WIS-CS-157-72
Computer Sciences Department
The University of Wisconsin
1210 West Dayton Street
Madison, Wisconsin 53706

THE SAC-1 POLYNOMIAL FACTORIZATION SYSTEM*

by

G. E. Collins and D. R. Musser†

Technical Report #157@

March, 1972

Received: July 21, 1972

*Research supported by National Science Foundation grants GJ-239 and GJ-30125X, the Mathematics Research Center, the Madison Academic Computing Center, and the Wisconsin Alumni Research Foundation.

†Present address: Department of Computer Science, University of Texas, Austin, Texas 78732.

@This report also appears as University of Wisconsin Madison Academic Computing Center Technical Report No. 30.

TABLE OF CONTENTS

1. Introduction

The SAC-1 Polynomial Factorization System is a set of Fortran subprograms for efficient factorization of univariate polynomials over the integers into polynomials which are irreducible over the integers (an irreducible polynomial A over the integers is one which has no factors besides 1,-1,A,-A). The algorithms employed for this operation are based on the use of mod p factorizations and constructions based on Hensel's Lemma as suggested by Zassenhaus, and were originally presented in [MUS71].

The Polynomial Factorization System is a new subsystem of SAC-1 (for Symbolic and Algebraic Calculation - version 1), a Fortran-based system, for performing operations on multivariate polynomials and rational functions with infinite-precision coefficients. The capabilities of the SAC-1 system are surveyed in [COL71a] and detailed manuals for previous subsystems appear in [COL67,COL68a,COL68b,COL68c,COL69a,COL70a,COL70b,COL72]. The present sybsystem uses subprograms from the List Processing, Integer Arithmetic, Polynomial, Modular Arithmetic, and the Polynomial G.C.D. and Resultant Calculation Systems. From the latter subsystem only the new modular PGCD subprogram is used; the older (reduced p.r.s.) PGCD subprogram of the Polynomial System could be used, at a savings in memory requirements, but at considerable expense in computing time for gcd calculations. However, often only one gcd calculation will be required during

factorization of a polynomial (see Sections 5.1 and 5.6).

A new COMMON block, TR5, is required and is described in Section 5.5.

Four of the subprograms included in the system, IAND, IOR, ILS, and MEMBER, are machine-dependent; the Fortran versions which are included may have to be revised or rewritten in machine language for use on other machines. For a great many machines, probably including all those on which SAC-1 has been implemented, the changes required will be very minor; see Section 3.

1.1. Theoretical Background

The classical solution to the polynomial factoring problem is Kronecker's algorithm [VDW49, Section 2.5], but this method is known to be very inefficient. The algorithms used in the Polynomial Factorization System are based on two recent breakthroughs: (1) the development of an efficient algorithm for mod p factorization by Berlekamp [BER68, Chapter 6] and (2) the suggestion by Zassenhaus [ZAS69] that one could combine Berlekamp's algorithm and Hensel's p-adic construction [VDW49, Section 76] to obtain a practical method of factoring polynomials with integer coefficients.

Berlekamp's algorithm, which is also discussed in [KNU69, Section 4.6.2], opened the way to the design of factoring algorithms for polynomials with integer coefficients based on mod p factorizations. However, Berlekamp's original algorithm is efficient only for small primes, so that factorization modulo of a single prime is generally not sufficient to determine factorizations over

the integers.

More recently, Berlekamp has proposed a new algorithm [BER70] which appears to be reasonably efficient even for very large primes. However, while his original algorithm has been implemented in the SAC-1 Modular Arithmetic System [COL69, Section 3.8], the new algorithm is quite complicated, and, to date, no implementation has been published.

Zassenhaus showed, however, that under certain conditions a construction based on "Hensel's Lemma" could be used to progress from a mod p factorization to a corresponding factorization modulo any power of p. Taking p^j sufficiently large, we can determine from consideration of all mod p^j factorizations all true factorizations. The number of mod p^j factorizations is the same as the number of mod p factorizations.

In more detail, let Z denote the ring of integers, and Z/(m) denote the ring of integers modulo m. Assuming the elements of Z/(m) are represented by the integers 0,1,...,m-1, with addition and multiplication performed modulo m, there is a unique homomorphism h_m of Z onto Z/(m) such that $h_m(i) = i$ for $0 \leq i < m$. h_m induces a unique homomorphism h_m^* of Z[x] onto (Z/(m))[x] such that $h_m^*(a) = h_m(a)$ for $a \in Z$ and $h_m^*(x) = x$. h_m^* will also be denoted by h_m.

Given a polynomial C(x) over Z to be factored, is not difficult to compute a bound b for the coefficients of any factor of C. (Interestingly, the coefficients of a factor can be larger than those of C itself. However, bounds can be obtained without

actually factoring C; such bounds are discussed in Section 4). Let m be an odd integer > 2b and R be the set of integers in the range $-\frac{m}{2} < n < \frac{m}{2}$.

Then the coefficients of every factor of C lie in R, and corresponding to any polynomial G in Z/(m)[x] is a unique polynomial F with coefficients in R such that $h_m(F) = G$. Thus we know that each factor G of $h_m(C)$ uniquely determines a polynomial F with coefficients in R such that $h_m(F) = G$. Suppose C = AB; then $h_m(C) = h_m(A)h_m(B)$, and thus if we have some way of enumerating all of the factors of $h_m(C)$, then when we consider the factor $G = h_m(A)$ we obtain F = A. The factors of C can therefore be determined by considering each factor G of $h_m(C)$ over Z/(m) and testing, by division, whether the corresponding polynomial F is a factor of C.

If m = p, a prime, then Z/(p) is a field called the Galois field of order p and also denoted by GF(p). In this case, $h_p(C)$ has a unique factorization into prime polymomials in GF(p)[x] from which the set of all factors of $h_p(C)$ may be computed. The complete factorization of polynomials in GF(p)[x] may be accomplished by means of one of Berlekamp's algorithms, as mentioned above.

If m is not prime, then (Z/(m))[x] is not a unique factorization domain. (Z/(m) is not even an integral domain.) However, if $m = p^j$, a power of a prime, then certain polynomials in (Z/(m))[x] do have a "complete factorization" (see Section 5.3).

A complete factorization mod p^j may be determined from a complete factorization mod p by means of the Hensel algorithms which are described in Section 5. Thus m may be taken as a power of a small prime p, and Berlekamp's original algorithm may be used for the factorizations over GF(p).

Before describing the details of the algorithms which are based on these ideas, we define the terminology used in the description and computing time analyses of algorithms in Sections 1.2 and 1.3, and describe a number of algorithms for basic operations in Sections 2, 3 and 4.

1.2. Polynomial and List Terminology

A polynomial over the integers,

$$A(x) = a_n x^n + a_{n-1} x^{n-1} + \ldots + a_1 x + a_0$$

with $a_n \neq 0$ is said to have degree n, leading coefficient a_n, and trailing coefficient (or constant term) a_0; we write

$$\deg A = n, \quad \mathrm{ldcf}\ A = a_n, \quad \mathrm{tlcf}\ A = a_0.$$

By convention, we define

$$\deg 0 = -\infty, \quad \mathrm{ldcf}\ 0 = 0, \quad \mathrm{tlcf}\ 0 = 0.$$

We say A(x) is monic if ldcf A = 1, positive if ldcf A > 0, primitive if the greatest common divisor of $a_0, a_1, \ldots, a_n$ is unity, and squarefree if it is the product of distinct irreducible factors.

Polynomials are assumed to be represented by list structures as specified in [COL68b], although little direct use is made of this fact in the algorithm descriptions given in this report.

However, a number of the algorithms perform some list processing operations, so we shall define here the notation used to describe these operations.

A list can be defined simply as a finite sequence, with the understanding that some of the elements in the sequence may be other lists. Let A be the list $(a_1,a_2,\ldots,a_n)$; we define

$$\text{first}(A) = a_1,\ \text{second}(A) = a_2,\ \text{last}(A) = a_n$$

$$\text{tail}(A) = (a_2,\ldots,a_n)$$

$$\text{prefix}(a,A) = (a,a_1,\ldots,a_n)$$

$$\text{inverse}(A) = (a_n,\ldots,a_1)$$

$$\text{length}(A) = n.$$

If A = (), the empty list, we define prefix(a,A) = (a), inverse(A) = (), length(A) = 0.

When the elements of A are integers or polynomials over the integers, (or their list representations), we define

$$\Sigma A = a_1 + a_2 + \ldots + a_n,$$

$$\Pi A = a_1 \cdot a_2 \ldots a_n.$$

We define $\Sigma() = 0$ and $\Pi() = 1$.

The following two subprograms provide basic operations on lists and polynomials.

Algorithm M = LAST(L). L is an arbitrary list. The output is M = last(L) if L is nonempty, M = 0 if L is empty. (If last(L) is a list then its reference count is not increased.)

Algorithm T = PTLCF(P). P is an arbitrary polynomial. The output is T = tlcf(P). (If tlcf(P) $\neq$ 0, the reference count of the list for tlcf(P) is increased by one.)

1.3. Computing Time Analysis Terminology

The results of computing time analyses of the various algorithms given in this report are stated in terms of the dominance relation, instead of the traditional O-notation.

The dominance relation is defined as follows ([COL71]). Let f and g be real-valued functions defined on some set S. We write $f \preceq g$ in case there is a positive real number c such that $f(x) \leq c \cdot g(x)$ for all $x \in S$ and we say that f is dominated by g (or that g dominates f). If $f \preceq g$ and $g \preceq f$ we write $f \sim g$ and say that f and g are codominant. Codominance is clearly an equivalence relation. Note that this definition encompasses functions of several variables since the elements of S may be n-tuples.

As most of the algorithms to be discussed involve operations on large integers, computing times are dependent on the size of these integers. We shall assume that integers are represented in radix notation with some arbitrary integer base $\beta > 1$. By the length of an integer n we then mean the number of β-digits in its representation. We shall use L(n) consistently to denote the length of n, with respect to some implicit base β. In the SAC-1 Integer Arithmetic System [COL68a], an integer is represented as a list of its β-digits, where β is generally chosen to be near the largest machine integer for the particular machine on which the system is implemented.

For the expression of bounds on the time for operations on

polynomials it is convenient to define two norms for polynomials, as follows. Let $A(x) = \sum_{i=0}^{m} a_i x^i$ be a polynomial over the integers. Define

$$|A|_1 = \sum_{i=0}^{m} |a_i|, \quad |A|_\infty = \max_{0 \le i \le m} |a_i|.$$

2. Modular Arithmetic

The SAC-1 Modular Arithmetic System [COL69] can perform arithmetic on polynomials modulo any positive integer m which is a Fortran integer (single-precision integer). In the factoring algorithms to be described in Section 5 we have need of some operations modulo an integer m which can be much larger than the bound on Fortran integers. Thus in this section we describe algorithms for these operations which take as input an L-integer modulus. ("L-integer" abbreviates "list-integer", reflecting the fact that large integers are represented in SAC-1 by lists of digits.)

Besides the L-integer modulus m, which is assumed to be odd and > 1, the inputs and outputs for each algorithm are other L-integers or univariate polynomials with L-integer coefficients. These polynomials are assumed to have the same list representation as in the SAC-1 Polynomial System [COL68b]. The inputs and outputs have, however, the additional property that their coefficients are bounded by m/2 (except, of course for the inputs to the algorithms which perform a reduction modulo m.)

Algorithm b = MIMOD(m,a) (Modular algorithm, reduction of an integer mod m). The inputs are L-integers m and a, where m is odd and > 1. The output is the unique L-integer b such that $b \equiv a \pmod{m}$ and $|b| < m/2$. Computing time: $\preceq L(m)L(a)$.

Algorithm B = MPMOD(m,A) (Modular algorithm, reduction of a polynomial mod m). m is an L-integer which is odd and > 1, and A is an L-integer or a univariate polynomial with L-integer coefficients. The output is the unique polynomial B with L-integer coefficients bounded by m/2 and satisfying $B \equiv A \pmod{m}$. (B is an L-integer if A is.)

Computing time: $\preceq (1+\deg A)L(m)L(|A|_\infty)$.

Algorithm B = MPLPR(m,A) (Modular polynomial list product). The inputs are m, an odd L-integer > 1, and a list $A = (A_1,\ldots,A_r)$ of polynomials over Z with $|A_i|_\infty < m/2$ for each i. The output is a polynomial $B \equiv \prod A \pmod{m}$ with $|B|_\infty < m/2$.

(1) If A is empty set $B \leftarrow 1$ and exit; otherwise, set $B \leftarrow$ first(A).

(2) Set $A \leftarrow$ tail(A). If A is empty, exit.

(3) Set $B \leftarrow$ MPMOD(m,B·first(A)) and go to (2).

Theorem MPLPR(m,A). Assume $r > 0$, $n_i = \deg A_i$, $n = n_1+\ldots+n_r$. Then the computing time for MPLPR(m,A) is $\preceq (n+1)(n+r)L(m)^2$.

Remarks: If $n_i > 0$ for all i then $n \geq r$, hence the time is $\preceq (n+1)^2L(m)^2$. If $n_i = 0$ for all i then the time is $\preceq rL(m)^2$.

The proof of Theorem MPLPR, and of other computing time theorems to follow, is given in [MUS71].

The algorithm given here is superior to the alternative of applying MPMOD only once to the product $A_1 \cdot A_2 \cdots A_r$, which has a total computing time $\preceq r(n+1)(n+r)L(m)^2$.

Algorithm b = MRECIP(m,a) (Modular algorithm, reciprocal of a mod m). The inputs are an odd L-integer $m > 1$ and an L-integer a bounded by m/2 and relatively prime to m. The output is an L-integer b bounded by m/2 and satisfying $ab \equiv 1 \pmod{m}$. The Extended Euclidean Algorithm [KNU68, p.14] is used.
Computing time: $\preceq L(m)^2$.

Algorithm L = MPQREM(m,A,B) (Modular polynomial quotient and remainder). The inputs are an odd L-integer $m > 1$ and univariate polynomials A and B with L-integer coefficients bounded by m/2; the leading coefficient of B must be relatively prime to m. The output is the list L = (Q,R) where Q and R are the quotient and remainder obtained upon dividing A by B using arithmetic modulo m; the coefficients of Q and R are bounded by m/2.
Computing time: $\preceq (k+1)(h-k+1)\ L(m)^2$ where h = deg A, k = deg B, provided $h \geq k$.

Algorithm MPSPEQ(m,A,B,S,T,U,Y,Z) (Modular polynomial solution of a polynomial equation). The inputs are:

m, an odd L-integer > 1;

A,B,S,T, univariate polynomials with L-integer coefficients bounded by m/2, satisfying $AS+BT \equiv 1 \pmod{m}$, with ldcf A relatively prime to m;

U, a univariate polynomial with L-integer coefficients.

The outputs are Y and Z, univariate polynomials with L-integer coefficients bounded by m/2 such that $AY+BZ \equiv U \pmod{m}$ and $\deg Z < \deg A$.

(1) Set $W \leftarrow \text{MPMOD}(m,U)$, $V \leftarrow \text{MPMOD}(m,TW)$.

(2) Using Algorithm MPQREM, compute $Q,Z \in Z[x]$ such that

$$V \equiv AQ+Z \pmod{m}, \quad \deg Z < \deg A,$$

$$|Q|_\infty < m/2, \quad |Z|_\infty < m/2.$$

(3) Set $Y \leftarrow \text{MPMOD}(m,SQ+BQ)$.

Theorem MPSPEQ. Assume $h = \deg A > 0$, $k = \deg B > 0$, $n = \max(h+k, \deg U)$, $\deg T < h$, $\deg S < k$, $u = |U|_1$ and $L(n) \sim 1$. Then the computing time for Algorithm MPSPEQ is

$$\leq n^2 L(m)^2 + n\, L(m)L(u).$$

3. Set Operations

In some of the factoring algorithms to be described in Section 5, a subalgorithm is required for solving the following problem: given positive integers $n_1, n_2, \ldots, n_r$ (not necessarily distinct) and an integer n, for what sets $J \subseteq \{1,\ldots,r\}$ is $\Sigma_{j \in J} n_j = n$? In the factoring algorithms, the n_i are the degrees of the irreducible factors of a polynomial and the problem is to produce all factors of a given degree n. In this section we present a simple, efficient solution to the general problem of determining the subsets J, which we shall refer to as sum index sets.

We first consider the solution to a simpler problem: is there any set $J \subset \{1,\ldots,r\}$ such that $\Sigma_{j\in J}\, n_j = n$? This problem can be attacked by constructing the <u>sumset</u> of $n_1,n_2,\ldots,n_r$, which we define to be the set of all sums $\Sigma_{j\in J}\, n_j$ such that $J \subset \{1,\ldots,r\}$; we then have merely to determine whether n is a member of the sumset of $n_1,\ldots,n_r$.

The following algorithm computes the sumset S of $n_1,\ldots,n_r$.

(1) Set $S^{(0)} \leftarrow \{0\}$. ($S^{(0)}$ is the sumset of the empty set.)

(2) For $j = 1,\ldots,r$: Set $S^{(j)} \leftarrow S^{(j-1)} \cup (S^{(j-1)}+\{n_j\})$ ($S^{(j)}$ is the sumset of $n_1,n_2,\ldots,n_j$).

(3) Set $S \leftarrow S^{(r)}$.

In step (2), $S^{(j-1)}+\{n_j\}$ is the set $\{a+n_j : a \in S^{(j-1)}\}$.

To solve the original problem of finding sum index sets, we present the following recursive algorithm.

<u>Algorithm G(j,n,J)</u> (Generation of sum index sets). Let $n_1,n_2,\ldots,n_r$ be fixed positive integers and $S^{(0)},S^{(1)},\ldots,S^{(r)}$ be sets such that $S^{(i)}$ is the sumset of $n_1,n_2,\ldots,n_r$ for $0 \le i \le r$. ($S^{(0)} = \{0\}$.) Given a nonnegative integer j, an integer $n \in S^{(j)}$, and a set J, this algorithm outputs all sets $J' \cup J$ such that $J' \subset \{1,\ldots,j\}$ and $\Sigma_{i\in J'}\, n_i = n$. (Thus, if $n \in S^{(r)}$, performing $G(r,n,\emptyset)$ causes all sets $J \subset \{1,\ldots,r\}$ such that $\Sigma_{i\in J}\, n_i = n$ to be output).

(1) If $n = 0$, output J and exit.

(2) If $n-n_j \in S^{(j-1)}$, perform $G(j-1,n-n_j,\{j\} \cup J)$.

(3) If $n \in S^{(j-1)}$, perform $G(j-1,n,J)$. Exit.

The SAC-1 implementation of these algorithms is based on the idea of representing sets of integers by bit strings. Since the binary representation of an integer is unique, there is a one-to-one correspondence between non-negative integers and finite sets of non-negative integers, defined by

$$i \longleftrightarrow J \text{ iff } i = 2^{j_1}+\ldots+2^{j_n} \text{ and } J = \{j_1,\ldots,j_n\}.$$

We shall write $J = \mathcal{J}(i)$ and $i = \mathcal{I}(J)$ when $i \longleftrightarrow J$.

If i is a non-negative integer, let

$$i = \sum_{r\geq 0} b_r^{(i)} 2^r, \quad b_r^{(i)} \in \{0,1\}$$

denote its unique binary representation. We define operations "$\wedge$" (logical product) and "$\vee$" (logical sum) on "bits" (binary digits) in the usual way, and on non-negative integers as follows:

$$i \wedge j = \sum_{r\geq 0} (b_r^{(i)} \wedge b_r^{(j)}) 2^r; \quad i \vee j = \sum_{r>0} (b_r^{(i)} \vee b_r^{(j)}) 2^r.$$

With these definitions the following identities obviously hold:

$$\mathcal{J}(i \wedge j) = \mathcal{J}(i) \cap \mathcal{J}(j),$$
$$\mathcal{J}(i \vee j) = \mathcal{J}(i) \cup \mathcal{J}(j),$$
$$\mathcal{J}(2^n i) = \mathcal{J}(i) + \{n\}.$$

Thus the computation of $S \cup (S + \{n\})$ required in the sumset algorithm discussed above can be performed by computing $i \vee (i \cdot 2^n)$, where $i \longleftrightarrow S$.

The following algorithm computes $k = i \wedge j$ from nonnegative integers i and j.

(1) Set $k \leftarrow 0$, $m \leftarrow i$, $n \leftarrow j$, $r \leftarrow 1$.

(2) If $m = 0$ or $n = 0$, exit.

(3) Set $p \leftarrow m \bmod 2$, $m \leftarrow \lfloor m/2 \rfloor$,

$q \leftarrow n \bmod 2$, $n \leftarrow \lfloor n/2 \rfloor$,

$$t \quad \begin{cases} 1 & \text{if } p = 1 \text{ and } q = 1, \\ 0 & \text{otherwise,} \end{cases}$$

$k \leftarrow rt+k$, $r \leftarrow 2r$, and go to (2).

This algorithm simply determines the bits in the binary representations of i and j and constructs k accordingly. If the arithmetic indicated is performed using the SAC-1 Integer Arithmetic System operations, then i and j may be of arbitrary size.

Most binary computers have hardware logical operations $\wedge$ and $\vee$ for single precision integers, and on such machines the above algorithm can be made more efficient: Let β be the base used in SAC-1 system (a positive integer whose base β representation is $d_n\beta^n + d_{n-1}\beta^{n-1} + \ldots + d_o$ is represented by the list $(d_o, d_1, \ldots, d_n)$; zero is represented by the empty list), and let $\ell = \lfloor \log_2 \beta \rfloor$. Replace "2" throughout the algorithm by "2^ℓ" and compute t (which will be a β-digit and therefore will be single-precision) using the hardware $/\!\!/$ operation.

In fact, on a binary machine β will generally be a power of 2 and we will thus have $2^\ell = \beta$, so that the algorithm can be rewritten using list operations in place of the arithmetic operations:

(1) Set $k \leftarrow ( )$, $m \leftarrow i$, $n \leftarrow j$.

(2) If $m = ( )$ or $n = ( )$, set $k \leftarrow$ inverse(k) and exit.

(3) Set $p \leftarrow$ first(m), $m \leftarrow$ tail(m),

$q \leftarrow$ first(n), $n \leftarrow$ tail(n),

$t \leftarrow p \wedge q$ (where"$\wedge$" here is the hardware operation),

$k \leftarrow$ prefix(t,k),

and go to (2).

The latter algorithm (named IAND) is the one which actually appears in the program listings in Section 8. This algorithm and the other logical operation algorithms, whose descriptions follow, should be considered "primitives", i.e., machine-dependent algorithms which, for efficiency, must be written especially for a particular machine, based on the characteristics of the hardware. It is assumed in this version that a hardware (or software) "$\wedge$" operation is accessible in Fortran via a function called AND. If this is not the case on a particular machine, AND will have to be programmed.

Other logical and set operation algorithms which are provided in a similar way are:

Algorithm k = IAND(i,j) (Integer AND) The inputs are non-negative L-integers i and j; the output is the L-integer $k = i \wedge j$.

Computing time: $\sim \min(L(i),L(j))$

Algorithm k = IOR(i,j) (Integer OR). The inputs are non-negative L-integers i and j; the output is the L-integer $k = i \vee j$.

Computing time: $\sim \max(L(i),L(j))$

Algorithm j = ILS(i,n) (Integer Left Shift) The inputs are an L-integer i and a non-negative Fortran integer n. The output is the L-integer $j = 2^n i$.

Computing time: $\sim L(i) + n$.

Algorithm b = MEMBER(n,i). The inputs are a Fortran integer n and a non-negative L-integer i. The output is a Fortran integer b such that if $n \in \mathcal{J}(i)$ then b = 1; otherwise, b = 0. ($b = \lfloor i/2^n \rfloor \bmod 2$.)

Computing time: $\sim \min(n+1, L(i))$.

Remarks similar to those for AND apply to the OR function which is called in IOR. For SAC-1 implementation in which β is a power of 2, the only change which might be required to the given Fortran versions of these algorithms is to change the definition of WL, the base 2 logarithm of β, in ILS and MEMBER. If β is not a power of 2, then these subprograms could be rewritten on the basis of the first algorithm given for "$\wedge$", or based on a list representation of finite sets of integers.

Algorithm S = SUMSET(N) (Sumset of N). The input N is a list $(n_1, n_2, \ldots, n_r)$ of positive Fortran integers. The output is a list $S = (i_o, i_1, \ldots, i_r)$ of L-integers i_k such that $\mathcal{J}(i_k)$ is the sumset of $n_1, \ldots, n_k$, for $k = 0, \ldots, r$. ($i_o = 1$, representing $\{0\}$, the sumset of the empty set.)

(1) Set $i \leftarrow 1$, $S \leftarrow (i)$, $N' \leftarrow N$.

(2) If N' = ( ), go to (3). Otherwise,
set $n \leftarrow$ first(N'), $i \leftarrow i \vee 2^n i$, prefix i to S, set $N' \leftarrow$ tail(N'), and repeat this step.

(3) Set $S \leftarrow$ inverse(S) and exit.

Theorem SUMSET. Assume $r > 0$ and $n = n_1 + \ldots + n_r$. Then the computing time for SUMSET(N) is $\sim \sum_{j=1}^{r} (n_1 + \ldots + n_j) \leq rn$.

We now come to the implementation of Algorithm G. This algorithm outputs all "sum index sets" satisfying certain criteria. Here we shall be generating sum index lists J and we could produce as output

a list of all sum index lists which satisfy the given criteria. Since, however, we may not use all of the sum index lists generated, this would be wasteful of time and storage. Thus we shall set up the algorithm so that each sum index list generated is made available immediately. Also we shall show the stacking mechanism necessary to implement recursion in a language such as Fortran which does not allow explicit recursion.

Algorithm GEN (STACK, N,S,n,k,J) (Generate sum index lists). The inputs are:

STACK, a first order list (explained below);

N, a nonempty list $(n_1,\ldots,n_r)$ of positive Fortran integers;

S, a nonempty list $(i_o,\ldots,i_r)$ of L-integers such that

$\mathcal{S}(i_k)$ is the sumset of $n_1,\ldots,n_k$, for $0 \leq k \leq n$;

n, a Fortran integer;

k, a Fortran integer satisfying $1 \leq k \leq r$.

The outputs are STACK and J, a list of indices $(j_1,\ldots,j_v)$ such that

$$1 \leq j_1 < \ldots < j_v \leq k \text{ and } n_{j_1} + \ldots + n_{j_v} = n. \qquad (*)$$

With fixed values for N, S and n, repeatedly performing GEN will yield all lists $J = (j_1,\ldots,j_v)$ which satisfy (*). The algorithm must be performed initially with STACK = ( ). When all such lists have been generated the values of STACK and J will be ( ). If there are no such lists then STACK = ( ) and J = ( ) will be output the first time the algorithm is performed.

In the algorithm, by "stack j" we mean "prefix j to STACK", and by "unstack j" we mean "set j$\leftarrow$ first(STACK), STACK$\leftarrow$ tail(STACK)".

(1) If STACK $\neq$ ( ), unstack j, set n$\leftarrow$ 0 and go to (6).

(2) [Initialize.] Set j$\leftarrow$ length(N), J$\leftarrow$ ( ), N'$\leftarrow$ inverse(N) S'$\leftarrow$ inverse(S'), R$\leftarrow$ 10, and, while j > k, repeat the following: set N'$\leftarrow$ tail(N'), S'$\leftarrow$ tail(S') and j$\leftarrow$ j-1. (Steps (3)-(9) correspond to the recursive Algorithm G(j,n,J); this step has initialized to perform the algorithm with j = k and J = ( ).)

(3) [Output?] If n = 0, stack j, and exit. (This exit allows the current value of J to be used outside the algorithm; the algorithm may be reentered to generate another sum index list, provided that neither STACK nor J is altered outside the algorithm.)

(4) [Recursion necessary?] If n-first(N') $\notin \mathcal{D}$(second(S')), go to (7).

(5) [Perform G(j-1,n-n_j,$\{j\} \cup$ J).] Stack N',S',R, set n$\leftarrow$ n-first(N') N'$\leftarrow$ tail(N'), S'$\leftarrow$ tail(S'), R$\leftarrow$ 6, J$\leftarrow$ prefix(j,J), j$\leftarrow$ j-1, and go to (3).

(6) Unstack R,S',N', set j$\leftarrow$ j+1, n$\leftarrow$ n+first(N'), J$\leftarrow$ tail(J).

(7) [Recursion necessary?] If n $\notin \mathcal{D}$(second(S')), go to (R).

(8) [Perform G(j-1,n,J).] Stack N',S',R, set N'$\leftarrow$ tail(N'), S'$\leftarrow$ tail(S'), R$\leftarrow$ 9, j$\leftarrow$ j-1, and go to (3).

(9) Unstack R,S',N', set j$\leftarrow$ j+1, and go to (R).

(10) Exit (this exit is taken when all sum index lists have been generated).

Computing time: The time for each execution of GEN (producing one sum index list J) is $\leq$ rs, where r = length(N), s = ΣN.

The order in which Algorithm GEN produces the sum index lists will be important in the application made in Section 5.3 (Algorithm PFP1). Let $J = (j_1,\ldots,j_v)$ and $K = (k_1,\ldots,k_w)$ be sum index lists output by the algorithm. Then

$$j_v > k_w \Rightarrow J \text{ precedes } K \text{ (in order of output).}$$

This may be seen from the fact that the algorithm generates all index lists which end with the highest index before those which do not. (In fact the order of output could be completely characterized by saying that the inverses of the sum index lists appear in reverse lexicographical order).

We conclude this section with a description of an algorithm which will be used in conjunction with Algorithm GEN.

Given an arbitrary list $A = (a_1,\ldots,a_m)$ and a list $I = (i_1,\ldots,i_n)$ of integers satisfying $1 \le i_1 \le \ldots \le i_n \le m$, we define $A_I = (a_{i_1},\ldots,a_{i_n})$. ($A_I = ( )$ if $A = ( )$).

Algorithm B = SELECT(A,I) (Select A_I from A). The inputs are lists A and I as described above (the integers in I being Fortran integers); the output is the list $B = A_I$. Those elements of A which are lists are borrowed for use in B.

Computing time: $\sim i_n + 1$.

4. Factor Coefficient Bounds

In [MUS71, Sec. 3.4] we derived several bounds on the integer coefficients of factors of a given polynomial over the integers. Most of these bounds are based on the theory of Lagrange interpolating polynomials, particularly the following two theorems:

Theorem D. Let $L_j^{(m)}(x)$, $-m \le j \le m$, be the Lagrange interpolating polynomials for the points $-m,\ldots,0,\ldots,m$, where $m \ge 1$. (Thus

$$L_j^{(m)}(x) = \prod_{\substack{i=-m \\ i \neq j}}^{m} \frac{x - i}{j - i} .)$$

Then

a. $$\left| L_j^{(m)} \right|_1 = \frac{|j| + 1}{j^2 + 1} \cdot \frac{\prod_{i=1}^{m} (i^2 + 1)}{(m + j)!\ (m - j)!} ;$$

b. $$\left| L_o^{(m)} \right|_1 < 4 ;$$

c. $$\sum_{j=-m}^{m} \left| L_o^{(m)} \right|_1 < 4\sqrt{\pi m} .$$

Theorem E. Let $C \in Z[x]$ and let A be a factor of C of degree k. Assume $C(j) \neq 0$ for all integers j such that $|j| \le m = \lceil k/2 \rceil$. Then

a. $$|A|_1 \le \sum_{j=-m}^{m} |C(j)| \left| L_j^{(m)} \right|_1 ;$$

b. $$|A|_1 \le 4 \sum_{j=-m}^{m} |C(j)| ;$$

c. $$|A|_1 \le 4\sqrt{\pi m} \max_{|j| \le m} |C(j)| .$$

The bound given by part b of Theorem E is simplest to compute, but we shall now show that only a bit more computational effort is required for the bound in part a.

Let m be fixed, let $P = \prod_{i=1}^{m} (i^2+1)$, and let $f_j = P/[(m+j)!(m-j)!]$. From Theorem D, part a,

$$\left|L_j^{(m)}\right|_1 = \frac{|j|+1}{j^2+1} f_j.$$

It is easy to compute f_j from f_{j-1}:

$$f_j = \frac{P\,(m-j+1)}{(m+j)(m+j-1)!(m-j+1)!} = \frac{m-j+1}{M+J} f_{j-1}.$$

Hence the following algorithm computes $b = \sum_{j=-m}^{m} |C(j)| \left|L_j^{(m)}\right|$ exactly:

(1) Set $f \leftarrow \prod_{i=1}^{m}(i^2+1)/(m!)^2$, $b \leftarrow f \cdot |C(0)|$, $j \leftarrow 1$.

(2) (Now $b = \sum_{i=-j+1}^{j-1} |C(i)| \left|L_j^{(m)}\right|$, $f = f_{j-1}$.) If $j > m$, exit.

(3) Set $f \leftarrow (m-j+1)f/(m+j)$,

$b \leftarrow b + (j+1)f[|C(-j)| + |C(j)|]/(j^2+1)$,

$j \leftarrow j+1$, and go to (2).

This algorithm is the basis of the bounding algorithm PFB1 which we shall now describe. The main modification is to arrange the computation so that only integer arithmetic is required, rather than rational arithmetic. This gives a somewhat larger bound, which is nevertheless smaller than that given by part b of Theorem E. It is also necessary to check for cases in which C(i) = 0 for one or more of the points i used, and to remove linear factors (x-i) in such cases.

Algorithm PFB1 (C_o, m, C, L, b). (Polynomial factor bounding algorithm, 1 variable) The inputs are a polynomial C_o over Z of positive degree and a positive Fortran integer m. The outputs are a polynomial C over Z, a list L of linear polynomials such that $C_o = C \amalg L$,

and a positive L-integer b such that if A is any factor of C of degree $\leq$ m, then $|A|_1 \leq b$. (Possibly $C = C_o$ and $L = ( )$.)

(1) Set $C \leftarrow C_o$, $L \leftarrow ( )$.

(2) If deg C = 1, set $b \leftarrow |C|_1$ and exit.

(3) Compute C(0). If $C(0) \neq 0$, go to (5). Otherwise, set $j \leftarrow 0$.

(4) Set $C(x) \leftarrow C(x)/(x-j)$, prefix x-j to L, and go to (2).

(5) Set $f \leftarrow 4$, $b \leftarrow f \cdot |C(0)|$, $j \leftarrow 1$.

(6) (Now $b \geq \sum_{i=-j+1}^{j-1} |C(i)| \; |L_j^{(m)}|_1$, $f \geq f_{j-1}$, $C_o = C \bmod L$ and $C(i) \neq 0$ for $|i| \leq j-1$.) If $j > m$, exit.

(7) Compute C(j). If C(j) = 0, go to (4).

(8) Compute C(-j). If C(-j) = 0, set $j \leftarrow -j$ and go to (4).

(9) Set $f \leftarrow \lceil (m-j+1)f/(m+j) \rceil$,

$b \leftarrow b + \lceil ((j+1)f[|C(-j)| + |C(j)|])/(j^2+1) \rceil$,

$j \leftarrow j+1$, and go to (6).

Note: The Fortran version of this algorithm is written with the further assumption that $m^2 + 1 < \beta$, where β is the base used in the SAC-1 integer arithmetic system.

Theorem PFB1. Let n = deg C_o and γ be a bound on the norm ($|\ |_1$) of any factor of C_o. Let k be the number (counting multiplicity) of linear factors x-i of C_o such that $|i| \leq m$. Assume $L(n) \sim 1$, $L(m) \sim 1$. Then the computing time for Algorithm PFB1 is

$$(k+1)\ m[n^2+nL(\gamma)].$$

5. Main Algorithms

5.1. Algorithm PSFREE

The first factoring algorithm to be described is an algorithm for finding the squarefree factors of a given primitive polynomial over Z. As defined in Section 1.2, a polynomial over Z is squarefree if it is the product of distinct irreducible factors.

The following algorithm is based on Algorithm 2.4S of [MUS71] and is similar to Horowitz' algorithm PSQFRE in [COL70a]. Either algorithm could be used to obtain a squarefree factorization, but PSFREE is probably somewhat faster when the input is not already squarefree (if the input is squarefree, the two algorithms perform essentially the same computation). Also, the inclusion of PSFREE in the Polynomial Factorization System makes this system independent of the Rational Function Integration System [COL70a].

Algorithm L = PSFREE(A) (Polynomial squarefree factorization) The input A is a non-constant, primitive, positive polynomial over Z. The output is a list $L = (A_1, A_2, \ldots, A_t)$ where $A = A_1 A_2^2 \ldots A_t^t$, each A_i is a primitive, squarefree, positive polynomial over Z, $\deg(A_t) > 0$, and $A_1, A_2, \ldots, A_t$ are pairwise relatively prime. (L = (A) if A is itself squarefree).

(1) Set $L \leftarrow ( )$, $B \leftarrow \gcd(A, A')$, $C \leftarrow A/B$.

(2) If B = 1, prefix C to L, invert L, and exit.

(3) Set $D \leftarrow \gcd(B, C)$ and prefix C/D to L. (Now $B = A_{j+1} A_{j+2}^2 \ldots A_t^{t-j}$,

$C = A_j A_{j+1} \cdots A_t$, $D = A_{j+2} \cdots A_t$ and $L = (A_j, \ldots, A_1)$, where j is the number of times this step has been executed.)

(4) Set $B \leftarrow B/D$, $C \leftarrow D$, and go to 2.

The gcd calculations in steps (1) and (3) should be performed using the fast modular PGCD algorithm of the Polynomial G.C.D. and Resultant System [COL72], rather than the (reduced p.r.s.) algorithm in the original Polynomial system. (It should be noted that the PSQFRE algorithm of [COL70a] erroneously refers to PGCD1; this should be changed to PGCD if PSQFRE is used instead of PSFREE.)

Theorem PSFREE. Let $n = \deg(A)$ and γ be a bound on $|B|_1$ for any factor B of A. Then the computing time for PSFREE is

$$\leq tn^3 L(\gamma)^2.$$

5.2. Algorithm PFH1

The algorithm to be described in this section is based on a modified "Hensel's Lemma" construction due to Zassenhaus. Given a polynomial over the integers and a factorization modulo some integer prime p, the algorithm computes factorizations modulo $p^2, p^4, p^8, p^{16}, \ldots$ in successive iterations until a desired power of p has been reached. The algorithm is basically similar to one for Hensel's construction given in [KNU69, Section 4.6.2].

In the algorithm, we use the notation h_p for the induced homomorphism of Z[x] onto GF(p)[x], as defined in Section 1.1.

Algorithm PFH1 (p,m,C,$\bar{A}$,$\bar{B}$,$\bar{S}$,$\bar{T}$,A,B) (Polynomial factorization based on the Quadratic Hensel Algorithm, 1 variable). The inputs are:

p, an odd positive prime integer (Fortran integer);

$m = p^j$ for some positive integer j (m is an L-integer);

C, a primitive positive polynomial over Z;

$\overline{A},\overline{B},\overline{S},\overline{T}$, polynomials over GF(p) such that

$$h_p(C) = \overline{AB} \text{ and } \overline{AS} + \overline{BT} = 1.$$

The outputs are polynomials A,B over Z such that

$C \equiv AB \pmod{m}$;

$h_p(A) = \overline{A}$, $h_p(B) = \overline{B}$;

$|\text{ldcf } A| < p/2$; $|A|_\omega$, $|B|_\omega < m/2$.

Note: The conditions $h_p(A) = \overline{A}$ and $|\text{ldcf } A| < p/2$ imply that ldcf A is a unit mod m and deg A = deg $\overline{A}$.

(1) [Initialize.] Set $q \leftarrow p$ and obtain $A,B,S,T \in Z[x]$ such that $h_p(A) = \overline{A},\ldots,h_p(T) = \overline{T}$, $|A|_\omega < p/2,\ldots,|T|_\omega < p/2$. (This may be done conveniently using Algorithm CPGARN described in [COL69a]).

(2) [Done?] If q = m, exit. (This exit is taken only if m = p.)

(3) [Compute Y,Z.] (Now $A,B,S,T \in Z[x]$, $C \equiv AB$ and $AS+BT \equiv 1 \pmod{q}$, $h_p(A) = \overline{A}$, $h_p(B) = \overline{B}$, $|\text{ldcf } A| < p/2$, $|A|_\omega$, $|B|_\omega$, $|S|_\omega$, $|T|_\omega < q/2$.) Set $U \leftarrow (C-AB)/q$. If $q^2 > m$, set $\breve{q} \leftarrow m/q$, $\breve{A} \leftarrow \text{MPMOD}(\breve{q},A),\ldots,\breve{T} \leftarrow \text{MPMOD}(\breve{q},B)$; otherwise, set $\breve{q} \leftarrow q$, $\breve{A} \leftarrow A,\ldots,\breve{T} \leftarrow T$. (Now $|\breve{A}|_\omega,\ldots,|\breve{T}|_\omega < \breve{q}/2$.) Apply Algorithm MPSPEQ to $\breve{q},\breve{A},\breve{B},\breve{S},\breve{T},U$, obtaining $Y,Z \in Z[x]$ such that $\breve{A}Y + \breve{B}Z \equiv U \pmod{\breve{q}}$, $|Y|_\omega < \breve{q}/2$, $|Z|_\omega < \breve{q}/2$ and deg Z < deg A.

(4) [Compute A*, B* and check for end.] Set $A^* \leftarrow A+qZ$, $B^* \leftarrow B+qY$.

(Then $C \leftarrow A^*B^* \pmod{q\breve{q}}$; $h_p(A^*) = \bar{A}$, $h_p(B^*) = \bar{B}$, $|\mathrm{ldcf}\ A^*| < p/2$, and $|A^*|_\infty$, $|B^*|_\infty < q\breve{q}/2$.) If $q^2 \geq m$ (in which case $q\breve{q} = m$), set $A \leftarrow A^*$, $B \leftarrow B^*$ and exit.

(5) [Compute Y_1, Z_1.] Set $U_1 \leftarrow (A^*S+B^*T-1)/q$. Apply Algorithm MPSPEQ to q, A, B, S, T, U_1, obtaining $Y_1, Z_1 \in Z[x]$ such that $AY_1+BZ_1 \equiv U_1 \pmod{q}$, $|Y_1|_\infty$, $|Z_1|_\infty < q/2$ and $\deg Z_1 < \deg Z$.

(6) [Compute S^*, T^*.] Set $S^* \leftarrow S-qY_1$, $T^* \leftarrow T-qZ_1$. (Then $A^*S^*+B^*T^* \equiv 1 \pmod{q^2}$, $|S^*|_\infty$, $|T^*|_\infty < q^2/2$).

(7) [Advance.] Replace q, A, B, S, T by q^2, A^*, B^*, S^*, T^*, and go to (3).

Theorem PFH1. Assume in Algorithm PFH1 that $\deg \bar{A} > 0$, $\deg \bar{B} > 0$, $\deg \bar{S} < \deg \bar{A}$, $\deg \bar{T} < \deg \bar{B}$, $n = \deg C$, $L(n) \geq 1$, and $c = |C|_1$. Then the computing time for the algorithm is

$$n^2 L(n)^2 + nL(m)L(c).$$

5.3. Algorithm PFC1

Given a polynomial C(x) over the integers and a mod p factorization into several factors, we can obtain, by repeated application of Algorithm PFH1, a corresponding factorization mod p^j. That is, each mod p^j factor obtained will correspond to one of the mod p factors. This factorization is obtained in the following algorithm.

Algorithm F = PFC1 (p,m,C,G) (Polynomial factorization, construction of mod p^j factors, 1 variable).

The inputs are:

p, an odd positive prime integer (Fortran integer);

$m = p^j$ for some positive integer j (m is an L-integer);

C, a primitive positive polynomial over Z such that $p \nmid \text{ldcf } C$ and $h_p(C)$ is squarefree over GF(p);

$G = (G_1,\ldots,G_r)$ where $r \geq 2$, each G_i is a monic polynomial over GF(p), of positive degree, and

$h_p(C) \equiv (\text{ldcf } h_p(C))\, G_1 \ldots G_r$.

The output is a list $F = (F_1,\ldots,F_r)$ of polynomials over Z such that

$C = (\text{ldcf } C)\, F_1 \ldots F_r \pmod m$

$h_p(F_i) = G_i$, $\deg F_i = \deg G_i$, F_i is monic, $|F_i|_\omega \leq m/2$, $i = 1,\ldots,r$.

(1) Set $\overline{C} \leftarrow h_p(C)$, $G' \leftarrow G$, $F \leftarrow ( )$.

(2) Set $\overline{A} \leftarrow \text{first}(G')$, $G' \leftarrow \text{tail}(G')$, $\overline{B} \leftarrow \overline{C}/\overline{A}$.

(3) Using Algorithm CPEGCD, obtain $\overline{S}$ and $\overline{T}$ over GF(p) such that $\overline{AS} + \overline{BT} = 1$.

(4) Apply Algorithm PFH1 to $p,m,C,\overline{A},\overline{B},\overline{S},\overline{T}$ and let A and B be the output.

(5) Prefix A to F and set $C \leftarrow B$, $\overline{C} \leftarrow \overline{B}$.

(6) If $\text{tail}(G') \neq ( )$, go to (2).

(7) Set $\breve{c} \leftarrow \text{MRECIP}(m,\text{ldcf } C)$, $A \leftarrow \text{MPMOD}(m,\breve{c}\cdot C)$. Prefix A to F, invert F, and exit.

Theorem PFC1. Let $n = \deg C$, $c = |C|_1$, $r = \text{length}(G)$ and assume $L(n) \sim 1$. Then the computing time for Algorithm PFC1 is

$$\leq rn^2 L(m)^2 + n\, L(m)L(c).$$

5.4. Algorithm PFP1

If Algorithm PFC1 of the previous section is applied to complete mod p factorization (factorization into irreducibles) of the given polynomial C, the corresponding factorization $F = (F_1,F_2,\ldots,F_t)$

obtained is a "complete factorization mod p^j" of C, in a sense which is precisely defined in [MUS71, p. 67]. For the purpose at hand, it suffices to note that this implies that $C \equiv (\text{ldcf } C)\prod F \pmod m$ and that to every factorization C = AB over the integers there corresponds a unique subset H of $\{F_1,F_2,\ldots,F_r\}$ such that $A \equiv (\text{ldcf } A)\prod H \pmod{p^j}$.

In the statement of the algorithm we use the notation N_J, where N and J are lists, as defined at the end of Section 3.

Algorithm F = PFP1 (m,C,G,C) (Polynomial factorization of a primitive polynomial, 1 variable). The inputs are:

m = p^j for some positive prime integer p and positive integer j (m is an L=integer);

C, a non-constant, primitive, positive polynomial over Z such that m/2 bounds the coefficients of any factor A* of C* = (ldcf C) C for which deg A* $\leq \lfloor$(deg C)/2$\rfloor$ and ldcf A* $|$ ldcf C;

G, a list $(G_1,\ldots,G_r)$ of monic polynomials such that

a. $C \equiv (\text{ldcf } C)\prod G \pmod m$,

b. for every factorization C = AB over Z there is a unique subset H of $\{G_1,\ldots,G_r\}$ such that A = (ldcf A)$\prod H$

(these conditions, as noted above, are satisfied by the list output by PFC1, when applied to a complete factorization of $h_p(C)$ over GF(p));

D, a positive L-integer representing a set of positive integers which contains the set $\{d: d = \text{deg } A,\ A|C,\ 0 < d \leq \lfloor(\text{deg } C)/2\rfloor\}$.

The output of the algorithm is a list F of the prime positive polynomials over Z such that $C = \pi F$.

(1) Set $F \leftarrow ( )$; $d \leftarrow 1$; $N \leftarrow (n_1,\ldots,n_r)$, where $n_i = \deg G_i$; $T \leftarrow (t_1,\ldots,t_r)$, where $t_i = \text{tlcf } G_i$, $k \leftarrow r \leftarrow \text{length}(G)$.

(2) Set $c \leftarrow \text{ldcf } C$, $C^* \leftarrow c \cdot C$, $t^* \leftarrow \text{tlcf } C^*$, $S \leftarrow \text{SUMSET}(N)$, $\mathcal{D} \leftarrow D \wedge \text{last}(S)$.

(3) If $d > \lfloor(\deg C)/2\rfloor$, prefix C to F and exit. If $d \notin \mathcal{D}$ or $k = 0$, go to (7).

(4) Using Algorithm GEN, generate a new list $J = (j_1,\ldots,j_v)$ such that $1 \leq j_1 < \ldots < j_v \leq k$ and $\Sigma N_J = d$. If all such lists have already been found, go to (7).

(5) Set $t \leftarrow \text{MPLPR}(m,\text{prefix}(c,T_J))$. If $t \nmid t^*$, go to (4).

(6) Set $A^* \leftarrow \text{MPLPR}(m,\text{prefix}(c,G_J))$. If $A^* \nmid C^*$, go to (4). Otherwise, set $B^* \leftarrow C^*/A^*$ and go to (8).

(7) Set $d \leftarrow d+1$, $k \leftarrow r$ and go to (3).

(8) Set $A \leftarrow \text{pp}(A^*)$, prefix A to F, and set $C \leftarrow B^*/\text{ldcf } A$. Construct $K = (k_1,\ldots,k_w)$ such that $1 \leq k_1 < \ldots < k_w \leq r$, $\{j_1,\ldots,j_v\} \cap \{k_1,\ldots,k_w\} = \emptyset$ and $\{j_1,\ldots,j_v\} \cup \{k_1,\ldots,k_w\} = \{1,\ldots,r\}$. Set $G \leftarrow G_K$, $N \leftarrow N_K$, $T \leftarrow T_K$, $r \leftarrow r-v$, $k \leftarrow j_v-v$, and go to (2).

<u>Theorem PFP1</u>. Let $C = C_1C_2\ldots C_e$, $d_i = \deg C_i$, $1 \leq d_1 \leq \ldots \leq d_e$, $n = \deg C$, and

$$\mu = \begin{cases} \max\{d_{e-1}, \lfloor d_e/2\rfloor\} & \text{if } e > 1. \\ \lfloor n/2\rfloor & \text{if } e = 1. \end{cases}$$

Let δ be the number of products $P = \prod G_j$ such that $J = (j_1, \ldots, j_v)$, $1 \le j_1 < \ldots < j_v \le r = \text{length}(G)$, and $1 \le \deg P \le \mu$, and let γ be the number of these products satisfying the additional condition that MIMOD(m,(ldcf C)(tlcf P)) is a divisor of (ldcf C)(tlcf C). Finally, let γ be a bound on $|A|_1$ for any factor A of C. Then the computing time for PFP1 is

$$\le [(\gamma+1-e)\mu n^2 + \gamma\mu^2 + \delta\mu]\, L(m)^2$$
$$+ (\gamma\mu n + \delta)\, L(m)L(\gamma) + \delta r n$$
$$\le \delta\mu n\, L(m)[n\, L(m) + L(\gamma)]$$

where $\delta \le \min(2^r, r^\mu)$.

5.5. Algorithm PFZ1

In this algorithm, factorizations modulo several primes are considered, and the prime p which yields the fewest irreducible factors is chosen for input to the Hensel algorithms. This is superior to use of a single prime, since it reduces the probability that the mod p factorization will have many more irreducible factors than the integer factorization. This is of critical importance since the computing time for Algorithm PFP1 can be an exponential function of the number of irreducible factors mod p.

Also, important information is obtained from the mod p factorizations about the possible degrees of factors of the input polynomial C. The set of degrees of factors of C must be contained in the set D_p of degrees of mod p factors for any prime p, and therefore must be contained in $D_{p_1} \cap D_{p_2} \cap \ldots \cap D_{p_\nu}$ where $p_1, p_2, \ldots, p_\nu$

are the primes for which factorizations are carried out. D_p is just the sumset of the list of degrees of the irreducible factors mod p, and is thus easily computed using Algorithm SUMSET (Section 3). If C is irreducible then we will often find

$$D_{p_1} \cap D_{p_2} \cap \ldots = \{0, \deg C\}$$

after a few primes $p_1, p_2, \ldots$ have been tried, thus proving irreducibility without ever having to apply the Hensel algorithms.

Two algorithms, which are implemented in the SAC-1 Modular Arithmetic System [COL69a] are used for the mod p factorizations. CPBERL implements Berlekamp's algorithm and computes the complete squarefree factorization of any monic squarefree polynomial A over GF(p). The computing time for CPBERL is $\doteq pn^3$ (assuming $L(p) \sim 1$).

For the purposes of finding a prime which gives few irreducible factors and of computing the degree sets D_p, it is not actually necessary to obtain the complete factorization modulo each of the primes tested. All that is necessary is a list of the degrees of the irreducible factors and this can be obtained from the output of Algorithm CPDDF (distinct degree factorization). Given a monic squarefree polynomial A over GF(p) to be factored, CPDDF produces a list $((d_1,A_1),\ldots,(d_s,A_s))$ where the d_i are positive integers, $d_1 < d_2 < \ldots < d_s$, and A_i is the product of all monic irreducible factors of A which are of degree d_i. Thus $A = A_1 \ldots A_s$ and this is a complete factorization just in case no two irreducible factors of A have the same degree.

The computing time for CPDDF is $\leq n^3$ if $L(p) \sim 1$; hence this algorithm is practicable for much larger primes than CPBERL. Even for small primes it appears to be about 20 per cent faster, according to empirical results. Therefore this algorithm is used to obtain the lists of the degrees of irreducible factors over GF(p) for several primes in the first phase of Algorithm PFZ1, and CPBERL is applied only to the single prime selected and the reducible factors output by CPDDF for that prime.

Algorithm F = PFZ1(C) (Polynomial factorization based on Algorithm 2.7.1Z, 1 variable). The input C is a non-constant, primitive, squarefree, positive polynomial over Z. Two other inputs, ν and SPRIME, are required in COMMON block TR5 (COMMON/TR5/NU,SPRIME). ν is a positive Fortran integer which specifies the maximum number of primes for which mod p factorizations should be tried. SPRIME is a list of small odd positive prime integers (Fortran integers). This list should be of length no less than, say, $\min(20,2\nu)$, so that it will not be exhausted in practical application (if the list is exhausted then the algorithm terminates with no output.) Such a list can be conveniently generated using subprogram GENPR of the Modular Arithmetic System.

The output of the algorithm, provided the list SPRIME is not exhausted, is a list F of the prime positive polynomials over Z such that $C = \prod F$.

(1) [Initialize for factoring modulo ν different primes.] Set SP $\leftarrow$ SPRIME, RMIN $\leftarrow$ deg C + 1, NP $\leftarrow$ 1, D $\leftarrow$ $2^{d^*+1}-1$, where $d^* = \lfloor(\deg C)/2\rfloor$. (D represents the set $\{0,1,\ldots,d^*\}$).

(2) [Get next prime.] If SP = ( ), stop. Otherwise, set $p \leftarrow$ first(SP), SP $\leftarrow$ tail(SP).

(3) [h_p(ldcf C) = 0?] If $p \mid$ ldcf C, go to (2).

(4) [h_p(C) squarefree?] Set $\overline{C} \leftarrow h_p(C)$, $B \leftarrow \gcd(\overline{C},\overline{C}')$. If deg B $\neq$ 0, go to (2). (We have deg B = 0 iff $\overline{C}$ is squarefree.)

(5) [Apply CPDDF.] Apply Algorithm CPDDF to the monic associate of $\overline{C}$, obtaining a list $G = ((d_1,A_1),\ldots,(d_s,A_s))$, where the d_i are positive integers, $d_1 < \ldots < d_s$, and A_i is the product of all prime monic factors of $\overline{C}$ which are of degree d_i. (Thus $\overline{C}$ = (ldcf $\overline{C}$) $A_1 \ldots A_s$.)

(6) [Construct $N = (n_1,\ldots,n_k)$, where the n_i are the degrees of the prime factors of $\overline{C}$.] Set N $\leftarrow$ ( ) and for each (d,A) on G, prefix d to N, (deg A)/d times.

(7) [$\overline{C}$ prime?] Set r $\leftarrow$ length(N). If r = 1, set F $\leftarrow$ (C) and exit.

(8) [Compute sumset.] Set S $\leftarrow$ SUMSET(N), D $\leftarrow$ D $\wedge$ last(S). If D = 1, set F $\leftarrow$ (C) and exit.

(9) [New minimum number of factors?] If r < RMIN, set RMIN $\leftarrow$ r, $p^* \leftarrow$ p, $G^* \leftarrow$ G.

(10) [ν factorizations tried?] Set NP $\leftarrow$ NP + 1. If NP $\leq \nu$, go to (2).

(11) [Pick prime which yields minimum number of factors.] Set $p \leftarrow p^*$, $G \leftarrow G^*$, H $\leftarrow$ ( ).

(12) [Obtain complete factorization of $h_p(C)$.] For each (d,A) on G: if d = deg A, prefix A to H; otherwise apply Algorithm CPBERL to p and A, obtaining a list $\breve{F}$ of the prime monic factors of A, and concatenate $\breve{F}$ with H. Then set G ← inverse(H).

(13) [Obtain bound.] Set d ← $\lfloor \log_2 D \rfloor$ (d is then the maximum of the degrees in the set represented by D) and apply Algorithm PFB1 to C and d, obtaining C* over Z, a list F of monic linear polynomials such that C = C*$\prod$F and an integer b such that $|A|_1 \leq b$ for any factor A of C* of degree $\leq$ d. If deg C* = 1, prefix C* to F and exit. Otherwise, set C ← C* and, for each linear factor A on F, set $\bar{A} \leftarrow h_p(A)$ and search for and remove $\bar{A}$ from G. If length(G) = 1, prefix C to F and exit.

(14) [Compute modulus.] Set m ← p, q ← 2b(ldcf C). While m $\prec$ q repeat: set m ← mp.

(15) [Apply PFC1.] Apply Algorithm PFC1 to p,m,C,G, obtaining a list $F^{(1)} = (F_1^{(1)},\ldots,F_t^{(1)})$ of polynomials over Z such that

$$C \equiv (\text{ldcf } C)\, F_1^{(1)} \cdots F_t^{(1)} \pmod m,$$

$$h_p(F_i^{(1)}) = G_i,\ \deg F_i^{(1)} = \deg G_i,$$

$F_i^{(1)}$ is monic, and $|F|_1^{(1)} \leq m/2$, i = 1,...,t.

(16) [Apply PFP1.] Apply Algorithm PFP1 to m,C,$F^{(1)}$,D, obtaining a list $F^{(2)}$ of the prime positive polynomials over Z such that C = $\prod F^{(2)}$. Concatenate $F^{(2)}$ with F and exit.

As we remarked earlier, if C is irreducible (and ν is sufficiently large), often only the first phase of the algorithm, steps (1)-(10), will be performed. Therefore we give separate

consideration in the following computing time analysis to the two phases.

Theorem PFZ1. a. Let $p_1 \le p_2 \le \ldots \le p_\nu$ be the ν smallest odd positive prime integers such that $p_i \nmid$ ldcf C and C is squarefree modulo p_i for $1 \le i \le \nu$; and θ be the number of odd positive primes $\le p_\nu$. Also, let $c = |C|_1$, $n = \deg C$, and assume $L(p_\nu)$, $L(n) \sim 1$. Then the computing time for steps (1)-(10) of Algorithm PFZ1 is

$$\le \nu n^3 + \theta n^2 + \theta n L(c).$$

b. Let $r = \min_{1 \le i \le \nu} N(p_i, C)$,

where $N(p,C)$ is the number of irreducible factors of C modulo p; k be the number of linear factors x-i of C such that $|i| \le |n/2|$; γ and μ be defined as in Theorem PFP1 and m be the integer computed in step (13). Then the computing time for steps (11)-(16) of Algorithm PFZ1 is

$$\le p_\nu n^3 + (k+1)n^2[n+L(\gamma)] + \min(2^r, r^\mu)\,\mu n L(m)[nL(m) + L(\gamma)].$$

c. $p_\nu \le \nu + n\,L(c)$.

d. $L(m) \le n+L(\gamma)$.

e. Hence the time for steps (1)-(10) is

$$\le n^2 L(c)^2 + (n^3+\nu n)L(c) + \nu n^3,$$

and for steps (11)-(15) is

$$\le \nu n^3 + n^4 L(c) + \min(2^r, r^\mu)\mu n^2 (n+L(\gamma))^2.$$

and thus for the entire algorithm the time is

$$\le (n^4+\nu n)L(c) + \nu n^3 + \min(2^r, r^\mu)\mu n^2 (n+L(\gamma))^2.$$

f. If we assume $p_\nu \sim 1$ (i.e., if we restrict the set of inputs C to those for which p_ν is no greater than some fixed bound) then

the time for steps (1)-(10) is $\preceq n^3 + n\,L(c)$ and for steps (11)-(15) is

$$\preceq \min(2^r, r^\mu)_{\mu} n^2 (n + L(\gamma))^2;$$

the latter bound is a bound for the entire algorithm.

Remark: The assumption in f is actually made in the algorithm since the list SPRIME of small primes used by the algorithm is of finite length.

5.6. Algorithm PFACT1

The final algorithm which we shall describe is PFACT1, which factors an arbitrary nonzero polynomial A over the integers. PFACT1 first factors out the content of A using Algorithm PCPP of the SAC-1 Polynomial System. The content itself is not factored into prime integers by the algorithm since this may not be necessary in some applications. The primitive part of A is factored into squarefree polynomials using Algorithm PSFREE. Finally, Algorithm PFZ1 is used to factor each of the squarefree factors of degree > 1 into prime factors.

Algorithm F = PFACT1 (A) (Polynomial factorization, 1 variable). The input A must be a non-constant polynomial over Z. The output is a list $F = (c, A_1, A_2, \ldots, A_r)$, where c is the content of A and $A_1, A_2, \ldots, A_r$ are the unique prime positive factors of pp(A). Note that $c < 0$ if ldcf $A < 0$. (c is an L-integer).

(1) Using Algorithm PCPP obtain the positive content c and the positive primitive part P of A. If ldcf $A < 0$, set $c \leftarrow -c$. Then set $F \leftarrow (c)$, $i \leftarrow 1$, $Q \leftarrow$ PSFREE(P). (Then $Q = (Q_1, \ldots, Q_t)$, where

$P = Q\ Q_2^2 \cdots\ Q_t^t$, $\gcd(Q_i,Q_j) \sim 1$ for $i \neq j$, and each Q_i is a primitive squarefree positive polynomial.)

(2) Set $C \leftarrow$ first(Q), $Q \leftarrow$ tail(Q). If deg C = 0, go to (4). If deg C = 1, prefix C to F, i times.

(3) Set $\overline{F} \leftarrow$ PFZ1(C), and for each E on the list $\overline{F}$, prefix E to F, i times.

(4) Set $i \leftarrow i+1$. If $Q \neq ( )$, go to (2). Otherwise, invert F and exit.

Theorem PFACT1. Let n = deg A, γ be a bound on $|B|_\omega$ for any factor B of A, and $A = Q_1 Q_2^2 Q_3^3 \cdots\ Q_t^t$ be the complete squarefree factorization of A. Then the computing time for Algorithm PFACT1 is

$$\preceq\ tn^3 L(\gamma)^2\ +\ \sum_{i=1}^{t}\ \tau_i$$

there τ_i is the computing time required to apply PFZ1 to Q_i.

6. Empirical Results

We shall conclude by presenting several tables of empirical results obtained during testing of the algorithms described in this manual.

Although numerous tests have been conducted, no systematic empirical study of computing times has been made. The data presented here, however, are indicative of the behavior of the algorithms and demonstrate the practicality of their application to polynomials with large degree and large coefficients.

The tables give the computing times observed when PFACT1 was applied to "random" polynomials. In order to concisely describe these polynomials, let us define $P(b,d_1,d_2,\ldots,d_k)$ to be the set of

all polynomials over the integers having k irreducible factors $C_1, C_2, \ldots, C_k$ such that $|C_i|_\infty < b$ and deg $C_i = d_i$, $1 \leq i \leq k$. Polynomials in $P(b, d_1, d_2, \ldots, d_k)$ were obtained for the tests by generating polynomials C_i of degree d_i, $1 \leq i \leq k$, each with coefficients randomly chosen in the closed interval $[-b+1, b-1]$, and forming their product. (Actually a polynomial generated in this way is not necessarily in $P(b, d_1, d_2, \ldots, d_k)$, since C_i may be reducible. But the probability of C_i being reducible is very small and none of factors generated were found to be reducible by PFACT1). Note that the size of the coefficients of the products obtained in this way will be about b^k.

For example, Table 1 gives computing comes for five polynomials in $P(2^7, 2, 3, 5)$; these are polynomials of degree 10 whose coefficients are about $2^{21} = 2 \cdot 10^6$ in size. The times are in seconds on a Univac 1108 computer.

Column I of each table gives the time required to compute the content and primitive part of the input polynomial and the squarefree factorization of the primitive part. (All of the polynomials tested were squarefree). Column II gives the time for steps (1)-(10) of PFZ1, in which the mod p factorizations are computed. Column III gives the time for steps (11)-(15), in which algorithms CPBERL, PFB1, and PFC1 are applied. (Most of this time was spent in PFH1). Column IV gives the time for step (16), the application of PFP1. The last column shows the total time.

The number in parentheses following the time in Column II is the number of different primes for which mod p factorizations were obtained. For reducible polynomials this was, of course, always equal to the value $\nu = 5$ used during all of the tests. Irreducible polynomials often required fewer than 5 primes, and thus steps (11)-(16) of PFZ1 were not executed. The variation in the number of primes used in the irreducible cases accounts for the large variations in the computing times in these cases.

Table I

Computing Times for Polynomials of Degree 10 in $P(2^7,2,3,5)$

No.	I	II	III	IV	Total
1	0.21	1.69 (5)	5.21	0.46	7.57
2	0.19	1.61 (5)	6.65	0.49	8.94
3	0.18	1.44 (5)	4.79	0.42	6.83
4	0.19	1.48 (5)	8.43	0.71	10.81
5	0.18	1.95 (5)	5.55	0.48	8.26

Table 2

Computing Times for Polynomials of Degree 15 in $P(2^7,3,5,7)$

No.	I	II	III	IV	Total
1	0.29	4.65 (5)	14.10	0.87	19.91
2	0.26	4.58 (5)	19.25	0.90	24.99
3	0.27	4.18 (5)	19.40	1.17	23.85

Table 3

Computing Times for (Irreducible) Polynomials in $P(2^7,10)$

No.	I	II	III	IV	Total
1	0.16	1.25 (4)	---	---	1.41
2	0.18	1.65 (5)	4.25	0.17	6.24
3	0.16	0.22 (1)	---	---	0.38
4	0.19	0.75 (2)	---	---	0.94
5	0.18	0.56 (2)	---	---	0.74

Table 4

Computing Times for (Irreducible) Polynomials in $P(2^7,15)$

No.	I	II	III	IV	Total
1	0.22	2.80 (4)	---	---	3.02
2	0.23	4.23 (5)	9.65	0.18	14.29
3	0.22	1.06 (2)	---	---	1.28
4	0.24	4.30 (5)	9.85	0.12	14.51
5	0.24	2.48 (3)	---	---	2.72

Table 5

Computing Times for (Irreducible) Polynomials in $P(2^7,20)$

No.	I	II	III	IV	Total
1	0.30	3.09 (2)	---	---	3.39
2	0.28	3.56 (3)	---	---	3.84
3	0.31	6.34 (5)	20.71	0.23	27.36
4	0.32	8.36 (5)	16.28	0.23	25.19
5	0.32	7.87 (5)	18.90	0.32	27.41

Table 6

Computing Times for (Irreducible) Polynomials in $P(2^{20},10)$

No.	I	II	III	IV	Total
1	0.16	0.47 (2)	---	---	0.63
2	0.17	1.66 (5)	6.03	0.18	8.04
3	0.18	1.45 (5)	8.54	0.22	10.39
4	0.17	0.43 (2)	---	---	0.70
5	0.20	0.59 (2)	---	---	0.79

Table 7

Computing Times for (Irreducible) Polynomials in $P(2^{20},15)$

No.	I	II	III	IV	Total
1	0.24	2.48 (3)	---	---	2.72
2	0.26	3.12 (5)	18.20	0.22	21.80
3	0.24	1.04 (2)	---	---	1.28
4	0.26	0.77 (1)	---	---	1.03
5	0.24	1.32 (2)	---	---	1.56

Table 8

Computing Times for (Irreducible) Polynomials in $P(2^{20},20)$

No.	I	II	III	IV	Total
1	0.53	5.63 (4)	---	---	6.16
2	0.56	5.95 (4)	---	---	6.51
3	0.58	4.53 (4)	---	---	5.11
4	0.53	2.96 (2)	---	---	3.49
5	0.60	1.32 (1)	---	---	1.92

7. References

[BER68] E. R. Berlekamp, Algebraic Coding Theory, McGraw-Hill, Inc., New York, 1968.

[BER70] E. R. Berlekamp, Factoring Polynomials over Large Finite Fields, Mathematics of Computation, November, 1970.

[COL67] George E. Collins, The SAC-1 List Processing System, University of Wisconsin Computer Sciences Department Technical Report No. 129, July, 1971.

[COL68a] George E. Collins and James R. Pinkert, The Revised SAC-1 Integer Arithmetic System, University of Wisconsin Computing Center Technical Report No. 9, Nov., 1968.

[COL68b] George E. Collins, The SAC-1 Polynomial System, University of Wisconsin Computer Sciences Department Technical Report No. 115, March, 1971.

[COL68c] George E. Collins, The SAC-1 Rational Function System, University of Wisconsin Computing Center, Technical Report No. 8, Sept. 1971.

[COL69a] George E. Collins, L. E. Heindel, E. Horowitz, M. T. McClellan, and D. R. Musser, The SAC-1 Modular Arithmetic System, University of Wisconsin Technical Report No. 10, June, 1969.

[COL70a] George E. Collins and Ellis Horowitz, The SAC-1 Partial Fraction Decomposition and Rational Function Integration System, University of Wisconsin Computer Sciences Dept. Technical Report No. 80, February 1970.

[COL70b] George E. Collins and Lee E. Heindel, The SAC-1 Polynomial Real Zero System, University of Wisconsin Computing Center Technical Report No. 18, August, 1970.

[COL71] George E. Collins, The Calculation of Multivariate Polynomial Resultants, JACM, Vol. 18, No. 4 (Oct. 1971), pp. 515-532.

[COL71a] George E. Collins, The SAC-1 System: An Introduction and Survey, Proceedings of the Second Symposium on Symbolic and Algebraic Manipulation, Los Angeles, March, 1971.

[COL72] George E. Collins, The SAC-1 Polynomial G.C.D. and Resultant System, Computer Sciences Dept. Technical Report No. 145, February 1972.

[KNU68] Donald E. Knuth, The Art of Computer Programming, Vol I: Fundamental Algorithms, Addison-Wesley Publishing Co., Reading, Mass., 1968.

[KNU69] Donald E. Knuth, The Art of Computer Programming, Vol. II: Seminumerical Algorithms, Addison-Wesley Publishing Co., Reading, Mass., 1968.

[MUS71] David R. Musser, Algorithms for Polynomial Factorization, Computer Sciences Dept. Technical Report No. 134, September 1971.

[VDW49] B. L. Van der Waerden, Modern Algebra, Vol. 1, trans. by Fred Blum, Frederick Ungar Publishing Co., New York, 1949.

[ZAS69] Hans Zassenhaus, On Hensel Factorization, I, Journal of Number Theory 1, 291-311 (1969).

8. Fortran Program Listings

GEN

```
      SUBROUTINE GEN(STACK,NN,SS,N0,K,JJ)
      INTEGER STACK,SS,SP,R,TSP,XXX
      INTEGER FIRST,PFA,PFL,TAIL,CINV
C  GENERATION OF SUM INDEX LISTS.
 1    IF (STACK .EQ. 0) GO TO 2
      CALL DECAP(J,STACK)
      N = 0
      GO TO 6
C  INITIALIZE.
 2    J = LENGTH(NN)
      JJ = 0
      N = N0
      NP = CINV(NN)
      SP = CINV(SS)
      STACK = PFL(SP,PFL(NP,STACK))
      R = 1
 25   IF (J .LE. K) GO TO 3
      NP = TAIL(NP)
      SP = TAIL(SP)
      J = J-1
      GO TO 25
C  STEPS 3 THROUGH 9 CORRESPOND TO A RECURSIVE PROCEDURE G(J,N,JJ).
C  STEP 2 HAS INITIALIZED TO PERFORM THE PROCEDURE WITH J=K AND JJ=0.
C
C  CHECK IF A SUM INDEX LIST HAS BEEN GENERATED.
 3    IF (N .NE. 0) GO TO 4
      STACK = PFA(J,STACK)
      RETURN
C  THIS EXIT ALLOWS THE CURRENT VALUE OF JJ TO BE USED
C  OUTSIDE THE SUBROUTINE.  THE SUBROUTINE MAY BE REENTERED
C  TO GENERATE ANOTHER SUM INDEX LIST, PROVIDED THAT
C  NEITHER STACK NOR JJ IS ALTERED OUTSIDE THE SUBROUTINE.
C
C  CHECK IF RECURSION IS NECESSARY.
 4    TSP = TAIL(SP)
      IF (MEMBER(N-FIRST(NP),FIRST(TSP)).EQ. 0) GO TO 7
C  CALL RECURSIVE PROCEDURE G(J-1,N-NNG),PREFIX(J,JJ)).
 5    STACK = PFA(R,PFA(SP,PFA(NP,STACK)))
      N = N-FIRST(NP)
      NP = TAIL(NP)
      SP = TSP
      R = 2
      JJ = PFA(J,JJ)
      J = J-1
      GO TO 3
 6    CALL DECAP(R,STACK)
      CALL DECAP(SP,STACK)
      CALL DECAP(NP,STACK)
      J = J+1
```

```
      N = N+FIRST(NP)
      CALL DECAP(XXX,JJ)
      TSP = TAIL(SP)
C  CHECK IF RECURSION IS NECESSARY.
 7    IF (MEMBER(N,FIRST(TSP)) .EQ. 0) GO TO 99
C  CALL RECURSIVE PROCEDURE G(J-1,N,JJ)
 8    STACK = PFA(R,PFA(SP,PFA(NP,STACK)))
      NP = TAIL(NP)
      SP = TSP
      R = 3
      J = J-1
      GO TO 3
 9    CALL DECAP(R,STACK)
      CALL DECAP(SP,STACK)
      CALL DECAP(NP,STACK)
      J = J + 1
 99   GO TO (10,6,9), R
C  THE FOLLOWING EXIT IS TAKEN WHEN ALL SUM INDEX LISTS
C  HAVE BEEN GENERATED.
 10   CALL ERASE(STACK)
      RETURN
      END
IAND
      INTEGER FUNCTION IAND(A,B)
      INTEGER A,B
      INTEGER C,FIRST,PFA,TAIL,T1,T2,T3
C  IT IS ASSUMED THAT BETA (THE BASE OF THE SAC-1 INTEGER ARITH.
C  SYSTEM) IS A POWER OF 2.
 1    C = 0
      T1 = A
      T2 = B
 2    IF (T1 .EQ. 0 .OR. T2 .EQ. 0) GO TO 3
      T3 = AND(FIRST(T1),FIRST(T2))
      C = PFA(T3,C)
      T1 = TAIL(T1)
      T2 = TAIL(T2)
      GO TO 2
 3    IAND = INV(C)
      RETURN
      END
ILS
      INTEGER FUNCTION ILS(L,N)
      INTEGER PFA,Q,R,T,WL
C  IT IS ASSUMED THAT BETA (THE BASE OF THE SAC-1 INTEGER ARITH.
C  SYSTEM) IS A POWER OF 2.
      WL = 33
C  WL = BASE 2 LOGARITHM OF BETA.
1     Q = N/WL
      R = N-Q*WL
      T = PFA(2**R,0)
```

```
      ILS = IPROD(L,T)
      CALL ERLA(T)
2     IF (Q .EQ. 0) RETURN
      ILS = PFA(0,ILS)
      Q = Q-1
      GO TO 2
      END
IOR
      INTEGER FUNCTION IOR(A,B)
      INTEGER A,B
      INTEGER BORROW,C,FIRST,PFA,TAIL,T1,T2,T3
C  IT IS ASSUMED THAT BETA (THE BASE OF THE SAC-1 INTEGER ARITH.
C  SYSTEM) IS A POWER OF 2.
1     C = 0
      T1 = A
      T2 = B
2     IF (T1 .EQ. 0) GO TO 4
      IF (T2 .EQ. 0) GO TO 3
      T3 = OR(FIRST(T1),FIRST(T2))
      C = PFA(T3,C)
      T1 = TAIL(T1)
      T2 = TAIL(T2)
      GO TO 2
3     T2 = T1
4     IOR = INV(C)
      CALL SSUCC(BORROW(T2),C)
      RETURN
      END
LAST
      FUNCTION LAST(L)
      INTEGER T,TAIL,FIRST
 1    M = L
      IF (L .EQ. 0) GO TO 4
 2    T = TAIL(M)
      IF (T .EQ. 0) GO TO 3
      M = T
      GO TO 2
 3    LAST = FIRST(M)
      RETURN
 4    LAST = M
      RETURN
      END
MEMBER
      INTEGER FUNCTION MEMBER(N,S)
      INTEGER N,S
      INTEGER FIRST,Q,R,T,TAIL,T1,WL
C  IT IS ASSUMED THAT BETA (THE BASE OF THE SAC-1 INTEGER ARITH.
C  SYSTEM) IS A POWER OF 2.
      WL = 33
C  WL = BASE 2 LOGARITHM OF BETA.
```

```
1       IF (N .LT. 0) GO TO 4
        T = S
        Q = N/WL
        R = N-Q*WL
2       IF (T .EQ. 0) GO TO 4
        IF (Q.EQ. 0) GO TO 3
        T = TAIL(T)
        Q = Q-1
        GO TO 2
3       T1 = FIRST(T)/2**R
        MEMBER = T1-(T1/2)*2
        RETURN
4       MEMBER = 0
        RETURN
        END
MIMOD
        INTEGER FUNCTION MIMOD(M,A)
        INTEGER M,A
        INTEGER R,R1,TR
1       R = IREM(A,M)
        IF (R .EQ. 0) GO TO 3
        IF (ISIGNL(R) .GT. 0) GO TO 2
        R1 = ISUM(R,M)
        CALL ERLA(R)
        R = R1
2       TR = ISUM(R,R)
        IF (ICOMP(TR,M) .LT. 0) GO TO 25
        R1 = IDIF(R,M)
        CALL ERLA(R)
        R = R1
25      CALL ERLA(TR)
3       MIMOD=R
        RETURN
        END
MPLPR
        FUNCTION MPLPR(M,A)
        INTEGER A,AA,B,FA,T,BORROW,FIRST,PFA,PPROD,TAIL
 1      IF (A .NE. 0) GO TO 15
        MPLPR = PFA(1,0)
        RETURN
 15     B = BORROW(FIRST(A))
        AA = TAIL(A)
 2      IF (AA .EQ. 0) GO TO 4
 3      CALL ADV(FA,AA)
        T = PPROD(B,FA)
        CALL PERASE(B)
        B = MPMOD(M,T)
        CALL PERASE(T)
        GO TO 2
 4      MPLPR = B
```

```
      RETURN
      END
MPMOD
      INTEGER FUNCTION MPMOD(M,A)
      INTEGER M,A
      INTEGER AA,B,C,D,E,PFA,PFL,PVBL,TAIL,TYPE
 1    B = 0
      IF (A .EQ. 0) GO TO 4
      IF (TYPE(A) .NE. 0) GO TO 15
      B=MIMOD(M,A)
      GO TO 4
 15   AA = TAIL(A)
 2    CALL ADV(C,AA)
      CALL ADV(E,AA)
      D=MIMOD(M,C)
      IF (D .NE. 0) B = PFA(E,PFL(D,B))
      IF (AA .NE. 0 ) GO TO 2
 3    IF (B .NE. 0) B = PFL(PVBL(A),INV(B))
 4    MPMOD = B
      RETURN
      END
MPQREM
      INTEGER FUNCTION MPQREM(M,A,B)
      INTEGER M,A,B
      INTEGER BD,BL,BORROW,BR,C,D,J
      INTEGER PDEG,PDIF,PFA,PFL,PLDCF,PRED,PSPROD,PVBL,Q
      INTEGER R,RD,RL,RR,TEMP,TEMP1
1     R = BORROW(A)
      BL = PLDCF(B)
      C = MRECIP(M,BL)
      CALL ERLA(BL)
      Q = 0
      BR = PRED(B)
      BD = PDEG(B)
2     RD = PDEG(R)
      J = RD-BD
      IF (J .LT. 0 .OR. R .EQ. 0) GO TO 3
      RL = PLDCF(R)
      TEMP = IPROD(C,RL)
      D=MIMOD(M,TEMP)
      CALL ERLA(TEMP)
      TEMP = PSPROD(BR,D,J)
      RR = PRED(R)
      CALL PERASE(R)
      TEMP1 = PDIF(RR,TEMP)
      R = MPMOD(M,TEMP1)
      CALL PERASE(TEMP1)
      CALL PERASE(TEMP)
      CALL PERASE(RR)
      CALL ERLA(RL)
```

```
      Q = PFA(J,PFL(D,Q))
      GO TO 2
3     IF (Q .NE. 0) Q = PFL(PVBL(A),INV(Q))
      MPQREM = PFL(Q,PFL(R,0))
      CALL PERASE(BR)
      CALL ERLA(C)
      RETURN
      END
MPSPEQ
      SUBROUTINE MPSPEQ(M,A,B,S,T,U,Y,Z)
      INTEGER M,A,B,S,T,U,Y,Z
      INTEGER PPROD,PSUM,Q,TEMP,TEMP1,TEMP2,V,W
 1    W = MPMOD(M,U)
      TEMP = PPROD(T,W)
      V = MPMOD(M,TEMP)
      CALL PERASE(TEMP)
 2    TEMP = MPQREM(M,V,A)
      CALL DECAP(Q,TEMP)
      CALL DECAP(Z,TEMP)
      CALL PERASE(V)
 3    TEMP = PPROD(S,W)
      TEMP1 = PPROD(B,Q)
      TEMP2 = PSUM(TEMP,TEMP1)
      CALL PERASE(TEMP1)
      CALL PERASE(TEMP)
      CALL PERASE(Q)
      Y = MPMOD(M,TEMP2)
      CALL PERASE(TEMP2)
      CALL PERASE(W)
      RETURN
      END
MRECIP
      INTEGER FUNCTION MRECIP(M,X)
      INTEGER M,X
      INTEGER A1,A2,A3,BORROW,FIRST
      INTEGER PFA,Q,TAIL,TEMP,Y1,Y2,Y3
      A1 = BORROW(M)
      A2 = BORROW(X)
      IF (ISIGNL(A2) .GT. 0) GO TO 5
      TEMP = ISUM(M,A2)
      CALL ERLA(A2)
      A2 = TEMP
5     Y1= 0
      Y2 = PFA(1,0)
      GO TO 2
1     TEMP = IQR(A1,A2)
      CALL DECAP(Q,TEMP)
      CALL DECAP(A3,TEMP)
      TEMP = IPROD(Y2,Q)
      Y3 = IDIF(Y1,TEMP)
```

```
      CALL ERLA(TEMP)
      CALL ERLA(Q)
      CALL ERLA(A1)
      A1 = A2
      A2 = A3
      CALL ERLA(Y1)
      Y1 = Y2
      Y2 = Y3
2     IF (FIRST(A2)  .NE. 1 .OR. TAIL(A2) .NE. 0) GO TO 1
4     MRECIP = Y2
      CALL ERLA(A1)
      CALL ERLA(A2)
      CALL ERLA(Y1)
      RETURN
      END
PFACT1
      INTEGER FUNCTION PFACT1(A)
      INTEGER A
      INTEGER BORROW,C,CONT,D,E,F,FF,FIRST
      INTEGER PCPP,PDEG,PFL,PFZ1,PP,PSFREE,Q,TAIL,TEMP
 1    TEMP = PCPP(A)
      CALL DECAP(CONT,TEMP)
      CALL DECAP(PP,TEMP)
      IF (ISIGNL(FIRST(TAIL(A))) .GT. 0) GO TO 15
      TEMP = INEG(CONT)
      CALL ERLA(CONT)
      CONT = TEMP
 15   FF = PFL(CONT,0)
      Q = PSFREE(PP)
      CALL PERASE(PP)
      I = 1
 2    CALL DECAP(C,Q)
      D = PDEG(C)
      IF (D .EQ. 0) GO TO 28
      IF (D .GT. 1) GO TO 24
      DO 22 J = 1,I
 22   FF = PFL(BORROW(C),FF)
      GO TO 28
 24   F = PFZ1(C)
 25   CALL DECAP(E,F)
      DO 26 J = 1,I
 26   FF = PFL(BORROW(E),FF)
      CALL PERASE(E)
      IF (F .NE. 0) GO TO 25
 28   CALL PERASE(C)
      I = I+1
      IF (Q .NE. 0) GO TO 2
      PFACT1 = INV(FF)
      RETURN
      END
```

```
PFB1
      SUBROUTINE PFB1(C0,M,C,L,B)
      INTEGER C0,C,B,V,T,F,AV,V1,V2,AV1,AV2,T1,T2
      INTEGER BORROW,PDEG,PTLCF,PFL,PVBL,PFA,PQ,PSUBST
      INTEGER FIRST,TAIL
 1    C = BORROW(C0)
      L = 0
      B = 0
 2    IF (PDEG(C) .NE. 1) GO TO 3
      T=TAIL(C)
      CALL ERLA(B)
      B=IABSL(FIRST(T))
      T=TAIL(TAIL(T))
      IF(T.EQ.0)RETURN
      V1=B
      V2=IABSL(FIRST(T))
      B=ISUM(V1,V2)
      CALL ERLA(V1)
      CALL ERLA(V2)
      RETURN
 3    V = PTLCF(C)
      IF (V .NE. 0) GO TO 5
      T = PFA(1,0)
      LF = PFL(PVBL(C),PFL(BORROW(T),T))
      GO TO 41
 4    LF = PFL(PVBL(C),PFL(PFA(1,0),PFA(1,PFL(PFA(-J,0),PFA(0,0)))))
 41   T = PQ(C,LF)
      CALL ERASE(C)
      C = T
      L = PFL(LF,L)
      GO TO 2
 5    F = 4
      T = PFA(F,0)
      AV = IABSL(V)
      CALL ERLA(V)
      CALL ERLA(B)
      B = IPROD(AV,T)
      CALL ERLA(AV)
      CALL ERLA(T)
      J = 1
 6    IF (J .GT. M) RETURN
 7    T = PFA(J,0)
      V1 = PSUBST(T,C)
      CALL ERLA(T)
      IF (V1 .EQ. 0) GO TO 4
 8    T = PFA(-J,0)
      V2 = PSUBST(T,C)
      CALL ERLA(T)
      IF (V2 .NE. 0) GO TO 9
      CALL ERLA(V1)
```

```
      J = -J
      GO TO 4
 9    K1 = (M-J+1) * F
      K2 = M + J
      F = K1 / K2
      IF (K1 .NE. F*K2) F = F+1
      T = PFA((J+1)*F, )
      AV1 = IABSL(V1)
      CALL ERLA(V1)
      AV2 = IABSL(V2)
      CALL ERLA(V2)
      T1 = ISUM(AV1,AV2)
      T2 = IPROD(T1,T)
      CALL ERLA(AV1)
      CALL ERLA(AV2)
      CALL ERLA(T1)
      CALL ERLA(T)
      T = PFA(J*J+1,0)
      T1 = IQR(T2,T)
      CALL ERLA(T)
      CALL ERLA(T2)
      CALL DECAP(T,T1)
      CALL DECAP(T2,T1)
      IF (T2 .EQ. 0) GO TO 95
      CALL ERLA(T2)
      T1 = PFA(1,0)
      T2 = ISUM(T,T1)
      CALL ERLA(T)
      T = T2
      CALL ERLA(T1)
 95   T1 = ISUM(B,T)
      CALL ERLA(B)
      B = T1
      CALL ERLA(T)
      J = J +1
      GO TO 6
      END
PFC1
      INTEGER FUNCTION PFC1(P,M,C0,G)
      INTEGER P,M,C0,G
      INTEGER A,AB,B,BB,BORROW,C,CB,CPEGCD,CPMOD,CPQREM,CRECIP,CSPROD
      INTEGER F,FIRST,GS,PFL,PIP
      INTEGER PLDCF,SB,TAIL,TB,TEMP,W,WI,XXX
 1    C = BORROW(C0)
      F = 0
      CB = CPMOD(P,C)
 2    GS = G
 21   IF (TAIL(GS) .EQ. 0) GO TO 3
      AB = FIRST(GS)
      TEMP = CPQREM(P,CB,AB)
```

```
      CALL DECAP(BB,TEMP)
      CALL DECAP(XXX,TEMP)
      IF (FIRST(AB) .GE. FIRST(BB)) GO TO 22
      TEMP = CPEGCD(P,BB,AB)
      CALL DECAP(TB,TEMP)
      CALL DECAP(SB,TEMP)
      GO TO 23
22    TEMP = CPEGCD(P,AB,BB)
      CALL DECAP(SB,TEMP)
      CALL DECAP(TB,TEMP)
23    CALL DECAP(W,TEMP)
      WI = CRECIP(P,W)
      TEMP = CSPROD(P,SB,WI,0)
      CALL ERLA(SB)
      SB = TEMP
      TEMP = CSPROD(P,TB,WI,0)
      CALL ERLA(TB)
      TB = TEMP
      CALL PFH1(P,M,C,AB,BB,SB,TB,A,B)
      F = PFL(A,F)
      CALL PERASE(C)
      C = B
      CALL ERLA(CB)
      CB = BB
      CALL ERLA(SB)
      CALL ERLA(TB)
      GS = TAIL(GS)
      GO TO 21
3     LC = PLDCF(C)
      LCI = MRECIP(M,LC)
      CALL ERLA(LC)
      TEMP = PIP(C,LCI)
      CALL ERLA(LCI)
      CALL PERASE(C)
      C = MPMOD(M,TEMP)
      CALL PERASE(TEMP)
      PFC1 = INV(PFL(C,F))
      CALL ERLA(CB)
      RETURN
      END
PFH1
      SUBROUTINE PFH1(P,M,C,AB,BB,SB,TB,A,B)
      INTEGER P,M,C,AB,BB,SB,TB,A,B
      INTEGER AS,BS,CPGARN,ONE,PDIF,PFA,PFL
      INTEGER PIP,PPROD,PSQ,PSUM,PVBL,Q,Q2,S,SS,T,TEMP,TEMP1,TEMP2
      INTEGER TS,U,V,Y,Y1,Z,Z1,AT,BT,ST,TT,QT
1     Q = PFA(P,0)
      Q2 = 0
      V = PFL(PVBL(C),0)
      ONE = PFA(1,0)
```

```
      A = CPGARN(ONE,0,P,AB,V)
      B = CPGARN(ONE,0,P,BB,V)
      S = CPGARN(ONE,0,P,SB,V)
      T = CPGARN(ONE,0,P,TB,V)
      CALL ERASE(V)
      CALL ERLA(ONE)
      ONE = PFL(PVBL(C),PFL(PFA(1,0),PFA(0,0)))
2     IF (ICOMP(Q,M) .LT. 0) GO TO 3
21    CALL ERLA(Q)
      CALL ERLA(Q2)
      CALL PERASE(ONE)
      CALL PERASE(S)
      CALL PERASE(T)
      RETURN
3     TEMP = PPROD(A,B)
      TEMP1 = PDIF(C,TEMP)
      CALL PERASE(TEMP)
      U = PSQ(TEMP1,Q)
      CALL PERASE(TEMP1)
      Q2 = IPROD(Q,Q)
      IC = ICOMP(Q2,M)
      IF (IC .LE. 0) GO TO 32
      QT = IQ(M,Q)
      AT = MPMOD(QT,A)
      BT = MPMOD(QT,B)
      ST = MPMOD(QT,S)
      TT = MPMOD(QT,T)
      CALL MPSPEQ(QT,AT,BT,ST,TT,U,Y,Z)
      CALL PERASE(AT)
      CALL PERASE(BT)
      CALL PERASE(ST)
      CALL PERASE(TT)
      CALL ERLA(QT)
      GO TO 34
32    CALL MPSPEQ(Q,A,B,S,T,U,Y,Z)
34    CALL PERASE(U)
4     TEMP = PIP(Z,Q)
      CALL PERASE(Z)
      AS = PSUM(A,TEMP)
      CALL PERASE(TEMP)
      TEMP = PIP(Y,Q)
      CALL PERASE(Y)
      BS = PSUM(B,TEMP)
      CALL PERASE(TEMP)
      IF (IC .LT. 0) GO TO 5
      CALL PERASE(A)
      A = AS
      CALL PERASE(B)
      B = BS
      GO TO 21
```

```
5     TEMP = PPROD(AS,S)
      TEMP1 = PPROD(BS,T)
      TEMP2= PSUM(TEMP,TEMP1)
      CALL PERASE(TEMP1)
      CALL PERASE(TEMP)
      TEMP = PDIF(ONE,TEMP2)
      MU1 = PSQ(TEMP,Q)
      CALL PERASE(TEMP2)
      CALL PERASE(TEMP)
      CALL MPSPEQ(Q,A,B,S,T,MU1,Y1,Z1)
      CALL PERASE(MU1)
6     TEMP = PIP(Y1,Q)
      CALL PERASE(Y1)
      SS = PSUM(S,TEMP)
      CALL PERASE(TEMP)
      TEMP = PIP(Z1,Q)
      CALL PERASE(Z1)
      TS = PSUM(T,TEMP)
      CALL PERASE(TEMP)
7     CALL ERLA(Q)
      Q = Q2
      CALL PERASE(A)
      CALL PERASE(B)
      CALL PERASE(S)
      CALL PERASE(T)
      A = AS
      B = BS
      S = SS
      T = TS
      GO TO 2
      END
PFP1
      INTEGER FUNCTION PFP1(M,CC0,GG0,DS)
      INTEGER M,CC0,GG0,DS
      INTEGER  A,AA,AAS,BBS,BORROW,C,CC,CCS,D,DEGSET,FF,FIRST
      INTEGER GG,GGJJ,GGS,PDEG,PFA,PFL
      INTEGER PIP,PLDCF,PPP,PQ,PSQ,PTLCF,PVBL,REM,SELECT,SS
      INTEGER STACK,SUMSET,T,TAIL,TCCS,TEMP,TT,TTJJ,R,V

1     CC = BORROW(CC0)
      GG = BORROW(GG0)
      FF = 0
      D = 1
      NN = 0
      TT = 0
      GGS = GG
15    CALL ADV(AA,GGS)
      NN = PFA(PDEG(AA),NN)
      TT = PFL(PTLCF(AA),TT)
      IF (GGS .NE. 0) GO TO 15
```

```
      NN = INV(NN)
      TT = INV(TT)
      R = LENGTH(GG)
      K = R
2     C = PLDCF(CC)
      CCS = PIP(CC,C)
      TCCS = PTLCF(CCS)
      SS = SUMSET(NN)
      DEGSET = IAND(DS,LAST(SS))
3     IF (D .LE. PDEG(CC)/2) GO TO 35
      FF = PFL(CC,FF)
      CALL PERASE(GG)
      CALL ERLA(NN)
      CALL ERASE(TT)
      CALL ERLA(C)
      CALL ERLA(TCCS)
      CALL PERASE(CCS)
      CALL ERASE(SS)
      CALL ERLA(DEGSET)
      PFP1 = INV(FF)
      RETURN
35    IF (MEMBER(D,DEGSET) .EQ. 0 .OR. K .EQ. 0) GO TO 7
4     STACK = 0
41    CALL GEN(STACK,NN,SS,D,K,JJ)
      IF (JJ .EQ. 0) GO TO 7
5     TTJJ = SELECT(TT,JJ)
      TEMP = PFL(BORROW(C),TTJJ)
      T = MPLPR(M,TEMP)
      CALL ERASE(TEMP)
      REM = IREM(TCCS,T)
      CALL ERLA(T)
      IF (REM .EQ. 0) GO TO 6
      CALL ERLA(REM)
      GO TO 41
6     GGJJ = SELECT(GG,JJ)
      TEMP = PFL(PFL(PVBL(CC),PFL(BORROW(C),PFA(0,0))),GGJJ)
      AAS = MPLPR(M,TEMP)
      CALL ERASE(TEMP)
      BBS = PQ(CCS,AAS)
      IF (BBS .GT. 0) GO TO 8
      CALL PERASE(AAS)
      GO TO 41
7     D = D+1
      K = R
      GO TO 3
8     AA = PPP(AAS)
      CALL PERASE(AAS)
      FF = PFL(AA,FF)
      CALL PERASE(CC)
      A = PLDCF(AA)
```

```
      CC = PSQ(BBS,A)
      CALL ERLA(A)
      CALL PERASE(BBS)
      JJP = JJ
      KK = 0
      DO 96 I = 1,R
      IF (JJP .EQ. 0) GO TO 95
      IF (I .NE. FIRST(JJP)) GO TO 95
      JJP = TAIL(JJP)
      GO TO 96
 95   KK = PFA(I,KK)
 96   CONTINUE
      KK = INV(KK)
      V = LENGTH(JJ)
      JV = LAST(JJ)
      CALL ERLA(JJ)
      CALL ERASE(STACK)
      TEMP = SELECT(GG,KK)
      CALL ERASE(GG)
      GG = TEMP
      TEMP = SELECT(NN,KK)
      CALL ERASE(NN)
      NN = TEMP
      TEMP = SELECT(TT,KK)
      CALL ERASE(TT)
      TT = TEMP
      R = R - V
      K = JV - V
      CALL ERLA(KK)
      CALL ERLA(C)
      CALL ERLA(TCCS)
      CALL PERASE(CCS)
      CALL ERASE(SS)
      CALL ERLA(DEGSET)
      GO TO 2
      END
PFZ1
      INTEGER FUNCTION PFZ1(C0)
      INTEGER C,C0
      INTEGER B,BORROW,CB,CBP,CB1,CMOD,CMONIC,CONC,CPBERL,CPDDF,CPDRV
      INTEGER CPGCD1,CPMOD,D,DB,DFD,DS,F,FD,FDF,FIRST,F1,G,GP,GS,GI,H
      INTEGER ONE,P,PDEG,PFA,PFC1,PFL,PFP1,PONE,PS,Q,R
      INTEGER RMIN,SP,SPRIME,SS,SUMSET,TAIL,TEMP
      INTEGER CS,FP,A,AP,PP
      COMMON /TR5/ NU,SPRIME
 1    SP = SPRIME
      C = BORROW(C0)
      RMIN = PDEG(C)+1
      GS = 0
      NP = 1
```

```
      ONE = PFA(1,0)
      TEMP = ILS(ONE,PDEG(C)/2+1)
      DS = IDIF(TEMP,ONE)
      CALL ERLA(ONE)
      CALL ERLA(TEMP)
2     IF (SP .EQ. 0) STOP
      CALL ADV(P,SP)
3     IF (CMOD(P,FIRST(TAIL(C))) .EQ. 0) GO TO 2
4     CB = CPMOD(P,C)
      CBP = CPDRV(P,CB)
      IF (CBP .NE. 0) GO TO 44
42    CALL ERLA(CB)
      GO TO 2
44    B = CPGCD1(P,CB,CBP)
      DB = FIRST(B)
      CALL ERLA(B)
      CALL ERLA(CBP)
      IF (DB .NE. 0) GO TO 42
5     CB1 = CMONIC(P,CB)
      CALL ERLA(CB)
      G = CPDDF(P,CB1)
      CALL ERLA(CB1)
6     NN = 0
      GP = G
61    CALL ADV(DFD,GP)
      CALL ADV(D,DFD)
      CALL ADV(FD,DFD)
      KD = FIRST(FD)/D
      DO 62 K = 1,KD
62    NN = PFA(D,NN)
      IF (GP .NE. 0) GO TO 61
7     R = LENGTH(NN)
      IF (R .NE. 1) GO TO 8
      CALL ERLA(NN)
71    F = PFL(BORROW(C),0)
      CALL ERASE(G)
      CALL ERASE(GS)
      GO TO 99
8     SS = SUMSET(NN)
      TEMP = IAND(DS,LAST(SS))
      CALL ERLA(DS)
      DS = TEMP
      CALL ERASE(SS)
      CALL ERLA(NN)
      IF (PONE(DS) .EQ. 1) GO TO 71
9     IF (R .GE. RMIN) GO TO 95
      RMIN = R
      PS = P
      CALL ERASE(GS)
      GS = G
```

```
      GO TO 10
95    CALL ERASE(G)
10    NP = NP+1
      IF (NP .LE. NU) GO TO 2
11    P = PS
      G = GS
      H = 0
12    CALL DECAP(DFD,G)
      CALL DECAP(D,DFD)
      CALL DECAP(FD,DFD)
      IF (D .NE. FIRST(FD)) GO TO 122
      H = PFL(FD,H)
      GO TO 125
122   FDF = CPBERL(P,FD)
      H = CONC(FDF,H)
      CALL ERLA(FD)
125   IF (G .NE. 0) GO TO 12
      G = INV(H)
13    D = PDEG(C)/2
1301  IF (MEMBER(D,DS) .EQ. 1) GO TO 1302
      D = D-1
      GO TO 1301
1302  CALL PFB1(C,D,CS,F,B)
      IF (PDEG(CS) .NE. 1) GO TO 132
      F = PFL(CS,F)
131   CALL ERLA(B)
      CALL ERASE(G)
      GO TO 99
132   CALL PERASE(C)
      C = CS
      FP = F
133   IF (FP .EQ. 0) GO TO 135
      CALL ADV(A,FP)
      AP = CPMOD(P,A)
      H = G
      G = PFA(0,G)
      GP = G
134   IF (ICOMP(FIRST(H),AP) .EQ. 0) GO TO 1341
      GP = H
      H = TAIL(H)
      IF (H .NE. 0) GO TO 134
      GO TO 1342
1341  CALL DECAP(GI,H)
      CALL ERLA(GI)
      CALL SSUCC(H,GP)
1342  CALL DECAP(GP,G)
      CALL ERLA(AP)
      GO TO 133
135   IF (TAIL(G) .NE. 0) GO TO 14
      F = PFL(BORROW(C),F)
```

```
      GO TO 131
 14   PP = PFA(P,0)
      M = BORROW(PP)
      TEMP = IPROD(B,FIRST(TAIL(C)))
      Q = ISUM(TEMP,TEMP)
      CALL ERLA(TEMP)
      CALL ERLA(B)
 142  IF(ICOMP(M,Q) .GE. 0) GO TO 145
      TEMP = IPROD(M,PP)
      CALL ERLA(M)
      M = TEMP
      GO TO 142
 145  CALL ERLA(PP)
      CALL ERLA(Q)
 15   F1 = PFC1(P,M,C,G)
      CALL ERASE(G)
 16   F = CONC(PFP1(M,C,F1,DS),F)
      CALL ERLA(M)
      CALL ERASE(F1)
 99   PFZ1 = F
      CALL ERLA(DS)
      CALL PERASE(C)
      RETURN
      END
PSFREE
      INTEGER FUNCTION PSFREE(A)
      INTEGER A,AP,B,C,D,T
      INTEGER FIRST,PDERIV,PGCD,PDEG,BORROW,PQ,PFL
      L=0
1     AP=PDERIV(A,FIRST(A))
      B=PGCD(A,AP)
      CALL PERASE(AP)
      IF(PDEG(B).GT.0)GO TO 15
      C=BORROW(A)
      GO TO 2
15    C=PQ(A,B)
2     IF(PDEG(B).GT.0)GO TO 3
      L=INV(PFL(C,L))
      CALL PERASE(B)
      GO TO 99
3     D=PGCD(B,C)
      IF(PDEG(D).GT.0)GO TO 35
      L=PFL(BORROW(C),L)
      GO TO 4
35    L=PFL(PQ(C,D),L)
4     IF(PDEG(D).EQ.0)GO TO 45
      T=PQ(B,D)
      CALL PERASE(B)
      B=T
45    CALL PERASE(C)
```

```
      C=D
      GO TO 2
99    PSFREE=L
      RETURN
      END
PTLCF
      INTEGER FUNCTION PTLCF(P)
      INTEGER P
      INTEGER BORROW,FIRST,Q,R,TAIL,TYPE
 1    Q = P
      IF (Q .EQ. 0) GO TO 25
      IF (TYPE(Q) .EQ. 0) GO TO 4
      Q = TAIL(Q)
 2    R = TAIL(Q)
      IF (FIRST(R) .EQ. 0) GO TO 3
      Q = TAIL(R)
      IF (Q .NE. 0) GO TO 2
 25   PTLCF = 0
      RETURN
 3    PTLCF = BORROW(FIRST(Q))
      RETURN
 4    PTLCF = BORROW(Q)
      RETURN
      END
SELECT
      INTEGER FUNCTION SELECT(A,I)
      INTEGER A,I
      INTEGER AS,B,BORROW,FIRST,PFA,PFL,TAIL,TYPE
 1    N = 1
      AS = A
      IS = I
      B = 0
 2    IF (AS .EQ. 0 .OR. IS .EQ. 0) GO TO 5
      K = FIRST(IS)
 3    IF (N .GE. K) GO TO 4
      AS = TAIL(AS)
      N = N+1
      IF (AS .EQ. 0) GO TO 5
      GO TO 3
 4    IF (TYPE(AS) .EQ. 1) GO TO 42
      B = PFA(FIRST(AS),B)
      GO TO 44
 42   B = PFL(BORROW(FIRST(AS)),B)
 44   IS = TAIL(IS)
      AS = TAIL(AS)
      N = N+1
      GO TO 2
 5    SELECT = INV(B)
      RETURN
      END
```

```
SUMSET
      INTEGER FUNCTION SUMSET(N)
      INTEGER FIRST,PFA,PFL,R,S,T,TAIL
 1    R = PFA(1,0)
      S = PFL(R,0)
      NP = N
 2    IF (NP .EQ. 0) GO TO 3
      T = ILS(R,FIRST(NP))
      R = IOR(R,T)
      CALL ERLA(T)
      S = PFL(R,S)
      NP = TAIL(NP)
      GO TO 2
 3    SUMSET = INV(S)
      RETURN
      END
```

9. Index of Algorithms

BIBLIOGRAPHIC DATA SHEET	1. Report No. WIS-CS-157-72	2.	3. Recipient's Accession No.
4. Title and Subtitle The SAC-1 Polynomial Factorization System			5. Report Date March 1972 6.
7. Author(s) G. E. Collins and D. R. Musser			8. Performing Organization Rept. No.
9. Performing Organization Name and Address Computer Sciences Department University of Wisconsin 1210 West Dayton Street Madison, Wisconsin 53706			10. Project/Task/Work Unit No. 11. Contract/Grant No. GJ-30125X
12. Sponsoring Organization Name and Address National Science Foundation Washington, D. C. 20550			13. Type of Report & Period Covered 14.

15. Supplementary Notes

16. Abstracts

The SAC-1 Polynomial Factorization System is the ninth subsystem of the SAC-1 System, a Fortran system for performing operations on multivariate polynomials and rational functions with exact, infinite-precision coefficients. The SAC-1 Polynomial Factorization System provides efficient algorithms and subprograms for the factorization of univariate integral polynomials into their irreducible divisors using, most notably, Berlekamp's algorithm and Hensel's Lemma. For each algorithm there is given a semi-formal description, its theoretical computing time, and a Fortran IV program listing. Tables of observed computing times are given for the factorization of representative polynomials of degrees up to 20 with coefficients up to 6 decimal digits, the computing times on a UNIVAC 1108 ranging up to about 30 seconds.

17. Key Words and Document Analysis. 17a. Descriptors

Polynomials, SAC-1, Factorization, Algebraic Algorithms, Computational Algebra, Analysis of Algorithms, Irreducibility, Hensel's Lemma, Berlekamp's Algorithm, SAC-1, Fortran, List Processing, Modular Arithmetic, Set Operations, Algebra.

17b. Identifiers/Open-Ended Terms

17c. COSATI Field/Group

18. Availability Statement	19. Security Class (This Report) UNCLASSIFIED	21. No. of Pages 65
	20. Security Class (This Page UNCLASSIFIED	22. Price