MODULAR ARITHMETIC AND FINITE FIELD THEORY:

A TUTORIAL

E. Horowitz

# Department of Computer Science Cornell University Ithaca, N.Y. 14850

### ABSTRACT

The paradigm of algorithm analysis has achieved major pre-eminence in the field of symbolic and algebraic manipulation in the last few years. A major factor in its success has been the use of modular arithmetic. Application of this technique has proved effective in reducing computing times for algorithms covering a wide variety of symbolic mathematical problems. This paper is intended to review the basic theory underlying modular arithmetic. In addition, attention will be paid to certain practical problems which arise in the construction of a modular arithmetic system.

A second area of importance in symbol manipulation is the theory of finite fields. A recent algorithm for polynomial factorization over a finite field has led to faster algorithms for factorization over the field of rationals. Moreover, the work in modular arithmetic often consists of manipulating elements in a finite field. Hence, this paper will outline some of the major theorems for finite fields, hoping to provide a basis from which an easier grasp of these new algorithms can be made.

KEYWORDS: Modular arithmetic, finite fields, exact multiplication, symbol manipulation; CR CATEGORIES: 3.15, 3.64, 5.11.

# Introduction

The use of modular arithmetic in the area of mathematical symbol manipulation has gained increasing importance in the last few years. The major reason is because modular arithmetic allows us to perform exact multiplication faster than any of the conventional algorithms. Therefore, for complex operations such as polynomial greatest common divisor calculation or exact solution of linear systems of equations, where many multiplications of large integers are required, the use of modular arithmetic can produce substantial savings in computing times. A second use of modular arithmetic has been in the area of polynomial factorization over the field of rationals. However, the advantage gained here is not the ability for fast multiplication. Rather we can regard the solution of problems using modular arithmetic as a mapping from one domain (the integers) to another (the integers modulo p). The solution of the problem in this new domain is in some sense "easier" to obtain than in the former. Then, several of these solutions may be used to calculate the desired solution in the original domain. Hence, an effort to develop efficient methods for solution of problems over the integers has lead to a search for efficient solutions in the domain of integers modulo p. The use of modular arithmetic has both increased the efficiency of many symbolic operations and has given us a new point of view

for constructing fast algorithms. Related to the theory of modular arithmetic is the theory of finite fields. The study of this area has been accelerated by its application to the theory of error-correcting codes. By relating each digit of a given code to an element in a finite field, it was found possible to derive an algebraic equation whose roots represented the digits which were in error. The decoding problem was then reduced to forming this equation and finding its roots. Newer methods still rely upon performing arithmetic in either finite fields or in the ring of integers modulo m.

In Section 2 we will develop the theory of modular arithmetic and show how it can be used to effect a decrease in computing time for many different algorithms. In Section 3, the theoretical framework of finite field theory will be presented. Either one of these sections may be read independently of the other. In Section 4, a brief outline of some of the uses of these techniques will be covered. An extensive bibliography of recent work is included at the end of the paper.

### 2. Modular Arithmetic

In the Introduction it was stated that modular arithmetic gave us a new efficient way for performing arithmetic operations on integers. In this section we will discuss three questions pertaining to the use of this approach. First, what is the representation for integers and how do we transform an integer into this representation? Second, how are arithmetic operations performed on the integers while they are in this modular representation? Finally, how can we transform back from this modular form to the conventional integer representation?

We define the binary operator <u>mod</u> as follows: a mod b=a-b[a/b], if  $b\neq 0$ ; a mod 0=a. (1) From the definition it follows that  $0\leq a/b-[a/b]=(a mod b)/b\leq 1$ , if  $b\neq 0$ ;

therefore, if b>0 then  $0\le a \mod b\le b$  and if  $b\le 0$ then  $0\ge a \mod b\ge b$ . Since  $a-(a \mod b)$  is an integral multiple of b, we may regard a mod b as the remainder when a is divided by b. Though the definition of mod holds when a and b are arbitrary real numbers, from now on we will restrict their values to be integers. We say that two non-zero integers a and b are relatively prime if they have no common factor other than unity. Equivalently we say that the greatest common divisor of a and b is 1, i,e. gcd(a,b)=1.

The idea of a modular representation for integers is to choose several moduli,  $p_1, \ldots, p_r$ which satisfy certain conditions and then to work indirectly with "residues", namely  $a_i = a \mod p_i, 1 \le i \le r$ . Thus, the integer a is represented by the r-tuple  $(a_1, \ldots, a_r)$ . It is simple to compute  $(a_1, \ldots, a_r)$  from an integer a by means of the division command on any computer. The computing time for transforming into modular representation is clearly proportional to r.

Now, how can we perform arithmetic on integers in this form? It is helpful to introduce some notation from number theory, the notion of congruence. If the difference of two integers a and b is divisible by p, we shall say that a is congruent to b modulo p and use the notation

 $a \equiv b \pmod{p}$  .

There are some elementary properties of congruences.

<u>Theorem 2.1</u>. If  $a\equiv b$  and  $c\equiv d$ , then  $a\pm c\equiv b\pm d$  and  $ac\equiv bd (mod p)$ .

<u>Theorem 2.2</u>. If  $ac \equiv bd$ ,  $a \equiv b$  and gcd(a,p)=1, then  $c \equiv d \pmod{p}$ .

<u>Theorem 2.3</u>. For  $p \neq 0, c \neq 0$  a  $\equiv b \pmod{p}$  if and only if ac $\equiv bc \pmod{pc}$ .

<u>Theorem 2.4</u>. If  $p \neq 0, c \neq 0$  and the gcd(p,c)=1, then  $a \equiv b \pmod{p}$  if and only if  $a \equiv b \pmod{p}$ and  $a \equiv b \pmod{c}$ .

Proofs of these theorems can be found in any elementary book on number theory such as [10].

Theorem 2.1 tells us that we can perform addition, subtraction, and multiplication modulo p. Theorem 2.2 says that division is possible when the divisor is relatively prime to the modulus. The operations of addition, subtraction, multiplication and division which result from these two theorems are collectively called <u>modular arithmetic</u>.

If a is represented by the r-tuple  $(a_1, \ldots, a_r)$ where  $a_i = a \mod p_i$  and if b is represented as  $(b_1, \ldots, b_r)$  where  $b_i \equiv b \mod p_i$  and then we have the following:

(a<sub>1</sub>,...,a<sub>r</sub>)<sup>±</sup>(b<sub>1</sub>,...,b<sub>r</sub>) = ((a<sub>1</sub><sup>±</sup>b<sub>1</sub>)modp<sub>1</sub>,..., (a<sub>r</sub><sup>±</sup>b<sub>r</sub>)mod p<sub>r</sub>) ,

 $(a_1, \ldots, a_r) \cdot (b_1, \ldots, b_r) = (a_1b_1 \mod p_1, \ldots, a_rb_r \mod p_r)$ .

We would like to perform the operations  $(a_i^{\pm}b_i) \mod p_i, a_i^{\phantom{i}b_i} \mod p_i$  as fast as possible. We can avoid entirely the division operation for addition and subtraction if we restrict the  $p_i$  to be single precision positive numbers on the computer with which we are working. To avoid overflow we require  $p_i < 2\gamma$ 

where  $\gamma$  is the largest integer representable by one word of the computer. Then the following formulas apply:

$$(a_{i}+b_{i}) \mod p_{i} = \begin{cases} a_{i}+b_{i} & \text{if } a_{i}+b_{i} < p_{i} ; \\ a_{i}+b_{i}-p_{i} & \text{if } a_{i}+b_{i} > p_{i} ; \end{cases}$$

$$(2)$$

$$(a_{i}+b_{i}) \mod p_{i} = \begin{cases} a_{i}-b_{i} & \text{if } a_{i}-b_{i} > 0 ; \\ a_{i}-b_{i} & \text{if } a_{i}-b_{i} > 0 ; \end{cases}$$

$$(3)$$

$$(a_i - b_i) \mod p_i = \begin{cases} (a_i - b_i + p_i) & \text{if } a_i - b_i < 0 \end{cases}$$

After we have performed the desired sequence of arithmetic operations, we are left with the r-tuple  $(c_1, \ldots, c_r)$ . We now need some way of transforming back from modular form with the assurance that the resulting integer is the correct one. The ability to do this is guaranteed by the following theorem which was first proven in full generality by L. Euler in 1734. Let  $p_1, \ldots, p_r$  be positive integers which are pairwise relatively prime. Let r and  $p = \Pi_p_i$ i=1 let b, a1,...,a be integers. Then, there is exactly one integer "a" which satisfies the conditions  $b \le a \le b + p$ , and  $a \equiv a_i \pmod{p_i}$  for  $1 \le i \le r$ . (4) <u>Proof</u> (due to H L Garner [13]): If  $a \equiv x \pmod{p_i}$ for l<i<r, then a-x is a multiple of p, for all i. Since the p $_{
m i}$  are pairwise relatively prime, it follows that a-x is a multiple of p. Thus, there can be only one solution which satisfies (4). We can construct this solution in the following way: Let s , be defined such that

Theorem 2.5. (Chinese Remainder Theorem):

Then, let

$$t_1 \leftarrow a_1 \mod p_1$$
,  
 $t_2 \leftarrow (a_2 - t_1) \le s_{12} \mod p_2$  (5)  
 $t_3 \leftarrow ((a_3 - t_1) \le s_{13} - t_2) \le s_{23} \mod p_3$   
 $\cdots$ 

$$t_r^{+}(...(a_r^{-t_1})s_{1r}^{-t_2})s_{2r}^{-}...t_{r-1})s_{(r-1)r}mod p_r$$

Then

 $a=t_{r}p_{r-1}\cdots p_{1}+\cdots+t_{3}p_{2}p_{1}+t_{2}p_{1}+t_{1}$ 

satisfies the conditions  $0 \le a \le p$ ,  $a \equiv a_i \mod p_i$ ,

 $1 \le i \le r$ . If [0,p) is not the desired range, any multiple of p can be added or subtracted after conversion is completed.

Thus, the Chinese Remainder Theorem guarantees that we can use a modular representation for numbers in any consecutive interval of

 $p = \prod_{i=1}^{n} p_i$  integers. That is, there is a unique i=1

result and it can be obtained using the procedure which is outlined in the proof of Theorem 2.5.

There are several aspects of this theorem which should be especially studied if a computer program is to be written. A total of  $\binom{r}{2}$  constants, s must first be calculated.

These are easily obtained by using Euclid's algorithm which determines x,y such that  $xp_i+yp_j=gcd(p_i,p_j)=1$ , see [18] or [5]. Remember that the moduli are pairwise relatively prime. The  $t_i$ ,  $1 \le i \le r$  may be found according to (5). Note that all arithmetic is mod  $p_i$  and this capability is already contained within the system.

In actual practice Theorem 5 is often implemented for only two moduli. Examples of this can be found in the systems of Brown, [4,p. 28] and Collins [8]. The algorithm then proceeds as follows: we are given  $p_1, p_2$  where  $p_1$  is generally much larger than  $p_2$  and  $a_1, a_2$  such that  $|a_1| < p_1/2, 0 \le a_2 < p_2$ .

1) Find  $s_{12}$  such that  $s_{12}p_1 \equiv 1 \mod p_2$ 

2) let  $t_1 \leftarrow a_1 \mod p_1$  $t_2 \leftarrow (a_2 - t_1) \le s_{12} \mod p_2$  $a \leftarrow t_1 + t_2 p_1$ 

Thus, a satisfies  $0 \le a \le p_1 p_2$  and  $a \equiv a_1 \mod p_1$ for  $1 \le i \le 2$ . Instead of  $p_1$  being a single modulus it is generally equal to the product of the previously used moduli. Also,  $a_1$  is the current tentative solution. Then a is the new tentative solution and  $p_1$  is updated by setting  $p_1 \le p_1 p_2$ . The correct solution is obtained when either the proper number of moduli have been processed or when the "tentative" solution satisfies some prearranged condition. The computing time for the above version of Garner's method is proportional to  $\log_{10} p_1$ .

In order to process as few moduli as possible we would like to choose the  $\mathbf{p}_{i}$  to be very

large. In order to avoid the division operation for addition and subtraction, the moduli should be single precision. Theorem 5 requires that the moduli be pairwise relatively prime. Therefore, the moduli are easily chosen to be a set of consecutive, single precision primes. Given a computer whose word length is k bits, there will be a minimum of  $\frac{2^{k-1}}{2^{k-1}/k=q}$  primes in

the interval  $[2^k, 2^{k+1}]$ . Since k ranges between 30-60,  $g \ge 10^7$ . Algorithms to compute these primes can be found in either [18,p.143] or [8,p.4].

A practical and important issue in the use of a modular arithmetic system is the determination of r, the number of moduli, that must be processed until the correct answer can be obtained. Unfortunately, it is difficult to test whether or not overflow has occurred as the result of an addition, subtraction, or multiplication when using modular representation. In [19,p.257], Knuth shows that any method which tests for overflow must rely on all the residues at once. Thus, the computing time for this check would nullify the advantages gained from the modular representation.

One technique for deciding on the number of moduli is to estimate from the inputs a bound for the maximum size of any resulting integer. For example, in [6,pp.215-216], Collins uses Hadamard's theorem to derive a bound for the coefficients of the polynomial which is the greatest common divisor of the two given polynomials. He uses this bound to determine the number of moduli (in this case single precision odd primes, called p.) which must be processed before he can apply the Chinese Remainder Theorem. The calculation of this bound requires first the summing of the magnitude of the coefficients of the input polynomials, say d and e. Then the least integers r,s are computed such that

 $2^{r} \ge d, 2^{s} \ge e$ . Then, if the degrees of the polynomials are m and n, compute t=ms+nr and u=[t/h]+1 where h satisfies  $2^{h} \le p_{i}$  and  $2^{h}$ 

is usually about half the largest integer which can be stored in one computer word. After u primes have been successfully processed, the correct solution is guaranteed.

Another technique for determining when the computations are complete is to constantly maintain a "tentative" solution. After each new modulus is processed, a check can be made to see if the correct solution has been computed. Since this test may well require multiprecision calculations, its total computing time must be small. Suppose this technique is used in a modular arithmetic based algorithm for determing  $C(x) = \gcd(A(x), B(x))$ . For an actual example see [4,p.312]. Then, after each new modulus is processed the tentative solution  $\overline{C}(x)$  is obtained.

If  $\overline{C}(x) | A(x)$  and  $\overline{C}(x) | B(x)$  then  $\overline{C}(x)=C(x)$ . This approach can be especially efficient if the bound for the resulting integers is much larger than their true size. In this case we are processing the minimum number of moduli which are required at the expense of a test after each modulus is used.

Of course, some problems do not allow for simple efficient tests for a correct solution. A case in point is the exact solution of linear systems of equations. A matrix multiplication and a vector compare is necessary to see if no more moduli need be processed. The computing time here is prohibitive and rather an a priori bound for the number of moduli is calculated. Finally, some problems are not amenable to a test at all. An example of this would be determinant calculation.

Let us now consider a more formal approach to modular arithmetic. Let I stand for the integral domain of the integers and consider the mapping  $h_m: I \rightarrow I/(m)$ , from I onto the ring of integers

mod m. If m=p, a prime then the elements of I/(p), namely  $\{0,1,2,\ldots,p-1\}$  form a finite field usually called the Galois field with p elements and designated as GF(p). This mapping h constitutes a homomorphism because it is an onto mapping and if  $a,b\in I$  it follows that

 $h_p(a) \cdot h_p(b) = h_p(a \cdot b), h_p(a) + h_p(b) = h_p(a+b)$ . We can

extend this modular homomorphism to polynomial domains in the natural way. If  $A(x_1, ..., x_r)$  is an r-variable polynomial over I, let  $h_p^* : I[x_1, ..., x_r] \rightarrow GF(p)[x_1, ..., x_r]$  where  $h_p^*(x_i) = x_i$  and  $h_p^*(c) = h_p(c)$  where c is a numerical

coefficient of A.

Now, a modular arithmetic system for symbol manipulation must first provide a reasonable number of single precision primes, say 50-100. Then a subprogram is needed which applies  $h_p$ to any integer or more generally  $h_p^*$  to any multivariable polynomial with integer coefficients. If  $A(x_1, \ldots, x_r)$  has  $n_i$  as the maximum degree of  $x_i$  in A for  $1 \le i \le r$  and if N bounds in magnitude the numerical coefficients of A, then the computing time to obtain  $h_p^*(A(x_1, \dots, x_r))$  is proportional to  $n_1 n_2 \dots n_r (\log N)$ . The inverse operation consists of applying Garner's version of the Chinese Remainder Theorem. Here the inputs are two polynomials  $A(x_1, \ldots, x_r)$  over GF(p) and  $B(x_1, \ldots, x_r)$  over I and two integers Q and p where Q is relatively prime to p. The output is the unique polynomial  $C(x_1, \ldots, x_r)$  over I which satisfies  $C \equiv B \pmod{p}$ ,  $C \equiv A \pmod{p}$  and the coefficients of C are less than  $p \cdot Q/2$  in magnitude. If both A and B have maximum degree  $n_{i}$  in  $x_{i}$ , then the computing time, using Garner's version is proportional to  $n_1 n_2 \dots n_r (\log Q)$ .

Another homomorphism which has proven useful in conjunction with the modular homomorphism is the <u>evaluation</u> <u>homomorphism</u>, E<sub>b</sub>. When

applied to some polynomial  $A(x_1, ..., x_r)$  it produces  $C(x_1, ..., x_{r-1}) = A(x_1, ..., x_{r-1}, b)$ . In

order to minimize the computing time for algorithms with multivariable polynomial arguments, the modular homomorphism is applied first. Then, the evaluation homomorphisms are applied to polynomials with coefficients in GF(p). If n<sub>i</sub> is the maximum degree of x<sub>i</sub> in A

for  $1 \le i \le r$ , the computing time to form  $C(x_1, \ldots, x_{r-1})$  is proportional to  $n_1 n_2 \ldots n_r$ . The iverse operation requires an interpolation algorithm. As with the Chinese Remainder Theorem an iterative algorithm can be used. The inputs are three polynomials

 $B(x_r) = \prod_{\substack{0 \le i \le m}} (x_r - b_i)$  where the  $b_i$  are distinct

elements of GF(p),  $A(x_1, \ldots, x_r)$  over GF(p) with degree m or less in  $x_r$  and  $C(x_1, \ldots, x_{r-1})$ over GF(p). The output is the unique polynomial  $D(x_1, \ldots, x_r)$  of degree m+1 or less in  $x_r$  defined as

$$D(\mathbf{x}_1, \ldots, \mathbf{x}_r) =$$

{ $C(x_1, ..., x_{r-1}) - A(x_1, ..., x_{r-1})$ }.B(x\_r)/B(b), + $A(x_1, ..., x_r)$ 

Thus we see that  $D(x_1, \ldots, x_{r-1}, b_i) =$ 

 $A(x_1, ..., x_{r-1}, b_i)$  for  $0 \le i \le m$  and  $D(x_1, ..., x_{r-1}, b) = C(x_1, ..., x_{r-1})$ . The computing time can clearly be made proportional to  $n_1 n_2 ... n_{r-1} m$ where  $n_i$  is the degree of  $x_i$  in A and m is

defined above. Precise algorithms for evaluation homomorphism and interpolation can be found in [8,p.17].

Both Brown in [4] and Collins in [7] use modular and evaluation homomorphisms to produce efficient algorithms for multivariable gcd and resultant calculation, respectively. The modular homomorphism is applied first and a solution over  $GF(p_i)$  is obtained using evaluation homomorphisms. These are applied and reduce the single problem to several with inputs of univariate polynomials and coefficients over  $GF(p_i)$ . The solutions are inter-

polated to form a single multivariable solution over  $GF(p_i)$ . After each  $p_i$  has been processed a tentative solution is constructed via the iterative version of the Chinese Remainder Theorem. After a sufficient number of primes have been done the correct answer has been produced.

Besides the increased efficiency of this approach an important auxiliary benefit is derived. When this schema is applied to gcd or resultant calculation, the natural way of calculating a polynomial remainder sequence can be used over GF(p). Also, in linear systems algorithms with multivariable polynomial elements when the system is finally reduced to one with elements in GF(p), the classical Gaussian elimination can be used to solve the system. Thus, the previously used methods (e.g. reduced p.r.s. algorithm for gcd calculation) which were the most efficient known, have been outmoded. The use of modular and evaluation homomorphisms has returned us to our classical methods and provides a much more esthetic way of efficiently solving these problems.

#### 3. Finite Fields

In the previous section we have discussed the application of modular arithmetic to symbol manipulation. In general, this technique is used by mapping the problem from the domain of integers to the domain GF(p), the finite field with a prime number of elements p. The algorithm was then carried out using arithmetic in GF(p) or GF(p)[x]. It is useful then to study and understand more fully some of the properties of arithmetic in these fields.

The theory of finite fields has also proved very useful in the area of coding theory, see [1]. In connection with this work, E R Berlekamp in 1967 discovered a fast factorization algorithm for polynomials with coefficients in a finite field. It turns out that one can obtain the factors of an nth degree polynomial over GF(p) faster than one can find the factors of an arbitrary n-bit binary integer. This algorithm for factoring modulo p while useful in itself, has aided in constructing new algorithms for factoring polynomials with integer coefficients.

Therefore, the application of finite field theory has provided both new algorithms (factorization) and a new way of speeding up our old algorithms. In this section we will develop the theory of finite fields and show how certain properties make them especially useful in mathematical symbol manipulation.

<u>Definition</u> A field F is a set of elements . including 0 and 1 for which the operations of addition and multiplication are closed, associative, and commutative. Multiplication distributes over addition such that  $a^*(b+c) =$ ab+ac where  $a,b,c,\in F$ . For every non-zero element  $a \in F$  there is a unique reciprocal, 1/a such that  $a^*(1/a) = 1$ . For every  $a \in F$  there is a unique negative, -a, such that a+(-a) = 0, 0+a=a=1\*a and  $0\cdot a=0$ .

The <u>order</u> of a field is the number of elements in the field. The rational, real and complex numbers are all examples of infinite fields. If p is a prime, then the integers mod p form a finite field of order p, designated as GF(p).

If a field F contains "a", then it must also contain  $a^{-2}, a^{-1}, 1, a, a^2, \ldots$ . If these powers of "a" are not all distinct, then for some m,n we have  $a^m = a^n$  or  $a^{m-n} = 1$ . The least positive integer n for which  $a^n = 1$  is called the <u>order of a</u>. If the order of "a" is n, then  $1, a, a^2, \ldots a^{n-1}$  are all distinct. In a finite field, each element can have only a finite number of distinct powers. Hence, every non-zero element in a finite field has a finite order.

The following three theorems establish some elementary facts about the orders of elements in a field. Proofs of these theorems can be found in [1,p.89].

<u>Theorem 3.1</u>. If a has order n, then  $a^m = 1$  if and only if m is a multiple of n.

<u>Theorem 3.2</u>. If a has order m, b has order n and gcd(m,n)=1 then ab has order mn.

Theorem 3.3. If a has order n, then the order of  $a^k$  is n/gcd(n,k).

Definition If a is an element of a finite field F and "a" has order n, then "a" is said to be a <u>primitive nth root of unity</u>. If the order of F is q, then "a" is said to be a <u>primitive field element</u> if the order of "a" is q-1.

In [1], Berlekamp establishes the existence of a primitive field element for any finite field of order q. If "a" is this element, then  $1,a,a^2,\ldots,a^{q-2}$  constitutes all non-zero elements in the field. An immediate consequence of this result is that every element in the field satisfies the equation  $x^{q}-x=0$ . If "a" is the primitive field element, then  $x^{q-1}-1=\prod(x-a^i)=\prod(x-b)$ 

i=0 b€GF(q) b≠0

Since lf F, it follows that (  $\overset{\frown}{\Sigma}$  l)  $\in$  F for all i=l

n. If these elements are not all distinct, then there exists a least p such that p  $\Sigma$  1=0 i=1 in the field. This number p is called the <u>characteristic</u> of the field. If n is non- $\Sigma$  1 i=1

zero for every n, then we say that the characteristic is  $\infty$ . In a field of p-1 characteristic p, the elements 1,1+1,...,  $\Sigma$  1 i=1

are called the field integers.

Theorem 3.4. The characteristic of any field is either  $\infty$  or a prime number p.

 $\frac{\text{Proof}}{\Sigma \ 1 \approx 0 \text{ and } \Sigma \ 1 \neq 0, \text{ then we can}}$ i=1 multiply by 1/  $\Sigma$  1 and obtain  $\Sigma$  1=0. Hence, if  $\Sigma$  1=0, i = 1i=1 then either  $\sum_{i=1}^{m} \sum_{i=1}^{n} \sum_{i$ If the field has characteristic  $\infty$ , then there are an infinite number of distinct elements and the order of the field is  $\infty$ . If the field has characteristic p, its order may be finite or infinite. For example, the integers mod p form a finite field of characteristic p while the set of rational functions A(x)/B(x) with coefficients from the integers mod p form an infinite field of characteristic p. Theorem 3.5. In a field F of characteristic p, the field integers form a subfield of order p isomorphic to the field of integers mod p. Proof The field integers are closed under the

four arithmetic operations. For any integer k < p, we can find integers r,s such that rk+sp=1, [18,p. 302]. Therefore, in the field

 $(\Sigma 1)(\Sigma 1)=1$  so that  $\Sigma 1$  is the reciprocal i=1 i=1 i=1 k

of  $\Sigma$  1. This subfield is usually called i=1

the prime subfield of F.

<u>Theorem 3.6</u>. In any field of characteric p,  $x^{P}-a^{P}=(x-a)^{P}$ .

<u>Proof</u> Applying the binomial theorem to  $(\mathbf{x}-\mathbf{a})^p$ , we get  $(\mathbf{x}-\mathbf{a})^p = \sum_{i=0}^{p} {p \choose i} \mathbf{x}^i (-\mathbf{a})^{p-1}$ . But for  $0 \le i \le p$ , i=0 ${p \choose i} = (p(p-1) \dots (p-i+1))/i! \equiv 0 \pmod{p}$  since the

numerator contains a factor of p which cannot be removed. Hence only  $x^{p}-a^{p}$  remain.

In fact, a more general theorem can be proven. Theorem 3.7. If  $a_1, \ldots a_k \in F, F$  a field of

characteristic p, then

 $\begin{pmatrix} k & n & k & n \\ (\sum_{i=1}^{n} a_i)^p &= \sum_{i=1}^{n} a_i^p \text{ for all } n.$ 

Now suppose that A(x), B(x) are any polynomials with coefficients in GF(p). Then

 $(A(x)+B(x))^{p}=A(x)^{p}+{\binom{p}{i}}A(x)^{p-1}B(x)+...+$  ${\binom{p}{p-1}}A(x)B(x)^{p-1}+B(x)^{p}=A(x)^{p}+B(x)^{p}$ 

since  $\binom{p}{i}$  is divisible by p for  $1 \le i \le p-1$ .

Now there is a definite relationship between the characteristic p and the order q of any finite field.

Theorem 3.8. The order of a finite field is a power of its characteristic.

Proof [21, p.116] If we consider the prime subfield P of any field F of order q and characteristic p, then there is in F a maximal set of linearly independent elements a<sub>1</sub>,..., a<sub>n</sub> with respect to P. Every element in (6) F is of the form  $c_1a_1+c_2a_2+\ldots+c_na_n$ with uniquely determined coefficients c, EP. Since for every c<sub>i</sub>,p values are possible by Theorem 3.5, thus there are exactly  $p^n$  expressions for (6). Since these constitute all elements in F, thus q=p<sup>n</sup>.

We are now able to establish the following theorem concerning finite fields and irreducible polynomials. Proofs can be found in [1, p.103].

Theorem 3.9. Every element in a field of order q satisfies the equation

 $x^{q^{n}} - x = 0$  for every n.

Theorem 3.10. Every irreducible polynomial of degree d over a field of order q divides  $x^{q}$  - x if k is a multiple of d.

<u>Theorem 3.11</u>. In a field of order q,  $x^{q^{n}}-x$  factors into the product of all monic irreducible polynomials whose degrees divide k. Therefore, in I mod p,  $x^p$  -x factors into the product of all monic irreducible polynomials

whose degrees divide k. In a field of order p<sup>k</sup>,x<sup>p</sup> -x factors into p<sup>k</sup> linear factors. Since I mod p is a subfield

of the order  $p^k$  we can equate the factorization and obtain:

Theorem 3.12. If f(x) is an irreducible polynomial of degree m over I mod p and m k, then a field of order p must contain m roots of f(x).

Let f(x) be any irreducible polynomial of degree k over the integers mod p. Then any finite field of order  $p^k$  must contain k roots of f(x). If a is one root, then every element of the field is expressible as a polynomial in a degree less than k. Thus, if p is any prime and k is any integer, then there exists a unique finite field of order  $p^k$ .

These properties of finite fields play a key part in the new factorization algorithm of Berlekamp, see [19, p.381] and [2]. Also, they underlie the general techniques of modular arithmetic which have been discussed in Section 2. In some of the papers given at this conference, namely [4], [7], and [2], the foregoing theory and methods are extensively used.

#### 4. Applications

The idea of modular arithmetic and congruences goes back to classical number theory. An exposition of these concepts can be found in many books, for example, see [10]. After the advent of computers, modular arithmetic was reinvestigated as an approach for performing fast arithmetic operations by the central

processing unit of a digital computer. For a complete discussion of the problem and results in this area see [22].

In the past decade many papers have appeared which describe mathematical software packages for certain applications which use modular arithmetic. An early such paper was by H. Takahasi and Y. Ishibashi, [20], in which applications such as matrix inversion, determinant calculation and interpolation are discussed. More recent work on the exact solution of linear systems has been done by Borosh and Fraenkel, [3], Howell and Gregory in [16] and [17] and by Horowitz in [14] and [9]. The problem of polynomial greatest common divisor calculations using modular arithmetic has been treated by W.S.Brown in [4], G.Collins in [6] and D.Knuth in [19, pp.393-395]. A modular approach for the Extended Euclidean algorithm for univariate polynomials is given by Horowitz in [15]. A modular algorithm for computing multivariate resultants is given by G.Collins in [7].

The factorization of polynomials with coefficients either in GF(p) or in I can be found in [19, pp.381-398]. In [1] an excellent review of Berlekamp's method for polynomial factorization any many examples can be found. A new improved version is discussed by him in [2].

# References

- [1] Berlekamp, E.R, Algebraic Coding Theory, McGraw-Hill Book Co., New York, 1968, Chapters 2,4, and 6.
- [2] Berlekamp, E.R, "Factoring polynomials over large finite fields," Mathematics of Computation, July, 1970.
- [3] Borosh, I. and A.S. Fraenkel, "Exact solution of linear equations with rational coefficients by congruence techniques," Mathematics of Computation, Vol.20, No. 93 (January 1966), pp. 107-112.
- [4] Brown, W.S., "On Euclid's algorithm and the computation of polynomial greatest common divisors," Proceedings of the Second Symposium on Symbolic and Algebraic Manipulation, Los Angeles, March 1971.
- [5] Collins, G.E., "Computing multiplicative inverses in GF(p)," Mathematics of Computation, Vol.23, No.105 (January 1969, pp. 197-200.
- [6] Collins, G.E. "Computing time analysis of some arithmetic and algebraic algorithms," Proceedings of the IBM 1968 Summer Institute on Symbolic Mathematics by Computer, IBM Boston Programming Center, Cambridge, Mass, June 1969, pp. 195-232.
- [7] Collins, G.E. "The Calculation of Multivariate Polynomial Resultants," Proceedings of the Second Symposium on Symbolic and Algebraic Manipulation, Los Angeles, March, 1971.
- [8] Collins, G.E., and E Horowitz, et al., "The SAC-1 modular arithmetic system," Computing Center and Computer Sciences Department, University of Wisconsin, Technical Reference No. 10, June 1969.
- [9] Collins, G.E. and E Horowitz, "The SAC-1 partial fraction decomposition and rational function integration system." Computing

Center and Computer Sciences Department, University of Wisconsin, Technical Reference No. 80, February 1970.

- [10] Dickson, E.L. Introduction to the Theory of Numbers, Dover Publications, Inc, New York, 1929.
- [11] Feldman, H.A., "Some symbolic computations in finite fields", Proceedings of the IBM Summer 1968 Institute on Symbolic Mathematics by Computer, IBM Boston Programming Center, Cambridge, Mass, June 1969, pp. 79-96.
- [12] Fraenkel, A.S., "The use of index calculus and Mersenne primes for the design of a high speed digital multiplier," Journal of the ACM, Vol. 8 (1961), pp. 87-96.
- [13] Garner, H.L. "The residue number system" IRE Transactions, EC-8 (1956), pp 140-147.
- [14] Horowitz, E, "Algorithms for partial fraction decomposition and rational function integration," Proceedings of the Second Symposium on Symbolic and Algebraic Manipulation, Los Angeles, March 1971.
- [15] Horowitz, E. Algorithms for Symbolic Integration of Rational Functions, PhD Dissertation, University of Wisconsin, Madison, Wisconsin, November 1969.
- [16] Howell, J.A. and R.T. Gregory, "An algorithm for solving linear algebraic equations using residue arithmetic I," BIT, Vol.9 (1969),pp 200-224.
- [17] Howell, J.A. and R.T.Gregory, "Solving linear equations using residue arithmetic-Algorithm II", TNN-95, Computation Center University of Texas at Austin, September 1969.
- [18] Knuth, D.E. The Art of Computer Programming, Vol. 1: Fundamental Algorithms, Addison-Wesley, 1968, p. 302.
- [19] Knuth, D.E. The Art of Computer Programming, Vol. 2, Seminumerical Algorithms, Addison-Wesley, 1969, pp. 248-257.
- [20] Takahasi, H., and Y. Ishibashi, "A new method for exact calculation by a digital computer, Information Processing in Japan, Vol. 1 (1961), pp. 28-42.
- [21] Van der Waerden, B.L., <u>Modern Algebra</u>, translated by F. Blum, Vol. 1, New York, Ungar Publishing, 1949.
- [22] Szabo, N.S. and R.I. Tanaka, <u>Residue</u> <u>Arithmetic and It's application to</u> <u>Computer Technology</u>, McGraw-Hill, New York, 1967.