THE SAC-1 RATIONAL FUNCTION SYSTEM

by

George E. Collins

Technical Report #135

September 1971

Computer Sciences Department
The University of Wisconsin
1210 West Dayton Street
Madison, Wisconsin 53706

THE SAC-1 RATIONAL FUNCTION SYSTEM*

by

George E. Collins

Technical Report #135†

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

†This report is also distributed as Madison Academic Computing Center Technical Report #8.

Table of Contents

1. Introduction

The SAC-1 Rational Function System is the fourth subsystem of the SAC-1 System for Symbolic and Algebraic Calculation. The three prior subsystems are the List Processing System [4], the Infinite Precision Integer Arithmetic System [5], and the Polynomial System [6]. Numerous additional subsystems are in progress or are being planned. The documentations of the previous systems are available upon request, and familiarity with these will be assumed in the present document wherever this is convenient.

All SAC-1 subsystems are programmed in FORTRAN in strict accordance with the A.S.A. specifications [1]. Each subsystem is a collection of many subprograms whose common purpose is to perform a certain class of operations appropriate to a specific class of data. The data class for the present system is the class of all multivariate rational functions with infinite-precision integer coefficients. Since the rational functions in no variables are included as a special case, the Rational Function System also provides operations on infinite-precision rational numbers. The operations provided are the arithmetic operations (addition, subtraction, multiplication, division), input and output (including conversions between internal and external canonical forms), differentiation, substitution (including numerical evaluation as a special case) and a few miscellaneous operations.

In the following sections, the system is described at several levels. Section 2 describes the external and internal representations of the data. Section 3 is a user's description of the subprograms, providing just that information which is needed to apply the system in most cases. Section 4 illustrates the use and capability of the system. The example application chosen is the exact inversion of matrices with rational number or rational function entries, and this itself is further specialized to Hilbert matrices and a generalization thereof. In Section 5 we describe and discuss the algorithms employed to perform the operations and this includes, to some extent, programming techniques related to the data

structures that are used. Finally, an appendix contains FORTRAN listings of all the subprograms. These may be consulted for a more detailed study of the algorithms. They may also be used, together with similar listings in the previous SAC-1 documents, to implement SAC-1 on any computer with a FORTRAN compiler which fulfills the A.S.A. standards of [1]. At present, SAC-1 has been or is being implemented on CDC 1604, CDC 3600, CDC 6400, IBM 360/65, GE 645 and UNIVAC 1108 computers. The author would appreciate notification of other implementations.

Currently, a fifth SAC-1 system for operations on polynomials with rational function (or rational number) coefficients is in progress. This will be followed by a system for exact solution of linear systems with coefficients which are integers, rational numbers, polynomials or rational functions. Also in progress are an improved infinite-precision integer arithmetic system and an improved polynomial system, embodying faster algorithms for g.c.d. calculation.

2. External and Internal Data Structures

The canonical form of a non-zero rational function in the SAC-1 Rational Function system is a pair $(P(X_1,\ldots,X_n), Q(X_1,\ldots,X_n))$ such that P and Q are relatively prime and Q is a positive polynomial in the sense defined in [6]. P is the numerator and Q is the denominator. As implied by the notation, P and Q contain the same variables, $X_1,\ldots,X_n$, ordered in the same way. However, a polynomial need not depend on all the variables which it contains; either P or Q, or both, may be of degree zero in any given variable. In particular, for example, Q may be independent of all its variables, in which case Q is isomorphic to an integer, and (P,Q) is isomorphic to a polynomial with rational number coefficients. Also, n=o is allowed, in which case we obtain a rational number as a special case. Note that P and Q must be relatively prime in the strong sense that their greatest common divisor is 1 (not merely a polynomial of degree zero). Since, for any non-zero polynomial Q, exactly one of the polynomials Q or -Q is positive, it follows that the

canonical form is indeed unique, for a given ordering of the variables. But the ordering of variables for a given rational function may conveniently be changed using subprograms provided in the system.

Internally, the rational function (P,Q) is represented by the list (P*,Q*) where P* and Q* are the lists which represent P and Q, respectively, as defined in [6]. It follows that the list representing a non-zero rational function is easily distinguished from a list representing an infinite-precision integer by its properties of having a length of two and having a list as its first element. The rational function zero is treated as a special case and is represented by the null list, as are the infinite-precision integer zero and the polynomial zero. By convention, the location of the null list is zero. So a rational function is zero if and only if its location is zero.

If an arithmetic operation is to be performed on two rational functions, they must be compatible in the sense that they must contain the same variables, these variables being ordered the same in each. As a special case, however, the rational function zero is compatible with any rational function. Similarly, if one rational function is to be substituted into another, certain conditions are imposed on the variables and their ordering. These restrictions result in a system in which the operations are performed much more rapidly. In most applications the requirements are naturally satisfied most of the time and where they are not, the reordering operations may be conveniently applied.

The external form of a rational function is a string of characters defined in terms of the same canonical form that is used internally. We have defined in [6] the character string, $\overline{P}$, which represents externally any polynomial P. The character string which externally represents a non-zero rational function (P,Q) is defined to be the string $\overline{P}$bbb/bbb$\overline{Q}$ where each b represents a blank (blanks are actually used, not b's). The external canonical form of the rational function zero is just +0.

For input to the system, some slight deviations from external canonical form are permitted. The slash must occur either in the same record

as the last character of $\overline{P}$ or in the following record. Similarly, the first character of $\overline{Q}$ must occur either in the same record as the slash or in the following record. Apart from these requirements, the number of blanks on either side of the slash is arbitrary. It may even be zero. Recall, however, that if $\overline{P}$ is an integer, its last character is a blank following the last digit. For a description of the deviations permitted in the polynomials themselves, consult [6]. All output produced by the system is in strict canonical form and hence any punched card output may be used as input at a later time. The external character string of any rational function always begins in the first character position of a 72-character record, is continued into as many additional records as may be required, and the last record is completed with blanks.

3. The Subprogram Functions

The 15 rational function subprograms are divided below into five categories and each subprogram is defined with respect to its function and mode of use. Except as otherwise noted, subprograms do not redefine their arguments. Most of the subprograms are of the type function; these are distinguished below by the specification of a function value in the description.

3.1 Input-Output

The positioning of external character strings on records is specified above in Section 2. Whether output is printed or punched depends on the logical unit specified according to local convention, as explained in [4].

R=RREAD(U). U is a logical unit number. If unit U is initially positioned at the first record of a syntactically correct rational function, this rational function will be read and converted to internal canonical form as a list, R will be assigned as value the location of this list (a non-negative integer), and unit U will be left positioned at the next record following those which contained the rational function. If

unit U was initially positioned at an end-of-file, R is assigned the value -1. If a syntactic error is detected (not all errors are detected), R is assigned the value -2 and unit U is left positioned at some indeterminate record.

RWRITE (U,R). U is a logical unit number and R is a rational function (more precisely, the location of the internal list representation of a rational function). The rational function is converted to external canonical form and written on unit U as a sequence of records. Note that after output, the rational function still exists internally; if it is no longer needed, it should be erased using RERASE (described below). Note also that conversion and output proceed simultaneously; as soon as one record of output is produced it is written out and conversion of the next record commences.

3.2 Arithmetic

T=RSUM(R,S). R and S are compatible rational functions. T=R+S, a rational function compatible with R and S.

T=RDIF(R,S). R and S are compatible rational functions. T=R-S, a rational function compatible with R and S.

T=RPROD(R,S). R and S are compatible rational functions. T=R·S, a rational function compatible with R and S.

T=RQ(R,S). R and S are compatible rational functions, $S \neq 0$. T=R/S, a rational function compatible with R and S.

T=RINV(R). R is a non-zero rational function. $T=R^{-1} = 1/R$, a rational function compatible with R.

3.3 Differentiation

T=RDERIV(R,V). R is a rational function, V is a polynomial variable (as defined in [6]). T is the derivative of R with respect to V, a rational function compatible with R. Note that V need not occur in R; if it doesn't, the derivative is zero. In particular, R may be a rational number.

3.4 Substitution

T=RPSUB(R,P). R is a rational function, say $R(X_1,\ldots,X_n)$. P is a polynomial $P(X_1,\ldots,X_n,Y)$. If P is non-zero, then the numerator and denominator of R (unless R=o) must be compatible with P, and then T is the result of substituting R for Y in P, Y being the main variable of P, and T is compatible with R. If P=o, then T=o.

T=RSUBST(R,S). R and S are rational functions, $R(X_1,\ldots,X_n)$ and $S(X_1,\ldots,X_n,Y)$, Y being the main variable of S. $T=T(X_1,\ldots,X_n)$ is the result, $S(X_1,\ldots,X_n,R(X_1,\ldots,X_n))$, of substituting R for Y in S. As a special case, S may be zero and then T=o. Notice that this operation is undefined in case the result of substituting R into the denominator of S is zero. In this case T is given the value -1.

3.5 Others

RERASE(R). R is an arbitrary rational function. The list representation of R is erased in the sense of [4]. Thus RERASE has the same effect as ERASE when applied to a rational function, but RERASE executes considerably faster since it takes advantage of the known structure of the list representation of a rational function.

The following two subprograms are useful for reordering the variables of a rational function and introducing new variables as a means of satisfying the compatibility conditions for arithmetic and substitution.

L=RVLIST(R). R is an arbitrary rational function, say $R(X_1,\ldots,X_n)$. L is the list of variables $(X_1,\ldots,X_n)$, X_n being the main variable of R. If n=o, i.e. if R is zero or a rational number, then L is the null list.

T=RORDER(R,L). R is an arbitrary rational function, say $R(X_1,\ldots,X_m)$. L is a list of distinct variables $(Y_1,\ldots,Y_n)$ such that each X_i is some Y_j. Hence $o\leq m\leq n$. T is a rational function $T(Y_1,\ldots,Y_n)$ which is equivalent to $R(X_1,\ldots,X_m)$. In other words, T is the rational function resulting from reordering the variables or R and, possibly, introducing others. However, R remains intact.

The following two subprograms may be used to construct rational functions from polynomials.

T=RPOLY(P). P is an arbitrary polynomial, say $(P(X_1,\ldots,X_n)$. T is the rational function $T(X_1,\ldots,X_n)$ equivalent to $P(X_1,\ldots,X_n)$. That is, P is the numerator of T and the denominator of T is 1 expressed as a polynomial in $(X_1,\ldots,X_n)$, except that if P=0, then T is just zero.

T=RPOLY2(P,Q). P and Q are compatible polynomials $P(X_1,\ldots,X_n)$ and $Q(X_1,\ldots,X_n)$, $Q\neq 0$. T is the rational function $T(X_1,\ldots,X_n)=$ $P(X_1,\ldots,X_n)/Q(X_1,\ldots,X_n)$. Thus if P=0, then T=0. Otherwise, T is the rational function whose numerator is $\overline{P}$ and whose denominator is $\overline{Q}$, obtained by reducing P and Q to lowest terms with the further condition that $\overline{Q}$ is a positive polynomial.

Note especially the following application of RPOLY2. Suppose we wish to read a rational function P/Q, but we do not know whether P and Q are relatively prime and it would be difficult to determine this by hand. We cannot use RREAD since we might violate the relative primality requirement of canonical form. The simple solution is to read P and Q separately using the PREAD subprogram of [6], then apply RPOLY2. RREAD does not automatically reduce its inputs to lowest terms since this would be wasteful, polynomial greatest common divisor calculating being a relatively time consuming operation.

4. An Application Example

As an illustration of how to use the SAC-1 Rational Function System, and as a demonstration of its capability, we include here a main program, HILB, which generates an n by n Hilbert matrix $H^{(n)}_{i,j}$ defined by $H^{(n)}_{ij}=1/(i+j-1)$, and uses the system to invert it, exactly, and print out the result. HILB uses a subroutine, INVERT, to perform the inversion. INVERT is a general matrix inversion subroutine which can invert any non-singular matrix whose elements are compatible rational functions or rational numbers. As an example of inversion in the rational function case, we also wrote a program, GHILB, which generates the generalized Hilbert matrix $G^{(n)}$, defined by $G^{(n)}_{ij}= 1/(i+j-X)$, inverts it, and prints the

result. These generalized Hilbert matrices were previously considered by Engeli, [8], who has used his SYMBAL system to invert some of them.

The two main programs and the inversion subprogram are listed below at the end of this section. The inversion subprogram, INVERT, has three arguments, MA, N, and TWON, which are, respectively, a two-dimensional array name, the number of rows in the array, and the number of columns in the array. The number of columns in the array must be twice the number of rows, n. The first n columns of the array must contain the matrix to be inverted when INVERT is called, and the last n columns should be unused. INVERT first creates an nxn identity matrix in the last n columns. The given matrix is then inverted by the Gauss elimination method, the inverse appearing in the last n columns. During the inversion the given matrix is gradually altered and after the inversion all elements in the first n columns are zero. Whenever an element of the array is altered during the course of the inversion, the previous element is erased as a list. Of course the values actually stored in the array are just the locations of the lists which represent the rational function matrix elements.

In spite of its rather general usefulness, the subroutine INVERT is not considered to be a part of the Rational Function System. As mentioned in the introduction, a later SAC-1 subsystem will be dedicated to operations on linear systems with rational function and polynomial entries, including matrix inversion and solution of systems of equations. This subsystem will employ several algorithms which are generally much faster for exact calculation than Gauss elimination over the field of rational functions. More specifically, the algorithms will employ calculations carried out in finite fields, GF(p), with prime numbers of elements, as in [2], [10], and [11].

The HILB program was performed with N=5,10 and 15 on a CDC 1604. Computing times were respectively 9, 93 and 386 seconds. The GHILB program was performed with N=3,4 and 5 with corresponding computing times of 34,118 and 306 seconds. The need for faster algorithms is thus apparent. As a sample, a few of the lines of output for the cases N=15

and N=5 are reproduced following the programs.

The main programs display several principles and practices which are worth noting. Every SAC-1 main program must declare the labelled common blocks TR1 and TR2 as in lines 2 and 3. An array in this program called SPACE, must be declared as in line 4, which will be converted to the available space list by the subroutine BEGIN, as in line 9, prior to the call of any other SAC-1 subprogram. All SAC-1 subprogram names and all variables occurring in the program should be declared type integer as in lines 5, 6 and 7. SYMLST must be initialized to zero, as in line 8, prior to any SAC-1 subprogram call. Negligence in attending to any of these tedious little chores can, unfortunately, result in rather disastrous consequences.

Either of these programs can be used as a desirable test by anyone implementing the system. The last few lines embody what might be termed a complete erasure test. Each element of the inverse matrix is erased in statement 12 after it is printed. In the following statement the symbol list, SYMLST, which is automatically created during input as explained in [6], is erased. At this point all lists which were created should be erased and the available space list, AVAIL, should contain the same number of cells as it did initially. The next three statements compute and print out the length of AVAIL, in this case 5000 since each cell contains two words in the CDC 1604 implementation. If this test fails, the system should be carefully checked over to isolate the source of difficulty.

```
      PROGRAM HILB
      COMMON /TR1/ AVAIL,STAK,RECORD(72)
      COMMON /TR2/ SYMLST
      DIMENSION SPACE(10000),H(15,30)
      INTEGER AVAIL,STAK,RECORD,SYMLST,SPACE,H
      INTEGER PREAD,PFA,PFL,PVBL,PDIF,LENGTH
      INTEGER IN,OUT,TWON,POLX,T1,T2,DEN,NUM
      SYMLST=0
      CALL BEGIN(SPACE,10000)
      IN=50
      OUT=51
      N=15
      TWON=2*N
      DO 10 I=1,N
      DO 10 J=1,N
      T1=PFA(1,0)
      T2=PFA(I+J-1,0)
10    H(I,J)=PFL(T1,PFL(T2,0))
      T1=TIMEF(0)
      CALL INVERT(H,N,TWON)
      T2=TIMEF(0)
      L=(T2-T1)/1000
      PRINT 11,L
11    FORMAT (19H1TIME FOR INVERSION,I5,8H SECONDS//)
      NPLUS1=N+1
      DO 12 I=1,N
      DO 12 J=NPLUS1,TWON
      K=J-N
      PRINT 13,I,K
13    FORMAT (4HROW,I3,9H   COLUMN,I3)
      CALL RWRITE(OUT,H(I,J))
12    CALL ERASE(H(I,J))
      CALL ERASE(SYMLST)
      L=LENGTH(AVAIL)
      PRINT 14,L
14    FORMAT(16H1LENGTH OF AVAIL,I7)
      STOP
      END
```

```
      PROGRAM GHILB
      COMMON /TR1/ AVAIL,STAK,RECORD(72)
      COMMON /TR2/ SYMLST
      DIMENSION SPACE(10000),H(5,10)
      INTEGER AVAIL,STAK,RECORD,SYMLST,SPACE,H
      INTEGER PREAD,PFA,PFL,PVBL,PDIF,LENGTH
      INTEGER IN,OUT,TWON,POLX,T1,T2,DEN,NUM
      SYMLST=0
      CALL BEGIN(SPACE,10000)
      IN=50
      OUT=51
      N=5
      TWON=2*N
      POLX=PREAD(IN)
      DO 10 I=1,N
      DO 10 J=1,N
      T1=PFA(I+J,0)
      T2=PFL(PVBL(POLX),PFL(T1,PFA(0,0)))
      DEN=PDIF(POLX,T2)
      CALL PERASE(T2)
      T1=PFA(-1,0)
      NUM=PFL(PVBL(POLX),PFL(T1,PFA(0,0)))
10    H(I,J)=PFL(NUM,PFL(DEN,0))
      CALL PERASE(POLX)
      T1=TIMEF(0)
      CALL INVERT(H,N,TWON)
      T2=TIMEF(0)
      L=(T2-T1)/1000
      PRINT 11,L
11    FORMAT (19H1TIME FOR INVERSION,I5,8H SECONDS//)
      NPLUS1=N+1
      DO 12 I=1,N
      DO 12 J=NPLUS1,TWON
      K=J-N
      PRINT 13,I,K
13    FORMAT (4HOROW,I3,9H   COLUMN,I3)
      CALL RWRITE(OUT,H(I,J))
12    CALL RERASE(H(I,J))
      CALL ERASE(SYMLST)
      L=LENGTH(AVAIL)
      PRINT 14,L
14    FORMAT(16H1LENGTH OF AVAIL,I7)
      STOP
      END
```

```
      SUBROUTINE INVERT(MA,N,TWON)
      DIMENSION MA(N,TWON)
      COMMON /TR1/ AVAIL,STAK,RECORD(72)
      COMMON /TR2/ SYMLST
      INTEGER AVAIL,STAK,RECORD,SYMLST
      INTEGER BORROW,PFA,PFL
      INTEGER RORDER,RVLIST,RINV,RPROD,RDIF
      INTEGER RNONE,RFONE,VL,TWON,TEMP,TEMP2,RPIVOT
C     *****CREATION OF IDENTITY MATRIX*****
      IONE=PFA(1,0)
      RNONE=PFL(BORROW(IONE),PFL(IONE,0))
      DO 10 I=1,N
      IF (MA(I,1).NE.0) GO TO 11
10    CONTINUE
      GO TO 80
11    VL=RVLIST(MA(I,1))
      RFONE=RORDER(RNONE,VL)
      CALL RERASE(RNONE)
      CALL ERASE(VL)
      NPLUS1=N+1
      DO 15 I=1,N
      DO 15 J=NPLUS1,TWON
      MA(I,J)=0
      IF ((N+I).EQ.J) MA(I,J)=BORROW(RFONE)
15    CONTINUE
      CALL RERASE(RFONE)
C     *****TRIANGULARIZATION*****
      DO 36 J=1,N
      JPLUS1=J+1
      IF (MA(J,J).NE.0) GO TO 30
      IF (J.EQ.N) GO TO 80
      DO 20 I=JPLUS1,N
      IF (MA(I,J).NE.0) GO TO 21
20    CONTINUE
      GO TO 80
21    DO 22 K=J,TWON
      TEMP=MA(I,K)
      MA(I,K)=MA(J,K)
22    MA(J,K)=TEMP
30    RPIVOT=RINV(MA(J,J))
      DO 31 I=J,TWON
      TEMP=MA(J,I)
      MA(J,I)=RPROD(MA(J,I),RPIVOT)
31    CALL RERASE(TEMP)
      CALL RERASE(RPIVOT)
      IF (J.EQ.N) GO TO 40
      DO 36 I=JPLUS1,N
      DO 35 K=JPLUS1,TWON
      TEMP=MA(I,K)
      TEMP2=RPROD(MA(I,J),MA(J,K))
      MA(I,K)=RDIF(TEMP,TEMP2)
      CALL RERASE(TEMP)
35    CALL RERASE(TEMP2)
      CALL RERASE(MA(I,J))
36    MA(I,J)=0
C     *****DIAGONALIZATION*****
40    J=N
41    I=J-1
42    DO 45 K=NPLUS1,TWON
```

```
      TEMP=MA(I,K)
      TEMP2=RPROD(MA(I,J),MA(J,K))
      MA(I,K)=RDIF(TEMP,TEMP2)
      CALL RERASE(TEMP)
45    CALL RERASE(TEMP2)
      CALL RERASE(MA(I,J))
      MA(I,J)=0
      I=I-1
      IF (I.NE.0) GO TO 42
      J=J-1
      IF (J.NE.1) GO TO 41
C     *****ERASE DIAGONAL*****
      DO 50 I=1,N
      CALL RERASE(MA(I,I))
50    MA(I,I)=0
      RETURN
C     *****PRINT DIAGNOSTIC*****
80    PRINT 81
81    FORMAT (16H1SINGULAR MATRIX)
      STOP
      END
```

INVERSE OF 15X15 HILBERT MATRIX.

ROW 15 COLUMN 1
+116338140C / +1

ROW 15 COLUMN 2
-244310094000 / +1

ROW 15 COLUMN 3
+12704124888C0C / +1

ROW 15 COLUMN 4
-287960164128000 / +1

ROW 15 COLUMN 5
+3563507031084000 / +1

ROW 15 COLUMN 6
-27082653436238400 / +1

ROW 15 COLUMN 7
+135413267181192000 / +1

ROW 15 COLUMN 8
-464274058906944000 / +1

ROW 15 COLUMN 9
+111715945424483400 / +1

ROW 15 COLUMN 10
-190330869582453200 / +1

ROW 15 COLUMN 11
+228397043978943840C / +1

ROW 15 COLUMN 12
-188757887585904000 / +1

ROW 15 COLUMN 13
+1022438557756980000 / +1

ROW 15 COLUMN 14
-326656343898680000 / +1

ROW 15 COLUMN 15
+466709062712400000 / +1

INVERSE OF 5X5 GENERALIZED HILBERT MATRIX

ROW 5 COLUMN 1
(-1X**9+54X**8-1266X**7+16884X**6-140889X**5+761166X**4-2655764X**3+5735736X**2-6999840X**1+3628800X**0) / (+576X**0)

ROW 5 COLUMN 2
(+1X**9-58X**8+1474X**7-21532X**6+199129X**5-1208242X**4+4806276X**3-12075768X**2+17370720X**1-10886400X**0) / (+144X**0)

ROW 5 COLUMN 3
(-1X**9+62X**8-1694X**7+26768X**6-269549X**5+1793498X**4-7883476X**3+22069272X**2-35693280X**1+25401600X**0) / (+96X**0)

ROW 5 COLUMN 4
(+1X**9-66X**8+1926X**7-32616X**6+353229X**5-2536974X**4+12083564X**3-368C2824X**2+6503616CX**1-50803200X**0) / (+144X**0)

ROW 5 COLUMN 5
(-1X**9+7CX**8-2170X**7+39100X**6-451273X**5+3459670X**4-17617980X**3+57465000X**2-1C8936576X**1+9144576CX**0) / (+576X**0)

5. The Algorithms

In the following we describe and discuss the algorithms employed in the subprograms for the arithmetic operations, differentiation and substitution. The subprograms for input-output and the miscellaneous operations are quite trivial and uninteresting, and the listings in the appendix should be consulted directly.

The algorithms for the arithmetic operations are those which were previously used by Brown in the ALPAK system [3] and, before that, by Henrici, [9], in subroutines for single-precision rational number arithmetic. The differentiation algorithm is also due to Brown. Henrici introduced his algorithms to avoid single-precision overflow of intermediate results insofar as possible. Brown rediscovered these algorithms as a means of minimizing the time required to calculate greatest common divisors in reducing results to lowest terms, and devised a similar algorithm for differentiation. For convenient reference these algorithms are reproduced below and are followed by a rough analysis of the extent to which they reduce computing time.

T=RSUM(R,S). If R=0, set T=S; if S=0, set T=R. Otherwise, we have $R=R_1/R_2$ and $S=S_1/S_2$ where gcd $(R_1,R_2)=1$, gcd $(S_1,S_2)=1$, and R_2 and S_2 are positive polynomials. Next compute B=gcd (R_2, S_2). If B=1, then compute $T_1=R_1S_2+R_2S_1$. If $T_1=0$, then T=0. Otherwise, compute $T_2=R_2S_2$. From B=1 it follows that gcd $(T_1,T_2)=1$. Also, T_2 is a positive polynomial so $T=T_1/T_2$ is in canonical form. If $B\neq 1$, compute $\overline{R}_2=R_2/B$ and $\overline{S}_2=S_2/B$, then $T_1=R_1\overline{S}_2+S_1\overline{R}_2$. If $\overline{T}_1=0$, set T=0. Otherwise, compute $\overline{T}_2=R_2\overline{S}_2$. Next, compute C=gcd $(\overline{T}_1,B)$. If C=1, then $T=\overline{T}_1/\overline{T}_2$ is in canonical form. Otherwise, compute $T_1^*=\overline{T}_1/C$ and $T_2^*=\overline{T}_2/C$. Then $T=T_1^*/T_2^*$, is in canonical form.

T=RDIF(R,S). If S=0, T=R. Otherwise, $S=S_1/S_2$ and RSUM is applied to R and $-S=(-S_1)/S_2$.

T=RPROD(R,S). If R=0 or S=0, then T=0. Otherwise, we have $R=R_1/R_2$ and $S=S_1/S_2$ where gcd $(R_1,R_2)=1$, gcd $(S_1,S_2)=1$, and R_2 and S_2 are positive polynomials. Compute A=gcd (R_1,S_2) and B=gcd (R_2,S_1). Then

compute $\overline{R}_1=R_1/A$, $\overline{S}_2=S_2/A$, $\overline{R}_2=R_2/B$ and $\overline{S}_1=S_1/B$, except that if A=1 simply set $\overline{R}_1=R_1$ and $\overline{S}_2=S_2$, and if B=1, set $\overline{R}_2=R_2$ and $\overline{S}_1=S_1$. Finally, compute $\overline{T}_1=\overline{R}_1\overline{S}_1$ and $\overline{T}_2=\overline{R}_2\overline{S}_2$. Then $T=\overline{T}_1/\overline{T}_2$ in canonical form.

T=RQ(R,S). If R=0, set T=0. Otherwise T=RPROD(R,S^{-1}). If $S=S_1/S_2$ in canonical form, then $S^{-1}=S_2/S_1$ or $(-S_2)/(-S_1)$ in canonical form according as to whether S_1 is a positive polynomial or not.

T=RINV(S). S^{-1} is computed as in RQ above.

T=RDERIV(R,V). If R=0, set T=0. Otherwise, we have $R=R_1/R_2$ in canonical form. For any F let F′ be the derivative of F with respect to V. Compute R_1' and R_2'. If $R_2'=0$ and $R_1'=0$, set T=0. If $R_2'=0$ and $R_1'\neq 0$, compute $G=\gcd(R_1',R_2)$. If G=1, $T=R_1'/R_2$ in canonical form. If G≠1, compute $T_1=R_1'/G$ and $T_2=R_2/G$. Then $T=T_1/T_2$ in canonical form. If $R_2'\neq 0$, compute $G=\gcd(R_2,R_2')$. If G=1, compute $T_1=R_1'R_2-R_1R_2'$. If $T_1=0$, set T=0. Otherwise, compute $T_2=R_2^2$. Then $T=T_1/T_2$ in canonical form. If G≠1, compute $\overline{R}_2=R_2/G$ and $\overline{R}_2'=R_2'/G$, then $\overline{T}_1=R_1'\overline{R}_2-R_1\overline{R}_2'$. If $\overline{T}_1=0$, set T=0. Otherwise, compute $T_2=R_2\overline{R}_2$, then $H=\gcd(\overline{T}_1,G)$. If H=1, then $T=\overline{T}_1/\overline{T}_2$ in canonical form. If H≠1, compute $T_1^*=\overline{T}_1/H$ and $T_2^*=\overline{T}_2/H$. Then $T=T_1^*/T_2^*$ in canonical form.

A thorough analysis of the Henrici-Brown algorithms above for addition, multiplication and differentiation would be a considerable task which will not be attempted here. Since the three algorithms are of the same general nature, we shall restrict attention to multiplication, this being the simplest of the three.

We first consider a one-parameter family of cases in which two rational numbers are to be multiplied. We assume that R and S are non-zero and, in fact, that the numerators and denominators of R and S all contain approximately d digits (the base is immaterial). We will also assume that $A=\gcd(R_1,S_2)$ and $B=\gcd(R_2,S_1)$ are each approximately td digits along. As a result of the considerations in [7], it is reasonable to assume that the time to compute either A or B will be approximately $G(1-t)d^2$, for some constant G. Also, the time to compute $\overline{R}_1,\overline{R}_2,\overline{S}_1$ or $\overline{S}_2$ will be approximately $Dt(1-t)d^2$, for some constant D. The time to

compute $\overline{T}_1$ or $\overline{T}_2$ will be approximately $M(1-t)^2d^2$, for some constant M. So the total computing time will be $[2G(1-t)+4Dt(1-t)+2M(1-t)^2]d^2$ approximately. If the same approximations are used to estimate the time for the classical multiplication algorithm ($T_1=R_1S_1$, $T_2=R_2S_2$, $A=$ gcd (T_1,T_2), $\overline{T}_1=T_1/A$, $\overline{T}_2=T_2/A$), we obtain $[4G(1-t)+4Dt(1-t)+2M]d^2$ as the total computing time. This shows that the Henrici-Brown algorithm is always faster for this class of problems. In general, the constant G will be appreciably larger than either M or D. This being so, the Henrici-Brown algorithm will be approximately twice as fast as the classical algorithm for the class of problems considered.

Now suppose that R and S are univariate rational functions. Then we know that most of the computing time for either algorithm will be devoted to g.c.d. calculation, since the time to compute the g.c.d. of two n-th degree polynomials is approximately proportional to n^4 by [7]. It follows easily from this that the Henrici-Brown algorithm will be approximately eight times as fast as the classical algorithm in this case.

As the number of variables in our rational functions increase, it is fairly clear that the advantage of the Henrici-Brown algorithms will increase. While this analysis is far from definitive, we think it provides strong reasons for attaching considerable importance to the Henrici-Brown algorithms.

Now we consider the substitution algorithm.

T=RPSUB(R,P). If P=0, T=0. If R=0, PSUBST is used and the result is converted to a rational function using RPOLY. Otherwise, the substitution is effected by Horner's method.

T=RSUBST(R,S). If S=0, then T=0. Otherwise, let $S=S_1/S_2$. V=RPSUB (R,S_2) is computed. If V=0, the substitution is undefined and T=-1. Otherwise, U=RPSUB(R,S_1) is computed. If U=0, then T=0. Otherwise, T=U/V, which is computed by RQ.

7. Acknowledgements

Development of the SAC-1 Rational Function System has been supported by the University of Wisconsin Computing Center through funding by the Graduate School and the Wisconsin Alumni Research Foundation. L. E. Heindel, Ellis Horowitz and M. T. McClellan were all responsible for significant portions of the programming and testing. Their sustained interest and assistance have contributed significantly to the continuing development of SAC-1. I would also like to express my appreciation to all those persons, at the University of Wisconsin, throughout the United States, and in many other countries, who in numerous ways have expressed their interest in the earlier portions of the system, and in the objectives of those portions yet to come.

References

1. American Standards Association, A Programming Language for Information Processing on Automatic Data Processing Systems, C.A.C.M. Vol. 7, No. 10 (October 1964), pp. 591-625.

2. Borosh, I. and Fraenkel, A.S. Exact Solutions of Linear Equations with Rational Coefficients by Congruence Techniques. Math. Comp:, Vol. 20, No. 93 (January 1966), pp. 107-112.

3. Brown, W. S., Hyde, J. P., and Tague, B. A., The ALPAK System for Nonnumerical Algebra on a Digital Computer - II: Rational Functions of Several Variables and Truncated Power Series with Rational Function Coefficients. B.S.T.J. Vol. 43, No. 1 (March, 1964), pp. 785-804.

4. Collins, G. E. The SAC-1 List Processing System. University of Wisconsin Computing Center Report, July 1967.

5. Collins, G. E. The SAC-1 Integer Arithmetic System. University of Wisconsin Computing Center Report, September 1967.

6. Collins G. E. The SAC-1 Polynomial System. University of Wisconsin Technical Reference 2: 1968, January 1968.

7. Collins G. E. Computing Time Analyses of Some Arithmetic and Albegraic Algorithms. University of Wisconsin Computer Sciences Technical Report 36, July 1968.

8. Engeli, M. E. Achievements and Problems in Formula Manipulation. Invited Paper for IFIP Congress, August 1968.

9. Henrici, Peter. A Subroutine for Computations with Rational Numbers. J.A.C.M., Vol. 3, No. 1 (January 1956), pp. 6-9.

10. Newman, Morris. Solving Equations Exactly. Journal of Research N.B.S., Vol. 71B, No. 4 (October-December 1967), pp. 171-179.

11. Takahasi, H. and Ishibashi, Y. A New Method for "Exact Calculation" by a Digital Computer. Inf. Proc. in Japan, Vol. 1 (1961), pp. 28-42.

```
      INTEGER FUNCTION RDERIV(R,S)
      INTEGER A1B,A2B,G,H,HOLD1,HOLD2,P,P1,P2,P1P,P2P,P2B,P2BP,R,TEMP,S
      INTEGER BORROW,PDERIV,PFL,PGCD,PONE,PPROD,PQ,PDIF
      P=R
      G=0
      RDERIV=0
      TEMP=S
      IF (P.EQ.0) RETURN
      CALL ADV(P1,P)
      CALL ADV(P2,P)
      P1P=PDERIV(P1,TEMP)
      P2P=PDERIV(P2,TEMP)
      P2B=BORROW(P2)
      P2BP=BORROW(P2P)
      IF (P2P.EQ.0) GO TO 4
      G=PGCD(P2,P2P)
      TEMP=PONE(G)
      IF (TEMP.EQ.1) GO TO 1
      P2B=PQ(P2,G)
      CALL PERASE(P2)
      P2BP=PQ(P2P,G)
      CALL PERASE(P2P)
    1 HOLD1=PPROD(P1P,P2B)
      HOLD2=PPROD(P1,P2BP)
      CALL PERASE(P1P)
      CALL PERASE(P2P)
      A1B=PDIF(HOLD1,HOLD2)
      A2B=PPROD(P2,P2B)
      CALL PERASE(HOLD1)
      CALL PERASE(HOLD2)
      CALL PERASE(P2B)
      CALL PERASE(P2BP)
      P1=BORROW(A1B)
      P2=BORROW(A2B)
      IF (A1B.EQ.0) RETURN
      IF (TEMP.EQ.1) GO TO 2
      H=PGCD(A1B,G)
    6 TEMP=PONE(H)
      IF (TEMP.EQ.1) GO TO 3
      P1=PQ(A1B,H)
      P2=PQ(A2B,H)
      CALL PERASE(A1B)
      CALL PERASE(A2B)
    3 CALL PERASE(H)
    2 RDERIV=PFL(P1,PFL(P2,0))
      CALL PERASE(A1B)
      CALL PERASE(A2B)
      CALL PERASE(G)
      RETURN
    4 IF (P1P.NE.0) GO TO 5
      CALL PERASE(P2B)
      CALL PERASE(P2BP)
      RETURN
```

```
    5 P1=P1P
      CALL PERASE(P2P)
      CALL PERASE(P2B)
      CALL PERASE(P2BP)
      A1B=BORROW(P1)
      A2B=BORROW(P2)
      H=PGCD(A1B,P2)
      GO TO 6
      END

      INTEGER FUNCTION RDIF(X,Y)
      INTEGER P,Q,Q1,Q2,Z,X,Y
      INTEGER BORROW,PFL,PNEG,RSUM
      P=X
      Q=Y
      IF (Q.NE.0) GO TO 1
      RDIF=BORROW(P)
      RETURN
    1 CALL ADV(Q1,Q)
      CALL ADV(Q2,Q)
      Q=PNEG(Q1)
      Z=PFL(Q,PFL(BORROW(Q2),0))
      RDIF=RSUM(P,Z)
      CALL RERASE(Z)
      RETURN
      END

      SUBROUTINE RERASE(X)
      INTEGER X,R,P,N
      INTEGER COUNT
      R=X
      IF (R.EQ.0) RETURN
    2 N=COUNT(R)-1
      IF (N.EQ.0) GO TO 1
      CALL SCOUNT(N,R)
      RETURN
    1 CALL DECAP(P,R)
      CALL PERASE(P)
      IF (R.NE.0) GO TO 2
      RETURN
      END
```

```
      INTEGER FUNCTION RINV(R)
      COMMON /TR1/AVAIL,STAK,RECORD(72)
      COMMON /TR2/SYMLST
      INTEGER AVAIL,STAK,RECORD,SYMLST
      INTEGER DEM,NUM,R,RAT
      INTEGER BORROW,PFL,PNEG,PSIGN
      RAT=R
      CALL ADV(NUM,RAT)
      CALL ADV(DEM,RAT)
      IF (PSIGN(NUM).EQ.-1) GO TO 1
      NUM=BORROW(NUM)
      DEM=BORROW(DEM)
2     RINV=PFL(DEM,PFL(NUM,0))
      RETURN
1     NUM=PNEG(NUM)
      DEM=PNEG(DEM)
      GO TO 2
      END

      INTEGER FUNCTION RORDER(R,L)
      COMMON /TR1/AVAIL,STAK,RECORD(72)
      COMMON /TR2/SYMLST
      INTEGER AVAIL,STAK,RECORD,SYMLST
      INTEGER DDEM,DEM,L,NUM,R,RAT,VARS,DNUM
      INTEGER PFL,PNEG,PORDER,PSIGN,BORROW
      RAT=R
      VARS=L
      IF (RAT.EQ.0) GO TO 2
      IF (VARS.EQ.0) GO TO 3
      CALL ADV(NUM,RAT)
      CALL ADV(DEM,RAT)
      NUM=PORDER(NUM,VARS)
      DEM=PORDER(DEM,VARS)
      IF (PSIGN(DEM).NE.-1) GO TO 1
      DNUM=NUM
      NUM=PNEG(NUM)
      CALL PERASE(DNUM)
      DDEM=DEM
      DEM=PNEG(DEM)
      CALL PERASE(DDEM)
1     RORDER=PFL(NUM,PFL(DEM,0))
      RETURN
2     RORDER=0
      RETURN
3     RORDER=BORROW(RAT)
      RETURN
      END
```

```
      INTEGER FUNCTION RPOLY(P)
      COMMON /TR1/AVAIL,STAK,RECORD(72)
      COMMON /TR2/SYMLST
      INTEGER AVAIL,STAK,RECORD,SYMLST
      INTEGER DEM,DUM,NUM,P,VARS
      INTEGER BORROW,PFA,PFL,PORDER,PVLIST
      NUM=P
      IF (NUM.EQ.0) GO TO 2
      VARS=PVLIST(NUM)
      DEM=PFA(1,0)
      IF (VARS.EQ.0) GO TO 1
      DUM=DEM
      DEM=PORDER(DEM,VARS)
      CALL PERASE(DUM)
      CALL ERASE(VARS)
1     RPOLY=PFL(BORROW(NUM),PFL(DEM,0))
      RETURN
2     RPOLY=0
      RETURN
      END

      INTEGER FUNCTION RPOLY2(P1,P2)
      COMMON /TR1/AVAIL,STAK,RECORD(72)
      COMMON /TR2/SYMLST
      INTEGER AVAIL,STAK,RECORD,SYMLST
      INTEGER DEM,DUM,GCD,NUM,P1,P2
      INTEGER BORROW,PGCD,PFL,PNEG,PONE,PQ,PSIGN
      NUM=P1
      DEM=P2
      IF (NUM.NE.0) GO TO 1
      RPOLY2=0
      RETURN
1     GCD=PGCD(NUM,DEM)
      IF (PONE(GCD).NE.1) GO TO 2
      NUM=BORROW(NUM)
      DEM=BORROW(DEM)
      GO TO 3
2     NUM=PQ(NUM,GCD)
      DEM=PQ(DEM,GCD)
3     CALL PERASE(GCD)
      IF (PSIGN(DEM).EQ.1) GO TO 4
      DUM=NUM
      NUM=PNEG(NUM)
      CALL PERASE(DUM)
      DUM=DEM
      DEM=PNEG(DEM)
      CALL PERASE(DUM)
4     RPOLY2=PFL(NUM,PFL(DEM,0))
      RETURN
      END
```

```
      INTEGER FUNCTION RPROD(X,Y)
      COMMON /TR1/AVAIL,STAK,RECORD(72)
      COMMON /TR2/SYMLST
      INTEGER AVAIL,STAK,RECORD,SYMLST
      INTEGER P,P1,P1B,P2,P2B,Q,Q1,Q1B,Q2,Q2B,A,TEMP,X,Y
      INTEGER BORROW,PFL,PGCD,PONE,PPROD,PQ
      P=X
      Q=Y
      RPROD=0
      IF (P.NE.0) GO TO 1
      RETURN
1     IF (Q.NE.0) GO TO 2
      RETURN
2     CALL ADV(P1,P)
      CALL ADV(P2,P)
      P1=BORROW(P1)
      P2=BORROW(P2)
      CALL ADV(Q1,Q)
      CALL ADV(Q2,Q)
      Q1=BORROW(Q1)
      Q2=BORROW(Q2)
      A=PGCD(P1,Q2)
      P1B=P1
      Q2B=Q2
      TEMP=PONE(A)
      IF (TEMP.EQ.1) GO TO 3
      P1B=PQ(P1,A)
      Q2B=PQ(Q2,A)
      CALL PERASE(P1)
      CALL PERASE(Q2)
3     CALL PERASE(A)
      P2B=P2
      Q1B=Q1
      A=PGCD(P2,Q1)
      TEMP=PONE(A)
      IF (TEMP.EQ.1) GO TO 4
      P2B=PQ(P2,A)
      Q1B=PQ(Q1,A)
      CALL PERASE(P2)
      CALL PERASE(Q1)
4     CALL PERASE(A)
      P=PPROD(P1B,Q1B)
      CALL PERASE(P1B)
      CALL PERASE(Q1B)
      Q=PPROD(P2B,Q2B)
      CALL PERASE(P2B)
      CALL PERASE(Q2B)
      RPROD=PFL(P,PFL(Q,RPROD))
      RETURN
      END
```

```
      INTEGER FUNCTION RPSUB(R1,P1)
      COMMON /TR1/AVAIL,STAK,RECORD(72)
      COMMON /TR2/SYMLST
      INTEGER AVAIL,STAK,RECORD,SYMLST
      INTEGER R,R1,P,P1,PI,PJ,FLAG,DUMP,IP,EJ
      INTEGER RPOLY,RSUM,RPROD,PSUBST
      R=R1
      P=P1
      RPSUB=0
      IF (P.EQ.0) RETURN
      IF (R.EQ.0) GO TO 5
      FLAG=0
      CALL ADV(DUMP,P)
      CALL ADV(PI,P)
      CALL ADV(IP,P)
6     IF (P.NE.0) GO TO 1
      FLAG=1
      GO TO 2
1     CALL ADV(PJ,P)
      CALL ADV(EJ,P)
      IP=IP-EJ
2     DUMP=RPSUB
      PI=RPOLY(PI)
      RPSUB=RSUM(DUMP,PI)
      CALL RERASE(PI)
      CALL RERASE(DUMP)
      IF (IP.EQ.0) RETURN
      DO 3 I=1,IP
      DUMP=RPSUB
      RPSUB=RPROD(DUMP,R)
3     CALL RERASE(DUMP)
4     IF (FLAG.EQ.1) RETURN
      PI=PJ
      IP=EJ
      GO TO 6
5     DUMP=PSUBST(R,P)
      RPSUB=RPOLY(DUMP)
      CALL PERASE(DUMP)
      RETURN
      END
```

```
      INTEGER FUNCTION RQ(X,Y)
      COMMON /TR1/AVAIL,STAK,RECORD(72)
      COMMON /TR2/SYMLST
      INTEGER AVAIL,STAK,RECORD,SYMLST
      INTEGER P,Q,Q1,Q2,TEMP,X,Y
      INTEGER BORROW,PSIGN,PNEG,PFL,RPROD
      P=X
      Q=Y
      IF (P.NE.0) GO TO 1
      RQ=0
      RETURN
1     CALL ADV(Q1,Q)
      CALL ADV(Q2,Q)
      Q1=BORROW(Q1)
      Q2=BORROW(Q2)
      TEMP=PSIGN(Q1)
      IF (TEMP.NE.-1) GO TO 2
      CALL PERASE(Q1)
      Q1=PNEG(Q1)
      CALL PERASE(Q2)
      Q2=PNEG(Q2)
2     Q=PFL(Q2,PFL(Q1,0))
      RQ=RPROD(P,Q)
      CALL RERASE(Q)
      RETURN
      END
```

```
      INTEGER FUNCTION RREAD(U)
      COMMON /TR1/AVAIL,STAK,RECORD(72)
      COMMON /TR2/SYMLST
      INTEGER AVAIL,STAK,RECORD,SYMLST
      INTEGER I,J,K,P,Q,R,U
      INTEGER PFL
      CALL READ(U,RECORD)
      Q=RECORD(1)
      IF (Q.EQ.-1) GO TO 11
      I=1
      CALL PREADS(Q,U,I)
      IF (Q.EQ.-1.OR.Q.EQ.-2.OR.Q.EQ.0) GO TO 11
      P=Q
      J=0
      IF (I.LT.72) GO TO 3
1     CALL READ(U,RECORD)
      Q=RECORD(1)
      IF (Q.EQ.-1) GO TO 10
      I=0
      J=1
3     I=I+1
      Q=-2
      DO 4 K=I,72
      R=RECORD(K)
      IF (R.EQ.39) GO TO 5
4     IF (R.NE.44) GO TO 10
      IF (J) 1,1,10
5     I=K
      J=0
      IF (I.LT.72) GO TO 7
6     I=0
      J=1
      CALL READ(U,RECORD)
      Q=RECORD(1)
      IF (Q.EQ.-1) GO TO 10
7     I=I+1
      Q=-2
      DO 8 K=I,72
8     IF (RECORD(K).NE.44) GO TO 9
      IF (J) 6,6,10
9     CALL PREADS(Q,U,K)
      IF (Q.EQ.-1.OR.Q.EQ.-2) GO TO 10
      RREAD=PFL(P,PFL(Q,0))
      RETURN
10    CALL PERASE(P)
11    RREAD=Q
      RETURN
      END
```

```
      INTEGER FUNCTION RSUBST(R1,S1)
      COMMON /TR1/AVAIL,STAK,RECORD(72)
      COMMON /TR2/SYMLST
      INTEGER AVAIL,STAK,RECORD,SYMLST
      INTEGER R,R1,S,S1,SDEM,SNUM,U,V
      INTEGER RPSUB,RQ
      R=R1
      S=S1
      IF (S.NE.0) GO TO 9
10    RSUBST=0
      RETURN
9     CALL ADV(SNUM,S)
      CALL ADV(SDEM,S)
      V=RPSUB(R,SDEM)
      IF (V.NE.0) GO TO 1
17    RSUBST=-1

      RETURN
1     U=RPSUB(R,SNUM)
      IF (U.NE.0) GO TO 3
      CALL RERASE(V)
      GO TO 10
3     RSUBST=RQ(U,V)
      CALL RERASE(U)
      CALL RERASE(V)
      RETURN
      END
```

```
      INTEGER FUNCTION RSUM(X,Y)
      COMMON /TR1/AVAIL,STAK,RECORD(72)
      COMMON /TR2/SYMLST
      INTEGER AVAIL,STAK,RECORD,SYMLST
      INTEGER B,P,P1,P2,P3,Q,Q1,Q2,TEMP,Z1,Z2,X,Y
      INTEGER BORROW,PGCD,PONE,PFL,PPROD,PQ,PSUM
      P=X
      Q=Y
      IF (P.NE.0) GO TO 1
      RSUM=BORROW(Q)
      RETURN
1     IF (Q.NE.0) GO TO 2
      RSUM=BORROW(P)
      RETURN
2     CALL ADV(P1,P)
      CALL ADV(P2,P)
      P2=BORROW(P2)
      P3=BORROW(P2)
      CALL ADV(Q1,Q)
      CALL ADV(Q2,Q)
      Q2=BORROW(Q2)
      B=PGCD(Q2,P2)
      TEMP=PONE(B)
      IF (TEMP.EQ.1) GO TO 3
      P=PQ(P2,B)
      CALL PERASE(P2)
      Q=PQ(Q2,B)
      CALL PERASE(Q2)
      P2=P
      Q2=Q
3     P=PPROD(P1,Q2)
      Q=PPROD(Q1,P2)
      CALL PERASE(P2)
      Z1=PSUM(P,Q)
      CALL PERASE(P)
      CALL PERASE(Q)
      IF (Z1.NE.0) GO TO 6
      CALL PERASE(P3)
      CALL PERASE(B)
      CALL PERASE(Q2)
      RSUM=0
      RETURN
6     Z2=PPROD(P3,Q2)
      CALL PERASE(P3)
      CALL PERASE(Q2)
      IF (TEMP.EQ.1) GO TO 4
      P=PGCD(B,Z1)
      TEMP=PONE(P)
      IF (TEMP.EQ.1) GO TO 5
      Q=PQ(Z1,P)
```

```
      CALL PERASE(Z1)
      Z1=Q
      Q2=PQ(Z2,P)
      CALL PERASE(Z2)
      Z2=Q2
5     CALL PERASE(P)
4     CALL PERASE(B)
      RSUM=PFL(Z1,PFL(Z2,C))
      RETURN
      END

      INTEGER FUNCTION RVLIST(R)
      COMMON /TR1/AVAIL,STAK,RECORD(72)
      COMMON /TR2/SYMLST
      INTEGER AVAIL,STAK,RECORD,SYMLST
      INTEGER R,S,T
      INTEGER FIRST,PVLIST,TYPE
      S=R
      IF (S.EQ.0) GO TO 1
      T=FIRST(S)
      IF (TYPE(T).EQ.C) GO TO 1
      RVLIST=PVLIST(T)
      RETURN
1     RVLIST=0
      RETURN
      END
```

```
      SUBROUTINE RWRITE(U,R)
      COMMON /TR1/AVAIL,STAK,RECORD(72)
      COMMON /TR2/SYMLST
      INTEGER AVAIL,STAK,RECORD,SYMLST
      DIMENSION D(7)
      DATA (D(I),I=1,7)/44,44,44,39,44,44,44/
      INTEGER D,I,J,U,R
      INTEGER FIRST,TAIL
      IF (R.EQ.0) GO TO 8
      I=1
      CALL PWRITS(FIRST(R),U,I)
      IF (RECORD(I).EQ.44) I=I-1
      J=0
1     IF (J.EQ.7) GO TO 4
      IF (I.EQ.72) GO TO 2
      J=J+1
      I=I+1
      RECORD(I)=D(J)
      GO TO 1
2     CALL WRITE(U,RECORD)
      I=0
3     I=I+1
      J=J+1
      RECORD(I)=D(J)
      IF (J.LT.7) GO TO 3
      GO TO 5
4     IF (I.LT.72) GO TO 5
      I=0
      CALL WRITE(U,RECORD)
5     I=I+1
      CALL PWRITS(FIRST(TAIL(R)),U,I)
      IF (I.EQ.72) GO TO 7
      I=I+1
      DO 6 J=I,72
6     RECORD(J)=44
7     CALL WRITE(U,RECORD)
      RETURN
8     CALL PWRITE (U,R)
      RETURN
      END
```