Canonical representation of polynomials *

Graeme Taylor

January 15, 2005

Polynomials are a vital part of many branches of algebra, but in a mathematical context such as Galois theory their use is either abstracted to the extent that explicit calculation or description is unneccessary, or examples are simple enough to muddle through with a few rules of thumb and the ability (with practice) to 'spot' familiar patterns, common factors and so on.

Such heuristics translate poorly to a computing environment, however. Most programming languages are ill-equipped to handle polynomials as mathematical objects even if they could evaluate one with blinding speed. A wide range of computer algebra systems have emerged to meet this need, and their underlying approach to polynomials as a data structure varies. What is important, however, is that there is an underlying defintion- a rigorous framework for the manipulation of polynomials.

A common problem in mathematics is that you can say the same thing in many ways. Many 'think of a number' magic tricks work in this way- you are lead through a calculation that always gives the same answer, but the path is twisted enough for this to be non-obvious to a participant. The mathematical work around for this is the equivalence class- two objects are held to be equivalent if some rule, known as the equivalence relation, applies. Two objects drawn from the same class are fundamentally the same, whilst objects from different classes have some essential difference between them.

There may be many objects in a class, and then it is useful to have a label for that class- a representative. This can be considered a canonical representation of all the objects from its class. Should we wish to work with any objects, we re-write them in canonical form and manipulate those instead. Thus the choice of representative should be made to ensure calculations are as simple as possible- such as writing fractions in lowest terms before adding or multiplying them.

*First appeared on Everything2, at *http://www.everything2.com/index.pl?node_id=1693221*

Determining which equivalence class a polynomial falls into is fairly easy- assuming you write each power only once, equivalence of two polynomials holds iff their corresponding coefficients are equal. The challenge is deciding what the canonical representation should then be. For instance, would $(1+x)(1-x)$ be preferable to $1-x^2$? It turns out that we can be even more pedantic than that- should we instead use $(-1)x^2+1$? How about $1+0x-x^2$?

It depends, of course, on the use to which you wish to put the polynomial. The first rendition makes it easy to identify factors that may be cancelled for division, whereas the last would allow for very rapid addition of a pair of degree 2 polynomials by summing the relevant coefficients. Some canonical forms may be more desirable for their mathematical properties (or simply more elegant), but the conversion to them could introduce an unacceptable computational overhead. Algorithms for multiplication, greatest common divisors or so on might behave better given inputs of a certain shape, such as handling the highest order terms first. For instance, factoring a polynomial is generally more difficult than multiplying out the factors to check you've got it right. Yet a research group working in the late 80's found that the REDUCE computer algebra system could factorise $x^{1155}-1$ with about 2MB of memory- yet any attempt to remultiply the factors exhausted the memory of the system- a quirk of the way its approach to multiplication generated intermediate steps significantly more complicated than either input or eventual output. Today's machines have far more than 2MB of memory at their disposal, and run faster too- but intolerably slow or memory-hogging problems are easily devised.

Polynomials in a single variable

Here, the general approach is to expand out the polynomial into a sum of powers of the variable with a single appropriate coefficient for each variable. Convention is to start at the highest power and work down to the constant term. Within this fairly restrictive classification, however, there is still room for variation. The debate concerns the treatment of zero coefficients- working with either a dense or sparse representation.

Dense representation

Consider the polynomials $3x^2+7x-5 and x2+2$. In a dense representation, we list every coefficient, regardless of whether it is zero. Then the information needed is as follows- the variable we are working in, the degree n of the polynomial, and an array of coefficients (which will be of size $n+1$). For our examples, the data structure could look something like this:

```
(x,2,3,7,-5)
(x,2,1,0,2)
```

In reality, the numerical values would probably be pointers to data structures storing the coefficient, as CA systems often deal with large integers beyond the limits of a regular signed int.

For clarity however we'll just note that here and assume that all numerical values 'fit' in the data structure.

The benefits of this approach are obvious- addition and subtraction can be carried out by direct comparison of the arrays and multiplication is also fairly easy to implement. (Division can be perilous since it may introduce fractions which are challenging in their own right for computers). The downside is apparent with polynomials such as these, however: $x^{10} + 3$ would become `(x,10,1,0,0,0,0,0,0,0,0,0,3)`; so imagine the mess $x^{1000} + 3$ would cause.

Sparse Representation

The clue to resolving this should be apparent from the example itself- when writing $x^{1000} + 3$, I didn't bother with the terms of zero coefficient- $0x^{999}$, $0x^{998}$, $0x^{997}$... by their very absence they were assumed to be zero. This motivates the sparse representation of a polynomial, where individual terms are described as a power-coefficient pair and any omitted power is assumed to have coefficient 0. Thus for $x^{1000} + 3$, we'd want something along the lines of the following:

`(x,1000,1,0,3)`

This tells us that we have a polynomial in variable x, then dives straight into the description- 1000,1 gives the $1x^{1000}$ term, 0,3 gives us the $3x^0$ term, and everything else is a zero. You might expect multiplication, addition and so-on to be more involved here, and you'd be correct. In particular, things are emphatically worse when there are no zero coefficients- not only are your arithmetical algorithms less efficient (increasing time), your data structure is larger (increasing memory use).
Consider a tweak of our original example: $x^4 - x^3 + 3x^2 + 7x - 5$.
In dense representation, this is `(x,4,1,-1,3,7,-5)`.
In sparse representation, this is the considerably less elegant `(x,4,1,3,-1,2,3,1,7,0,-5)`

Nonetheless, this analysis rests on the polynomial being completely dense- and this is rarely the case. Hence the convention for CA systems is to work in a sparse representation. Then it is the number of terms in a polynomial, rather than its degree, which influences the time taken to perform calculations on it- meaning that high degree problems can be deftly dealt with. To drive this point home, consider the following example.

By the difference of two squares, $(x^{1000} + 1)(x^{1000} - 1) = x^{2000} - 1$. To verify this via a dense representation, a million multiplications are needed. For a sparse representation, just four are required. Hence although the sparse multiplication is more work, the increase in the number of calculations required for a dense calculation is so massive that you're easily better off with the sparse representation.

Polynomials in more than one variable

In case the single variable case hasn't completely sold you on sparse representation, consider the polynomial in four variables $w^6x^6y^6z^6 - 1$. Then each variable in the dense representation can have degree from 0 to 6. With 4 variables, this leaves us trying to describe $4^7 = 16384$ coefficients, as opposed to the two we want for a sparse version. Clearly this more than offsets the overhead of describing to which monomial (product of powers of variables) the coefficient corresponds.

Instead, the concern with multivariate examples is not with zero coefficients for powers, but in which order to describe the powers. Restricting our interest to three variables x, y, z, is x^2y more important than xyz or y^2? To resolve this, we need a concept of ordering.

Purely lexicographical ordering

Here, we are concerned with a dictionary-like approach where monomials are ranked based on their relative positions alphabetically. Then an extra appearance of an x outweighs any other number of y's and z's. For instance, given x^3y^2z, we'd think of this as `xxxyyz`, which appears before `xxyyyyzzz` in a dictionary and so appears before $x^2y^4z^3$. This is appropriate in cases where the earlier variables are somehow dominant- the presence of the extra x means more than the factor y^2z^2.

The system is not quite like a dictionary- `xx` outweighs `x` so that we can preserve the decreasing order of the powers. But if you consider x as $x\_$, where `_` is a bonus letter added to the very end of your alphabet, then it behaves as expected.

Given this alphabetical dominance, we can effectively write a multivariate polynomial as a polynomial purely in the first variable, whose coefficients are themselves polynomials. This gives rise to the

Sparse recursive representation

Suppose we have

$$x^2y + x^2 + xy^2 + x + y^3 - 1.$$

Then this purely lexicographical ordering (with $x > y$) can be re-written as

$$x^2(y + 1) + x(y^2 + 1) + (y^3 - 1)$$

Which itself can be recursively described as

```
(x,2,(y,1,1,0,1),1,(y,2,1,0,1),0,(y,3,1,0,-1))
```

Where instead of pointers to numerical coefficients for x, we point to other polynomials, this time in y.

Total degree ordering

Given a monomial $x^a y^b z^c$, we can define its total degree as $a + b + c$. Then the monomials are arranged in order of decreasing total degree. Where two expressions match in total degree (e.g. x^2yz and xy^2z), a tie-breaker is used. This may be lexicographical order as above, but for certain practical reasons reverse/inverse lexicographical is often preferred (see later). Weighting systems may also be employed to vary the calculation of total degree- for instance, $a + 2b + c$ would favour terms with y's in them.

Regardless of the exact details, the end result is a list of the monomials in accordance with some consistent rules. Unlike the pure lex case, this may not readily factorise in a way that allows for nesting of the polynomials, so the sparse recursive representation cannot be used. Instead, the distributed form is used.

Distributed form

This is still a sparse representation, for the reasons discussed. Rather than specifying pairs of powers and coefficients, we'll need an n-tuple that indicates the coefficient and the powers of each variable. With our example

$$x^2y + x^2 + xy^2 + x + y^3 - 1,$$

it looks like this:

```
(x,y,2,1,1,2,0,1,1,2,1,1,0,1,0,3,1,0,0,-1)
```

The header information states that we have two variables, and that we give the power of x then of y. So 2,1,1 is interpreted as power 2 for x, power 1 for y, and a coefficient of 1: hence x^2y; whilst 0,3,1 gives $x^0y^31 = y^3$ etc.

The total degree reverse lexicographical ordering

Whilst arguments for each ordering system can be made dependant on use (e.g. pure lex is highly helpful when working on Grbner Base problems), total degree reverse lexicographical is a common choice for representation. It's helpful to note therefore that this isn't simply an inversion of the order of the variables, but does something slightly more subtle. Consider $(x + y + z)^3$. In tdeg-lex with $x > y > z$ we obtain

$$x^3 + 3x^2y + 3x^2z + 3xy^2 + 6xyz + 3xz^2 + y^3 + 3y^2z + 3yz^2 + z^3$$

Yet, in tdeg-rlex we have

$$x^3 + 3x^2y + 3xy^2 + y^3 + 3x^2z + 6xyz + 3y^2z + 3xz^2 + 3yz^2 + z^3$$

All the same terms are appearing (obviously), but in the second formulation we don't introduce the third variable, z, until all the terms featuring just x and y have been listed. For some algorithms, this turns out to be a more efficient way of handling things.

A note on Maple

One of the best known computer algebra systems is Maple, and it has a slight quirk in the way it handles ordering. Whilst most CAs will assume a lexicographical ordering, a Maple worksheet bases its ordering on whatever it happens to see first. The logic is presumably that if you typed in $y + x$ instead of $x + y$, then you did so for a reason and subsequent orderings should value y over x. This means that you can set things up how you want them more easily, but also means that calculations cut and pasted from one sheet may end up in a different canonical form. If you don't spot that the statements obtained are equivalent, this can cause much frustration as you try to fix something that isn't actually broken!