First steps towards Computational Polynomials in Lean

James H. Davenport

Departments of Computer Science and Mathematical Sciences
University of Bath
Bath BA2 7AY, United Kingdom
masjhd@bath.ac.uk 0000-0002-3982-7545

Abstract—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.

This is based on the author's experience in computer algebra, and [Dav22]. We consider polynomials in commuting variables over a commutative ring R (non-commutativity is considered in [Dav22, §2.4]). To avoid a plethora of "...", we will tend to use three variables x, y, z as our example, or x if we want a vector of variables.

I. STATE OF LEAN

A. Existing support

- 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".

B. Why both?

The reader might ask: "Isn't univariate just multivariate with σ being a singleton?" In one sense this is true, but not very helpful. If I have two univariate polynomials over a field $f:=x^a+\cdots$ and $g:=x^b+\cdots$, then I can divide one by the other (which way round depends on whether a>b) to get a remainder $h:=x^c+\cdots$ with $c<\min(a,b)$. In Axiom-speak, the exponents $\mathbf N$ are an OrderedCancellationAbelianMonoid, i.e. $a\geq b\to \infty$

The author is partially supported by EPSRC under grant EP/T015713/1. This research started at the Hausdorff Institute supported by Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) under Germany's Excellence Strategy – EXC-2047/1 – 390685813. Several participants at https://www.mathematics.uni-bonn.de/him/programs/current-trimester-program/him-trimester-program-prospects-of-formal-mathematics have given useful suggestions, and Mario Carneiro has been a constant source of advice and implementation. The author is grateful to the SYNASC referees, James McKinna and Ali Uncu for comments on drafts.

 $\exists c: a = b + c$. This is not true of multivariates: consider $f = x^1y^0$ and $g = x^0y^1$. If the polynomials are not over a field, but merely an integral domains, we can still do pseudodivision of univariates. Subject to the appropriate conditions on the ground field, this means that univariates over a field form a Euclidean domain, whereas multivariates do not.

C. A curious case

Lean rings include the ring with one element, so that 0=1. Hence polynomials over this ring are trivial, as x can't have a non-zero coefficient. This creates various special cases. For the moment we allow this case, though JHD does wonder whether it is introducing too much inefficiency. This will need to be checked, but fortunately Lean has good profiling tools [Ull24].

II. REPRESENTATIONS OF POLYNOMIALS

We have various decisions to make over the computational representation of polynomials.

A. A preliminary choice

Do we store zero terms?

Dense All terms in $f = \sum_{i=0}^{n} a_i x_i$ from i = 0 to i = n are stored, whether or not $a_i = 0$. Well-suited to a vector representation.

Sparse Only store the terms with $a_i \neq 0$, but need to store i with a_i to tell $x^2 + 2$ from $x^2 + 2x$.

General-purpose representation in computer algebra systems are always sparse, otherwise you look stupid if you can't handle $x^{10000000000} + 1$

B. Geobuckets

The obvious way to implement a sparse data structure is as a list of pairs (i,a_i) , sorted according to i and with duplicates combined. This is very close to the pen-and-paper representation. The cost of adding polynomials with m and n terms is m+n-1 comparisons. But note that the cost of addition is not associative: if p,q,r are polynomials with l,m,n terms respectively, and no combination of terms happens, then p+(q+r) takes l+2(m+n)-2 comparisons while (p+q)+r takes 2(l+m)+n-2 comparisons. This shows up in Gröbner basis computations, where we are typically reducing a long polynomial p by the generators, which tend to be much shorter. This asymmetry is addressed by the geobucket data

structure [Yan98], which has been adopted by many specialist systems [Abb15], [Sch15].

A polynomial is stored as an (unevaluated) sum of polynomials, with the kth polynomial having at most c^k terms (typically c=4) — hence geometrically increasing buckets. If we add a regular polynomial with ℓ terms to a geobucket, we add it to bucket k with $c^{k-1} < \ell \le c^k$, and if the result has more than c^k terms, we add that to bucket k+1, and cascade the overflow as necessary. A further advantage of the geobucket structure is given in §IV-A.

C. An improvement

[Yan98, Figure 2] suggests looking for the leading coefficient among the leading coefficients of the buckets. At least one of [Abb15], [Sch15] told the author that instead they ensured that the leading coefficient was always the leading coefficient of the largest bucket.

D. Related work

The geobucket structure can be seen as deferring the expensive addition of differently-sized objects until the last possible moment. In this sense it has similarities to the Hierarchical Representation of [ZCJM06], as improved in [SLB24]. The principal difference is that in the hierarchical representation the user decides when to "unveil" (i.e. use the internal structure) an object, whereas in geobuckets the system automatically unveils the appropriately sized bucket, and actively updates the buckets. Hence the two systems have different utilities.

III. CHOICES OF MULTIVARIATE POLYNOMIAL REPRESENTATIONS

Mathematically, R[x,y,z] is the same structure as R[x][y][z] (were it not for [Buz24a], the author would have written R[x,y,z]=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 ≺ on the monomials, which tells us whether x^αy^βz^γ ≺ x^{α'}y^{β'}z^{γ'}. It is necessary in Gröbner base theory, and helpful in implementation, to assume that ≺ is compatible with multiplication: xⁱ ≺ x^j ⇒ x^{i+k} ≺ 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 actually stored $(\alpha + \beta + \gamma, \alpha, \beta, \gamma, c)$, i.e. the total degree first.

• **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" above. 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

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

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

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

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

- 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. Most systems therefore use recursive, and for example, when implementing Gröbner bases in Reduce, which is recursive, [GM88] implemented their own distributed polynomials, and converted in/out.

A. Addition of Sparse polynomials

This should be simple: merge the lists except when both have the same exponent, when we add the coefficients (and handle the case where the sum is zero specially). This should translate into Lean like this (in fact for the multivariate case). The code is shown in Figure 1, where nvars is the number of variables and MvDegrees can be thought of as the nvarstuple of degrees in these variables.

Note that Lean requires a recursive definition to be proven 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 Lean doesn't recognise this, as it says below (where we have used "R" for "recursion").

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

²In practice it also stores the total degree, to make total degree order comparisons faster.

The notation $x \times \theta$ ((i, a) :: x) means "call it xx, but also deconstruct it into (i, a) as the head, and x as the tail.

```
argument #5 was not used for structural R
failed to eliminate recursive application
  addCore xx y
```

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

structural R cannot be used

[McK24] reports that an "obvious" translation into Agda has a similar issue with automatic proof of termination, and it needs a "non-obvious" translation to allow the automatic termination prover to conclude termination. Proofs of termination in Agda are discussed in [Agd24].

The author's Lean solution is

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

which requires the xx@ syntax to ensure that xx was defined. We need more than this, though. A true polynomial is defined in Figure 2 (where Wordering is the type for the well-ordering on monomials), and in particular we require the list of terms to be sorted. This requires a theorem as in Figure 3. We are currently working on a proof of this.

B. Polynomial Multiplication

Naive polynomial multiplication (but see §IV-B) is relatively simple to implement given addition. Similarly the proofs that this algorithm gives well-formed results should follow from those results for addition. We have yet to see what the challenge is for correctness.

IV. POLYNOMIAL VERIFICATION

[Buz24b] states that, asked to verify $f \in \langle f_1, \ldots, f_k \rangle$, Lean asked Sage (actually Singular) to compute the co-factors λ_i , and Lean just verifies that $f = \sum^N \lambda_i f_i$. This verification isn't really a Gröbner basis computation, and any polynomial representation would be appropriate. Which is best? Experiments will need to be performed, but an obvious guess would be the same representation as Sage outputs.

Let #f denote the number of terms in the polynomial f.

A. A specialist input mechanism?

Note that, at least in the worst case, the λ_i may be much larger than f or f_i . For example, in proving non-singularity of Weierstrass-form elliptic curves [Mac24, §3.2], the largest f_i is $-y^2 + 2pxz + 3qz^2i := f_3$, but the largest λ_i is

$$\begin{array}{l} (256/3)p^{12}x^2 + 128qp^{11}xz + 2304q^2p^9x^2 + 2592q^3p^8xz - \\ 64qp^{10}y^2 + 23328q^4p^6x^2 + 17496q^5p^5xz - 1296q^3p^7y^2 + \\ 104976q^6p^3x^2 + 39366q^7p^2xz - 8748q^5p^4y^2 + \\ 177147q^8x^2 - 19683q^7py^2 =: \lambda_1 \end{array}$$

with 13 terms, and in fact $f_1\lambda_1$ has 26 terms.

If the λ_i are output in decreasing order, then the naive algorithm "read a term; add it to the polynomial" using the method of §III-A, will take $O((\#\lambda_i)^2)$ operations. A geobucket implementation (§II-B) would make this $O(\#\lambda_i\log(\#\lambda_i))$. However, an "intelligent" algorithm that knew it was reading in a specific order would be simply $O(\#\lambda_i)$.

B. [Joh74], as described in [MP09]

This is an algorithm for sparse polynomial multiplication. Let f and g be polynomials stored in a sparse format that is sorted with respect to a monomial order \prec . We shall write the terms of f as $f=f_1+f_2+\cdots+f_{\# f}$ and g as $g=g_1+g_2+\cdots+g_{\# g}$. We require g to be stored as a list, so that $g-g_1$ (and then $g-g_1-g_2$ etc.) can be computed on O(1) time and 0 space (i.e. CDR in Lisp). Our task is to compute the product $h=f\times g=\sum_{i=1}^{\# f}\sum_{j=1}^{\# g}f_ig_j$ term-by-term, starting with the greatest.

We use a (balanced) binary heap to merge each (unevaluated) $f_i \times g$, so the sort key is $\deg(f_i) + \deg(g)$, and the heap has #f entries. After an $f_i \times g_j$ is extracted, we insert $f_i \times (g_{j+1} + \cdots)$. If the leading monomial of the resulting heap is the same as the just-extracted $f_i \times g_j$, we add the coefficients and continue. Since the size of the heap is #f, the cost of rebalancing after a single extraction is $\log \#f$, the number of extractions is #f #g, and therefore the total cost is $\#f \#g \log \#f$.

We could therefore solve the problem from [Buz24b] by computing each $\lambda_i f_i$ this way, with λ_i being g_i as we might expect $\#\lambda_i > \#f_i$ at least in the worst case. This would still

```
@[ext] structure MvSparsePoly (R : Type) [CommRing R] (nvars : N) [WOrdering nvars] : Type where
terms : List (MvDegrees nvars x R)
sorted : terms.Sorted (·.1 > ·.1)
nonzero : ∀ x ∈ terms, x.2 ≠ ∅
```

Fig. 3. Sorted addition of multivariate polynomials

```
theorem addCore_sorted : \forall \{x \ y : List \ (MvDegrees \ nvars \times R)\},\
| x.Sorted (\cdot.1 > \cdot.1) \rightarrow y.Sorted \ (\cdot.1 > \cdot.1) \rightarrow (addCore x y).Sorted \ (\cdot.1 > \cdot.1) := by
```

requiring storing each $\lambda_i f_i$ (but if we computed $\sum \lambda_i f_i$ as we went along, we would store the partial sum and the single $\lambda_i f_i$ being added).

C. Verification without product construction

The challenge is to compute $\sum^n f_i \lambda_i$ efficiently. In particular $f_i \lambda_i$ may well be much larger than either f_i or λ_i As in [Joh74], represent each $\lambda_i f_i$ (unevaluated) by a binary heap H_i . Now create a binary heap H whose elements are the heaps H_i . Extract the leading coefficient of H (as above, this may be the sum of several such, and indeed, because we are expecting cancellation in $\sum^N \lambda_i f_i$, it may well be zero). H has N elements, so the overall cost of this operation is $\sum^N \# \lambda_i \# f_i(\log N + \log \# f_i)$. Note that there is no $\log \# \lambda_i$ term — this is because we are leveraging the fact that the λ_i are already sorted.

Write $f = \sum^N f_i \lambda_i$. Then we certainly need to store³ the λ_i . We need to store the f_i , which are stored in the heap structures H_i , taking space $O(\sum \# f_i)$ (probably with a higher constant of proportionality). There's also O(N) for the master heap H. There's also the cost of the output. Note that the output f is constructed term-by-term, and no term is deleted, so the space cost is $O(\# f + \sum_N \# f_i)$.

A further improvement, rather than verifying $f = \sum^N f_i \lambda_i$, is to verify $0 = (-1)f + \sum^N f_i \lambda_i$, so there is no output to construct. It would also be possible to run the heaps with minimal degree at the root (i.e. verifying the trailing coefficient first), which might be more advantageous in the case where there is actually an error, i.e. $0 \neq (-1)f + \sum^N f_i \lambda_i$.

D. Interaction with geobuckets

[Joh74] assumed that the g_i are stored as lists: what if they are stored as geobuckets? Then the process of replacing g by $g-g_1$ is no longer O(1), but depends on how the overall geobucket structure is stored: if it's a vector of b buckets, then the cost is $O(b) = O(\log \# g_i)$. There are (at least) three possible routes.

1) Do an initial conversion of g to list structure, adding the buckets from small to large, and then using [Joh74]. The cost is #g in time (and space!).

- 2) Instead of storing f_ig in the [Joh74] heap, store each $f_ig^{(j)}$ where the $g^{(j)}$ are the buckets of g. This means there are b times as many entries in the heap, so multiplies the cost of [Joh74] by $O(\log b) = O(\log \log \#g)$
- 3) In practice we would probably do a hybrid: add the small buckets together but store $f_i g^{(j)}$ separately for large $g^{(j)}$, where "large" is a tuning parameter.

REFERENCES

- [Abb15] J.A. Abbott. Geobuckets in CoCoA. Personal Communication at Dagstuhl, 2015.
- [Agd24] Agda authors. Termination checking. https://agda.readthedocs.io/en/ v2.7.0-rc3/language/with-abstraction.html#termination-checking, 2024
- [Buz24a] K. Buzzard. Grothendieck's use of equality. https://arxiv.org/abs/ 2405.10387, 2024.
- [Buz24b] K. Buzzard. We outsource the computation of witnesses to ideal membership. Personal Communication at Hausdorff Institute 19 June, 2024.
- [Dav22] J.H. Davenport. Computer Algebra. To be published by C.U.P.: http://staff.bath.ac.uk/masjhd/JHD-CA.pdf, 2022.
- [DGT91] 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.
- [GM88] R. Gebauer and H.M. Möller. On an installation of Buchberger's Algorithm. J. Symbolic Comp., 6:275–286, 1988.
- [HV24] C. Hofstadler and T. Verron. Short proofs of ideal membership. J. Symbolic Comp. Article 102325, 125, 2024.
- [Joh74] S.C. Johnson. Sparse Polynomial Arithmetic. In *Proceedings EUROSAM 74*, pages 63–71, 1974.
- [Mac24] H.R. Macbeth. Algebraic computations in Lean. https://hrmacbeth.github.io/computations_in_lean/, 2024.
- [McK24] J. McKinna. Termination in Agda. Personal Communication at Hausdorff Institute August, 2024.
- [MP09] M.B. Monagan and R. Pearce. Parallel sparse polynomial multiplication using heaps. *Proc. ISSAC 2009*, pages 263–270, 2009.
- [MP14] 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.
- [Sch15] H. Schönemann. Geobuckets in SINGULAR. Personal Communication at Dagstuhl, 2015.
- [SLB24] D. Stocco, M. Larcher, and E. Bertolazzi. LEM (large expression management). https://github.com/StoccoDavide/LEM, 2024.
- [Ull24] S. Ullrich. Profiling Tools in Lean. Presentation at https://www.mathematics.uni-bonn.de/him/programs/current-trimester-program/him-trimester-program-prospects-of-formal-mathematics, 2024
- [Yan98] T. Yan. The geobucket data structure for polynomials. J. Symbolic Comp., 25:285–294, 1998.
- [ZCJM06] W. Zhou, J. Carette, D.J. Jeffrey, and M.B. Monagan. Hierarchical Representations with Signatures for Large Expression Management. In *Proceedings AISC 2006 LNAI 4120*, pages 254–268, 2006.

 $^{^3}$ We should note that different choices of λ_i can have different storage requirements: see [HV24].