Algorithms for Polynomial GCD Computation over Algebraic Function Fields

Mark van Hoeij*
Department of Mathematics
Florida State University
Tallahassee, FL 32306-4510, USA.

Michael Monagan†
Department of Mathematics
Simon Fraser University
Burnaby, B.C. Canada. V5A 1S6.

ABSTRACT

Let L be an algebraic function field in $k \geq 0$ parameters $t_1, \ldots, t_k$. Let f_1, f_2 be non-zero polynomials in $L[x]$. We give two algorithms for computing their gcd. The first, a modular GCD algorithm, is an extension of the modular GCD algorithm of Brown for $\mathbb{Z}[x_1, \ldots, x_n]$ and Encarnacion for $\mathbb{Q}(\alpha)[x]$ to function fields. The second, a fraction-free algorithm, is a modification of the Moreno Maza and Rioboo algorithm for computing gcds over triangular sets. The modification reduces coefficient growth in L to be linear. We give an empirical comparison of the two algorithms using implementations in Maple.

1. INTRODUCTION

Let $D = \mathbb{Z}[t_1, \ldots, t_k]$ and $F = \mathbb{Q}(t_1, \ldots, t_k)$. Let $m(z) \in F[z]$ be monic and irreducible of degree d and let $L = F[z]/\langle m(z) \rangle$. L is an algebraic function field of degree d in k parameters $t_1, \ldots, t_k$. In our examples, if $k = 1$ we use t without subscript to denote the parameter. Our problem is to compute the gcd of two non-zero polynomials in $L[x]$. We denote the input polynomials by f_1 and f_2 and their monic gcd by g.

Our first algorithm, presented in section 2, is a modular GCD algorithm. It uses rational number and rational function reconstruction to recover the coefficients of the gcd and trial division to prove the correctness of the result. Like Encarnacion's algorithm [3], our algorithm is *output sensitive,* that is, the number of primes and evaluation points it uses depends on the size of the gcd and not on bounds based on the size of the inputs. As in [5], our algorithm does not compute discriminants. We show also how to use our algorithm to compute gcd's in $L[x_1, \ldots, x_n]$ by moving the variables $x_2, \ldots, x_n$ into F.

*Supported by NSF grant 0098034.

†Supported by NSERC of Canada and the MITACS NCE of Canada.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.
ISSAC'04, July 4–7, 2004, Santander, Spain.
Copyright 2004 ACM 1-58113-827-X/04/0007 ...$5.00.

We may also compute g using the Euclidean algorithm. But, there is a linear growth in the degrees and heights of the coefficients in F in the Euclidean algorithm which causes it to bog down doing arithmetic in F. Arithmetic with fractions in F and hence gcd computation in D can be eliminated using a fraction-free approach. In [6] Moreno Maza and Rioboo show how to do this for univariate gcd computation modulo a triangular set for which our setting, $L[x]$, is a special case. In section 3 we demonstrate that their algorithm has a serious coefficient swell in D. We modify their algorithm to have linear growth by doing $O(\delta)$ gcd computations in D where $\delta = \max(\deg_x f_1, \deg_x f_2) - \deg_x g + 1$. In section 4 we compare Maple implementations for the two algorithms to demonstrate their effectiveness.

Both of our algorithms work with associates (scalar multiples) of f_1, f_2 and g. We make some definitions. A non-zero polynomial in $D[z, x]$ is said to be *primitive wrt* (z, x) if the gcd of its coefficients in D is 1. Let f be nonzero in $L[x]$. The *denominator* of f is the polynomial $\mathrm{den}(f) \in D$ of least (total) degree in $(t_1, \ldots, t_k)$ and with smallest integer content such that $\mathrm{den}(f)f$ is in $D[z, x]$. The *primitive associate* $\tilde{f}$ of f is the associate of $\mathrm{den}(f)f$ which is primitive in $D[z, x]$. These definitions for $\mathrm{den}(f)$ and $\tilde{f}$ are unique up to sign; we impose uniqueness by requiring them to have positive leading coefficient in a term ordering. The *monic associate* $\check{f}$ of f is defined as $\check{f} = \check{h}$ where $h = \mathrm{monic}(f)$. Here $\mathrm{monic}(f)$ is defined as $\mathrm{lc}_x(f)^{-1} f$ where $\mathrm{lc}_x(f) \in L$ is the *leading coefficient* of f wrt x. Notice that $\deg_z(\mathrm{lc}_x \tilde{f}) = 0$.

Examples: Let $f = 2tx - t/(1-t) \in \mathbb{Q}(t)[x]$. Then $\mathrm{den}(f) = t - 1$, $\mathrm{monic}(f) = x - 1/(2 - 2t)$ and $\tilde{f} = \check{f} = (2t - 2)x + 1$. Let $f = 6zx - 3t \in \mathbb{Q}(t)[z][x]$ where $z = \sqrt{t}$. Then $\tilde{f} = 2zx - t$, $\mathrm{monic}(f) = x - z/2$ and $\check{f} = 2x - z$.

To provide the reader with an overview of the two algorithms, we first work through an example. Let $z = \sqrt{t}$. Consider the input polynomials

$$f_1 = x^2 + \frac{-2\,t + 3}{3} zx + \frac{5}{t}x + \frac{5}{t}z + \frac{-2t^2}{3},$$
$$f_2 = zx^2 + \frac{5}{t}zx + \frac{-3 + 2t^2}{3}x + \frac{-2t}{3}z + \frac{5}{t}.$$

Since $f_1 = (x + z)(x - 2tz/3 + 5/t)$ and $f_2 = (zx + 1)(x - 2tz/3 + 5/t)$ the monic gcd g of f_1 and f_2 is the polynomial $x - 2tz/3 + 5/t$. Both algorithms output $\check{g} = 3tx - 2t^2 z + 15$.

On input of f_1 and f_2 they first compute

$$\check{f}_1 = 3tx^2 + (-2t^2+3t)zx + 15x + 15z - 2t^3,$$
$$\check{f}_2 = 3tzx^2 + 15zx + (-2t^3+3t)x - 2t^2z + 15.$$

The Fraction-Free Algorithm

Let $r_1, r_2, r_3, \ldots, r_n, r_{n+1} = 0$ be the remainder sequence for the Euclidean algorithm with input $r_1 := \check{f}_1, r_2 := \check{f}_2$. Our fraction-free algorithm computes the remainder sequence $\check{r}_1$, $\check{r}_2$, $\check{r}_3$, $\ldots$, $\check{r}_n = \check{g}$, $\check{r}_{n+1} = 0$ without introducing fractions in $\mathbb{Q}(t)$. We multiply $\check{f}_2$ by z, a quasi-inverse (see section 3) of $\mathrm{lc}_x \check{f}_2$, to eliminate z from the leading coefficient of $\check{f}_2$.

$$p_2 := z\check{f}_2 = 3t^2x^2 + (-2t^3+3t)zx + 15tx + 15z - 2t^3.$$

Since p_2 is primitive, $p_2 = \tilde{f}_2 = \tilde{r}_2$. Using pseudo-division to avoid fractions in $\mathbb{Q}(t)$, we compute the remainder of $\mu\check{f}_1$ divided by p_2 in x where $\mu = \mathrm{lc}_x p_2 = 3t^2$ is an element of D, all working modulo $m(z) = z^2 - t$. We obtain

$$p_3 = 3t(t-1)zx + 15(t-1)z - 2t^3(t-1).$$

To minimize coefficient growth in $\mathbb{Z}[t]$ we make p_3 primitive wrt (z, x); we compute and divide out by $\gcd(3t(t-1), 15(t-1), -2t^3(t-1)) = t-1$. We obtain

$$p_3 := p_3/(t-1) = 3tzx + 15z - 2t^3.$$

Now we compute $p_3 := \tilde{r}_3$ by first multiplying p_3 by z and then making the result primitive. We obtain

$$p_3 := zp_3/t = 3tx - 2t^2z + 15.$$

Now we divide p_2 by p_3 using pseudo-division modulo $m(z)$. The pseudo-remainder is 0, thus, $\check{g} = p_3$ and we are done.

The Modular GCD Algorithm

We compute the gcd of $\check{f}_1$ and $\check{f}_2$ modulo a sequence of primes. Unlike the fraction-free algorithm, we do not compute $\tilde{f}_2$, because inverting $\mathrm{lc}_x(f_2)$ may lead to a blowup. Suppose we start with the prime $p = 11$. We will obtain

$$g_p = tx + 3t^2z + 5 \bmod 11.$$

We apply Wang's rational reconstruction (see [7, 3]) to the coefficients of g_p modulo p. It fails so we compute the gcd modulo a second prime, $q = 13$. We will obtain

$$g_q = tx - 5t^2z + 5 \bmod 13.$$

Note that we normalized the leading coefficient (of tx) to be 1 for both images. We apply the Chinese remainder theorem to obtain $g_m \bmod m = pq$ such that $g_m \equiv g_p \bmod p$ and $g_m \equiv g_q \bmod q$.

$$g_m = tx + 47t^2z + 5 \bmod 143.$$

We apply rational reconstruction to the coefficients of g_m modulo $m = 143$. This time we succeed. We obtain

$$h = tx - \frac{2}{3}t^2z + 5.$$

Now we clear denominators; we set

$$h := 3h = 3tx - 2t^2z + 15$$

and test if $h|\check{f}_1$ and $h|\check{f}_2$. It does, therefore $h = \check{g}$ and we are done. It remains to describe how we compute g_p and g_q. We do so by computing the gcd of $\check{f}_1(t, x)$ and $\check{f}_2(t, x)$ modulo p at a sequence of values for t in $\mathbb{Z}_p$. The evaluation point $t = 0$ causes $\mathrm{lc}_x(\check{f}_2)$ to vanish, so it is *lc-bad* (defined in section 2.1). Our algorithm avoids such t. Now consider $t = 1$ (our algorithm chooses evaluation points at random, but for this example we will use $t = 1, 2, \ldots$). We run the Euclidean algorithm modulo $p = 11$ on

$$f_1(1, x) = 3x^2 + zx + 4x + 4z - 2, \quad \text{and}$$
$$f_2(1, x) = 3zx^2 + 4zx + x - 2z + 4.$$

modulo $m(1, z) = z^2 - 1$. This is a univariate computation over $\mathbb{Z}_p[z]/\langle m(1, z)\rangle$, a finite ring, so no coefficient growth can occur. It succeeds because it did not try to divide by zero divisor in $\mathbb{Z}_p[z]/\langle z^2 - 1\rangle$. The output is

$$g_1 = \gcd(f_1(1, x), f_2(1, x)) = x^2 + 4zx + 5x + 5z + 3$$

Now we apply rational function reconstruction to the coefficients w.r.t. (z, x). It succeeds with output $h = g_1$, i.e., with constant rational functions in t. We test if $h|f_1(t, x) \bmod 11$ and $h|f_2(t, x) \bmod 11$. It doesn't so we take a new evaluation point $t = 2$ and run the Euclidean algorithm over $\mathbb{Z}_p[z]/\langle m(2, z)\rangle$. We obtain

$$g_2 = \gcd(f_1(2, x), f_2(2, x)) \bmod 11 = x - 5z - 3$$

which has lower degree than g_1. This tells us (see section 2.1) that $t = 1$ is *unlucky* and thus g_1 cannot be an associate of $g(1, x) \bmod 11$. We discard g_1. Continuing, we apply rational function reconstruction to g_2 and obtain $h = g_2$. We test if $h|f_1(t, x) \bmod 11$ and $h|f_2(t, x) \bmod 11$. It doesn't so we take a new evaluation point $t = 3$ and compute

$$g_3 = \gcd(f_1(3, x), f_2(3, x)) = x - 2z - 2.$$

We apply the Chinese remainder theorem to obtain the gcd modulo $(t-2)(t-3)$. We obtain

$$c = x + (3t)z + (t-5).$$

We apply rational function reconstruction to the coefficients of c to reconstruct numerators of degree 1 in t and denominators of degree 0 in t. This succeeds with $h = c$. We test if $h|f_1(t, x) \bmod 11$ and $h|f_2(t, x) \bmod 11$. It doesn't so we take a new evaluation point $t = 4$ and compute

$$g_4 = \gcd(f_1(4, x), f_2(4, x)) = x + z + 4.$$

Apply the Chinese remainder theorem to c and g_4 to obtain

$$c = x + 3tz - 3t^2 + 5t - 1$$

the gcd modulo $(t-2)(t-3)(t-4)$. We apply rational function reconstruction to the coefficients of c to reconstruct numerators of degree 1 and denominators of degree 1 in t. It succeeds with output

$$h = x + \frac{3t}{1}z + \frac{5}{t}.$$

Now we clear the denominators in t. We obtain

$$h := th = tx + 3t^2z + 5$$

Since $h|f_1(t, x) \bmod 11$ and $h|f_2(t, x) \bmod 11$ we are done with $p = 11$. In this example all prime numbers used were good. However, we did encounter an lc-bad evaluation $t = 0$ and an unlucky evaluation $t = 1$. It is also possible to hit a zero divisor in the middle of the Euclidean algorithm but such a fail prime/evaluation did not occur in the example. In section 2.1 we study the different problem cases that can occur before describing the algorithm in section 2.2 which we can prove (using Theorem 1 and 2) is correct.

2. THE MODULAR GCD ALGORITHM

2.1 lc-bad, fail, unlucky, and good primes

Let $f_1, f_2 \in L[x]$ where $m \in F[z]$, $L = F[z]/\langle m\rangle$ and $F = \mathbb{Q}(t_1, \ldots, t_k)$. Let $g \in L[x]$ be their monic gcd. Our algorithm in section 2.2 replaces f_1, f_2, m by their primitive associates, so we can assume that $m = \check{m} \in D[z]$ and $f_i = \check{f}_i \in D[z, x] \pmod m$ where $D = \mathbb{Z}[t_1, \ldots, t_k]$. Let $l_m = \mathrm{lc}_z(m) \in D$, and $l_i = \mathrm{lc}_x(f_i) \in D[z]$. If I is a prime ideal in D, and if $l_m \in I$ or $l_1 \in \langle I, m\rangle$ or $l_2 \in \langle I, m\rangle$ then I is called an *lc-bad prime*. Such I will be avoided.

Let I be a maximal ideal in D. Denote $\overline{D} = D/I$ and $\overline{R} = D[z]/\langle I, m\rangle$. For $s \in D[z, x]$ define $\overline{s} := s \bmod \langle I, m\rangle$. Thus $\overline{s} \in \overline{R}[x]$. Note that $\overline{l}_m$, $\overline{l}_1$ and $\overline{l}_2$ are not zero because we avoid lc-bad I. The maximal ideals I used in our algorithm are of the form $I = \langle p, t_1 - \alpha_1, \ldots, t_k - \alpha_k\rangle$, with p a prime number and $\alpha_i \in \mathbb{Z}$. For such I we can identify $\overline{D}$ resp. $\overline{R}$ with $\mathbb{Z}_p$ resp. $\mathbb{Z}_p[z]/\langle m(\alpha_1, \ldots, \alpha_k, z)\rangle$.

Definition: A *monic gcd* of $\overline{f}_1, \overline{f}_2$ is a monic polynomial $g_I \in \overline{R}[x]$ for which $\langle \overline{f}_1, \overline{f}_2\rangle = \langle g_I\rangle$ as $\overline{R}[x]$-ideals.

In general, a monic gcd need not exist, but if it does then it is unique. The Euclidean algorithm applied to $\overline{f}_1, \overline{f}_2 \in \overline{R}[x]$ fails if it tries to invert a leading coefficient in the polynomial remainder sequence that happens to be a zero-divisor (for more details see section 2 in [5]). In this case I is called a *fail prime*. This can happen even if the monic gcd of $\overline{f_1}, \overline{f_2}$ exists. As an example, take $f_1 = f_2 + 1$, $f_2 = (z+1)x + t$, $m = z^2 + 7tz - 1$. Then $\langle 7, t - \alpha\rangle$ resp. $\langle p, t - 0\rangle$ is a fail prime for any integer α resp. prime number p.

We may assume (if not, interchange f_1, f_2) that the Euclidean algorithm applied to $\overline{f}_1, \overline{f}_2$ first inverts the leading coefficient of $\overline{f}_2$ in order to divide $\overline{f}_1$ by $\overline{f}_2$. So if I is not lc-bad nor fail then $\overline{l}_2$ is a unit in $\overline{R}$, which is important in the proof of the theorem below. Note that if the implementation (without first interchanging) always inverts $\overline{l}_2$ like in [5] then we may use the *asymmetric definition of lc-bad*: I is lc-bad when $\overline{l}_m = 0$ or $\overline{l}_2 = 0$.

Assume now that the Euclidean algorithm applied to $\overline{f_1}, \overline{f_2}$ does not fail. Then the monic gcd $g_I \in \overline{R}[x]$ exists and will be the output. Let $d_I := \deg_x g_I$ and $d_g := \deg_x g$.
If $d_I > d_g$ then I is called an *unlucky prime*.

If $\mathrm{den}(g) \notin I$ then $\overline{g} \in \overline{R}[x]$ is defined, $\overline{g} := \overline{\mathrm{den}(g)g}/\overline{\mathrm{den}(g)}$. If this $\overline{g}$ equals g_I then I is called a *good prime*.

THEOREM 1. *With the above notations, if I is not lc-bad nor fail, then it is either unlucky or good.*

We will prove a more general statement:

THEOREM 2. *Let D be an integral domain, F its field of fractions, let $m \in D[z]$ (not necessarily irreducible in $F[z]$). Let $L = F[z]/\langle m\rangle$. Let $f_1, f_2 \in D[z, x]$ represent two non-zero polynomials $f_1, f_2 \in L[x]$, and let $g \in L[x]$ be any monic common divisor of f_1, f_2 in $L[x]$. Let I be a maximal ideal in D that is not lc-bad, let $\overline{D} = D/I$ and $\overline{R} = D[z]/\langle I, m\rangle$, and assume that the result g_I of the Euclidean algorithm applied to $\overline{f}_1, \overline{f}_2 \in \overline{R}[x]$ is not "failed". Then $d_I \geq d_g$.
Furthermore, if $d_I = d_g$ then $\overline{g}$ is defined and $\overline{g} = g_I$, and the monic gcd of f_1, f_2 exists and equals g.*

Proof: If the Euclidean algorithm does not fail then the extended Euclidean algorithm will not fail either, and will produce polynomials $s_I, t_I \in \overline{R}[x]$ such that $s_I\overline{f}_1 + t_I\overline{f}_2 = g_I$ and

$$\deg_x s_I < d_2 - d_I, \quad \deg_x t_I < d_1 - d_I.$$

Here $d_1 = \deg_x f_1$, $d_2 = \deg_x f_2$, $d_I = \deg_x g_I$, and $d_g = \deg_x g$. Note that the degree-bound given for t_I is not optimal because $\deg_x \overline{f}_1$ could be smaller than d_1 if we use the asymmetric definition of lc-bad.

Let $D_I = \{c \in F \mid \mathrm{den}(c) \notin I\} = \{a/b \mid a, b \in D,\ b \notin I\}$. Then we can identify $D_I/\langle I\rangle$ with $\overline{D}$. An element $c \in D_I$ is a unit in D_I if and only if its image $\overline{c}$ in $\overline{D}$ is not zero. Denote $R_I = \{a/b \in L \mid a \in D[z] \bmod m,\ b \in D,\ b \notin I\}$ which we can identify with $D_I[z]/\langle m\rangle$ (note that l_m is a unit in D_I because I is not lc-bad, hence division with remainder by m works as usual in $D_I[z]$). Then we can identify $R_I/\langle I\rangle = D_I[z]/\langle I, m\rangle = \overline{R}$.

Consider the free R_I-modules: $H = \{h \in R_I[x] \mid \deg_x h < d_1 + d_2 - d_I\}$, $H_0 = \{h \in R_I[x] \mid \deg_x h < d_I\}$, $G = H/H_0$ and $S = \{(s, t) \mid s, t \in R_I[x],\ \deg_x s < d_2 - d_I,\ \deg_x t < d_1 - d_I\}$. Now define a map $\phi : S \to G$ given by $\phi(s, t) = sf_1 + tf_2 + H_0$. Likewise (just replace R_I by $\overline{R}$) define free $\overline{R}$-modules $\overline{H}$, $\overline{H}_0$, $\overline{S}$ and $\overline{G}$, as well as the map $\overline{\phi} : \overline{S} \to \overline{G}$ given by $\overline{\phi}(s, t) = s\overline{f}_1 + t\overline{f}_2 + \overline{H}_0$. A basis for G resp. S as R_I-modules is $B_G := \{x^i \mid d_I \leq i < d_1 + d_2 - d_I\}$ resp. $B_S := \{(x^i, 0), (0, x^j) \mid i < d_2 - d_I,\ j < d_1 - d_I\}$. The same B_G resp. B_S is also a basis for $\overline{G}$ resp. $\overline{S}$.

Counting B_S and B_G one sees that S and G are free R_I-modules of rank $n_0 := d_1 + d_2 - 2d_I$. Viewed as free D_I-modules, their ranks are $n := \deg_z(m)n_0$, so ϕ can be represented by an n by n matrix M with entries in D_I. In the same way, we can view $\overline{S}$ and $\overline{G}$ as $\overline{D}$-vector spaces of dimension n, and $\overline{\phi}$ is given by an n by n matrix $\overline{M}$ which equals $M \bmod \langle I\rangle$.

We will show that $\overline{\phi}$ is one to one. The extended Euclidean algorithm produces $(s_I, t_I) \in \overline{S}$ with $s_I\overline{f}_1 + t_I\overline{f}_2 = g_I$. Now suppose that $\overline{\phi}(s, t) = 0$, i.e. $s\overline{f}_1 + t\overline{f}_2 \in \overline{H}_0$. Now $\overline{f}_1, \overline{f}_2$, and hence $s\overline{f}_1 + t\overline{f}_2$ are divisible by g_I (divisions by g_I work because g_I is monic). But g_I has higher degree than any element of $\overline{H}_0$. Hence $s\overline{f}_1 + t\overline{f}_2 = 0$. Then $s\overline{f}_1$ and thus also $ss_I\overline{f}_1$ are divisible by $\overline{f}_2$ (divisions by $\overline{f}_2$ work because $\overline{l}_2$ is a unit). Then $0 \equiv ss_I\overline{f}_1 \equiv s(s_I\overline{f}_1 + t_I\overline{f}_2) \equiv sg_I \bmod \overline{f}_2$. So sg_I vanishes mod $\overline{f}_2$, yet it has lower degree than $\overline{f}_2$, so sg_I must be 0, which implies $s = 0$ because g_I is monic. Now $t\overline{f}_2 = s\overline{f}_1 + t\overline{f}_2 = 0$, which implies $t = 0$ because $\overline{l}_2$ is a unit. This proves that $\overline{\phi}$ is one to one, so $\det(\overline{M}) \in \overline{D}$ is not zero. So the image of $\det(M)$ in $\overline{D}$ is not zero. Then $\det(M)$ is a unit in D_I. This implies that ϕ is invertible.

Now take $w = x^{d_I} + H_0 \in G$ and $\overline{w} = x^{d_I} + \overline{H}_0 \in \overline{G}$. Note that $\overline{w} = g_I + \overline{H}_0$ and hence $\overline{\phi}^{-1}(\overline{w}) = (s_I, t_I)$. Now consider $\phi^{-1}(w)$, which is in S, and hence can be written as (s, t) where $s, t \in R_I[x]$. By definition of ϕ, we see that $sf_1 + tf_2 + H_0 = x^{d_I} + H_0$ so $h := sf_1 + tf_2 \in R_I[x]$ is a monic polynomial of degree d_I.

Since $\overline{M}$ is $M \bmod \langle I\rangle$ we see that $s_I = \overline{s}$, $t_I = \overline{t}$ and $g_I = \overline{h}$. Here $\overline{s}, \overline{t}, \overline{h}$ are the images of s, t, h in $\overline{R}[x]$. Now $g \in L[x]$ was assumed to be a monic common factor of f_1, f_2, so it must be a monic factor in $L[x]$ of h as well. This shows $d_g \leq \deg_x h = d_I$. If $d_g = d_I$ then the monic polynomial h is divisible by the monic polynomial g of the same degree, and hence $g = h$. Then $g \in R_I[x]$ so $\overline{g}$ is defined and equals $\overline{h} = g_I$. Since g is assumed to be a monic common factor of f_1, f_2, and $g = sf_1 + tf_2$ we see that g is the monic gcd of f_1, f_2. This completes the proof.

2.2 Algorithm MODGCD

Recall that $D = \mathbb{Z}[t_1, \ldots, t_k]$, $F = \mathbb{Q}(t_1, \ldots, t_k)$, and $L = F[z]/\langle m(z) \rangle$. Our modular GCD algorithm calls subroutine M which calls recursive subroutine P which in turn calls the Euclidean algorithm over a finite ring.

Algorithm MODGCD
Input: Non-zero polynomials $f_1, f_2 \in L[x]$ and $m(z)$.
Output: $\check{g}$, where g is the monic gcd of f_1 and f_2 in $L[x]$.

Call subroutine M with input $\check{f}_1, \check{f}_2$ and $\check{m}(z)$.

Subroutine M.
Input: $f_1, f_2 \in D[z]/\langle m(z) \rangle[x]$ and $m(z) \in D[z]$.
Output: $\check{g}$, where g is the monic gcd of f_1 and f_2.

1. Set $n = 1$.
2. Main Loop: Take a new prime p_n that is not lc-bad.
3. Let $g_n \in \mathbb{Z}_{p_n}[t_1, \ldots, t_k][z, x]$ be the output of subroutine P applied to $f_1, f_2, m(z) \bmod p_n$.
4. If $g_n =$"failed" then go back to step 2.
5. If $g_n = 1$ then return 1.
6. Select from $\{g_1, \ldots g_n\}$ those with the same leading term (in pure lexicographic order with $x > t_1 > \cdots > t_k$) as g_n. Combine those using Chinese remaindering to obtain $c \bmod m_c$. Set $n = n + 1$.
7. Apply integer rational reconstruction to obtain h from $c \bmod m_c$. If this fails, go back to step 2.
8. Clear fractions in $\mathbb{Q}$: Set $h = \check{h}$.
9. Trial division: If $h|f_1$ and $h|f_2$ then return h, otherwise, go back to step 2.

Subroutine P.
Input: $f_1, f_2 \in D_p[z]/\langle m(z) \rangle[x]$ and $m(z) \in D_p[z]$ where $D_p = \mathbb{Z}_p[t_1, \ldots, t_k]$.
Output: Either $\check{g}$, the primitive associate of the monic gcd of f_1 and f_2, or "failed."

0. If $k = 0$ then output the result of the Euclidean algorithm applied to f_1, f_2 (which returns "failed" if it tries to divide by a zero-divisor).
1. Set $n = 1$ and $d = 1$.
2. Main Loop: Take a random new evaluation point α_n in $\mathbb{Z}_p$ that is not lc-bad.
3. Let $g_n \in \mathbb{Z}_p[t_1, \ldots, t_{k-1}][z, x]$ be the output of subroutine P applied to $f_1, f_2, m(z)$ at $t_k = \alpha_n$.
4. If $g_n =$"failed" then
   Set $d = d + 1$.
   If $d > n$ output "failed", else go back to step 2.
5. If $g_n = 1$ then return 1.
6. Select from $\{g_1, \ldots g_n\}$ those with the same leading term in $x, t_1, \ldots, t_{k-1}$ as g_n. Chinese remainder those to obtain $c \bmod m_c(t_k)$. Set $n = n + 1$.
7. Apply rational function reconstruction to the coefficients of c to obtain $h \in \mathbb{Z}_p(t_k)[t_1, \ldots, t_{k-1}][z, x]$ s.t. $h \equiv c \bmod m_c(t_k)$. If this fails, go back to step 2.
8. Clear fractions in $\mathbb{Z}_p(t_k)$: Set $h = \check{h}$.
9. Trial division: If $h|f_1$ and $h|f_2$ then return h, otherwise, go back to step 2.

If we often encounter "failed", or if we run out of evaluation points in step 2 in subroutine P, then we should increase the size of the primes used (this will not be necessary in practice if one uses 30 bit primes). Since lc-bad and fail prime/evaluations are discarded in steps 2 and 4, of subroutines M and P, the prime/evaluations remaining are either unlucky or good by Theorem 2. The g_i combined in step 6 have the same leading term, thus the same degree in x, and hence they are either all unlucky or all good.

The images g_i in step 6 must be scaled in a consistent way before applying Chinese remaindering, see also Remark 2 below. The rational reconstruction in step 7 in subroutine M is accomplished with Wang's algorithm [7, 2, 4]. Suppose this succeeds and step 7 finds h. If $h|f_1$ and $h|f_2$ then $\check{h} = \check{g}$ and M terminates. If either trial division fails then Theorem 1 implies either m_c is not yet large enough to recover the coefficients in $\mathbb{Q}$ or all primes dividing m_c were unlucky.

Zero divisors in subroutine P

Subroutine P attempts to compute (if it exists) the gcd of f_1, f_2 in characteristic p by choosing points $\alpha = (\alpha_1, \ldots, \alpha_k)$ in $\mathbb{Z}_p^k$ and computing $\gcd(f_1(\alpha), f_2(\alpha))$ with the Euclidean algorithm (in step 0). This is a gcd computation over a finite ring which may fail if a zero divisor is encountered. If we use sufficiently large prime numbers and *random* evaluation points then the probability of such failure is small, so our modular GCD algorithm will terminate.

We explain the role of the variable d in subroutine P. Suppose $m(z) = z^2 + 7t - 1$ and $f_1 = x^2 + t$ and $f_2 = (z + 1)x + t$. If subroutine M chose the prime $p = 7$ then when the Euclidean algorithm is called by algorithm P (in step 0) with inputs $f_1(\alpha)$ and $f_2(\alpha)$, the first division would fail when it attempts to invert $\mathrm{lc}_x(f_2) = z + 1$ which is a zero-divisor for any choice of $\alpha \in \mathbb{Z}_7$. So when $p = 7$ then step 0 fails with probability 1. Suppose subroutine M chose a prime $p \neq 7$. This time only one evaluation point ($\alpha = 0$ for t) results in a zero divisor being encountered, so step 0 fails with probability $1/p$. When most evaluations fail (in the example when $p = 7$) then $d > n$ will soon hold in step 4, so subroutine P wastes little time on p and gives up quickly. If most evaluations are good, and if subroutine P has already computed many good g_n, then the test $d > n$ prevents, with high probability, that a few unlucky choices in step 2 could cause a lot of useful work to be lost.

Remark 1: It is essential that the evaluation point $\alpha_n \in \mathbb{Z}_p$ be chosen at random in step 2. Consider $m(z) = z^2 + (t - 1)z - 1$ and $f_2 = (z+1)x + t$. If subroutine P were to always start with $\alpha_1 = 1$, then on this example, for any p it would always start with hitting a zero divisor and then give up in step 4. This would cause subroutine M to loop.

Reconstruction of the parameters in subroutine P

We explain the choice of representation used by subroutine P for the coefficients of g and how they are reconstructed. Suppose $g = x + \sqrt{s}/(s - t^2) - t/(s - t^2)$ is the gcd of the inputs of subroutine P. Three possible choices for *canonical* representations are:
(1) $x + \sqrt{s}/(s - t^2) - t/(s - t^2)$,
(2) $(s - t^2)x + \sqrt{s} - t$, and
(3) $(\sqrt{s} + t)x + 1$.
Number (1) is the monic gcd $g \in \mathbb{Z}_p(s, t)[\sqrt{s}, x]$. (2) is $\check{g} \in$

$\mathbb{Z}_p[s,t][\sqrt{s}][x]$. It is obtained from (1) by clearing fractions. If there remains a variable that does not appear in $m(z)$ then (3) is obtained from (2) by dividing (2) by the gcd of the coefficients in x, i.e., by $\gcd(s-t^2, \sqrt{s}-t) = \sqrt{s}-t$.

We prefer (2) to (1) because we avoid arithmetic with fractions in s,t and can compute it using only univariate rational function reconstruction and univariate gcd computation. Although form (3) may be smaller than (2), computing it requires recursive gcd computations with one less variable which could be expensive. A precise definition of form (2) is now given. We require that

(i) g is *a* gcd of f_1, f_2 (in characteristic p).
(ii) g is in $D_p[z,x]$ where $D_p = \mathbb{Z}_p[t_1,\ldots,t_k]$.
(iii) g is primitive in $D_p[z,x]$.
(iv) $\mathrm{lc}_x(g)$ is in D_p, i.e., $\deg_z(\mathrm{lc}_x\, g) = 0$, and
(v) (for uniqueness) $\mathrm{lc}_{t_1,\ldots,t_k}(\mathrm{lc}_x\, g) = 1$ under a term ordering.

We show how subroutine P outputs form (2) using univariate rational function reconstruction. For $p = 11$, $t = 1$, suppose we computed the gcd at $s = 2,3,4$ with the Euclidean algorithm and have

$$g(2,1) \bmod 11 = x + 1z - 1,$$
$$g(3,1) \bmod 11 = x - 5z + 5,$$
$$g(4,1) \bmod 11 = x + 4z - 4.$$

In step 7 subroutine P applies rational function reconstruction to the coefficients of $z^i x^j$ to find rational functions in $\mathbb{Z}_p(s)$ of the form $(as+b)/(cs+d)$. We obtain

$$h(s,1) \bmod 11 = \frac{1}{1}x + \frac{1}{s-1}z + \frac{-1}{s-1}.$$

Now clear denominators to obtain form (2) at $t = 1$.

$$g(s,1) \bmod 11 := (s-1)h(s,1) = (s-1)x + z - 1.$$

We repeat this for $t = 2,3,4$ so that we can reconstruct rational functions in t of the form $(at^2+bt+c)/(dt+e)$ for the coefficients of $s^i z^j x^k$. We obtain

$$h(s,t) \bmod 11 = (\frac{1}{1}s + \frac{-t^2}{1})x + \frac{1}{1}zs + \frac{-t}{1}.$$

There are no denominators in t to clear in our example. Thus we obtain $(s-t^2)x + z - t$. Since the leading term, in lexicographical order with $x > s > t$, is $1sx$, condition (v) already holds so no rescaling needs to be done.

Remark 2: In step 6 in subroutines M and P, there are two reasons we only combine images with the same leading term. First, this way we Chinese remainder only images that are scaled by condition (v) in a consistent way. Second, it prevents reconstruction problems arising from an *unlucky content*. Suppose that p_1 is a good prime, $\check{g} = tx + t + p_1 z$ and that subroutine M chooses prime p_1. Then $\check{g} \bmod p_1 = tx + t$ has an unlucky content t, and g_1 will be the monic associate of $tx+t$ which is $g_1 = x+1$. Hence g_1 is *not* equal to $\check{g} \bmod p_1$. Therefore, we must not Chinese remainder it with images g_i that are equal to $\check{g} \bmod p_i$. The easiest way to prevent this is to Chinese remainder only those g_i together that have the same leading term. This way g_1 will not contribute to the reconstruction of $\check{g}$. Note that if $\mathrm{lc}_{t_1,\ldots,t_k}(\mathrm{lc}_x(\check{g}))$ vanishes mod a good prime p_2, then the image mod p_2 will end up not being used even if there is no unlucky content. So our algorithm may compute images like these g_1 or g_2 that end up not being used. The reason we make no effort to avoid (as in Brown's algorithm) such images *before* they have been computed is because that would require additional computations and slow down our algorithm (a simple check on $\mathrm{lc}_t(\mathrm{lc}_x(f_i))$ without computing some resultant (mod p) is not sufficient, take for example $f_1 = f_2 = (t^2 + p_2\, z)x + 1$ where $z = \sqrt{t^7}$).

Remark 3: The algorithm as stated assumes $m(z)$ is irreducible so L is a field. Our implementation does not check this assumption because that would be expensive, and in many applications where L is known to be a field, unnecessary. There are applications for gcds in $L[x]$ when L is not a field. In this case it is possible that our algorithm as stated will loop. Take for example $m(z) = z^2 - t^2$ and $f_1 = f_2 = (z-t)x + 1$. The algorithm can be modified to handle this case. The basic idea is to reconstruct the zero divisor from some of its images while we reconstruct the gcd, and terminate when trial division confirms either a zero divisor or a gcd.

The Trial Divisions

In step 9 in subroutine P the trial divisions $g|f_1$ and $g|f_2$ take place in characteristic p. Since g, f_1 and f_2 are in $D_p[z]/\langle m(z)\rangle[x]$ where $D_p = \mathbb{Z}_p[t_1,\ldots,t_k]$ and $m(z) \in D_p[z]$ we can use pseudo-division in $D_p[z][x]$ modulo $m(z)$ where we divide by $m(z)$ also using pseudo-division. Thus we can do the trial divisions without any gcd computation in D_p. However, this results in a linear growth in size of the coefficients in D_p. In [5] we presented an algorithm for doing trial divisions of polynomials in $\mathbb{Z}[z][x]$ modulo $m(z) \in \mathbb{Z}[z]$ which uses pseudo-division *and* some gcds in $\mathbb{Z}$ to minimize growth of the integer coefficients. The same algorithm can be applied here; the only difference is the coefficient ring, D_p instead of $\mathbb{Z}$. Our algorithm in [5] can also be used for the trial divisions in subroutine M. There f_1, f_2 and h are in $D[z]/\langle m(z)\rangle[x]$ where $D = \mathbb{Z}[t_1,\ldots,t_k]$. Again, we do pseudo-division in $D[z][x]$ modulo $m(z)$ using gcds in D. Because we have effective algorithms for gcd computation in D and D_p, this approach is effective in practice. To prevent the algorithm from doing many trial divisions, we build into the rational reconstruction algorithm some redundancy so that if it succeeds with output h then $\check{h} = \check{g}$ with high probability.

Rational Function Reconstruction

Let $n, d \in \mathbb{Z}_p[t]$ with $d \neq 0$, $\mathrm{lc}_t(d) = 1$, and $\gcd(n,d) = 1$. Let $m = m_c(t)$ be the modulus in subroutine P and (assuming $\gcd(d,m) = 1$) let $u = n/d \bmod m$. The rational reconstruction problem is: given u and m find n and d. The problem is solved by the Euclidean algorithm as follows (this is Wang's algorithm for $\mathbb{Q}$ (see [7]) modified for $\mathbb{Z}_p(t)$).

Algorithm Rational Reconstruction (RR)
Input: Polynomials $m, u \in \mathbb{Z}_p[t]$ with $\deg_t m > 0$.
Output: Either polynomials $a, b \in \mathbb{Z}_p[t]$ with $\deg_t a + \deg_t b < \deg_t m$, $\mathrm{lc}_t(b) = 1$, $\gcd(a,b) = 1$, $\gcd(b,m) = 1$ and $a/b \equiv u \bmod m$, or FAIL if no such a/b exists.

Set $M = \deg_t m$, $N = \lfloor M/2 \rfloor$, $D = M - N - 1$.
Set $(r_0, s_0) = (m, 0)$.
Set $(r_1, s_1) = (u, 1)$.

While $\deg_t r_1 > N$ do
  Set q to be the quotient of r_0 divided r_1.
  Set $(r_0, r_1) = (r_1, r_0 - q\,r_1)$.
  Set $(s_0, s_1) = (s_1, s_0 - q\,s_1)$.
Set $(a, b) = (r_1, s_1)$.
If $\deg_t b > D$ or $\gcd(b, m) \neq 1$ then output FAIL.
Make b monic: Set $u = \mathrm{lc}_t\, b$ and $(a, b) = (a/u, b/u)$.
Output (a, b).

If $N \geq \deg_t(n)$ and $D \geq \deg_t(d)$ then algorithm RR outputs (n, d), otherwise, it outputs FAIL or some $(a, b) \neq (n, d)$.

In each loop of subroutine P, the degree of $m_c(t_k)$ increases by 1. In step 7 in P, algorithm RR is applied to each coefficient of c until it fails. Thus algorithm RR will be applied many times to many coefficients before it succeeds. Suppose c has T terms in $t_1, \ldots, t_{k-1}, z, x$ and suppose subroutine P requires S points to succeed. Then the expected number of calls to algorithm RR is ST. Moreover, the trial divisions in subroutine P will be attempted many times with $h \neq \check{g}$.

We force algorithm RR to output FAIL with high probability when $M = \deg_t(m)$ is not large enough for RR to reconstruct a coefficient of h by setting $N = \lfloor (M - 1)/2 \rfloor$ and $D = M - N - 2$ so that $N + D = M - 2$, i.e., $\deg_t(m) = N + D + 2$. In the Euclidean algorithm, since $\deg_t(q) + \deg_t(r_1) + \deg_t(s_1) = \deg_t(m)$ the new settings of N and D mean we are forcing the q in algorithm RR that corresponds to the output to be of degree 2 or more. For u chosen at random in $\mathbb{Z}_p[t]$ of degree less than M, the probability of getting any q of degree 2 or more is $\mathrm{O}(M/p)$. Thus, by requiring one evaluation point more than the minimum needed, with high probability, we attempt the trial divisions once, and secondly, we reduce the expected number of calls to algorithm RR from $O(ST)$ to $O(S + T)$.

2.3 Multivariate Inputs

Let $f_1, f_2 \in L[x_1, ..., x_n]$ with $n > 1$. To compute the gcd g of f_1 and f_2 using our modular GCD algorithm do the following: write $f_1 = \sum a_i x_1^i$ and $f_2 = \sum b_j x_1^j$ where $a_i, b_j \in L[x_2, \ldots, x_n]$. Now recall that $\gcd(f_1, f_2) = cb$ where c is the gcd of the contents in x_1 of f_1 and f_2, i.e., $c = \gcd(a_i, b_j)$, and b is the gcd of $h_1 = f_1/c$ and $h_2 = f_2/c$. Computing c requires gcd computations in $L[x_2, \ldots, x_n]$, that is, in one less variable. To compute $\gcd(h_1, h_2)$ we view $x_2, \ldots, x_n$ as parameters, that is, we write h_1, h_2 as polynomials in $K[x_1]$ where $K = G[z]/\langle m(z) \rangle$ and $G = \mathbb{Q}(t_1, \ldots, t_k, x_2, \ldots, x_n)$. Now we compute $\check{g}$, the monic associate of their gcd, using our modular GCD algorithm. When $x_2, \ldots, x_n$ are regarded as polynomial variables, $\check{g}$ may have a non-trivial content in x_1 which needs to be computed and divided out to obtain b. That $\check{g}$ could have a non-trivial content is illustrated by the example $(s - t^2)x + \sqrt{s} - t$ given in section 2.2.

3. FRACTION-FREE ALGORITHMS

In this section we develop a *fraction-free* GCD algorithm for $L[x]$ based on the ideas of Moreno Maza and Rioboo in [6]. Their algorithm, a modification of the subresultant GCD algorithm, computes an associate of the gcd of two univariate polynomials modulo a triangular set of polynomials over an integral domain D. Our problem setting is a special case of theirs where we have a triangular set of one polynomial, $m(z) = \check{m}(z)$, which is irreducible over $\mathbb{Z}[t_1, \ldots, t_k]$. We recall the idea of pseudo-division in $D[x]$.

Definition: Let $a(x), b(x)$ be non-zero polynomials in $D[x]$ with $\deg a \geq \deg b$. Let $\delta = \deg a - \deg b + 1$. The *pseudo remainder* $\bar{r}$ and *pseudo quotient* $\bar{q}$ are the remainder and quotient, respectively, of μa divided by b where $\mu = \mathrm{lc}_x\, b^\delta$. Thus $\mu a = b\bar{q} + \bar{r}$ where $\bar{r} = 0$ or $\deg \bar{r} < \deg b$.

The point of pseudo-division is that $\bar{r}$ and $\bar{q}$ are in $D[x]$ and no fractions appear in the division algorithm when dividing μa by b. The subresultant GCD algorithm on input of non-zero f_1 and f_2 in $D[x]$, uses pseudo-division to compute an associate of the gcd of f_1 and f_2 in $D[x]$. It does this without introducing fractions and without gcd computation in D. Moreover, one can show that if $D = \mathbb{Z}[t_1, \ldots, t_k]$ the size of the integer coefficients and degree in each parameter of the coefficients of the pseudo-remainder sequence (PRS) grows linearly which is is optimal to within a constant factor. This is achieved by dividing pseudo-remainders by known exact divisors in D.

In [6], Moreno Maza and Rioboo modify the subresultant PRS to work for $R[x]$ where $R = D[z]/\langle m(z) \rangle$. The main idea is to multiply a pseudo-remainder $p(x) \in R[x]$ by a scalar $i \in R$, a quasi-inverse of $\mathrm{lc}_x(p)$, so that after multiplication, $\mathrm{lc}_x(ip) \in D$ and not $D[z]$. Now pseudo-division does not introduce fractions and the exact divisions (Moreno Maza and Rioboo prove that they remain exact) in the subresultant PRS are by elements of D not R. We recall the definition for quasi-inverse for our commutative ring R.

Definition: Let $u \in R$. Then $v \in R$ is a *quasi-inverse* of u if $uv = r$ for some $r \in D$.

Remark 4: The definition is unique up to multiplication by a non-zero element of D. To reduce coefficient growth we will want to use a quasi-inverse which is smallest.

Example: Let $u \in R = \mathbb{Z}[z]/\langle m \rangle$ where $m \in \mathbb{Z}[z]$ is monic. Then there exist polynomials $s, t \in \mathbb{Z}[z]$ such that $sm + tu = r$ where $r \in \mathbb{Z}$ is the resultant of m and u. Thus $v = t$ is a quasi-inverse of u in R. Let $g = \gcd(r, t)$. Now $t \in \mathbb{Z}[z]$ and so g is the gcd of r and the coefficients of t. Then t/g is also a quasi-inverse of u; it is a minimal quasi-inverse.

To compute a quasi-inverse of $u \in R = D[z]/\langle m(z) \rangle$ we can use either Collin's reduced PRS, or the subresultant PRS, or the primitive PRS to compute elements $s, t \in D[z]$ such that $sm + tu = r$ for some $r \in D$ and output t/g where $g = \gcd(r, t)$. Here is Maple code to accomplish this using the reduced PRS for $D = \mathbb{Z}[t_1, \ldots, t_k]$.

```
QuasiInv := proc(x,m,z)
local u,r0,r1,t0,t1,pr,pq,mu,beta;
  u := primpart(x,z);
  r0,r1,t0,t1 := m,u,0,1;
  beta := 1;
  while degree(r1,z)>0 do
    r0,r1 := r1,prem(r0,r1,z,'mu','pq');
    t0,t1 := t1,expand(mu*t0-pq*t1);
    if r1=0 then ERROR("hit zero divisor",r0) fi;
    divide(r1,beta,'r1');
    divide(t1,beta,'t1');
    beta := mu;
  od;
  divide(t1,gcd(r1,t1),'t1');
```

```
  return t1;
end;
```

Below is Maple code for the modified subresultant PRS for computing an associate of the gcd of f_1, f_2 in $D[z]/\langle m(z)\rangle$. Moreno Maza and Rioboo prove that the divisions by elements of D remain exact. Note, if $m(z)$ is not monic in $D[z]$ the divisions by $m(z)$ (the Maple command `rem(...,m,z);`) may be replaced by pseudo-division by $m(z)$ without breaking the exact division properties of the subresultant PRS.

```
MMRPRS := proc(f1,f2,x,m,z)
local p1,p2,p3,del,alp,psi,bet,i;
  p1,p2 := f1,f2;
  del := degree(p1,x)-degree(f2,x);
  if del<0 then return MMRPRS(f2,f1,x,m,z) fi;
  i := QuasiInv(lcoeff(p2,x),m,z);
  p2 := rem(i*p2,m,z);
  psi,bet,alp := -1,(-1)^(del+1),lcoeff(p2,x);
  while degree(p2,x)>0 do
    p3 := rem(prem(p1,p2,x),m,z);
    if p3=0 then return p2 fi;
    divide(p3,bet,'p3');
    i := QuasiInv(lcoeff(p3,x),m,z);
    p3 := rem(i*p3,m,z);
    if del>0 then
       divide((-alp)^del,psi^(del-1),'psi')
    fi;
    p1,p2 := p2,p3;
    if degree(p2,x)>0 then
       del := degree(p1,x)-degree(p2,x);
       bet := -alp*psi^del;
       alp := lcoeff(p2,x);
    fi;
  od;
  p2;
end:
```

However, multiplication of p_3 by a quasi-inverse $i \in D[z]$ introduces a new coefficient growth in D into the subresultant PRS. We observe that the growth is approximately cubic instead of linear. We illustrate this. For $D = \mathbb{Z}$, $m = z^3 + 80z^2 + 23z - 20$, we created two polynomials f_1 and f_2 of degree 21 and 20 in x respectively, with coefficients polynomials of degree 2 in z with integer coefficients chosen at random from (-100,100) in Maple as follows.

```
> c := () -> randpoly(z,degree=2,dense):
> f1 := randpoly(x,coeffs=c,degree=21,dense):
> f2 := randpoly(x,coeffs=c,degree=20,dense):
```

The data below shows the degree (column 1) and length in decimal digits of the largest integer coefficient of the polynomials p_3 appearing in the Moreno Maza and Rioboo PRS (columns 2 and 3) and the primitive PRS (columns 4 and 5). Columns 2 and 3 show, respectively, the length of the largest integer coefficient of p_3 before and after the quantity *bet* is divided out. Let r_3 denote the pseudo remainder of the primitive parts of p_1 and p_2. Columns 4 and 5 show, respectively, the length of the largest integer coefficient of r_3 and the primitive part of r_3.

The data shows that after 20 steps the length of the integers in the Moreno Maza and Rioboo algorithm is over 100 times longer than those in the primitive PRS. We see also that after an initial step, the size of the integers in column 5 grows each step by approximately 42 digits and the size of those in column 4 by approximately 62 digits. This is a linear growth. In columns 2 and 3 the growth is approximately cubic. There are no parameters present in our example. If there were a parameter t present in m and f_1 and f_2, what we would see is a similar growth in the degree in t, i.e., there would be a two dimensional growth.

	2	3	4	5
19	21	21	21	20
18	96	79	62	53
17	308	221	121	95
16	760	497	182	135
15	1572	945	241	174
14	2867	1609	303	216
13	4763	2527	365	257
12	7386	3742	428	299
11	10854	5293	490	342
10	15302	7229	554	384
9	20757	9592	619	427
8	27638	12417	678	465
7	35777	15750	743	509
6	45390	19630	807	553
5	56614	24105	872	596
4	69576	29218	936	639
3	84409	35014	1003	684
2	101248	41538	1072	729
1	120220	48826	1132	766
0	141447	56925	1193	810

Table 1: Integer Coefficient Growth

If instead of modifying the subresultant PRS we modify the primitive PRS to compute a gcd in $R[x]$ by making the pseudo remainders primitive in $D[x, z]$ then we get minimal (linear) coefficient growth. To do this we assume we can compute gcds in D effectively. The same idea can be applied to the general setting of the algorithm of Moreno Maza and Rioboo in [6] for gcd computation modulo a triangular set; instead of using the (extended) subresultant PRS use the (extended) primitive PRS throughout. Here is a Maple code to do this for $D = \mathbb{Z}[t_1, \ldots, t_k]$.

```
PrimitivePRS := proc(f1,f2,x,m,z)
local i,r1,r2,r3;
   r1 := primpart(f1,[x,z]);
   r2 := primpart(f2,[x,z]);
   while r2 <> 0 do
      i := QuasiInv(lcoeff(r2,x),m,z);
      r2 := primpart(prem(i*r2,m,z), [x,z]);
      r3 := prem(prem(r1,r2,x),m,z);
      r3 := primpart(r3,[x,z]);
      r1,r2 := r2,r3;
   od;
   r1;
end;
```

The number of gcds in $D = \mathbb{Z}[t_1, \ldots, t_k]$ that are computed when taking the primitive part of a polynomial in $D[x, z]$ of degree n in x and d in z can be effectively reduced from $O(nd)$ to 1 by computing the gcd of the smallest coefficient with a random linear combination of the others. Thus the expected total number of gcds in D that the primitive PRS does is $O(\delta)$ where $\delta = \max(\deg_x f_1, \deg_x f_2) - \deg_x g + 1$.

4. IMPLEMENTATION

We compare a Maple implementation of our algorithms on the following problem set. Let

$$m(z) = z^3 - (5-t)z^2 + (7-t^2)z - (9-t^3),$$

$$g = (10-4\,t)x^2 - 5\,xz^2 + (4\,t+1)xz + (11-17\,t^2+9\,t)x - 19\,z^2 + (-7\,t+6)z + (-11\,t^2+15\,t+3),$$

$$a = (18+10\,t)x^2 + 10\,xz^2 + (17\,t+2)xz + (2+17\,t^2+8\,t)x + 6\,z^2 + (17\,t+6)z + (4\,t^2-4\,t+2),$$

$$b = (-8-11\,t)x^2 - 14\,xz^2 + (8\,t-4)xz + (-17-5\,t^2+19\,t)x - 11\,z^2 + (17\,t-4)z + (-14\,t^2-19\,t-2).$$

For $n = 10$, $k = 0, 1, \ldots, n$ let $f_1 = g^k a^{n-k}$ and $f_2 = g^k b^{n-k}$. Thus there are 11 gcd problems, each of degree 20 in x and 20 in t where the gcd g^k has degree $2k$ in x and degree $2k$ in t with leading coefficient degree k in t. The reason for this choice of inputs is to compare the algorithms as the size of the gcd increases relative to the size of the cofactors. The modular GCD algorithm will do best for small k and the PRS algorithms will do best for large k.

The timings below (in CPU seconds) were obtained using Maple 9 on a Itanium PC running Linux. Column 1 is the time for the Moreno Maza Rioboo algorithm. Column 2 is the time for our primitive PRS algorithm. Column 3 is the time for our modular GCD algorithm (shown in parentheses is the number of primes used). Column 4 is the time for an improved version of the modular GCD algorithm.

k	1	2	3		4
0	NA	NA	0.07	(1)	0.06
1	NA	NA	0.77	(2)	0.22
2	NA	NA	1.56	(3)	0.54
3	NA	NA	2.86	(3)	0.84
4	NA	NA	4.39	(5)	1.24
5	NA	7459.7	5.43	(6)	1.99
6	NA	1488.7	7.70	(8)	2.61
7	NA	259.6	8.01	(9)	3.34
8	4473.1	42.8	7.70	(10)	4.18
9	31.1	1.2	7.22	(11)	5.07
10	0.0	0.0	6.29	(12)	5.72

Table 2: NA means not attempted

The data demonstrates clearly the superiority of our modular GCD algorithm. Much of the time in our implementation is spent doing trial divisions in subroutine P, that is, in the ring $\mathbb{Z}_p(t)[z]/\langle m(z)\rangle[x]$. Column 4 shows the improvement we obtained by making the following change to our modular GCD algorithm. We modify subroutine P so that if rational function reconstruction succeeds we assume the result is correct and simply return h without trial division. Thus when subroutine P calls itself recursively on input $A_i = f_1(t_k = \alpha_i)$, $B_i = f_2(t_k = \alpha_i)$, and $m_i = m_c(t_k = \alpha_i)$, it is now possible that an output g_i from subroutine P is wrong, i.e., g_i does not divide A_i or g_i does not divide B_i modulo $m_i(z)$. We apply the following strategy to detect an incorrect g_i so that subroutine P will eventually terminate. In subroutine P, if $k > 1$, periodically, we choose $\beta = (\beta_1, \ldots, \beta_{k-1}) \in \mathbb{Z}_p^{k-1}$ at random s.t. $\mathrm{lc}_x(g_i)(\beta) \neq 0$ and $\mathrm{lc}_z(m_i)(\beta) \neq 0$. We test whether $g_i(\beta)|A_i(\beta)$ and $g_i(\beta)|B_i(\beta)$ modulo $m_i(\beta, z)$. Thus we replace division over a function field in $k-1$ parameters with division over a finite ring. With high probability this check will identify an incorrect g_i. Once identified, we discard g_i in subroutine P and adjust the c and m_c accordingly. We modify subroutine M similarly to detect incorrect output from subroutine P.

It remains to say how often we make these checks. At step n, after computing g_n, if $2^j | n-1$ for some $j > 2$ we check $g_{n-2^j}, g_{n-2^j+1}, \ldots, g_{n-1}$ for correctness. Thus the first time we make any checks is at step $n = 9$ where we check $g_1, g_2, \ldots, g_8$. The next time is at step 17 when we check $g_1, g_2, \ldots, g_{16}$. The next is at step 24 when we check $g_{17}, \ldots, g_{24}$. Thus we check each g_i within 8 steps of being computed and then with decreasing frequency. In practice, because of the way we have designed the rational function reconstruction, the output from subroutine P is hardly ever wrong, hence, the infrequent checks are justified.

5. REFERENCES

[1] W. S. Brown (1971). On Euclid's Algorithm and the Computation of Polynomial Greatest Common Divisors, *J. ACM* **18**, pp. 476–504.

[2] G. E. Collins and M. J. Encarnacion (1995). Efficient Rational Number Reconstruction. *J. Symbolic Computation* **20**, pp. 287–297.

[3] M. J. Encarnacion (1995). Computing GCDs of Polynomials over Algebraic Number Fields, *J. Symbolic Computation* **20**, pp. 299–313.

[4] J. von zur Gathen and J. Gerhard (1999). *Modern Computer Algebra*. University of Cambridge Press.

[5] M. van Hoeij, M. B. Monagan, A Modular GCD Algorithm over Number Fields Presented with Multiple Field Extensions. *Proceedings of ISSAC '2002*, ACM Press, pp. 109–116, 2002.

[6] M. Moreno Maza, R. Rioboo (1995). Polynomial Gcd Computations over Towers of Algebraic Extensions, *Proc. of AAECC-11* Springer-Verlag LNCS **948** (1995), pp. 365–382.

[7] P. Wang (1981). A p-adic Algorithm for Univariate Partial Fractions. *Proceedings of SYMSAC '81*, ACM Press, pp 212-217.