POLYDIV: Enhanced Polynomial Division

Francis J. Wright
School of Mathematical Sciences
Queen Mary and Westfield College
University of London
Mile End Road, London E1 4NS, UK.
Email: `F.J.Wright@QMW.ac.uk`

6 November 1995

Abstract

This package provides better access to the standard internal polynomial division facilities of REDUCE and implements polynomial pseudo-division. It provides optional local control over the main variable used for division.

1 Introduction

The `polydiv` package provides several enhancements to the standard REDUCE algebraic-mode facilities for Euclidean division of polynomials. The source file (`polydiv.red`) should be compiled (using `faslout`) and loaded when required by

```
load_package polydiv;
```

The numerical coefficient domain is always that specified globally. Further examples are provided in the test and demonstration file `polydiv.tst`.

2 Polynomial Division

The `polydiv` package provides the infix operators `div` and `mod` (as used in Pascal) for the Euclidean quotient and remainder, e.g.

```
(x^2 + y^2) div (x - y);

         x + y

(x^2 + y^2) mod (x - y);

            2
         2*y
```

(They can also be used as prefix operators.)

It provides a Euclidean division operator `divide` that returns both the quotient and the remainder together as the first and second elements of a list, e.g.

```
divide(x^2 + y^2, x - y);


                 2
        {x + y,2*y }
```

(It can also be used as an infix operator.)

All Euclidean division operators (when used in prefix form, and including the standard `remainder` operator) accept an optional third argument, which specifies the main variable to be used during the division. The default is the leading kernel in the current global ordering. Specifying the main variable does not change the ordering of any other variables involved, nor does it change the global environment. For example

```
div(x^2 + y^2, x - y, y);


         - (x + y)

remainder(x^2 + y^2, x - y, y);


           2
        2*x

divide(x^2 + y^2, x - y, y);


                       2
        { - (x + y),2*x }
```

Specifying x as main variable gives the same behaviour as the default shown earlier, i.e.

```
divide(x^2 + y^2, x - y, x);


                 2
        {x + y,2*y }

remainder(x^2 + y^2, x - y, x);


           2
        2*y
```

3 Polynomial Pseudo-Division

The polynomial division discussed above is normally most useful for a univariate polynomial over a field, otherwise the division is likely to fail giving trivially a zero quotient and a remainder equal to the dividend. (A ring of univariate polynomials is a Euclidean domain only if the coefficient ring is a field.) For example, over the integers:

```
divide(x^2 + y^2, 2(x - y));

              2    2
        {0,x  + y }
```

The division of a polynomial $u(x)$ of degree m by a polynomial $v(x)$ of degree $n \leq m$ can be performed over any commutative ring with identity (such as the integers, or any polynomial ring) if the polynomial $u(x)$ is first multiplied by $\mathrm{lc}(v, x)^{m-n+1}$ (where lc denotes the leading coefficient). This is called *pseudo-division*. The `polydiv` package implements the polynomial pseudo-division operators `pseudo_divide`, `pseudo_quotient` (or `pseudo_div`) and `pseudo_remainder` as prefix operators (only). When multivariate polynomials are pseudo-divided it is important which variable is taken as the main variable, because the leading coefficient of the divisor is computed with respect to this variable. Therefore, if this is allowed to default and there is any ambiguity, i.e. the polynomials are multivariate or contain more than one kernel, the pseudo-division operators output a warning message to indicate which kernel has been selected as the main variable – it is the first kernel found in the internal forms of the dividend and divisor. (As usual, the warning can be turned off by making the switch setting "`off msg;`".) For example

```
pseudo_divide(x^2 + y^2, x - y);

        *** Main division variable selected is x

                   2
        {x + y,2*y }

pseudo_divide(x^2 + y^2, x - y, x);

                   2
        {x + y,2*y }

pseudo_divide(x^2 + y^2, x - y, y);

                         2
        { - (x + y),2*x }
```

If the leading coefficient of the divisor is a unit (invertible element) of the coefficient ring then division and pseudo-division should be identical, otherwise they are not, e.g.

```
divide(x^2 + y^2, 2(x - y));

              2    2
```

```
        {0,x  + y }

pseudo_divide(x^2 + y^2, 2(x - y));

        *** Main division variable selected is x

                      2
        {2*(x + y),8*y }
```

The pseudo-division gives essentially the same result as would division over the field of fractions of the coefficient ring (apart from the overall factors [contents] of the quotient and remainder), e.g.

```
on rational;

divide(x^2 + y^2, 2(x - y));

          1             2
        {---*(x + y),2*y }
          2

pseudo_divide(x^2 + y^2, 2(x - y));

        *** Main division variable selected is x

                      2
        {2*(x + y),8*y }
```

Polynomial division and pseudo-division can only be applied to what REDUCE regards as polynomials, i.e. rational expressions with denominator 1, e.g.

```
off rational;

pseudo_divide((x^2 + y^2)/2, x - y);

                 2    2
                x  + y
        ***** --------- invalid as polynomial
                   2
```

Pseudo-division is implemented in the `polydiv` package using an algorithm (D. E. Knuth 1981, *Seminumerical Algorithms*, Algorithm R, page 407) that does not perform any actual division at all (which proves that it applies over a ring). It is more efficient than the naive algorithm, and it also has the advantage that it works over coefficient domains in which REDUCE may not be able to perform in practice divisions that are possible mathematically. An example of this is coefficient domains involving algebraic numbers, such as the integers extended by $\sqrt{2}$, as illustrated in the file `polydiv.tst`.

The implementation attempts to be reasonably efficient, except that it always computes the quotient internally even when only the remainder is required (as does the standard remainder operator).