩ card-Suc-Diff1[of A x]
    by (subst coeff-sym-mpoly-monom-of-set-eq-0) auto
qed
moreover have coeff ?B (monom-of-set A) = 0
  using assms ⟨x ∈ A⟩ ⟨finite A⟩
  by (subst coeff-sym-mpoly-monom-of-set-eq-0') auto
moreover have coeff ?lhs (monom-of-set A) = (if card A = Suc k then 1 else
0)
  using assms A ⟨finite A⟩ finite-subset[of - insert x X] by (auto simp:
coeff-sym-mpoly)
  ultimately show ?thesis by simp
qed
qed
qed

lifting-update multiset.lifting
lifting-forget multiset.lifting

end

```

## 2 Power sum polynomials

```

theory Power-Sum-Polynomials
imports
  Symmetric-Polynomials.Symmetric-Polynomials
  HOL-Computational-Algebra.Field-as-Ring
  Power-Sum-Polynomials-Library
begin

```

### 2.1 Definition

For  $n$  indeterminates  $X_1, \dots, X_n$ , we define the  $k$ -th power sum polynomial as

$$p_k(X_1, \dots, X_n) = X_1^k + \dots + X_n^k.$$

**lift-definition** *powsum-mpoly-aux* :: *nat set* ⇒ *nat* ⇒ (*nat* ⇒<sub>0</sub> *nat*) ⇒<sub>0</sub> 'a :: {*semiring-1, zero-neq-one*} **is**

```

  λX k mon. if infinite X ∨ k = 0 ∧ mon ≠ 0 then 0
    else if k = 0 ∧ mon = 0 then of-nat (card X)
    else if finite X ∧ (∃ x ∈ X. mon = Poly-Mapping.single x k) then 1 else 0

```

**by** *auto*

**lemma** *lookup-powsum-mpoly-aux*:

*Poly-Mapping.lookup* (*powsum-mpoly-aux*  $X$   $k$ ) *mon* =  
 (if *infinite*  $X \vee k = 0 \wedge \text{mon} \neq 0$  then 0  
 else if  $k = 0 \wedge \text{mon} = 0$  then *of-nat* (*card*  $X$ )  
 else if *finite*  $X \wedge (\exists x \in X. \text{mon} = \text{Poly-Mapping.single } x \ k)$  then 1 else  
 0)  
**by** *transfer' simp*

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

**assumes** *finite*  $X$   $x \in X$   $k > 0$   
**shows** *Poly-Mapping.lookup* (*powsum-mpoly-aux*  $X$   $k$ ) (*Poly-Mapping.single*  $x$   
 $k$ ) = 1  
**using** *assms* **by** (*auto simp: lookup-powsum-mpoly-aux*)

**lemma** *lookup-sym-mpoly-aux-monom-singleton'*:

**assumes** *finite*  $X$   $k > 0$   
**shows** *Poly-Mapping.lookup* (*powsum-mpoly-aux*  $X$   $k$ ) (*Poly-Mapping.single*  $x$   
 $k$ ) = (if  $x \in X$  then 1 else 0)  
**using** *assms* **by** (*auto simp: lookup-powsum-mpoly-aux*)

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

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

**lift-definition** *powsum-mpoly* :: *nat set*  $\Rightarrow$  *nat*  $\Rightarrow$  'a :: {*semiring-1, zero-neq-one*}  
*mpoly* **is**

*powsum-mpoly-aux* .

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

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

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

**by** (*transfer, transfer*) *auto*

**lemma** *coeff-powsum-mpoly*:

*MPoly-Type.coeff* (*powsum-mpoly*  $X$   $k$ ) *mon* =  
 (if *infinite*  $X \vee k = 0 \wedge \text{mon} \neq 0$  then 0  
 else if  $k = 0 \wedge \text{mon} = 0$  then *of-nat* (*card*  $X$ )  
 else if *finite*  $X \wedge (\exists x \in X. \text{mon} = \text{Poly-Mapping.single } x \ k)$  then 1 else  
 0)  
**by** *transfer' (simp add: lookup-powsum-mpoly-aux)*

**lemma** *coeff-powsum-mpoly-0-right*:

*MPoly-Type.coeff* (*powsum-mpoly*  $X$  0) *mon* = (if  $\text{mon} = 0$  then *of-nat* (*card*  $X$ )  
else 0)  
**by** *transfer' (auto simp add: lookup-powsum-mpoly-aux)*

**lemma** *coeff-powsum-mpoly-singleton*:  
**assumes** *finite X k > 0*  
**shows**  $MPoly\text{-}Type.coeff (powsum\text{-}mpoly\ X\ k) (Poly\text{-}Mapping.single\ x\ k) = (if\ x \in X\ then\ 1\ else\ 0)$   
**using** *assms* **by** *transfer' (simp add: lookup-powsum-mpoly-aux)*

**lemma** *coeff-powsum-mpoly-singleton-eq-1 [simp]*:  
**assumes** *finite X x ∈ X k > 0*  
**shows**  $MPoly\text{-}Type.coeff (powsum\text{-}mpoly\ X\ k) (Poly\text{-}Mapping.single\ x\ k) = 1$   
**using** *assms* **by** *(simp add: coeff-powsum-mpoly-singleton)*

**lemma** *coeff-powsum-mpoly-singleton-eq-0 [simp]*:  
**assumes** *finite X x ∉ X k > 0*  
**shows**  $MPoly\text{-}Type.coeff (powsum\text{-}mpoly\ X\ k) (Poly\text{-}Mapping.single\ x\ k) = 0$   
**using** *assms* **by** *(simp add: coeff-powsum-mpoly-singleton)*

**lemma** *powsum-mpoly-0 [simp]*:  $powsum\text{-}mpoly\ X\ 0 = of\text{-}nat (card\ X)$   
**by** *(intro mpoly-eqI ext) (auto simp: coeff-powsum-mpoly-0-right of-nat-mpoly-eq poly-coeff-Const)*

**lemma** *powsum-mpoly-empty [simp]*:  $powsum\text{-}mpoly\ \{\}\ k = 0$   
**by** *(intro mpoly-eqI) (auto simp: coeff-powsum-mpoly)*

**lemma** *powsum-mpoly-altdef*:  $powsum\text{-}mpoly\ X\ k = (\sum_{x \in X}. monom (Poly\text{-}Mapping.single\ x\ k)\ 1)$

**proof** *(cases finite X)*  
**case** *[simp]: True*  
**show** *?thesis*  
**proof** *(cases k = 0)*  
**case** *True*  
**thus** *?thesis* **by** *auto*  
**next**  
**case** *False*  
**show** *?thesis*  
**proof** *(intro mpoly-eqI, goal-cases)*  
**case** *(1 mon)*  
**show** *?case* **using** *False*  
**by** *(cases ∃ x ∈ X. mon = Poly-Mapping.single x k)*  
*(auto simp: coeff-powsum-mpoly coeff-monom when-def)*  
**qed**  
**qed**  
**qed** *(auto simp: powsum-mpoly-infinite)*

Power sum polynomials are symmetric:

**lemma** *symmetric-powsum-mpoly [intro]*:  
**assumes**  $A \subseteq B$   
**shows** *symmetric-mpoly A (powsum-mpoly B k)*  
**unfolding** *powsum-mpoly-altdef*

**proof** (*rule symmetric-mpoly-symmetric-sum*)  
**fix**  $x \pi$   
**assume**  $x \in B \pi$  *permutes*  $A$   
**thus**  $\text{mpoly-map-vars } \pi$  ( $\text{MPoly-Type.monom } (\text{Poly-Mapping.single } x \ k) \ 1$ ) =  
 $\text{MPoly-Type.monom } (\text{Poly-Mapping.single } (\pi \ x) \ k) \ 1$   
**using** *assms* **by** (*auto simp: mpoly-map-vars-monom permutes-bij permutep-single*  
*bij-imp-bij-inv permutes-inv-inv*)  
**qed** (*use assms in <auto simp: permutes-subset>*)

**lemma** *insertion-powsum-mpoly [simp]: insertion f (powsum-mpoly X k) = ( $\sum_{i \in X} f \ i \ ^k$ )*  
**unfolding** *powsum-mpoly-altdef insertion-sum insertion-single* **by** *simp*

**lemma** *powsum-mpoly-nz:*  
**assumes** *finite X X  $\neq \{\}$  k > 0*  
**shows** ( $\text{powsum-mpoly } X \ k :: 'a :: \{\text{semiring-1, zero-neq-one}\} \text{mpoly}$ )  $\neq 0$   
**proof** –  
**from** *assms* **obtain**  $x$  **where**  $x \in X$  **by** *auto*  
**hence**  $\text{coeff } (\text{powsum-mpoly } X \ k) (\text{Poly-Mapping.single } x \ k) = (1 :: 'a)$   
**using** *assms* **by** (*auto simp: coeff-powsum-mpoly*)  
**thus** *?thesis* **by** *auto*  
**qed**

**lemma** *powsum-mpoly-eq-0-iff:*  
**assumes**  $k > 0$   
**shows**  $\text{powsum-mpoly } X \ k = 0 \iff \text{infinite } X \vee X = \{\}$   
**using** *assms* *powsum-mpoly-nz[of X k]* **by** (*auto simp: powsum-mpoly-infinite*)

## 2.2 The Girard–Newton Theorem

The following is a nice combinatorial proof of the Girard–Newton Theorem due to Doron Zeilberger [2].

The precise statement is this:

Let  $e_k$  denote the  $k$ -th elementary symmetric polynomial in  $X_1, \dots, X_n$ . This is the sum of all monomials that can be formed by taking the product of  $k$  distinct variables.

Next, let  $p_k = X_1^k + \dots + X_n^k$  denote that  $k$ -th symmetric power sum polynomial in  $X_1, \dots, X_n$ .

Then the following equality holds:

$$(-1)^k k e_k + \sum_{i=0}^{k-1} (-1)^i e_i p_{k-i}$$

**theorem** *Girard-Newton:*

**assumes** *finite X*

**shows**  $(-1)^k \wedge k * \text{of-nat } k * \text{sym-mpoly } X \ k +$

$(\sum_{i < k} (-1)^i * \text{sym-mpoly } X \ i * \text{powsum-mpoly } X \ (k - i)) =$

```

      (0 :: 'a :: comm-ring-1 mpoly)
    (is ?lhs = 0)
  proof -
    write Poly-Mapping.single (⟨sng⟩)

    define n where n = card X
    define A :: (nat set × nat) set
      where A = {(A, j). A ⊆ X ∧ card A ≤ k ∧ j ∈ X ∧ (card A = k → j ∈ A)}
    define A1 :: (nat set × nat) set
      where A1 = {A ∈ Pow X. card A < k} × X
    define A2 :: (nat set × nat) set
      where A2 = (SIGMA A: {A ∈ Pow X. card A = k}. A)

    have A-split: A = A1 ∪ A2 A1 ∩ A2 = {}
      by (auto simp: A-def A1-def A2-def)
    have [intro]: finite A1 finite A2
      using assms finite-subset[of - X] by (auto simp: A1-def A2-def intro!: fi-
nite-SigmaI)
    have [intro]: finite A
      by (subst A-split) auto

```

— We define a ‘weight’ function  $w$  from  $\mathcal{A}$  to the ring of polynomials as

$$w(A, j) = (-1)^{|A|} x_j^{k-|A|} \prod_{i \in A} x_i .$$

```

define w :: nat set × nat ⇒ 'a mpoly
  where w = (λ(A, j). monom (monom-of-set A + sng j (k - card A)) ((-1) ^
card A))

```

— The sum of these weights over all of  $\mathcal{A}$  is precisely the sum that we want to show equals 0:

```

have ?lhs = (∑ x ∈ A. w x)
proof -
  have (∑ x ∈ A. w x) = (∑ x ∈ A1. w x) + (∑ x ∈ A2. w x)
    by (subst A-split, subst sum.union-disjoint, use A-split(2) in auto)

  also have (∑ x ∈ A1. w x) = (∑ i < k. (-1) ^ i * sym-mpoly X i * powsum-mpoly
X (k - i))
  proof -
    have (∑ x ∈ A1. w x) = (∑ A | A ⊆ X ∧ card A < k. ∑ j ∈ X. w (A, j))
      using assms by (subst sum.Sigma) (auto simp: A1-def)
    also have ... = (∑ A | A ⊆ X ∧ card A < k. ∑ j ∈ X.
      monom (monom-of-set A) ((-1) ^ card A) * monom (sng j (k
- card A)) 1)
      unfolding w-def by (intro sum.cong) (auto simp: mult-monom)
    also have ... = (∑ A | A ⊆ X ∧ card A < k. monom (monom-of-set A)
((-1) ^ card A) *
      powsum-mpoly X (k - card A))
      by (simp add: sum-distrib-left powsum-mpoly-altdef)

```

**also have**  $\dots = (\sum (i,A) \in (\text{SIGMA } i:\{..<k\}. \{A. A \subseteq X \wedge \text{card } A = i\}).$   
 $\text{monom } (\text{monom-of-set } A) ((-1) \wedge i) * \text{powsum-mpoly } X (k -$   
 $i))$   
**by** (*rule sum.reindex-bij-witness*[of - snd  $\lambda A. (\text{card } A, A)$ ]) *auto*  
**also have**  $\dots = (\sum i < k. \sum A \mid A \subseteq X \wedge \text{card } A = i.$   
 $\text{monom } (\text{monom-of-set } A) 1 * \text{monom } 0 ((-1) \wedge i) * \text{powsum-mpoly } X (k -$   
 $i))$   
**using** *assms* **by** (*subst sum.Sigma*) (*auto simp: mult-monom*)  
**also have**  $\dots = (\sum i < k. (-1) \wedge i * \text{sym-mpoly } X i * \text{powsum-mpoly } X (k -$   
 $i))$   
**by** (*simp add: sum-distrib-left sum-distrib-right mpoly-monom-0-eq-Const*  
 $\text{mpoly-Const-power mpoly-Const-uminus algebra-simps}$   
 $\text{sym-mpoly-altdef}$ )  
**finally show** *?thesis* .  
**qed**

**also have**  $(\sum x \in \mathcal{A}2. w x) = (-1) \wedge k * \text{of-nat } k * \text{sym-mpoly } X k$   
**proof** –  
**have**  $(\sum x \in \mathcal{A}2. w x) = (\sum (A,j) \in \mathcal{A}2. \text{monom } (\text{monom-of-set } A) ((-1) \wedge$   
 $k))$   
**by** (*intro sum.cong*) (*auto simp: A2-def w-def mpoly-monom-0-eq-Const*  
 $\text{intro!: sum.cong}$ )  
**also have**  $\dots = (\sum A \mid A \subseteq X \wedge \text{card } A = k. \sum j \in A. \text{monom } (\text{monom-of-set}$   
 $A) ((-1) \wedge k))$   
**using** *assms finite-subset*[of -  $X$ ] **by** (*subst sum.Sigma*) (*auto simp: A2-def*)  
**also have**  $(\lambda A. \text{monom } (\text{monom-of-set } A) ((-1) \wedge k) :: 'a \text{ mpoly}) =$   
 $(\lambda A. \text{monom } 0 ((-1) \wedge k) * \text{monom } (\text{monom-of-set } A) 1)$   
**by** (*auto simp: fun-eq-iff mult-monom*)  
**also have**  $\text{monom } 0 ((-1) \wedge k) = (-1) \wedge k$   
**by** (*auto simp: mpoly-monom-0-eq-Const mpoly-Const-power mpoly-Const-uminus*)  
**also have**  $(\sum A \mid A \subseteq X \wedge \text{card } A = k. \sum j \in A. (-1) \wedge k * \text{monom}$   
 $(\text{monom-of-set } A) 1) =$   
 $((-1) \wedge k * \text{of-nat } k * \text{sym-mpoly } X k :: 'a \text{ mpoly})$   
**by** (*auto simp: sum-distrib-left sum-distrib-right mult-ac sym-mpoly-altdef*)  
**finally show** *?thesis* .  
**qed**

**finally show** *?thesis* **by** (*simp add: algebra-simps*)  
**qed**

— Next, we show that the weights sum to 0:

**also have**  $(\sum x \in \mathcal{A}. w x) = 0$

**proof** –

— We define a function  $T$  that is a involutory permutation of  $\mathcal{A}$ . To be more precise, it bijectively maps those elements  $(A,j)$  of  $\mathcal{A}$  with  $j \in A$  to those where  $j \notin A$  and the other way round. ‘Involutory’ means that  $T$  is its own inverse function, i. e.  $T(T(x)) = x$ .

**define**  $T :: \text{nat set} \times \text{nat} \Rightarrow \text{nat set} \times \text{nat}$

**where**  $T = (\lambda(A, j). \text{if } j \in A \text{ then } (A - \{j\}, j) \text{ else } (\text{insert } j A, j))$

**have**  $[simp]: T (T x) = x$  **for**  $x$   
**by**  $(auto simp: T-def split: prod.splits)$   
**have**  $[simp]: T x \in \mathcal{A}$  **if**  $x \in \mathcal{A}$  **for**  $x$   
**proof** –  
**have**  $[simp]: n \leq n - Suc\ 0 \iff n = 0$  **for**  $n$   
**by**  $auto$   
**show**  $?thesis$  **using**  $that$  **assms**  $finite-subset[of\ -\ X]$   
**by**  $(auto simp: T-def \mathcal{A}-def split: prod.splits)$   
**qed**  
**have**  $snd (T x) \in fst (T x) \iff snd\ x \notin fst\ x$  **if**  $x \in \mathcal{A}$  **for**  $x$   
**by**  $(auto simp: T-def split: prod.splits)$   
**hence**  $bij: bij\ betw\ T\ \{x \in \mathcal{A}. snd\ x \in fst\ x\}\ \{x \in \mathcal{A}. snd\ x \notin fst\ x\}$   
**by**  $(intro\ bij\ betwI[of\ -\ -\ T])\ auto$

— Crucially, we show that  $T$  flips the weight of each element:

**have**  $[simp]: w (T x) = -w\ x$  **if**  $x \in \mathcal{A}$  **for**  $x$

**proof** –

**obtain**  $A\ j$  **where**  $[simp]: x = (A, j)$  **by**  $force$

— Since  $T$  is an involution, we can assume w.l.o.g. that  $j \in A$ :

**have**  $aux: w (T (A, j)) = -w (A, j)$  **if**  $(A, j) \in \mathcal{A}$   $j \in A$  **for**  $j\ A$

**proof** –

**from**  $that$  **have**  $[simp]: j \in A\ A \subseteq X$  **and**  $k > 0$

**using**  $finite-subset[OF\ -\ assms, of\ A]$  **by**  $(auto\ simp: \mathcal{A}-def\ intro!: Nat.gr0I)$

**have**  $[simp]: finite\ A$

**using**  $finite-subset[OF\ -\ assms, of\ A]$  **by**  $auto$

**from**  $that$  **have**  $card\ A \leq k$

**by**  $(auto\ simp: \mathcal{A}-def)$

**have**  $card: card\ A = Suc (card (A - \{j\}))$

**using**  $card.remove[of\ A\ j]$  **by**  $auto$

**hence**  $card-less: card (A - \{j\}) < card\ A$  **by**  $linarith$

**have**  $w (T (A, j)) = monom (monom-of-set (A - \{j\}) + sng\ j (k - card (A - \{j\})))$

$((-1) \wedge card (A - \{j\}))$  **by**  $(simp\ add: w-def\ T-def)$

**also** **have**  $(-1) \wedge card (A - \{j\}) = ((-1) \wedge Suc (Suc (card (A - \{j\}))))$   
 $:: 'a)$

**by**  $simp$

**also** **have**  $Suc (card (A - \{j\})) = card\ A$

**using**  $card$  **by**  $simp$

**also** **have**  $k - card (A - \{j\}) = Suc (k - card\ A)$

**using**  $\langle k > 0 \rangle \langle card\ A \leq k \rangle card-less$  **by**  $(subst\ card)\ auto$

**also** **have**  $monom-of-set (A - \{j\}) + sng\ j (Suc (k - card\ A)) =$   
 $monom-of-set\ A + sng\ j (k - card\ A)$

**by**  $(transfer\ fixing: A\ j\ k)\ (auto\ simp: fun-eq-iff)$

**also** **have**  $monom \dots ((-1) \wedge Suc (card\ A)) = -w (A, j)$

**by**  $(simp\ add: w-def\ monom-uminus)$

**finally** **show**  $?thesis$  .

```

qed

show ?thesis
proof (cases j ∈ A)
  case True
  with aux[of A j] that show ?thesis by auto
next
  case False
  hence snd (T x) ∈ fst (T x)
  by (auto simp: T-def split: prod.splits)
  with aux[of fst (T x) snd (T x)] that show ?thesis by auto
qed
qed

```

We can now show fairly easily that the sum is equal to zero.

```

have *: A = {x ∈ A. snd x ∈ fst x} ∪ {x ∈ A. snd x ∉ fst x}
  by auto
have (∑ x ∈ A. w x) = (∑ x | x ∈ A ∧ snd x ∈ fst x. w x) + (∑ x | x ∈ A ∧
snd x ∉ fst x. w x)
  using ⟨finite A⟩ by (subst *, subst sum.union-disjoint) auto
also have (∑ x | x ∈ A ∧ snd x ∉ fst x. w x) = (∑ x | x ∈ A ∧ snd x ∈ fst
x. w (T x))
  using sum.reindex-bij-betw[OF bij, of w] by simp
also have ... = -(∑ x | x ∈ A ∧ snd x ∈ fst x. w x)
  by (simp add: sum-negf)
finally show (∑ x ∈ A. w x) = 0
  by simp
qed

```

```

finally show ?thesis .
qed

```

The following variant of the theorem holds for  $k > n$ . Note that this is now a linear recurrence relation with constant coefficients for  $p_k$  in terms of  $e_0, \dots, e_n$ .

**corollary** *Girard-Newton'*:

**assumes** *finite X and  $k > \text{card } X$*

**shows**  $(\sum_{i \leq \text{card } X}. (-1)^{\wedge i} * \text{sym-mpoly } X \ i * \text{powsum-mpoly } X \ (k - i)) =$   
 $(0 :: 'a :: \text{comm-ring-1 mpoly})$

**proof** –

**have**  $(0 :: 'a \text{ mpoly}) = (\sum_{i < k}. (-1)^{\wedge i} * \text{sym-mpoly } X \ i * \text{powsum-mpoly } X$   
 $(k - i))$

**using** *Girard-Newton[of X k] assms* **by** *simp*

**also have**  $\dots = (\sum_{i \leq \text{card } X}. (-1)^{\wedge i} * \text{sym-mpoly } X \ i * \text{powsum-mpoly } X$   
 $(k - i))$

**using** *assms* **by** *(intro sum.mono-neutral-right) auto*

**finally show** *?thesis ..*

**qed**

The following variant is the Newton–Girard Theorem solved for  $e_k$ , giving us an explicit way to determine  $e_k$  from  $e_0, \dots, e_{k-1}$  and  $p_1, \dots, p_k$ :

**corollary** *sym-mpoly-recurrence*:

**assumes**  $k: k > 0$  **and** *finite*  $X$

**shows**  $(\text{sym-mpoly } X \ k :: 'a :: \text{field-char-0 mpoly}) =$   
 $-\text{smult } (1 / \text{of-nat } k) (\sum i=1..k. (-1) ^ i * \text{sym-mpoly } X (k - i) * \text{powsum-mpoly } X \ i)$

**proof** –

**define**  $e \ p :: \text{nat} \Rightarrow 'a \ \text{mpoly}$  **where**  $[\text{simp}]$ :  $e = \text{sym-mpoly } X \ p = \text{powsum-mpoly } X$

**have**  $*$ :  $0 = (-1) ^ k * \text{of-nat } k * e \ k +$   
 $(\sum i < k. (-1) ^ i * e \ i * p \ (k - i) :: 'a \ \text{mpoly})$

**using** *Girard-Newton*[*of X k*] **assms** **by** *simp*

**have**  $0 = (-1) ^ k * \text{smult } (1 / \text{of-nat } k) (0 :: 'a \ \text{mpoly})$   
**by** *simp*

**also have**  $\dots = \text{smult } (1 / \text{of-nat } k) (\text{of-nat } k) * e \ k +$   
 $\text{smult } (1 / \text{of-nat } k) (\sum i < k. (-1) ^ (k+i) * e \ i * p \ (k - i))$

**unfolding** *smult-conv-mult*

**using**  $k$  **by** (*subst*  $*$ ) (*simp* *add*: *power-add* *sum-distrib-left* *sum-distrib-right* *field-simps*)

*del*: *div-mult-self3* *div-mult-self4* *div-mult-self2* *div-mult-self1*)

**also have**  $\text{smult } (1 / \text{of-nat } k :: 'a) (\text{of-nat } k) = 1$

**using**  $k$  **by** (*simp* *add*: *of-nat-monom* *smult-conv-mult* *mult-monom* *del*: *monom-of-nat*)

**also have**  $(\sum i < k. (-1) ^ (k+i) * e \ i * p \ (k - i)) = (\sum i=1..k. (-1) ^ i * e$   
 $(k-i) * p \ i)$

**by** (*intro* *sum.reindex-bij-witness*[*of - \lambda i. k - i \ \lambda i. k - i*])  
*(auto simp: minus-one-power-iff)*

**finally show** *?thesis* **unfolding** *e-p-def* **by** *algebra*

**qed**

Analogously, the following is the theorem solved for  $p_k$ , giving us a way to determine  $p_k$  from  $e_0, \dots, e_k$  and  $p_1, \dots, p_{k-1}$ :

**corollary** *powsum-mpoly-recurrence*:

**assumes**  $k: k > 0$  **and** *X*: *finite*  $X$

**shows**  $(\text{powsum-mpoly } X \ k :: 'a :: \text{comm-ring-1 mpoly}) =$   
 $(-1) ^ (k + 1) * \text{of-nat } k * \text{sym-mpoly } X \ k -$   
 $(\sum i=1..<k. (-1) ^ i * \text{sym-mpoly } X \ i * \text{powsum-mpoly } X \ (k - i))$

**proof** –

**define**  $e \ p :: \text{nat} \Rightarrow 'a \ \text{mpoly}$  **where**  $[\text{simp}]$ :  $e = \text{sym-mpoly } X \ p = \text{powsum-mpoly } X$

**have**  $*$ :  $0 = (-1) ^ k * \text{of-nat } k * e \ k +$   
 $(\sum i < k. (-1) ^ i * e \ i * p \ (k - i) :: 'a \ \text{mpoly})$

**using** *Girard-Newton*[*of X k*] **assms** **by** *simp*

**also have**  $\{..<k\} = \text{insert } 0 \ \{1..<k\}$

**using** *assms* **by** *auto*

**finally have**  $(-1) ^ k * \text{of-nat } k * e \ k + (\sum i=1..<k. (-1) ^ i * e \ i * p \ (k -$   
 $i)) + p \ k = 0$

**using** *assms* **by** (*simp* *add*: *algebra-simps*)

**from** *add.inverse-unique*[*OF this*] **show** *?thesis* **by** *simp*  
**qed**

Again, if we assume  $k > n$ , the above takes a much simpler form and is, in fact, a linear recurrence with constant coefficients:

**lemma** *powsum-mpoly-recurrence'*:

**assumes**  $k: k > \text{card } X$  **and**  $X: \text{finite } X$

**shows**  $(\text{powsum-mpoly } X \ k :: 'a :: \text{comm-ring-1 mpoly}) =$

$-(\sum_{i=1.. \text{card } X}. (-1)^i * \text{sym-mpoly } X \ i * \text{powsum-mpoly } X \ (k - i))$

**proof** –

**define**  $e \ p :: \text{nat} \Rightarrow 'a \ \text{mpoly}$  **where** [*simp*]:  $e = \text{sym-mpoly } X \ p = \text{powsum-mpoly } X$

**have**  $p \ k = (-1)^{k+1} * \text{of-nat } k * e \ k - (\sum_{i=1..<k}. (-1)^i * e \ i * p \ (k - i))$

**unfolding** *e-p-def* **using** *assms* **by** (*intro powsum-mpoly-recurrence*) *auto*

**also have**  $\dots = -(\sum_{i=1..<k}. (-1)^i * e \ i * p \ (k - i))$

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

**also have**  $(\sum_{i=1..<k}. (-1)^i * e \ i * p \ (k - i)) = (\sum_{i=1.. \text{card } X}. (-1)^i * e \ i * p \ (k - i))$

**using** *assms* **by** (*intro sum.mono-neutral-right*) *auto*

**finally show** *?thesis* **by** *simp*

**qed**

**end**

### 3 Power sum puzzles

**theory** *Power-Sum-Puzzle*

**imports**

*Power-Sum-Polynomials*

*Polynomial-Factorization.Rational-Root-Test*

**begin**

#### 3.1 General setting and results

We now consider the following situation: Given unknown complex numbers  $x_1, \dots, x_n$ , define  $p_k = x_1^k + \dots + x_n^k$ . Also, define  $e_k := e_k(x_1, \dots, x_n)$  where  $e_k(X_1, \dots, X_n)$  is the  $k$ -th elementary symmetric polynomial.

What is the relationship between the sequences  $e_k$  and  $p_k$ ; in particular, how can we determine one from the other?

**locale** *power-sum-puzzle* =

**fixes**  $x :: \text{nat} \Rightarrow \text{complex}$

**fixes**  $n :: \text{nat}$

**begin**

We first introduce the notation  $p_k := x_1^k + \dots + x_n^k$ :

**definition**  $p$  where  $p\ k = (\sum_{i < n}. x\ i \wedge k)$

**lemma**  $p-0$  [simp]:  $p\ 0 = \text{of-nat } n$   
 by (simp add: p-def)

**lemma**  $p\text{-altdef}$ :  $p\ k = \text{insertion } x\ (\text{powsum-mpoly } \{..<n\}\ k)$   
 by (simp add: p-def)

Similarly, we introduce the notation  $e_k = e_k(x_1, \dots, x_n)$  where  $e_k(X_1, \dots, X_n)$  is the  $k$ -th elementary symmetric polynomial (i. e. the sum of all monomials that can be formed by taking the product of exactly  $k$  distinct variables).

**definition**  $e$  where  $e\ k = (\sum Y \mid Y \subseteq \{..<n\} \wedge \text{card } Y = k. \text{prod } x\ Y)$

**lemma**  $e\text{-altdef}$ :  $e\ k = \text{insertion } x\ (\text{sym-mpoly } \{..<n\}\ k)$   
 by (simp add: e-def insertion-sym-mpoly)

It is clear that  $e_k$  vanishes for  $k > n$ .

**lemma**  $e\text{-eq-0}$  [simp]:  $k > n \implies e\ k = 0$   
 by (simp add: e-altdef)

**lemma**  $e-0$  [simp]:  $e\ 0 = 1$   
 by (simp add: e-altdef)

The recurrences we got from the Girard–Newton Theorem earlier now directly give us analogous recurrences for  $e_k$  and  $p_k$ :

**lemma**  $e\text{-recurrence}$ :

**assumes**  $k: k > 0$   
**shows**  $e\ k = -(\sum_{i=1..k}. (-1)^i * e\ (k - i) * p\ i) / \text{of-nat } k$   
**using** *assms* **unfolding**  $e\text{-altdef } p\text{-altdef}$   
**by** (*subst sym-mpoly-recurrence*)  
 (*auto simp: insertion-sum insertion-add insertion-mult insertion-power insertion-sym-mpoly*)

**lemma**  $p\text{-recurrence}$ :

**assumes**  $k: k > 0$   
**shows**  $p\ k = -\text{of-nat } k * (-1)^k * e\ k - (\sum_{i=1..<k}. (-1)^i * e\ i * p\ (k - i))$   
**using** *assms* **unfolding**  $e\text{-altdef } p\text{-altdef}$   
**by** (*subst powsum-mpoly-recurrence*)  
 (*auto simp: insertion-sum insertion-add insertion-mult insertion-diff insertion-power insertion-sym-mpoly*)

**lemma**  $p\text{-recurrence''}$ :

**assumes**  $k: k > n$   
**shows**  $p\ k = -(\sum_{i=1..n}. (-1)^i * e\ i * p\ (k - i))$   
**using** *assms* **unfolding**  $e\text{-altdef } p\text{-altdef}$   
**by** (*subst powsum-mpoly-recurrence'*)  
 (*auto simp: insertion-sum insertion-add insertion-mult insertion-diff*)

*insertion-power insertion-sym-mpoly)*

It is clear from this recurrence that if  $p_1$  to  $p_n$  are rational, then so are the  $e_k$ :

```

lemma e-in-Rats:
  assumes  $\bigwedge k. k \in \{1..n\} \implies p\ k \in \mathbb{Q}$ 
  shows  $e\ k \in \mathbb{Q}$ 
proof (cases  $k \leq n$ )
  case True
  thus ?thesis
proof (induction  $k$  rule: less-induct)
  case (less  $k$ )
  show ?case
  proof (cases  $k = 0$ )
  case False
  thus ?thesis using assms less
    by (subst e-recurrence) (auto intro!: Rats-divide)
  qed auto
qed
qed auto

```

Analogously, if  $p_1$  to  $p_n$  are rational, then so are all the other  $p_k$ :

```

lemma p-in-Rats:
  assumes  $\bigwedge k. k \in \{1..n\} \implies p\ k \in \mathbb{Q}$ 
  shows  $p\ k \in \mathbb{Q}$ 
proof (induction  $k$  rule: less-induct)
  case (less  $k$ )
  consider  $k = 0 \mid k \in \{1..n\} \mid k > n$ 
  by force
  thus ?case
proof cases
  assume  $k > n$ 
  thus ?thesis
    using less assms by (subst p-recurrence'') (auto intro!: sum-in-Rats Rats-mult e-in-Rats)
  qed (use assms in auto)
qed

```

Next, we define the unique monic polynomial that has  $x_1, \dots, x_n$  as its roots (respecting multiplicity):

**definition**  $Q :: \text{complex poly where } Q = (\prod_{i < n.} [-x\ i, 1:])$

```

lemma degree-Q [simp]: Polynomial.degree  $Q = n$ 
  by (simp add: Q-def degree-prod-eq-sum-degree)

```

```

lemma lead-coeff-Q [simp]: Polynomial.coeff  $Q\ n = 1$ 
  using monic-prod[of {..<n}  $\lambda i. [-x\ i, 1:]$ ]
  by (simp add: Q-def degree-prod-eq-sum-degree)

```

By Vieta's Theorem, we then have:

$$Q(X) = \sum_{k=0}^n (-1)^{n-k} e_{n-k} X^k$$

In other words: The above allows us to determine the  $x_1, \dots, x_n$  explicitly. They are, in fact, precisely the roots of the above polynomial (respecting multiplicity). Since this polynomial depends only on the  $e_k$ , which are in turn determined by  $p_1, \dots, p_n$ , this means that these are the *only* solutions of this puzzle (up to permutation of the  $x_i$ ).

**lemma** *coeff-Q: Polynomial.coeff Q k = (if k > n then 0 else (-1) ^ (n - k) \* e (n - k))*

**proof** (*cases k ≤ n*)

**case** *True*

**thus** *?thesis*

**using** *coeff-poly-from-roots[of {..<n} k x] by (auto simp: Q-def e-def)*

**qed** (*auto simp: Polynomial.coeff-eq-0*)

**lemma** *Q-altdef: Q = (∑ k ≤ n. Polynomial.monom ((-1) ^ (n - k) \* e (n - k)) k)*

**by** (*subst poly-as-sum-of-monoms [symmetric] (simp add: coeff-Q)*)

The following theorem again shows that  $x_1, \dots, x_n$  are precisely the roots of  $Q$ , respecting multiplicity.

**theorem** *mset-x-eq-poly-roots-Q: {#x i. i ∈ # mset-set {..<n}#} = poly-roots Q*

**proof** –

**have** *poly-roots Q = (∑ i < n. {#x i#})*

**by** (*simp add: Q-def poly-roots-prod*)

**also have** *... = {#x i. i ∈ # mset-set {..<n}#}*

**by** (*induction n (auto simp: lessThan-Suc)*)

**finally show** *?thesis ..*

**qed**

**end**

### 3.2 Existence of solutions

So far, we have assumed a solution to the puzzle and then shown the properties that this solution must fulfil. However, we have not yet shown that there *is* a solution. We will do that now.

Let  $n$  be a natural number and  $f_k$  some sequence of complex numbers. We will show that there are  $x_1, \dots, x_n$  so that  $x_1^k + \dots + x_n^k = f_k$  for any  $1 \leq k \leq n$ .

**locale** *power-sum-puzzle-existence =*

**fixes** *f :: nat ⇒ complex and n :: nat*

**begin**

First, we define a sequence of numbers  $e'$  analogously to the sequence  $e$  before, except that we replace all occurrences of the power sum  $p_k$  with  $f_k$  (recall that in the end we want  $p_k = f_k$ ).

```
fun e' :: nat ⇒ complex
  where e' k = (if k = 0 then 1 else if k > n then 0
                else -(∑ i=1..k. (-1) ^ i * e' (k - i) * f i) / of-nat k)
```

```
lemmas [simp del] = e'.simps
```

```
lemma e'-0 [simp]: e' 0 = 1
by (simp add: e'.simps)
```

```
lemma e'-eq-0 [simp]: k > n ⇒ e' k = 0
by (auto simp: e'.simps)
```

Just as before, we can show the following recurrence for  $f$  in terms of  $e'$ :

```
lemma f-recurrence:
  assumes k: k > 0 k ≤ n
  shows f k = -of-nat k * (-1) ^ k * e' k - (∑ i=1..<k. (-1) ^ i * e' i * f
  (k - i))
proof -
  have -of-nat k * e' k = (∑ i=1..k. (-1) ^ i * e' (k - i) * f i)
    using assms by (subst e'.simps) (simp add: field-simps)
  hence (-1) ^ k * (-of-nat k * e' k) = (-1) ^ k * (∑ i=1..k. (-1) ^ i * e' (k -
  i) * f i)
    by simp
  also have ... = f k + (-1) ^ k * (∑ i=1..<k. (-1) ^ i * e' (k - i) * f i)
    using assms by (subst sum.last-plus) (auto simp: minus-one-power-iff)
  also have (-1) ^ k * (∑ i=1..<k. (-1) ^ i * e' (k - i) * f i) =
    (∑ i=1..<k. (-1) ^ (k - i) * e' (k - i) * f i)
    unfolding sum-distrib-left by (intro sum.cong) (auto simp: minus-one-power-iff)
  also have ... = (∑ i=1..<k. (-1) ^ i * e' i * f (k - i))
    by (intro sum.reindex-bij-witness[of - λi. k - i λi. k - i]) auto
  finally show ?thesis
    by (simp add: algebra-simps)
```

**qed**

We now define a polynomial whose roots will be precisely the solution  $x_1, \dots, x_n$  to our problem.

```
lift-definition Q' :: complex poly is λk. if k > n then 0 else (-1) ^ (n - k) * e'
(n - k)
using eventually-gt-at-top[of n] unfolding cofinite-eq-sequentially
by eventually-elim auto
```

```
lemma coeff-Q': Polynomial.coeff Q' k = (if k > n then 0 else (-1) ^ (n - k) *
e' (n - k))
by transfer auto
```

**lemma** *lead-coeff-Q'*: *Polynomial.coeff Q' n = 1*  
**by** (*simp add: coeff-Q'*)

**lemma** *degree-Q' [simp]*: *Polynomial.degree Q' = n*  
**proof** (*rule antisym*)  
**show** *Polynomial.degree Q' ≥ n*  
**by** (*rule le-degree*) (*auto simp: coeff-Q'*)  
**show** *Polynomial.degree Q' ≤ n*  
**by** (*rule degree-le*) (*auto simp: coeff-Q'*)  
**qed**

Since the complex numbers are algebraically closed, this polynomial splits into linear factors:

**definition** *Root* :: *nat ⇒ complex*  
**where** *Root = (SOME Root. Q' = (∏ i < n. [:-Root i, 1:]))*

**lemma** *Root*: *Q' = (∏ i < n. [:-Root i, 1:])*  
**proof** –  
**obtain** *rs where rs: (∏ r ← rs. [:-r, 1:]) = Q' length rs = n*  
**using** *fundamental-theorem-algebra-factorized[of Q'] lead-coeff-Q' by auto*  
**have** *Q' = (∏ r ← rs. [:-r, 1:])*  
**by** (*simp add: rs*)  
**also have** *... = (∏ r = 0..<n. [:-rs ! r, 1:])*  
**by** (*subst prod-list-prod-nth*) (*auto simp: rs*)  
**also have** *{0..<n} = {..<n}*  
**by** *auto*  
**finally have**  $\exists$  *Root. Q' = (∏ i < n. [:-Root i, 1:])*  
**by** *blast*  
**thus** *?thesis*  
**unfolding** *Root-def* **by** (*rule someI-ex*)  
**qed**

We can therefore now use the results from before for these  $x_1, \dots, x_n$ .

**sublocale** *power-sum-puzzle Root n* .

Vieta's theorem gives us an expression for the coefficients of  $Q'$  in terms of  $e_k(x_1, \dots, x_n)$ . This shows that our  $e'$  is indeed exactly the same as  $e$ .

**lemma** *e'-eq-e: e' k = e k*  
**proof** (*cases k ≤ n*)  
**case** *True*  
**from** *True* **have**  $e' k = (-1) \wedge k * \text{poly.coeff } Q' (n - k)$   
**by** (*simp add: coeff-Q'*)  
**also have**  $Q' = (\prod x < n. [:-\text{Root } x, 1:])$   
**using** *Root* **by** *simp*  
**also have**  $(-1) \wedge k * \text{poly.coeff } \dots (n - k) = e k$   
**using** *True coeff-poly-from-roots[of {..<n} n - k Root]*  
**by** (*simp add: insertion-sym-mpoly e-altdef*)  
**finally show**  $e' k = e k$  .

**qed** *auto*

It then follows by a simple induction that  $p_k = f_k$  for  $1 \leq k \leq n$ , as intended:

```
lemma p-eq-f:  
  assumes  $k > 0$   $k \leq n$   
  shows  $p\ k = f\ k$   
  using assms  
proof (induction k rule: less-induct)  
  case (less k)  
  thus  $p\ k = f\ k$   
    using p-recurrence[of k] f-recurrence[of k] less by (simp add: e'-eq-e)  
qed  
  
end
```

Here is a more condensed form of the above existence theorem:

```
theorem power-sum-puzzle-has-solution:  
  fixes  $f :: \text{nat} \Rightarrow \text{complex}$   
  shows  $\exists \text{Root}. \forall k \in \{1..n\}. (\sum i < n. \text{Root } i \wedge k) = f\ k$   
proof -  
  interpret power-sum-puzzle-existence f .  
  from p-eq-f have  $\forall k \in \{1..n\}. (\sum i < n. \text{Root } i \wedge k) = f\ k$   
    by (auto simp: p-def)  
  thus ?thesis by blast  
qed
```

### 3.3 A specific puzzle

We now look at one particular instance of this puzzle, which was given as an exercise in *Abstract Algebra* by Dummit and Foote (Exercise 23 in Section 14.6) [1].

Suppose we know that  $x + y + z = 1$ ,  $x^2 + y^2 + z^2 = 2$ , and  $x^3 + y^3 + z^3 = 3$ . Then what is  $x^5 + y^5 + z^5$ ? What about any arbitrary  $x^n + y^n + z^n$ ?

```
locale power-sum-puzzle-example =  
  fixes  $x\ y\ z :: \text{complex}$   
  assumes  $xyz: x + y + z = 1$   
            $x^2 + y^2 + z^2 = 2$   
            $x^3 + y^3 + z^3 = 3$   
begin
```

We reuse the results we have shown in the general case before.

```
definition f where  $f\ n = [x,y,z] ! n$ 
```

```
sublocale power-sum-puzzle f 3 .
```

We can simplify *p* a bit more now.

```
lemma p-altdef':  $p\ k = x \wedge k + y \wedge k + z \wedge k$ 
```

**unfolding**  $p$ -def  $f$ -def **by** (*simp add: eval-nat-numeral*)

**lemma**  $p$ -base [*simp*]:  $p (Suc\ 0) = 1$   $p\ 2 = 2$   $p\ 3 = 3$   
**using**  $xyz$  **by** (*simp-all add: p-altdef'*)

We can easily compute all the non-zero values of  $e$  recursively:

**lemma**  $e$ -Suc-0 [*simp*]:  $e (Suc\ 0) = 1$   
**by** (*subst e-recurrence; simp*)

**lemma**  $e$ -2 [*simp*]:  $e\ 2 = -1/2$   
**by** (*subst e-recurrence; simp add: atLeastAtMost-nat-numeral*)

**lemma**  $e$ -3 [*simp*]:  $e\ 3 = 1/6$   
**by** (*subst e-recurrence; simp add: atLeastAtMost-nat-numeral*)

Plugging in all the values, the recurrence relation for  $p$  now looks like this:

**lemma**  $p$ -recurrence''':  $k > 3 \implies p\ k = p (k-3) / 6 + p (k-2) / 2 + p (k-1)$   
**using**  $p$ -recurrence''[of  $k$ ] **by** (*simp add: atLeastAtMost-nat-numeral*)

Also note again that all  $p_k$  are rational:

**lemma**  $p$ -in-Rats':  $p\ k \in \mathbb{Q}$   
**proof** –  
**have** \*:  $\{1..3\} = \{1, 2, (3::nat)\}$   
**by** *auto*  
**also have**  $\forall k \in \dots\ p\ k \in \mathbb{Q}$   
**by** *auto*  
**finally show** *?thesis*  
**using**  $p$ -in-Rats[of  $k$ ] **by** *simp*

**qed**

The above recurrence has the characteristic polynomial  $X^3 - X^2 - \frac{1}{2}X - \frac{1}{6}$  (which is exactly our  $Q$ ), so we know that can now specify  $x$ ,  $y$ , and  $z$  more precisely: They are the roots of that polynomial (in unspecified order).

**lemma**  $xyz$ -eq:  $\{\#x, y, z\} = \text{poly-roots } [-1/6, -1/2, -1, 1:]$

**proof** –  
**have**  $\text{image-mset } f (mset\text{-set } \{..<3\}) = \text{poly-roots } Q$   
**using**  $mset$ -x-eq-poly-roots- $Q$  .  
**also have**  $\text{image-mset } f (mset\text{-set } \{..<3\}) = \{\#x, y, z\}$   
**by** (*simp add: numeral-3-eq-3 lessThan-Suc f-def Multiset.union-ac*)  
**also have**  $Q = [-1/6, -1/2, -1, 1:]$   
**by** (*simp add: Q-altdef atMost-nat-numeral Polynomial.monom-altdef power3-eq-cube power2-eq-square*)  
**finally show** *?thesis* .

**qed**

Using the rational root test, we can easily show that  $x$ ,  $y$ , and  $z$  are irrational.

**lemma**  $xyz$ -irrational:  $\text{set-mset } (\text{poly-roots } [-1/6, -1/2, -1, 1::\text{complex}]) \cap \mathbb{Q} = \{\}$

**proof** –

```
define  $p :: \text{rat poly}$  where  $p = [-1/6, -1/2, -1, 1:]$   
have rational-root-test  $p = \text{None}$   
  unfolding p-def by code-simp  
hence  $\neg(\exists x :: \text{rat}. \text{poly } p \ x = 0)$   
  by (rule rational-root-test)  
hence  $\neg(\exists x \in \mathbb{Q}. \text{poly } (\text{map-poly of-rat } p) \ x = (0 :: \text{complex}))$   
  by (auto simp: Rats-def)  
also have map-poly of-rat  $p = [-1/6, -1/2, -1, 1 :: \text{complex}]$   
  by (simp add: p-def of-rat-minus of-rat-divide)  
finally show ?thesis  
  by auto
```

**qed**

This polynomial is *squarefree*, so these three roots are, in fact, unique (so that there are indeed  $3! = 6$  possible permutations).

```
lemma rsquarefree: rsquarefree  $[-1/6, -1/2, -1, 1 :: \text{complex}]$   
  by (rule coprime-pderiv-imp-rsquarefree)  
  (auto simp: pderiv-pCons coprime-iff-gcd-eq-1 gcd-poly-code gcd-poly-code-def  
  content-def primitive-part-def gcd-poly-code-aux-reduce pseudo-mod-def pseudo-divmod-def  
  Let-def Polynomial.monom-altdef normalize-poly-def)
```

```
lemma distinct-xyz: distinct  $[x, y, z]$ 
```

```
  by (rule rsquarefree-imp-distinct-roots[OF rsquarefree]) (simp-all add: xyz-eq)
```

While these roots *can* be written more explicitly in radical form, they are not very pleasant to look at. We therefore only compute a few values of  $p$  just for fun:

```
lemma  $p \ 4 = 25 / 6$  and  $p \ 5 = 6$  and  $p \ 10 = 15539 / 432$   
  by (simp-all add: p-recurrence''')
```

Lastly, let us (informally) examine the asymptotics of this problem.

Two of the roots have a norm of roughly  $\beta \approx 0.341$ , while the remaining root  $\alpha$  is roughly 1.431. Consequently,  $x^n + y^n + z^n$  is asymptotically equivalent to  $\alpha^n$ , with the error being bounded by  $2 \cdot \beta^n$  and therefore goes to 0 very quickly.

For  $p(10) = \frac{15539}{432} \approx 35.97$ , for instance, this approximation is correct up to 6 decimals (a relative error of about 0.0001%).

**end**

To really emphasise that the above puzzle has a solution and the locale is not ‘vacuous’, here is an interpretation of the locale using the existence theorem from before:

```
notepad
```

```
begin
```

```
  define  $f :: \text{nat} \Rightarrow \text{complex}$  where  $f = (\lambda k. [1,2,3]! (k - 1))$ 
```

```

obtain Root :: nat  $\Rightarrow$  complex where Root:  $\bigwedge k. k \in \{1..3\} \implies (\sum i < 3. \text{Root } i \wedge k) = f k$ 
using power-sum-puzzle-has-solution[of 3 f] by metis
define x y z where x = Root 0 y = Root 1 z = Root 2
have x + y + z = 1 and x2 + y2 + z2 = 2 and x3 + y3 + z3 = 3
using Root[of 1] Root[of 2] Root[of 3] by (simp-all add: eval-nat-numeral x-y-z-def f-def)
then interpret power-sum-puzzle-example x y z
by unfold-locales
have p 5 = 6
by (simp add: p-recurrence'')
end

end

```

## References

- [1] D. S. Dummit and R. M. Foote. *Abstract Algebra*. Wiley, 2003.
- [2] D. Zeilberger. A combinatorial proof of Newton's identities. *Discrete Mathematics*, 49(3):319, 1984.