Direct Division in Factor Rings

Patrick Fitzpatrick
p.fitzpatrick@ucc.ie

Department of Mathematics
University College Cork
Ireland

Christopher Wolf*
Christopher.Wolf@esat.kuleuven.ac.be
chris@Christopher-Wolf.de
ESAT-COSIC
Katholieke Universiteit Leuven
Belgium

Date: 2004-12-13

This is an extended version. The bibliographical details of the original article are *P. Fitzpatrick and C. Wolf*: "Direct division in factor rings," Electronic Letters 38 No. 21 (2002), pp 1253-1254.

Abstract

Conventional techniques for division in the polynomial factor ring $\mathbb{F}[z]/\langle m \rangle$ or the integer ring $\mathbb{Z}_n$ use a combination of inversion and multiplication. We present a new algorithm that computes the division directly and therefore eliminates the multiplication step. The algorithm requires $2\,\mathrm{degree}(m)$ (resp. $2\log_2 n$) steps, each of which uses only shift and multiply-subtract operations.

1 Introduction

The development of fast division algorithms for polynomials and integer rings is motivated by a number of applications in coding theory and cryptography (see, for example, [BSS99, Chapter 1]). In particular, computations on elliptic curves need division in finite fields [MOV96], and also newer schemes like Hidden Field Equations, *e.g.*, see [WP04]. In a previous paper [PF98] an algorithm was presented for direct division in the finite field GF(2^k). Alternative algorithms are given in [PF03] and [**?**]. We generalise the latter to arbitrary factor rings $\mathbb{F}[z]/\langle m \rangle$ and to $\mathbb{Z}_n$, for odd n.

*At the time of writing the original article also with University College Cork

Let $\mathbb{F}$ be a field, not necessarily finite, and let $\mathbb{F}[z]$ be the ring of polynomials with coefficients in $\mathbb{F}$. We assume that we have effective algorithms for arithmetic operations in $\mathbb{F}$ (which is of course the case when $\mathbb{F} = \mathrm{GF}(2)$). We use ∂a for the degree of the polynomial a, and $[a]_i$ for the coefficient of z^i in $a \in \mathbb{F}[z]$. The zero polynomial has degree $\partial 0 = -1$. The greatest common divisor of polynomials f, g is denoted $\gcd(f, g)$. Let $m \in \mathbb{F}[z]$ be a fixed polynomial, with $[m]_0 \neq 0$, where we may assume $[m]_0 = 1$. The ideal generated by m is denoted $\langle m \rangle$. If m is irreducible then $\mathbb{F}[z]/\langle m \rangle$ is a field, otherwise it is a ring with zero divisors. This situation specialises to that treated in [ShC01] on taking $\mathbb{F} = \mathrm{GF}(2)$ and m an irreducible polynomial of degree k. Let f, g denote polynomials with $\partial f, \partial g < \partial m$ and $\gcd(g, m) = 1$. This permits us to view division of f by g in $\mathbb{F}[z]/\langle m \rangle$ as solving for q (the quotient) the congruence $f \equiv qg \bmod m$ which gives $f/g \equiv q \bmod m$.

2 Algorithm for polynomials

We observe that the nonempty subset $S \subseteq A = \mathbb{F}[z] \times \mathbb{F}[z]$ of pairs (a, b) that satisfy $af \equiv bg \bmod m$ forms an $\mathbb{F}[z]-$submodule since it is closed under subtraction, defined componentwise, and $\mathbb{F}[z]-$multiplication defined by $c(a, b) = (ca, cb)$. The following lemma is the key to the algorithm.

Lemma 2.1 *The subset $\{(g, f), (m, 0), (0, m)\}$ is a basis of S.*

PROOF. First observe that $(g, f), (m, 0), (0, m) \in S$. Since $\gcd(g, m) = 1$ there exist $r, s \in \mathbb{F}[z]$ such that $rg + sm = 1$. Suppose $(c, d) \in S$. Then $crg + csm = c$ and $cf \equiv dg \bmod m$ which implies $crf = drg + em$ for some $e \in \mathbb{F}[z]$. Together, these equations allow us to express (c, d) in the form $(c, d) = cr(g, f) + cs(m, 0) + (sd - e)(0, m)$. □

Note that the elements of such a basis are only defined up to constant multiples.

Informally, the main idea of the algorithm is to start with the basis given by $\{(g, f), (m, 0), (0, m)\}$ and to convert it into the basis $\{(1, q), (u, v), (0, m)\}$, where q, u, v are to be determined, by a sequence of steps each of which reduces the first component of one of the first two basis elements by a factor of z. Let $B = \{(a_1, b_1), (a_2, b_2), (0, m)\}$ be an intermediate basis and suppose that $\partial a_1 \geq \partial a_2$. We subtract a multiple of a_2 from a_1 to eliminate the constant term of a_1 and then divide by z. To keep the second component correct we subtract the same multiple of b_2 from b_1. We also subtract a suitable multiple of m in order to eliminate the constant term and then divide the

Input: $f, g, m \in \mathbb{F}[z], (g, m) = 1, [m]_0 = 1$
Output: $q \equiv fg^{-1} \bmod m$

DirectDivisionPolynomial (f, g)
$a_1 \leftarrow g, b_1 \leftarrow f, a_2 \leftarrow m, b_2 \leftarrow 0, i \leftarrow 1, j \leftarrow 2;$
while $\partial a_1 > 0$ **and** $\partial a_2 > 0$ **do**
  if $\partial a_j > \partial a_i$ **then** $i \leftrightarrow j$**;**
  if $[a_j]_0 = 0$ **then** $i \leftrightarrow j$**;**
  $b_i \leftarrow \frac{1}{z}\left(b_i - \frac{[a_i]_0}{[a_j]_0} b_j - \left([b_i]_0 - \frac{[a_i]_0}{[a_j]_0}[b_j]_0\right) m\right);$
  $a_i \leftarrow \frac{1}{z}\left(a_i - \frac{[a_i]_0}{[a_j]_0} a_j\right);$
done
return $b_i/[a_i]_0$**;**

Figure 1: Division Algorithm for Polynomials

second component by z. The property of being a basis is preserved at each step, and since the degrees of the first components are reducing, it follows that a constant multiple of $(1, q)$ must appear eventually. The algorithm is as shown in Figure 1, and the next result gives a formal proof of correctness.

Theorem 2.2 *After each iteration the following properties hold*

(i) $B = \{(a_1, b_1), (a_2, b_2), (0, m)\}$ *is a basis of* S

(ii) $\gcd(a_1, a_2) = 1$.

After at most $2\partial m$ *iterations the basis contains a constant multiple of* $(1, q)$.

PROOF. It is clear that at each iteration $B \subseteq S$, so to prove the first assertion we need only check that an arbitrary element $(c, d) = r(a_i, b_i) + s(a_j, b_j) + t(0, m)$ is expressible in terms of the new basis B', where we use the notation of the algorithm for indices i, j and dashes to represent updated values. The following equation gives the required expressions:

$$(c, d) = rz(a_i', b_i') + \left(s + r\frac{[a_i']_0}{[a_j']_0}\right)(a_j', b_j') + \left(t + r[b_i']_0 - r\frac{[a_i']_0}{[a_j']_0}[b_j']_0\right)(0, m).$$

Next, $\gcd(a_1, a_2) = \gcd(g, m) = 1$ initially, and it is clear that $\gcd(a_1', a_2') = \gcd(a_1, a_2)$. Finally, by virtue of the reducing sum of degrees $\partial a_1 + \partial a_2$ we must eventually obtain $a_i' = 0$. At that point a_j' is a non-zero constant, otherwise $\gcd(a_1, a_2)$ would have positive degree. It follows that the algorithm reaches a basis containing an element whose first component is a non-zero constant, and we can make this the stopping criterion. The degree conditions imply that this element is a constant multiple of the uniquely defined element $(1, q)$, since there is no other element $(a, b) \in S$ with $a = 1, \partial b < m$. It takes at most $\partial m + 1$ iterations to reduce ∂m to zero and at most ∂m iterations to reduce ∂g to zero. Therefore the stopping criterion is satisfied after at most $2\partial m$ iterations. $\square$

```
Input:   X(t), Y(t)
Output:  R(t) ≡ X(t) · Y^{-1}(t) (mod M(t))

DirectDivisionUnrolled (X, Y)
        A←Y, B←M, U ←X, V←0
        while ∂A > 0 and ∂B > 0 do
         if a_0 = 0 then
          A←A/z;
          if u_0 = 0 then U←U/z
           else U←[U(t) − (u_0/m_0) · M(t)]/z;
         elif b_0 = 0 then
          B←B/z;
          if v_0 = 0 then V←V/z
           else V←(V(t) − (v_0/m_0) · M(t))/z;
         elif ∂A > ∂B then
          U←U(t) − (a_0/b_0) · V(t);
          if u_0 = 0 then U←U/z
           else U←[U − (u_0/m_0) · M(t)]/z;
          A←[A(t) − (a_0/b_0) · B(t)]/z;
         else
          V←V(t) − (b_0/a_0) · U(t);
          if v_0 = 0 then V←V/z
           else V←[V − (v_0/m_0) · M(t)]/z;
          B←[B(t) − (b_0/a_0) · A(t)]/z;
         endif
        done
        if ∂A = 0 then return (1/a_0) · U(t);
        return (1/b_0) · V(t);
```

Figure 2: Division Algorithm for Polynomials (unrolled)

For efficiency, it is also possible to "unroll" the above algorithm, see Figure 2 for details. In particular, [ShC01] noted that such an algorithm is more efficient than the one from Figure 1. However, the proofs about correctness and running time from this section still apply.

Input: $X(t), Y(t)$
Output: $R(t) \equiv X(t) \cdot Y^{-1}(t) \pmod{M(t)}$

DirectDivisionShort (X, Y)
    $A \leftarrow Y$, $B \leftarrow M$, $U \leftarrow X$, $V \leftarrow 0$
    while (true)
      if $a_0 = 0$
        $A \leftarrow A/z$;
        $U \leftarrow [U(t) - (u_0/m_0) \cdot M(t)]/z$;
      elif $\partial A = 0$ **then return** $(1/a_0) \cdot U(t)$**;**
      elif $\partial A < \partial B$ **then** $A \leftrightarrow B$**,** $U \leftrightarrow V$**;**
      else
        $U \leftarrow U(t) - (a_0/b_0) \cdot V(t)$;
        $A \leftarrow A(t) - (a_0/b_0) \cdot B(t)$;
      endif
    done

Figure 3: Division Algorithm for Polynomials (code-size efficient)

When not speed but code-size is the issue, the algorithm from Figure 3 seems to be the best option. All computations are done in one single loop — and there is no extra memory requirement when compared with the other implementations. However, we want to stress that all these algorithms have been implemented in software and a hardware implementation may use different optimisations.

3 Algorithm for $\mathbb{Z}_n$

Input: $r, s \in \mathbb{Z}_n, (s, n) = 1$
Output: $q \equiv rs^{-1} \bmod n$

DirectDivisionShortIntegers (r, s)
$a_1 \leftarrow s, b_1 \leftarrow r, a_2 \leftarrow n, b_2 \leftarrow 0, i \leftarrow 1, j \leftarrow 2;$
while $(a_i > 1)$
  if $a_i < a_j$ **then** $i \leftrightarrow j$**;**
  if $[a_j]_0 = 0$ **then** $i \leftrightarrow j$**;**
  $b_i \leftarrow (b_i - b_j[a_i]_0)/2;$
  if $b_i < 0$ **then** $b_i \leftarrow b_i + n$**;**
  $a_i \leftarrow (a_i - a_j[a_i]_0)/2;$
done
return b_i;

Figure 4: Short Algorithm for Division in $\mathbb{Z}_n$

This algorithm can be adapted for $\mathbb{Z}_n$, n odd. In this section we present this algorithm, leaving to the reader the precise details of the proof of correctness. We express all integers in base 2 and denote by $[a_1]_0$ the least significant bit of the integer a_1. The algorithm is given in Figure 4.

By an argument similar to that in Theorem 2 the number of iterations at most $2\log_2 n$.

4 Conclusions

The algorithm presented in this paper can be used to compute $fg^{-1} \pmod m$ for $f, g, m \in \mathbb{F}[z]$, where $[m]_0 \neq 0, \gcd(g, m) = 1$, and $rs^{-1} \pmod n$ for $r, s, n \in \mathbb{Z}$, where n is odd, and $\gcd(s, n) = 1$. The division is carried out directly rather than as a combination of inversion and multiplication. Its complexity is $2\partial m$ (resp. $2\log_2 n$). In contrast, division based on the extended Euclidean algorithm has the same complexity in computing only the inverse of g or s, and thereafter an additional multiplication step is needed.

References

[BSS99] I. BLAKE, G. SEROUSSI, N. SMART: *Elliptic Curves in Cryptography*, Cambridge University Press, 1999.

[PF98] E. M. POPOVICI, P. FITZPATRICK, 'Division algorithm over GF(2^m)', *Elect. Lett.* 34:19, 1998, 1843–1844.

[MOV96] ALFRED J. MENEZES and PAUL C. VAN OORSCHOT and SCOTT A. VANSTONE: *Handbook of Applied Cryptography*, CRC Press, 1996, ISBN 0-8493-8523-7. `http://www.cacr.math.uwaterloo.ca/hac/`

[PF03] E. POPOVICI and P. FITZPATRICK: *Algorithm and architecture for a multiplicative Galois field processor*, IEEE Trans. Inform. Thy, 49 (2003), 3303-3307.

[ShC01] SHEULING CHANG SHANTZ: 'From Euclid's GCD to Montgomery multiplication to the great divide', *Sun Microsystems, SML Technical Report*, SMLI TR-2001-95, 2001. `http://research.sun.com/research/techrep/2001/abstract-95.html`

[WP04] CHRISTOPHER WOLF and BART PRENEEL. Asymmetric Cryptography: Hidden Field Equations. In *European Congress on Computational Methods in Applied Sciences and Engineering 2004*. P. Neittaanmäki, T. Rossi, S. Korotov, E. Oñate, J. Périaux, and D. Knörzer, editors, Jyväskylä University, 2004. 20 pages, extended version: `http://eprint.iacr.org/2004/072/`.