Towards Verified Polynomial Factorisation

James Davenport
masjhd@bath.ac.uk

University of Bath

Thanks to IPAM at UCLA for prompting this, and many colleagues, at Dagshuhl 23401, CICM 2023 [Dav23], the Hausdorff Institute for Mathematics, and elsewhere, for input
Partially supported by EPSRC grant EP/T015713, and Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) under Germany's Excellence Strategy – EXC-2047/1 – 390685813.

16 September 2024
See https://arxiv.org/abs/2409.09533

Trusting Software

It would be nice to trust software.

- Gödel's Incompleteness implies unconditional trust is essentially impossible
- Theorem provers (Lean etc.) tend to have a small kernel (ideally 100s of lines), and the rest (tactics etc.) do not need to be trusted, as an error there results in a "proof" that the kernel rejects.
- It is possible to produce programs (e.g. UK's National Air Traffic System) with 100Ks of lines, with formally proved properties.

But the proof is constructed along with the program: retrospectively proving such a program correct is probably impossible (and the program is probably incorrect — see [Cha22] for a well-tested 100-tweet program).

Also computer algebra programs are 10Ms of lines, going back 40 years or more.

"Verify the Computer Algebra system" is probably impossible.

Trusting Instances of Software

So what's plan B?

- If we can't prove the program correct, can we prove that *this instance* is correct?
- This isn't as new an idea as I thought it was: [MMNS11].

  A certifying algorithm is an algorithm that produces, with each output, a certificate or witness (easy-to-verify proof) that the particular output has not been compromised by a bug. A user of a certifying algorithm inputs x, receives the output y and the certificate w, and then checks, either manually or by use of a program, that w proves that y is a correct output for input x.

- This is particularly relevant when y contains negative assertions.

SAT solving, and verifying UNSAT

The quintessenial NP-complete problem [Coo66]:

Problem

Given a Boolean statement $\Phi(x_1,\ldots,x_n)$ *produce*

either $f : \{x_i\} \mapsto \{T, F\}$ *such that*
$\Phi(f(x_1),\ldots,f(x_n)) = T$ *(a satisfying assignment)*

or $\perp$ *indicating that no satisfying assignment exists.*

The first can be verified easily enough: what about the second? Since at least 2016, contestants in the annual SAT contests have been required to produce proofs (occasionally 2PB! [Heu18]) in DRAT format, which can be checked ([Heu23] says there are subtleties to "easy" checking).

Polynomial Factorisation

The base case is polynomials in $\mathbf{Z}[x]$.

Problem (Factorisation)

Given $f \in \mathbf{Z}[x]$, write $f = \prod f_i$ where the f_i are irreducible elements of $\mathbf{Z}[x_1, \ldots, x_n]$.

Verifying that $f = \prod f_i$ is, at least relatively, easy. The hard part (mentioned, but not addressed, in [BP99, §6.3]) is verifying that the f_i are *irreducible*.
The author knows of no implementation of polynomial factorisation that produces any evidence, let alone a proof, of this.

Problem (Factorisation in the SAT style)

Given $f \in \mathbf{Z}[x_1, \ldots, x_n]$, produce

either a proper factor g of f,

or $\perp$ indicating that no such g exists.

Square-free and outsourcing

Texts and articles begin "We may as well assume f is square-free", i.e. that $\gcd(f, f') = 1$ (or more accurately that it has degree 0).
This would involve implementing Euclid's Algorithm in our theorem prover *and* proving it correct.
Alternatively we could outsource this: the algebra system could emit λ, μ such that $\lambda f + \mu f'$ has degree 0, and the theorem prover could verify this.
There is an efficient algorithm for the verification: [Dav24, §IV.C].

Factorisation Algorithm

The basic algorithm goes back to [Zas69]: step M is a later addition [Mus75], and the H' variants are also later.

1. Choose a prime p (not dividing the leading coefficient of f) such that $f \pmod{p}$ is also square-free.
2. Factor f modulo p as $\prod f_i^{(1)} \pmod{p}$: f_i irreducible.

M Take five p and compare the factorisations.

3. If f can be shown to be irreducible from modulo p factorisations, return f.
4. Let B be such that any factor of f has coefficients less than B in magnitude, and n such that $p^n \geq 2B$. [Landau–Mignotte]
5. Use Hensel's Lemma to lift the factorisation to $f = \prod f_i^{(n)} \pmod{p^n}$

H Starting with singletons and working up, take subsets of the $f_i^{(n)}$, multiply them together and check whether, regarded as polynomials over $\mathbf{Z}$ with coefficients in $[-B, B]$, they divide f — if they do, declare that they are irreducible factors of f.

Algorithm Notes

H' Use some alternative technique, originally [LLL82], but now e.g. [ASZ00, HvHN11] to find the true factor corresponding to $f_1^{(n)}$, remove $f_1^{(n)}$ and the other $f_i^{(n)}$ corresponding to this factor, and repeat.

In practice, there are a lot of optimisations, which would greatly complicate a proof of correctness of an implementation of this algorithm.

We found that, although the Hensel construction is basically neat and simple in theory, the fully optimised version we finally used was as nasty a piece of code to write and debug as any we have come across [MN81].

Since if f is irreducible modulo p, it is irreducible over the integers, the factors produced from singletons in step 7 are easily proved to be irreducible. Unfortunately, the chance that an irreducible polynomial of degree n is irreducible modulo p is $1/n$.

Algorithm Notes [Dav23]

A factorisation algorithm could, even though no known implementation does, relatively easily produce the required information for a proof of irreducibility unless the recombination step is required.

Note that *verifying* the Hensel lifting, the “nasty piece” from [MN81] is easy: the factors just have to have the right degrees from the factorisation of $f \pmod p$ and multiply to give $f \pmod{p^n}$.

Building test cases for the various edge cases was extremely difficult.

Step [H] is relatively easy to verify: this combination divides and no smaller combination divides. The variants in [H'] are interesting: I have not found an easy route.

If [H'] finds a factor that is a product of k p-adic factors, then we can use [H] to verify this by checking that the $2^k - 2$ subsets do not give factors.

But if [H'] says “irreducible”, I know no easy proof.

Further Reflections

M Take five p and compare the factorisations.

Not just "take the best". Rather we look for incompatibilities, so if a degree 4 factors as 3,1 modulo one prime and 2,2 modulo another, it's actually irreducible, and so on.

? What's the best division of labour between the algebra system and the theorem prover?

[Mus75] suggests taking five primes, though more recently [LP97] show that, if the Galois group is S_n, seven is asymptotically right. For any degree d, the probability that a random polynomial with coefficients $\leq H$ has Galois group S_n tends to 1 as H tends to infinity. [DS00] looks at other Galois groups.

Special case: Irreducible f

It is clearly sufficient, even if not efficient, to proceed as follows.

1) Ask the algebra system for a factorisation

$$f = \prod_{i=1}^{k} f_i. \tag{1}$$

2) If $k > 1$ verify that this is a factorisation, i.e that (1) is true.
3) For $i = 1 \ldots k$ do:

3.1) Ask the algebra system for hints, essentially a certificate that f_i is irreducible;

3.2) Verify these hints.

The inefficiency comes from the fact that step 3.1 is recomputing things, or variants of things, that were computed in step 1.

Proving that f is irreducible modulo p

There are various routes: [Ber67], [Ber70] and [CZ81]: currently experimenting with the last of these.

Theorem

A square-free polynomial f of degree d is irreducible modulo prime $p \neq 2$ if $\forall i \in [1, \lceil d/2 \rceil]$, $\gcd(f, x^{p^i} - x) = 1$.

But we can't directly use the "gcd $= 1$ verifying" trick above directly, because we don't want to compute the co-factors of $f, x^{p^i} - x$.

Rather, let $f^* = x^{p^i} - x \pmod f$ computed by both theorem prover and computer algebra system, by repeated squaring $\pmod f$. Then the algebra system gives λ, μ such that $\lambda f + \mu f^* = 1$.

A: The Simple Certificate

This consists of a prime p, and the assertion that f is irreducible modulo p. This is wonderful if it works, but there two obstacles.
The first is that we may not find such a p easily: if the Galois group of f is the symmetric group S_n, the probability of f being irreducible modulo a prime p is $1/n$.
The second is that such p may not even exist: [SD69] shows how to construct f with no such p.

B: The pre-Musser Certificate

This consists of a prime p, a number n, and a set of polynomials $f_j \in \mathbf{Z}[x]$ together with the following assertions.

1. $f = \prod f_j \pmod{p^n}$.
2. Each f_j, considered as a polynomial modulo p, is irreducible.
3. Any factor of f over $\mathbf{Z}$ must have coefficients $< p^n/2$ in absolute value. [Landau–Mignotte Bound]
4. No proper (nontrivial) subset $\{f_k\} \subsetneq \{f_j\}$ has the property that $\prod_k f_k$, considered as a polynomial in $\mathbf{Z}[x]$ with coefficients $< p^n/2$ in absolute value, is a factor of f.

C: The Simple post-Musser Certificate

This consists of a number k, a set of primes $p_i : i = 1 \ldots k$, and some sets of polynomials $\{f_{i,j} : j = 1 \ldots n_i\} i = 1 \ldots k$ together with the following assertions.

1. For every i, $f = \prod f_{i,j} \pmod{p_i}$.
2. Each $f_{i,j}$, considered as a polynomial modulo p_i, is irreducible.
3. For each $k : 0 < k < \deg f$ there is an i such that the factorisation $f = \prod f_{i,j} \pmod{p_i}$ is incompatible with a factorisation of f as a degree k polynomial and a $\deg f - k$ co-factor.

The classic example is when a degree 4 polynomial factors modulo p_1 as two irreducible quadratics and module p as a linear times an irreducible cubic. Then the two quadratics rule out $k = 1, 3$ and p_2 rules out $k = 2$.

But the Swinnerton-Dyer polynomials [SD69] are examples where condition 3 may never be met. The simplest example is $x^4 + 1$ which is irreducible, but factors as two quadratics, or more, modulo every prime.

D: Complex post-Musser Certificate

While initially researching this project, using FLINT [FLI23] as our computer algebra system, [DBCC24] discovered that this can generate a more complex proof of irreducibility. It consists of the union of the data of the two previous certificates, and their assertions, except that the last assertions of each are merged to give

1. For each $k : 0 < k < \deg f$ for which there isn't i such that the factorisation $f = \prod f_{i,j} \pmod{p_i}$ is incompatible with a factorisation of f as a degree k polynomial and a $\deg f - k$ co-factor, all subsets $\{f_\ell\} \subsetneq \{f_j\}$ such that $k = \sum \deg f_\ell$, have the property that $\prod_\ell f_\ell$, considered as a polynomial in $\mathbf{Z}[x]$ with coefficients $< p^n/2$ in absolute value, is not a factor of f.

In the case of a Swinnerton-Dyer polynomial, the Musser clause doesn't buy us anything, and we revert to checking all combinations.

But there are polynomials in the FLINT test suite for which this gives a significant improvement.

Landau–Mignotte Bound I

We use $||\cdot||$ for the L_2 norm of a polynomial.

Lemma ([Mig74])

Let $P(X)$ be a polynomial with complex coefficients and α be a nonzero complex number. Then

$$||(X+\alpha)P(X)|| = |\alpha|||(X+\overline{\alpha}^{-1})P(X)||.$$

[Mig74] uses the notation $a_{-1} = a_{m+1} = 0$, and writes

$$\begin{aligned} P(X) &= \sum_{k=0}^{m} a_k X^k, \\ Q(X) &= (X+\alpha)P(X) = \sum_{k=0}^{m+1}(a_{k-1}+\alpha a_k)X^k \\ R(X) &= (X+\overline{\alpha}^{-1})P(x) = \sum_{k=0}^{m+1}(a_{k-1}+\overline{\alpha}^{-1}a_k)X^k \end{aligned}$$

Landau–Mignotte Bound II

$$||Q||^2 = \sum_{k=0}^{m+1} |a_{k-1} + \alpha a_k|^2 = \sum_{k=0}^{m+1} (a_{k-1} + \alpha a_k)\overline{(a_{k-1} + \alpha a_k)}$$

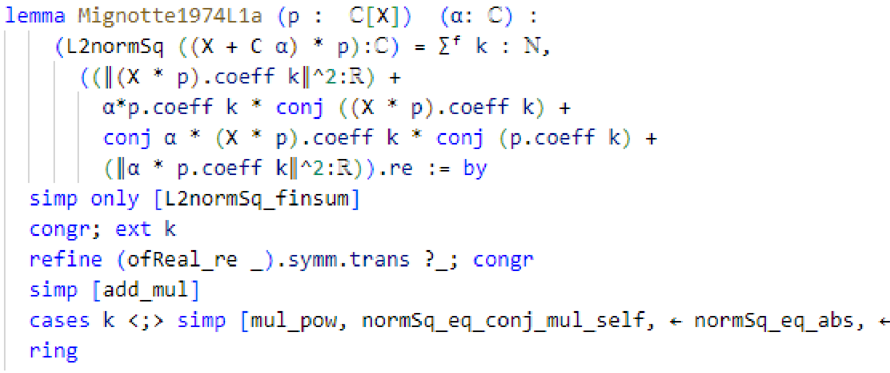
which expands to

$$\sum_{k=0}^{m+1} \left(|a_{k-1}|^2 + \alpha a_k \overline{a_{k-1}} + \overline{\alpha} a_{k-1} \overline{a_k} + |\alpha^2||a_k|^2\right). \qquad (2)$$

This is accomplished in Lean by the following code.

Landau–Mignotte Bound III

Figure: Mignotte Lemma 1a in Lean



```
lemma Mignotte1974L1a (p : ℂ[X]) (α: ℂ) :
    (L2normSq ((X + C α) * p):ℂ) = ∑ᶠ k : ℕ,
      ((‖(X * p).coeff k‖^2:ℝ) +
        α*p.coeff k * conj ((X * p).coeff k) +
        conj α * (X * p).coeff k * conj (p.coeff k) +
        (‖α * p.coeff k‖^2:ℝ)).re := by
  simp only [L2normSq_finsum]
  congr; ext k
  refine (ofReal_re _).symm.trans ?_; congr
  simp [add_mul]
  cases k <;> simp [mul_pow, normSq_eq_conj_mul_self, ← normSq_eq_abs, ←
  ring
```

Landau–Mignotte Bound IV

$$\sum_{k=0}^{m+1} \left(|a_{k-1}|^2 + \alpha a_k \overline{a_{k-1}} + \overline{\alpha} a_{k-1} \overline{a_k} + |\alpha^2||a_k|^2\right). \qquad (2)$$

[Mig74] then says "Expanding $|a|^2||R||^2$ yields the same sum". However, if we expand $|a|^2||R||^2$ naively as above, we actually get

$$\sum_{k=0}^{m+1} \left(|\alpha^2||a_{k-1}|^2 + \alpha a_k \overline{a_{k-1}} + \overline{\alpha} a_{k-1} \overline{a_k} + |a_k|^2\right). \qquad (3)$$

In general (2) and (3) are different: the $|\alpha|^2$ multiplies different terms. And indeed, for any k the index-k summands in (2) and (3) do differ. However, it is legitimate to re-express (2) as:

$$\sum_{k=0}^{m+1} \left(|a_{k-1}|^2\right) + \sum_{k=0}^{m+1} \left(\alpha a_k \overline{a_{k-1}} + \overline{\alpha} a_{k-1} \overline{a_k}\right) + \sum_{k=0}^{m+1} \left(|\alpha^2||a_k|^2\right)$$

Landau–Mignotte Bound V

and then as

$$||P||^2 + \sum_{k=0}^{m+1} (\alpha a_k \overline{a_{k-1}} + \overline{\alpha} a_{k-1} \overline{a_k}) + |\alpha^2| ||P||^2. \tag{4}$$

A similar operation on (3) gives (5) :

$$|\alpha^2| ||P||^2 + \sum_{k=0}^{m+1} (\alpha a_k \overline{a_{k-1}} + \overline{\alpha} a_{k-1} \overline{a_k}) + ||P||^2, \tag{5}$$

and now the equality between (4) and (5) is obvious to humans
This "similarly" took days to solve, and the "obvious" currently isn't to Lean.

Current State

Landau–Mignotte	Still trying to complete the proof in Lean.
Polynomials	(as programs rather than abstract objects) Working on their implementation.
$\mathbf{Z}$ and $\mathbf{Z}/p\mathbf{Z}$ relationship	Trying to work out best route.
Outsourcing	Is done elsewhere [Mac24, **?**] and my other talk.

Thank you, and jobs

- Any questions?
- Permanent (subject to probation) jobs at Bath:
- ED11636 Lecturers in Computer Science — Jobs at Bath
- `https://www.bath.ac.uk/jobs/Vacancy.aspx?id=25307&forced=1`
- we are particularly looking for individuals with research interests in areas around formal mathematics and computer assisted reasoning, including but not limited to:
- proof assistants (e.g. Agda, Coq, Isabelle, Lean)
- certified mathematical libraries
- logical systems, proof theory and type theory
- certified programming and program synthesis
- automated reasoning
- applications of AI and machine learning to formal mathematics

Bibliography I

J.A. Abbott, V. Shoup, and P. Zimmermann.
Factorization in $\mathbf{Z}[x]$: The Searching Phase.
In C. Traverso, editor, *Proceedings ISSAC 2000*, pages 1–7, 2000.

E.R. Berlekamp.
Factoring Polynomials over Finite Fields.
Bell System Tech. J., 46:1853–1859, 1967.

E.R. Berlekamp.
Factoring Polynomials over Large Finite Fields.
Math. Comp., 24:713–735, 1970.

C. Ballarin and L.C. Paulson.
A pragmatic approach to extending provers by computer algebra — with applications to coding theory.
Fundam. Informaticae, 39:1–20, 1999.

Bibliography II

R. Chapman.
Panel session at CPHC 2022.
https://people.bath.ac.uk/masjhd/CPHC2022/Publish/Chapman-cphc_2022_panel_chapman.pptx, 2022.

S.A. Cook.
On the minimum computation time of functions.
PhD thesis, Department of Mathematics Harvard University, 1966.

D.G. Cantor and H. Zassenhaus.
A New Algorithm for Factoring Polynomials over Finite Fields.
Math. Comp., 36:587–592, 1981.

Bibliography III

J.H. Davenport.
Proving an Execution of an Algorithm Correct?
In Catherine Dubois and Manfred Kerber, editors, *Proceedings CICM 2023*, volume 14101 of *Springer Lecture Notes in Computer Science*, pages 255–269, 2023.

J.H. Davenport.
First steps towards Computational Polynomials in Lean.
`https://arxiv.org/abs/2408.04564`, 2024.

James H. Davenport, Alex Best, Mario Carneiro, and Edgar Costa.
Formal verification of computer algebra (factorisation).
In Andrej Bauer, Katja Berčič, Florian Rabe, Nicolas Thiéry, and Jure Taslak, editors, *Automated mathematics: integrating proofs, algorithms and data (Dagstuhl Seminar 23401)*,

Bibliography IV

volume 13, page 16, Dagstuhl, Germany, 2024. Schloss Dagstuhl – Leibniz-Zentrum für Informatik.

J.H. Davenport and G.C. Smith.
Fast recognition of alternating and symmetric groups.
J. Pure Appl. Algebra, 153:17–25, 2000.

FLINT team.
FLINT: Fast Library for Number Theory, 2023.
Version 2.9.0, `https://flintlib.org`.

M.J.H. Heule.
Schur number five.
AAAI'18/IAAI'18/EAAI'18: Proceedings of the Thirty-Second AAAI Conference on Artificial Intelligence and Thirtieth Innovative Applications of Artificial Intelligence Conference. and Eighth AAAI Symposium on Educational Advances in Artificial Intelligence Article No.: 808, pages 6598–6606, 2018.

Bibliography V

M.J. Heule.
Organising SAT contests and DRAT proofs.
Personal comunication 15 February 2023, 2023.

W. Hart, M. van Hoeij, and A. Novocin.
Practical polynomial factoring in polynomial time.
In *Proceedings ISSAC 2011*, pages 163–170, 2011.

A.K. Lenstra, H.W. Lenstra Jun., and L. Lovász.
Factoring Polynomials with Rational Coefficients.
Math. Ann., 261:515–534, 1982.

T. Łuczak and L. Pyber.
On random generation of the symmetric group.
In *Proceedings Combinatorics geometry and probability*, pages 463–470, 1997.

Bibliography VI

H.R. Macbeth.
Algebraic computations in Lean.
https://hrmacbeth.github.io/computations_in_lean/, 2024.

M. Mignotte.
An Inequality about Factors of Polynomials.
Math. Comp., 28:1153–1157, 1974.

R.M. McConnell, K. Mehlhorn, S. Näher, and P. Schweitzer.
Certifying algorithms.
Computer Science Review, 5:119–161, 2011.

P.M.A. Moore and A.C. Norman.
Implementing a Polynomial Factorization and GCD Package.
In *Proceedings SYMSAC 81*, pages 109–116, 1981.

Bibliography VII

D.R. Musser.
Multivariate Polynomial Factorization.
J. ACM, 22:291–308, 1975.

H.P.F. Swinnerton-Dyer.
Letter to E.R. Berlekamp.
Mentioned in [Ber70], 1969.

H. Zassenhaus.
On Hensel Factorization I.
J. Number Theory, 1:291–311, 1969.