

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

1	Auxiliary material	2
1.1	Miscellaneous	2
1.2	The set of roots of a univariate polynomial	5
2	Power sum polynomials	7
2.1	Definition	8
2.2	The Girard–Newton Theorem	10
3	Power sum puzzles	11
3.1	General setting and results	11
3.2	Existence of solutions	13
3.3	A specific puzzle	15

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 {})

<proof>

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

<proof>

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

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

<proof>

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

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

<proof>

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

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

<proof>

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

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

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

<proof>

lemma *coprimeI-primes:*

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

assumes *a ≠ 0 \vee b ≠ 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>

lemma *coprime-pderiv-imp-squarefree*:

assumes *coprime p (pderiv p)*

shows *squarefree p*

<proof>

lemma *squarefree-field-poly-iff*:

fixes $p :: 'a :: \{\text{field-char-0, euclidean-ring-gcd, semiring-gcd-mult-normalize}\}$ *poly*

assumes [*simp*]: $p \neq 0$

shows $\text{squarefree } p \longleftrightarrow \text{coprime } p \text{ (pderiv } p)$

<proof>

lemma *coprime-pderiv-imp-rsquarefree*:

assumes $\text{coprime } (p :: 'a :: \text{field-char-0 poly}) \text{ (pderiv } p)$

shows *rsquarefree p*

<proof>

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

<proof>

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

<proof>

lemma *order-eq-0-iff*: $p \neq 0 \implies \text{order } x p = 0 \longleftrightarrow \text{poly } p x \neq 0$

<proof>

lemma *order-pos-iff*: $p \neq 0 \implies \text{order } x p > 0 \longleftrightarrow \text{poly } p x = 0$

<proof>

lemma *order-prod*:

assumes $\bigwedge x. x \in A \implies f x \neq 0$

shows $\text{order } x (\prod_{y \in A}. f y) = (\sum_{y \in A}. \text{order } x (f y))$

<proof>

lemma *order-prod-mset*:

assumes $0 \notin \# A$

shows $\text{order } x (\text{prod-mset } A) = \text{sum-mset (image-mset (order } x) A)$

<proof>

lemma *order-prod-list*:

assumes $0 \notin \text{set } xs$

shows $\text{order } x (\text{prod-list } xs) = \text{sum-list (map (order } x) xs)$

<proof>

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

<proof>

lemma *smult-0-right [simp]*: $\text{MPoly-Type.smult } p 0 = 0$

<proof>

lemma *mult-smult-right* [simp]:

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

shows $p * \text{MPoly-Type.smult } c \ q = \text{MPoly-Type.smult } c \ (p * q)$

<proof>

lemma *mapping-single-eq-iff* [simp]:

$\text{Poly-Mapping.single } a \ b = \text{Poly-Mapping.single } c \ d \iff b = 0 \wedge d = 0 \vee a = c \wedge b = d$

<proof>

lemma *monom-of-set-plus-monom-of-set*:

assumes $A \cap B = \{\}$ *finite A finite B*

shows $\text{monom-of-set } A + \text{monom-of-set } B = \text{monom-of-set } (A \cup B)$

<proof>

lemma *mpoly-monom-0-eq-Const*: $\text{monom } 0 \ c = \text{Const } c$

<proof>

lemma *mpoly-Const-0* [simp]: $\text{Const } 0 = 0$

<proof>

lemma *mpoly-Const-1* [simp]: $\text{Const } 1 = 1$

<proof>

lemma *mpoly-Const-uminus*: $\text{Const } (-a) = -\text{Const } a$

<proof>

lemma *mpoly-Const-add*: $\text{Const } (a + b) = \text{Const } a + \text{Const } b$

<proof>

lemma *mpoly-Const-mult*: $\text{Const } (a * b) = \text{Const } a * \text{Const } b$

<proof>

lemma *mpoly-Const-power*: $\text{Const } (a \wedge n) = \text{Const } a \wedge n$

<proof>

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

<proof>

lemma *insertion-of-nat* [simp]: $\text{insertion } f \ (\text{of-nat } n) = \text{of-nat } n$

<proof>

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

$\text{insertion } f \ (\text{monom } (\text{monom-of-set } X) \ c) = c * (\prod_{i \in X}. f \ i)$

<proof>

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

<proof>

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

assumes *finite A*

shows $\text{sym-mpoly } A 0 = 1$

<proof>

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

assumes $k > \text{card } A$

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

<proof>

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

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

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

<proof>

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

assumes *finite X* $\neg Y \subseteq X$ *finite Y*

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

<proof>

1.2 The set of roots of a univariate polynomial

lift-definition *poly-roots* :: 'a :: idom poly \Rightarrow 'a multiset **is**

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

<proof>

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

<proof>

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

<proof>

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

assumes $p \neq 0$

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

<proof>

lemma *in-poly-roots-iff [simp]*: $p \neq 0 \implies x \in \# \text{poly-roots } p \iff \text{poly } p x = 0$

<proof>

lemma *set-mset-poly-roots*: $p \neq 0 \implies \text{set-mset } (\text{poly-roots } p) = \{x. \text{poly } p x = 0\}$

<proof>

lemma *count-poly-roots'*: *count* (*poly-roots* *p*) *x* = (*if* *p* = 0 *then* 0 *else* *order* *x* *p*)
⟨*proof*⟩

lemma *poly-roots-const* [*simp*]: *poly-roots* [:*c*:] = {#}
⟨*proof*⟩

lemma *poly-roots-linear* [*simp*]: *poly-roots* [:-*x*, 1:] = {#*x*#}
⟨*proof*⟩

lemma *poly-roots-monom* [*simp*]: *c* ≠ 0 ⇒ *poly-roots* (*Polynomial.monom* *c* *n*)
= *replicate-mset* *n* 0
⟨*proof*⟩

lemma *poly-roots-smult* [*simp*]: *c* ≠ 0 ⇒ *poly-roots* (*Polynomial.smult* *c* *p*) =
poly-roots *p*
⟨*proof*⟩

lemma *poly-roots-mult*: *p* ≠ 0 ⇒ *q* ≠ 0 ⇒ *poly-roots* (*p* * *q*) = *poly-roots* *p* +
poly-roots *q*
⟨*proof*⟩

lemma *poly-roots-prod*:
assumes $\bigwedge x. x \in A \implies f\ x \neq 0$
shows *poly-roots* (*prod* *f* *A*) = ($\sum x \in A. \text{poly-roots } (f\ x)$)
⟨*proof*⟩

lemma *poly-roots-prod-mset*:
assumes 0 ∉ # *A*
shows *poly-roots* (*prod-mset* *A*) = *sum-mset* (*image-mset* *poly-roots* *A*)
⟨*proof*⟩

lemma *poly-roots-prod-list*:
assumes 0 ∉ *set* *xs*
shows *poly-roots* (*prod-list* *xs*) = *sum-list* (*map* *poly-roots* *xs*)
⟨*proof*⟩

lemma *poly-roots-power*: *p* ≠ 0 ⇒ *poly-roots* (*p* ^ *n*) = *repeat-mset* *n* (*poly-roots*
p)
⟨*proof*⟩

lemma *rsquarefree-poly-roots-eq*:
assumes *rsquarefree* *p*
shows *poly-roots* *p* = *mset-set* {*x*. *poly* *p* *x* = 0}
⟨*proof*⟩

lemma *rsquarefree-imp-distinct-roots*:
assumes *rsquarefree* *p* **and** *mset* *xs* = *poly-roots* *p*
shows *distinct* *xs*

<proof>

lemma *poly-roots-factorization*:

fixes $p\ c\ A$

assumes [*simp*]: $c \neq 0$

defines $p \equiv \text{Polynomial.smult } c\ (\text{prod-mset } (\text{image-mset } (\lambda x. [-x, 1:])\ A))$

shows $\text{poly-roots } p = A$

<proof>

lemma *fundamental-theorem-algebra-factorized'*:

fixes $p :: \text{complex poly}$

shows $p = \text{Polynomial.smult } (\text{Polynomial.lead-coeff } p)$

$(\text{prod-mset } (\text{image-mset } (\lambda x. [-x, 1:])\ (\text{poly-roots } p)))$

<proof>

lemma *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>

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

<proof>

lemma *sym-mpoly-insert*:

assumes *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>

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* \Rightarrow *nat* \Rightarrow (*nat* \Rightarrow_0 *nat*) \Rightarrow_0 'a :: {*semiring-1, zero-neq-one*} **is**

$\lambda X k$ *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
 <proof>

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)
 <proof>

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
 <proof>

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)
 <proof>

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

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

powsum-mpoly-aux <proof>

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

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

lemma *coeff-powsum-mpoly*:

$MPoly\text{-Type.coeff } (pows\text{-}mpoly\ X\ k)\ mon =$
 (if infinite $X \vee k = 0 \wedge mon \neq 0$ then 0
 else if $k = 0 \wedge mon = 0$ then of-nat (card X)
 else if finite $X \wedge (\exists x \in X. mon = Poly\text{-}Mapping.single\ x\ k)$ then 1 else
 0)
 ⟨proof⟩

lemma *coeff-pows-mpoly-0-right*:
 $MPoly\text{-Type.coeff } (pows\text{-}mpoly\ X\ 0)\ mon = (if\ mon = 0\ then\ of\text{-}nat\ (card\ X)$
 else 0)
 ⟨proof⟩

lemma *coeff-pows-mpoly-singleton*:
 assumes finite $X\ k > 0$
 shows $MPoly\text{-Type.coeff } (pows\text{-}mpoly\ X\ k)\ (Poly\text{-}Mapping.single\ x\ k) = (if$
 $x \in X$ then 1 else 0)
 ⟨proof⟩

lemma *coeff-pows-mpoly-singleton-eq-1* [simp]:
 assumes finite $X\ x \in X\ k > 0$
 shows $MPoly\text{-Type.coeff } (pows\text{-}mpoly\ X\ k)\ (Poly\text{-}Mapping.single\ x\ k) = 1$
 ⟨proof⟩

lemma *coeff-pows-mpoly-singleton-eq-0* [simp]:
 assumes finite $X\ x \notin X\ k > 0$
 shows $MPoly\text{-Type.coeff } (pows\text{-}mpoly\ X\ k)\ (Poly\text{-}Mapping.single\ x\ k) = 0$
 ⟨proof⟩

lemma *pows-mpoly-0* [simp]: $pows\text{-}mpoly\ X\ 0 = of\text{-}nat\ (card\ X)$
 ⟨proof⟩

lemma *pows-mpoly-empty* [simp]: $pows\text{-}mpoly\ \{\}\ k = 0$
 ⟨proof⟩

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

Power sum polynomials are symmetric:

lemma *symmetric-pows-mpoly* [intro]:
 assumes $A \subseteq B$
 shows *symmetric-mpoly* A (*pows-mpoly* $B\ k$)
 ⟨proof⟩

lemma *insertion-pows-mpoly* [simp]: $insertion\ f\ (pows\text{-}mpoly\ X\ k) = (\sum_{i \in X}. f\ i\ ^k)$
 ⟨proof⟩

lemma *pows-mpoly-nz*:

assumes *finite X* $X \neq \{\}$ $k > 0$
shows $(\text{powsum-mpoly } X \ k :: 'a :: \{\text{semiring-1, zero-neq-one}\} \text{ mpoly}) \neq 0$
 $\langle \text{proof} \rangle$

lemma *powsum-mpoly-eq-0-iff*:

assumes $k > 0$
shows $\text{powsum-mpoly } X \ k = 0 \iff \text{infinite } X \vee X = \{\}$
 $\langle \text{proof} \rangle$

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 $(-1)^k \wedge k * \text{of-nat } k * \text{sym-mpoly } X \ k +$
 $(\sum_{i < k} (-1)^i * \text{sym-mpoly } X \ i * \text{powsum-mpoly } X \ (k - i)) =$
 $(0 :: 'a :: \text{comm-ring-1 mpoly})$
(is ?lhs = 0)
 $\langle \text{proof} \rangle$

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

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 $(\text{sym-mpoly } X \ k :: 'a :: \text{field-char-0 mpoly}) =$

$-smult (1 / of-nat k) (\sum i=1..k. (-1) ^ i * sym-mpoly X (k - i) * powsum-mpoly X i)$
 ⟨proof⟩

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

⟨proof⟩

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 $k: k > card X$ **and** $X: finite X$

shows $(powsum-mpoly X k :: 'a :: comm-ring-1 mpoly) =$
 $-(\sum i=1..card X. (-1) ^ i * sym-mpoly X i * powsum-mpoly X (k -$
 $i))$

⟨proof⟩

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 $x :: nat \Rightarrow complex$

fixes $n :: nat$

begin

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

definition *p where* $p k = (\sum i < n. x i ^ k)$

lemma *p-0 [simp]*: $p\ 0 = \text{of-nat } n$
 ⟨proof⟩

lemma *p-altdef*: $p\ k = \text{insertion } x\ (\text{powsum-mpoly } \{..<n\}\ k)$
 ⟨proof⟩

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-altdef*: $e\ k = \text{insertion } x\ (\text{sym-mpoly } \{..<n\}\ k)$
 ⟨proof⟩

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

lemma *e-eq-0 [simp]*: $k > n \implies e\ k = 0$
 ⟨proof⟩

lemma *e-0 [simp]*: $e\ 0 = 1$
 ⟨proof⟩

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

lemma *e-recurrence*:

assumes $k: k > 0$

shows $e\ k = -(\sum i=1..k. (-1)^i * e\ (k - i) * p\ i) / \text{of-nat } k$
 ⟨proof⟩

lemma *p-recurrence*:

assumes $k: k > 0$

shows $p\ k = -\text{of-nat } k * (-1)^k * e\ k - (\sum i=1..<k. (-1)^i * e\ i * p\ (k - i))$
 ⟨proof⟩

lemma *p-recurrence''*:

assumes $k: k > n$

shows $p\ k = -(\sum i=1..n. (-1)^i * e\ i * p\ (k - i))$
 ⟨proof⟩

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⟩

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>

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

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

lemma *degree-Q [simp]*: *Polynomial.degree* $Q = n$

<proof>

lemma *lead-coeff-Q [simp]*: *Polynomial.coeff* $Q \ n = 1$

<proof>

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* $Q \ k = (\text{if } k > n \text{ then } 0 \text{ else } (-1)^{\wedge(n-k)} * e_{(n-k)})$

<proof>

lemma *Q-altdef*: $Q = (\sum_{k \leq n}. \text{Polynomial.monom } ((-1)^{\wedge(n-k)} * e_{(n-k)}) \ k)$

<proof>

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*: $\{\#x \ i. \ i \in \# \text{ mset-set } \{..<n\}\#\} = \text{poly-roots } Q$

<proof>

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 $f :: \text{nat} \Rightarrow \text{complex}$ **and** $n :: \text{nat}$
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 $e' :: \text{nat} \Rightarrow \text{complex}$
where $e' k = (\text{if } k = 0 \text{ then } 1 \text{ else if } k > n \text{ then } 0$
 $\text{else } -(\sum_{i=1..k} (-1)^i * e' (k - i) * f i) / \text{of-nat } k)$

lemmas [*simp del*] = $e'.\text{simps}$

lemma $e'-0$ [*simp*]: $e' 0 = 1$
 $\langle \text{proof} \rangle$

lemma $e'-eq-0$ [*simp*]: $k > n \implies e' k = 0$
 $\langle \text{proof} \rangle$

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

lemma *f-recurrence*:
assumes $k: k > 0 \ k \leq n$
shows $f k = -\text{of-nat } k * (-1)^k * e' k - (\sum_{i=1..<k} (-1)^i * e' i * f$
 $(k - i))$
 $\langle \text{proof} \rangle$

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

lift-definition $Q' :: \text{complex poly}$ **is** $\lambda k. \text{if } k > n \text{ then } 0 \text{ else } (-1)^{(n - k)} * e'$
 $(n - k)$
 $\langle \text{proof} \rangle$

lemma *coeff-Q'*: $\text{Polynomial.coeff } Q' k = (\text{if } k > n \text{ then } 0 \text{ else } (-1)^{(n - k)} * e'$
 $(n - k))$
 $\langle \text{proof} \rangle$

lemma *lead-coeff-Q'*: $\text{Polynomial.coeff } Q' n = 1$
 $\langle \text{proof} \rangle$

lemma *degree-Q'* [*simp*]: $\text{Polynomial.degree } Q' = n$
 $\langle \text{proof} \rangle$

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

definition *Root* :: $\text{nat} \Rightarrow \text{complex}$
where $\text{Root} = (\text{SOME } \text{Root}. Q' = (\prod_{i < n} [:-\text{Root } i, 1:]))$

lemma *Root*: $Q' = (\prod_{i < n} [:-\text{Root } i, 1:])$

<proof>

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

sublocale *power-sum-puzzle Root n* *<proof>*

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>

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 $p k = f k$

<proof>

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>

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* *<proof>*

We can simplify p a bit more now.

lemma *p-altdef': p k = x ^ k + y ^ k + z ^ k*

<proof>

lemma *p-base* [*simp*]: $p (Suc\ 0) = 1$ $p\ 2 = 2$ $p\ 3 = 3$
 ⟨*proof*⟩

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

lemma *e-Suc-0* [*simp*]: $e (Suc\ 0) = 1$
 ⟨*proof*⟩

lemma *e-2* [*simp*]: $e\ 2 = -1/2$
 ⟨*proof*⟩

lemma *e-3* [*simp*]: $e\ 3 = 1/6$
 ⟨*proof*⟩

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

lemma *p-recurrence'''*: $k > 3 \implies p\ k = p (k-3) / 6 + p (k-2) / 2 + p (k-1)$
 ⟨*proof*⟩

Also note again that all p_k are rational:

lemma *p-in-Rats'*: $p\ k \in \mathbb{Q}$
 ⟨*proof*⟩

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*⟩

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*⟩

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

lemma *rsquarefree*: $\text{rsquarefree} [-1/6, -1/2, -1, 1 :: \text{complex}]$
 ⟨*proof*⟩

lemma *distinct-xyz*: $\text{distinct} [x, y, z]$
 ⟨*proof*⟩

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** $p\ 10 = 15539 / 432$
 ⟨*proof*⟩

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 α is roughly 1.431. Consequently, $x^n + y^n + z^n$ is asymptotically equivalent to α^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  
  <proof>  
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.