Pencil, Chalkboard, Q.E.D.

Hand-Derived Vieta Formulas Checked in ACL2

Pablo O. Cayuela

UTN Facultad Regional Córdoba, Argentina

ACL2 Workshop 2026 — Rump session

Step 1 — the table of successive multiplication

Control design: poles are chosen, the characteristic polynomial $P(s) = \prod(s - a_i)$ is needed, with $b_i = (-1)^{n-i} e_{n-i}(a)$ (Cardano–Vieta).

Multiply in one factor $(s - a_n)$ at a time: shift the row, scale it by $-a_n$, add.

Example, roots $3, -5, -11, -1$; last step $\times(s+1)$:

	s^4	s^3	s^2	s^1	s^0
$\times s$	1	13	7	-165	
$\times(-a_4)$		1	13	7	-165
sum	1	14	20	-158	-165

$$b_p^{(n)} = b_{p-1}^{(n-1)} - a_n\, b_p^{(n-1)}, \qquad b_n^{(n)} = 1$$

Each step is synthetic division (Ruffini) run backwards: multiply by $(s - a_n)$ instead of dividing.

Completeness check: the coefficient of s^{n-h} has $\binom{n}{h}$ products, a row of Pascal's triangle.

Step 2 — how the closed formulas were found

Two unfoldings of the recurrence: $m = n - p$ nested choices of roots, $p = 0, \ldots, n$.

Descending reading (handwritten sheets): each index lower than the previous:

$$b_p = (-1)^m \prod_{j=1}^{m} \sum_{i_j = m-j+1}^{i_{j-1}-1} a_{i_j}, \qquad i_0 \equiv n+1$$

Ascending reading (chalkboard): each index higher than the previous:

$$b_p = (-1)^m \prod_{j=1}^{m} \sum_{i_j = i_{j-1}+1}^{n-m+j} a_{i_j}, \qquad i_0 \equiv 0$$

Same sum? The mirror $i \mapsto n+1-i$ seems to turn one reading into the other.

Checked by: exact symbolic expansion for $n = 1, \ldots, 7$. An earlier draft fixed $i_1 = n$; the check caught it.

Equivalent to eqs. (6)–(7) of the CVM 2026 paper, with $q = p + j - 1$.

Step 3 — from "checked for $n \leq 7$" to "for every n"

- Polynomial = coefficient list, constant term first; Horner evaluation.
- The table step is *literally* a function:

```
(defun mult-by-root (coeffs a)            ; shift + scale
  (poly-add (poly-shift coeffs) (poly-scale coeffs (- a))))
(defun poly-from-roots (roots)            ; the table, a1 multiplied last
  (if (endp roots) (list 1)
    (mult-by-root (poly-from-roots (cdr roots)) (car roots))))
(defun prod-factors (roots x)             ; obviously-correct spec
  (if (endp roots) 1 (* (- x (car roots)) (prod-factors (cdr roots) x))))
```

- Specification = the plain product $\prod(s - a_i)$. No formula of mine is trusted as a spec.
- One induction on the list of roots = all degrees at once: soundness, degree, monicity (Theorem 1) with no hints.
- Certified in ACL2 *and* ACL2(r): statements range over all ACL2 numbers, so in ACL2(r) they reach irrational and complex poles: $\pm\sqrt{2} \mapsto s^2 - 2$, $-1 \pm j\sqrt{2} \mapsto s^2 + 2s + 3$ (certified).

Cardano–Vieta: the coefficients are the e_k

$e_k(a_1, \ldots, a_n)$ = sum of all products of k *distinct* roots. For (a, b, c):

$$(s-a)(s-b)(s-c) = s^3 - \underbrace{(a+b+c)}_{e_1} s^2 + \underbrace{(ab+ac+bc)}_{e_2} s - \underbrace{abc}_{e_3}$$

In ACL2: a product either includes the first root, or it does not.

```
(defun elem-sym (k lst)
  (cond ((zp k) 1) ((endp lst) 0)
        (t (+ (* (car lst) (elem-sym (1- k) (cdr lst)))   ; includes a1
              (elem-sym k (cdr lst))))))                  ; does not
```

Theorem (`cardano-vieta`), for every n and $0 \le k \le n$:
$b_k = (-1)^{n-k}\, e_{n-k}(a_1, \ldots, a_n)$ (certified)

- Corollary: `elem-sym` is symmetric, invariant under any permutation of the roots.

Level 1 — each hand-derived reading gives b_p

Write $D_p(a)$ and $A_p(a)$ for the descending and ascending nested sums of Step 2.

Theorems (`eq6-descending-reading`, `eq7-ascending-reading`, `eq8-readings-agree`), for every n and every $0 \le p \le n$:

$$b_p \;=\; (-1)^{n-p}\, D_p(a) \;=\; (-1)^{n-p}\, A_p(a) \qquad \text{(certified)}$$

- D_p and A_p are ACL2 functions with *exactly* the nested structure of the formulas: one recursive call per level of the product, one per step of the sum.
- Proof: each reading equals e_{n-p}, then Cardano–Vieta. Key lemma: e_k of a prefix peels off its last root, $e_k(a_1..a_r) = e_k(a_1..a_{r-1}) + a_r\, e_{k-1}(a_1..a_{r-1})$.
- This settles the *value*. It says nothing yet about the *terms*.

Levels 2 and 3 — the same terms, each exactly once

Each summand is the product of the roots picked by a chain of indices $(i_1, \ldots, i_m)$, $m = n - p$. **Theorems**, for every n and p (certified):

1. Each reading is the sum, over its chains, of $\prod_j a_{i_j}$.
2. **Mirror:** $i \mapsto n + 1 - i$ maps the descending chains *exactly* onto the ascending ones (in reverse order).
3. **Count:** each reading has $\binom{n}{m}$ chains (Pascal), none repeated.
4. **Which chains:** exactly the strictly decreasing (resp. increasing) chains of m indices from $1..n$: every set of m distinct roots appears exactly once.

Example $n = 4$, $p = 2$:

descending chains	(2,1)	(3,1)	(3,2)	(4,1)	(4,2)	(4,3)
mirror $i \mapsto 5 - i$	(3,4)	(2,4)	(2,3)	(1,4)	(1,3)	(1,2)

the second row is the list of ascending chains, read backwards.

Gaps found and closed

Gap	Now
"Cardano–Vieta" theorem never mentioned `elem-sym`	`cardano-vieta`: $b_k = (-1)^{n-k} e_{n-k}$ (certified)
Python cross-check used $(-1)^p$ instead of $(-1)^{n-p}$	Fixed; checked against direct expansion, even *and* odd n
Nested sums (6)–(7) checked only for $n \leq 7$	Both equal b_p for every n (certified)
Pascal count of terms checked by hand	$\binom{n}{m}$ terms, none repeated, every n (certified)
Mirror $i \mapsto n+1-i$ argued by hand	Term by term, exact (certified)

An outside check found a bug that the author's own numeric tests missed.

How it was built: an AI agent, with ACL2 as the referee

- Claude Code drives a live ACL2 session through an MCP server: propose → run → read the failed checkpoint → revise.
- The agent may write any proof script; only ACL2's "Q.E.D." counts.
- ACL2 validates the *statements as written*, certified from clean sources.
- It does not validate that the statements match the intent, nor the script, the figures, the prose.
- Lesson: a certified book is evidence about the *statements*; humans still audit the *claims*.

For discussion

1. **Is it already somewhere?** We proved that `elem-sym` is symmetric. A text search of the community books finds no development of elementary symmetric polynomials, Vieta's formulas or Newton–Girard identities. Has anyone done it, in ACL2 or elsewhere?
2. **The converse and the bridge.** Now verified, in separate books: Newton–Girard on lists of numbers, and, for upper-triangular matrices, the characteristic polynomial computed from the traces (ACL2 and ACL2(r)); determinant = Leibniz's (ACL2). Open: general matrices, and the fundamental theorem of symmetric polynomials. Which multivariate representation would make it tractable? Medina et al. (2000) have sparse polynomials with a normal form, but no substitution or variable permutation.
3. Extracting a *single* coefficient in $\Theta(p(n-p))$: verify the truncated recurrence?
4. How should we report AI-assisted ACL2 development so that readers see what was and wasn't checked?

Books: github.com/pcayuela/acl2-rump-poly-roots

branches `rumpsession`, `newton-girard`, `traces`, `leibniz`

pcayuela@frc.utn.edu.ar

Theorem 1 — the table is sound

```
(defthm poly-from-roots-correct
  (equal (poly-eval (poly-from-roots roots) x)
         (prod-factors roots x)))            ; no hints needed
(defthm len-poly-from-roots
  (equal (len (poly-from-roots roots)) (+ 1 (len roots))))
(defthm car-last-poly-from-roots-is-one       ; monic
  (equal (nth (len roots) (poly-from-roots roots)) 1))
```

- Soundness, degree and monicity: proved by the induction that the definition suggests.
- Library lemmas on `poly-add` needed `true-listp`/`number-listp`: `(+ nil 0)` $= 0$ makes some equations true numerically but not as `equal`.

Theorem 3 — the order of the roots does not matter

The two orders of unfolding are the same product of linear factors, in opposite order:

```
(defthm mult-by-root-commutes          ; (s-a)(s-b)P = (s-b)(s-a)P
  (implies (true-listp p)
           (equal (mult-by-root (mult-by-root p a) b)
                  (mult-by-root (mult-by-root p b) a))))

(defthm poly-from-roots-permutation-invariant   ; any order, not only reversal
  (implies (perm x y)
           (equal (poly-from-roots x) (poly-from-roots y))))
```

- The only non-automatic proof: restate `mult-by-root` as add/shift/scale, then a `:use` chain.
- `perm`: the community definition (`kestrel/lists-light/perm`). Pull a root out of the middle (`poly-from-roots-remove1`), apply it last.
- `(defcong perm equal (poly-from-roots x) 1)`: ACL2 now reorders the roots by itself inside any expression.
- Reversal is a special case: `(perm (rev x) x)` by one induction, and the reversal theorem is re-derived from this one.

ACL2(r): irrational and complex roots

The theorems quantify over all ACL2 numbers. Same books, certified unchanged in ACL2(r):

- Plain ACL2 has no number whose square is 2 (`there-is-no-sqrt-2`, `nonstd/nsa/sqrt-2`).
- In ACL2(r): `(acl2-sqrt 2)` is real and irrational (`sqrt-2-is-irrational`).
- Roots $\pm\sqrt{2}$: the table gives $s^2 - 2$.
- Conjugate poles $\sigma \pm j\omega$, any reals: $s^2 - 2\sigma s + (\sigma^2 + \omega^2)$.
- Poles $-1 \pm j\sqrt{2}$: $s^2 + 2s + 3$.

Book `poly-roots-acl2r-examples.lisp`, marked `cert_param: (uses-acl2r)`.

The chains in ACL2

```
(defun desc-ch (k i m)          ; same recursion as the descending sum
  (cond ((zp k) (list nil))
        ((or (not (natp i)) (not (natp m)) (< m i)) nil)
        (t (append (cons-each i (desc-ch (- k 1) (- k 1) (- i 1)))
                   (desc-ch k (+ i 1) m)))))

(defthm readings-are-mirror-images
  (implies (and (natp p) (natp n) (<= p n))
           (equal (mirror-chains (+ n 1) (desc-ch (- n p) (- n p) n))
                  (rev (asc-ch (- n p) 1 (+ p 1))))))
```

- The proof needed an auxiliary enumeration of the increasing chains that runs its first index *downwards*, to match the mirrored descending recursion step by step.
- Book `poly-roots-nested-chains.lisp`.