First steps towards Computational Polynomials in Lean

James Davenport
masjhd@bath.ac.uk

University of Bath
Thanks to 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/2408.04564

Background

The proof assistant Lean has support for abstract polynomials, but this is not necessarily the same as support for computations with polynomials.

Lean is also a functional programming language, so it should be possible to implement computational polynomials in Lean.

It turns out not to be as easy as the naive author thought.

We consider polynomials in commuting variables over a commutative ring R.

Existing Support in Lean

- **Univariate.** See `https://leanprover-community.github.io/mathlib4_docs/Mathlib/Algebra/Polynomial/Basic.html`.
- **Multivariate.** See `https://leanprover-community.github.io/mathlib4_docs/Mathlib/Algebra/MvPolynomial/Basic.html`.

  This "creates the type `MvPolynomial` σ R, which mathematicians might denote $R[X_i : i \in \sigma]$".

Note That σ might be infinite, whereas in the rest of the document, by "multivariate" we mean "in a fixed set of variables".

Lean allows the `Ring` with one element, so that $1 = 0$. Currently, we accept this.

These are abstract mathematical objects, not data structures as such.

Dense or Sparse

This is a fundamental decision in computer algebra: $x^2 + 1$ or $x^2 + 0x + 1$?

Like most general-purpose computer algebra systems, we have opted for sparse, essentially representing $\sum a_i x^i$ as a list of pairs (i, a_i).

In this representation, adding a polynomial of m terms to one with n terms requires $\leq m + n - 1$ comparisons of exponents, which is obvious and apparently non-controversial.

However, if f, g, h have l, m, n terms respectively

$(f + g) + h$ needs $\leq 2l + 2m + n - 2$ such comparisons

$f + (g + h)$ needs $\leq l + 2m + 2n - 2$ such comparisons

Especially in Gröbner bases, one is often adding small polynomials repeatedly to large ones.

Geobuckets [Yan98]

- A polynomial is stored as an (unevaluated) sum of polynomials, with the kth polynomial having at most c^k terms (typically $c = 4$) — hence *geo*metrically increasing *buckets*.
- If we add a regular polynomial with ℓ terms to a geobucket, we add it to bucket k with $c^{k-1} < \ell \leq c^k$, and if the result has more than c^k terms, we add that to bucket $k+1$, and cascade the overflow as necessary.
- In the absence of cascading overflows, the cost of adding ℓ terms is $O(\ell)$, irrespective of the size of the whole polynomial.
- The *amortised* cost of the cascading adds is small.

We don't currently intend to implement these, but it's worth remembering them.

Multivariate Polynomials I

Mathematically, $R[x, y, z]$ is the same structure as $R[x][y][z]$. When it comes to computer representations, this leads to a major choice.

- **Distributed.** This is $R[x, y, z]$, and is the representation of choice for Gröbner base algorithms. We will normally fix in advance our set of variables, and a total order $\prec$ on the monomials, which tells us whether $x^\alpha y^\beta z^\gamma \prec x^{\alpha'} y^{\beta'} z^{\gamma'}$. It is necessary in Gröbner base theory, and helpful in implementation, to assume that $\prec$ is compatible with multiplication: $\mathbf{x^i} \prec \mathbf{x^j} \Rightarrow \mathbf{x^{i+k}} \prec \mathbf{x^{j+k}}$.
  If we have k variables, then the obvious technique is to store the term $cx^\alpha y^\beta z^\gamma$ as $(\alpha, \beta, \gamma, c)$ (or possibly a record structure.

  However, since we often use total degree orders in Gröbner base computation, the Axiom implementation[1] actually stored $(\alpha + \beta + \gamma, \alpha, \beta, \gamma, c)$, i.e. the total degree first.

[1] This is now also done in Maple: [MP14].

Multivariate Polynomials II

- **Recursive.** A typical representation for, say, $x^3 - 2x$ would be $(x,(3,1),(1,-2))$, i.e. a list starting with the variable, then ordered pairs as in "Sparse". In a typed language we might have a record type `[variable,list]`. Hence $z^2(y^2+2)+(3y+4) \in R[y][z]$ would be represented as

$$(z,(2,(y,(2,1),(0,2))),(0,(y,(1,3),(0,4)))). \qquad (1)$$

What about $z^2(y^2+2)+(3x+4) \in R[x][y][z]$? There are (at least) two options.

- **Dense in variables.** In this option it would be represented as

$$\begin{gathered}(z,(2,(y,(2,(x,(0,1))),(0,(x,(0,2))))),\\(0,(y,(0,(x,(1,3),(0,4))))))).\end{gathered} \qquad (2)$$

- **Sparse in variables.** In this option it would be represented as

$$(z,(2,(y,(2,1),(0,2))),(0,(x,(1,3),(0,4)))). \qquad (3)$$

Multivariate Polynomials III

So "Sparse in variables" would seem easier, but the snag is that we can meet two polynomials with different main variables, and we need some way of deciding which is the 'main' variable, else we can end up with polynomials in y whose coefficients are polynomials in x whose coefficients are polynomials in y, which is not well-formed.

- **Other.** There are other options, with interesting complexity-theoretic implications, but not used in mainstream computer algebra: see [Dav22, §2.1.5].

Experience in Axiom, as in [DGT91], shows that it may be useful to be able to talk about univariate polynomials in an unspecified variable, i.e. just a list of (exponent,coefficient) pairs with no variable specified.

Recursive is suited to algorithms such as g.c.d., factorisation, integration etc., in fact almost everything except Gröbner bases. Reduce, which is recursive, has a special distributed form for Gröbner bases [GM88] .

Implementation (Distributed)

`nvars` is the number of variables and `MvDegrees` is the `nvars`-tuple of degrees in these variables, with an ordering.

```
def addCore : List (MvDegrees nvars × R) → List (MvDegrees
                            → List (MvDegrees nvars × R)
  | [], yy => yy
  | xx, [] => xx
  | xx@((i, a) :: x), yy@((j, b) :: y) =>
    if i < j then
      (j, b) :: addCore xx y
    else if j < i then
      (i, a) :: addCore x yy
    else  -- check for a+b=0
      ( fun c => if c=0 then addCore x y
                            else (i, c) :: addCore x y) (a+b)
```

The notation `xx@((i, a) :: x)` means "call it `xx`, but also deconstruct it into `(i, a)` as the head, and `x` as the tail.
Termination of this recursive definition is obvious.

Implementation (Snag 1)

Lean requires a well-typed recursive definition to terminate.
Any programmer would say that termination is obvious, as every recursive call is on less (either less x or less y, or possibly both), but the Lean type-checker doesn't recognise this, as it says below

```
fail to show termination for
  MvSparsePoly.addCore
with errors
argument #5 was not used for structural recursion
  failed to eliminate recursive application
    addCore xx y

argument #6 was not used for structural recursion
  failed to eliminate recursive application
    addCore x yy

structural recursion cannot be used
```

Implementation (Answer to Snag 1)

[McK24]: "How weak of Lean, much easier in Agda" — but actually not, on experimentation.
In both, the proof of termination is required *in the type checker*, which isn't the full theorem prover.
Solution:

```
termination_by xx yy =>
    xx.length + yy.length
```

Note that this requires the `xx@` syntax to state it.

Implementation (Snag 2)

For this to work the polynomials must be sorted (with a well-ordering `WOrdering nvars`).

Figure: Sorted addition of multivariate polynomials

```
theorem addCore_sorted : ∀ {x y : List (MvDegrees nvars × R)},
    x.Sorted (·.1 > ·.1) → y.Sorted (·.1 > ·.1) →
    (addCore x y).Sorted (·.1 > ·.1) := by
```

Proof of this is still "work in progress"

Outsourcing Polynomial Algebra

In many cases, such as gcd (my other talk) or Gröbner bases/ideal membership [Buz24, Mac24], we can outsource the computation.

gcd $h = \gcd(f, g,)$: $\exists f' : f = f'h$; $\exists g' : g = g'h$;
$\exists \lambda, \mu : h = \lambda f + \mu g$

Ideal $f \in \langle f_1, \ldots, f_k \rangle$: $\exists \lambda_i : f = \sum \lambda_i f_i$

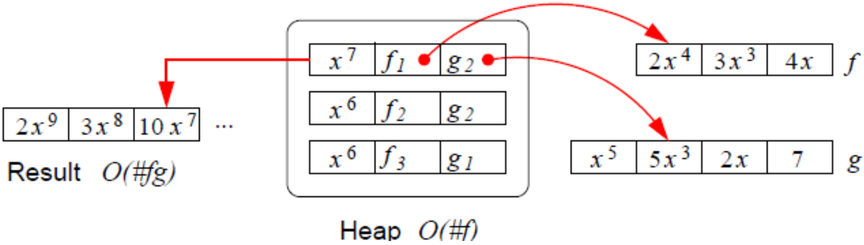
... Other applications.

Get an (untrusted) algebra system to compute the cofactors $\exists$ and the (trusted) prover just verifies the identity: $0 = \sum_{=1}^{N} \mu_i g_i$.
But (especially for ideal membership), the μ_i may be much larger than the g_i, and individual summands $\mu_i g_i$ larger still.

“Recall” Heap Multiplication [Joh74] (from [MP09])

Put the terms f_i of f into a heap by degree ($O(\#f)$ heapify), then regard this as sorted by degree of $f_i g$, and extract the terms of fg: after each extraction we update the heap ($O(\log \#f)$).

Fig. 1. Multiplication Using a Heap of Pointers (Johnson, 1974).



- Note that next term will have collision $f_2g_2 + f_3g_1$
- Total cost $O(\#f\#g \log \#f)$ so choose $\#f \leq \#g$.

Heap Verification Algorithm

- Build a [Joh74] heap for each $\mu_i g_i$ (using smaller of the two as f, assume g_i): cost $\sum \# g_i$.
- Build a heap of these, using $\deg(\mu_i g_i)$ as our criterion: cost N.
- Start extracting terms (which should all be 0 after we add all the contributions). Rebalance the outer heap and the relevant $\mu_i g_i$ heap.
- Cost $\log N \sum^N \#\mu_i \# g_i + \sum^N \#\mu_i \# g_i \log \# g_i$ comparisons/ coefficient operations.
- Additional space cost: that of the heaps: $N + \sum^N \# g_i$, irrespective of $\#\mu_i$.

How to handle geobuckets μ_i here

1) Flatten the geobuckets first: cost $O(\sum \#\mu_i)$ but potentially a lot of space.
2) If $\mu_i = \sum_j \mu_{i,j}$ have more [Joh74] heaps $g_i\mu_{i,j}$ rather than a single heap $g_i\mu_i$. Increases N to $N + \sum \log \#\mu_i$.
3) Handle μ_i being a Geobucket, which may incur more space consumption compared with moving down a pointer in a list-based g_i.

Topic for further reearch!

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

K. Buzzard.
We outsource the computation of witnesses to ideal membership.
Personal Communication at Hausdorff Institute 19 June, 2024.

J.H. Davenport.
Computer Algebra.
To be published by C.U.P.: http://staff.bath.ac.uk/masjhd/JHD-CA.pdf, 2022.

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.

Bibliography II

J.H. Davenport, P. Gianni, and B.M. Trager.
Scratchpad's View of Algebra II: A Categorical View of Factorization.
In S.M. Watt, editor, *Proceedings ISSAC 1991*, pages 32–38, 1991.

R. Gebauer and H.M. Möller.
On an installation of Buchberger's Algorithm.
J. Symbolic Comp., 6:275–286, 1988.

S.C. Johnson.
Sparse Polynomial Arithmetic.
In *Proceedings EUROSAM 74*, pages 63–71, 1974.

- Heather Macbeth.
  Algorithm and Abstraction in Formal Mathematics.
  In Kevin Buzzard, Alicia Dickenstein, Bettina Eick, Anton Leykin, and Yue Ren, editors, *Mathematical Software — ICMS 2024*, volume 14749 of *Springer Lecture Notes in Computer Science*, pages 12–28. Springer, 2024.
- J. McKinna.
  Termination in Agda.
  Personal Communication at Hausdorff Institute August, 2024.
- M.B. Monagan and R. Pearce.
  Parallel sparse polynomial multiplication using heaps.
  In *Proceedings ISSAC 2009*, pages 263–270, 2009.

Bibliography IV

M. Monagan and R. Pearce.
POLY : A new polynomial data structure for Maple 17.
In R. Feng, Ws. Lee, and Y. Sato, editors, *Proceedings Computer Mathematics ASCM2009 and ASCM2012*, pages 325–348, 2014.

T. Yan.
The geobucket data structure for polynomials.
J. Symbolic Comp., 25:285–294, 1998.