Data Structures and Algorithms for Polynomials

Sub-project leader: Michael Monagan
Contributors: Craig Pastro, Janez Ales, Allan Wittkopf, Jennifer de Kleine.

This work is supported by the MITACS NCE and NSERC of Canada.

Our first goal is to design and implement a new data structure which facilitates the efficient implemenation of fast modular methods for multivariate polynomial greatest common divisor (GCD), resultant, and factorization computations over the rationals, finite fields, and algebraic number fields, and secondly, to design some new and implement some existing modular algorithms which exploit this data structure. Efficient polynomial arithmetic routines are the most important facility in a computer algebra system such as Maple. Almost every application of a computer algebra system makes use of them. The data structure that we have chosen may be described as a ‘‘recursive dense’’ data structure. It supports an arbitrary number of field extensions over Q or GF(p). We show an example. First we input a polynomial in Maple then convert it into the new representation.

```
> restart; read recden;
> a := 7*y-x^3*y^2*z+4*y*z^2+3*x;
```

$$a := 7\,y - x^3\,y^2\,z + 4\,y\,z^2 + 3\,x$$

```
> A := convert(a,POLYNOMIAL,[x,y,z]);
```

$$A := (7 + 4\,z^2)\,y + 3\,x - x^3\,y^2\,z$$

This is how the new representation is displayed. Below is a printout for the actual representation. The first component [0, [x, y, z], [ ]] contains the ring information, namely, that we are in characteristic 0 (the rationals), the polynomial is in three variables x, y, z and there are no algebraic extensions, that is, Q[x, y, z]. The second component contains the coefficients of the polynomial. Think of the polynomial which is in the ring Q[x, y, z] as being a polynomial in x whose coefficients are polynomials in y whose coefficients are polynomials in z whose coefficients are rationals, that is, in Q[z][y][x]. At each level in the data structure we represent a polynomial by an array (or vector) of coefficients. The quantity [7,0,4] for example is a vector of three coefficients representing $7 + 4\,z^2$.

```
> lprint(A);
POLYNOMIAL([0, [x, y, z], []],[[0, [7, 0, 4]], [[3]], 0, [0, 0, [0, -1]]])
```

To represent the polynomial over a number field, say Q(z) where $z^3 = 2$, we input the polynomial as

```
> A := convert(a,POLYNOMIAL,[x,y,z],z^3-2);
```

$$A := ((7 + 4\,z^2)\,y + 3\,x - x^3\,y^2\,z) \bmod \mathrm{anglebracket}(z^3 - 2)$$

```
> lprint(A);
POLYNOMIAL([0, [x, y, z], [[-2, 0, 0, 1]]],[[0, [7, 0, 4]], [[3]], 0, [0, 0, [0, -1]]
])
```

where the algebraic extension $z^3 - 2$ is encoded as [-2, 0, 0, 1]. To compute over a finite field, say GF(4), we represent field elements as polynomials in GF(2)[z] modulo $z^2 + z + 1$ and input the polynomial as

```
> A := convert(a,POLYNOMIAL,[x,y,z],z^2+z+1,2);
```

$$A := (y + x + x^3\,y^2\,z) \bmod \mathrm{anglebracket}(z^2 + z + 1,\,2)$$

```
> lprint(A);
POLYNOMIAL([2, [x, y, z], [[1, 1, 1]]],[[0, [1]], [[1]], 0, [0, 0, [0, 1]]])
```

At the bottom of the data structure we are representing polynomials in one variable over the integers modulo p by a vector of machine integers. This is important for the overall efficiency of the algorithms which apply the modular method. For the special case of GF(2) as in this example, we will want to use a bit-vector representation.

We have implemented basic arithmetic operations addition, multiplication, division for polynomials over the integers, any finite field, or number field. Our first main contribution is a complete implementation of the dense modular method for multivariate polynomial GCD computation over all these fields. Two new algorithmic ideas are highlighted in the examples below. Jennifer de Kleine and Allan Wittkopf are presently working on a new design implementation for the well known sparse modular GCD algorithm of Zippel over these fields. We have an implementation for monic polynomials and are presently implementing a new idea for handling the non-monic case. Dr. Janez Ales has implemented univariate polynomial factorization. He is presently working on improvements to a new bivariate factorization algorithm of Gao in [3].

GCDs in GF(q)[X]

Consider the problem of computing the GCD of a(w, x, y) and b(w, x, y) in GF(q)[w,x,y]. The basic modular method evaluates the inputs at $y = \alpha$ for points α in GF(q), computes the GCD of a(w, x, α) and b(w, x, α) recursively (in one less variable) for sufficiently many α so that the GCD in GF(q)[w,x,y] can be obtained by interpolating y from the images in GF(q)[w, x]. But if q is small there will not be enough points in the field to do this. Instead of choosing points from GF(q^k) , an extension field, Kaltofen and Monagan in [1] propose the following method. Think of the polynomial as being in GF(q)[y][w, x]. Since GF(q)[y] is a Euclidean ring, we may choose primes $y - \alpha$ where α runs over all elements of the field, and then also monic primes of higher degree, of which there are infinitely many. We compute the GCD modulo those primes, and then apply the Chineese remainder theorem to reconstruct the GCD in y . In doing this we gain one order of magnitude in efficiency. In the example below, which is over GF(4), you can see four primes of degree one in y namely, $y - 0$, $y - 1$, $y - w$, $y - w + 1$, then three primes of quadratic degree in y being used. Thus the first four recursive GCD computations are done in GF(4)[x] and the next three are done in GF(4^2)[x] where the finite field GF(16) is being represented by a quadratic extension in w over GF(4).

```
> read PGCD;
  p := 2:
  X := [w,x,y]:
  g := expand(randpoly(X,degree=5,dense)) mod p:
  a := expand(randpoly(X,degree=2,dense)*g) mod p:
  b := expand(randpoly(X,degree=2,dense)*g) mod p:
  X := [x,y,w]:
  A := convert(a,POLYNOMIAL,X,w^2+w+1,p);
  B := convert(b,POLYNOMIAL,X,w^2+w+1,p);
  G :=  PGCD(A,B);
```

$$A := (w+1+w\,y^2+y^3+(w+1)\,y^5+(w+1)\,y^6+y^7+(1+(w+1)\,y^2+w\,y^3+y^4+w\,y^5)\,x + (w\,y^2+(w+1)\,y^3+w\,y^4)\,x^2+(w+1+w\,y+(w+1)\,y^2+y^3)\,x^3+(1+w\,y+w\,y^2+y^3)\,x^4 + (y^2+1+y)\,x^5+(1+y)\,x^6) \bmod \text{anglebracket}(w^2+w+1,\,2)$$

$$B := (w+w\,y^3+(w+1)\,y^4+w\,y^5+y^6+y^7+(w+(w+1)\,y+(w+1)\,y^3+(w+1)\,y^4+w\,y^5+y^6)\,x + (y+w\,y^2+w\,y^4)\,x^2+(w+1+y+y^3)\,x^3+(w+1+y+(w+1)\,y^2+y^3)\,x^4+w\,x^5+x^6+x^7) \bmod \text{anglebracket}(w^2+w+1,\,2)$$

```
PGCD: GCD in GF(2)[w][x, y] where w^2+w+1
... mod <y-0>
... mod <y-1>
... mod <y-w>
... mod <y-w+1>
... mod <w+y+y^2>, a field extension of GF(4)
... mod <w+1+y+y^2>, a field extension of GF(4)
... mod <1+w*y+y^2>, a field extension of GF(4)
```

$$G := (w+1+(w+1)\,y+w\,y^2+w\,y^4+y^5+(w+w\,y+w\,y^2+y^4)\,x+(w+w\,y+w\,y^2+y^3)\,x^2 + (1+(w+1)\,y+y^2)\,x^3+x^5) \bmod \text{anglebracket}(w^2+w+1,\,2)$$

This problem comes to us from Hans Dobbertin. It is an application from Coding Theory. It is a GCD problem in four variables over GF(2) of degree 9 in A and 8 in B,C and Z. The two polynomials P and Q are relatively prime. Maple 6 is unable to solve this problem. It was still going after 10 minutes.

```
> A,B,C,Z := 'A,B,C,Z':
  z := (A^2*B^2*Z^2+A^2*B^2+B^2*Z^2+Z^2+B^2)/Z:
  a := (B^2*C^2*A^2+B^2*C^2+C^2*A^2+A^2+C^2)/A:
```

```
b := (C^2*Z*B^2+C^2*Z+Z*B^2+B^2+Z)/B:
c := (Z*A*C^2+Z*A+A*C^2+C^2+A)/C:
P := Z^2*A^3*B^2*C^2*(z^2*b^2*c^2+(a+z+1)^2*(a+c^2+1)):
P := expand(P) mod 2: P := convert(P,POLYNOMIAL,[A,B,C,Z],2):
Q := diffrpoly(P,A);
st := time(): G := PGCD(P,Q); (time()-st)*seconds;
```

```
PGCD: Gcd in GF(2)[A, B, C, Z]
... mod <1+Z^2+Z^5>, a field extension of GF(2)
PGCD: GCD in GF(2)[Z][A, B, C] where 1+Z^2+Z^5
... mod <C-0>
PGCD: GCD in GF(2)[Z][A, B] where 1+Z^2+Z^5
... mod <B-Z>
... mod <B-1+Z>
... mod <B-Z^2>
... mod <B-1+Z^2>
... mod <B-Z+Z^2>
... mod <B-1+Z+Z^2>
... mod <B-Z^3>
... mod <B-1+Z^3>
... mod <B-Z+Z^3>
... mod <B-1+Z+Z^3>
... mod <C-Z>
PGCD: GCD in GF(2)[Z][A, B] where 1+Z^2+Z^5
... mod <B-Z>
```

$$G := 1 \bmod 2$$

$$1.834\ \mathit{seconds}$$

GCDs in Q(α)[X]

This is a GCD problem in Q($\sqrt{2}+\sqrt{3}$)[x,y] . We represent elements of the number field as polynomials in Q[α] / < $\alpha^4 - 10\,\alpha^2 + 1$ > where $\alpha^4 - 10\,\alpha^2 + 1$ is the minimal polynomial for $\sqrt{2}+\sqrt{3}$ over Q. In [2] what we propose to improve efficiency of the standard modular approach in the literature, by an order of magnitude, is to first, as is normally done, map the coefficients modulo a prime p . This means we are computing over the finite ring R = Zp[α]/<$\alpha^4 - 10\,\alpha^2 + 1$>. But also, choose the prime such that $\alpha^4 - 10\,\alpha^2 + 1$ factors into distinct linear factors modulo p . The probability that this occurs for a randomly chosen prime is one in four for this example. What you see below in the output for the first prime $p = 40009$ is that $w^4 - 10\,w^2 + 1 = (w+22552)\,(w+5434)\,(w+17457)\,(w+34575)$. Now, for each root, e.g. $w = -22552$, evaluate the inputs at the root. So the polynomial coefficients are first mapped into R then into Zp where the arithmetic will now be done directly in the hardware. This needs to be done for each root and then the GCD over R is recovered by interpolation. All this is repeated for enough primes so that the GCD over the number field can be reconstructed. In the output below you will see output from three routines, MGCD, AGCD, and PGCD. MGCD is choosing the primes, computing GCDs in R[x, y] using AGCD, and applying Chinese remainder theorem. Three primes are used. AGCD is computing the roots of $\alpha^4 - 10\,\alpha^2 + 1$ modulo the primes, and for each root, computes a GCD in Zp[x, y] using PGCD then interpolates the four results. PGCD is computing GCDs in Zp[x, y] using the modular method by evaluating the inputs at $y = \alpha$ for $\alpha = 0, 1, 2, 3, 4, 5$.

```
> restart; read recden; read MGCDnf; read AGCD; read PGCD;
> X := [x,y,w]:
  g := w*x^3+w^2*y^3+randpoly(X,degree=2,dense):
  a := expand(randpoly(X,degree=2,dense)*g):
  b := expand(randpoly(X,degree=2,dense)*g):
  A := convert(a,POLYNOMIAL,X,w^4-10*w^2+1):
  B := convert(b,POLYNOMIAL,X,w^4-10*w^2+1):
  G := MGCD(A,B);
MGCD: GCD in Q[w][x, y] where w^4-10*w^2+1 = 0
MGCD: I will split w^4-10*w^2+1
 MGCD: Prime  40009 !
```

```
  AGCD: w^4+39999*w^2+1 = (w+5434)*(w+22552)*(w+34575)*(w+17457) mod 40009
   AGCD: | w = 34575
    PGCD: Gcd in GF(40009)[x, y]
    ... mod <y-0>
    ... mod <y-1>
    ... mod <y-2>
    ... mod <y-3>
    ... mod <y-4>
    ... mod <y-5>
   AGCD: | w = 17457
    PGCD: Gcd in GF(40009)[x, y]
    ... mod <y-0>
    ... mod <y-1>
    ... mod <y-2>
    ... mod <y-3>
    ... mod <y-4>
    ... mod <y-5>
   AGCD: | w = 5434
    PGCD: Gcd in GF(40009)[x, y]
    ... mod <y-0>
    ... mod <y-1>
    ... mod <y-2>
    ... mod <y-3>
    ... mod <y-4>
    ... mod <y-5>
   AGCD: | w = 22552
    PGCD: Gcd in GF(40009)[x, y]
    ... mod <y-0>
    ... mod <y-1>
    ... mod <y-2>
    ... mod <y-3>
    ... mod <y-4>
    ... mod <y-5>
  AGCD: images done for w^4+39999*w^2+1 - interpolating ...
 MGCD: Prime 40013 ... 40031 !
  AGCD: w^4+40021*w^2+1 = (w+30712)*(w+3836)*(w+36195)*(w+9319) mod 40031
   AGCD: | w = 9319
    PGCD: Gcd in GF(40031)[x, y]
    ... mod <y-0>
    ... mod <y-1>
    ... mod <y-2>
    ... mod <y-3>
    ... mod <y-4>
    ... mod <y-5>
   AGCD: | w = 36195
    PGCD: Gcd in GF(40031)[x, y]
    ... mod <y-0>
    ... mod <y-1>
    ... mod <y-2>
    ... mod <y-3>
    ... mod <y-4>
    ... mod <y-5>
   AGCD: | w = 3836
    PGCD: Gcd in GF(40031)[x, y]
    ... mod <y-0>
    ... mod <y-1>
    ... mod <y-2>
    ... mod <y-3>
    ... mod <y-4>
    ... mod <y-5>
   AGCD: | w = 30712
    PGCD: Gcd in GF(40031)[x, y]
    ... mod <y-0>
    ... mod <y-1>
    ... mod <y-2>
    ... mod <y-3>
    ... mod <y-4>
    ... mod <y-5>
  AGCD: images done for w^4+40021*w^2+1 - interpolating ...
 MGCD: Prime 40037 ...40039 ...40063 ...40087 ...40093 ...40099 ...40111 ...40123 ...
40127 !
  AGCD: w^4+40117*w^2+1 = (w+9328)*(w+11361)*(w+30799)*(w+28766) mod 40127
   AGCD: | w = 30799
    PGCD: Gcd in GF(40127)[x, y]
    ... mod <y-0>
    ... mod <y-1>
    ... mod <y-2>
    ... mod <y-3>
    ... mod <y-4>
    ... mod <y-5>
```

```
    AGCD: | w = 28766
     PGCD: Gcd in GF(40127)[x, y]
     ... mod <y-0>
     ... mod <y-1>
     ... mod <y-2>
     ... mod <y-3>
     ... mod <y-4>
     ... mod <y-5>
    AGCD: | w = 9328
     PGCD: Gcd in GF(40127)[x, y]
     ... mod <y-0>
     ... mod <y-1>
     ... mod <y-2>
     ... mod <y-3>
     ... mod <y-4>
     ... mod <y-5>
    AGCD: | w = 11361
     PGCD: Gcd in GF(40127)[x, y]
     ... mod <y-0>
     ... mod <y-1>
     ... mod <y-2>
     ... mod <y-3>
     ... mod <y-4>
     ... mod <y-5>
  AGCD: images done for w^4+40117*w^2+1 - interpolating ...
MGCD: Trial Divisions
```

$$G := (-47 - 909\,w + 91\,w^3 + (-26 - 620\,w + 62\,w^3)\,y + (310\,w - 31\,w^3)\,y^2 + y^3\,w + (-12 - 180\,w + 18\,w^3 + (-500\,w + 50\,w^3)\,y)\,x + (-610\,w + 61\,w^3)\,x^2 + x^3) \bmod \text{anglebracket}(w^4 - 10\,w^2 + 1)$$

References

[1] E. Kaltofen, M.B. Monagan. On the Genericity of the Modular Polynomial GCD Algorithm. *Proceedings of ISSAC '99*, ACM Press, pp. 59--66, 1999.
[2] M. B. Monagan, A. Wittkopf. On the Design and Implementation of Brown's algorithm over the Integers and Number Fields, *Proceedings of ISSAC '2000*, ACM Press, pp 225-233, 2000.
[3] S. Gao. Factoring Multivariate Polynomials via Partial Differential Equations. Unpublished manuscript, August 7th, 2000.