Proving an Execution of an Algorithm Correct? The case of Polynomial Factorisation

James Davenport
masjhd@bath.ac.uk
With Edgar Costo, Alex Best, Mario Carneiro

University of Bath
Thanks to IPAM at UCLA for prompting this, and many colleagues, especially at Dagstuhl seminar 23401, for input
Partially supported by EPSRC grant EP/T015713

5 October 2023

General Situation

Do I believe the output from my (complicated, optimised, unverified) computer algebra system?
See JHD's paper at CICM 2023 [Dav23], but note that the same question, in different settings, was asked by Mehlhorn [Meh11].
[Dav23] looked at three examples.

Polynomial Factorisation $f = f_1 f_2 \cdots f_k$ *and* the f_i is irreducible.

Integration The assertion "unintegrable" is correct.

Real Algebraic Geometry The assertion that the semi-algebraic variety is empty (UNSAT) is correct.

The last is the most important question, but factorisation is easy to explain and a good case study in its own right.

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 $\mathbf{Z}[x]$.

Verifying that $f = \prod f_i$ is, at least relatively, easy. The hard part is verifying that the f_i are *irreducible*. JHD knows of no implementation of polynomial factorisation that produces any evidence, let alone a proof, of this.
We may as well assume f is square-free (this would be a rather separate verification question).

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 $\prod f_i^{(1)} \pmod{p}$.

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 $p^n \geq 2B$.
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 5 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

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.

Progress at Dagstuhl

1. We can extract from the implementation in FLINT [tea23] of the algorithm with [H], at essentially no cost, the key data that we believe a verifier would need to confirm the irreducibility.
2. But this is not necessarily the most efficient verification.
3. We *think* that a more efficient verification would need negligibly more work.
4. We haven't built a verification.
5. The "hard" theorems are stated, but what about the "easy" ones, mappings such as "regarded as polynomials over $\mathbf{Z}$ with coefficients in $[-B, B]$"?
6. Needs more theorem prover input.

But We have discovered improvements to FLINT, and at least one new research question in computer algebra.

\+ FLINT also has [H'], but we haven't looked at this yet.

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.

James Harold 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.

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.

Bibliography II

K. Mehlhorn.
Certifying Algorithms.
https://people.mpi-inf.mpg.de/~mehlhorn/ftp/CertifyingAlgs.pdf, 2011.

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

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

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

Bibliography III

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