Power Sum Polynomials
and the Girard–Newton Theorem

Manuel Eberl

March 17, 2025

Abstract

This article provides a formalisation of the symmetric multivariate polynomials known as *power sum polynomials*. These are of the form $p_n(X_1, \ldots, X_k) = X_1^n + \ldots + X_k^n$. 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 $x_1, \ldots, x_k$, define $p_j := x_1^j + \ldots + x_k^j$. Then for each vector $a \in \mathbb{C}^k$, show that there is exactly one solution to the system $p_1 = a_1, \ldots, p_k = a_k$ up to permutation of the x_i and determine the value of p_i for $i > k$.

Contents

1 Auxiliary material

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* ∈ *A* ⟹ *f x* ∈ ℚ) ⟹ *sum f A* ∈ ℚ
⟨*proof*⟩

lemma (**in** *monoid-mult*) *prod-list-distinct-conv-prod-set*:
distinct xs ⟹ *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*..< *length xs*. *xs* ! *i*)
⟨*proof*⟩

lemma *gcd-poly-code-aux-reduce*:
gcd-poly-code-aux p 0 = *normalize p*
q ≠ *0* ⟹ *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* ∨ *b* ≠ *0*
assumes $\bigwedge$*p*. *prime p* ⟹ *p dvd a* ⟹ *p dvd b* ⟹ *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* :: {*field-char-0*,*euclidean-ring-gcd*,*semiring-gcd-mult-normalize*} *poly*
assumes [*simp*]: $p \neq 0$
shows *squarefree p* ⟷ *coprime p* (*pderiv p*)
⟨*proof*⟩

lemma *coprime-pderiv-imp-rsquarefree*:
assumes *coprime* (*p* :: *'a* :: *field-char-0 poly*) (*pderiv p*)
shows *rsquarefree p*
⟨*proof*⟩

lemma *poly-of-nat* [*simp*]: *poly* (*of-nat n*) *x* = *of-nat n*
⟨*proof*⟩

lemma *poly-of-int* [*simp*]: *poly* (*of-int n*) *x* = *of-int n*
⟨*proof*⟩

lemma *order-eq-0-iff*: $p \neq 0 \Longrightarrow$ *order x p* = *0* ⟷ *poly p x* ≠ *0*
⟨*proof*⟩

lemma *order-pos-iff*: $p \neq 0 \Longrightarrow$ *order x p* > *0* ⟷ *poly p x* = *0*
⟨*proof*⟩

lemma *order-prod*:
assumes $\bigwedge x.\ x \in A \Longrightarrow f\ x \neq 0$
shows *order x* ($\prod y \in A.\ f\ y$) = ($\sum y \in A.$ *order x* (*f y*))
⟨*proof*⟩

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

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

lemma *order-power*: $p \neq 0 \Longrightarrow$ *order x* (*p* ^ *n*) = *n* ∗ *order x p*
⟨*proof*⟩

lemma *smult-0-right* [*simp*]: *MPoly-Type.smult p 0* = *0*
⟨*proof*⟩

lemma *mult-smult-right* [*simp*]:
fixes c :: $'a$ :: *comm-semiring-0*
shows $p * $ *MPoly-Type.smult* $c\ q$ = *MPoly-Type.smult* $c\ (p * q)$
⟨*proof*⟩

lemma *mapping-single-eq-iff* [*simp*]:
Poly-Mapping.single $a\ b$ = *Poly-Mapping.single* $c\ d \longleftrightarrow b = 0 \wedge d = 0 \vee a = c \wedge b = d$
⟨*proof*⟩

lemma *monom-of-set-plus-monom-of-set*:
assumes $A \cap B = \{\}$ *finite* A *finite* B
shows *monom-of-set* A + *monom-of-set* B = *monom-of-set* $(A \cup B)$
⟨*proof*⟩

lemma *mpoly-monom-0-eq-Const*: *monom* $0\ c$ = *Const* c
⟨*proof*⟩

lemma *mpoly-Const-0* [*simp*]: *Const* $0 = 0$
⟨*proof*⟩

lemma *mpoly-Const-1* [*simp*]: *Const* $1 = 1$
⟨*proof*⟩

lemma *mpoly-Const-uminus*: *Const* $(-a) = -$*Const* a
⟨*proof*⟩

lemma *mpoly-Const-add*: *Const* $(a + b)$ = *Const* a + *Const* b
⟨*proof*⟩

lemma *mpoly-Const-mult*: *Const* $(a * b)$ = *Const* a $*$ *Const* b
⟨*proof*⟩

lemma *mpoly-Const-power*: *Const* $(a\ \hat{}\ n)$ = *Const* $a\ \hat{}\ n$
⟨*proof*⟩

lemma *of-nat-mpoly-eq*: *of-nat* n = *Const* (*of-nat* n)
⟨*proof*⟩

lemma *insertion-of-nat* [*simp*]: *insertion* f (*of-nat* n) = *of-nat* n
⟨*proof*⟩

lemma *insertion-monom-of-set* [*simp*]:
insertion f (*monom* (*monom-of-set* X) c) = $c * (\prod i{\in}X.\ f\ i)$
⟨*proof*⟩

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

lemma *sym-mpoly-0* [*simp*]:
assumes *finite* A
shows *sym-mpoly* $A\ 0 = 1$
⟨*proof*⟩

lemma *sym-mpoly-eq-0* [*simp*]:
assumes $k >$ *card* A
shows *sym-mpoly* $A\ k = 0$
⟨*proof*⟩

lemma *coeff-sym-mpoly-monom-of-set-eq-0*:
assumes *finite* $X\ Y \subseteq X$ *card* $Y \neq k$
shows *MPoly-Type.coeff* (*sym-mpoly* $X\ k$) (*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 *MPoly-Type.coeff* (*sym-mpoly* $X\ k$) (*monom-of-set* Y) $= 0$
⟨*proof*⟩

1.2 The set of roots of a univariate polynomial

lift-definition *poly-roots* :: *'a* :: *idom poly* ⇒ *'a multiset* **is**
$\lambda p\ x.$ *if* $p = 0$ *then* 0 *else order* $x\ p$
⟨*proof*⟩

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

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

lemma *count-poly-roots* [*simp*]:
assumes $p \neq 0$
shows *count* (*poly-roots* p) $x =$ *order* $x\ p$
⟨*proof*⟩

lemma *in-poly-roots-iff* [*simp*]: $p \neq 0 \Longrightarrow x \in\#$ *poly-roots* $p \longleftrightarrow$ *poly* $p\ x = 0$
⟨*proof*⟩

lemma *set-mset-poly-roots*: $p \neq 0 \Longrightarrow$ *set-mset* (*poly-roots* p) $= \{x.$ *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 ⋀*x*. *x* ∈ *A* ⟹ *f x* ≠ *0*
shows *poly-roots* (*prod f A*) = (∑ *x*∈*A*. *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 $p \equiv$ *Polynomial.smult c* (*prod-mset* (*image-mset* ($\lambda x.$ [:$-x$, *1*:]) *A*))
  shows *poly-roots p* = *A*
⟨*proof*⟩

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*⟩

lemma *poly-roots-eq-imp-eq*:
  fixes *p q* :: *complex poly*
  assumes *Polynomial.lead-coeff p* = *Polynomial.lead-coeff q*
  assumes *poly-roots p* = *poly-roots q*
  shows *p* = *q*
⟨*proof*⟩

lemma *Sum-any-zeroI′*: ($\bigwedge x.$ *P x* $\Longrightarrow$ *f x* = *0*) $\Longrightarrow$ *Sum-any* ($\lambda x.$ *f x when P x*) = *0*
  ⟨*proof*⟩

lemma *sym-mpoly-insert*:
  assumes *finite X* $x \notin X$
  shows (*sym-mpoly* (*insert x X*) (*Suc k*) :: ′*a* :: *semiring-1 mpoly*) =
        monom (*monom-of-set* {*x*}) *1* ∗ *sym-mpoly X k* + *sym-mpoly X* (*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 $X_1, \ldots, X_n$, we define the k-th power sum polynomial as

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

lift-definition *powsum-mpoly-aux* :: *nat set* ⇒ *nat* ⇒ (*nat* $\Rightarrow_0$ *nat*) $\Rightarrow_0$ *'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
⟨*proof*⟩

lemma *lookup-powsum-mpoly-aux*:
Poly-Mapping.lookup (powsum-mpoly-aux X 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)
⟨*proof*⟩

lemma *lookup-sym-mpoly-aux-monom-singleton* [*simp*]:
assumes *finite X x ∈ 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 ∈ X then 1 else 0)*
⟨*proof*⟩

lemma *keys-powsum-mpoly-aux*: *m ∈ keys (powsum-mpoly-aux A k) ⟹ keys m ⊆ A*
⟨*proof*⟩

lift-definition *powsum-mpoly* :: *nat set* ⇒ *nat* ⇒ *'a* :: {*semiring-1*,*zero-neq-one*} *mpoly* **is**
powsum-mpoly-aux ⟨*proof*⟩

lemma *vars-powsum-mpoly-subset*: *vars (powsum-mpoly A k) ⊆ A*
⟨*proof*⟩

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

lemma *coeff-powsum-mpoly*:

MPoly-Type.coeff (*powsum-mpoly X 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*)
⟨*proof*⟩

lemma *coeff-powsum-mpoly-0-right*:
MPoly-Type.coeff (*powsum-mpoly X 0*) *mon* = (*if mon* = *0 then of-nat* (*card X*) *else 0*)
⟨*proof*⟩

lemma *coeff-powsum-mpoly-singleton*:
assumes *finite X k* > *0*
shows *MPoly-Type.coeff* (*powsum-mpoly X k*) (*Poly-Mapping.single x k*) = (*if x* ∈ *X then 1 else 0*)
⟨*proof*⟩

lemma *coeff-powsum-mpoly-singleton-eq-1* [*simp*]:
assumes *finite X x* ∈ *X k* > *0*
shows *MPoly-Type.coeff* (*powsum-mpoly X k*) (*Poly-Mapping.single x k*) = *1*
⟨*proof*⟩

lemma *coeff-powsum-mpoly-singleton-eq-0* [*simp*]:
assumes *finite X x* ∉ *X k* > *0*
shows *MPoly-Type.coeff* (*powsum-mpoly X k*) (*Poly-Mapping.single x k*) = *0*
⟨*proof*⟩

lemma *powsum-mpoly-0* [*simp*]: *powsum-mpoly X 0* = *of-nat* (*card X*)
⟨*proof*⟩

lemma *powsum-mpoly-empty* [*simp*]: *powsum-mpoly* {} *k* = *0*
⟨*proof*⟩

lemma *powsum-mpoly-altdef*: *powsum-mpoly X k* = ($\sum$ *x*∈*X*. *monom* (*Poly-Mapping.single x k*) *1*)
⟨*proof*⟩

Power sum polynomials are symmetric:

lemma *symmetric-powsum-mpoly* [*intro*]:
assumes *A* ⊆ *B*
shows *symmetric-mpoly A* (*powsum-mpoly B k*)
⟨*proof*⟩

lemma *insertion-powsum-mpoly* [*simp*]: *insertion f* (*powsum-mpoly X k*) = ($\sum$ *i*∈*X*. *f i* ^ *k*)
⟨*proof*⟩

lemma *powsum-mpoly-nz*:

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

lemma *powsum-mpoly-eq-0-iff*:
assumes *k > 0*
shows *powsum-mpoly X k* = *0* $\longleftrightarrow$ *infinite X* $\vee$ *X* = {}
⟨*proof*⟩

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 $X_1, \ldots, X_n$. This is the sum of all monomials that can be formed by taking the product of k distinct variables.

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

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* ∗ *of-nat k* ∗ *sym-mpoly X k* +
(∑ *i<k*. (*−1*) ^ *i* ∗ *sym-mpoly X i* ∗ *powsum-mpoly X* (*k* − *i*)) =
(*0* :: *'a* :: *comm-ring-1 mpoly*)
(**is** *?lhs* = *0*)
⟨*proof*⟩

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 $e_0, \ldots, e_n$.

corollary *Girard-Newton'*:
assumes *finite X* **and** *k > card X*
shows (∑ *i≤card X*. (*−1*) ^ *i* ∗ *sym-mpoly X i* ∗ *powsum-mpoly X* (*k* − *i*)) =
(*0* :: *'a* :: *comm-ring-1 mpoly*)
⟨*proof*⟩

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

corollary *sym-mpoly-recurrence*:
assumes *k*: *k > 0* **and** *finite X*
shows (*sym-mpoly X k* :: *'a* :: *field-char-0 mpoly*) =

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

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

corollary *powsum-mpoly-recurrence*:
assumes *k*: $k > 0$ **and** *X*: *finite X*
shows (*powsum-mpoly X k* :: *'a* :: *comm-ring-1 mpoly*) =
$(-1)\ \hat{}\ (k + 1) * of\text{-}nat\ k * sym\text{-}mpoly\ X\ k -$
$(\sum i{=}1..{<}k.\ (-1)\ \hat{}\ i * sym\text{-}mpoly\ X\ i * powsum\text{-}mpoly\ X\ (k - i))$
⟨*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*) =
$-(\sum i{=}1..card\ X.\ (-1)\ \hat{}\ i * sym\text{-}mpoly\ X\ i * powsum\text{-}mpoly\ X\ (k - i))$
⟨*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 $x_1, \ldots, x_n$, define $p_k = x_1^k + \ldots + x_n^k$. Also, define $e_k := e_k(x_1, \ldots, x_n)$ where $e_k(X_1, \ldots, X_n)$ 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* ⇒ *complex*
fixes *n* :: *nat*
begin

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

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

lemma *p-0* [*simp*]: *p 0 = of-nat n*
⟨*proof*⟩

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

Similarly, we introduce the notation $e_k = e_k(x_1, \ldots, x_n)$ where $e_k(X_1, \ldots, X_n)$ 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* = (∑ *Y* | *Y* ⊆ {..<*n*} ∧ *card Y = k. prod x Y*)

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

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

lemma *e-eq-0* [*simp*]: *k > n* ⟹ *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* = −(∑ *i=1..k.* (− *1*) ^ *i* ∗ *e* (*k* − *i*) ∗ *p i*) / *of-nat k*
⟨*proof*⟩

lemma *p-recurrence*:
assumes *k*: *k > 0*
shows *p k* = −*of-nat k* ∗ (−*1*) ^ *k* ∗ *e k* − (∑ *i=1..<k.* (−*1*) ^ *i* ∗ *e i* ∗ *p* (*k* − *i*))
⟨*proof*⟩

lemma *p-recurrence''*:
assumes *k*: *k > n*
shows *p k* = −(∑ *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 ⋀*k. k* ∈ {*1..n*} ⟹ *p k* ∈ ℚ
shows *e k* ∈ ℚ
⟨*proof*⟩

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

lemma *p-in-Rats*:
assumes $\bigwedge k.\ k \in \{1..n\} \Longrightarrow p\ k \in \mathbb{Q}$
shows $p\ k \in \mathbb{Q}$
⟨*proof*⟩

Next, we define the unique monic polynomial that has $x_1, \ldots, x_n$ 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 $x_1, \ldots, x_n$ 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 $p_1, \ldots, p_n$, 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*⟩

lemma *Q-altdef*: $Q = (\sum k{\le}n.$ *Polynomial.monom* $((-1)$ ^ $(n - k) * e\ (n - k))$ $k)$
⟨*proof*⟩

The following theorem again shows that $x_1, \ldots, x_n$ 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*⟩

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 $x_1, \ldots, x_n$ so that $x_1^k + \ldots + x_n^k = f_k$ for any $1 \le k \le n$.

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*
  ⟨*proof*⟩

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

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*⟩

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

lift-definition Q' :: *complex poly* **is** λ*k*. *if k* > *n then 0 else* (−*1*) ^(*n* − *k*) ∗ e' (*n* − *k*)
  ⟨*proof*⟩

lemma *coeff-*Q': *Polynomial.coeff* Q' *k* = (*if k* > *n then 0 else* (−*1*) ^(*n* − *k*) ∗ e' (*n* − *k*))
  ⟨*proof*⟩

lemma *lead-coeff-*Q': *Polynomial.coeff* Q' *n* = *1*
  ⟨*proof*⟩

lemma *degree-*Q' [*simp*]: *Polynomial.degree* Q' = *n*
⟨*proof*⟩

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*⟩

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

sublocale *power-sum-puzzle Root n* ⟨*proof*⟩

Vieta's theorem gives us an expression for the coefficients of Q' in terms of $e_k(x_1, \ldots, x_n)$. 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 ≤ 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* :: *nat ⇒ complex*
  shows ∃ *Root*. ∀ *k*∈{*1..n*}. (∑ *i*<*n*. *Root i* ^ *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* :: *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* ⟹ *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* ∈ 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*#} = *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*: *set-mset* (*poly-roots* [:−*1*/*6*, −*1*/*2*, −*1*, *1*::*complex*:]) ∩ 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*: *rsquarefree* [:−*1*/*6*, −*1*/*2*, −*1*, *1* :: *complex*:]
⟨*proof*⟩

lemma *distinct-xyz*: *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.