Foundations of Computer Algebra: From Newton to Hensel

Marc Moreno Maza
University of Western Ontario
CS 424 / CS 556 – Winter 2006

January 7, 2008

1 Division with remainder using Newton iteration

We combine in this section the results about the fast polynomial multiplication based on the FFT with Newton iteration in order to provide a fast polynomial division with remainder.

1.1 Classical division with remainder

We assume that R is a commutative ring with identity element. We consider the classical algorithm for univariate polynomial division with remainder stated over R instead of a field. However, Algorithm 1 is well defined because we assume that the leading coefficient of the polynomial b is a unit.

Algorithm 1

Input: univariate polynomials $a = \Sigma_0^n \; a_i x^i$ and $b = \Sigma_0^m \; b_i x^i$ in $R[x]$ with respective degrees n and m such that $n \geq m \geq 0$ and b_m is a unit.

Output: the quotient q and the remainder r of a w.r.t. b. Hence $a = bq + r$ and $\deg r \; < \; m$.

$r := a$
for $i = n - m, n - m - 1, \ldots, 0$ **repeat**
  if $\deg r = m + i$ **then**
    $q_i := \mathsf{leadingCoefficient}(r) \, / \, b_m$
    $r := r - q_i x^i b$
  else $q_i := 0$
$q := \Sigma_0^{n-m} \; q_i x^i$
return (q, r)

Proposition 1 *Let a and b two univariate polynomials in $R[x]$ with respective degrees n and m such that $n \geq m \geq 0$ and the leading coefficient of b is a unit. Then there exist unique polynomials q and r such that $a = bq + r$ and $\deg r \; < \; m$. The polynomials q and r are called the* quotient *and the* remainder *of a w.r.t. to b. Algorithm 1 compute them in $(2m + 1)(n - m + 1) \in \mathcal{O}(n\, m)$ operations in R.*

PROOF $\triangleright$ We leave to the reader the proof of the uniqueness of the quotient and the remainder of a w.r.t. to b. So we focus now on the complexity result Consider an iteration of the **for** loop where $\deg r = m + i$ holds at the begining of the loop. Observe that r and $q_i x^i b$ have the same leading coefficient. Since $\deg b = m$, computing the reductum of $q_i x^i b$ requires m operations. Then subtracting the reductum of $q_i x^i b$ to that of r requires again m operations. Hence each iteration of the **for** loop requires at most $2m + 1$ operations in R, since we need also to count 1 for the computation of q_i. The number of **for** loops is $n - m + 1$. Therefore Algorithm 1 requires $(2m + 1)(n - m + 1)$. $\triangleleft$

Remark 1 One can derive from Algorithm 1 a bound for the coefficients of q and r, which is needed for performing a modular version (based for instance on the CRT). Let $||a||$, $||b||$ and $||r||$ be the max-norm of a, b and r. Let $\mid b_m \mid$

be the absolute value (over $\mathbb{Z}$) or the norm (over $\mathbb{C}$) of the leading coefficient of b. Then we have

$$||r|| \;\leq\; ||a|| \left(1 + \frac{||b||}{\mid b_m \mid}\right)^{n-m+1} \tag{1}$$

1.2 The quotient as a modular inverse

We shall see now that the complexity result of Proposition 1 can be improved. To do so, we will show that q can be computed from a and b by performing essentially one multiplication in $R[x]$. We start with the equation

$$a(x) \;=\; q(x)\, b(x) + r(x) \tag{2}$$

where a, b, q and r are in the statement of Proposition 1. Replacing x by $1/x$ and multiplying the equation by x^n leads to the new equation:

$$x^n\, a(1/x) \;=\; \left(x^{n-m} q(1/x)\right)\; \left(x^m\, b(1/x)\right) \;+\; x^{n-m+1} \left(x^{m-1}\, r(1/x)\right) \tag{3}$$

In Equation (3) each of the rational fractions $a(1/x)$, $b(1/x)$, $q(1/x)$ and $r(1/x)$ is multiplied by x^e such that e is an upper bound for the degree of its denominator. So Equation (3) is in fact an equation in $R[x]$. Definition 1 and Proposition 2 highlight this fact. Then Proposition 3 and Theorem 1 suggest Algorithm 2 which provides an efficient way to compute the inverse of a univariate polynomial in x modulo a power of x. Finally we obtain Algorithm 3 for fast division with remainder based on Equation (3).

Definition 1 *For a univariate polynomial $p = \Sigma_0^d\, p_i x^i$ in $R[x]$ with degree d and an integer $k \geq d$, the* reversal of order k *of p is the polynomial denoted by $\mathrm{rev}_k(p)$ and defined by*

$$\mathrm{rev}_k(p) \;=\; x^k\, p(1/x) \;=\; \Sigma_{k-d}^k\, p_{k-i} x^i. \tag{4}$$

When $k = d$ the polynomial $\mathrm{rev}_k(p)$ is simply denoted by $\mathrm{rev}(p)$. Hence we have

$$\mathrm{rev}(p) \;=\; p_d + p_{d-1} x + \cdots + p_1 x^{d-1} + p_0 x^d. \tag{5}$$

Proposition 2 *With a, b, q and r as abovce, we have*

$$\mathrm{rev}_n(a) \;\equiv\; \mathrm{rev}_{n-m}(q)\; \mathrm{rev}_m(b) \quad \mod\; x^{n-m+1} \tag{6}$$

PROOF ▷ Indeed with Definition 1 Equation (3) reads

$$\mathrm{rev}_n(a) \;=\; \mathrm{rev}_{n-m}(q)\,\mathrm{rev}_m(b) + x^{n-m+1}\,\mathrm{rev}_{m-1}(r) \tag{7}$$

leading to the desised result. ◁

Remark 2 If R is a field then we know that $\mathrm{rev}_{n-m}(q)$ is invertible modulo x^{n-m+1}. Indeed, $\mathrm{rev}_{n-m}(q)$ has constant coefficient 1 and thus the gcd of $\mathrm{rev}_{n-m}(q)$ and x^{n-m+1} is 1. The case where R is not a field leads also to a simple and surprising solution as we shall see in the next section.

1.3 Modular inverses using Newton iteration

Let R be a commutative ring with identity element. Given $f \in R[x]$ and $\ell \in \mathbb{N}$ such that $f(0) = 1$ compute the polynomials $g \in R[x]$ such that

$$f\,g \equiv 1 \mod x^{\ell} \;\text{ and }\; \deg g < \ell. \tag{8}$$

In Proposition 3 we observe that if there is a solution, then it is unique.

Proposition 3 *If Equation (8) has a solution $g \in R[x]$ with degree less than ℓ then it is unique.*

PROOF ▷ Indeed, let g_1 and g_2 be solutions of Equation (8). Then the product $f(g_1 - g_2)$ is a multiple of x^{ℓ}. Since $f(0) = 1$ then $g_1(0) - g_2(0)$ must be 0. Hence there is a constant $c \in R$ and polynomials h_1, h_2 with degree less than $\ell - 1$ such that

$$g_1(x) \;=\; h_1(x)\,x + c \;\text{ and }\; g_2(x) \;=\; h_2(x)\,x + c \tag{9}$$

It follows that $f(h_1 - h_2)$ is a multiple of $x^{\ell-1}$. By repeating the same argument we show that $h_1(0) = h_2(0)$. Then by induction on ℓ we obtain $g_1 = g_2$. ◁

Remark 3 Since Equation (8) is an equation in $R[x]/\langle x^{\ell}\rangle$, a solution of this equation can be viewed as an approximation of a more general problem. Think of truncated Taylor expansions! So let us recall from numerical analysis the celebrated Newton iteration and let $\phi(g) = 0$ be an equation that we want to solve, where $\phi : \mathbb{R} \longmapsto \mathbb{R}$ is a differentiable function. From a suitable initial approximation g_0, the sequence, called *Newton iteration step*,

$$g_{i+1} \;=\; g_i - \frac{\phi(g_i)}{\phi'(g_i)} \tag{10}$$

allows to compute subsequent approximations and converge toward a desired solution. In our case we have $\phi(g) = 1/g - f$ and the Newton iteration step is

$$g_{i+1} \;=\; g_i - \frac{1/g_i - f}{-1/{g_i}^2} \;=\; 2g_i - f\,{g_i}^2. \tag{11}$$

Theorem 1 *Let R be a commutative ring with identity element. Let f be a polynomial in $R[x]$ such that $f(0) = 1$. Let $g_0, g_1, g_2, \ldots$ be the sequence of polynomials defined for all $i \geq 0$ by*

$$\left\{ \begin{array}{rcl} g_0 & = & 1 \\ g_{i+1} & \equiv & 2g_i - f\,{g_i}^2 \mod x^{2^{i+1}} \end{array} \right. \tag{12}$$

Then for $i \geq 0$ we have

$$f\,g_i \;\equiv\; 1 \mod x^{2^i} \tag{13}$$

Proof $\triangleright$ By induction on $i \geq 0$. For $i = 0$ we have $x^{2^i} = x$ and thus

$$f\,g_i \;\equiv\; f(0)\,g_0 \;\equiv\; 1 \times 1 \;\equiv\; 1 \mod x^{2^i} \tag{14}$$

For the induction step we have

$$\begin{array}{rcll} 1 - f\,g_{i+1} & \equiv & 1 - f(2g_i - f\,{g_i}^2) & \mod x^{2^{i+1}} \\ & \equiv & 1 - 2\,f\,g_i + f^2\,{g_i}^2 & \mod x^{2^{i+1}} \\ & \equiv & (1 - f\,g_i)^2 & \mod x^{2^{i+1}} \\ & \equiv & 0 & \mod x^{2^{i+1}} \end{array} \tag{15}$$

Indeed $f\,g_i \equiv 1 \mod x^{2^i}$ means that x^{2^i} divides $1 - f\,g_i$. Thus $x^{2^{i+1}} = x^{2^i + 2^i} = x^{2^i}\,x^{2^i}$ divides $(1 - f\,g_i)^2$. $\triangleleft$

Algorithm 2

Input: $f \in R[x]$ such that $f(0) = 1$ and $\ell \in \mathbb{N}$.

Output: $g \in R[x]$ such that $f\,g \equiv 1 \mod x^\ell$

$g_0 := 1$
$r := \lceil \log_2(\ell) \rceil$
for $i = 1 \cdots r$ **repeat**
  $g_i := (2g_{i-1} - f\,{g_{i-1}}^2) \mod x^{2^i}$
return g_r

Definition 2 A *multiplication time* is a function $\mathsf{M} : \mathbb{N} \longrightarrow \mathbb{R}$ such that for any commutative ring R with a 1, for every $n \in \mathbb{N}$, any pair of polynomials in $R[x]$ of degree less than n can be multiplied in at most $\mathsf{M}(n)$ operations of R. In addition, M must satisfy $\mathsf{M}(n)/n \geq \mathsf{M}(m)/m$, for every $m, n \in \mathbb{N}$, with $n \geq m$. This implies the *superlinearity* properties, that is, for every $m, n \in \mathbb{N}$

$$\mathsf{M}(nm) \geq m\mathsf{M}(n), \quad \mathsf{M}(n+m) \geq \mathsf{M}(m) + \mathsf{M}(n) \quad \text{and} \quad \mathsf{M}(n) \geq n. \tag{16}$$

Example 1 Examples of multiplication times are:

- Classical: $d \longmapsto 2d^2$;
- Karatsuba: $d \longmapsto C\,d^{\log_2(3)}$ with some C that can be taken equal to 9;
- FFT over an arbitrary ring: $d \longmapsto C\,d\log(d)\log(\log(d))$ for some C that can taken equal to 64 [CK91].

Note that the FFT-based multiplication in degree d over a ring that supports the FFT (that is, possessing primitive n-th root of unity, where n is a power of 2 greater than $2d$) can run in $C\,d\log(d)$ operations in R, with some $C \geq 18$.

Theorem 2 *Algorithm 2 computes the inverse of f modulo x^ℓ in* $3\mathsf{M}(\ell)+0(\ell)$ *operations in R.*

PROOF $\triangleright$ Theorem 1 tell us that Algorithm 2 computes the inverse of f modulo x^{2^r}. Since x^ℓ divides x^{2^r}, the result is also valid modulo x^ℓ. Before proving the complexity result, we point out the following relation for $i = 1 \cdots r$.

$$g_i \equiv g_{i-1} \mod x^{2^{i-1}} \tag{17}$$

Indeed, by virtue of Theorem 1 we have

$$\begin{array}{rcll} g_i & \equiv & 2g_{i-1} - f\, {g_{i-1}}^2 & \mod x^{2^i} \\ & \equiv & 2g_{i-1} - f\, {g_{i-1}}^2 & \mod x^{2^{i-1}} \\ & \equiv & g_{i-1}(2 - f\, g_{i-1}) & \mod x^{2^{i-1}} \\ & \equiv & g_{i-1}(2 - 1) & \mod x^{2^{i-1}} \\ & \equiv & g_{i-1} & \mod x^{2^{i-1}} \end{array} \tag{18}$$

Therefore when computing g_i we only care about powers of x in the range $x^{2^{i-1}} \cdots x^{2^i}$. This says that

- half of the computation of g_r is made during the last iteration of the **for** loop,
- a quater is made when computing g_{r-1} etc.

Now recall that

$$\frac{1}{2} + \frac{1}{4} + \frac{1}{8} + \cdots = 1 \tag{19}$$

So **roughly** the cost of the algorithm is in the order of magnitude of the cost of the last iteration. which consists of

- two multiplications of polynomials with degree less than 2^r,
- a multiplication of a polynomial (with degree less than 2^r) by a constant,
- truncations modulo x^{2^r}
- a subtraction of polynomials with degree less than 2^r.

leading to $2\mathsf{M}(2^r) + O(2^r)$ operations in R. **But** this was not a formal proof, although the principle was correct. Let us give a more formal proof.

The cost for the i-th iteration is

- $\mathsf{M}(2^{i-1})$ for the computation of ${g_{i-1}}^2$,

- $\mathsf{M}(2^i)$ for the product $f\, {g_{i-1}}^2 \bmod x^{2^i}$,
- and then the opposite of the upper half of $f{g_{i-1}}^2$ modulo x^{2^i} (which is the upper half g_i) takes 2^{i-1} operations.

Thus we have $\mathsf{M}(2^i) + \mathsf{M}(2^{i-1}) + 2^{i-1} \leq \frac{3}{2}\,\mathsf{M}(2^i) + 2^{i-1}$, resulting in a total running time:

$$\sum_{1 \leq i \leq r} \frac{3}{2}\,\mathsf{M}(2^i) + 2^{i-1} \leq (\frac{3}{2}\,\mathsf{M}(2^r) + 2^{r-1}) \sum_{1 \leq i \leq r} 2^{i-r} < 3\,\mathsf{M}(2^r) + 2^r = 3\,\mathsf{M}(\ell) + \ell \tag{20}$$

since $2\mathsf{M}(n) \leq \mathsf{M}(2n)$ for all $n \in \mathbb{N}$ $\lhd$

Remark 4 Once again in Algorithm 2 for $i = 1 \cdots r$ we have

$$g_i \;\equiv\; g_{i-1} \mod x^{2^{i-1}} \tag{21}$$

So when implementing Algorithm 2 one should be extremely careful in not recomputing the *low terms* of g_i that come from g_{i-1}.

Remark 5 Algorithm 2 can be adapted to the case where $f(0)$ is a unit different from 1 by initializing g_0 to the inverse of $f(0)$ instead of 1. If $f(0)$ is not a unit, then no inverse of f modulo x^ℓ exists. Indeed $f\,g \equiv 1 \mod x^\ell$ implies $f(0)g(0) = 1$ which says that $f(0)$ is a unit.

Remark 6 Let us take a closer look at the computation of

$$g_{i-1}(2 - f\,g_{i-1}) \mod x^{2^i} \tag{22}$$

in Algorithm 2. Consider first the product $f\,g_{i-1}$. It satisfies:

$$f\,g_{i-1} \;\equiv\; 1 \mod x^{2^{i-1}} \tag{23}$$

Moreover, the polynomials f and g_{i-1} can be seen as polynomials with degrees less than 2^i and 2^{i-1} respectively. Hence, there exist polynomials $S, T \in R[x]$ with degree less than 2^{i-1} such that we have:

$$f\,g_{i-1} = 1 + Tx^{2^{i-1}} + Sx^{2^i}. \tag{24}$$

We are **only** interested in computing T. In order to avoid computing S, let us observe that we have

$$f\, g_{i-1} \;\equiv\; (1+S) + T x^{2^{i-1}} \mod x^{2^i} - 1. \tag{25}$$

In other words, the *upper part* (that is, the terms of degree at least 2^{i-1}) of the convolution product of $f\, g_{i-1}$ gives us **exactly** T.

So let us assume from now on that we have at hand a primitive 2^i-th root of unity, such that we can compute DFT's. Therefore, we can compute T at the cost of one multiplication in degree less than 2^{i-1}.

Consider now that we have computed $2 - f\, g_{i-1} \mod x^{2^i}$. Viewing $2 - f\, g_{i-1}$ and g_{i-1} as polynomials with degrees less than 2^i and 2^{i-1} respectively, there exist polynomials $U, V, W \in R[x]$ with degree less than 2^{i-1} such that

$$g_{i-1}(2 - f\, g_{i-1}) = U + V x^{2^{i-1}} + W x^{2^i} \tag{26}$$

We know that $g_{i-1} \equiv U \mod x^{2^{i-1}}$. Hence, we are **only** interested in computing V. Similarly to the above, we observe that

$$g_{i-1}(2 - f\, g_{i-1}) \;\equiv\; (U+W) + V x^{2^{i-1}} \mod x^{2^i} - 1 \tag{27}$$

Therefore, using DFT, we can compute V at the cost of one multiplication in degree less than 2^{i-1}.

It follows that, in the complexity analysis above (in the proof of Theorem 2) we can replace $\mathsf{M}(2^i) + \mathsf{M}(2^{i-1})$ by $\mathsf{M}(2^{i-1}) + \mathsf{M}(2^{i-1})$ leading to $2\,\mathsf{M}(\ell) + O(\ell)$ instead of $3\,\mathsf{M}(\ell) + O(\ell)$.

1.4 Division with remainder using Newton iteration

Algorithm 3

Input: $a, b \in R[x]$ with $b \neq 0$ monic.

Output: q, r such that $a = b\,q + r$ and $\deg r \ < \ \deg b$.

$n := \deg a$
$m := \deg b$
if $n < m$ **then**
  $q := 0$
  $r := a$
else
  $f := \mathrm{rev}_m(b)$
  g inverse of f modulo x^{n-m+1}
  $q := \mathrm{rev}_n(a)\,g \mod x^{n-m+1}$
  $q := \mathrm{rev}_{n-m}(q)$
  $r := a - b\,q$
return (q, r)

Theorem 3 *Let R be a commutative ring with identity element. Let a and b be univariate polynomials over R with respective degrees n and m such that $n \geq m \geq 0$ and $b \neq 0$ monic. Algorithm 3 computes the quotient and the remainder of a w.r.t. b in $4\,\mathbf{M}(n-m) + \mathbf{M}(\max(n-m, m)) + \mathcal{O}(n)$ operations in R*

Proof $\triangleright$ Indeed, Algorithm 3 consists essentially in

- 3 multiplications in degree $n - m + 1$, plus operations in $O(n - m + 1)$ operations in R, in order to compute g (by virtue of Theorem 1),
- one multiplication in degree $n - m + 1$ to compute q,
- one multiplication in degree $\max(n - m, n)$, and one subtraction in degree n to compute r.

$\triangleleft$

Remark 7 If several divisions by a given b needs to be performed then we may precompute the inverse of $\mathrm{rev}_m(b)$ modulo some powers $x, x^2, \ldots$ of x. Assuming that R possesses suitable primitive roots of unity, we can also save their DTF.

Remark 8 In Theorem 3, the complexity estimate becomes $3\,\mathbf{M}(n-m) + \mathbf{M}(\max(n-m,n)) + \mathcal{O}(n)$ if the *middle product* technique described in Remark 6 applies. Moreover, if $n - m \leq n$ holds, the $\mathcal{O}(n)$ can be repalced by $\mathcal{O}(m)$.

1.5 Fast Extended Euclidean Algorithm

In this section, we assume that R is a field. Algorithm 6 has the same specifications as the EEA and runs in $33\,\mathsf{M}(d)\log(d) + O(d\,\log(d))$ operations $(+, \times, \div)$ in R for input polynomials in degree d.

The key algorithm is Algorithm 4, called the *Half-GCD* algorithm. This algorithm originated in the ideas of Lehmer, Knuth and Schönhage. Though, it has appeared in many books including [AA74, Yap93, GG99] and in many other publications, it is frequently specified incorrectly. This creates an implementation challenge. We adapted Yap's [Yap93]

Algorithm 4

Input: a, $b \in R[x]$, where R is Field, and $n = \deg a \geq \deg b$

Output: A regular 2×2 matrix M which reduces $gcd(a, b)$ to $gcd(c, d)$ where $\deg c \geq (\deg a)/2 > \deg d$ and $M \left(\begin{smallmatrix} a \\ b \end{smallmatrix}\right) = \left(\begin{smallmatrix} c \\ d \end{smallmatrix}\right)$.

M_{hgcd}(a,b) $==$

1 $\quad d := \deg(a);\ m := \lceil \frac{d}{2} \rceil;$

2 $\quad$ **if** $\deg(b) < m$ **then return** $\left(\begin{smallmatrix} 1 & 0 \\ 0 & 1 \end{smallmatrix}\right)$

3 $\quad a^{\uparrow} := a \text{ quo } x^m$

4 $\quad b^{\uparrow} := b \text{ quo } x^m$

5 $\quad M_1 := \mathrm{M}_{\text{hgcd}}(a^{\uparrow}, b^{\uparrow})$

6 $\quad \left(\begin{smallmatrix} t \\ s \end{smallmatrix}\right) := M_1 \left(\begin{smallmatrix} a \\ b \end{smallmatrix}\right)$

7 $\quad$ **if** $s = 0$ **then return** M_1

8 $\quad (q, r) := \text{QuotientRemainder}(t, s)$

9 $\quad$ **if** $r = 0$ **then**

9.1 $\qquad M_2 := \left(\begin{smallmatrix} 0 & 1 \\ 1 & -q \end{smallmatrix}\right)$

9.2 $\qquad$ **return** $M_2 M_1$

10 $\quad v := \text{LeadingCoefficient}(r)^{-1}$

11 $\quad \overline{r} := rv$

12 $\quad M_2 := \left(\begin{smallmatrix} 0 & 1 \\ v & -vq \end{smallmatrix}\right)$

13 $\quad \ell := 2m - \deg(s)$

14 $\quad s^{\uparrow} := s \text{ quo } x^{\ell}$

15 $\quad r^{\uparrow} := \overline{r} \text{ quo } x^{\ell}$

16 $\quad M_3 := \mathrm{M}_{\text{hgcd}}(s^{\uparrow}, r^{\uparrow})$

17 $\quad$ **return** $M_3 M_2 M_1$

Algorithm 5

INPUT: $a,\ b \in R[x]$, where R is Field, and $\deg a \geq \deg b$

OUTPUT: A regular 2×2 matrix M such that $M \left(\begin{smallmatrix} a \\ b \end{smallmatrix}\right) = \left(\begin{smallmatrix} \gcd(a,b) \\ 0 \end{smallmatrix}\right)$ and M_{11}, M_{12} equal the Bézout coefficients s and t such that $sa + tb = \gcd(a, b)$.

$\mathrm{M_{gcd}}$(a,b) $==$
$M_1 := \mathrm{M_{hgcd}}(a, b);$
$\left(\begin{smallmatrix} t \\ s \end{smallmatrix}\right) := M_1 \left(\begin{smallmatrix} a \\ b \end{smallmatrix}\right)$
if $s = 0$ **then return** M_1
$(q, r) :=$ QuotientRemainder(t, s)
if $r = 0$ **then**
$M_2 := \left(\begin{smallmatrix} 0 & 1 \\ 1 & -q \end{smallmatrix}\right)$
return $M_2 \times M_1$
$v :=$ LeadingCoefficient$(r)^{-1}$
$M_2 := \left(\begin{smallmatrix} 0 & 1 \\ v & -vq \end{smallmatrix}\right)$
$\overline{r} := rv$
$M_3 := \mathrm{M_{gcd}}(s, \overline{r})$
return $M_3\, M_2\, M_1$

Algorithm 6

INPUT: $a,\ b \in R[x]$, where R is Field, and $\deg a \geq \deg b$

OUTPUT: $gcd(a, b)$ and the Bézout coefficients s and t satisfying $sa + tb = gcd(a, b)$.

FEEA(a, b) $==$
if $\deg a > \deg b$ **then**
$M := M_{gcd}(a, b);$
$g := M_{11}\, a + M_{12}\, b;$
return $(g, M_{12}, M_{11});$
else if $\deg a = \deg b$ **then**
return FEEA$(b, a \bmod b)$
else return FEEA(b, a)

References

[AA74] J.D. Ullman A.V. Aho, J.E. Hopcroft. *Design and Analysis of Computer Algorithms*. Addison-Wesley, Reading, Massachusetts, USA, 1974.

[BL82] Bruno Buchberger and Rüdiger Georg Konrad Loos. *Algebraic Simplification*, pages 11–43. Springer Verlag, Wien–New York, 1982.

[CK91] D.G. Cantor and E. Kaltofen. On fast multiplication of polynomials over arbitrary algebras. *Acta Informatica*, 28:693–701, 1991.

[DST93] J. Davenport, Y. Siret, and E. Tournier. *Calcul Formel*. Masson, 1993. 2^{e} édition révisée.

[GCL92] K.O. Geddes, S.R. Czapor, and G. Labahn. *Algorithms for Computer Algebra*. Kluwer Academic Publishers, 1992.

[GG99] J. von zur Gathen and J. Gerhard. *Modern Computer Algebra*. Cambridge University Press, 1999.

[Yap93] C.K. Yap. *Fundamental Problems in Algorithmic Algebra*. Princeton University Press, 1993.