Polynomials in Lean

Notes by J.H.Davenport

19 June 2024

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 avod a plethora of "...", we will tend to use three variables x, y, z as our example, or $\mathbf{x}$ if we want a vector of variables.

1 Lean

1.1 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".

1.2 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 $f := x^a + \cdots$ and $g := x^b + \cdots$, then I can divide one by the other (which way round depends on $a > b$) to get $h := x^c + \cdots$ with $c < \min(a, b)$. In Axiom-speak, the exponents $\mathbf{N}$ are an `OrderedCancellationAbelianMonoid`, i.e. $a \geq b \rightarrow \exists c : a = b + c$. This is not true of multivariates: consider $f = x^1y^0$ and $g = x^0y^1$.

1.3 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. **To be checked.**

2 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 one looks stupid if you can't handle $x^{1000000000} + 1$

2.1 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. This 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 *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.

2.2 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.

3 Choices of Multivariate Polynomial Representations

Mathematically, $R[x, y, z]$ is the same structure as $R[x][y][z]$ (were it not for [Buz24a], I would have written $R[x, y, z] = R[x][y][z]$). When it comes to computer representations, this leads to a major choice.

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

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: $\mathbf{x^i} \prec \mathbf{x^j} \Rightarrow \mathbf{x^{i+k}} \prec \mathbf{xj+k}$.

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.

Recursive A typical representation for, say, x^3-2x would be $(x,(3,1),(0,-2))$, 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

$$(z,(2,(y,(2,1),(0,2))),(0,(y,(1,3),(0,4)))). \tag{1}$$

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

$$(z,(2,(y,(2,(x,(0,1))),(0,(x,(0,2))))),(0,(y,(0,(x,(1,3),(0,4)))))). \tag{2}$$

Sparse in variables In this option it would be represented as

$$(z,(2,(y,(2,1),(0,2))),(0,(x,(1,3),(0,4)))). \tag{3}$$

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.

A practical note 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.

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

3.1 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).

```
def addCore : List (MvDegrees nvars × R) → List (MvDegrees nvars × R)
                      → List (MvDegrees nvars × R)
  | [], y => y
  | x, [] => x
  | 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  -- Don't we want to worry about a+b=0
      (i, a + b) :: addCore x y
```

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. Any programmer would say that termination is obvious, as every recursive call is on less, but Lean doesn't recognise this.

```
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
```

The solution is

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

4 A new (to me) question

[Buz24b] states that, asked to verify $f \in \langle f_1, \ldots, f_k \rangle$, Lean asked Sage (presumably actually Singular) to compute the co-factors λ_i, and Lean just verifies that $f = \sum^N \lambda_i f_i$. This isn't really a Gröbner basis computation, and any polynomial representation would be appropriate. Which is best? Note that, at least in the worst case, the λ_i may be much larger than f or f_i.

4.1 [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 $g = g_1 + g_2 + \cdots + g_{\#g}$. Our task is to compute the product $h = f \times g = \sum_{i=1}^{\#f} \sum_{j=1}^{\#g} f_i g_j$.

We use a binary heap to merge each $f_i \times g$. After an $f_i \times g_j$ is extracted, we insert $f_i \times g_{j+1}$. If the leading monomial of the heap is the same as the just-extracted $f_i \times g_j$, we add the coefficients and continue. So 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 compute 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.

4.2 New(?) idea

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.

References

[Abb15] J.A. Abbott. Geobuckets in CoCoA. *Personal Communication*, 2015.

[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 19 June 2024*, 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.

[Joh74] S.C. Johnson. Sparse Polynomial Arithmetic. In *Proceedings EUROSAM 74*, pages 63–71, 1974.

[MP09] M.B. Monagan and R. Pearce. Parallel sparse polynomial multiplication using heaps. In *Proceedings 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*, pages 325–348, 2014.

[Sch15] H. Schönemann. Geobuckets in SINGULAR. *Personal Communication*, 2015.

[Yan98] T. Yan. The geobucket data structure for polynomials. *J. Symbolic Comp.*, 25:285–294, 1998.