EFFICIENT DIVISION ALGORITHMS IN EUCLIDEAN DOMAINS

R. Moenck
and
J. Allen

Research Report CS-75-18

Department of Computer Science

University of Waterloo
Waterloo, Ontario, Canada
July 1975

This research was supported by NRC Grant No. A-5549.

ABSTRACT

It is shown that efficient division algorithms can be stated and analyzed in the general algebraic setting of a euclidean domain instead of for a particular domain as is the usual case. The general division problem is reduced to that of computing a "psuedo-inverse" of the divisor in the domain. An algorithm is given for computing psuedo-inverses and this yields the result that the number of steps used to perform a division is proportional to that needed to multiply two elements of similar size in the domain. As corollaries this gives Cook's $O(n \log n \log \log n)$ step integer division algorithm and a $O(n \log n)$ polynomial division algorithm due to Strassen.

I. INTRODUCTION

The classical long division method uses $2(m-n+1)(n+1) = O(mn)$ arithmetic operations to divide a polynomial of degree m by one of degree n. A similar bound holds for the long division method applied to integers with m and n digits respectively. While this algorithm is adequate for the calculations one might perform using pencil and paper, if we were solving very large problems using a computer we would be interested in methods which use less operations.

We can contrast the situation for division with that for multiplication. The classical multiplication algorithms use $O(n^2)$ arithmetic operations. However, it is known that polynomials over a ring which supports a fast fourier transform (FFT) can be multiplied in O(n log n) arithmetic operations. Schönhage and Strassen [SS71] used this FFT approach when producing their integer multiplication algorithm which works in O(n log n log log n) operations.

For division Cook [Co 66] showed how to consider the integer division problem as one of Newton iteration. It follows from this that integer division is bounded by the successive multiplications used in the iteration and so may be performed in the O(n log n log log n) time bound. Fast polynomial division does not appear to have been considered until Moenck and Borodin [MB72] exhibited on $O(n \log^2 n)$ algorithm based on the FFT multiplication algorithm. Shortly afterward, Strassen [Str 73] and Schönhage [Sch 73] independently developed an O(n log n) algorithm. The O(n log n) division algorithm is based on an algorithm for expanding the reciprocal of a power series (cf [Sie 72]). Kung [Ku 73b] later showed, this is essentially a form of Newton iteration.

The reason for this interest in the efficiency of division is partially due to the result by Borodin and Moenck [Bor 74] that the efficiency of multipoint evaluation and the Chinese Remainder Algorithm in an EUD is log reducible to that of division in the domain.

In this paper we show that it is not necessary to use the powerseries over the domain in which one wishes to perform division, nor are the associated notions of approximation explicitly needed. Instead one can attack the problem in the general algebraic setting of an Euclidean Domain [EUD] and perform all the computation in that domain. Instead of using the machinery of approximation we can compare the relative magnitude of elements of the domain with respect to the domain valuation. Even in this very much weaker framework we can show that the number of steps used for division in an EUD is proportional to that for multiplication of similar sized elements in the domain. As corollaries we can derive Cook's integer division result and the fast polynomial division algorithm.

One of the aims of this paper is to illustrate the fact that efficient algorithms may be stated and analyzed in a general algebraic setting and need not repeatedly be specified for particular domains. For further examples of this claim see [Moe 73].

Before we begin our discussion we must specify our model of computation. We shall use a random access machine (cf Cook [Co 72]) with a storage access cost function of unity. Although Cook's model does not provide for single precision multiplication as a primitive operation it is otherwise well suited to our purposes.

The measure of efficiency of an algorithm will be the number of register-register arithmetic instructions executed on variables or constants. A unit cost for each operation will be used for all domains. The operations are assumed to yield exact results. Branching, looping and indexing instructions while conceptually and practically useful will be neglected. The rationale for this is that their number can only change in proportion to the nontrivial arithmetic instructions. We will further refine the analysis by considering two possibilities. First we will have a total operations measure which counts all arithmetic operations. Alternatively we will use a measure first proposed by Ostrowski [Ost 54] and used by many other authors, where we count only non-scalar multiplications or divisions.

We must also specify the data structures involved in storing elements of an EUD. We will assume that polynomials will be stored as a dense array of coefficients with one coefficient per register. Integers will be considered to be in base β representation (possibly $\beta=2$) and each β digit will be stored in one machine register. Elements from other domains will be stored analogously. In this way a storage scheme similar to that for a finite word length machine is obtained.

II. Some Definitions

As explained earlier one of the aims of the paper is to formulate fast division algorithms in the general setting of a Euclidean Domain. We can use the familiar:

Definition 1: A Euclidean Domain E is an integral domain with a degree function (valuation)

$$\partial : E^* \to Z^+$$

where $E^* = E - \{0\}$ and Z^+ is the set of positive integers. (We will also use certain proper subsets of the positive real numbers as the range of the degree function. These subsets will be well ordered by their isomorphism with the non-negative integers), the degree function U satisfies

1) $\partial(A * B) \geq \partial(A), \quad A, B \in E^*$

2) E has a division algorithm

ie. $\forall U, V \in E^*, \ \exists Q, R \in E$ such that

$$U = V * Q + R$$

with $\partial(R) < \partial(V)$ or $R = 0$

This is the algorithm we wish to relate to the efficiency of multiplication in the domain. A basic result from ring theory is:

Lemma 1: Under the conditions:

$$\partial(Q) \leq \partial(U) - \partial(V) \tag{2.1}$$

$$\partial(R) < \partial(V) \tag{2.2}$$

Q and R are unique for given $U, V \in E$.

In order to standardize the notation for the timing function of the various domains we will make:

Assumption 1:

$$A, B \in E^* , \qquad \partial(A) \geq \partial(B)$$

$$\left.\begin{aligned} &1) \quad \partial(A * B) = \partial(A) + \partial(B) \\ &2) \quad \partial(A+B) \leq \partial(2A) = \partial(2) + \partial(A) \end{aligned}\right\} \qquad (2.3)$$

or in general terms:

$$\partial(A+B) \leq \max(\partial(A), \partial(B)) + \xi$$

where $\xi = \partial(2)$.

If $\xi \neq 0$, and $\partial(A) \neq \partial(B)$ then we have a strict inequality.

$$\partial(A+B) < \max(\partial(A), \partial(B)) + \xi$$

we assume $\xi = \partial(2) < 1$. [$\xi = 1$ can be dealt with in a straightforward manner as a special case]

eg. For three particular domains of interest:

i) $F[x]$ - the polynomials over a field F : $\partial(A) = \deg(A)$

$$\xi = \partial(2) = 0.$$

ii) $\mathbb{Z}$ - the integers : $\partial(A) = \log_\beta |A|$, $\beta > 2$

so $\xi = \partial(2) = \log_\beta 2 < 1$. (If B=2, ξ=1)

iii) $\mathbb{Z}[i]$ - the gaussian integers : $\partial(A) = \log_\beta |A|$,

$\beta > 2$ where $|a+bi| = \sqrt{a^2+b^2}$ and

$$\xi = \partial(2) = \log_\beta |2| = \log_\beta 2 < 1.$$

It should be pointed out that this restricts somewhat the possible domains which can be considered. For example fields fall within the scope of the definition of EUD's given above, if $A, B \in E^*$

$$\partial(A) = 1 \text{ and } \partial(A*B) = \partial(A) * \partial(B).$$

However such properties violate the basic assumption of computational complexity that some elements of the domain must be harder to compute with than others. The degree properties (2.3) are acceptable for all the

conventional EUD's.

Assumption 2:

We shall use $\partial(A)$ both as a valuation of E and as a measure of storage cost of an element of E. In fact a non-zero element will be assumed to occupy $\lceil \partial(A) \rceil + C$ units of storage, for some small positive integer constant C. This is owing to the assumptions above that storage is allocated in integral units. Zero will occupy C storage units. It is this integral storage cost which will be used as a parameter of the timing function of the algorithms.

This assumption implicitly restricts the possible representations of an element. For example the unary representation of integers is excluded. However the possible representations conform to those which are intuitively the correct ones from the complexity point of view.

In addition we shall define a function D(N) which generates a canonical element of the domain for a given N.

Definition 2: Given a Euclidean Domain E, we define the function $D : Z^+ \to E$ (where Z^+ are the positive integers) such that

$$\forall N \in Z^+ , \qquad \exists a_N \in E \text{ where}$$

$$D(N) = a_N \text{ and } \partial(a_N) = N \tag{2.4}$$

Given D(0) and D(1), D(N) can be defined inductively by:

$$D(N_1) * D(N_2) = D(N_1 + N_2) \quad .$$

We can think of D(N) generating a basis set for E. For the three example domains:

i) If $D = F[x]$, $D(N) = x^N$

ii) If $D = Z$, $D(N) = \beta^N$

iii) If $D = \mathbb{Z}[i]$ $\quad D(N) = (\beta i)^N$

(cf. Knu 69] p 171).

Finally we introduce some notation. $[V]_k$ can be thought of as the leading degree k portion of an element V or E with the appropriate basis element factored out.

Definition 3: $[V]_k = \dfrac{V - V \bmod (D(\lfloor n-k \rfloor))}{D(\lfloor n-k \rfloor)}$

where $\partial(V) = n$.

eg. if $U(x) = x^3 + 2x^2 + 3x + 4$

then $[U]_1 = x + 2$.

But n and k are not necessarily integers.

We want to develop bounds on $\partial([V]_k)$. We may assume $n > k > \xi$ without loss of generality. This is because $n > k$ is a necessary condition for $[V]_k$ to be defined, and the cases $n < \xi$ or $k < \xi$ will not affect the asymptotic complexity of the division algorithm.

We will show

$$n - \lfloor n-k \rfloor - \xi \leq \partial([V]_k) \leq n - \lfloor n-k \rfloor$$

Upper bound:

From def (3) we see $V = [V]_k \cdot D(\lfloor n-k \rfloor) + V \bmod (D(\lfloor n-k \rfloor))$

Using 2.1 we get $\partial([V]_k) \leq \partial(V) - \partial(D(\lfloor n-k \rfloor)) = n - \lfloor n-k \rfloor$ ... (2.5)

Lower bound:

First we will develop a lower bound for $\partial([V]_k+1)$

From def. (3) we get

$$\partial(V) = \partial([V]_k \cdot D(\lfloor n-k \rfloor) + V \bmod (D(\lfloor n-k \rfloor)))$$

$$\leq \partial([V]_k \cdot D(\lfloor n-k \rfloor) + D(\lfloor n-k \rfloor)) \text{ since } \partial(V \bmod (D(\lfloor n-k \rfloor)) < \partial(D([n-k]))$$

$$= \partial(([V]_k + 1) \cdot D(\lfloor n-k \rfloor))$$

Therefore $\partial([V]_k + 1) \geq \partial(V) - \partial(D(\lfloor n-k \rfloor))$

$$= n - \lfloor n-k \rfloor$$

Consider $\partial([V]_k)$

if $\partial([V]_k) < 1$, then $\xi = \partial(2) > \partial([V]_k + 1) = n - \lfloor n-k \rfloor \geq k$

this implies $\xi > k$ which was eliminated above.

Therefore we may assume $\partial([V]_k) \geq 1$

To obtain the upper bound we consider

$$\partial(2 \cdot [V]_k) \geq \partial([V]_k + 1) \text{ since } \partial([V]_k) \geq 1$$

$$\Longrightarrow \quad \partial(2) + \partial([V]_k) \geq n - \lfloor n-k \rfloor$$

$$\Longrightarrow \quad \partial([V]_k) \geq n - \lfloor n-k \rfloor - \xi \,, \quad \xi = \partial(2)$$

Combining this with 2.5 we get

$$n - \lfloor n-k \rfloor \geq \partial([V]_k) \geq n - \lfloor n-k \rfloor - \xi \ldots \qquad (2.6)$$

III. Division in the Domain

First we establish a useful lemma which shows that we need only know a limited amount about the divisor and dividend to determine the quotient in a division.

Lemma 2: Consider the division $U = Q \cdot V + R$ in E. The quotient Q ($\partial(Q) \leq k = \partial(U) - \partial(V)$) is "almost determined" by $[V]_{k+2\xi}$ and $[U]_{2k+2\xi}$.

ie. if $[U]_{2k+2\xi} = Q' \cdot [V]_{k+2\xi} + R'$

then $U = Q' \cdot V + R''$, with $\partial(R'') < \partial(V) + \xi$.

eg. In $F[x]$, $\xi = 0$ so we have determined Q'.

In $\mathbb{Z}$, Q' is at most one from the true quotient

ie. the remainder is less than $2 \cdot V$.

Proof:

Let $\partial(U) = m$, $\partial(V) = n$, $\partial(Q) \leq k = m-n$,

remembering that n,m and k are not necessarily in $\mathbb{N}$.

Notice that:

$$m-2k = m-(m-n) -k = n-k \quad \ldots \quad (3.1)$$

Now by definition:

$$U = [U]_{2k+2\xi} D(\lfloor m-2k-2\xi \rfloor) + U' \quad \ldots \quad (3.2)$$

$$V = [V]_{k+2\xi} D(\lfloor n-k-2\xi \rfloor) + V' \quad \ldots \quad (3.3)$$

Let Q' be the quotient of $[U]_{2k+2\xi}$ and $[V]_{k+2\xi}$

ie. $[U]_{2k+2\xi} = Q' \cdot [V]_{k+2\xi} + R'$.

So $\partial(Q') \leq \partial([U]_{2k+2\xi}) - \partial([V]_{k+2\xi})$ by condition 2.1

$$\leq m - \lfloor m-2k-2\xi \rfloor - (n-\lfloor n-k-2\xi \rfloor -\xi)$$

using inequality (2.6) upper bound on $\partial([U]_{2k+2\xi})$ and lower bound on $\partial([V]_{k+2\xi})$.

$$\leq n + k - \lfloor m-2k-2\xi \rfloor - (n-(n-k-2\xi)-\xi)$$

$$= n + k - \lfloor m-2k-2\xi \rfloor - (k+\xi)$$

$$= n - \lfloor m-2k-2\xi \rfloor - \xi \ .$$

So $\partial(Q') \leq n - \lfloor m-2k-2\xi \rfloor - \xi$ ... (3.4)

or $n - \lceil n-k-2\xi \rceil - \xi$ using (3.1). Now consider:

$R'' = U - Q' \cdot V$ if we show $\partial(R'') < \partial(V)+\xi$ then we are done.

$$= (\lceil U \rceil_{2k+2\xi} D(\lfloor m-2k-2\xi \rceil) + U')$$

$$- Q' (\lceil V \rceil_{k+2\xi} D(\lfloor n-k-2\xi \rfloor) + V')$$

from expanding U and V with (3.2) and (3.3).

$$= (\lceil U \rceil_{2k+2\xi} D(\lfloor n-k-2\xi \rfloor) - Q' \lceil V \rceil_{k+2\xi} D(\lfloor n-k-2\xi \rfloor))$$

$$+ (U' - Q'V')$$

$$= (\lceil U \rceil_{2k+2\xi} - Q' \lceil V \rceil_{k+2\xi}) D(\lfloor n-k-2\xi \rfloor)$$

$$+ (U' - Q' \cdot V') \quad \ldots \quad (3.5)$$

Now consider the first term of the addition (3.5)

$$\partial(\lceil U \rceil_{2k+2\xi} - Q' \cdot \lceil V \rceil_{k+2\xi}) = \partial(R')$$

$$< \partial(\lceil V \rceil_{k+2\xi}) \leq n - \lfloor n-k-2\xi \rfloor$$

So $\partial((\lceil U \rceil_{2k+2\xi} - Q' \cdot \lceil V \rceil_{k+2\xi}) D(\lfloor n-k-2\xi \rfloor))$

$$< n - \lfloor n-k-2\xi \rfloor + \lfloor n-k-2\xi \rfloor = n \quad \ldots \quad (3.6)$$

Consider the second term of the addition (3.5): $(U' - Q' V')$

$\partial(U') < \lfloor m-2k-2\xi \rceil = \lfloor n-k-2\xi \rfloor$ from (3.2)

$\partial(V') < \lfloor n-k-2\xi \rfloor$ from (3.3)

$\partial(Q') < n - \lfloor n-k-2\xi \rfloor - \xi$ from (3.4)

Hence $\partial(U' - Q'V') \leq \max(\partial(U'), \partial(Q'V')) + \xi$

$$\leq \max(\lfloor n-k-2\xi \rfloor, n-\xi) + \xi$$

$$= n - \xi + \xi = n \quad (3.7)$$

So now we have:

$$\begin{aligned}
\partial(R'') &= \partial(([U]_{2k+2\xi} - Q'[V]_{k+2\xi})D(\lfloor n-k-2\xi \rfloor) \\
&\quad + (U' - Q'V')) \\
&\leq \max\ (\partial(([U]_{2k+2\xi} - Q' \cdot [V]_{k+2\xi}) \cdot D(\lfloor n-k-2\xi \rfloor)), \\
&\quad \partial(U' - Q'V')) + \xi \\
&< \max\ (n,n) + \xi \qquad \text{from (3.6) and (3.1)} \\
&= n + \xi
\end{aligned}$$

So $\partial(R) < n + \xi$ as required. □

One approach to performing the division is to compute the quotient Q. Knowing the quotient we can compute the remainder in one domain multiplication and one domain addition. From lemma 2 we know that we need only consider $[U]_{2k+2\xi}$ and $[V]_{k+2\xi}$, of any U and V to determine the quotient. Even if we only "almost determine" the quotient then we can correct it in time linear with respect to the storage used by the quotient and so this will not affect any general time bound.

So without loss of generality we will assume U and V are in this form. Translating this condition into a degree relation between U and V we get:

$$2\partial(V) - 2\xi \geq \partial(U) > 2\partial(V) - 2\xi - 1 \qquad \ldots$$

This is derived as follows, let $\partial(V) = n$, $\partial(U) = m$.

Now $U = [U]_{2k+2\xi}$

$$\Rightarrow \quad \frac{U - U \bmod (D(\lfloor m-2k-2\xi \rfloor))}{D(\lfloor m-2k-2\xi \rfloor)} = U$$

$$\Rightarrow \quad \lfloor m-2k-2\xi \rfloor = 0$$

$$\Rightarrow \quad 0 \leq m - 2k - 2\xi < 1$$

$$\Rightarrow \quad 0 \leq m - 2\,(m-n) - 2\xi < 1$$

$\Rightarrow 0 \leq 2n - m - 2\xi < 1$

$\Rightarrow -2n + 2\xi \leq -m < -2n + 2\xi + 1$

$\Rightarrow 2n - 2\xi \geq m > 2n - 2\xi - 1$.

The problem of computing the quotient can be attacked by way of computing a "psuedo-inverse" P of the divisor V. This is because of the following central lemma.

Lemma 3:

If there exists a $P \in E$, $\partial(V) = n$, $2n-2\xi+1 < \partial(U) = m \leq 2n-2\xi$ such that

$$P \cdot V = S = D(\lfloor k \rfloor) + S' \qquad \ldots \qquad (3.8)$$

where $\partial(S') < \lfloor k \rfloor - \partial(V) = \lfloor k \rfloor - n$ for $\lfloor k \rfloor > n$

and $\partial(D(\lfloor k \rfloor) + S') \geq \partial(D(\lfloor 2n \rfloor))$

then $[P \times U]_{m-n}$ is "close" to Q the quotient of U and V.

ie. $U = [P \times U]_{m-n} \cdot V + A$, $\partial(A) \leq \partial(V) + \xi = n + \xi$

Examples

In $Q[x]$, $\xi = \partial(2) = 0$

Let $V = x^2-3$, $U = x^4+3x^3+x$

So $\partial(V) = n = 2$ and $\partial(U) = m = 4$ $(= 2n-2\xi)$

Now if $P = x^2+3$, then $P \cdot V = x^4-9 = D(4)+S'$ where $S' = -9$. So

$$[P \cdot U]_{m-n} = [(x^2+3)(x^4+3x^2+x)]_{4-2}$$

$$= [x^6+3x^5+3x^4+10x^2+3x]_2$$

$$= x^2+3x+3$$

is 'close' to the quotient. In fact, we have calculated the quotient exactly here because $\xi = 0$.

In Z, $\partial(x) = \log_{10} x$

let $V = 17, \quad U = 40$

then $\partial(V) = 1.2304$ and $\partial(U) = 1.6021$. Now if

$P = 59,$

$P \cdot V = 1003 = D(3) + 3, \; S' = 3$

So $[P \cdot V]_{m-n} = [59.40]_{.3716} = [2360]_{.3716}$

$$= \frac{2360 - 2360 \bmod (D(\lfloor 3.3729 - .3716 \rfloor))}{D(\lfloor 3.3729 - .3716 \rfloor)}$$

$$= 2$$

which is the correct quotient.

Proof: Let $\partial(V) = n, \quad \partial(U) = m$

then $\partial(P \cdot V) = \partial(D(\lfloor k \rfloor) + S') \leq \partial(D(\lfloor k \rfloor)) + \xi = \lfloor k \rfloor + \xi$

So $\partial(P) \leq \lfloor k \rfloor - n + \xi \quad \ldots$ (3.9)

Also $\partial(P \cdot V) = \partial(D(\lfloor k \rfloor) + S') \geq \partial(D(\lfloor k \rfloor)) = \lfloor k \rfloor$

So $\partial(P) \geq \lfloor k \rfloor - n \; . \quad \ldots$ (3.10)

We are going to be working with $[P \cdot U]_{m-n}$

So by defn. 3 we want to find the bounds on:

$D(\lfloor \partial(P \cdot U) - (m-n) \rfloor) = D(\lfloor \partial(P) + m - m + n \rfloor)$

$$= D(\lfloor \partial(P) + n \rfloor)$$

So by (3.9) and (3.10): $\lfloor k \rfloor + \xi \geq \partial(P \cdot V) = \partial(P) + \partial(V)$

and so $= \partial(P) + n \geq \lfloor k \rfloor$

$$\lfloor \lfloor k \rfloor + \xi \rfloor \geq \lfloor \partial(P) + n \rfloor \geq \lfloor k \rfloor$$

Therefore $\lfloor \partial(P) + n \rfloor = \lfloor k \rfloor$ since $\xi < 1$.

Consider the potential quotient:

$$[P \cdot U]_{m-n} = \frac{P \cdot U - W}{D(\lfloor \partial(P) + n \rfloor)} = \frac{P \cdot U - W}{D(\lfloor k \rfloor)}$$

where $W \equiv P \cdot U \bmod D(\lfloor \partial(P) + n \rfloor)$

$$\equiv P \cdot U \bmod D(\lfloor k \rfloor) \quad \ldots \quad (3.14)$$

Consider:

$$T = [P \cdot U]_{m-n} \cdot V$$

$$= \frac{P \cdot U \cdot V - W \cdot V}{D(\lfloor k \rfloor)} = \frac{U \cdot D(\lfloor k \rfloor) + U \cdot S' - W \cdot V}{D(\lfloor k \rfloor)} \ldots (3.13)$$

using the hyp. that $P \cdot V = D(\lfloor k \rfloor) + S'$

We want to show: $U \cdot S' - WV = A \cdot D(\lfloor k \rfloor)$

for some A, $\partial(A) < n + \xi$.

$\partial(U \cdot S) = \partial(S') + \partial(U) < \lfloor k \rfloor - n + m = \lfloor k \rfloor + (m-n)$

$\partial(W \cdot V) = \partial(W) + \partial(V) < \lfloor k \rfloor + n$ ... from (3.14)

This implies: $\partial(S'U - WV) < \max(\lfloor k \rfloor + m-n, \lfloor k \rfloor + n) + \xi$

$= \lfloor k \rfloor + n + \xi$

since $n > m-n$ by hyp.

Now let $S'U - WV = A \cdot D(\lfloor k \rfloor) + B$

but $B = (S'U - WV) \bmod D(\lfloor k \rfloor)$

$= P \cdot V \cdot U \bmod D(\lfloor k \rfloor) - P \cdot U \cdot V \bmod D(\lfloor k \rfloor) = 0$

using defn. of S' (3.8) and W (3.14)

So $S'U - WV = A \cdot D(\lfloor k \rfloor)$

Now $\partial(A \cdot D(\lfloor k \rfloor)) = \partial(S'U - WV) < \lfloor k \rfloor + n + \xi$

$\Rightarrow \partial(A) < \lfloor k \rfloor + n + \xi - \partial(D(\lfloor k \rfloor)) = n + \xi$.

Now consider:

$$[P \cdot U]_{m-n} V = \frac{U \cdot D(\lfloor k \rfloor) + A \cdot D(\lfloor k \rfloor)}{D(\lfloor k \rfloor)}$$

$$= U + A$$

So $U = [P \cdot U]_{m-n} \cdot V + (-A)$

and $\partial(-A) = \partial(A) < n + \xi$ □

This result reduces the general division problem to that of computing a psuedo-inverse P of V. If we can compute P then we can form $[P \cdot U]_{m-n}$ in one domain multiplication and in time linear with respect to $\lfloor m-n \rfloor$ form the correct quotient Q. We multiply Q by V and subtract it from U to get the remainder R. Thus we can compute the remainder given P in two domain multiplications and some operations which are linear with respect to the storage used by the divisor and dividend.

To ease the computation of P we will scale V such that:

$$V' = c\ V,$$

where $\partial(V') = 2^r$, $r = \lceil \log(\partial(V)) \rceil$ and $c \in E$.

In addition we will require that V' is of the form:

$$V' = D(2^r) + v_1\ D(2^r - 1) + \tilde{V}$$

where $0 \leq \partial\ (v_1) < 1$.

We may compute the psuedo-inverse of V'. This will not be very different from that of V and any difference there is will disappear in the computation of $[P \cdot U]_{m-n}$.

Algorithm: Quot (V)

Input: $V \in E$ where $\partial(V) = n$, $r = \lceil \log n \rceil$

Output: P the pseudo-inverse of the scaled V' such that:

$$P \cdot V = S = D(2^{r+1} + \ell) + S'$$

where $\partial(S') < (2^{r+1} + \ell) - 2^r$

and ℓ depends on $v_1 = \ell \geq \frac{1+\partial(v_1)}{1-\partial(v_1)}$.

Steps:

1) Basis: V':= scale (V); find v_1;

choose $\ell > \frac{1 + \partial(v_1)}{1 - \partial(v_1)}$;

$$P_0 := \sum_{i=0}^{\ell} D(i)\ (-v_1)^{\ell-i} ;$$

2) Iteration: for k:=1 until r do

begin $T_k := P_{k-1} * [V']_2 k$;

$R_k := [P_{k-1}(2D(2^{k+1}) - [T_k]_2 k)]_2 k+\ell$

if ξ=0 then $P_k := R_k$

else form $P_k \cdot [V']_2 k$ and if necessary correct P_k

end; □

The algorithm is invoked as P:= Quot (V);

The validity of the algorithm follows from:

Lemma 4: For the Quot algorithm for all $0 \leq k \leq r$

$$R_k [V']_2 k = S_k = D(2^{k+1} + \ell) + S'_k$$

where $\partial(S'_k) \leq 2^k + \ell + 3\ \xi$.

By assumption we can correct R_k to P_k in linear time such that:

$$P_k [V']_2 k = D(2^{k+1} + \ell) + S''_k$$

where $\partial(S''_k) < (2^{k+1} + \ell) - 2^k = 2^k + \ell$.

Proof:

First we note that by definition $\partial(R_k) \leq 2^k + \ell$

Basis: k=0, P_1 is chosen such that:

$$S_0 = P_0\,[V']_1 = \sum_{i=0}^{\ell} D(i)(-v_1)^{\ell-i}(D(1) + v_1)$$

$$= D(\ell+2) - v_1^{\ell+1} \quad .$$

In this case $\partial(S_0') = \partial(v_1^{\ell+2}) = (\ell+2)\partial(v_1)$

but $\ell > (1+\partial(v_1))/(1-\partial(v_1))$ ie: $(\ell+1)\partial(v_1) < \ell$

so $\partial(S_0') < (\ell+1) - 1 = \partial(S) - \partial(S) - \partial([V']_1)$.

so the conditions of the lemma follow.

Induction: we shall assume that the assertion is true up to k. Now we can follow the steps of the algorithm and see what is computed.

$$T_k = P_{k-1}\,[V']_{2^k} = P_{k-1}\,([V']_{2^{k-1}}\,D(2^{k-1}) + \tilde{V})$$

$$= P_{k-1}[V']_{2^{k-1}}\,D(2^{k-1}) + P_{k-1}\,\tilde{V}$$

$$= D(3*2^{k-1} + \ell) + S'_{k-1}\,D(2^{k-1}) + P_{k-1}\,\tilde{V} \text{ by hyp.}$$

But $\partial(S_{k-1}D(2^{k+1})) < 2^{k-1} + 2^{k-1} + \ell = 2^k + \ell$

$\partial(P_{k-1}\tilde{V}) < 2^{k-1} + 2^{k-1} + \ell = 2^k + \ell$.

and so $\partial(S'_{k-1}\,D(2^{k-1}) + P_{k-1}\,\tilde{V}) < 2^k + \ell + \xi$.

This means that T_k may be written as:

$$T_k = D(3\cdot 2^{k-1} + \ell) + T'D(2^{k-1} + \ell) + T'' \ldots \quad (2$$

where $\partial(T') < 2^{k-1} + \xi$ and $T'' \equiv (S'_{k-1}D(2^{k-1}) + P_{k-1}\tilde{V}) \bmod D(2^{k-1} + \ell)$

$$\partial(T'') < 2^{k-1} + \ell \ .$$

So $[T_k]_{2^k} = D(2^k) + T'$.

which means that:

$$R_k = [P_{k-1}\,(D(2^k) - T')]_{2^k+\ell} \ .$$

Consider what happens when we form $[V']_{2^k}\,R_k$

we must show this is of the form $D(2^{k+1}+\ell) + S_k'$.

Now:

$$[V']_{2^k} R_k = [V]_{2^k} [P_{k-1}(D(2^k) - T')]_{2^k+\ell}$$

$$= [[V]_{2^k} \cdot P_{k-1} \cdot (D(2^k) - T')]_{2^{k+1}+\ell}$$

$$= [(D(3\text{-}2^{k-1} + \ell) + T'D(2^{k-1} + \ell) + T'') \cdot (D(2^k - T')]_{2^{k+1}+\ell} \quad \text{by (2)}$$

$$= [D(5 \cdot 2^{k-1} + \ell) + T''D(2^k) - T'^2 D(2^{k-1} + \ell) - T'T'']_{2^{k+1}+\ell}$$

$$= [(D(2^{k+1}+\ell) + W)D(2^{k-1}) + W']_{2^{k+1}+\ell}$$

$$= D(2^{k+1}+\ell) + W$$

where $W = T''D(2^{k-1}) - T'^2 D(\ell) + [T'T'']_{m-2^{k-1}}$ and $m = \partial(T') + \partial(T'')$

$$\partial(W) \leq \max \{\partial(T''D(2^{k-1})), \partial(T'^2 D(\ell)), \partial([T'T'']_{m-2^{k-1}} + \xi\}$$

$$= \max \{(2^{k-1}+\ell) + 2^{k-1},\ 2(2^{k-1}+\xi)+\ell,\ (2^{k-1} + \xi) + (2^{k-1}+\ell) - 2^{k-1} + \xi\}$$

$$= 2^k + \ell + 3\xi \ .$$

Therefore $\partial(S_k') < 2^k + \ell + 3\xi$ □

By this development we see that the Quot algorithm generates a pseudo-inverse P of V. This leads us to the central:

Theorem 5: Division is directly reducible to multiplication in an EUD.

Proof: Let Q(N) be the number of steps required to perform the Quot algorithm on a divisor V of length no greater than N. We assume that M(N) the number of steps used to multiply two domain elements of length N is of the form:

$$M(N) = N\ U(N)$$

where U(N) is a uniform non decreasing function in N. Then by inspecting the algorithm we derive the recurrence relation:

$$Q(2^r) = Q(2^{r-1}) + 3M(2^{r-1})$$

$$= 3 \sum_{i=1}^{r-1} M(2^i) \leq 3U(2^{r-1}) \sum_{i=1}^{r-1} 2^i$$

$$= 3U(2^{r-1})(2 \cdot 2^{r-1}-1)$$

$$= 6M(2^{r-1}) - 3U(2^{r-1}).$$

In other words the number of operations used to perform the Quot algorithm on elements of length 2^r is no more than that to perform 6 multiplications of elements of length 2^{r-1}.

As observed earlier the remainder in a division can be found in 2 more multiplications of domain elements of length 2^r. □

This yields immediately Cook's [Co66] result:

Corollary 6: A 2N digit integer can be divided by an N digit integer in O(N log N log log N) steps.

Also we have the polynomial division of algorithm derived by various authors.

Corollary 7:

Division of a polynomial of degree 2N by one of degree N can be performed in 8N-2 field multiplications or 36N log N + 60N arithmetic operations.

Proof: In the Ostrowski measure: (M(N) the number of field multiplications to multiply two polynomials of degree N is 2N+1 (see for example [Bor74]).

$$Q(N) = 4N-2$$

The time to compute $[P \cdot U]_{m-n} V$ given P is 4N which gives a total of 8N-2. Counting total operations we have, the number of arithmetic operations used to multiply two polynomials of degree N to 9N log N + 19N + lower degree terms (ibid). So

$$Q(N) = 18N \log N + 20N - 2 \log N$$

Including the time for computing $[P \cdot U]_{m-n} V$ this gives 36N log N + 60N operations if we allow 2N operations for the correction of $[P \cdot U]_{m-n}$ and subtracting to get the remainder. □

IV Summary

We have discussed the problem of devising an efficient division algorithm in a general euclidean domain. To this end we have shown that we need use only very weak properties of an EUD (in particular its valuation) to produce such an algorithm. While many algorithms have been stated in an abstract setting before, our contribution is in showing that an analysis of the efficiency of a particular algorithm can also be made in algebraic terms This has the advantage that it unifies algorithms and their efficiency measures for various domains into one common algebraic framework. What we show is that the efficiency of division is directly related to the efficiency of more primitive operations such as multiplication and addition of the domain.

The approach we have taken is to show that only a limited portion of the divisor and dividend need be considered to produce a unique quotient. Further we have shown that division can be performed by computing a psuedo-inverse of the divisor. This can be done within the domain and avoids the use of quotient fields which are poor structures to work with from the point of view of efficiency. An algorithm which is a variant of Newton's Method is given for computing the psuedo-inverse. The number of steps this algorithm uses is shown to be directly reducible to the number of steps needed for multiplication in the domain.

This yeilds as corollaries Cook's result that integer division is directly reducible to integer multiplication and that of several authors that polynomial division is directly reducible to the time for polynomial multiplication.

Acknowledgement

We wish to thank Professor A. Borodin for his help and encouragement in this work.

BIBLIOGRAPHY

Bor74 A. Borodin and R. Moenck: Fast Modular Transforms; JCSS (July 1974).

Co66 S. A. Cook: On the Minimum Computation Time of Functions; Thesis Harvard Univ. 1966 pp. 26-50.

Co72 S. A. Cook: Linear Time Simulation of Deterministic Two-Way Pushdown Automata; Proc. IFIP Cong. 1971, North-Holland London 1972, Vol. 1, pp. 75-81.

Knu69 D. Knuth: The Art of Computer Programming Vol. II: Seminumerical Algorithms; Addison-Wesley, Reading Mass. 1969. pp. 172.

Ku73a H. T. Kung: Fast Evaluation and Interpolation; Carnegie-Mellon, Dept. Comp. Sci. Tech. Report Jan. 73.

Ku73b H. T. Kung: On Computing Reciprocals of Power Series; Carnegie-Mellon, Dept. of Computer Science Tech. Report Sept. 73.

MB72 R. Moenck and A. Borodin: Fast Modular Transforms via Division; Proc. 13th Symp. on Switching and Automata Theory pp. 90-96 (revised as [Bor74]).

Moe73 R. Moenck: Studies in Fast Algebraic Algorithms; Ph.D. Thesis U. of Toronto 1973.

Ost54 A. Ostrowski: On Two Problems in Abstract Algebra Connected with Horner's Rule; Studies Presented to R. von Mises, Academic Press, New York, 1954 pp 40-48.

Sie72 M. Sieveking: An Algorithm for Division of Power Series; Computing Vol. 10, No. 1-2, pp. 153-156, 1972.

Sch73 A. Schönhage: Private Communication.

SS71 A. Schönhage and V. Strassen: Fast Multiplication of Large Numbers; Computing 7 (1971) pp. 281-292.

Str73 V..Strassen: Die Berechnungskomplexität von elementarsymetrischen Funktionen und von Interpolations koeffizienten; Numer. Math. Vol. 20, No. 3 1973 pp. 238-251.