

## Maple Lecture 14. Normalizing, Collecting, and Sorting

Finding simpler ways to write expressions is an essential task of computer algebra and falls within the general category of “term rewriting”, an important discipline in computer science. A short overview can be found in [1]. In this lecture we restrict ourselves to studying the tools Maple offers to normalize polynomials, following the approach taken in [2, Sections 7.3 to 7.6]. We end with a probability one test for equality and some notes on the normalization of expressions with approximate coefficients.

### 14.1 Canonical and normal form

An important problem in symbolic computation is the test for equality:

```
[> e1 := x*(1+y); e2 := x + x*y;
```

To test for equality, we need to use the **evalb** (eval boolean) command:

```
[> e1 = e2; evalb(e1=e2);
```

While the expressions are mathematically equivalent, for Maple they are not the same.

The canonical form for an expression is *the unique* representation which reduces the test on equality to

1. bringing the expression to its canonical form;
2. testing whether canonical forms are the same.

For univariate polynomials, *the* canonical form is the expanded polynomial, after removing of superfluous terms (e.g.:  $x - x = 0$ ), and reordering in descending order.

Normalization is the name of the process of bringing a mathematical expression into *a* (not *the*) normal form. For rational expressions we have seen that normal removes all common factors and expands numerator and denominator. Depending on whether we expand or factor numerator and denominator, we have four possible choices of a normal form.

For multivariate polynomials, there are many possible ways to order the variables and monomials – so we no longer uniqueness. In the next section we will see some tools to reduce multivariate polynomials to various normal forms.

### 14.2 Collection and Sorting

With the command **collect**, we view a polynomial in several variables as a polynomial in one (leading) variable:

```
[> e1c1 := collect(e1,x); e2c2 := collect(e2,y);  
[> evalb(e1c1=e2c2);
```

The above instructions show that we have to settle for a convention. In this case we have to take the same leading variable:

```
[> e2c1 := collect(e2,x);  
[> evalb(e1c1=e2c1);
```

The expressions above,  $e_1$  and  $e_2$ , are toys to illustrate the problem. In reality, the expressions may be huge and require a lot of resources to manipulate.

The following commands illustrate the command **collect** on a more general multivariate polynomial

```
[> p := sum(sum(sum('i+j+k)*x^i*y^j*z^k','i'=0..2),'j'=0..2),'k'=0..2);
```

We can view  $p$  as a polynomial in  $x$ :

```
[> collect(p,x);
```

We can view  $p$  as a polynomial in  $z$  with coefficients polynomials in  $y$ , whose coefficients are polynomials in  $x$ :

```
[> rp := collect(p,[z,y,x],'recursive');
```

A more involved normal form for multivariate polynomials is the recursive Horner form. Observe however that the conversion to the Horner form takes the default order of variables:

```
[> convert(rp,'horner');
[> convert(p,'horner');
```

To change the order of variables :

```
[> convert(p,'horner',[z,y,x]);
```

This illustrates that there is no such thing as “**em the**” Horner form for multivariate polynomials.

To order the monomials in a fully expanded polynomial, we must first decide on the order of the variables, e.g.  $x > y > z$ . Then we can order the monomials along the degrees of the monomials (tdeg) or as in a dictionary, using the pure lexicographic order (plex).

This extends to more general expressions:

```
[> gp := subs(x=sin(u),y=cos(v),z=tan(w),p);
[> collect(gp,sin(u));
[> sort(gp,[sin(u),cos(v),tan(w)],'plex');
```

### 14.3 A Numerical Probability One Test on Equality

When even rewriting expressions into a normal form is too expensive or simply impossible, then we can resort to a numerical probability one test, provided we have a way to evaluate the expressions.

The test on equality between two expressions  $e1$  and  $e2$  is performed as follows:

1. pick a random number:  $r$ ;
2. evaluate  $e1$  and  $e2$  in  $r$ ;
3. compare  $e1(r)$  with  $e2(r)$ .

Unless we are really unlucky in our choice of  $r$ , the test will be reliable – whence we call it a probability one test. To increase our confidence (we could have picked  $r$  from a common factor), we can repeat the test with other random numbers.

For the two expressions  $e1$  and  $e2$  defined above, we perform the random test as follows:

```
[> r := [rand(),rand()];           # we have two variables x and y
[> e1r := subs(x=r[1],y=r[2],e1);  # evaluate e1 in r
[> e2r := subs(x=r[1],y=r[2],e2);  # evaluate e2 in r
[> evalb(e1r=e2r);                 # check for equality
```

For numerical expressions with approximate coefficients, we may have to work with a tolerance on the errors to decide whether a number is zero or not. See the description on **fnormal** below.

Maple has the command **testeq** – used as `testeq(a=b)` – which is described as a “random polynomial-time equivalence tester”. The “random” means that the test depends on random numbers and that there is a probability that the test may give the wrong answer. The advantage of the test is that it runs in “polynomial-time”, which is normally much faster than using symbolic normalization.

In numerical analysis, a small tolerance  $\epsilon$  is often used to test on zero (for example, to stop a converging iterative process), i.e.: a number  $x$  for which  $|x| \leq \epsilon$  is considered as zero. In an expression with floating-point constants, we may want to discard all numbers smaller than some  $\epsilon$ . The command **fnormal** implements such a floating-point normalization.

## 14.4 Assignments

1. Consider the polynomial  $p = (x^2 + xy + x + y)(x + y)$ .
  - (a) Write the polynomial as a polynomial in  $x$  with coefficients in  $y$ .
  - (b) Order the monomials in  $p$  in total degree using  $x > y$ .
  - (c) Order the monomials in  $p$  in pure lexicographic order, using  $x > y$ .

2. The command **q := subs(x=p,x=p,x=p,p)**; on  $p = x^2 + 3$  produces

$$q := \left( \left( (x^2 + 3)^2 + 3 \right)^2 + 3 \right) + 3.$$

- (a) Transform  $q$  into its fully expanded form, with the terms sorted in descending order.
  - (b) Transform the fully expanded form of  $q$  into the original nested form obtained with `subs`.
3. Verify (numerically) the following equality:  $\ln(\tan(\frac{1}{2}x + \frac{1}{4}\pi)) = \operatorname{arcsinh}(\tan(x))$ .
4. Consider the expressions  $x + 1$  and  $\frac{x^2-1}{x-1}$ .
  - (a) Are these expressions symbolically the same?  
Give the Maple command(s) to illustrate your answer.
  - (b) Verify the equality numerically. Show how a numerical test can go wrong.
5. Do **f := factor(p,complex)**; on  $p = x^8 + 3x^5 - 2x + 9$ . Expand the numerical factorization **f** and execute **fnormal**, adjusting the parameter `digits` of `fnormal` till the expanded **f** looks almost as  $p$ .  
Execute **fnormal** on the difference between the expanded **f** and  $p$ .

## References

- [1] R. Bündgen. Term rewriting systems. In J. Grabmeier, E. Kaltofen, and V. Weispfennig, editors, *Computer Algebra Handbook. Foundations, Applications, Systems*, pages 132–137. Springer-Verlag, 2003.
- [2] A. Heck. *Introduction to Maple*. Springer-Verlag, third edition, 2003.