SYMPLR

SYmbolic Multivariate Polynomial Linearization and Reduction

Debra J Richardson
Lori A Clarke
Debi L Bennett

COINS Technical Report 78-16
July 1978

This work was supported by the National Science Foundation under grant # NSFMCS 77-02101 and the U. S. Air Force Office of Scientific Research under grant # AFOSR 77-3287.

ABSTRACT

This report describes the implementation and use of SYMPLR, SYmbolic Multivariate Polynomial Linearization and Reduction, a system that reduces polynomials of multiple variables. The system was designed to be (reasonably) machine-independent and has been written in ANSI FORTRAN for the purpose of portability. SYMPLR was created in response to the need for polynomial simplification in ATTEST, Automatic TesT Enhancement SysTem [1]. The interface of SYMPLR with ATTEST is described. This interface may be generalized for the use of SYMPLR with any system requiring polynomial reduction. In addition, the use of SYMPLR as a stand-alone system is outlined. Examples of the polynomial simplification performed by SYMPLR are given. This document serves as both a users' manual and a programmers' manual for SYMPLR.

TABLE OF CONTENTS

1. Introduction

SYMPLR, SYmbolic Multivariate Polynomial Linearization and Reduction, is a system that reduces symbolic polynomials of multiple variables to a canonical form and attempts to make them linear. The inputs to SYMPLR are free-format alphanumeric polynomials in infix notation. If an error is encountered during the parsing phase of an input expression, simplification cannot be performed and an error flag is returned. On the other hand, if a valid expression is input, then reduction is performed and the resulting polynomial expression is output. A status code indicating the result of the reduction process is also returned. The resulting polynomial can be output in an alphanumeric representation in infix notation. In addition, if the resulting polynomial is linear, a vector of the coefficients for each variable can also be provided.

SYMPLR can be interfaced with any system that requires polynomial simplification, either as a subroutine or as a stand-alone system. A main program SYMPLRP is provided for the use of SYMPLR as a stand-alone system. When used in this manner, SYMPLR reads an expression from an input file; if the input expression is valid, the simplified polynomial is written to an output file along with a status message, otherwise an error message is issued. When invoked as a FORTRAN subroutine, an expression is passed as a parameter to SYMPLR; the resulting polynomial and a status code and error flag are returned as parameters.

SYMPLR was created in response to the need for polynomial

simplification in ATTEST, Automatic TesT Enhancement SysTem [1]. ATTEST generates test data for programs written in ANSI FORTRAN. The system symbolically executes a program path, generating constraints on the program's input variables. Provided the constraints are linear, a solution is found that furnishes input test data to drive execution down the path. The purpose of SYMPLR is to reduce the polynomials that form the constraints defining the domain of a path, and to attempt to make these polynomials linear.

Simplification was previously done by the ALPAC/ALTRAN system [2] developed at Bell Telephone Laboratories. ALPAC/ALTRAN is a more powerful system than is needed by ATTEST. This power created problems such as excessive time and space allocation. In addition, the ALPAC/ALTRAN system lacked portability; permission must be granted by Bell Telephone Laboratories before the system can be installed at a new location, and the machine dependencies require an adjustment period each time a move to a new operating system is made.

ATTEST is designed to be easily distributed to new locations. Portability, ease of installation, reasonable size and speed are all important considerations in the development of such a system. It became necessary to write a specialized polynomial simplifier tailored to the types of expressions generated by ATTEST. SYMPLR is the response to this need.

This report describes SYMPLR, a FORTRAN system for the reduction and (where possible) linearization of multivariate polynomials. Chapter 2 gives an overview of the input accepted, output generated, and simplification performed by SYMPLR. Chapter 3 provides the input/output

requirements for SYMPLR and instructions for the use of SYMPLR, both as a stand-alone system and as a subroutine. Chapter 4 describes the implementation; a full description of the data structures and the flow of the system are presented. Appendix A outlines each of the program modules written for SYMPLR. Appendix B displays the call graph for the SYMPLR system. Appendix C contains the definitions of the SYMPLR COMMON blocks and global variables as well as the initializations necessary for those variables. Appendix D describes the files that must be available for execution of SYMPLR. Appendix E gives examples of input and resulting output, demonstrating the simplification performed. Appendix F shows the evolution of the data structures through the reduction of an expression. Chapter 3 along with appendix D serves as a users' manual for SYMPLR; chapter 4 along with appendices A, B and C provide a programmers' manual.

2. Overview of SYMPLR

The expressions input to SYMPLR are variable-length, free-format, symbolic polynomials in an alphanumeric representation. SYMPLR performs an algorithmic simplification process on a polynomial and outputs the reduced polynomial in a canonical form. The polynomial is output in an alphanumeric representation, and if it is linear, as a vector of coefficients associated with each variable.

2.1. Input to SYMPLR

A valid input expression to SYMPLR is a polynomial in any number of variables. The variables can be either integer or real variables. Integer variables are denoted by In, while real variables are denoted by Xm, where n and m are integer subscripts used for identifying and ordering individual variables. Constants are represented by real and integer numbers. An expression may contain the operators of addition, subtraction, multiplication, division, and exponentiation, which are denoted by +, -, *, /, and **, respectively. An expression is parsed assuming an infix notation and the normal hierarchy of operations. Exponentiation has higher precedence than multiplication and division, which have precedence over addition and subtraction, while parentheses may be used to override the normal operator precedence.

2.2. The Normal Reduction Process and the Canonical Form

The determined process of simplification which SYMPLR performs transforms any valid polynomial into a canonical form. This canonical form is defined as follows:

a polynomial has a single numerator and a single denominator;

the numerator is an ordered sum of ordered terms (the orderings of the sums and terms are defined below);

the denominator is an ordered sum of ordered terms;

a term contains a coefficient, zero or more variables, and an exponent for each variable;

a coefficient is a real number;

an exponent is a whole number or an expression exponent;

an expression exponent is a polynomial (in canonical form) without a denominator and without any expression exponents;

coefficients, exponents and denominators of unity (1.0) are not shown in the output of the reduced polynomial.

The numerator and denominator of a reduced polynomial output by SYMPLR are ordered sums of ordered terms. Within each term, the integer variables are ordered by subscript numbers and precede all real variables, which are also ordered by their subscripts. The terms in the sum are ordered first by degree, while the order of terms of equal degree is determined as follows: all terms containing integer variables precede terms containing only real variables, and terms containing lower subscripted variable factors come first. For example, the polynomial

```
I1*I2 + I2*I3 + I2*X1 + 5.6*X2*X4
```

is in the appropriate order, whereas

```
I1*I2 + 5.3*I3*I4 + X1 + I3*X1
```

is not in the appropriate order.

The canonical form includes the requirement that all terms in the polynomial have a common denominator. If the input expression is

```
(3 / (I1 + I2)) + (4 / (I3 + I4)),
```

the reduced polynomial is

```
(4.0*I1 + 4.0*I2 + 3.0*I3 + 3.0*I4)
/ (I1*I3 + I1*I4 + I2*I3 + I2*I4).
```

The simplification process utilizes the following algebraic properties of operations on multivariate polynomials:

`p + 0 = p`	additive identity;
`p - p = 0`	additive inverse;
`a*v + b*v = (a+b) * v`	addition of common variables;
`b*w + a*v = a*v + b*w`	commutativity of addition;
`p * 0 = 0`	multiplicative zero;
`p * 1 = p`	multiplicative identity;
`p / p = 1`	multiplicative inverse;
`p/q * r/s = (p*r) / (q*s)`	multiplication of fractional polynomials;
`(p + q) * r = p*r + q*r`	distribution of multiplication over addition;
`p/r + q/r = (p+q) / r`	addition of polynomials with common denominators;
`p/q + r/s = (p*s + r*q) / q*s`	fractional extension to form a common denominator;
`p ** 0 = 1`	exponential zero;
`p ** 1 = p`	exponential identity;
`p ** n = p**(n-1) * p`	repeated multiplication of exponentiation;
`p ** (-n) = 1 / p**n`	exponentiation with a negative exponent;
`(v*w) ** e = v**e * w**e`	distribution of exponentiation over multiplication;
`p**e * p**f = p ** (e+f)`	exponentiation with equal bases;
`p ** e ** f = p ** (e*f)`	exponentiation of a power;

where p, q, r and s are polynomials, a and b are real numbers, v and w

are variables, n is a whole number, and e and f are expression exponents.

2.3. Special Linearization Transformation

Any polynomial, p, which is generated by ATTEST and input to SYMPLR for reduction is part of a constraint of the form

p (rop) 0,

where (rop) is a relational operator. Since ATTEST can solve the constraints it generates only in the case that the polynomials are linear, it behooves SYMPLR to do all it can to transform the expressions into linear polynomials. Hence, it was decided that SYMPLR would also be capable of performing transformations on the polynomials that are based on certain knowledge of the application of these expressions. If the polynomial p is of the form c / q, where c is a number and q is a polynomial, after the normal reduction has been completed, then the following transformation may be performed without changing the result of the relation in ATTEST (this helps, and is done, only when q is linear):

c/q (rop) 0 -> c*q / q**2 (rop) 0, q**2 >= 0 -> c*q (rop) 0.

This transformation is optional and can be requested or rejected by an input directive.

2.4. Limitations of SYMPLR

SYMPLR has certain limitations that have been imposed in this implementation. Some of these restrictions may be easily modified for another implementation, while others would require major changes in the

algorithm.

Although SYMPLR is designed to accept variable-length input expressions and to reduce polynomials in any number of variables, limits are imposed in the main program in order to dimension the necessary arrays. The length of the alphanumeric representation of an input expression or of the reduced polynomial, which is output, is restricted to 800 characters (this is broken down into ten records of 80 characters each). In addition, bounds must be set on the subscripts of the variables and the sum of these bounds cannot exceed 100; this places a limit on the number of distinct variables that may occur in an input expression. Both of these limits can be easily modified for another implementation. A violation of these restrictions is detected during input.

Within the simplification process, the operations of addition, subtraction, and multiplication have no restrictions. The division process is capable of simplifying fractional polynomials by performing polynomial division and removing common factors. The polynomial division algorithm provides the quotient polynomial when the denominator divides the numerator evenly; otherwise (when the denominator does not evenly divide the numerator), it reduces the fractional polynomial.

Exponentiation has the greatest limitations in the SYMPLR system. The exponentiation algorithm is capable of handling all whole-number exponents, although certain limitations have been imposed to curb the effort expended in converting a polynomial to canonical form. In this implementation, the expansion process of exponentiation is not performed for exponents that exceed the value 7.0. The expansion process is also

excluded if the basis (the polynomial to be exponentiated) contains more than five terms (note that in the case of a fractional polynomial, there may be up to five terms in the numerator and up to five terms in the denominator). These two limitations on the complexity allowed in the exponentiation process are only restrictions for this implementation and may be changed easily.

The exponentiation algorithm has only limited ability to handle non-whole-valued and expression exponents, due in part to the impossibility of raising a symbolic polynomial to a non-whole exponent. An input expression may contain variables or single terms (no sums of terms) raised to a non-whole number or to an expression. These exponents may be only one level deep - that is, the expression exponents may not contain non-whole-valued or expression exponents. In addition, expression exponents may not be fractional polynomials - that is, they may consist of a numerator only. The presence of invalid expression exponentiation is not detected during the parse, but if at any time an operation involving invalid expression exponentiation is attempted, the process will be discontinued. The actual exponentiation process cannot be performed when the exponent is a non-whole number or an expression, but these exponents are carried along throughout the simplification process since it may be possible to cancel exponents and remove the complication in the expression. It was decided that although exponentiation to non-whole exponents is quite limited, it would be beneficial to delay the discontinuation of the reduction process as long as possible. Exponents exceeding the value 7.0 are also treated in this manner. The limitations in the exponentiation process regarding non-whole-valued and expression exponents are inherent in the SYMPLR

algorithm and would require major modification to lessen the restrictions.

2.5. Output from SYMPLR

The reduced form of an expression that has undergone processing by SYMPLR will be output only if both the parsing and simplificaion phases are completed without error. If the parsing phase detects an invalid input expression, an error is output. If an input expression succeeds in passing the parsing phase, but the simplification process encounters something in the expression that cannot be resolved due to one of the limitations stated above, the status will indicate the reason for discontinuation.

The status associated with each successfully reduced polynomial indicates the result of the reduction. If the expression was successfully reduced to a linear polynomial without the use of the special linearization transformation mentioned above, the status will indicate that the polynomial is linear.. If the normal reduction process does not yield a linear expression, but the linearization transformation is applied and makes the polynomial linear, the status will indicate that the polynomial is linearized. If any non-linearities remain in the reduced polynomial, but the parsing of the input expression and the simplification process were not erroneous, the status will indicate that the polynomial is non-linear. In each of these cases, the reduced polynomial can be output in its alphanumeric representation, but only if the status indicates that the polynomial is linear or linearized can the vector of coefficients be output.

When used as a stand-alone system, SYMPLR reports any errors associated with the input or the status of a polynomial by printing an appropriate message. When used as a subroutine, SYMPLR merely returns an error flag and a status code. The error and status messages and values are defined in section 3.2.1.

3. User Specifications

SYMPLR may be used as a stand-alone system or invoked as a subroutine from a FORTRAN program. The SYMPLR subroutine must have certain files available, while the SYMPLR mainline accesses the same files plus additional files. Several of these files are mentioned below; a description of each file, its use and its contents are provided in appendix D. Each required file must be associated with a logical unit number for FORTRAN I/O and must be set up by the appropriate operating system commands.

Regardless of whether the SYMPLR subroutine or the SYMPLR mainline is used, SYMPLR requires specific input directives, accepts a general form for the input expression, and produces various forms of output, depending on the input directives and input expression. These input and output requirements are described below, followed by specifications for using SYMPLR as a subroutine and as a main program.

3.1. Input Requirements

3.1.1. Input Directives

SYMPLR requires several input directives, which define the operations to be performed, the output to be produced, and the data structures that are necessary for the reduction process. These directives are described below:

LINEARIZATION directive

The LINEARIZATION directive specifies whether the simplification process is to include the special linearization transformation, which uses the knowledge of the particular applications of ATTEST (this transformation is described in section 2.3 and section 4.3). This directive may have the values

TRUE -- perform the special linearization transformation;
FALSE -- do not perform the special linearization transformation.

OUTPUT directive

The OUTPUT directive specifies the type of output that is desired. Output of the reduced polynomials has two possible forms - an alphanumeric representation of the simplified polynomial and a vector of coefficients. A vector of coefficients can only be output when the input expression is reduced to a linear polynomial. The OUTPUT directive may have the values

0 -- output vector of coefficients only;
1 -- output alphanumeric representation only;
2 -- output both representations.

SUBSCRIPT directive

The SUBSCRIPT directive indicates the maximum number of integer and real variables that the input polynomials can contain - that is, the bound on the subscripts for each type of variable. The sum of these is the maximum number of variables. The SUBSCRIPT directive consists of two non-negative integer values

<IBND> -- bound on the integer variables' subscripts;
<XBND> -- bound on the real variables' subscripts.

3.1.2. Input Expressions

An INPUT EXPRESSION is an operator infix polynomial with the normal precedence ordering of the operators (**: *,/: +,-), parentheses overriding the ordering. An INPUT EXPRESSION is an alphanumeric string that may contain the following items:

In -- an integer variable subscripted by n, where n is an integer and 1 <= n <= IBND;
Xm -- a real variable subscripted by m, where m is an integer and 1 <= m <= XBND;
( ) + - * / ** -- the legal operator symbols;
real and integer numbers -- constants, where real numbers cannot have more than five digits before the decimal point and five digits after the decimal point, and integer numbers cannot have more than ten digits.

No blanks can be embedded within individual items, but blanks may occur anywhere else within an expression.

3.2. Output Requirements

3.2.1. Output of Error and Status

SYMPLR returns information on any errors encountered in the parsing of an input expression or errors caused by invalid input directives as well as the status of the reduction process for an input expression. The information provided is explained below:

ERROR

The ERROR value indicates the validity of the input and the outcome of the parsing phase. The meanings of the various ERRORs are defined below, where the numerical value returned from the subroutine is given followed by the message (in upper case) issued by the main program (when

necessary, any further explanation is given in lower case):

0 -- no error occured in the input;
1 -- LENGTH OF REDUCED POLYNOMIAL EXCEEDS DIMENSION OF EXPRESSION ARRAY (if the polynomial has been simplified and if it is linear, then the vector of coefficients is still valid);
2 -- LENGTH OF INPUT EXPRESSION EXCEEDS DIMENSION OF EXPRESSION ARRAY;
3 -- INVALID SUBSCRIPT BOUNDS (subscript bounds have not been initialized or they are too large for the dimension set for the vector of coefficients);
4 -- INTEGER VARIABLE SUBSCRIPTED BEYOND BOUND ENCOUNTERED;
5 -- REAL VARIABLE SUBSCRIPTED BEYOND BOUND ENCOUNTERED;
6 -- ILLEGAL CHARACTER ENCOUNTERED;
7 -- UNBALANCED PARENTHESES ENCOUNTERED;
8 -- INPUT EXPRESSION NOT IN IN-FIX NOTATION.

STATUS

The STATUS indicates the result of the polynomial reduction process. The meanings of the various STATUSes are defined below, where the numerical value returned from the subroutine is given followed by the message (in upper case) issued by the main program (when necessary, further explanation is given in lower case):

0 -- REDUCED POLYNOMIAL IS LINEAR;
1 -- REDUCED POLYNOMIAL IS LINEARIZED (by application of the special linearization transformation);
2 -- REDUCED POLYNOMIAL IS NON-LINEAR;
3 -- INVALID EXPRESSION EXPONENTIATION ENCOUNTERED (a multi-level expression or fractional exponent occured in the polynomial or more than one term was raised to an expression exponent);
4 -- OVERLY COMPLEX EXPONENTIATION ENCOUNTERED (exponentiation with too many terms in the basis or too large an exponent occured);
5 -- REDUCED POLYNOMIAL IS UNDEFINED (division by zero occured);
6 -- no reduction performed due to an error in the input.

3.2.2. Reduced Polynomial

A variety of forms of the REDUCED POLYNOMIAL corresponding to an

input expression may be output by SYMPLR. The form is dependent on the type of output requested and the outcome of the parsing and reduction phases. These forms of the REDUCED POLYNOMIAL, along with the cases in which each is output, are described below:

INPUT EXPRESSION

The INPUT EXPRESSION will be output as it was read in if either the parsing phase detects an invalid expression, one of the input directives or parameters is invalid, or the reduction process was discontinued. In any of these cases, the reduction process is incomplete and the reduced polynomial cannot be acheived.

ALPHANUMERIC REPRESENTATION

The ALPHANUMERIC REPRESENTATION will be output if the reduction process runs to completion and the OUTPUT directive is one or two. The ALPHANUMERIC REPRESENTATION has much the same format as the INPUT EXPRESSION; the polynomial is in the canonical form described in section 2.2.

VECTOR OF COEFFICIENTS

The VECTOR OF COEFFICIENTS will be output if the reduced polynomial is linear or linearized, and the OUTPUT directive is zero or two. A linear polynomial is a sum of terms in the form

C1 * I1 + ... + Cn * In + Cn+1 * X1 + ... + Cn+m * Xm + Cn+m+1,

where each Ci is a real-valued coefficient (possibly 0.0 or 1.0), and n = IBND, the maximum subscript for integer variables, m = XBND, the maximum subscript for real variables. The VECTOR OF COEFFICIENTS contains the coefficients Ci in the same order as the linear polynomial form listed above, and hence has the form

(C1, ... Cn, Cn+1, ... Cn+m, Cn+m+1).

For example, if IBND is set to 2 and XBND is set to 1, and the reduced polynomial is 5*I2 + X1 - 4, then the VECTOR OF COEFFICIENTS is (0, 5, 1, -4).

3.3. SYMPLR as a Subroutine

When SYMPLR is used as a FORTRAN subroutine, the calling program must declare the global variables necessary for SYMPLR. Appendix C provides the definitions of the SYMPLR COMMON blocks and describes the global variables that are contained in the COMMON blocks. Many of these global variables must be initialized, which may be done in a BLOCK DATA subprogram associated with the calling program. Appendix C also specifies the form of the BLOCK DATA subprogram as it is required by SYMPLR. Alternatively, these variables could be given the necessary values by assignment statements in the calling program. In addition, the calling program must invoke two subprograms to prepare the database used by SYMPLR, after which SYMPLR may be invoked.

3.3.1. Preparation of the SYMPLR Database

The SYMPLR database is manipulated by the subprograms in a database management system that was originally developed at the University of Colorado. A description of the design, implementation and use of this system can be found in [3] or [4]. The system has undergone various modifications at both the University of Colorado and the University of Massachusetts; at the present time, these changes are documented only in memoes between the two groups. The source code for this system is

included with the source code for SYMPLR. SYMPLR's use of this database management system is described in chapter 4. Before any manipulation of the database is attempted, two database subprograms must be called; these must thus be called before SYMPLR may be invoked. The first of these initializes the database, while the second opens the database file. The function of the initialization routine, INITRN, is to read in the values that must be associated with the database nodes and fields and to build the tables necessary for creating and accessing the database as specified. The database initialization information is specified on a binary file, COMDAT, which is provided with the source code for SYMPLR. The database initialization routine is invoked by

```
CALL INITRN(COMDAT)
```

where COMDAT is an integer variable that specifies the unit number of the file containing the database initialization information. INITRN initializes the values of the tables and variables used by the database accessing routines, and must be called before any manipulation of the database is attempted.

Following initialization of the database, a random access file must be opened on which the database will reside. The subroutine OPNDBF must be called in order to open a database file; OPNDBF also associates the database table with this newly-opened file and updates a global array that keeps track of allocation on the open files. This subprogram is invoked by

```
CALL OPNDBF(NEWDB, DATBAS, MASTR, MILEN, DBTAB, DBDIM)
```

Each of the parameters is explained below; its input/output classification is specified in parentheses beside the parameter name.

NEWDB (input)

NEWDB specifies whether the database is a new or old file. NEWDB is an integer variable that must have the value 0 or 1. The value 0 indicates a new file, which must be opened for the first time, while the value 1 indicates an old file, which must be reopened. A new database file should be opened for SYMPLR - thus NEWDB = 0 - since SYMPLR always begins with a fresh database.

DATBAS (input)

DATBAS specifies the unit number of the file on which the database pages reside. DATBAS is an integer variable. The database file, DATBAS, is a temporary random access file.

MASTR (output)

MASTR is user-allocated storage for the master index array. MASTR is a one-dimensional integer array. MASTR is modified only by operating system random access I/O routines and is thus machine dependent; it is necessary for use on CDC machines.

MIDIM (input)

MIDIM specifies the dimension of the master index array, MASTR. MIDIM is an integer variable and must be positive; MIDIM = PAGES + (1 + DBDIM/IPGSIZ). DBDIM is an integer variable that specifies the dimension of the database table (see the explanation of the DBDIM parameter below). IPGSIZ is a global variable that specifies the size of the database pages; IPGSIZ is contained in the COMMON block DBPAGE (see the explanation of the COMMON blocks in appendix C) and is initialized by INITRN. PAGES is an integer variable that specifies the number of database pages that are allowed, and must be at least two(2).

If only two pages are allowed, no paging of the database will be performed, since there are always two pages in the database area, which remains in core. If PAGES is greater than two, paging of the database is allowed. Some indication of the sufficiency of the number of pages allowed can be obtained from the database utilization statistics, which can be output when the database file is rewound after an expression is simplified (see the explanation of the DBUTIL parameter below). When an exceptionally complex expression is input and there are not enough pages allocated, a run-time error will occur. To alleviate this problem, PAGES should be increased.

DBTAB (output)

DBTAB is the database table array. DBTAB is a one-dimensional integer array. DBTAB contains the information concerning allocation of the database file, a pointer to the database description, which defines the nodes and fields of the database, and a pointer to the prefix page for this database, which contains the page table.

DBDIM (input)

DBDIM specifies the dimension of the database table array, DBTAB. DBDIM is an integer variable and must be positive; DBDIM = 5 + NUMDB. NUMDB specifies the number of databases. When SYMPLR is the only user of the database system (as is most often the case), there is only one database - thus DBDIM = 6.

Note that PAGES, DBTAB and DBDIM must be passed as parameters to SYMPLR (see section 3.3.2).

The database file should also be closed following the processing

done by SYMPLR. This is done by calling the database file close routine, CLSDBF, which may be invoked by

```
CALL CLSDBF(DBTAB,DBDIM,DBUTIL)
```

where DBTAB and DBDIM are as specified above, and DBUTIL is an integer variable specifying the unit number of a file on which a message indicating the utilization statistics of this database run will be output. The SYMPLR subroutine, however, rewinds the database file after processing an expression (see the explanation of the DBUTIL parameter below); hence, at the point of closing the database, the utilization statistics will indicate that no pages were allocated for this file. If DBUTIL = 0, the utilization statistics are suppressed.

3.3.2. Invocation of the SYMPLR Subroutine

SYMPLR accepts the alphanumeric representation of an expression in a one-dimensional array EXPRS, in addition to the necessary input directives and other inputs as arguments. If invalid input (expression, directive, or other argument) occurs, the EXPRS array is returned unchanged (that is, the INPUT EXPRESSION is returned) along with an ERROR flag indicating what was invalid. If the input is valid, simplification is attempted. If simplification is discontinued due to a limitation of SYMPLR, the EXPRS array is also returned unchanged along with a STATUS code indicating the reason for discontinuation. Otherwise, the REDUCED POLYNOMIAL is returned in its ALPHANUMERIC REPRESENTATION in the EXPRS array along with a VECTOR OF COEFFICIENTS and a STATUS code. SYMPLR may be invoked from a FORTRAN program by

```
      CALL SYMPLR (LINDR, OUTDR, IBND, XBND, EXPRS, EXLEN, EXDIM,
     +      COEFS, CFDIM, STATUS, ERROR, PAGES, DBTAB, DBDIM, DBUTIL)
```

The SYMPLR input parameters consist of an expression for simplification, all input directives and various other parameters necessary for the implementation. The SYMPLR output parameters consist of the resulting expression, which may or may not be reduced, an error flag and a status code. Each of the parameters is explained below; its input/output classification is specified in parentheses beside the parameter name.

LINDR (input)

LINDR specifies the value of the LINEARIZATION directive. LINDR is a logical variable.

OUTDR (input)

OUTDR specifies the value of the OUTPUT directive. OUTDR is an integer variable and may have the value 0, 1 or 2. If an invalid value is input, the default value of 2 is assumed.

IBND (input)

IBND specifies the upper bound on subscripts of integer variables. IBND is an integer variable and must be non-negative. No default value is assumed.

XBND (input)

XBND specifies the upper bound on subscripts of real variables. XBND is an integer variable and must be non-negative. No default value is assumed.

EXPRS (input/output)

EXPRS is an array containing the expression. EXPRS is a one-dimensional integer array, whose dimension is specified by EXDIM (see the

explanation of the EXDIM parameter below). It is interpretted as containing one character per array element, left justified. On input, EXPRS provides the INPUT EXPRESSION, whose format is as stated in section 3.1.2. On output, EXPRS returns the resulting expression. If the the simplification is incomplete, or the OUTPUT directive has the value zero (VECTOR OF COEFFICIENTS ONLY requested), then the EXPRS array returns the INPUT EXPRESSION. Otherwise, if the polynomial was reduced successfully and the OUTPUT directive is one or two, the EXPRS array returns the ALPHANUMERIC REPRESENTATION of the REDUCED POLYNOMIAL.

EXLEN (input/output)

EXLEN specifies the length of the expression - that is, the number of valid characters in the EXPRS array. EXLEN is an integer variable and must be positive and less than or equal to EXDIM (see the explanation of the EXDIM parameter below). On input, EXLEN provides the length of the INPUT EXPRESSION, while on output, EXLEN returns the length of the REDUCED POLYNOMIAL. If EXLEN is greater than EXDIM on input or the length of the resulting expression exceeds EXDIM on output, an error is returned.

EXDIM (input)

EXDIM specifies the dimension of the EXPRS array, which provides the INPUT EXPRESSION and returns the REDUCED POLYNOMIAL. EXDIM is an integer variable and must be positive. The dimension of the EXPRS array is necessary, in addition to the length of the expression, because the REDUCED POLYNOMIAL may be longer than the INPUT EXPRESSION and hence the actual space available in EXPRS must be known.

COEFS (output)

COEFS is an array to return the VECTOR OF COEFFICIENTS, which contains one coefficient for each variable and a constant term. COEFS is a one-dimensional real array, which must be dimensioned to contain at least CFDIM elements (see the explanation of the the CFDIM parameter below). If the reduced polynomial is linear and the OUTPUT directive has the value zero or two, then the COEFS array returns the VECTOR OF COEFFICIENTS for the linear REDUCED POLYNOMIAL. Otherwise (the reduced polynomial is not linear or only the ALPHANUMERIC REPRESENTATION is requested), the COEFS array returns unchanged.

CFDIM (input)

CFDIM specifies the dimension of the COEFS array, which returns the VECTOR OF COEFFICIENTS. CFDIM is an integer variable and must be positive. The COEFS array must be large enough to accomodate a coefficient for each variable and a constant term - that is, CFDIM >= IBND + XBND + 1. If this restriction is violated, an ERROR is returned.

STATUS (output)

STATUS returns a STATUS code indicating the outcome of the reduction process performed by SYMPLR. STATUS is an integer variable, whose values are defined in section 3.2.1.

ERROR (output)

ERROR returns an ERROR flag indicating the validity of the input to SYMPLR. ERROR is an integer variable, whose values are defined in section 3.2.1.

PAGES (input)

PAGES specifies the number of pages allowed in the database. PAGES is

an integer variable and must be at least two(2) (see the explanation of the PAGES variable with the MIDIM parameter above).

DBTAB (input)

DBTAB is the database table. DBTAB is a one-dimensional integer array. DBTAB is initialized when the database file is opened by the invocation of OPNDBF in the calling program (see the explanation of the DBTAB parameter above for a more thorough description).

DBDIM (input)

DBDIM specifies the dimension of the DBTAB array. DBDIM is an integer variable and must be at least six(6) (see the explanation of the DBDIM parameter above for a more thorough description).

DBUTIL (input)

DBUTIL specifies the unit number of a file on which a message indicating the utilization statistics of this run will be output. The database file, DATBAS, is rewound before returning from the SYMPLR subroutine. At this point, the utilization of the database pages is output on the file, DBUTIL, after which it is reinitialized to reflect the rewinding of the database file. If utilization statistics, which are helpful in determining the number of pages that should be allowed (the PAGES parameter specified above) are desired, they must be obtained when the file is rewound rather than when it is closed. If database utilization statistics are not needed, DBUTIL = 0 causes the statistics to be suppressed.

3.4. SYMPLR as a Stand-Alone System

As a stand-alone system, SYMPLR accepts alphanumeric input from a file, INPEXP, while output is written to another file, REDPLY. The input to the SYMPLR main program includes the INPUT EXPRESSIONs, values for the input directives and comments (which are ignored). The INPUT EXPRESSION is echoed, along with the results of processing by SYMPLR. If the input is invalid, an ERROR message is issued. If the input is valid, but the reduction process is discontinued, a STATUS message is printed. On the other hand, if the input expression is successfully reduced, a STATUS message is printed, which states the result of the reduction, and the REDUCED POLYNOMIAL is output as either a VECTOR OF COEFFICIENTS or in its ALPHANUMERIC REPRESENTATION or both.

3.4.1. Input to the SYMPLR Main Program

The input file, INPEXP, for SYMPLR consists of input directives and input expressions for simplification. Each record in the input file must not exceed 80 characters and may be one of the types described below:

COMMENT record

A COMMENT record is solely for the purposes of documenting the input file and has no effect on the reduction process of SYMPLR. A COMMENT record must have a "C" in column one. A COMMENT record may appear anywhere in the input file, except between two records of an input expression.

LINEARIZATION record

A LINEARIZATION record specifies the value of the LINEARIZATION directive. A LINEARIZATION record must have an "L" in column one, followed by the value of the directive, "T" (TRUE) or "F" (FALSE), which may appear anywhere on the line. This type of record may appear anywhere in the input file. In the absence of a LINEARIZATION record, the default value for the LINEARIZATION directive is FALSE. If an invalid LINEARIZATION directive is input, the value of the directive is unchanged. The value of the LINEARIZATION directive remains set for all following expressions until explicitly changed by another LINEARIZATION record.

OUTPUT record

An OUTPUT record specifies the value of the OUTPUT directive - what type of output is desired. An OUTPUT record must have an "O" in column one, followed by the value to be assigned to the directive, 0 or 1 or 2, which may appear anywhere on the line. This type of record may appear anywhere in the input file. In the absence of an OUTPUT record, the directive value defaults to 2. If an invalid OUTPUT directive is input, the value of the directive is unchanged. The OUTPUT directive remains set for all following expressions until explicitly changed by another OUTPUT record.

SUBSCRIPT record

A SUBSCRIPT record indicates the upper bounds on subscripts of real and integer variables. A SUBSCRIPT record must have an "S" in column one, followed by two positive integer values which may appear anywhere on the line (separated by at least one space). The first of these numbers

specifies <IBND>, the upper bound on subscripts of integer variables, while the second specifies <XBND>, the upper bound on subscripts of real variables. A SUBSCRIPT record must appear before any expressions are input; no default bounds are set (if no SUBSCRIPT record precedes an input expression, it will not be processed, but will be output unchanged along with an ERROR). The bounds on the subscripts remain set for all following expressions until explicitly changed by another SUBSCRIPT record.

INPUT EXPRESSION record

An INPUT EXPRESSION record may occur anywhere in the input file (provided it is preceded by a SUBSCRIPT record), and is essentially free-format. An expression may be up to ten records in length and is terminated by a dollar sign ($). No directive records may appear between records of the same expression - that is, before the end of expression is encountered.

EOF record

An EOF record terminates the input file; all records occurring after an EOF record are ignored. An EOF record must have an "E" in column one.

3.4.2. Output from the SYMPLR Main Program

The output file, REDPLY, produced by the SYMPLR mainline contains copies of the comments, echoes of the input directives, as well as sequences of records for each polynomial. Each output record consists of 80 characters. The types of output records that may be found in the output file are described below:

COMMENT record

A COMMENT record is merely a copy of a COMMENT record encountered in the input file.

LINEARIZATION record

A LINEARIZATION record ehoes the value input for the LINEARIZATION directive by specifying either LINEARIZATION TRANSFORMATION INCLUDED or LINEARIZATION TRANSFORMATION OMITTED, depending on whether the input directive is TRUE or FALSE, respectively. If an invalid LINEARIZATION record is encountered in the input file, the LINEARIZATION record produced will be INVALID LINEARIZATION DIRECTIVE, following a copy of the invalid record.

OUTPUT record

An OUTPUT record echoes the value input for the OUTPUT directive by specifying either OUTPUT VECTOR OF COEFFICIENTS ONLY, OUTPUT ALPHANUMERIC REPRESENTATION ONLY or OUTPUT BOTH REPRESENTATIONS, depending on whether the value of the directive is 0, 1 or 2, respectively. If an invalid OUTPUT record is encountered in the input file, the OUTPUT record produced will be INVALID OUTPUT DIRECTIVE, following a copy of the invalid record.

SUBSCRIPT record

A SUBSCRIPT record echoes the values assigned to the subscript bounds for integer and real variables by specifying SUBSCRIPT BOUNDS ARE <IBND> (INTEGER) AND <XBND> (REAL). If invalid bounds occur, the SUBSCRIPT record produced will be INVALID SUBSCRIPT DIRECTIVE, following a copy of the invalid record.

REDUCED POLYNOMIAL records

When an expression is input, SYMPLR produces a sequence of output records. The INPUT EXPRESSION is always output, followed by an indication of any errors or the status of the reduction process. An ERROR message is output if the parsing phase is incomplete, while a STATUS message is output if the simplification phase is embarked upon. If the simplification phase is discontinued for any reason, nothing more will be output for this input expression. Otherwise, the REDUCED POLYNOMIAL is output, as either a VECTOR OF COEFFICIENTS or an ALPHANUMERIC REPRESENTATION or both (depending on whether the reduced polynomial is linear and the type of output requested). No VECTOR OF COEFFICIENTS is output for a non-linear reduced polynomial.

EOF record

The EOF record is output when an EOF record is encountered in the input file. This terminates the output file produced by the SYMPLR main program.

4. Implementation Details

SYMPLR, along with all supporting subprograms, is written predominantly in ANSI FORTRAN, with the goal of portability in mind. Machine dependencies have been kept to a minimum, and all have been well-documented, for ease of installation under another operating system.

SYMPLR is a modular system, consisting of one main driving routine (which is available as a main program and as a subroutine), a library of subprograms that perform parsing, reduction, and output of the symbolic multivariate polynomials, and a library of subprograms that manipulate the SYMPLR database. The main program, subroutines and functions in the former (SYMPLR's) library are described in Appendix A, and the flow through these is described in section 4.2.

The SYMPLR database is manipulated by the subprograms in a database management system, which allows for the creation and modification of a collection of lists, each organized as a linked list or as a sequential list (table). A list is an ordered collection of nodes. A node consists of a collection of sublists and individual data items or fields. The subprograms enable the creation of such a database and access to the nodes and fields by symbolic name, making the manipulation of data structures much more transparent than it would normally be in FORTRAN. The system also allows the specification of a data type for a field, and perform error checking on the data stored in particular fields.

4.1. The SYMPLR Database

The database is the backbone of SYMPLR in that its structure influenced the algorithms used in the simplification process. Since the polynomials to be reduced can be arbitrarily complex, it was important that the data structure be easily initialized and manipulated. It was necessary that the tasks of finding the next subexpression for reduction, maintaining the expressions in an ordered form, determining the equality of and ratio between terms, among others, be efficient operations. The database must be workable, complete, and flexible.

Briefly, the database contains four data structures:

a binary tree that represents the structure of a polynomial;

a linked list of terms in an expression;

a sequential list (table) of bit vectors indicating the variables in a term;

a linked list of exponents of the variables in a term.

4.1.1. The Binary Tree Structure

The SYMPLR database is organized around a binary tree that represents the structure of the polynomial currently undergoing simplification. A binary tree is particularly suited to the application of expression simplification since it preserves the precedence ordering of the operators; the nesting of subexpressions and the correspondence between subexpressions is maintained. Each non-leaf node of the tree represents an operator which has not yet been applied. The operator has pointers to each of its operands, forming a subtree that represents a subexpression of the polynomial. Each leaf node of the binary tree

represents a subpolynomial that has been simplified by the application of operators.

Prior to simplification, the binary tree contains the polynomial as it was parsed. A node is set up for each operator, variable, and constant. The binary tree is then traversed by postorder traversal. Knuth [5, pages 305-345] describes methods for creation and traversal of binary trees. As the tree is traversed, each subtree is processed; the operator is applied to its operands. As an operator is applied, the simplification process is performed. When the traversal and simplification are complete, the root will represent the reduced polynomial.

The database nodes in the binary tree are called KBTREE nodes. Each KBTREE node has seven fields, described below:

KFLAGS - this field holds an integer value indicating the type of the node; the field may have the values
    0 -- expression
    1 -- operator
    2 -- constant
    3 -- blocked (exponentiation too complex)
    4 -- undefined (division by zero)
    5 -- case not handled (such as exponentiation to a multi-level expression or fractional exponent);

KNUMER - if the node represents an expression, then this field contains a pointer to the linked list of terms in the numerator; if the node represents an operator or a constant, this field has a null value;

KDENOM - if the node represents an expression with a denominator, then this field contains a pointer to the linked list of terms in the denominator; if the node represents an operator, an expression without a denominator, or a constant, this field has a null value;

KCONST - if the node represents a constant, then this field contains the real value of the constant; if the node represents an operator or an expression, this field has a

null value;

KOPVAL - if the node represents an operator, then this field contains the hollerith representation of that operator; if the node represents an expression or a constant, then this field has a null value;

KLINKL - if the node represents an operator, this field contains a pointer to the tree node's left descendant; if the node has no left descendant, this field has a null value;

KLINKR - if the node represents an operator this field contains a pointer to the tree node's right descendant; if the node has no right descendant, this field has a null value.

4.1.2. The Expression List Structure

The subexpressions of the polynomial are maintained as a linked list of terms. This structure facilitates the ordering of the terms by making insertion and deletion of terms within an expression easy. In addition, this makes the conversion to an ordered vector of coefficients efficient when the reduced polynomial is linear.

The tree nodes which represent expressions point to an expression list for the numerator and possibly one for the denominator. As a subtree of the binary tree is processed, subexpressions are built up by the operations of addition, subtraction, multiplication, division, and exponentiation. These subexpressions are stored in their reduced form in an expression list. Each term of an expression is an individual expression node and each of these term nodes are linked together as a circularly-linked list. There is an implicit "+" between each term of an expression, and the terms are ordered (as defined in section 2.2). A pointer to an expression list points to the last node (term) in the list.

The database nodes in an expression list are called KEXPRS nodes. Each KEXPRS node has four fields, described below:

KBITVC - if the node is not a constant term, then this field contains the pointer into the bit vector table (KBITTB) of this term's bit vector; if the node is a constant term, this field has a null value;

KCOEFF - this field contains the real-valued coefficient for this term;

KPNTEX - if any variable in this term has an exponent (other than 1.0), then this field contains a pointer to the linked list of exponents; if there are no exponents for this term, this field has a null value;

KNXTRM - this field holds the pointer to the next term (node) in the expression (list); if this is the last term, it points to the first term.

4.1.3. The Bit Vector Table

Each term in the expression may have multiple variables. In order to specify the relevant variables without an extreme waste of space in storing the subscripts, and to allow for efficient comparisons, a bit vector is allocated for each non-constant term. Each variable is associated with a bit in the vector, and if a given variable occurs in a term, the associated bit is set in the term's bit vector. There is an implicit "*" between each variable specified in the bit vector. The location of a variable's associated bit is determined by its subscript and whether or not it is integer or real. The size of the bit vector must be the sum of the two subscript bounds (IBND+XBND), which were input in the SUBSCRIPT directive (described in section 3.1). The first IBND bits refer to integer variables, while the last XBND bits refer to the real variables, and the subscripts determine the position of a variable in each class.

The bit vector table is a sequential teble in the database that is handled independently by specialized bit string management routines. The pointers to bit strings are flagged pointers that are offsets relative to the base of the bit vector table. Since essentially any number of variables may appear in an input polynomial (limited only by the SUBSCRIPT directive and the maximum set for the implementation), the bit vectors may be several words in length. The use of the database system relieves the dependence on machine word size at this level of implementation; machine dependencies are isolated in a small group of database routines.

4.1.4. The Exponent List Structure

Each term in the polynomial may have several variables that have exponents. In order to specify which variables in the expression have exponents and the values of those exponents, a circularly-linked list of the exponents for a term is allocated. Each exponent is associated with a variable in the term. The first exponent (node) corresponds to the first variable (first bit set) in the term, the second exponent node corresponds to the second variable in the term, and so forth. There are only as many exponent nodes allocated as are necessary for this sequence - that is, the last variable in the term that has an exponent is allocated a corresponding exponent node containing the value of the exponent, as are all preceding variables in the term, but those variables that follow do not receive exponent nodes. If a term has n variables, it may have 0, 1, . . . , or n exponent nodes in the list associated with this term. If an exponent is an expression, the expression list is pointed to by the exponent node (see section 2.4 for

a thorough discussion of the restrictions on expression exponents). This structure allows many of SYMPLR's subprograms for manipulating polynomials to be applied to expression exponents as well.

The database nodes in an exponent list are called KEXPON nodes. Each KEXPON node has five fields, described below:

KEXFLG - this field is a flag which specifies the type of the exponent; the field may have the values
0 -- constant
1 -- expression;

KEXVAL - if the exponent is a constant, then this field contains the real-valued constant; otherwise, this field has a null value;

KEXNUM - if the exponent is an expression, then this field points to a linked list of terms in the expression, which has a numerator only; otherwise, this field has a null value;

KNXEXP - this field holds the pointer to the next exponent (node) in the term (list); if this is the last exponent, it points to the first exponent.

Figure 4.1 shows the relationships between the database nodes. Appendix F gives an example of the evolution of the data structures throughout the reduction process.

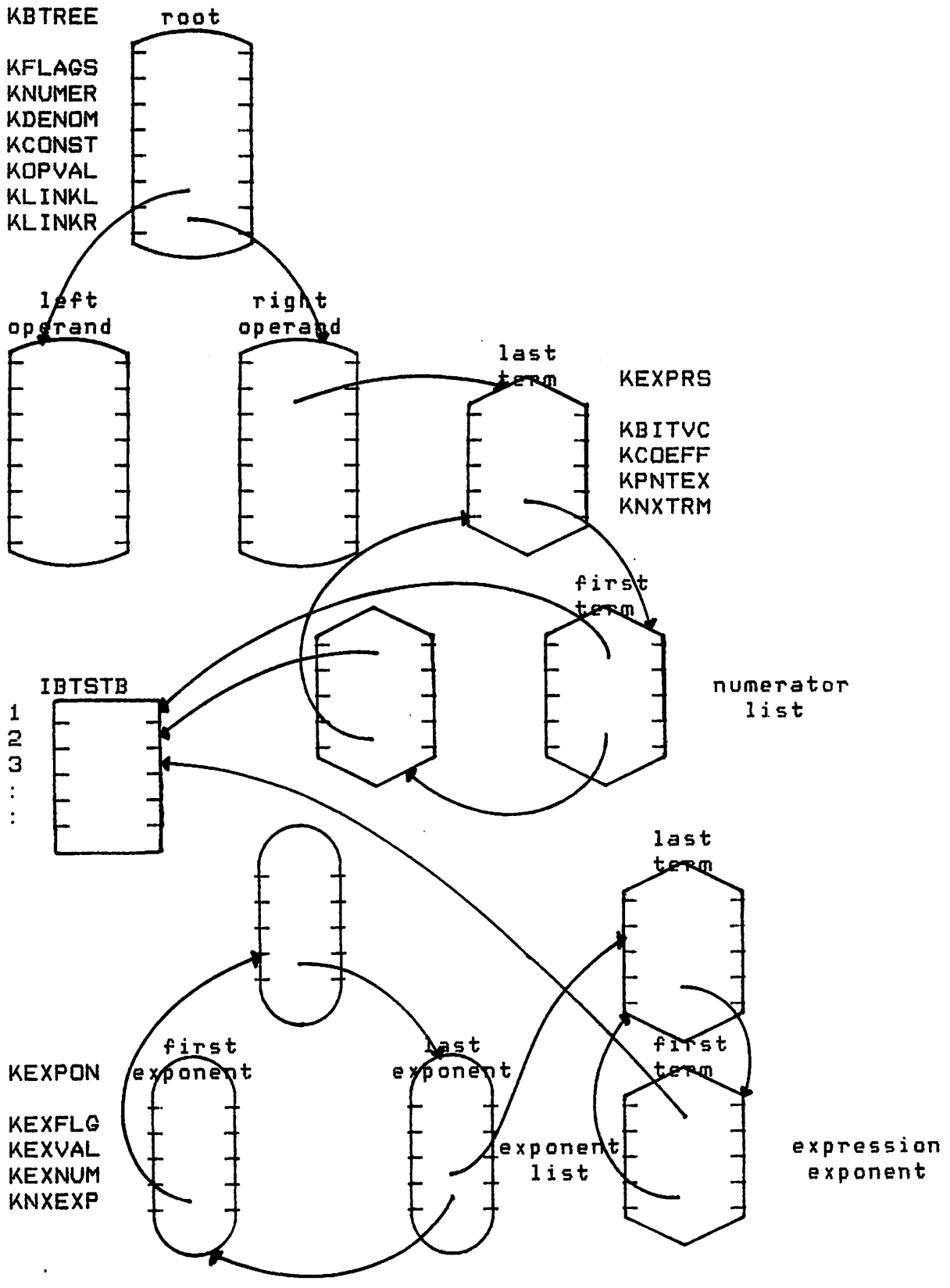


Figure 4.1

Relationships between the Database Nodes

4.2. The SYMPLR Algorithm

SYMPLR has two basic functions - parsing and simplification. The parsing phase includes lexical scan and structure recognition of an input expression, and generation of an internal form of a polynomial. The simplification phase includes transformation of the internal form into the reduced canonical form, and generation of the output for this equivalent polynomial. SYMPLR receives as input an expression to be reduced. The basic elements of an expression are variables, numbers, and operators. During lexical analysis, SYMPLR identifies these basic elements. The internal form of each basic element is created as a database node. The structure of the expression is recognized by determining the relationship between the various basic elements. The elements are then connected internally to reflect this structure. If the parsing phase is completed without error, the simplification phase is initiated. The reduction process is performed by addition, subtraction, multiplication, division, and exponentiation. After the polynomial has been transformed into its canonical form, the required output is generated.

Throughout the parsing phase, two stacks are used - an operator stack and an operand stack. Associated with each operator is a binding strength which indicates the operator's precedence value - 0 for left parenthesis, 1 for addition and subtraction, 2 for multiplication and division, 3 for exponentiation, and 5 for right parenthesis. An element of the operator stack contains the binding strength of the associated operator and a pointer to the operator's database node. An element of the operand stack contains a pointer to the operand's database node.

If in scanning, a number (coefficient, exponent, or constant term) is encountered, NUMBER is invoked. NUMBER creates a binary tree (KBTREE) node, and initializes the fields of this node. The value of the number is stored in the node. The processing of the number is completed by pushing the pointer to the tree node onto the operand stack.

If a variable (In or Xm) is scanned, VARIX is invoked. VARIX creates the proper database nodes - a binary tree (KBTREE) node, an expression list (KEXPRS) node, and a vector in the bit vector table (IBTSTB). The proper bit is set in the bit vector and all node fields are initialized. The fields KNUMER and KBITVC are filled by storing the pointers to the expression list node and the bit vector, respectively. The pointer to the tree node is pushed onto the operand stack and the variable has been processed.

When an operator is encountered in the lexical scan, PROOP is invoked. PROOP performs the structure recognition by interpreting the precedence of the operators encountered; it must ensure that operators with highest precedence are processed first. An operator is processed when it is the top element of the operator stack, and has precedence over the newly-scanned operator. An operator being processed is popped off the operator stack and POPVAR is called. POPVAR pops the operands that are on top of the operand stack, and puts the operator and its operands into a binary tree structure representing the subexpression. This subexpression is an operand for some other operator, so the pointer to this subtree is pushed onto the operand stack. If the newly-scanned operator for which PROOP is invoked is a left parenthesis (lowest

precedence) it is merely pushed onto the operator stack. If the newly-scanned operator is a right parenthesis (highest precedence), the operator stack is popped, and all operators are processed, until a left parenthesis is popped. For any other operator, a binary tree (KBTREE) node is created and its fields are initialized, including storing the operator value. The operator stack is then popped, and all operators processed, until an operator with a lesser binding strength occurs on the top of the stack. Then the pointer to the tree node for the current operator is pushed onto the operator stack. Lexical scanning of the expression is then resumed.

If at any time an error occurs during the parsing phase, such as a variable subscript out of bounds, an illegal character or unbalanced parentheses, SYMPLR outputs the appropriate ERROR flag, and the simplification phase cannot be performed.

On the other hand, if the parse has been successful when the end of the expression is encountered, ENDEX is called to finish the parsing and initiatate the simplification phase. The parsing phase is completed by processing all operators remaining on the operator stack. At this point, the entire polynomial is represented as a binary tree and TRAVEL is called to traverse the tree and drive the simplification.

TRAVEL traverses the tree representing the input polynomial in postorder traversal, which is defined recursively. If the binary tree is empty, it is traversed by doing nothing and otherwise the traversal proceeds in three steps: traverse the left subtree, traverse the right subtree, and process the root. As explained earlier, a subtree represents a subexpression where the root is the operator and its

descendants are the operands. When the root of a subtree is processed, both of its descendants must be leaf nodes - that is, both the left and the right operands must have been previously processed. Processing of a subtree involves applying the operator to its operands. TRAVEL invokes the subprogram corresponding to the operator. If the operator is "+" or "-", ADDSUB is called, if it is "*", MULT is called, if it is "/", DIVIDE is called, and if it is "**", EXPON is called. During each of these operations, any possible reduction of the current subexpression is performed. The simplified subexpression then replaces the root of the subtree (the former operator node) in the tree and TRAVEL continues the traversal.

In performing addition or subtraction, the ADDSUB routine (along with various subprograms which it invokes) performs a merge sort on the terms of the left and right operands. For addition, the terms of the augend (right operand) are shuffled into the addend (left operand) and like terms are combined, constantly maintaining the canonical ordering of the accumulating sum. Subtraction is performed by first negating the minuend (right operand) and merging it into the subtrahend (left operand) to obtain the difference.

In performing multiplication, the MULT routine (along with various subprograms that it invokes) does a term-by-term product of the left and right operands. Each term of the multiplicand (left operand) is multiplied by each term of the multiplicator (right operand) to form a new term. Each new term is added to the accumulating product using the addition algorithm, which maintains the correct order in the product.

In performing division, the DIVIDE routine (along with various

subprograms which it invokes) attempts to simplify the polynomial fraction formed by the left over the right operand in an attempt to obtain a quotient. The dividend (left operand) and the divisor (right operand) are compared for equality or difference by a constant ratio. Polynomial division is attempted; this succeeds only when the divisor evenly divides the dividend - that is, the result is a quotient with no remainder. Removal of any variables common to all terms of the dividend and divisor is also done.

In performing exponentiation, the EXPON routine (along with various subprograms that it invokes) raises the left to the right operand by repeated multiplication of the basis (left operand) to produce the power. This exponentiation process can be done only when the exponent (right operand) is a whole number, in which case this dictates the repetition factor of the multiplication. If the exponent is a non-whole number or an expression exponent and the basis (the symbolic polynomial to be exponentiated) is a single term, the exponent is stored in the database for this term in the hopes that some cancellation will occur later in the reduction process (see section 2.4 for a thorough discussion of this limitation).

If an error is encountered in the application of an operator - such as division by zero or invalid exponentiation - the STATUS code is assigned the appropriate value and traversal of the tree is discontinued. In this case, the input polynomial cannot be reduced further.

On the other hand, if the entire tree is traversed and each operation is successful, a polynomial that is equivalent to the input

polynomial has been obtained. This polynomial is found at the root of the tree and is in the reduced canonical form. If the reduced polynomial contains a denominator, one final reduction attempt is performed by invoking TRYLNR. TRYLNR has two functions. The first, which is applicable to all fractional polynomials, is the polynomial division of the denominator by the numerator. The second is the special linearization transformation (explained in section 2.3) and is performed only when this option has been selected (refer to the discussion of the LINEARIZATION directive in section 3.1).

With the reduction and linearization process finished, SYMPLR invokes the subprograms to generate the output. OUTREE converts the internal form of the polynomial to an ALPHANUMERIC REPRESENTATION. If the resulting polynomial is linear, OUTVEC forms a VECTOR OF COEFFICIENTS from the internal representation. The second phase of the SYMPLR algorithm is now complete.

Appendix A. SYMPLR Program Modules

PROGRAM SYMPLRP

SYMBOLIC MULTIVARIATE POLYNOMIAL LINEARIZATION AND REDUCTION MAIN PROGRAM

OVERVIEW:

SYMPLRP READS AN EXPRESSION FROM THE FILE, INPEXP, THE EXPRESSION IS IN INFIX NOTATION. SYMPLR FORMS A BINARY TREE OF THE POLYNOMIAL IN POLISH SUFFIX NOTATION AND THEN ATTEMPTS TO SIMPLIFY THE EXPRESSION. SYMPLRP WRITES THE RESULT ONTO THE FILE, REDPLY, ALONG WITH SOME ADDITIONAL INFORMATION.

METHOD OF USE:

FORTRAN MAIN ROUTINE

FILE PARAMETERS:

INPEXP - INPUT FILE FOR INPUT EXPRESSIONS (INPUT)
REDPLY - OUTPUT FILE FOR REDUCED POLYNOMIALS (OUTPUT)
COMDAT - FILE CONTAINING DATABASE DESCRIPTION
TEMPI - TEMPORARY FILE FOR INPUT EXPRESSION
DEBUGS - OUTPUT FILE FOR DEBUGGING PURPOSES
DATBAS - FILE FOR THE DATABASE
DBUTIL - FILE FOR DATABASE UTILIZATION STATISTICS

ERROR CONDITIONS:

IF AN ERROR IN AN INPUT EXPRESSION OR AN INPUT DIRECTIVE OCCURS, A MESSAGE WILL INDICATE THE ERROR

SPECIAL CONDITIONS:

AN INPUT EXPRESSION MAY NOT BE SIMPLIFIABLE, IN WHICH CASE A MESSAGE WILL INDICATE THE REASON FOR THE DISCONTINUATION OF THE REDUCTION PROCESS.

```
SUBROUTINE SYMPLR(LINDR, OUTDR, IBND, XBND, EXPRS, EXLEN, EXDIM,
                  COEFS, CFDIM, STAT, ERROR, DBTAB, DBDIM, DBUTIL)
```

SYMBOLIC MULTIVARIATE POLYNOMIAL LINEARIZATION AND REDUCTION

OVERVIEW:

SYMPLR INPUTS THE ALPHANUMERIC REPRESENTATION OF A POLYNOMIAL, WHICH IS IN IN-FIX NOTATION, IN THE ARRAY EXPRS. SYMPLR FORMS A BINARY TREE OF THE POLYNOMIAL AND THEN ATTEMPTS TO SIMPLIFY THE EXPRESSION. SYMPLR OUTPUTS THE ALPHANUMERIC REPRESENTATION OF THE REDUCED POLYNOMIAL IN THE ARRAY EXPRS. IF THE REDUCED POLYNOMIAL IS LINEAR, THE COEFFICIENTS OF THE VARIABLES ARE OUTPUT IN THE ARRAY COEFS. THE STATUS OF THE REDUCED POLYNOMIAL AND AN ERROR FLAG ARE RETURNED AS WELL.

METHOD OF USE:

```
INTEGER OUTDR, IBND, XBND
INTEGER EXDIM, EXLEN
INTEGER EXPRS(EXDIM)
INTEGER CFDIM
INTEGER STAT, ERROR
INTEGER DBDIM, DBTAB(DBDIM), DBUTIL
LOGICAL LINDR
REAL COEFS(CFDIM)
CALL SYMPLR(LINDR, OUTDR, IBND, XBND, EXPRS, EXLEN, EXDIM, COEFS,
            CFDIM, STAT, ERROR, DBTAB, DBDIM, DBUTIL)
```

PARAMETERS:

LINDR (INPUT) - LOGICAL FLAG INDICATING WHETHER THE SIMPLIFICATION IS BEING DONE FOR ATTEST OR SIMILAR SYSTEM. IN PARTICULAR, THIS FLAG SAYS WHETHER OR NOT TO 'LINEARIZE'

OUTDR (INPUT) - TYPE OF OUTPUT DESIRED (0=COEFFICIENTS, 1=ALPHANUMERIC, 2=BOTH)

IBND (INPUT) - INTEGER VARIABLE SUBSCRIPT BOUND

XBND (INPUT) - REAL VARIABLE SUBSCRIPT BOUND

EXPRS (INPUT/OUTPUT) - CHARACTER ARRAY FOR ALPHANUMERIC REPRESENTATION OF THE POLYNOMIAL

EXLEN (INPUT/OUTPUT) - LENGTH OF THE EXPRESSION IN EXPRS

EXDIM (INPUT) - DIMENSION OF EXPRS

COEFS (OUTPUT) - COEFFICIENTS OF A LINEAR POLYNOMIAL

CFDIM (INPUT) - DIMENSION OF COEFS

STAT (OUTPUT) - STATUS OF REDUCED POLYNOMIAL (SEE SPECIAL CONDITIONS)

ERROR (OUTPUT) - ERROR FLAG (SEE ERROR CONDITIONS)

DBTAB (INPUT) - DATABASE TABLE ARRAY (CONTAINS INFORMATION ON THE NODES AND FIELDS OF THE DATABASE)

DBDIM (INPUT) - DIMENSION OF DBTAB

DBUTIL (INPUT) - UNIT NUMBER OF FILE FOR DATABASE UTILIZATION STATISTICS

ERROR CONDITIONS:

ALL ERRORS OCCUR DUE TO INVALID INPUT; THE TYPE OF ERROR IS INDICATED BY THE VALUE OF THE ERROR FLAG AS FOLLOWS: ERROR =

NOERR (0) - NO ERROR
OVFLW (1) - EXPRS ARRAY OVERFLOWED, TOO SMALL FOR OUTPUT (POLYNOMIAL MAY HAVE BEEN SIMPLIFIABLE, CHECK STATUS, IF STAT=1 OR 2 COEFFICIENTS IN COEFS ARE VALID)
LGTRD (2) - EXLEN (LENGTH OF POLYNOMIAL) > EXDIM (DIMENSION OF EXPRS
INVSB (3) - INVALID SUBSCRIPT BOUNDS, IBND AND XBND
INTBB (4) - INTEGER VARIABLE SUBSCRIPTED BEYOND BOUND, IBND
RELBB (5) - REAL VARIABLE SUBSCRIPTED BEYOND BOUND, XBND
ILCHR (6) - ILLEGAL CHARACTER ENCOUNTERED IN EXPRESSION
UNPAR (7) - UNBALANCED PARENTHESES IN POLYNOMIAL
NINFX (8) - EXPRESSION IS NOT A POLYNOMIAL IN IN-FIX NOTATION

SPECIAL CONDITIONS:

ALTHOUGH THERE MAY BE NO ERROR INDICATED BY ERROR, THE INPUT POLYNOMIAL MAY NOT BE SIMPLIFIABLE, IN WHICH CASE THE STATUS OF THE REDUCED POLYNOMIAL WILL BE INDICATED BY THE VALUE OF STAT AS FOLLOWS: STAT =

LNEAR (0) - LINEAR POLYNOMIAL
LINZD (1) - POLYNOMIAL WAS 'LINEARIZED'
NONLN (2) - NON-LINEAR POLYNOMIAL (BUT SIMPLIFIABLE)
MFEXP (3) - MULTI-LEVEL OR FRACTIONAL EXPRESSION EXPONENT OCCURED IN THE POLYNOMIAL
OCEXP (4) - OVERLY COMPLEX EXPONENTIATION OCCURED IN THE POLYNOMIAL
UNDEF (5) - POLYNOMIAL UNDEFINED DUE TO DIVISION BY ZERO
INERR (6) - NOT SIMPLIFIABLE DUE TO AN ERROR IN THE PARAMETERS OR AN ERROR ENCOUNTERED DURING PARSING THE EXPRESSION (CHECK VALUE OF ERROR)

SUBROUTINE ADDCON(MPOLY, CONST, TFLAG, KAREA, KSIZE)

ADD A CONSTANT TO A POLYNOMIAL

OVERVIEW:

ADDCON ADDS A CONSTANT TERM (WITH VALUE OF CONST) TO A MULTIVARIATE POLYNOMIAL (POINTED TO BY MPOLY) THE RESULTANT POLYNOMIAL IS POINTED TO BY MPOLY ON OUTPUT.

METHOD OF USE:
INTEGER MPOLY, TFLAG, KSIZE
REAL CONST
DIMENSION KAREA(KSIZE)
CALL ADDCON(MPOLY, CONST, TFLAG, KAREA, KSIZE)

PARAMETERS:
MPOLY (INPUT/OUTPUT) - POINTER TO POLYNOMIAL, ADDEND / POINTER TO RESULTANT POLYNOMIAL, SUM
CONST (INPUT) - VALUE OF THE CONSTANT TERM, AUGEND
TFLAG (OUTPUT) - TYPE FLAG OF RESULT
KAREA (INPUT/OUTPUT) - DATABASE AREA
KSIZE (INPUT) - SIZE OF DATABASE

ERROR CONDITIONS:
NONE

SPECIAL CONDITIONS:
IF THE RESULT IS ZERO, MPOLY IS RETURNED EMPTY ALONG WITH A TYPE FLAG OF CONTR.

SUBROUTINE ADDEXP(OPVAL, LFTMP, RITMP, TFLAG, KAREA, KSIZE)

ADDITION OF EXPRESSION EXPONENTS

OVERVIEW:
ADDEXP PERFORMS ADDITION OR SUBTRACTION OF TWO EXPRESSION EXPONENTS. THE RIGHT EXPONENT (POINTED TO BY RITMP) IS ADDED TO OR SUBTRACTED FROM THE LEFT EXPONENT (POINTED TO BY LFTMP). THE RESULTANT EXPONENT IS POINTED TO BY LFTMP ON OUTPUT.

METHOD OF USE:
INTEGER OPVAL, LFTMP, RITMP, TFLAG, KSIZE
DIMENSION KAREA(KSIZE)
CALL ADDEXP(OPVAL, LFTMP, RITMP, TFLAG, KAREA, KSIZE)

PARAMETERS:
OPVAL(INPUT) - VALUE OF THE OPERATOR (PLUS, MINUS)
LFTMP (INPUT/OUTPUT) - POINTER TO LEFT EXPONENT, ADDEND / POINTER TO RESULTANT EXPONENT, SUM
RITMP (INPUT) - POINTER TO RIGHT EXPONENT, AUGEND
TFLAG (OUTPUT) - TYPE FLAG OF RESULT
KAREA (INPUT/OUTPUT) - DATABASE AREA

KSIZE (INPUT) - SIZE OF DATABASE AREA

ERROR CONDITIONS:
NONE

SPECIAL CONDITIONS:
IF THE RESULT IS ZERO, LFTMP IS RETURNED EMPTY ALONG WITH A TYPE FLAG OF CONTR. IF THE RESULT IS UNDEFINED, LFTMP IS RETURNED EMPTY ALONG WITH A TYPE FLAG OF UNDTR. IF THE CASE CANNOT BE HANDLED, LFTMP IS RETURNED EMPTY ALONG WITH A TYPE FLAG OF CNHTR.

SUBROUTINE ADDPLY(OPVAL, LFTMP, RITMP, TFLAG, KAREA, KSIZE)

ADDITION OF MULTIVARIATE POLYNOMIALS

OVERVIEW:
ADDPLY PERFORMS ADDITION OR SUBTRACTION OF TWO MULTIVARIATE POLYNOMIALS. THE RIGHT POLYNOMIAL (POINTED TO BY RITMP) IS ADDED TO OR SUBTRACTED FROM THE LEFT POLYNOMIAL(POINTED TO BY LFTMP). THE RESULTANT POLYNOMIAL IS POINTED TO BY LFTMP ON OUTPUT.

METHOD OF USE:
INTEGER OPVAL, LFTMP, RITMP, TFLAG, KSIZE
DIMENSION KAREA(KSIZE)
CALL ADDPLY(OPVAL, LFTMP, RITMP, TFLAG, KAREA, KSIZE)

PARAMETERS:
OPVAL (INPUT) - VALUE OF THE OPERATOR (PLUS, MINUS)
LFTMP (INPUT/OUTPUT) - POINTER TO LEFT POLYNOMIAL, ADDEND /
POINTER TO RESULTANT POLYNOMIAL, SUM
RITMP (INPUT) - POINTER TO RIGHT POLYNOMIAL, AUGEND
TFLAG (OUTPUT) - TYPE FLAG OF RESULT
KAREA (INPUT/OUTPUT) - DATABASE AREA
KSIZE (INPUT) - SIZE OF DATABASE AREA

ERROR CONDITIONS:
NONE

SPECIAL CONDITIONS:
IF THE RESULT IS ZERO, LFTMP IS RETURNED EMPTY ALONG WITH A TYPE FLAG OF CONTR. IF THE RESULT IS UNDEFINED, LFTMP IS RETURNED EMPTY ALONG WITH A TYPE FLAG OF UNDTR. IF THE CASE CANNOT BE

HANDLED, LFTMP IS RETURNED EMPTY ALONG WITH A TYPE FLAG OF CNHTR.

SUBROUTINE ADDSUB(OPVAL, LEFT, RIGHT, KAREA, KSIZE)

ADDITION AND SUBTRACTION

OVERVIEW:
ADDSUB PERFORMS ADDITION OR SUBTRACTION OF TWO OPERANDS, DEPENDING ON THE VALUE OF OPVAL. THE RIGHT OPERAND (POINTED TO BY RIGHT) IS ADDED TO OR SUBTRACTED FROM THE LEFT OPERAND (POINTED TO BY LEFT). THE RESULT IS STORED IN THE TOP OF THE TREE (I. E., THE RESULT REPLACES THE ADDITION OR SUBTRACTION OPERATOR NODE).

METHOD OF USE:
```
INTEGER OPVAL, LEFT, RIGHT, KSIZE
DIMENSION KAREA(KSIZE)
CALL ADDSUB(OPVAL, LEFT, RIGHT, KAREA, KSIZE)
```

PARAMETERS:
OPVAL (INPUT) - VALUE OF THE OPERAND (PLUS OR MINUS)
LEFT (INPUT) - POINTER TO THE LEFT OPERAND, ADDEND
RIGHT (INPUT) - POINTER TO THE RIGHT OPERAND, AUGEND
KAREA (INPUT/OUTPUT) - DATABASE AREA
KSIZE (INPUT) - SIZE OF DATABASE AREA

ERROR CONDITIONS:
NONE

SPECIAL CONDITIONS:
NONE

LOGICAL FUNCTION BITLEX(LFTBV, RITBV, KAREA, KSIZE)

LEXICOGRAPHICAL ORDERING OF BIT VECTORS

OVERVIEW:
BITLEX RETURNS .TRUE. IF LFTBV LEXICOGRAPHICALLY PRECEDES RITBV AND RETURNS .FALSE. OTHERWISE.

METHOD OF USE:

```
INTEGER LFTBV, RITBV, KSIZE
DIMENSION KAREA(KSIZE)
LOGICAL BITLEX
IF (BITLEX(LFTBV, RITBV, KAREA, KSIZE))
```

PARAMETERS:

LFTBV (INPUT) - POINTER TO LEFT BIT VECTOR
RITBV (INPUT) - POINTER TO RIGHT BIT VECTOR
KAREA (INPUT) - DATA BASE AREA
KSIZE (INPUT) - SIZE OF THE DATA BASE

ERROR CONDITIONS:

NONE

SPECIAL CONDITIONS:

NONE

SUBROUTINE CHRVAL(RCPOS, RECORD, RCDIM, IVAL, XVAL, CHRIX)

CONVERT CHARACTERS TO VALUE

OVERVIEW:

CHRVAL CHANGES A STRING OF CHARACTERS IN RECORD BEGINNING IN THE POSITION INDICATED BY RCPOS INTO AN INTEGER VALUE, IVAL OR A REAL VALUE, XVAL, DEPENDING ON THE VALUE OF CHRIX. IF CHRIX IS TRUE, AN INTEGER VALUE IS RETURNED, OTHERWISE A REAL VALUE IS RETURNED.

METHOD OF USE:

```
INTEGER RCPOS, RCDIM
INTEGER RECORD(RCDIM)
INTEGER IVAL
LOGICAL CHRIX
REAL XVAL
CALL CHRVAL(RCPOS, RECORD, RCDIM, IVAL, XVAL, CHRIX)
```

PARAMETERS:

RCPOS (INPUT/OUTPUT) - POSITION IN RECORD OF CHARACTER STRING
RECORD (INPUT) - EXPRESSION ARRAY CONTAINING CHARACTER STRING
RCDIM (INPUT) - DIMENSION OR LENGTH OF RECORD
IVAL (OUTPUT) - INTEGER VALUE OF CONVERTED CHARACTER STRING
XVAL (OUTPUT) - REAL VALUE OF CONVERTED CHARACTER STRING
CHRIX (INPUT) - .TRUE. IMPLIES INTEGER, .FALSE. IMPLIES REAL

ERROR CONDITIONS:
NONE

SPECIAL CONDITIONS:
NONE

SUBROUTINE DGRCON(PTERM, DGREE, TFLAG, KAREA, KSIZE)

DEGREE OF A TERM WITH CONSTANT EXPONENTS ONLY

OVERVIEW:
DGRCON DETERMINES THE DEGREE OF A MULTIVARITE POLYNOMIAL TERM, PTERM. THE DEGREE IS THE SUM OF ALL THE EXPONENTS OF THE VARIABLES IN THE TERM AND IS RETURNED AS A CONSTANT, DGREE. EXPRESSION EXPONENTS ARE NOT ALLOWED IN THE TERM.

METHOD OF USE:
```
INTEGER PTERM, DGREE, TFLAG, KSIZE
DIMENSION KAREA(KSIZE)
CALL DGRCON(PTERM, DGREE, TFLAG, KAREA, KSIZE)
```

PARAMETERS:
PTERM (INPUT) - POINTER TO MULTIVARIATE POLYNOMIAL TERM
DGREE (OUTPUT) - VALUE OF THE DEGREE OF THIS TERM
TFLAG (OUTPUT) - TYPE FLAG OF RESULT (DEGREE)
KAREA (INPUT) - DATABASE AREA
KSIZE (INPUT) - SIZE OF THE DATABASE

ERROR CONDITIONS:
NONE

SPECIAL CONDITIONS:
IF THIS CASE CANNOT BE HANDLED DUE TO AN EXPRESSION EXPONENT, DGREE IS RETURNED EMPTY ALONG WITH A TYPE FLAG OF CNHTR.

SUBROUTINE DGREXP(PTERM, DGREE, TFLAG, KAREA, KSIZE)

DEGREE OF A TERM WITH EXPRESSION EXPONENTS

OVERVIEW:
DGREXP DETERMINES THE DEGREE OF A MULTIVARITE POLYNOMIAL TERM, PTERM. THE DEGREE IS THE SUM OF ALL THE EXPONENTS OF THE VARIABLES IN THE TERM AND TAKES THE FORM OF A LIST OF EXPONENT NODES POINTED TO BY DGREE. EXPRESSION EXPONENTS ARE ALLOWED IN THE TERM.

METHOD OF USE:
```
INTEGER PTERM, DGREE, TFLAG, KSIZE
DIMENSION KAREA(KSIZE)
CALL DGREXP(PTERM, DGREE, TFLAG, KAREA, KSIZE)
```

PARAMETERS:
PTERM (INPUT) - POINTER TO MULTIVARIATE POLYNOMIAL TERM
DGREE (OUTPUT) - POINTER TO DEGREE OF THIS TERM
TFLAG (OUTPUT) - TYPE FLAG OF RESULT (DEGREE)
KAREA (INPUT) - DATABASE AREA
KSIZE (INPUT) - SIZE OF THE DATABASE

ERROR CONDITIONS:
NONE

SPECIAL CONDITIONS:
IF THE CASE CANNOT BE HANDLED DUE TO AN EXPRESSION EXPONENT NESTED TOO DEEP, DGREE IS RETURNED EMPTY ALONG WITH A TYPE FLAG OF CNHTR.

SUBROUTINE DIVCON(MPOLY, CONST, TFLAG, KAREA, KSIZE)

DIVIDE A POLYNOMIAL BY A CONSTANT

OVERVIEW:
DIVCON DIVIDES A MULTIVARIATE POLYNOMIAL (POINTED TO BY MPOLY) BY A CONSTANT TERM ((WITH VALUE OF CONST). THE RESULTANT POLYNOMIAL IS POINTED TO BY MPOLY ON OUTPUT.

METHOD OF USE:
```
INTEGER MPOLY, TFLAG, KSIZE
```

```
      REAL CONST
      DIMENSION KAREA(KSIZE)
      CALL DIVCON(MPOLY, CONST, TFLAG, KAREA, KSIZE)
```

PARAMETERS:

```
      MPOLY (INPUT/OUTPUT) - POINTER TO POLYNOMIAL, DIVIDEND / POINTER
             TO RESULTANT POLYNOMIAL, QUOTIENT
      CONST (INPUT) - VALUE OF THE CONSTANT TERM, DIVISOR
      TFLAG (OUTPUT) - TYPE FLAG OF RESULT
      KAREA (INPUT/OUTPUT) - DATABASE AREA
      KSIZE (INPUT) - SIZE OF DATABASE
```

ERROR CONDITIONS:

```
      *NONE*
```

SPECIAL CONDITIONS:

```
      IF THE RESULT IS UNDEFINED, MPOLY IS RETURNED EMPTY ALONG  WITH  A
      TYPE FLAG OF UNDTR.
```

SUBROUTINE DIVIDE(LEFT, RIGHT, KAREA, KSIZE)

DIVISION

OVERVIEW:

```
      DIVIDE  PERFORMS  DIVISION  OF  TWO  OPERANDS,  THE  LEFT  OPERAND
      (POINTED TO BY LEFT) IS DIVIDED BY THE RIGHT OPERAND  (POINTED  TO
      BY  RIGHT).    THE RESULT IS STORED IN THE TOP OF THE TREE (I.  E.,
      THE RESULT REPLACES THE DIVISION OPERATOR NODE).
```

METHOD OF USE:

```
      INTEGER LEFT, RIGHT, KSIZE
      DIMENSION KAREA(KSIZE)
      CALL DIVIDE(LEFT, RIGHT, KAREA, KSIZE)
```

PARAMETERS:

```
      LEFT (INPUT) - POINTER TO THE LEFT OPERAND, DIVIDEND
      RIGHT (INPUT) - POINTER TO THE RIGHT OPERAND, DIVISOR
      KAREA (INPUT/OUTPUT) - DATABASE AREA
      KSIZE (INPUT) - SIZE OF DATABASE AREA
```

ERROR CONDITIONS:

```
      *NONE*
```

SPECIAL CONDITIONS:
NONE

SUBROUTINE DIVIS(NUMER, DENOM, CONST, TFLAG, KAREA, KSIZE)

DIVISION, SEVERAL DIFFERENT METHODS

OVERVIEW:
DIVIS ATTEMPTS TO SIMPLIFY THE POLYNOMIAL FORMED WITH NUMER AS THE NUMERATOR AND DENOM AS THE DENOMINATOR WITH SEVERAL ATTEMPTS - THAT IS, IT CALLS SEVERAL DIFFERENT SUBROUTINES WHICH PERFORM VARIOUS DIVIDING FUNCTIONS.

METHOD OF USE:
INTEGER NUMER, DENOM, TFLAG, KSIZE
REAL CONST
DIMENSION KAREA(KSIZE)
CALL DIVIS(NUMER, DENOM, CONST, TFLAG, KAREA, KSIZE)

PARAMETERS:
NUMER (INPUT/OUTPUT) - POINTER TO NUMERATOR POLYNOMIAL, DIVIDEND / POINTER TO RESULTANT POLYNOMIAL, QUOTIENT
DENOM (INPUT/OUTPUT) - POINTER TO DENOMINATOR POLYNOMIAL, DIVISOR
CONST (OUTPUT) - VALUE IF POLYNOMIAL REDUCES TO A CONSTANT
TFLAG (OUTPUT) - TYPE FLAG OF RESULT
KAREA (INPUT/OUTPUT) - DATABASE AREA
KSIZE (INPUT) - SIZE OF DATABASE AREA

ERROR CONDITIONS:
NONE

SPECIAL CONDITIONS:
IF THE RESULT IS A CONSTANT, NUMER AND DENOM ARE RETURNED EMPTY ALONG WITH A TYPE FLAG OF CONTR. IF THE RESULT IS UNDEFINED, NUMER AND DENOM ARE RETURNED EMPTY ALONG WITH A TYPE FLAG OF UNDTR. IF THE CASE CANNOT BE HANDLED, NUMER AND DENOM ARE RETURNED EMPTY ALONG WITH A TYPE FLAG OF CNHTR.

SUBROUTINE DIVPLY(NUMER, DENOM, TFLAG, KAREA, KSIZE)

DIVISION OF MULTIVARIATE POLYNOMIALS

OVERVIEW:
DIVPLY ATTEMPTS TO DIVIDE TWO MULTIVARIATE POLYNOMIALS. IF THE DENOMINATOR (POINTED TO BY DENOM) EVENLY DIVIDES THE NUMERATOR (POINTED TO BY NUMER), THEN UPON RETURNING THE RESULTANT POLYNOMIAL IS POINTED TO BY NUMER, AND DENOM WILL BE EMPTY. IF THE DENOMINATOR IS NOT A FACTOR OF THE NUMERATOR, NO QUOTIENT WILL BE RETURNED AND NUMER AND DENOM WILL RETURNED UNCHANGED.

METHOD OF USE:
```
INTEGER NUMER, DENOM, TFLAG, KSIZE
DIMENSION KAREA(KSIZE)
CALL DIVPLY(NUMER, DENOM, TFLAG, KAREA, KSIZE)
```

PARAMETERS:
```
NUMER (INPUT/OUTPUT) - POINTER TO THE NUMERATOR POLYNOMIAL /
       POINTER TO THE RESULTANT QUOTIENT POLYNOMIAL
DENOM (INPUT/OUTPUT) - POINTER TO THE DENOMINATOR POLYNOMIAL
TFLAG (OUTPUT) - TYPE FLAG OF RESULT
KAREA (INPUT/OUTPUT) - DATABASE AREA
KSIZE (INPUT) - SIZE OF THE DATABASE
```

ERROR CONDITIONS:
NONE

SPECIAL CONDITIONS:
IF THE CASE CANNOT BE HANDLED, NUMER AND DENOM ARE RETURNED EMPTY ALONG WITH A TYPE FLAG OF CNHTR.

SUBROUTINE DIVTRM(NMTRM, DNTRM, QOTRM, TFLAG, KAREA, KSIZE)

DIVISION OF MULTIVARIATE TERMS

OVERVIEW:
DIVTRM ATTEMPTS TO DIVIDE TWO TERMS OF A MULTIVARIATE POLYNOMIAL. IF THE DENOMINATOR TERM (POINTED TO BY DNTRM) EVENLY DIVIDES THE NUMERATOR TERM (POINTED TO BY NMTRM), THE POINTER TO THE RESULTANT QUOTIENT TERM WILL BE RETURNED IN NQOUT. IF THE DENOMINATOR IS NOT A FAC- TOR OF THE NUMERATOR, QOTRM IS RETURNED EMPTY.

METHOD OF USE:

```
INTEGER NMTRM, DNTRM, QOTRM, TFLAG, KSIZE
DIMENSION KAREA(KSIZE)
CALL DIVTRM(NMTRM, DNTRM, QOTRM, TFLAG, KAREA, KSIZE)
```

PARAMETERS:

NMTRM (INPUT) - POINTER TO THE NUMERATOR TERM
DNTRM (INPUT) - POINTER TO THE DENOMINATOR TERM
QOTRM (OUTPUT) - POINTER TO THE RESULTANT QUOTIENT TERM
TFLAG (OUTPUT) - TYPE FLAG OF RESULT(TERM)
KAREA (INPUT/OUTPUT) - DATABASE AREA
KSIZE (INPUT) - SIZE OF THE DATABASE

ERROR CONDITIONS:

NONE

SPECIAL CONDITIONS:

IF THE CASE CANNOT BE HANDLED, QOTRM IS RETURNED EMPTY ALONG WITH A TYPE FLAG OF CNHTR.

SUBROUTINE DUPEEX(ORGEE, DUPEE, TFLAG, KAREA, KSIZE)

DUPLICATE AN EXPRESSION EXPONENT

OVERVIEW:

DUPEEX DUPLICATES AN EXPRESSION EXPONENT POINTED TO BY ORGEE. THE DUPLICATED LIST IS POINTED TO BY DUPEE. BIT VECTORS AND EXPONENT LISTS FOR THE TERMS ARE DUPLICATED AS WELL, ALTHOUGH EXPRESSION EXPONENTS ARE NOT ALLOWED, SINCE THESE WOULD CAUSES EXPONENTS NESTED TOO DEEP.

METHOD OF USE:

```
INTEGER ORGEE, DUPEE, TFLAG, KSIZE
DIMENSION KAREA(KSIZE)
CALL DUPEEX(ORGEE, DUPEE, TFLAG, KAREA, KSIZE)
```

PARAMETERS:

ORGEE (INPUT) - POINTER TO ORIGINAL EXPRESSION EXPONENT
DUPEE (OUTPUT) - POINTER TO DUPLICATED EXPRESSION EXPONENT
TFLAG (OUTPUT) - TYPE FLAG OF RESULT
KAREA (INPUT) - DATABASE AREA

KSIZE (INPUT) - SIZE OF DATABASE AREA

ERROR CONDITIONS:
NONE

SPECIAL CONDITIONS:
IF THE CASE CANNOT BE HANDLED, DUPEE IS RETURNED EMPTY ALONG WITH A TYPE FLAG OF CNHTR.

SUBROUTINE DUPEXP(ORGLE, DUPLE, TFLAG, KAREA, KSIZE)

DUPLICATE A LIST OF EXPONENTS

OVERVIEW:
DUPEXP DUPLICATES THE LIST OF EXPONENTS POINTED TO BY ORGLE. THE DUPLICATED LIST IS POINTED TO BY DUPLE.

METHOD OF USE:
```
INTEGER ORGLE, DUPLE, TFLAG, KSIZE
DIMENSION KAREA(KSIZE)
CALL DUPEXP(ORGLE, DUPLE, TFLAG, KAREA, KSIZE)
```

PARAMETERS:
ORGLE (INPUT) - POINTER TO LAST EXPONENT OF ORIGINAL LIST
DUPLE (OUTPUT) - POINTER TO LAST EXPONENT OF DUPLICATED LIST
TFLAG (OUTPUT) - TYPE FLAG OF RESULT
KAREA (INPUT/OUTPUT) - DATABASE AREA
KSIZE (INPUT) - SIZE OF DATABASE AREA

ERROR CONDITIONS:
NONE

SPECIAL CONDITIONS:
IF THE CASE CANNOT BE HANDLED, DUPLE IS RETURNED EMPTY ALONG WITH A TYPE FLAG OF CNHTR.

```
SUBROUTINE DUPTRM(ORGMP, DUPMP, TFLAG, KAREA, KSIZE)

      DUPLICATE A LIST OF TERMS

   OVERVIEW:
      DUPTRM  DUPLICATES  A  LIST  OF TERMS (A POLYNOMIAL) POINTED TO BY
      ORGMP.  THE DUPLICATED LIST IS POINTED TO BY DUPMP.   BIT  VECTORS
      AND EXPONENT LISTS FOR THE TERMS ARE DUPLICATED AS WELL.


   METHOD OF USE:
      INTEGER ORGMP, DUPMP, TFLAG, KSIZE
      DIMENSION KAREA(KSIZE)
      CALL DUPTRM(ORGMP, DUPMP, TFLAG, KAREA, KSIZE)


   PARAMETERS:
      ORGMP (INPUT) - POINTER TO ORIGINAL POLYNOMIAL
      DUPMP (OUTPUT) - POINTER TO DUPLICATED POLYNOMIAL
      TFLAG (OUTPUT) - TYPE FLAG OF RESULT
      KAREA (INPUT) - DATABASE AREA
      KSIZE (INPUT) - SIZE OF DATABASE AREA


   ERROR CONDITIONS:
      *NONE*


   SPECIAL CONDITIONS:
      IF  THE CASE CANNOT BE HANDLED, DUPMP IS RETURNED EMPTY ALONG WITH
      A TYPE FLAG OF CNHTR.



SUBROUTINE ENDEX(OPSTK, VARSK, NUMWD, LENSK, LINDR, STAT, ERROR,
                 KAREA, KSIZE)

      END OF EXPRESSION


   OVERVIEW:
      ENDEX POPS ANY REMAINING OPERATORS AND VARIABLES OFF  THE  STACKS,
      CALLS  TRAVEL TO TRAVERSE THE TREE, AND THEN DETERMINES THE STATUS
      CODE AND ERROR FLAG.


   METHOD OF USE:
      INTEGER OPSTK(NUMWD, LENSK), NUMWD, LENSK
      INTEGER IBND, STAT, ERROR, KSIZE
```

```
      LOGICAL LINDR
      DIMENSION KAREA(KSIZE)
      CALL ENDEX(OPSTK, VARSK, NUMWD, LENSK, IBND, LINDR, STAT, ERROR,
                 KAREA, KSIZE)
```

PARAMETERS:

OPSTK (INPUT) - THE OPERATOR STACK
VARSK (INPUT) - THE VARIABLE STACK
NUMWD (INPUT) - WIDTH OF THE STACKS
LENSK (INPUT) - LENGTH OF THE STACKS
IBND (INPUT) - INTEGER VARIABLE SUBSCRIPT BOUND
LINDR (INPUT) - LOGICAL FLAG WHICH SIGNALS WHETHER LINEARIZATION SHOULD BE PERFORMED IN TRYLNR (ATTEST DEPENDENT)
STAT (OUTPUT) - STATUS OF THE REDUCED POLYNOMIAL (SEE SPECIAL CONDITIONS)
ERROR (OUTPUT) - ERROR FLAG (SEE ERROR CONDITIONS)
KAREA (INPUT) - DATA BASE AREA
KSIZE (INPUT) - SIZE OF THE DATA BASE

ERROR CONDITIONS:

IF AN ERROR IS ENCOUNTERED IN THE PARSING OF THE EXPRESSION, THE VALUE OF THE ERROR FLAG WILL INDICATE THE TYPE OF ERROR SEE THE SYMPLR SUBROUTINE FOR EXPLANATION OF ERROR VALUES.

SPECIAL CONDITIONS:

ALTHOUGH THERE MAY BE NO ERROR INDICATED BY ERROR, THE EXPRESSION MAY NOT BE SIMPLIFIABLE, IN WHICH CASE THE VALUE OF STAT WILL INDICATE THE STATUS OF THE POLYNOMIAL (SEE THE SYMPLR SUBROUTINE FOR EXPLANATION OF STATUS CODES).

SUBROUTINE EXPADD(OPVAL, LFTEX, RITEX, SUMEX, TFLAG, KAREA, KSIZE)

EXPONENT ADDITION AND SUBTRACTION

OVERVIEW:

EXPADD ADDS OR SUBTRACTS (DEPENDING ON THE VALUE OF OPVAL) THE EXPONENTS OF THE NODES PONTED TO BY LFTEX AND RITEX, RETURNING THE SUM OR DIFFERENCE AS AN EXPONENT POINTED TO BY SUMEX. IF EITHER LFTEX OR RITEX HAS THE VALUE 0 OR 1, THERE WAS NO EXPONENT NODE (FOR THE VARIABLE TO WHICH THESE EXPONENTS CORRESPOND) BUT THE IMPLIED EXPONENT VALUE IS 0.0 OR 1.0 RESPECTIVELY. LIKEWISE, IF SUMEX IS RETURNED AS THE VALUE 0 OR 1, NO EXPONENT NODE WAS NECESSARY, BUT THE VALUE OF THE EXPONENT (FOR THE SAME VARIABLE) IS 0.0 OR 1.0, RESPECTIVELY. AN EXPONENT WITH SUCH A IMPLIED VALUE IS OF TYPE CONSTANT.

METHOD OF USE:

```
INTEGER OPVAL, LFTEX, RITEX, SUMEX, TFLAG, KSIZE
DIMENSION KAREA(KSIZE)
CALL EXPADD(OPVAL, LFTEX, RITEX, SUMEX, TFLAG, KAREA, KSIZE)
```

PARAMETERS:

OPVAL (INPUT) - VALUE OF THE OPERATOR (PLUS OR MINUS)
LFTEX (INPUT) - POINTER TO LEFT EXPONENT NODE
RITEX (INPUT) - POINTER TO RIGHT EXPONENT NODE
SUMEX (OUTPUT) - POINTER TO SUM (DIFFERENCE) EXPONENT NODE
TFLAG (OUTPUT) - TYPE FLAG OF RESULT
KAREA (INPUT/OUTPUT) - DATABASE AREA
KSIZE (INPUT) - SIZE OF DATABASE

ERROR CONDITIONS:

NONE

SPECIAL CONDITIONS:

IF THE CASE CANNOT BE HANDLED DUE TO AN EXPRESSION EXPONENT NESTED TOO DEEP, SUMEX IS RETURNED EMPTY ALONG WITH A TYPE FLAG OF CNHTR. IF A NEGATIVE EXPONENT IS GENERATED, THE POINTER TO IT IS RETURNED (FOR THOSE CALLING ROUTINES WHICH ALLOW IT) ALONG WITH A TYPE FLAG OF BLKTR (FOR THOSE THAT DON'T).

SUBROUTINE EXPCON(NUMER, DENOM, EXVAL, TFLAG, POWER, TERMS, KAREA, KSIZE)

MULTIVARIATE POLYNOMIAL EXPONENTIATION TO A CONSTANT POWER ONLY

OVERVIEW:

EXPCON RAISES THE POLYNOMIAL WITH NUMERATOR POINTED TO BY NUMER AND DENOMINATOR POINTED TO BY DENOM TO THE CONSTANT VALUE EXVAL. IF THE EXPONENTIATION IS SUCCESSFUL, THEN EXPCON RETURNS WITH NUMER AND DENOM POINTING TO THE NUMERATOR AND DENOMINATOR OF THE EXPONENTIATED POLYNOMIAL.

METHOD OF USE:

```
INTEGER NUMER, DENOM, TFLAG, TERMS, KSIZE
REAL EXVAL, POWER
DIMENSION KAREA(KSIZE)
CALL EXPCON(NUMER, DENOM, EXVAL, TFLAG, POWER, TERMS, KAREA,
            KSIZE)
```

PARAMETERS:
NUMER (INPUT/OUTPUT) - POINTER TO THE NUMERATOR
DENOM (INPUT/OUTPUT) - POINTER TO THE DENOMINATOR
EXVAL (INPUT) - VALUE OF THE CONSTANT POWER
TFLAG (OUTPUT) - TYPE FLAG OF THE RESULT
POWER (INPUT) - HIGHEST EXPONENT VALUE PERMITTED
TERMS (INPUT) - MAXIMUM NUMBER OF TERMS PERMITTED
KAREA (INPUT/OUTPUT) - DATABASE AREA
KSIZE (INPUT) - SIZE OF THE DATABASE

ERROR CONDITIONS:
NONE

SPECIAL CONDITIONS:
IF THE RESULT IS ZERO, NUMER AND DENOM ARE RETURNED EMPTY ALONG WITH A TYPE FLAG OF CNHTR. IF THE RESULT IS UNDEFINED, NUMER AND DENOM ARE RETURNED EMPTY ALONG WITH A TYPE FLAG OF UNDTR. IF THE EXPONENTIATION IS DISCONTINUED, NUMER AND DENOM ARE RETURNED EMPTY ALONG WITH A TYPE FLAG OF BLKTR.

SUBROUTINE EXPPLY(NUMER, DENOM, EXPLY, TFLAG, KAREA, KSIZE)

EXPONENTIATE A POLYNOMIAL TO A POLYNOMIAL

OVERVIEW:
EXPPLY RAISES THE POLYNOMIAL WITH NUMERATOR POINTED TO BY NUMER AND DENOMINATOR POINTED TO BY DENOM TO THE POLYNOMIAL POINTED TO BY EXPLY, WHICH IS AN EXPRESSION EXPONENT. NOTE THAT THE EXPONENTIATION IS DONE FOR ONE TERM OF NUMER AND DENOM ONLY, AND IT IS ASSUMED THAT THIS SINGLE COMPLEXITY IS CHECKED FOR IN THE CALLING ROUTINE. IF THE EXPONENTIATION IS SUCCESSFUL, EXPPLY RETURNS WITH NUMER AND DENOM POINTING TO THE NUMERATOR AND DENOMINATOR OF THE EXPONENTIATED POLYNOMIAL.

METHOD OF USE:
INTEGER NUMER, DENOM, EXPLY, TFLAG, KSIZE
DIMENSION KAREA(KSIZE)
CALL EXPPLY(NUMER, DENOM, EXPLY, TFLAG, KAREA, KSIZE)

PARAMETERS:
NUMER (INPUT/OUTPUT) - POINTER TO THE NUMERATOR
DENOM (INPUT/OUTPUT) - POINTER TO THE DENOMINATOR

EXPLY (INPUT) - POINTER TO THE EXPRESSION EXPONENT
TFLAG (OUTPUT) - TYPE FLAG OF THE RESULT
KAREA (INPUT/OUTPUT) - DATABASE AREA
KSIZE (INPUT) - SIZE OF THE DATABASE

ERROR CONDITIONS:
NONE

SPECIAL CONDITIONS:
IF AN EXPRESSION EXPONENT THAT IS NESTED TOO DEEP IS ENCOUNTERED, NUMER AND DENOM ARE RETURNED EMPTY ALONG WITH A TYPE FLAG OF CNHTR.

SUBROUTINE EXPON(LEFT, RIGHT, KAREA, KSIZE)

EXPONENTIATION

OVERVIEW:
EXPON RAISES THE CONTENTS OF THE NODE POINTED TO BY LEFT TO THE CONTENTS OF THE NODE POINTED TO BY RIGHT. IF THE EXPONENTIATION PROCESS IS SUCCESSFUL, EXPON PLACES THE POINTER TO THE RESULT AT THE TOP OF THE CURRENT TREE.

METHOD OF USE:
INTEGER LEFT, RIGHT, KSIZE
DIMENSION KAREA(KSIZE)
CALL EXPON(LEFT, RIGHT, KAREA, KSIZE)

PARAMETERS:
LEFT (INPUT) - POINTER TO THE LEFT NODE
RIGHT (INPUT) - POINTER TO THE RIGHT NODE
KAREA (INPUT) - DATA BASE AREA
KSIZE (INPUT) - SIZE OF THE DATA BASE AREA

ERROR CONDITIONS:
NONE

SPECIAL CONDITIONS:
EXPONENTIATION IS DISCONTINUED IF THE BASIS POLYNOMIAL HAS TOO MANY TERMS OR THE POWER IS TOO LARGE. AND FOR THE TREE TO HOLD THE SIMPLIFIED POLYNOMIAL

INTEGER FUNCTION EXPORD(LFTEX, RITEX, KAREA, KSIZE)

EXPONENT ORDER

OVERVIEW:
EXPORD DETERMINES THE ORDERING OF TWO EXPONENTS. EXPORD RETURNS
1 - IF LFTEX EQUALS RITEX
2 - IF LFTEX SHOULD PRECEDE RITEX
3 - IF LFTEX SHOULD FOLLOW RITEX
4 - IF THE CASE IS NOT HANDLED (SEE SPECIAL CONDITIONS)

METHOD OF USE:
INTEGER LFTEX, RITEX, KSIZE
DIMENSION KAREA(KSIZE)
INTEGER EXPORD
----- = EXPORD(LFTEX, RITEX, KAREA, KSIZE)

PARAMETERS:
LFTEX (INPUT) - POINTER TO LEFT EXPONENT NODE
RITEX (INPUT) - POINTER TO RIGHT EXPONENT NODE
KAREA (INPUT/OUTPUT) - DATABASE AREA
KSIZE (INPUT) - SIZE OF THE DATABASE

ERROR CONDITIONS:
NONE

SPECIAL CONDITIONS:
EXPRESSIONS WITH TERMS CONTAINING EXPRESSION EXPONENTS ARE NOT HANDLED. THIS LIMITATION IS DUE TO THE RECURSION WHICH WOULD BE INVOLVED IN ORDERING THIS SECOND LEVEL OF EXPRESSION EXPONENTS.

SUBROUTINE FACTOR(NUMER, DENOM, TFLAG, KAREA, KSIZE)

REMOVE COMMON FACTORS

OVERVIEW:
FACTOR REMOVES COMMON FACTORS IN THE NUMERATOR (POINTED TO BY NUMER) AND THE DENOMINATOR (POINTED BY DENOM). COMMON FACTOR ARE SINGLE VARIABLES. IF THE DENOMINATOR FACTORS OUT COMPLETELY, IT IS A CONSTANT AND IS DIVIDED INTO THE NUMER- ATOR AND AN EMPTY POINTER IS RETURNED FOR THE DENOMINATOR.

METHOD OF USE:

```
INTEGER NUMER, DENOM, TFLAG, KSIZE
DIMENSION KAREA(KSIZE)
CALL FACTOR(NUMER, DENOM, TFLAG, KAREA, KSIZE)
```

PARAMETERS:

NUMER (INPUT/OUTPUT) - POINTER TO NUMERATOR, DIVIDEND
DENOM (INPUT/OUTPUT) - POINTER TO DENOMINATOR, DIVISOR
TFLAG (OUTPUT) - TYPE FLAG OF RESULT
KAREA (INPUT/OUTPUT) - DATABASE AREA
KSIZE(INPUT) - SIZE OF DATABASE AREA

ERROR CONDITIONS:

NONE

SPECIAL CONDITIONS:

IF THE RESULT IS ZERO, NUMER AND DENOM ARE RETURNED EMPTY ALONG WITH A TYPE FLAG OF CONTR. IF THE RESULT IS UNDEFINED, NUMER AND DENOM ARE RETURNED EMPTY ALONG WITH A TYPE FLAG OF UNDTR. IF THE CASE CANNOT BE HANDLED, NUMER AND DENOM ARE RETURNED EMPTY ALONG WITH A TYPE FLAG OF CNHTR.

SUBROUTINE FACVAR(MPOLY, IVAR, TFLAG, KAREA, KSIZE)

FACTOR A VARIABLE OUT OF A MULTIVARIATE POLYNOMIAL

OVERVIEW:

FACVAR FACTORS THE VARIABLE SPECIFIED BY IVAR OUT OF EACH TERM OF THE POLYNOMIAL POINTED TO BY MPOLY

METHOD OF USE:

```
INTEGER MPOLY, IVAR, TFLAG, KSIZE
DIMENSION KAREA(KSIZE)
CALL FACVAR(MPOLY, IVAR, TFLAG, KAREA, KSIZE)
```

PARAMETERS:

MPOLY (INPUT/OUTPUT) - POINTER TO MULTIVARIATE POLYNOMIAL
IVAR (INPUT) - VARIABLE TO BE FACTORED OUT
TFLAG (OUTPUT) - TYPE FLAG OF RESULT
KAREA (INPUT/OUTPUT) - DATABASE AREA
KSIZE (INPUT) - SIZE OF DATABASE AREA

ERROR CONDITIONS:
NONE

SPECIAL CONDITIONS:
IF THE CASE CANNOT BE HANDLED, MPOLY IS RETURNED EMPTY ALONG WITH A TYPE FLAG OF CNHTR.

LOGICAL FUNCTION ISCNST(MPOLY, CONST, KAREA, KSIZE)

IS A MULTIVARIATE POLYNOMIAL SIMPLY A CONSTANT?

OVERVIEW:
ISCNST RETURNS FALSE IF THE MULTIVARIATE POLYNOMIAL (POINTED TO BY MPOLY) IS MORE THAN A CONSTANT. IF IT IS JUST A CONSTANT, ISCNST RETURNS TRUE, AND THE VALUE OF THE CONSTANT IS RETURNED IN CONST.)

METHOD OF USE:
```
INTEGER MPOLY, KSIZE
REAL CONST
DIMENSION KAREA(KSIZE)
LOGICAL ISCNST
IF (ISCNST(MPOLY, CONST, KAREA, KSIZE))
```

PARAMETERS:
MPOLY (INPUT) - POINTER TO THE MULTIVARIATE POLYNOMIAL
CONST (OUTPUT) - VALUE OF THE CONSTANT IF MPOLY IS A CONSTANT
KAREA (INPUT) - DATABASE AREA
KSIZE (INPUT) - SIZE OF DATABASE

ERROR CONDITIONS:
NONE

SPECIAL CONDITIONS:
NONE

LOGICAL FUNCTION ISLINR(MPOLY, KAREA, KSIZE)

IS A MULTIVARIATE POLYNOMIAL LINEAR?

```
   OVERVIEW:
      ISLINR  RETURNS TRUE IF THE MULTIVARIATE POLYNOMIAL (POINTED TO BY
      MPOLY) IS LINEAR, OTHERWISE ISLINR RETURNS TRUE.


   METHOD OF USE:
      INTEGER MPOLY, KSIZE
      DIMENSION KAREA(KSIZE)
      LOGICAL ISLINR
      IF (ISLINR(MPOLY, KAREA, KSIZE))


   PARAMETERS:
      MPOLY (INPUT) - POINTER TO THE MULTIVARIATE POLYNOMIAL
      KAREA (INPUT) - DATABASE AREA
      KSIZE (INPUT) - SIZE OF DATABASE


   ERROR CONDITIONS:
      *NONE*


   SPECIAL CONDITIONS:
      *NONE*



INTEGER FUNCTION KMPLXY(MPOLY, KAREA, KSIZE)


      COMPLEXITY COUNT OF A MULTIVARIATE POLYNOMIAL


   OVERVIEW:
      KMPLXY RETURNS THE COMPLEXITY COUNT  OF  A  POLYNOMIAL  POLYNOMIAL
      (POINTED TO BY MPOLY)


   METHOD OF USE:
      INTEGER MPOLY, KSIZE
      DIMENSION KAREA(KSIZE)
      INTEGER KMPLXY
      ------ = KMPLXY(MPOLY, KAREA, KSIZE)


   PARAMETERS:
      MPOLY (INPUT) - POINTER TO THE MULTIVARIATE POLYNOMIAL
      KAREA (INPUT) - DATABASE AREA
      KSIZE (INPUT) - SIZE OF DATABASE


   ERROR CONDITIONS:
```

```
      *NONE*

   SPECIAL CONDITIONS:
      IF THE POINTER IS EMPTY, KMPLXY IS RETURNED = 0.



SUBROUTINE MULCON(MPOLY, CONST, TFLAG, KAREA, KSIZE)

      MULTIPLY A POLYNOMIAL BY A CONSTANT

   OVERVIEW:
      MULCON  MULTIPLIES A MULTIVARIATE POLYNOMIAL (POINTED TO BY MPOLY)
      BY A CONSTANT TERM (WITH VALUE OF CONST).


   METHOD OF USE:
      INTEGER MPOLY, TFLAG, KSIZE
      REAL CONST
      DIMENSION KAREA(KSIZE)
      CALL MULCON(MPOLY, CONST, TFLAG, KAREA, KSIZE)


   PARAMETERS:
      MPOLY (INPUT/OUTPUT) - POINTER TO POLYNOMIAL, MULTIPLICAND
      CONST (INPUT) - VALUE OF THE CONSTANT TERM, MULTIPLIER
      TFLAG (OUTPUT) - TYPE FLAG OF RESULT
      KAREA (INPUT/OUTPUT) - DATABASE AREA
      KSIZE (INPUT) - SIZE OF DATABASE


   ERROR CONDITIONS:
      *NONE*


   SPECIAL CONDITIONS:
      IF THE RESULT IS ZERO, MPOLY IS RETURNED EMPTY ALONG WITH  A  TYPE
      FLAG OF CONTR.



SUBROUTINE MULPLY(LFTMP, RITMP, TFLAG, KAREA, KSIZE)

      MULTIPLICATION OF MULTIVARIATE POLYNOMIALS


   OVERVIEW:
```

MULPLY MULTIPLIES THE LEFT POLYNOMIAL (POINTED TO BY LFTMP) BY THE RIGHT POLYNOMIAL (POINTED TO BY RITMP). LFTMP POINTS TO THE RESULTANT POLYNOMIAL, THE PRODUCT.

METHOD OF USE:

```
INTEGER LFTMP, RITMP, TFLAG, KSIZE
DIMENSION KAREA(KSIZE)
CALL MULPLY(LFTMP, RITMP, TFLAG, KAREA, KSIZE)
```

PARAMETERS:

LFTMP (INPUT/OUTPUT) - POINTER TO LEFT POLYNOMIAL, MULTIPLICAND / POINTER TO RESULTANT POLYNOMIAL, PRODUCT
RITMP (INPUT) - POINTER TO RIGHT POLYNOMIAL, MULTIPLIER
TFLAG (OUTPUT) - TYPE FLAG OF RESULT
KAREA (INPUT/OUTPUT) - DATABASE AREA
KSIZE (INPUT) - SIZE OF DATABASE AREA

ERROR CONDITIONS:

NONE

SPECIAL CONDITIONS:

IF THE RESULT IS ZERO, LFTMP IS RETURNED EMPTY ALONG WITH A TYPE FLAG OF CONTR. IF THE RESULT IS UNDEFINED, LFTMP IS RETURNED EMPTY ALONG WITH A TYPE FLAG OF UNDTR. IF THE CASE CANNOT BE HANDLED, LFTMP IS RETURNED EMPTY ALONG WITH A TYPE FLAG OF CNHTR.

SUBROUTINE MULT(LEFT, RIGHT, KAREA, KSIZE)

MULTIPLICATION

OVERVIEW:

MULT PERFORMS MULTIPLICATION OF TWO OPERANDS, THE LEFT OPERAND (POINTED TO BY LEFT) IS MULTIPLIED BY THE RIGHT OPERAND (POINTED TO BY RIGHT). THE RESULT IS STORED IN THE TOP OF THE TREE (I. E., THE RESULT REPLACES THE MULTIPLICATION OPERATOR NODE).

METHOD OF USE:

```
INTEGER LEFT, RIGHT, KSIZE
DIMENSION KAREA(KSIZE)
CALL MULT(LEFT, RIGHT, KAREA, KSIZE)
```

PARAMETERS:

LEFT (INPUT) - POINTER TO THE LEFT OPERAND
RIGHT (INPUT) - POINTER TO THE RIGHT OPERAND
KAREA (INPUT/OUTPUT) - DATABASE AREA
KSIZE (INPUT) - SIZE OF DATABASE AREA

ERROR CONDITIONS:
NONE

SPECIAL CONDITIONS:
NONE AND FOR THE TREE TO HOLD THE SIMPLIFIED POLYNOMIAL

SUBROUTINE MULTRM(LFTRM, RITRM, PRTRM, TFLAG, KAREA, KSIZE)

MULTIPLICATION OF MULTIVARIATE TERMS

OVERVIEW:
MULTRM ATTEMPTS TO MULTIPLY TWO TERMS OF A MULTIVARIATE POLYNOMIAL. THE RESULT (POINTED TO BY PRTRM) IS THE PRODUCT OF THE LEFT TERM (POINTED TO BY LFTRM) AND THE RIGHT TERM (POINTED TO BY RITRM). MULTRM ALLOCATES THE NODE FOR THE PRODUCT WHOSE POINTER IS PRTRM. MULTRM ALSO ALLOCATES A BIT VECTOR AND CREATES AN EXPONENT LIST FOR THE PRODUCT TERM.

METHOD OF USE:
INTEGER LFTRM, RITRM, PRTRM, TFLAG, KSIZE
DIMENSION KAREA(KSIZE
CALL MULTRM(LFTRM, RITRM, PRTRM, TFLAG, KAREA, KSIZE)

PARAMETERS:
LFTRM (INPUT) - POINTER TO THE LEFT TERM
RITRM (INPUT) - POINTER TO THE RIGHT TERM
PRTRM (OUTPUT) - POINTER TO THE RESULTANT PRODUCT TERM
TFLAG (OUTPUT) - TYPE FLAG OF RESULT (TERM)
KAREA (INPUT) - DATABASE AREA
KSIZE (INPUT) - SIZE OF THE DATABASE

ERROR CONDITIONS:
NONE

SPECIAL CONDITIONS:
IF THE CASE CANNOT BE HANDLED, PRTRM IS RETURNED EMPTY ALONG WITH A TYPE FLAG OF CNHTR.

```
SUBROUTINE NEGATE(MPOLY, KAREA, KSIZE)

      NEGATE A MULTIVARIATE POLYNOMIAL

   OVERVIEW:
      NEGATE RETURNS THE MULTIVARIATE POLYNOMIAL (POINTED TO  BY  MPOLY)
      WITH ALL ITS COEFFICIENTS NEGATED.


   METHOD OF USE:
      INTEGER MPOLY, KSIZE
      DIMENSION KAREA(KSIZE)
      CALL NEGATE(MPOLY, KAREA, KSIZE)


   PARAMETERS:
      MPOLY (INPUT) - POINTER TO THE MULTIVARIATE POLYNOMIAL
      KAREA (INPUT) - DATABASE AREA
      KSIZE (INPUT) - SIZE OF DATABASE


   ERROR CONDITIONS:
      *NONE*


   SPECIAL CONDITIONS:
      *NONE*



SUBROUTINE NUMBER(EXPOS, EXPRS, EXDIM, VARSK, NUMWD, LENSK, KAREA,
                  KSIZE)

      SET UP A NUMBER


   OVERVIEW:
      NUMBER  SETS UP A TREE NODE IN THE DATABASE FOR A NUMBER WHICH HAS
      BEEN ENCOUNTERED IN THE INPUT EXPRESSION, EXPRS.  THE LOCATION  OF
      THE FIRST DIGIT IN EXPRS IS POINTED TO BY EXPOS.


   METHOD OF USE:
      INTEGER EXDIM, KSIZE
      INTEGER EXPOS, EXPRS(EXDIM)
```

```
      INTEGER VARSK(NUMWD, LENSK), NUMWD, LENSK
      DIMENSION KAREA(KSIZE)
      CALL NUMBER(EXPOS, EXPRS, EXDIM, VARSK, NUMWD, LENSK, KAREA,
                  KSIZE)
```

PARAMETERS:

```
   EXPOS (INPUT) - POINTER TO THE FIRST DIGIT OF THE NUMBER
   EXPRS (INPUT) - EXPRESSION ARRAY
   VARSK (INPUT) - VARIABLE STACK
   NUMWD (INPUT) - WIDTH OF THE VARIABLE STACK
   LENSK (INPUT) - LENGTH OF THE VARIABLE STACK
   KAREA (INPUT) - DATA BASE AREA
   KSIZE (INPUT) - SIZE OF THE DATA BASE
```

ERROR CONDITIONS:

```
   *NONE*
```

SPECIAL CONDITIONS:

```
   *NONE*
```

```
SUBROUTINE OUTEXP(EXPOS, EXPRS, EXDIM, EXPON, IBND, BVSIZ, STAT, ERROR,
                  KAREA, KSIZE)
```

```
   OUTPUT AN EXPRESSION EXPONENT
```

OVERVIEW:

```
   OUTEXP   OUTPUTS   AN   EXPRESSION   EXPONENT   IN   ITS   ALPHANUMERIC
   REPRESENTATION.
```

METHOD OF USE:

```
      INTEGER EXDIM
      INTEGER EXPOS, EXPRS(EXDIM)
      INTEGER EXPON, IBND, BVSIZ
      INTEGER STAT, ERROR, KSIZE
      DIMENSION KAREA(KSIZE)
      CALL OUTEXP(EXPOS, EXPRS, EXDIM, EXPON, IBND, BVSIZ, STAT, ERROR,
                  KAREA, KSIZE)
```

PARAMETERS:

```
   EXPOS (INPUT/OUTPUT) - POSITION IN EXPRS TO PUT EXPRESSION
          EXPONENT
   EXPRS (INPUT/OUTPUT) - EXPRESSION ARRAY IN WHICH TO PUT EXPONENT
   EXDIM (INPUT) - DIMENSION OF EXPRESSION ARRAY
```

```
   EXPON (INPUT) - POINTER TO EXPRESSION EXPONENT LIST
   IBND (INPUT) - THE INTEGER BOUNDS
   BVSIZ (INPUT) - SIZE OF THE BIT VECTOR
   STAT (OUTPUT) - STATUS OF RESULT POLYNOMIAL (SEE SPECIAL
          CONDITIONS)
   ERROR (OUTPUT) - ERROR FLAG (SEE ERROR CONDITIONS)
   KAREA(INPUT) - DATA BASE AREA
   KSIZE(INPUT) - SIZE OF THE DATA BASE


ERROR CONDITIONS:
   IF THE EXPRS ARRAY IS OVERFLOWED, ERROR = OVFLW (1) IS RETURNED.

SPECIAL CONDITIONS:
   IF AN EXPRESSION EXPONENT IS PRESENT IN THIS  EXPRESSION  EXPONENT
   STAT = MFEXP (3) IS RETURNED.



SUBROUTINE OUTREE(EXPRS, EXLEN, EXDIM, IBND, BVSIZ, STAT, ERROR,
                  KAREA, KSIZE)


   OUTPUT THE TREE


OVERVIEW:
   OUTREE   OUTPUTS   THE   REDUCED   POLYNOMIAL   IN   ITS   ALPHANUMERIC
   REPRESENTATION.


METHOD OF USE:
   INTEGER EXDIM, EXLEN
   INTEGER EXPRS(EXDIM)
   INTEGER IBND, BVSIZ, STAT, ERROR, KSIZE
   DIMENSION KAREA(KSIZE)
   CALL OUTREE(EXPRS, EXLEN, EXDIM, IBND, BVSIZ, STAT, ERROR, KAREA,
               KSIZE)


PARAMETERS:
   EXPRS (OUTPUT) - EXPRESSION ARRAY IN WHICH TO PUT POLYNOMIAL
   EXLEN (OUTPUT) - LENGTH OF THE REDUCED POLYNOMIAL
   EXDIM (INPUT) - DIMENSION OF EXPRESSION ARRAY
   IBND (INPUT) - THE INTEGER BOUNDS
   BVSIZ (INPUT) - SIZE OF THE BIT VECTOR
   STAT (OUTPUT) - STATUS OF RESULT POLYNOMIAL (SEE SPECIAL
          CONDITIONS)
   ERROR (OUTPUT) - ERROR FLAG (SEE ERROR CONDITIONS)
   KAREA (INPUT) - DATA BASE AREA
   KSIZE (INPUT) - SIZE OF THE DATA BASE
```

ERROR CONDITIONS:
IF THE EXPRS ARRAY IS OVERFLOWED, ERROR = OVFLW (1) IS RETURNED.

SPECIAL CONDITIONS:
IF AN EXPRESSION EXPONENT IS PRESENT IN THE FINAL POLYNOMIAL STAT = MFEXP (3) IS RETURNED.

SUBROUTINE OUTVEC(COEFS, CFDIM, BVSIZ, KAREA, KSIZE)

OUTPUT THE VECTOR

OVERVIEW:
OUTVEC OUTPUTS THE VECTOR OF COEFFICIENTS FOR THE SIMPLIFIED POLYNOMIAL. IT IS ASSUMED THAT THE POLYNOMIAL IS LINEAR IF OUTVEC IS CALLED; THIS MUST BE CHECKED OUTSIDE OF THIS ROUTINE.

METHOD OF USE:
```
INTEGER CFDIM, BVSIZ, KSIZE
DIMENSION COEFS(CFDIM)
DIMENSION KAREA(KSIZE)
CALL OUTVEC(COEFS, CFDIM, BVSIZ, KAREA, KSIZE)
```

PARAMETERS:
COEFS (OUTPUT) - OUTPUT VECTOR OF COEFFICIENTS
CFDIM (INPUT) - DIMENSION OF COEFS
BVSIZ (INPUT) - SIZE OF A BIT VECTOR
KAREA (INPUT) - DATA BASE AREA
KSIZE (INPUT) - SIZE OF THE DATA BASE

ERROR CONDITIONS:
NONE

SPECIAL CONDITIONS:
NONE

INTEGER FUNCTION PLYORD(LFTMP, RITMP, KAREA, KSIZE)

POLYNOMIAL ORDERING

OVERVIEW:
PLYORD DETERMINES THE ORDERING OF TWO MULTIVARIATE POLYNOMIALS. THE ORDERING IS DEPENDENT ON THE TERMS IN EACH EXPRESSION, THEIR DEGREE AND VARIABLES, LIKE THE ORDERING OF TERMS IN A SUM. PLYORD RETURNS
1 - IF LFTMP EQUALS RITMP
2 - IF LFTMP SHOULD PRECEDE RITMP
3 - IF LFTMP SHOULD FOLLOW RITMP
4 - IF THE CASE IS NOT HANDLED (SEE SPECIAL CONDITIONS)

METHOD OF USE:
INTEGER PLYORD, LFTMP, RITMP
----- = PLYORD(LFTMP, RITMP, KAREA, KSIZE)

PARAMETERS:
LFTMP (INPUT) - POINTER TO LAST TERM OF LEFT POLYNOMIAL
RITMP (INPUT) - POINTER TO LAST TERM OF RIGHT POLYNOMIAL
KAREA (INPUT) - DATABASE AREA
KSIZE (INPUT) - SIZE OF THE DATABASE

ERROR CONDITIONS:
NONE

SPECIAL CONDITIONS:
PLYORD DOES NOT HANDLE EXPRESSION EXPONENTS FOR THE POLYNOMIALS WHICH IT IS ORDERING. IF THIS CASE OCCURS, PLYORD RETURNS WITH VALUE 4. THIS RESTRICTION IS DUE TO THE LACK OF RECURSION IN FORTRAN, SINCE THE ORDER OF THOSE EXPRESSION EXPONENTS WOULD HAVE TO BE DETERMINED AS WELL. SINCE THIS ROUTINE IS ONLY CALLED FOR EXPONENTS WITHIN SYMPLR, EXPRESSION EXPONENTS ARE ALLOWED TO ONE LEVEL OF NESTING.

LOGICAL FUNCTION POP(STACK, NUMWD, LENSK, INFO)

POP THE STACK

OVERVIEW:
POP IS FALSE IF THE STACK IS EMPTY. IF THE STACK IS NOT EMPTY POP

IS TRUE AND THE ARRAY INFO CONTAINS THE TOP NUMWD WORDS OF THE STACK. THE STACK POINTER IS MAINTAINED IN THE FIRST WORD OF THE STACK ARRAY. THIS ELEMENT SHOULD BE INITIALIZED TO 1 (STACK(1,1)=1) BEFORE PUSH OR POP ARE FIRST CALLED.

METHOD OF USE:

```
INTEGER NUMWD, LENSK
INTEGER STACK(NUMWD, LENSK), INFO(NUMWD)
LOGICAL POP
IF (POP(STACK, NUMWD, LENSK, INFO))
```

PARAMETERS:

STACK (INPUT/OUTPUT) - 2 DIMENSIONAL ARRAY THAT CONTAINS THE STACK
NUMWD (INPUT) - THE NUMBER OF WORDS PER STACK ENTRY
LENSK (INPUT) - THE MAXIMUM LENGTH OF THE STACK
INFO (OUTPUT) - THE ARRAY CONTAINING THE STACK ENTRY ON TOP OF THE STACK

ERROR CONDITIONS:

NONE

SPECIAL CONDITIONS:

NONE

SUBROUTINE POPVAR(VARSK, NUMWD, LENSK, INFO, ERROR, KAREA, KSIZE)

POP THE VARIABLE STACK

OVERVIEW:

POPVAR POPS THE VARIABLE STACK (VARSK) TWICE FOR ANY OPERATOR (IN INFO) EXCEPT UNARY MINUS FOR WHICH IT POPS ONLY ONCE). POPVAR SETS THE LEFT AND RIGHT LINKS OF THE OPERATOR NODE.

METHOD OF USE:

```
INTEGER VARSK(NUMWD, LENSK), NUMWD, LENSK
INTEGER INFO(2), ERROR, KSIZE
DIMENSION KAREA(KSIZE)
CALL POPVAR(VARSK, NUMWD, LENSK, INFO, ERROR, KAREA, KSIZE)
```

PARAMETERS:

VARSK (INPUT) - THE VARIABLE STACK
NUMWD (INPUT) - NUMBER OF WORDS PER ENTRY IN THE STACK

```
   LENSK (INPUT) - LENGTH OF THE STACK
   INFO (INPUT) - BINDING STRENGTH AND ADDRESS OF THE OPERATOR
   ERROR (OUTPUT) - ERROR FLAG (SEE ERROR CONDITIONS)
   KAREA (INPUT/OUTPUT) - DATABASE AREA
   KSIZE (INPUT) - SIZE OF DATABASE


ERROR CONDITIONS:
   IF THE VARIABLE STACK IS EMPTY, THEN THE EXPRESSION IS NOT IN
   IN-FIX NOTATION AND ERROR = NINFX (8) IS RETURNED.


SPECIAL CONDITIONS:
   *NONE*



SUBROUTINE PROOP(OPVAL, OPBND, OPSTK, VARSK, NUMWD, LENSK, LASOP,
                 ERROR, KAREA, KSIZE)


   PROCESS AN OPERATOR


OVERVIEW:
   PROOP PROCESSES AN OPERATOR BY SETTING UP A TREE NODE FOR THE
   OPERATOR (IF NECESSARY) AND MANIPU- LATING THE OPERATOR STACK AS
   NECESSARY, SETTING UP THE POLYNOMIAL TREE ACCORDINGLY.


METHOD OF USE:
   INTEGER OPVAL, OPBND, NUMWD, LENSK, ERROR, KSIZE
   INTEGER OPSTK(NUMWD, LENSK), VARSK(NUMWD, LENSK)
   LOGICAL LASOP
   DIMENSION KAREA(KSIZE)
   CALL PROOP(OPVAL, OPBND, OPSTK, VARSK, NUMWD, LENSK, LASOP, KAREA,
              KSIZE)


PARAMETERS:
   OPVAL (INPUT) - VALUE OF THE OPERATOR (HOLLERITH)
   OPBND (INPUT) - BINDING STRENGTH OF THE OPERATOR
   OPSTK (INPUT) - THE OPERATOR STACK
   VARSK (INPUT) - THE VARIABLE STACK
   NUMWD (INPUT) - NUMBER OF WORDS PER ENTRY IN THE STACK
   LENSK (INPUT) - LENGTH OF THE STACK
   LASOP (INPUT/OUTPUT) - .TRUE.  IF LAST SYMBOL WAS AN OPERATOR
          .FALSE.  IF IT WAS A VARIABLE OR A ')'
   ERROR (OUTPUT) - ERROR FLAG (SEE ERROR CONDITIONS)
   KAREA (INPUT) - DATABASE AREA
   KSIZE (INPUT) - SIZE OF DATABASE
```

ERROR CONDITIONS:
IF EITHER OF THE STACKS (OPERATOR OR VARIABLE) IS EMPTY, THE EXPRESSION IS NOT IN IN-FIX NOTATION AND ERROR = NINFX (8) IS RETURNED. IF UNMATCHED PARENTHESES ARE ENCOUNTERED, ERROR = UNPAR (7) IS RETURNED.

SPECIAL CONDITIONS:
NONE

SUBROUTINE PUSH(STACK, NUMWD, LENSK, INFO)

PUSH THE STACK

OVERVIEW:
PUSH PUTS NUMWD ITEMS OF DATA ON THE STACK AND RESETS THE STACK POINTER. THE STACK POINTER IS MAINTAINED IN THE FIRST ELEMENT OF THE STACK ARRAY. THIS ELEMENT SHOULD BE INITIALIZED TO 1 (STACK(1,1)=1) BEFORE PUSH OR POP ARE FIRST CALLED.

METHOD OF USE:
```
INTEGER NUMWD, LENSK
INTEGER STACK(NUMWD, LENSK), INFO(NUMWD)
CALL PUSH(STACK, NUMWD, LENSK, INFO)
```

PARAMETERS:
STACK (INPUT/OUTPUT) - 2 DIMENSIONAL ARRAY THAT CONTAINS THE STACK
NUMWD (INPUT) - THE NUMBER OF WORDS PER STACK ENTRY
LENSK (INPUT) - THE MAXIMUM LENGTH OF THE STACK
INFO (OUTPUT) - AN ARRAY OF N WORDS TO BE PUT ON THE STACK

ERROR CONDITIONS:
IF THE STACK IS FULL, EXECUTION IS TERMINATED BY THE ERROR ROUTINE.

SPECIAL CONDITIONS:
NONE

SUBROUTINE SAME(NUMER, DENOM, RATIO, TFLAG, KAREA, KSIZE)

ARE TWO POLYNOMIALS THE SAME?

OVERVIEW:
SAME COMPARES THE TWO POLYNOMIALS POINTED TO BY NUMER AND DENOM TO DETERMINE IF THEY ARE EQUAL OR DIFFER BY A CONSTANT RATIO. IF THEY DO DIFFER BY A CONSTANT RATIO, NUMER AND DENOM ARE RETURNED EMPTY, RATIO IS THE CONSTANT AND TFLAG IS CONTR. IF THEY ARE EQUAL, THE RATIO IS 1.0. IF THE DIFFERENCE IS MORE SUBSTANTIAL, THEN THE POINTERS ARE RETURNED UNCHANGED AND RATIO IS EMPTY.

METHOD OF USE:
```
INTEGER NUMER, DENOM, TFLAG, KSIZE
REAL RATIO
DIMENSION KAREA(KSIZE)
CALL SAME(NUMER, DENOM, RATIO, TFLAG, KAREA, KSIZE)
```

PARAMETERS:
NUMER (INPUT/OUTPUT) - POINTER TO LAST TERM OF NUMERATOR
DENOM (INPUT/OUTPUT) - POINTER TO LAST TERM OF DENOMINATOR
RATIO (OUTPUT) - CONSTANT RATIO
TFLAG (OUTPUT) - TYPE FLAG OF RESULT
KAREA (INPUT) - DATABASE AREA
KSIZE(INPUT) - SIZE OF DATABASE AREA

ERROR CONDITIONS:
NONE

SPECIAL CONDITIONS:
IF A CONSTANT RATIO IS FOUND, A TYPE FLAG OF CONTR IS RETURNED. IF THE CASE CANNOT BE HANDLED, NUMER AND DENOM ARE RETURNED EMPTY ALONG WITH A TYPE FLAG OF CNHTR.

SUBROUTINE TRAVEL(OPSTK, NUMWD, LENSK, STAT, ERROR, KAREA, KSIZE)

TRAVEL THE TREE

OVERVIEW:
TRAVEL TRAVERSES THE BINARY TREE OF THE POLYNOMIAL BY OBTAINING OPERATORS AND OPERANDS FOR A CURRENT SUBTREE, DETERMINING THE OPERATION WHICH NEEDS TO BE PERFORMED AND CALLING THE PROPER

ROUTINE WITH OPERANDS TO EVALUATE.

METHOD OF USE:

```
INTEGER OPSTK(NUMWD, LENSK), NUMWD, LENSK
INTEGER STAT, ERROR, KSIZE
DIMENSION KAREA(KSIZE)
CALL TRAVEL(OPSTK, NUMWD, LENSK, STAT, ERROR, KAREA, KSIZE)
```

PARAMETERS:

OPSTK (INPUT/OUTPUT) - THE OPERATOR STACK
NUMWD (INPUT) - NUMBER OF WORDS PER STACK ENTRY
LENSK (INPUT) - LENGTH OF THE STACK
STAT (OUTPUT) - STATUS OF THE REDUCED POLYNOMIAL (SEE SPECIAL CONDITION
ERROR (OUTPUT) - ERROR FLAG (SEE ERROR CONDITIONS)
KAREA (INPUT/OUTPUT) - DATABASE AREA
KSIZE (INPUT) - SIZE OF DATABASE AREA

ERROR CONDITIONS:

IF THE EXPRESSION IS NOT IN IN-FIX NOTATION, ERROR = NINFX (8) IS RETURNED. IF UNBALANCED PARENTHESES ARE ENCOUNTERED, ERROR = UNPAR (7) IS RETURNED.

SPECIAL CONDITIONS:

IF A MULTI-LEVEL OR FRACTIONAL EXPRESSION EXPONENT IS ENCOUNTERED, STAT = MFEXP (3) IS RETURNED. IF OVERLY-COMPLEX EXPONENTIATION IS ENCOUNTERED, STAT = OCEXP (4) IS RETURNED. IF THE POLYNOMIAL IS UNDEFINED DUE TO DIVISION BY ZERO, STAT = UNDEF (5) IS RETURNED.

INTEGER FUNCTION TRMCON(LFTRM, RITRM, KAREA, KSIZE)

POLYNOMIAL TERM ORDERING WITH CONSTANT EXPONENTS ONLY

OVERVIEW:

TRMCON DETERMINES THE ORDERING OF TWO MULTIVARIATE POLYNOMIAL TERMS. THE ORDERING IS FIRST DEPENDENT ON THE DEGREE OF THE TERMS, WITH TERMS OF HIGHER DEGREE PRECEDING THOSE OF LOWER DEGREE, AND SECONDLY ON THE VARIABLES IN EACH POLYNOMIAL, JUST LIKE THE NORMAL ORDERING OF TERMS IN A SUM. TRMCON RETURNS

1 - IF LFTRM EQUALS RITRM UP TO THE COEFFICIENTS
2 - IF LFTRM SHOULD PRECEDE RITRM
3 - IF LFTRM SHOULD FOLLOW RITRM
4 - IF THE CASE IS NOT HANDLED (SEE SPECIAL CONDITIONS)

METHOD OF USE:

```
INTEGER LFTRM, RITRM, KSIZE
DIMENSION KAREA(KSIZE)
INTEGER TRMCON
----- = TRMCON(LFTRM, RITRM, KAREA, KSIZE)
```

PARAMETERS:

LFTRM (INPUT) - POINTER TO LEFT POLYNOMIAL TERM
RITRM (INPUT) - POINTER TO RIGHT POLYNOMIAL TERM
KAREA (INPUT) - DATABASE AREA
KSIZE (INPUT) - SIZE OF THE DATABASE

ERROR CONDITIONS:

NONE

SPECIAL CONDITIONS:

TRMCON DOES NOT HANDLE EXPRESSION EXPONENTS FOR THE TERMS WHICH IT IS ORDERING. IF AN EXPRESSION EXPONENT IS ENCOUNTERED, TRMCON RETURNS WITH VALUE 4. THIS RESTRICTION IS DUE TO THE LACK OF RECURSION IN FORTRAN, SINCE THE ORDER OF THOSE EXPRESSION EXPONENTS WOULD HAVE TO BE DETERMINED AS WELL. SINCE THIS ROUTINE IS ONLY CALLED FOR EXPONENTS WITHIN SYMPLR, EXPRESSION EXPONENTS ARE ALLOWED TO ONE LEVEL OF NESTING.

INTEGER FUNCTION TRMEXP(LFTRM, RITRM, KAREA, KSIZE)

POLYNOMIAL TERM ORDERING WITH EXPRESSION EXPONENTS

OVERVIEW:

TRMEXP DETERMINES THE ORDERING OF TWO MULTIVARIATE POLYNOMIAL TERMS. THE ORDERING IS FIRST DEPENDENT ON THE DEGREE OF THE TERMS, WITH TERMS OF HIGHER DEGREE PRECEDING THOSE OF LOWER DEGREE, AND SECONDLY ON THE VARIABLES IN EACH POLYNOMIAL, JUST LIKE THE NORMAL ORDERING OF TERMS IN A SUM. TRMEXP RETURNS

1 - IF LFTRM EQUALS RITRM UP TO THE COEFFICIENTS
2 - IF LFTRM SHOULD PRECEDE RITRM
3 - IF LFTRM SHOULD FOLLOW RITRM
4 - IF THE CASE IS NOT HANDLED (SEE SPECIAL CONDITIONS)

METHOD OF USE:

```
INTEGER LFTRM, RITRM, KSIZE
DIMENSION KAREA(KSIZE)
INTEGER TRMEXP
----- = TRMEXP(LFTRM, RITRM, KAREA, KSIZE)
```

PARAMETERS:
LFTRM (INPUT) - POINTER TO LEFT POLYNOMIAL TERM
RITRM (INPUT) - POINTER TO RIGHT POLYNOMIAL TERM
KAREA (INPUT) - DATABASE AREA
KSIZE (INPUT) - SIZE OF THE DATABASE

ERROR CONDITIONS:
NONE

SPECIAL CONDITIONS:
IF AN EXPRESSION EXPONENT NESTED TOO DEEP IS ENCOUNTERED, THIS CASE CANNOT BE HANDLED AND TRMEXP IS RETURNED WITH VALUE 4.

SUBROUTINE TRYLNR(NUMER, DENOM, LINDR, STAT, KAREA, KSIZE)

TRY TO LINEARIZE A POLYNOMIAL

OVERVIEW:
TRYLNR TRIES TO MAKE A POLYNOMIAL LINEAR BY TAKING ADVANTAGE OF THE WAY IN WHICH THESE POLYNOMIALS WILL BE USED IN LINDR. THIS ROUTINE SHOULD BE CALLED WITH LINDR=FALSE WHEN THIS SYSTEM IS USED FOR OTHER PURPOSES. TRYLNR IS CALLED WHEN THE POLYNOMIAL DID NOT SIMPLIFY TO A LINEAR POLYNOMIAL. TRYLNR ATTEMPTS TO DIVIDE THE DENOMINATOR BY THE NUMERATOR. IF THE DIVISION IS SUC- CESSFUL, THEN THE POLYNOMIAL IS NOW REDUCED TO THE CON- STANT 1.0 DIVIDED BY THE VALUE RESULTING FROM THE DIVI- SION. SINCE ATTEST ALWAYS WORKS WITH THESE POLYNOMIALS AS INEQUALITIES RELATED TO ZERO, WE CAN MULTIPLY BOTH SIDES OF THE INEQUALITY BY THE SQUARE OF THE DENOMINATOR AND STILL HAVE A VALID AND EQUIVALENT INEQUALITY. THIS WILL HELP ONLY IF THE NEW DENOMINATOR IS LINEAR, AND THIS IS THE ONLY CASE IN WHICH IT IS DONE. IF THIS CANNOT BE DONE (OR LINDR=FALSE), TRYLNR WILL RETURN THE RESULT OF THE DIVISION, PROVIDED THAT WAS SUCCESSFUL, OTHERWISE IT WILL RETURN THE POLYNOMIAL UNCHANGED (IF DIVISION DIDN'T WORK).

METHOD OF USE:
INTEGER NUMER, DENOM, STAT, KSIZE
LOGICAL LINDR
DIMENSION KAREA(KSIZE)
CALL TRYLNR(NUMER, DENOM, LINDR, STAT, KAREA, KSIZE)

PARAMETERS:

NUMER (INPUT/OUTPUT) - POINTER TO NUMERATOR, DIVIDEND
DENOM (INPUT/OUTPUT) - POINTER TO DENOMINATOR, DIVISOR
LINDR (INPUT) - LOGICAL FLAG WHICH SIGNALS WHETHER LINEARIZATION
SHOULD BE PERFORMED (ATTEST DEPENDENT)
STAT (OUTPUT) - STATUS OF REDUCED POLYNOMIAL
KAREA (INPUT/OUTPUT) - DATABASE AREA
KSIZE (INPUT) - SIZE OF DATABASE AREA

ERROR CONDITIONS:
NONE

SPECIAL CONDITIONS:
THIS IS A SPECIAL PURPOSE ROUTINE FOR USE IN ATTEST. LINDR SHOULD BE FALSE WHEN THIS SYSTEM IS USED MERELY AS A POLYNOMIAL SIMPLIFIER. THE INVERSE DIVISION (I. E., DENOMINATOR DIVIDED BY NUMERATOR) MAY BE DONE AS IT DOES SIMPLIFY THE POLYNOMIAL. THE MULTIPLICATION OF THE POLYNOMIAL BY THE SQUARE OF THE DENOMINATOR (INVERTING THE POLYNOMIAL), HOWEVER, SHOULD BE SKIPPED.

SUBROUTINE VALCHR(EXPOS, EXPRS, EXDIM, VALUE, CHRIX, ERROR)

CONVERT VALUE TO CHARACTERS

OVERVIEW:
VALCHR CHANGES A VALUE INTO A CHARACTER AND STORES THE CHARACTER STRING IN EXPRS BEGINNING AT THE POSITIONED INDICATED BY I.

METHOD OF USE:
```
INTEGER EXPOS, EXDIM
INTEGER EXPRS(EXDIM)
LOGICAL CHRIX
INTEGER ERROR
REAL VALUE
CALL VALCHR(EXPOS, EXPRS, EXDIM, VALUE, CHRIX, ERROR)
```

PARAMETERS:
EXPOS (INPUT/OUTPUT) - POSITION IN EXPRS TO PUT CHARACTER STRING
EXPRS (INPUT/OUTPUT) - EXPRESSION ARRAY IN WHICH TO PUT CHARACTER STRING
EXDIM (INPUT) - DIMENSION OF EXPRESSION ARRAY
VALUE (INPUT) - VALUE TO CONVERT TO CHARACTERSE
CHRIX (INPUT) - .TRUE. IMPLIES INTEGER, .FALSE. IMPLIES REAL
ERROR (OUTPUT) - ERROR FLAG (SEE ERROR CONDITIONS)

ERROR CONDITIONS:
IF THE EXPRS ARRAY IS OVERFLOWED, ERROR = OVFLW (1) IS RETURNED.

SPECIAL CONDITIONS:
NONE

SUBROUTINE VARIX(IORX, EXPOS, EXPRS, EXDIM, IBND, XBND, VARSK, NUMWD, LENSK, ERROR, KAREA, KSIZE)

VARIABLE INTEGER (I) OR REAL (X)

OVERVIEW:
VARIX PROCESSES A REAL OR INTEGER VARIABLE (I OR X) WHICH IS ENCOUNTERED DURING INPUT. VARIX SETS UP A NODE IN THE DATABASE ALONG WITH A BIT VECTOR INDICATING THE VARIABLE WHICH OCCURED; IF IORX IS TRUE, THEN THE VARIABLE IS AN INTEGER, OTHERWISE IT IS A REAL VARIABLE.

METHOD OF USE:
```
INTEGER EXPOS, EXDIM
INTEGER EXPRS(EXDIM)
INTEGER IBND, XBND, ERROR, KSIZE
VARSK(NUMWD, LENSK), NUMWD, LENSK
LOGICAL IORX
DIMENSION KAREA(KSIZE)
CALL VARIX(IORX, EXPOS, EXPRS, EXDIM, IBND, XBND, VARSK, NUMWD,
           LENSK, ERROR, KAREA, KSIZE)
```

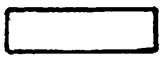
PARAMETERS:
IORX (INPUT) - INDICATES WHETHER THE VARIABLE IS REAL OR INTEGER
.TRUE. IMPLIES INTEGER, .FALSE. IMPLIES REAL
EXPOS (INPUT/OUTPUT) - POSITION IN THE EXPRS ARRAY OF THE VARIABLE / POSITION AFTER THE VARIABLE
EXPRS (INPUT) - EXPRESSION ARRAY
EXDIM (INPUT) - DIMENSION OF EXPRS
IBND (INPUT) - TOTAL NUMBER OF INTEGER VARIABLES
XBND (INPUT) - TOTAL NUMBER OF REAL VARIABLES
VARSK (INPUT) - VARIABLE STACK
NUMWD (INPUT) - NUMBER OF WORDS PER ENTRY IN THE STACK
LENSK (INPUT) - LENGTH OF THE STACK
ERROR (OUTPUT) - ERROR FLAG (SEE ERROR CONDITIONS)
KAREA (INPUT/OUTPUT) - DATA BASE AREA
KSIZE (INPUT) - SIZE OF THE DATABASE

ERROR CONDITIONS:
IF AN INTEGER VARIABLE SUBSCRIPTED BEYOND THE BOUND, IBND, IS ENCOUNTERED, ERROR = INTBB (4) IS RETURNED. IF A REAL VARIABLE SUBSCRIPTED BEYOND THE BOUND, XBND, IS ENCOUNTERED, ERROR = RELBB (5) IS RETURNED.

SPECIAL CONDITIONS:
NONE

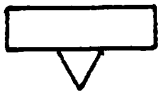
Appendix B. SYMPLR Call Graph

To follow is a call graph for the SYMPLR system (where the top level is the SYMPLR subroutine). The graph includes all calls to SYMPLR program modules (those described in Appendix A), but does not include calls to FORTRAN intrinsic routines or database manipulation routines. Shown below is a legend to the call graph symbols.

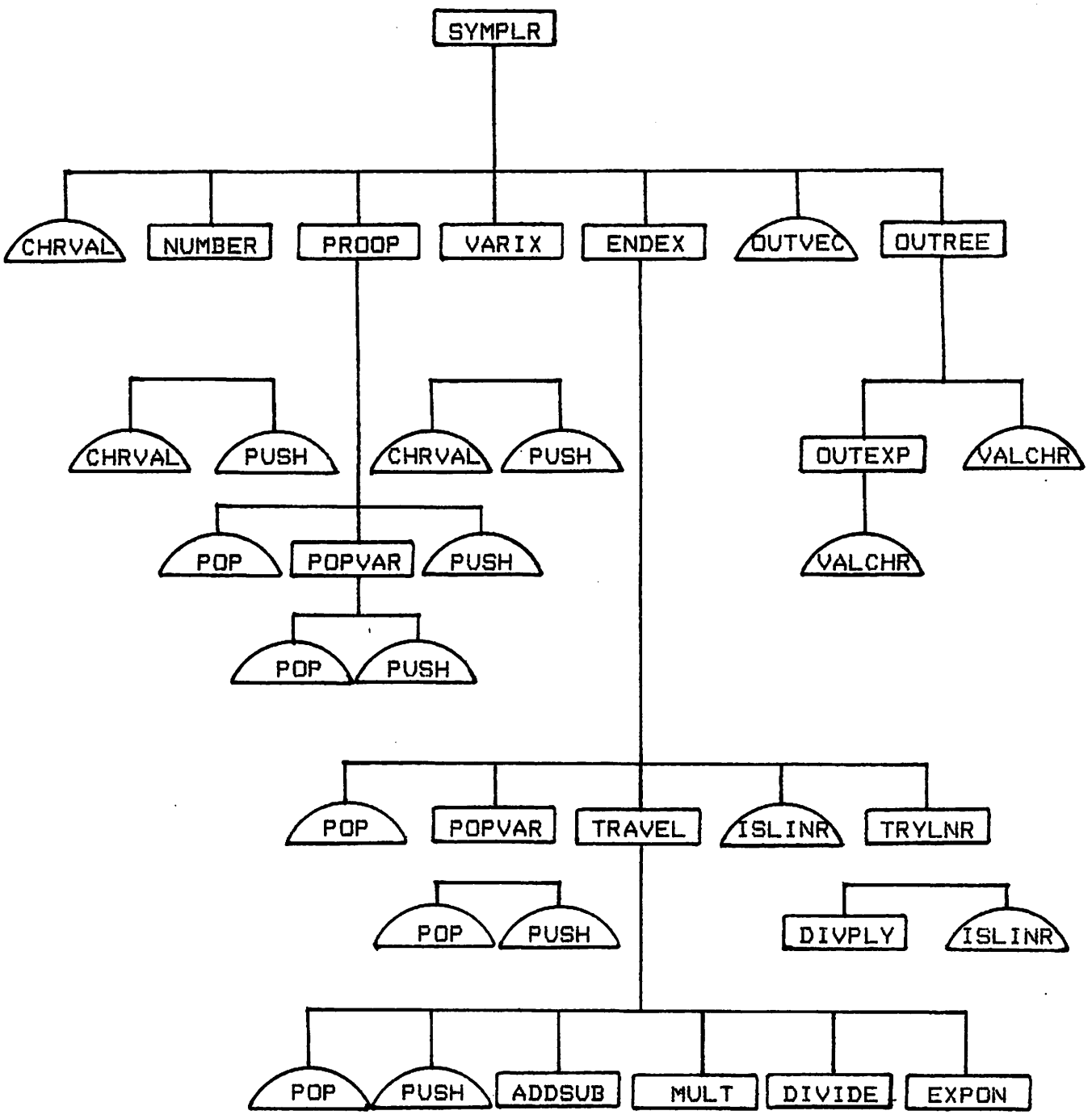


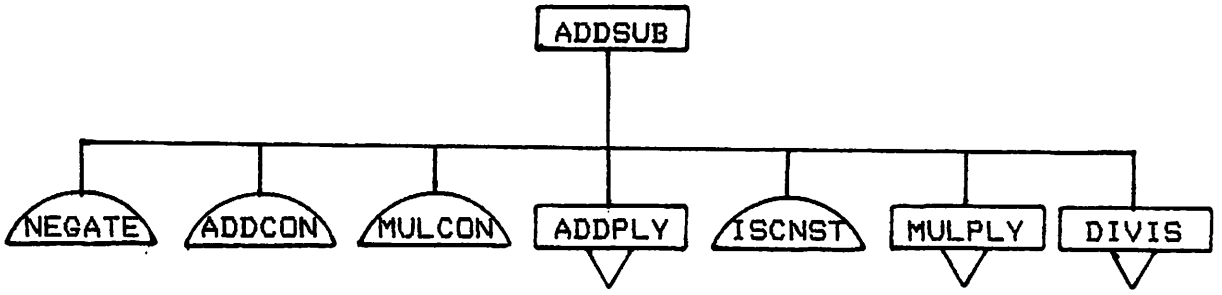
non-leaf module

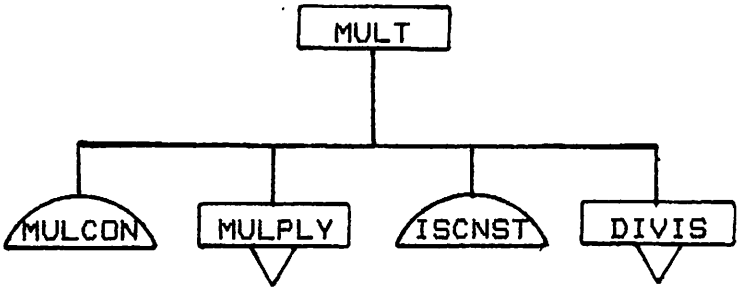
leaf module

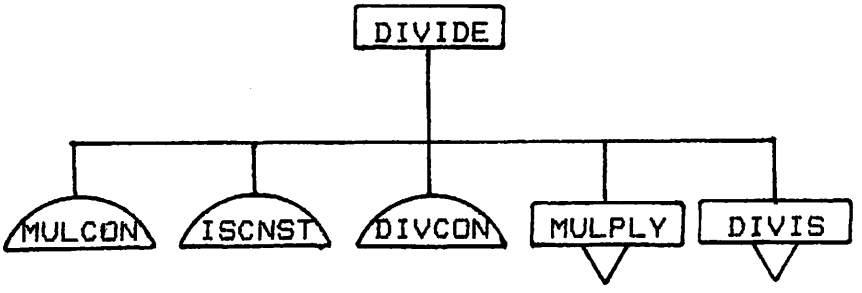


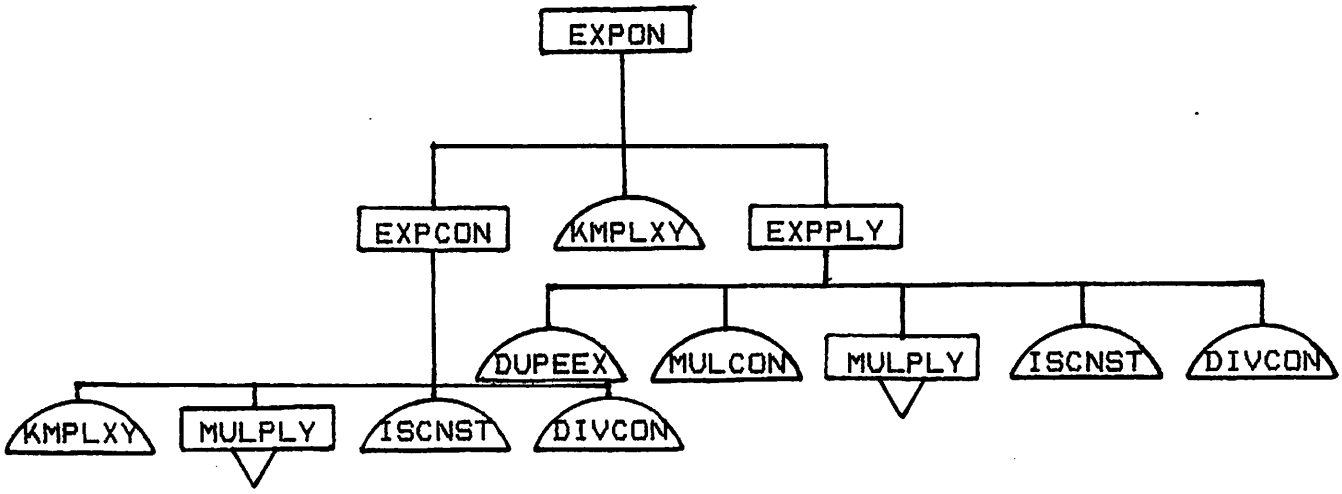
connector

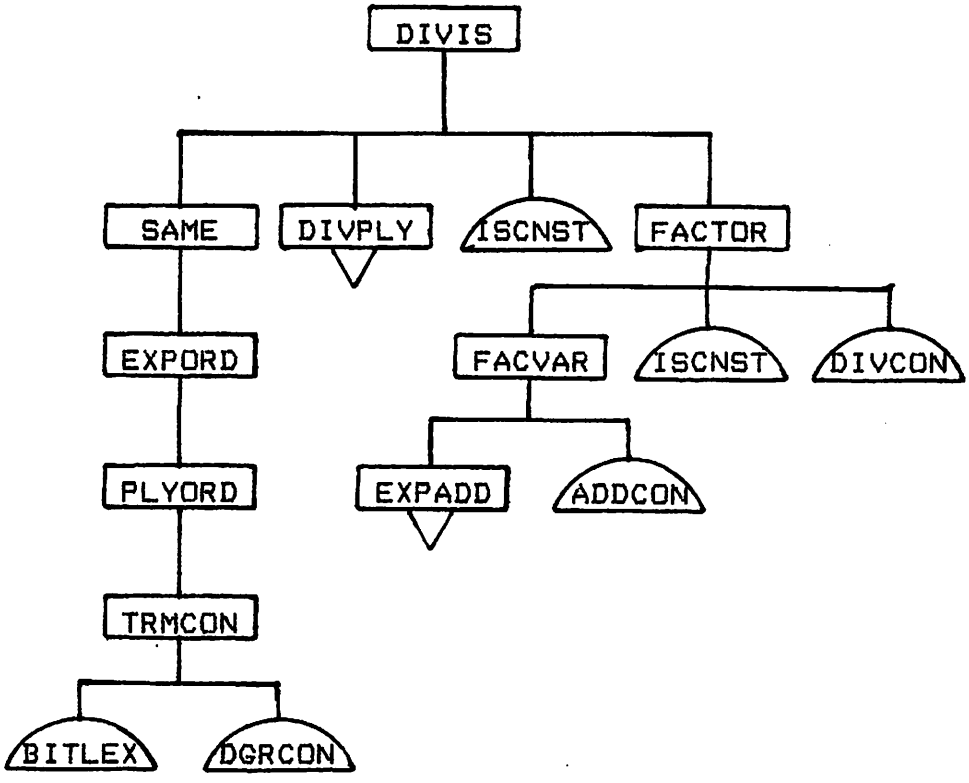

SYMPLR
CHRVAL
NUMBER
PROOP
VARIX
ENDEX
OUTVEC
OUTREE
CHRVAL
PUSH
CHRVAL
PUSH
OUTEXP
VALCHR
POP
POPVAR
PUSH
VALCHR
POP
PUSH
POP
POPVAR
TRAVEL
ISLINR
TRYLNR
POP
PUSH
DIVPLY
ISLINR
POP
PUSH
ADDSUB
MULT
DIVIDE
EXPON

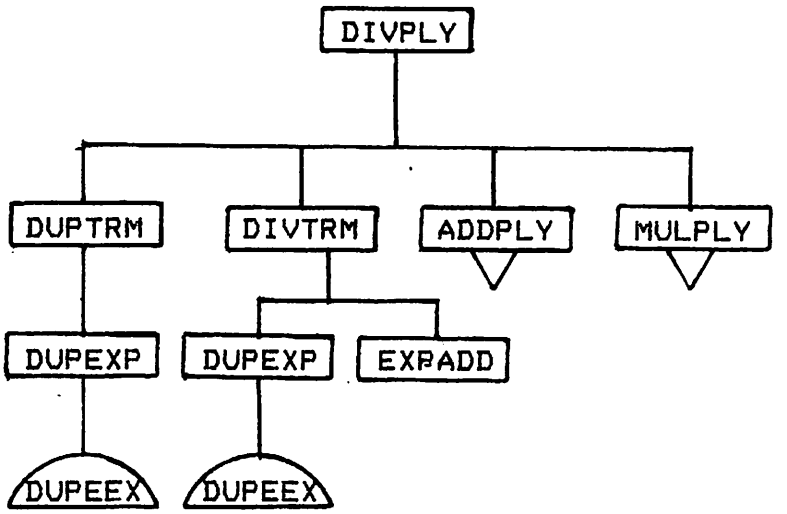

ADDSUB
NEGATE
ADDCON
MULCON
ADDPLY
ISCNST
MULPLY
DIVIS

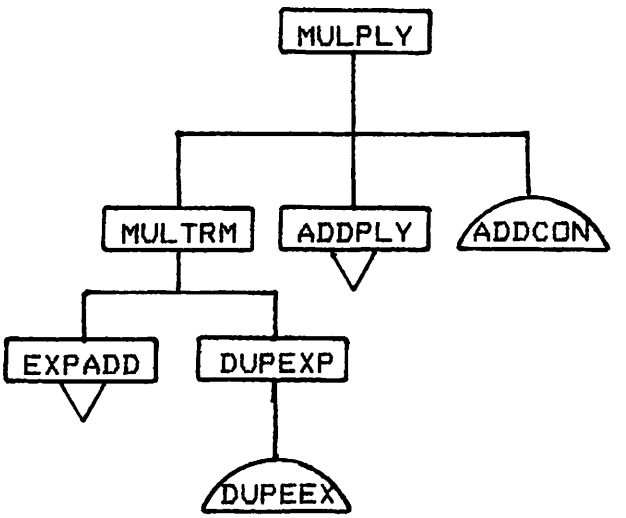

MULT
MULCON
MULPLY
ISCNST
DIVIS

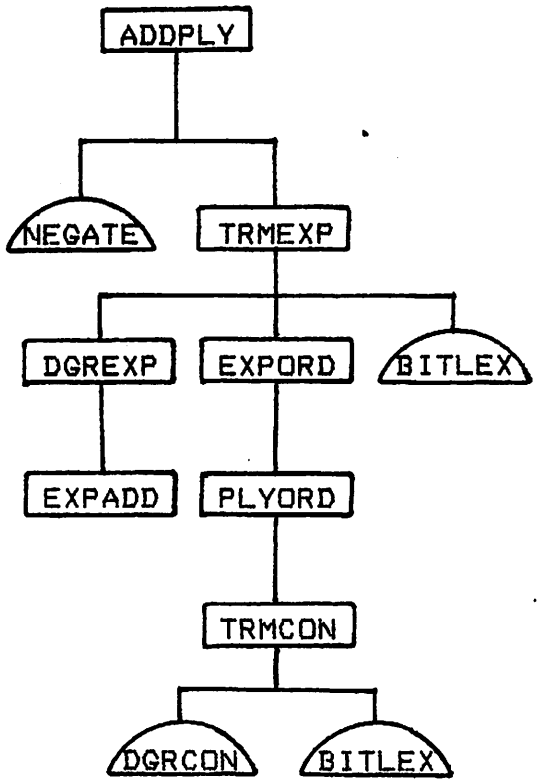

DIVIDE
MULCON
ISCNST
DIVCON
MULPLY
DIVIS

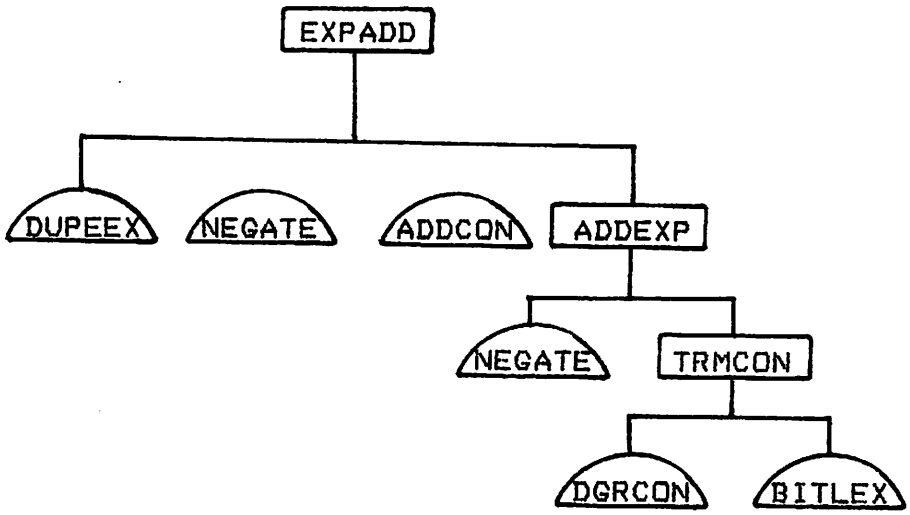

EXPON
EXPCON
KMPLXY
EXPPLY
DUPEEX
MULCON
MULPLY
ISCNST
DIVCON
KMPLXY
MULPLY
ISCNST
DIVCON


DIVIS
SAME
DIVPLY
ISCNST
FACTOR
EXPORD
FACVAR
ISCNST
DIVCON
PLYORD
EXPADD
ADDCON
TRMCON
BITLEX
DGRCON


DIVPLY
DUPTRM
DIVTRM
ADDPLY
MULPLY
DUPEXP
DUPEXP
EXPADD
DUPEEX
DUPEEX


MULPLY
MULTRM
ADDPLY
ADDCON
EXPADD
DUPEXP
DUPEEX


ADDPLY
NEGATE
TRMEXP
DGREXP
EXPORD
BITLEX
EXPADD
PLYORD
TRMCON
DGRCON
BITLEX


EXPADD
DUPEEX
NEGATE
ADDCON
ADDEXP
NEGATE
TRMCON
DGRCON
BITLEX

Appendix C. SYMPLR Global Variables

SYMPLR requires certain global variables, which must be declared and initialized in any program that invokes SYMPLR. Section C.1 provides definitions of the COMMON blocks and their variables, along with descriptions of those variables. Each of these COMMON blocks must be declared in the calling program. Section C.2 specifies the initializations that must be performed, by providing the form of the BLOCK DATA section required by SYMPLR. These statements could be incorporated into an existing BLOCK DATA section associated with a main program using SYMPLR. On the other hand, these initializations could also be made by assignment statements in the calling program.

C.1. COMMON Blocks and Variables

INPOUT - INPOUT contains information on input and output files that are necessary for the database system.

COMMON /INPOUT/ IREADR, IPRINT, IPUNCH

IREADR - logical unit number of reader (not used)
IPRINT - logical unit number of printer (used in error routine)
IPUNCH - logical unit number of card punch (not used)

NODETS - NODETS consists of the name of all sequential and linked-list nodes of the SYMPLR database.

COMMON /NODETS/ IBTSTB, KBTREE, KEXPRS, KEXPON

IBTSTB - sequential table containing the bit vectors
KBTREE - linked-list node for a subexpression in the binary tree
KEXPRS - linked-list node for a term of an expression list
KEXPON - linked-list node for an exponent of an exponent list

FIELDS - FIELDS consists of the names of all fields of the SYMPLR database nodes.

COMMON /FIELDS/ IDUMMY, KFLAGS, KNUMER, KDENOM, KCONST, KOPVAL, KLINKL, KLINKR, KBITVC, KCOEFF, KPNTEX, KNXTRM, KEXFLG, KEXVAL, KEXNUM, KNXEXP

IDUMMY - dummy field required by the database system for the bit vector table
KLINKL - pointer to the left descendant of this tree node
KLINKR - pointer to the right descendant of this tree node
KNUMER - pointer to the numerator for this tree node
KDENOM - pointer to the denominator of this tree node
KOPVAL - holerith value of the operator represented by this tree node
KCONST - real constant value of this tree node
KFLAGS - type flag of this tree node
KBITVC - pointer to the bit vector for this term
KCOEFF - real coefficient value for this term
KPNTEX - pointer to the exponent list for this term
KNXTRM - pointer to the next term in this expression list
KEXFLG - type of this exponent
KEXVAL - real value of this exponent
KEXNUM - pointer to numerator for this exponent
KNXEXP - pointer to the next exponent in this exponent list

DBPAGE - DBPAGE contains the information that is necessary for the database paging and is initialized by the database initialization routine.

```
COMMON /DBPAGE/ IPAGSZ, MAXPGS, NDBFLS, MAXFLS, IFILINF(5,3)
```

IPAGSZ - size (number of words) in a page of the database
MAXPGS - maximum number of pages in the database
NDBFLS - number of database files currently open
MAXFLS - maximum number of database files
IFLINF - database file information

NOTEST - NOTEST contains the information necessary for database testing.

```
COMMON /NOTEST/ NOTEST
```

NOTEST - logical switch indicating whether or not to perform full testing for the database manipulation routines

DEBUG - DEBUG contains the information necessary for the debugging (developmental).

```
COMMON /DEBUG/ DEBUGS, BUGON
```

DEBUGS - file for output of debugging information
BUGON - logical switch indicating whether or not to output debugging information

MACHIN - MACHIN contains all variables that specify machine dependent information.

```
COMMON /MACHIN/ NSIGDD
```

NSIGDD - number of significant decimal digits in a floating point word of the machine in use

TREE - TREE consists of the information concerning the binary tree.

```
COMMON /TREE/ ROOT
```

ROOT - pointer to the root of the subtree currently being traversed

PAREN - PAREN makes sure the input expression has balanced parentheses.

COMMON /PAREN/ KPAR

KPAR - count of the parentheses in an expression where a left parenthesis causes an increment of one and a right parenthesis causes a decrement of one; if this count is ever negative, then the parentheses are unbalanced

BTVECS - BTVECS consists of all information concerning the bit vectors.

COMMON /BTVECS/ BVSIZ, BVTMP

BVSIZ - size of all bit vectors(maximum number of variables)
BVTMP - bit vector for temporary usage

OPERS - OPERS contains the hollerith values of operators used in the SYMPLR system.

COMMON /OPERS/ PLUS, MINUS, STAR, SLASH, STSTR, LPAR, RPAR

PLUS - hollerith value of a plus "+" (addition)
MINUS - hollerith value of a minus "-" (subtraction)
STAR - hollerith value of an asterisk "*" (multiplication)
SLASH - hollerith value of a slash "/" (division)
STSTR - hollerith value of two asterisks "**" (exponentiation)
LPAR - hollerith value of a left parenthesis "("
RPAR - hollerith value of a right parenthesis ")"

STATUS - STATUS contains the values of the STATUS codes returned from SYMPLR.

COMMON /STATUS/ LNEAR, LINZD, NONLN, MFEXP, OCEXP, UNDEF, INERR

LNEAR - reduced polynomial is linear
LINZD - reduced polynomial is "linearized"
NONLN - reduced polynomial is non-linear
MFEXP - invalid expression exponentiation encountered
OCEXP - overly-complex exponentiation encountered
UNDEF - reduced polynomial is undefined
INERR - error encountered in input (check ERROR value)

ERROR - ERROR contains the values of the ERROR flags returned from SYMPLR.

COMMON /ERROR/ NOERR, OVFLW, LGTRD, INVSB, INTBB, RELBB, ILCHR, UNPAR, NINFX

NOERR - no error encountered in input
OVFLW - EXPRS array overflowed
LGTRD - length of input expression greater than dimension of EXPRS array
INVBB - invalid subscript bounds
INTBB - integer variable subscripted beyond bound encountered
RELBB - real variable subscripted beyond bound encountered
ILCHR - illegal character encountered
UNPAR - unbalanced parenthesis encountered
NINFX - input expression not in in-fix notation

FLAGS - FLAGS contains the values of the flags stored in the SYMPLR database.

COMMON /FLAGS/ EXPTR, OPRTR, CONTR, BLKTR, UNDTR, CNHTR, CONEX, EXPEX

EXPTR - expression tree node flag
OPRTR - operator tree node flag
CONTR - constant tree node flag
BLKTR - blocked tree node flag
UNDTR - undefined tree node flag
CNHTR - case-not-handled node flag
CONEX - constant exponent node flag
EXPEX - expression exponent node flag

C.2. BLOCK DATA

```
BLOCK DATA

COMMON /INPOUT/ IREADR, IPRINT, IPUNCH

COMMON /NOTEST/ NOTEST

COMMON /DEBUG/ DEBUGS, BUGON

COMMON /MACHIN/ NSIGDD

COMMON /OPERS/ PLUS, MINUS, STAR, SLASH, STSTR, LPAR, RPAR

COMMON /STATUS/ LNEAR, LINZD, NONLN, MFEXP, OCEXP, UNDEF, INERR

COMMON /ERROR/ NOERR, OVFLW, LGTRD, INVSB, INTBB, RELBB, ILCHR,
       UNPAR, NINFX

COMMON /FLAGS/ EXPTR, OPRTR, CONTR, BLKTR, UNDTR, CNHTR, CONEX,
       EXPEX

DATA IREADR, IPRINT, IPUNCH /?, ?, ?/
DATA NOTEST /.FALSE./

DATA DEBUGS, BUGON /?, ?/

DATA NSIGDD /?/

DATA PLUS, MINUS, STAR, SLASH, STSTR, LPAR, RPAR
       /1H+, 1H-, 1H*, 1H/, 2H**, 1H(, 1H)/

DATA LNEAR, LINZD, NONLN, MFEXP, OCEXP, UNDEF, INERR
       /0, 1, 2, 3, 4, 5, 6/

DATA NOERR, OVFLW, LGTRD, INVSB, INTBB, RELBB, ILCHR, UNPAR, NINFX
       /0, 1, 2, 3, 4, 5, 6, 7, 8/

DATA EXPTR, OPRTR, CONTR, BLKTR, UNDTR, CNHTR, CONEX, EXPEX
       /0, 1, 2, 3, 4, 5, 0, 1/


END
```

NOTE: The question marks (?) specified for the values of IREADR, IPRINT, IPUNCH, DEBUGS, BUGON and NSIGDD indicate that these two variables are implementation dependent and should be chosen by the user. NSIGDD is the number of significant digits in a floating point number for the machine on which SYMPLR is being run. IREADR, IPRINT, IPUNCH and DEBUGS are the unit numbers of the files, and must be associated with logical devices (see description of files in appendix D). BUGON specifies whether tracing and other debugging information should be output (if BUGON is .FALSE., the file DEBUGS will not be used). IREADR and IPUNCH are never used by SYMPLR, but must be included in COMMON for the database system.

Appendix D. SYMPLR files

Described below are the files that are necessary for the operation of SYMPLR. Some of the files are not necessary for use of the SYMPLR subroutine, while others are not used in certain circumstances. The conditions under which each file need not be available are specified. The required files must be set up with the appropriate operating system commands before SYMPLR is run. The unit numbers indicated for each file below are those that the provided SYMPLR main program assumes; hence, the files must be associated with these logical units if the SYMPLR main program is used. If SYMPLR is invoked from a program other than the mainline provided, the logical unit numbers of the files that the SYMPLR subroutine accesses may be chosen as desired, and these must be initialized in the calling program.

FILE NAME	UNIT #	FORMAT	CONTENTS
COMDAT	1	unformatted 8 records with maximum of 812 words per record	input file for database initialization information for INITRN (see 3.3.1)
INPEXP	2	formatted maximum of 80 characters per record	input file for input expressions to SYMPLR main program (specifications in 3.4.1)
not required for SYMPLR subroutine			
REDPLY	3	formatted maximum of 80 characters per record	output file for reduced polynomials from SYMPLR main program (specifications in 3.4.2)
not required for SYMPLR subroutine			

FILE NAME	UNIT #	FORMAT	CONTENTS
TEMPI	4	formatted maximum of 80 characters per record	input/output file for temporary storage of an input expression by SYMPLR main program
not required for SYMPLR subroutine			
DEBUGS	6	formatted maximum of 80 characters per record	output file for tracing and other debugging information
used only when BUGON=.TRUE. (see appendix C)			
IPRINT	6	formatted maximum of 80 characters per record	output file for run-time errors in the database management system
DBUTIL	6	formatted 1 record with 72 characters	output file for database utilization statistics (see 3.3.2)
DATBAS	27	random access maximum of 100 records with maximum of 512 words per record	input/output file for temporary storage of the database pages opened by OPNDBF (see 3.3.1)

Appendix E. Examples of Simplifications

To follow is a sample run of the SYMPLR mainline. It provides examples of the simplifications which are performed as well as producing each possible status which can occur as the result of erroneous input data. Section E.1 shows the file which was input to SYMPLR and E.2 shows the corresponding output file.

E.1. Input File to SYMPLR

```
C
C  REQUEST BOTH TYPES OF OUTPUT
C
O  2
C
C  PRODUCE ERROR = INVSB - NO SUBSCRIPT BOUNDS SET YET
C  ****EXAMPLE 1****
C
(-I4 + 2*I3 + 3*I4 +1+1+1) * 4.2 $
C
C  SET SUBSCRIPT BOUNDS, FIVE INTEGER AND FIVE REAL VARIABLES
C
S  5  5
C
C  LINEAR EXPRESSION - REORDERING OF TERMS NECESSARY
C  ****EXAMPLE 2****
C
5.0*X4 - 12.5*I5 + 14*X2 - I3 - 189.6$
C
C  PRODUCE END OF EXPRESSION NOT ENCOUNTERED
C  ****EXAMPLE 3****
C
X1 + I1
C
C  EXAMPLES OF MULTIPLICATION
C
C  SIMPLE TERM MULTIPLICATION
C  ****EXAMPLE 4****
C
(3*X1*I1*X2+X1)*(X2*X3+2*I1*X1)$
C
C  CHANGE OUTPUT DIRECTIVE
C  REQUEST ALPHANUMERIC REPRESENTATION ONLY
C
O  1
C
C  ****EXAMPLE 5****
C
5.5*X1+(I1+I2*X1)*(I2*X3*X2)$
C
C  MULTIPLICATION AND COMBINATION (CANCELLATION) OF LIKE TERMS
C  ****EXAMPLE 6****
C
(X1-1)*(X1**7 + X1**6 + X1**5 +
X1**4 + X1**3 + X1**2 + X1**1 + 1) $
C
C  PRODUCE ERROR = ILCHR - ILLEGAL CHARACTER
C  ****EXAMPLE 7****
C
I1 + COS(X1)$
```

```
C
C  CHANGE SUBSCRIPT DIRECTIVE
C
S  6  12
C
C  PRODUCE ERROR = RELBB - REAL VARIABLE SUBSCRIPTED BEYOND BOUND
C  ****EXAMPLE 8****
C
X13 + 4.0*I1$
C
C  EXAMPLES OF DIVISION
C
C  FRACTIONAL POLYNOMIALS - COMMON DENOMINATOR NECESSARY
C  ****EXAMPLE 9****
C
(3/(X5 + I6)) + (4/(I3 - X12))$
C
C  ****EXAMPLE 10****
C
(1/(X1 - X2)) - (1/(X2 - X1)) + X1 + X2$
C
C  CHANGE OUTPUT DIRECTIVE
C  REQUEST BOTH TYPES OF OUTPUT
O  2
C
C  SIMPLE FACTORING OF A SINGLE VARIABLE
C  ****EXAMPLE 11****
C
(X1 * X4 + X1) / X1$
C
C  ****EXAMPLE 12****
C
5.5*X1+(I1+I2*X1)*(I2*X3*X2) /)L2I2 $
C
C  SIMPLE DIVISION AND FACTORING OF A SINGLE VARIABLE
C  ****EXAMPLE 13****
C
(I1*X1**2 + 2*I1*X1 + I1) / (I1*(X1 + 1))$
C
C  UNARY SUBTRACTION AND SIMPLE DIVISION
C  (EXAMPLE USED IN APPENDIX F)
C  ****EXAMPLE 14****
C
(-1 + X1**2)/(X1-1)$
C
C  REDUCTION TO A CONSTANT
C  ****EXAMPLE 15****
C
(X1**2-1)/(X1+1)/(X1-1)$
C
C  ATTEMPT TO CHANGE SUBSCRIPT DIRECTIVE
C  PRODUCE ERROR = INVSB - SUM OF TWO BOUNDS GREATER THAN LIMIT
C
```

```
S  50  51
C
C  PRODUCE ERROR = INTBB - INTEGER VARIABLE OUT OF BOUNDS
C  (OLD SUBSCRIPTS STILL IN EFFECT)
C  ****EXAMPLE 16****
C
I23 + X4 + X3$
C
C  EXAMPLES OF EXPONENTIATION
C  POSITIVE EXPONENTIATION OF NUMERATOR
C  ****EXAMPLE 17****
C
((I4 + 1) ** 3)$
C
C  PRODUCE STATUS = UNPAR - UNMATCHED PARENTHESES
C  ****EXAMPLE 18****
C
((X3 + X2) ** 4$
C
C  NEGATIVE EXPONENTIATION OF DENOMINATOR
C  ****EXAMPLE 19****
C
 1 / (X1+1) ** (-2) $
C
C  NEGATIVE EXPONENTIATION OF NUMERATOR
C  ****EXAMPLE 20****
C
 (X1+1) ** (-1) $
C
C  ZERO EXPONENTIATION
C  ****EXAMPLE 21****
C
(X1 + 2) ** 0 $
C
C  SIMPLIFICATION OF EXPONENT BEFORE EXPONENTIATION PERFORMED
C  ****EXAMPLE 22****
C
(X1+1)**(((((X1+1)*(X1+1))-
((X1+1)*(X1-1))-2)/X1)$
C
C  PRODUCE STATUS = OCEXP - EXPONENTIATION TOO COMPLEX
C  BASIS POLYNOMIAL HAS TOO MANY TERMS
C  ****EXAMPLE 23****
C
((I1 + I2 + I3 + I4 + I5 + I6) ** 3.0)$
C
C  EXPONENT TOO LARGE
C  ****EXAMPLE 24****
C
(I1 + I2) ** 8$
C
C  SET LINEARIZE DIRECTIVE TO TRUE
C  EXAMPLES OF APPLICATION OF SPECIAL LINEARIZATION TRANSFORMATION
```

```
C
L  T
C
C  ****EXAMPLE 25****
C
 X1 / (X1 * X4 + X1) $
C
C  ****EXAMPLE 26****
C
(X1 + 1) / (I1 + I2 + I3 + X1*I1 + X1*I2 + X1*I3)$
C
C  ****EXAMPLE 27****
C
(X1 + 1) / (X1**2 + 2*X1 + 1) $
C
C  ****EXAMPLE 28****
C
(X1 + 1) / (I1*X1**2 + 2*I1*X1 + I1) $
C
C  ****EXAMPLE 29****
C
(I1*(X1 + 1)) / (I1*X1**2 + 2*I1*X1 +I1) $
C
C  SET LINEARIZE DIRECTIVE TO FALSE
C  SAME EXAMPLES WITHOUT APPLICATION OF LINEARIZATION
C
L  F
C
C  ****EXAMPLE 30****
C
 X1 / (X1 * X4 + X1) $
C
C  ****EXAMPLE 31****
C
(X1 + 1) / (I1 + I2 + I3 + X1*I1 + X1*I2 + X1*I3)$
C
C  ****EXAMPLE 32****
C
(X1 + 1) / (X1**2 + 2*X1 + 1) $
C
C  ****EXAMPLE 33****
C
(X1 + 1) / (I1*X1**2 + 2*I1*X1 + I1) $
C
C  ****EXAMPLE 34****
C
(I1*(X1 + 1)) / (I1*X1**2 + 2*I1*X1 +I1) $
C
C  CHANGE OUTPUT DIRECTIVE
C  REQUEST VECTOR OF COEFFICIENTS ONLY
C
O  0
C
```

```
C  ****EXAMPLE 35****
C
X1 + 5.6*I1 + 4.6 * X2$
C
C  NON-LINEAR EXPRESSION (NO VECTOR OUTPUT)
C
C  ****EXAMPLE 36****
C
(X1 + I1) * X3 * (I1 +X3) $
C
C  CHANGE OUTPUT DIRECTIVE
C  REQUEST ALPHANUMERIC REPRESENTATION ONLY
C
O  2
C
C  CHANGE SUBSCRIPT DIRECTIVE
C
S  3  2
C
C
C  EXAMPLES OF EXPRESSION EXPONENTS
C
C  EXPRESSION EXPONENTS THAT REDUCE TO ZERO
C  ****EXAMPLE 37****
C
X1**(-I1) * X1**I1$
C
C  ****EXAMPLE 38****
C
I1**X1**(I2+1)*I1**(-I2*X1-X1)$
C
C  ****EXAMPLE 39****
C
(X1**(I1-1) + 2*X1**(I1-1)) * X1**(1-I1)$
C
C  EXPRESSION EXPONENTS THAT REDUCE TO A CONSTANT
C  ****EXAMPLE 40****
C
X1**(-I1) * X1**(I1+2)$
C
C  ****EXAMPLE 41****
C
I1*X1**2*X1**(I1+2)*I2*X1**(3-I1+I3)*X1**(-I3) + I2$
C
C  EXPRESSION EXPONENTS THAT DO NOT FULLY REDUCE
C  ****EXAMPLE 42****
C
I1**(X1+2)$
C
C  ****EXAMPLE 43****
C
I1**X1**I2$
C
```

```
C  ****EXAMPLE 44****
C
(X1**(1-I1) + 2*X1**(1-I1))$
C
C  PRODUCE STATUS = 3 - CASE NOT HANDLED
C  EXPRESSION EXPONENT NESTED TOO DEEP
C  ****EXAMPLE 45****
C
I1**(X1**I2)$
C
C  CANCELLATION OF EXPRESSION EXPONENTS BY DIVISION
C  ****EXAMPLE 46****
C
X1**I1/X1**I1$
C
C  ****EXAMPLE 47****
C
(X1**(I1-1) + 2*X1**(I1-1)) / X1**(I1-1)$
C
C  PRODUCE STATUS = MFEXP - CASE NOT HANDLED
C  MORE THAN ONE TERM RAISED TO AN EXPRESSION EXPONENT
C  ****EXAMPLE 48****
C
(X1+2) ** (I1) * X1**(-I1)$
C
C  ****EXAMPLE 49****
C
I3**X1 / I3**(X1-1) + I2**((X2-1)*(X2+1)) * I2**(1-X2**2)$
C
C  PRODUCE STATUS = 3 - CASE NOT HANDLED
C  MORE THAN ONE TERM RAISED TO AN EXPRESSION EXPONENT
C  ****EXAMPLE 50****
C
I1**X1 + I2**(I1+2*I1) + (I1+I2)**X1$
C
C  END OF FILE
C
EOF
```

E.2. Output File from SYMPLR

```
C
C  REQUEST BOTH TYPES OF OUTPUT
C
OUTPUT BOTH REPRESENTATIONS

C
C  PRODUCE ERROR = INVSB - NO SUBSCRIPT BOUNDS SET YET
C  ****EXAMPLE 1****
C
INPUT EXPRESSION
(-I4 + 2*I3 + 3*I4 +1+1+1) * 4.2 $
INVALID SUBSCRIPT BOUNDS IN EFFECT

C
C  SET SUBSCRIPT BOUNDS, FIVE INTEGER AND FIVE REAL VARIABLES
C
SUBSCRIPT BOUNDS ARE  5 (INTEGER) AND  5 (REAL)

C
C  LINEAR EXPRESSION - REORDERING OF TERMS NECESSARY
C  ****EXAMPLE 2****
C
INPUT EXPRESSION
5.0*X4 - 12.5*I5 + 14*X2 - I3 - 189.6$
REDUCED POLYNOMIAL IS LINEAR
VECTOR OF COEFFICIENTS
 0.0000000E+00  0.0000000E+00 -0.1000000E+01  0.0000000E+00 -0.1250000E+02
 0.0000000E+00  0.1400000E+02  0.0000000E+00  0.5000000E+01  0.0000000E+00
-0.1896000E+03
ALPHANUMERIC REPRESENTATION
- I3 - 12.5*I5 + 14.0*X2 + 5.0*X4 - 189.6

C
C  PRODUCE END OF EXPRESSION NOT ENCOUNTERED
C  ****EXAMPLE 3****
C
END OF EXPRESSION TERMINATOR NOT ENCOUNTERED
X1 + I1

C
C  EXAMPLES OF MULTIPLICATION
C
C  SIMPLE TERM MULTIPLICATION
C  ****EXAMPLE 4****
C
INPUT EXPRESSION
(3*X1*I1*X2+X1)*(X2*X3+2*I1*X1)$
REDUCED POLYNOMIAL IS NON-LINEAR
ALPHANUMERIC REPRESENTATION
6.0*I1**2.0*X1**2.0*X2 + 3.0*I1*X1*X2**2.0*X3 + 2.0*I1*X1**2.0 + X1*X2*X3
```

```
C
C   CHANGE OUTPUT DIRECTIVE
C   REQUEST ALPHANUMERIC REPRESENTATION ONLY
C
OUTPUT ALPHANUMERIC REPRESENTATION ONLY

C
C   ****EXAMPLE 5****
C
INPUT EXPRESSION
5.5*X1+(I1+I2*X1)*(I2*X3*X2)$
REDUCED POLYNOMIAL IS NON-LINEAR
ALPHANUMERIC REPRESENTATION
I2**2.0*X1*X2*X3 + I1*I2*X2*X3 + 5.5*X1

C
C   MULTIPLICATION AND COMBINATION (CANCELLATION) OF LIKE TERMS
C   ****EXAMPLE 6****
C
INPUT EXPRESSION
(X1-1)*(X1**7 + X1**6 + X1**5 +
X1**4 + X1**3 + X1**2 + X1**1 + 1) $
REDUCED POLYNOMIAL IS NON-LINEAR
ALPHANUMERIC REPRESENTATION
X1**8.0 - 1.0

C
C   PRODUCE ERROR = ILCHR - ILLEGAL CHARACTER
C   ****EXAMPLE 7****
C
INPUT EXPRESSION
I1 + COS(X1)$
ILLEGAL CHARACTER ENCOUNTERED

C
C   CHANGE SUBSCRIPT DIRECTIVE
C
SUBSCRIPT BOUNDS ARE  6 (INTEGER) AND 12 (REAL)

C
C   PRODUCE ERROR = RELBB - REAL VARIABLE SUBSCRIPTED BEYOND BOUND
C   ****EXAMPLE 8****
C
INPUT EXPRESSION
X13 + 4.0*I1$
REAL VARIABLE SUBSCRIPTED BEYOND BOUND ENCOUNTERED

C
C   EXAMPLES OF DIVISION
C
C   FRACTIONAL POLYNOMIALS - COMMON DENOMINATOR NECESSARY
C   ****EXAMPLE 9****
```

```
C
INPUT EXPRESSION
(3/(X5 + I6)) + (4/(I3 - X12))$
REDUCED POLYNOMIAL IS NON-LINEAR
ALPHANUMERIC REPRESENTATION
(3.0*I3 + 4.0*I6 + 4.0*X5 - 3.0*X12) /
 (I3*I6 + I3*X5 - I6*X12 - X5*X12)

C
C  ****EXAMPLE 10****
C
INPUT EXPRESSION
(1/(X1 - X2)) - (1/(X2 - X1)) + X1 + X2$
REDUCED POLYNOMIAL IS NON-LINEAR
ALPHANUMERIC REPRESENTATION
(- X1**3.0 + X1**2.0*X2 + X1*X2**2.0 - X2**3.0 - 2.0*X1 + 2.0*X2) /
 (- X1**2.0 + 2.0*X1*X2 - X2**2.0)

C
C  CHANGE OUTPUT DIRECTIVE
C  REQUEST BOTH TYPES OF OUTPUT
OUTPUT BOTH REPRESENTATIONS

C
C  SIMPLE FACTORING OF A SINGLE VARIABLE
C  ****EXAMPLE 11****
C
INPUT EXPRESSION
(X1 * X4 + X1) / X1$
REDUCED POLYNOMIAL IS LINEAR
VECTOR OF COEFFICIENTS
 0.0000000E+00  0.0000000E+00  0.0000000E+00  0.0000000E+00  0.0000000E+00
 0.0000000E+00  0.0000000E+00  0.0000000E+00  0.0000000E+00  0.1000000E+01
 0.0000000E+00  0.0000000E+00  0.0000000E+00  0.0000000E+00  0.0000000E+00
 0.0000000E+00  0.0000000E+00  0.0000000E+00  0.1000000E+01
ALPHANUMERIC REPRESENTATION
X4 + 1.0

C
C  ****EXAMPLE 12****
C
INPUT EXPRESSION
5.5*X1+(I1+I2*X1)*(I2*X3*X2) / I2 $
REDUCED POLYNOMIAL IS NON-LINEAR
ALPHANUMERIC REPRESENTATION
I2*X1*X2*X3 + I1*X2*X3 + 5.5*X1

C
C  SIMPLE DIVISION AND FACTORING OF A SINGLE VARIABLE
C  ****EXAMPLE 13****
C
INPUT EXPRESSION
(I1*X1**2 + 2*I1*X1 + I1) / (I1*(X1 + 1))$
```

```
REDUCED POLYNOMIAL IS LINEAR
VECTOR OF COEFFICIENTS
 0.0000000E+00  0.0000000E+00  0.0000000E+00  0.0000000E+00  0.0000000E+0
 0.0000000E+00  0.1000000E+01  0.0000000E+00  0.0000000E+00  0.0000000E+0
 0.0000000E+00  0.0000000E+00  0.0000000E+00  0.0000000E+00  0.0000000E+0
 0.0000000E+00  0.0000000E+00  0.0000000E+00  0.1000000E+01
ALPHANUMERIC REPRESENTATION
X1 + 1.0

C
C  UNARY SUBTRACTION AND SIMPLE DIVISION
C  (EXAMPLE USED IN APPENDIX F)
C  ****EXAMPLE 14****
C
INPUT EXPRESSION
(-1 + X1**2)/(X1-1)$
REDUCED POLYNOMIAL IS LINEAR
VECTOR OF COEFFICIENTS
 0.0000000E+00  0.0000000E+00  0.0000000E+00  0.0000000E+00  0.0000000E+0
 0.0000000E+00  0.1000000E+01  0.0000000E+00  0.0000000E+00  0.0000000E+0
 0.0000000E+00  0.0000000E+00  0.0000000E+00  0.0000000E+00  0.0000000E+0
 0.0000000E+00  0.0000000E+00  0.0000000E+00  0.1000000E+01
ALPHANUMERIC REPRESENTATION
X1 + 1.0

C
C  REDUCTION TO A CONSTANT
C  ****EXAMPLE 15****
C
INPUT EXPRESSION
(X1**2-1)/(X1+1)/(X1-1)$
REDUCED POLYNOMIAL IS LINEAR
VECTOR OF COEFFICIENTS
 0.0000000E+00  0.0000000E+00  0.0000000E+00  0.0000000E+00  0.0000000E+0
 0.0000000E+00  0.0000000E+00  0.0000000E+00  0.0000000E+00  0.0000000E+0
 0.0000000E+00  0.0000000E+00  0.0000000E+00  0.0000000E+00  0.0000000E+0
 0.0000000E+00  0.0000000E+00  0.0000000E+00  0.1000000E+01
ALPHANUMERIC REPRESENTATION
1.0

C
C  ATTEMPT TO CHANGE SUBSCRIPT DIRECTIVE
C  PRODUCE ERROR = INVSB - SUM OF TWO BOUNDS GREATER THAN LIMIT
C
S  50  51
INVALID SUBSCRIPT DIRECTIVE

C
C  PRODUCE ERROR = INTBB - INTEGER VARIABLE OUT OF BOUNDS
C  (OLD SUBSCRIPTS STILL IN EFFECT)
C  ****EXAMPLE 16****
C
INPUT EXPRESSION
```

```
I23 + X4 + X3$
INTEGER VARIABLE SUBSCRIPTED BEYOND BOUND ENCOUNTERED

C
C  EXAMPLES OF EXPONENTIATION
C  POSITIVE EXPONENTIATION OF NUMERATOR
C  ****EXAMPLE 17****
C
INPUT EXPRESSION
((I4 + 1) ** 3)$
REDUCED POLYNOMIAL IS NON-LINEAR
ALPHANUMERIC REPRESENTATION
I4**3.0 + 3.0*I4**2.0 + 3.0*I4 + 1.0

C
C  PRODUCE STATUS = UNPAR - UNMATCHED PARENTHESES
C  ****EXAMPLE 18****
C
INPUT EXPRESSION
((X3 + X2) ** 4$
UNBALANCED PARENTHESES ENCOUNTERED

C
C  NEGATIVE EXPONENTIATION OF DENOMINATOR
C  ****EXAMPLE 19****
C
INPUT EXPRESSION
 1 / (X1+1) ** (-2) $
REDUCED POLYNOMIAL IS NON-LINEAR
ALPHANUMERIC REPRESENTATION
X1**2.0 + 2.0*X1 + 1.0

C
C  NEGATIVE EXPONENTIATION OF NUMERATOR
C  ****EXAMPLE 20****
C
INPUT EXPRESSION
 (X1+1) ** (-1) $
REDUCED POLYNOMIAL IS NON-LINEAR
ALPHANUMERIC REPRESENTATION
(1.0) /
 (X1 + 1.0)

C
C  ZERO EXPONENTIATION
C  ****EXAMPLE 21****
C
INPUT EXPRESSION
(X1 + 2) ** 0 $
REDUCED POLYNOMIAL IS LINEAR
VECTOR OF COEFFICIENTS
 0.0000000E+00  0.0000000E+00  0.0000000E+00  0.0000000E+00  0.0000000E+0
 0.0000000E+00  0.0000000E+00  0.0000000E+00  0.0000000E+00  0.0000000E+0
```

```
 0.0000000E+00   0.0000000E+00   0.0000000E+00   0.0000000E+00   0.0000000E+00
 0.0000000E+00   0.0000000E+00   0.0000000E+00   0.1000000E+01
ALPHANUMERIC REPRESENTATION
1.0

C
C   SIMPLIFICATION OF EXPONENT BEFORE EXPONENTIATION PERFORMED
C   ****EXAMPLE 22*****
C
INPUT EXPRESSION
(X1+1)**((((X1+1)*(X1+1))-
((X1+1)*(X1-1))-2)/X1)$
REDUCED POLYNOMIAL IS NON-LINEAR
ALPHANUMERIC REPRESENTATION
X1**2.0 + 2.0*X1 + 1.0

C
C   PRODUCE STATUS = OCEXP - EXPONENTIATION TOO COMPLEX
C   BASIS POLYNOMIAL HAS TOO MANY TERMS
C   ****EXAMPLE 23****
C
INPUT EXPRESSION
((I1 + I2 + I3 + I4 + I5 + I6) ** 3.0)$
OVERLY-COMPLEX EXPONENTIATION ENCOUNTERED

C
C   EXPONENT TOO LARGE
C   ****EXAMPLE 24****
C
INPUT EXPRESSION
(I1 + I2) ** 8$
OVERLY-COMPLEX EXPONENTIATION ENCOUNTERED

C
C   SET LINEARIZE DIRECTIVE TO TRUE
C   EXAMPLES OF APPLICATION OF SPECIAL LINEARIZATION TRANSFORMATION
C
LINEARIZATION TRANSFORMATION INCLUDED

C
C   ****EXAMPLE 25****
C
INPUT EXPRESSION
 X1 / (X1 * X4 + X1) $
REDUCED POLYNOMIAL IS LINEARIZED
VECTOR OF COEFFICIENTS
 0.0000000E+00   0.0000000E+00   0.0000000E+00   0.0000000E+00   0.0000000E+00
 0.0000000E+00   0.0000000E+00   0.0000000E+00   0.0000000E+00   0.1000000E+01
 0.0000000E+00   0.0000000E+00   0.0000000E+00   0.0000000E+00   0.0000000E+00
 0.0000000E+00   0.0000000E+00   0.0000000E+00   0.1000000E+01
ALPHANUMERIC REPRESENTATION
X4 + 1.0
```

```
C
C  ****EXAMPLE 26****
C
INPUT EXPRESSION
(X1 + 1) / (I1 + I2 + I3 + X1*I1 + X1*I2 + X1*I3)$
REDUCED POLYNOMIAL IS LINEARIZED
VECTOR OF COEFFICIENTS
 0.1000000E+01  0.1000000E+01  0.1000000E+01  0.0000000E+00  0.0000000E+0
 0.0000000E+00  0.0000000E+00  0.0000000E+00  0.0000000E+00  0.0000000E+0
 0.0000000E+00  0.0000000E+00  0.0000000E+00  0.0000000E+00  0.0000000E+0
 0.0000000E+00  0.0000000E+00  0.0000000E+00  0.0000000E+00
ALPHANUMERIC REPRESENTATION
I1 + I2 + I3

C
C  ****EXAMPLE 27****
C
INPUT EXPRESSION
(X1 + 1) / (X1**2 + 2*X1 + 1) $
REDUCED POLYNOMIAL IS LINEAR
VECTOR OF COEFFICIENTS
 0.0000000E+00  0.0000000E+00  0.0000000E+00  0.0000000E+00  0.0000000E+0
 0.0000000E+00  0.1000000E+01  0.0000000E+00  0.0000000E+00  0.0000000E+0
 0.0000000E+00  0.0000000E+00  0.0000000E+00  0.0000000E+00  0.0000000E+0
 0.0000000E+00  0.0000000E+00  0.0000000E+00  0.1000000E+01
ALPHANUMERIC REPRESENTATION
X1 + 1.0

C
C  ****EXAMPLE 28****
C
INPUT EXPRESSION
(X1 + 1) / (I1*X1**2 + 2*I1*X1 + I1) $
REDUCED POLYNOMIAL IS NON-LINEAR
ALPHANUMERIC REPRESENTATION
(1.0) /
 (I1*X1 + I1)

C
C  ****EXAMPLE 29****
C
INPUT EXPRESSION
(I1*(X1 + 1)) / (I1*X1**2 + 2*I1*X1 +I1) $
REDUCED POLYNOMIAL IS LINEARIZED
VECTOR OF COEFFICIENTS
 0.0000000E+00  0.0000000E+00  0.0000000E+00  0.0000000E+00  0.0000000E+0
 0.0000000E+00  0.1000000E+01  0.0000000E+00  0.0000000E+00  0.0000000E+0
 0.0000000E+00  0.0000000E+00  0.0000000E+00  0.0000000E+00  0.0000000E+0
 0.0000000E+00  0.0000000E+00  0.0000000E+00  0.1000000E+01
ALPHANUMERIC REPRESENTATION
X1 + 1.0

C
```

```
C  SET LINEARIZE DIRECTIVE TO FALSE
C  SAME EXAMPLES WITHOUT APPLICATION OF LINEARIZATION
C
C
C  ****EXAMPLE 30****
C
INPUT EXPRESSION
 X1 / (X1 * X4 + X1) $
REDUCED POLYNOMIAL IS NON-LINEAR
ALPHANUMERIC REPRESENTATION
(1.0) /
 (X4 + 1.0)

C
C  ****EXAMPLE 31****
C
INPUT EXPRESSION
(X1 + 1) / (I1 + I2 + I3 + X1*I1 + X1*I2 + X1*I3)$
REDUCED POLYNOMIAL IS NON-LINEAR
ALPHANUMERIC REPRESENTATION
(1.0) /
 (I1 + I2 + I3)

C
C  ****EXAMPLE 32****
C
INPUT EXPRESSION
(X1 + 1) / (X1**2 + 2*X1 + 1) $
REDUCED POLYNOMIAL IS NON-LINEAR
ALPHANUMERIC REPRESENTATION
(1.0) /
 (X1 + 1.0)

C
C  ****EXAMPLE 33****
C
INPUT EXPRESSION
(X1 + 1) / (I1*X1**2 + 2*I1*X1 + I1) $
REDUCED POLYNOMIAL IS NON-LINEAR
ALPHANUMERIC REPRESENTATION
(1.0) /
 (I1*X1 + I1)

C
C  ****EXAMPLE 34****
C
INPUT EXPRESSION
(I1*(X1 + 1)) / (I1*X1**2 + 2*I1*X1 +I1) $
REDUCED POLYNOMIAL IS NON-LINEAR
ALPHANUMERIC REPRESENTATION
(1.0) /
 (X1 + 1.0)
```

```
C
C  CHANGE OUTPUT DIRECTIVE
C  REQUEST VECTOR OF COEFFICIENTS ONLY
C
OUTPUT VECTOR OF COEFFICIENTS ONLY

C
C  ****EXAMPLE 35****
C
INPUT EXPRESSION
X1 + 5.6*I1 + 4.6 * X2$
REDUCED POLYNOMIAL IS LINEAR
VECTOR OF COEFFICIENTS
 0.5600000E+01  0.0000000E+00  0.0000000E+00  0.0000000E+00  0.0000000E+00
 0.0000000E+00  0.1000000E+01  0.4600000E+01  0.0000000E+00  0.0000000E+00
 0.0000000E+00  0.0000000E+00  0.0000000E+00  0.0000000E+00  0.0000000E+00
 0.0000000E+00  0.0000000E+00  0.0000000E+00  0.0000000E+00

C
C  NON-LINEAR EXPRESSION (NO VECTOR OUTPUT)
C
C  ****EXAMPLE 36****
C
INPUT EXPRESSION
(X1 + I1) * X3 * (I1 +X3) $
REDUCED POLYNOMIAL IS NON-LINEAR

C
C  CHANGE OUTPUT DIRECTIVE
C  REQUEST ALPHANUMERIC REPRESENTATION ONLY
C
OUTPUT BOTH REPRESENTATIONS

C
C  CHANGE SUBSCRIPT DIRECTIVE
C
SUBSCRIPT BOUNDS ARE  3 (INTEGER) AND  2 (REAL)

C
C
C  EXAMPLES OF EXPRESSION EXPONENTS
C
C  EXPRESSION EXPONENTS THAT REDUCE TO ZERO
C  ****EXAMPLE 37****
C
INPUT EXPRESSION
X1**(-I1) * X1**I1$
REDUCED POLYNOMIAL IS LINEAR
VECTOR OF COEFFICIENTS
 0.0000000E+00  0.0000000E+00  0.0000000E+00  0.0000000E+00  0.0000000E+00
 0.1000000E+01
ALPHANUMERIC REPRESENTATION
1.0
```

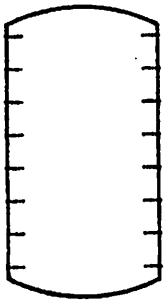
```
C
C  ****EXAMPLE 38****
C
INPUT EXPRESSION
I1**X1**(I2+1)*I1**(-I2*X1-X1)$
REDUCED POLYNOMIAL IS LINEAR
VECTOR OF COEFFICIENTS
 0.0000000E+00  0.0000000E+00  0.0000000E+00  0.0000000E+00  0.0000000E+00
 0.1000000E+01
ALPHANUMERIC REPRESENTATION
1.0

C
C  ****EXAMPLE 39****
C
INPUT EXPRESSION
(X1**(I1-1) + 2*X1**(I1-1)) * X1**(1-I1)$
REDUCED POLYNOMIAL IS LINEAR
VECTOR OF COEFFICIENTS
 0.0000000E+00  0.0000000E+00  0.0000000E+00  0.0000000E+00  0.0000000E+00
 0.3000000E+01
ALPHANUMERIC REPRESENTATION
3.0

C
C  EXPRESSION EXPONENTS THAT REDUCE TO A CONSTANT
C  ****EXAMPLE 40****
C
INPUT EXPRESSION
X1**(-I1) * X1**(I1+2)$
REDUCED POLYNOMIAL IS NON-LINEAR
ALPHANUMERIC REPRESENTATION
X1**2.0

C
C  ****EXAMPLE 41****
C
INPUT EXPRESSION
I1*X1**2*X1**(I1+2)*I2*X1**(3-I1+I3)*X1**(-I3) + I2$
REDUCED POLYNOMIAL IS NON-LINEAR
ALPHANUMERIC REPRESENTATION
I1*I2*X1**7.0 + I2

C
C  EXPRESSION EXPONENTS THAT DO NOT FULLY REDUCE
C  ****EXAMPLE 42****
C
INPUT EXPRESSION
I1**(X1+2)$
REDUCED POLYNOMIAL IS NON-LINEAR
ALPHANUMERIC REPRESENTATION
I1**(X1+2.0)
```

```
C
C  ****EXAMPLE 43****
C
INPUT EXPRESSION
I1**X1**I2$
REDUCED POLYNOMIAL IS NON-LINEAR
ALPHANUMERIC REPRESENTATION
I1**(I2*X1)

C
C  ****EXAMPLE 44****
C
INPUT EXPRESSION
(X1**(1-I1) + 2*X1**(1-I1))$
REDUCED POLYNOMIAL IS NON-LINEAR
ALPHANUMERIC REPRESENTATION
3.0*X1**(-I1+1.0)

C
C  PRODUCE STATUS = 3 - CASE NOT HANDLED
C  EXPRESSION EXPONENT NESTED TOO DEEP
C  ****EXAMPLE 45****
C
INPUT EXPRESSION
I1**(X1**I2)$
INVALID EXPRESSION EXPONENTIATION ENCOUNTERED

C
C  CANCELLATION OF EXPRESSION EXPONENTS BY DIVISION
C  ****EXAMPLE 46****
C
INPUT EXPRESSION
X1**I1/X1**I1$
REDUCED POLYNOMIAL IS LINEAR
VECTOR OF COEFFICIENTS
 0.0000000E+00  0.0000000E+00  0.0000000E+00  0.0000000E+00  0.0000000E+00
 0.1000000E+01
ALPHANUMERIC REPRESENTATION
1.0

C
C  ****EXAMPLE 47****
C
INPUT EXPRESSION
(X1**(I1-1) + 2*X1**(I1-1)) / X1**(I1-1)$
REDUCED POLYNOMIAL IS LINEAR
VECTOR OF COEFFICIENTS
 0.0000000E+00  0.0000000E+00  0.0000000E+00  0.0000000E+00  0.0000000E+00
 0.3000000E+01
ALPHANUMERIC REPRESENTATION
3.0
```

```
C
C  PRODUCE STATUS = MFEXP - CASE NOT HANDLED
C  MORE THAN ONE TERM RAISED TO AN EXPRESSION EXPONENT
C  ****EXAMPLE 48****
C
INPUT EXPRESSION
(X1+2) ** (I1) * X1**(-I1)$
INVALID EXPRESSION EXPONENTIATION ENCOUNTERED

C
C  ****EXAMPLE 49****
C
INPUT EXPRESSION
I3**X1 / I3**(X1-1) + I2**((X2-1)*(X2+1)) * I2**(1-X2**2)$
REDUCED POLYNOMIAL IS LINEAR
VECTOR OF COEFFICIENTS
 0.0000000E+00  0.0000000E+00  0.1000000E+01  0.0000000E+00  0.0000000E+0
 0.1000000E+01
ALPHANUMERIC REPRESENTATION
I3 + 1.0

C
C  PRODUCE STATUS = 3 - CASE NOT HANDLED
C  MORE THAN ONE TERM RAISED TO AN EXPRESSION EXPONENT
C  ****EXAMPLE 50****
C
INPUT EXPRESSION
I1**X1 + I2**(I1+2*I1) + (I1+I2)**X1$
INVALID EXPRESSION EXPONENTIATION ENCOUNTERED

C
C  END OF FILE
C
EOF
```

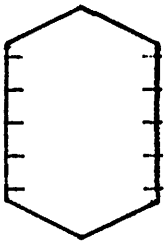
Appendix F. Example of the Evolution of the Data Structures

To follow is an example of how the data structures evolve throughout the reduction process. Each node type has a distinct shape for visual clarity. Figure F.1 provides a legend for the data structure schema - it shows all node types along with their respective fields. Figure F.2 shows the data structures after the input expression has been parsed. Figures F.3 through F.7 show the evolution of the data structures during the simplification of this expression. Each figure has two parts, the first showing the basic structure of the binary tree and the second showing the data structures in detail. Note that whenever a field of a node is empty, the value is null. Assume that the subscript bounds have been declared as IBND = 3 and XBND = 2 (hence bit vectors are 5 bits in length).



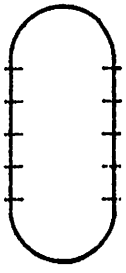
KFLAGS
KNUMER
KDENOM
KCONST
KOPVAL
KLINKL
KLINKR

KBTREE
linked-list node
for binary tree



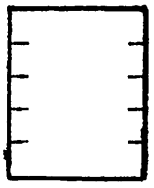
KBITVC
KCOEFF
KPNTEX
KNXTRM

KEXPRS
linked-list node
for expression list



KEXFLG
KEXVAL
KEXNUM
KNXEXP

KEXPON
linked-list node
for exponent list



1
2
:
:

IBTSTB
sequential table
for bit vectors

Figure F.1.

Legend to the Data Structures Schema

Figure F.2.

Input Expression

(-1.0 + X1 ** 2) / (X1 - 1)

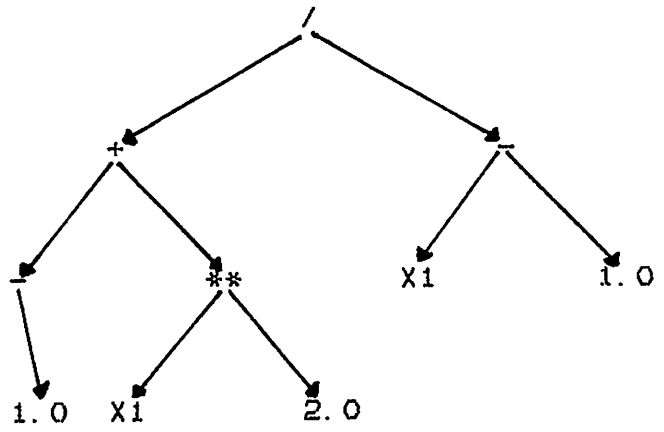


Figure F.2.1.

Binary Tree After Completion of Parse

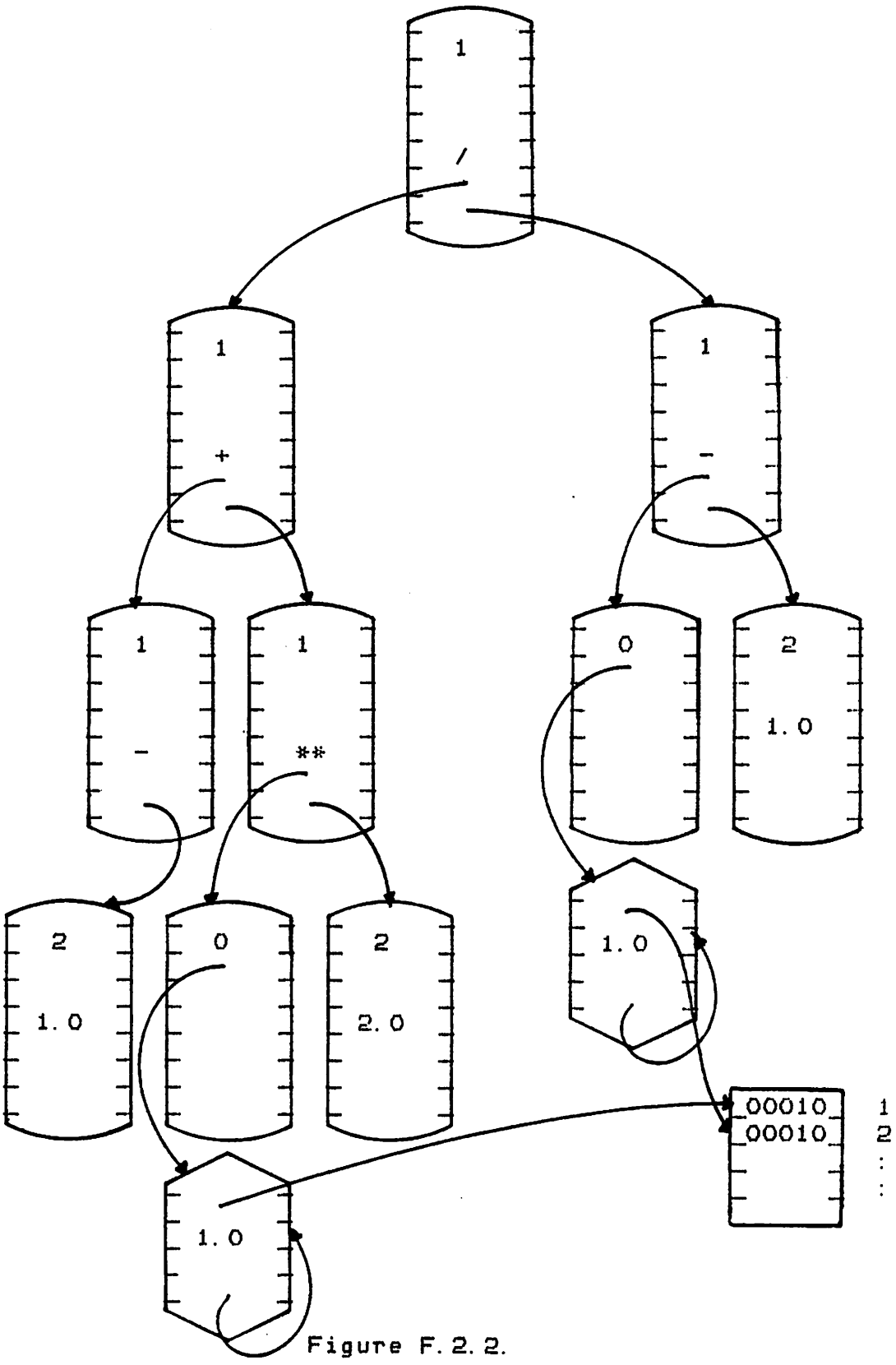


Figure F.2.2.

Data Structures After Completion of Parse

Figure F.3.

Unary Subtraction for simplification of

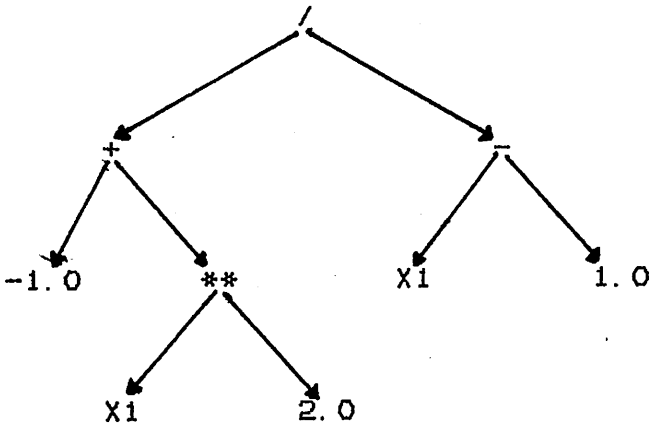


Figure F.3.1.

Binary Tree After Unary Subtraction (- 1.0)

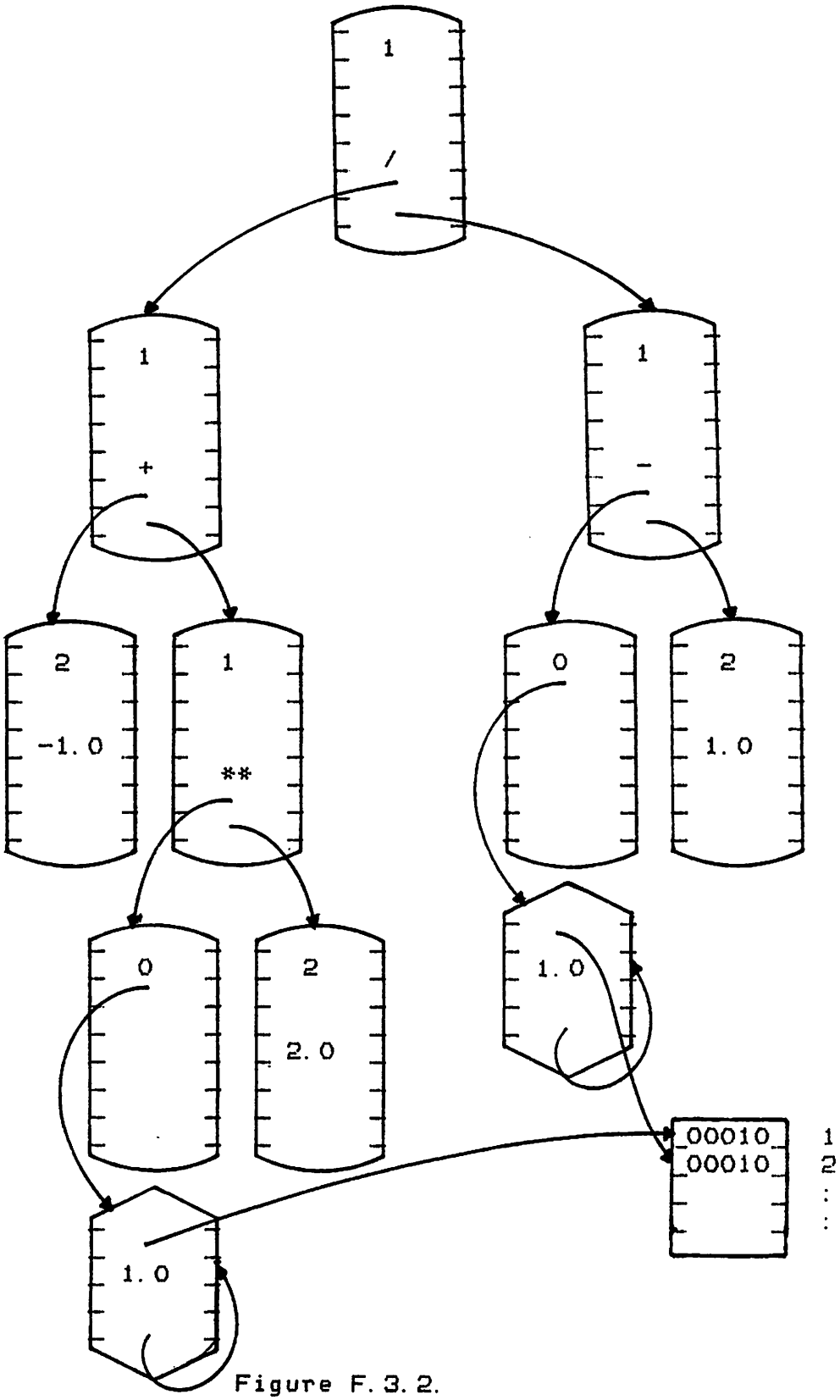
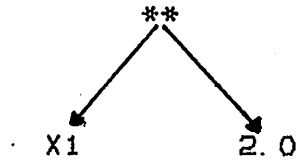


Figure F.3.2.

Data Structures After Unary Subtraction (- 1.0)

Figure F. 4.

Exponentiation for simplification of



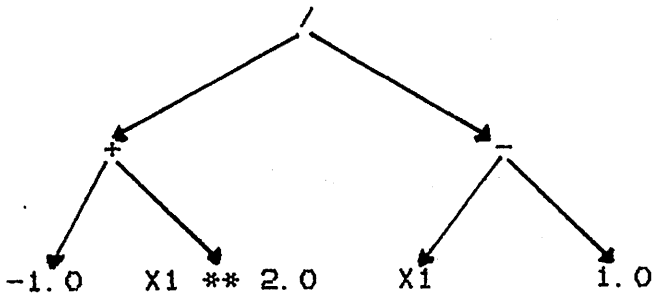


Figure F. 4. 1.

Binary Tree After Exponentiation (X1 ** 2. 0)

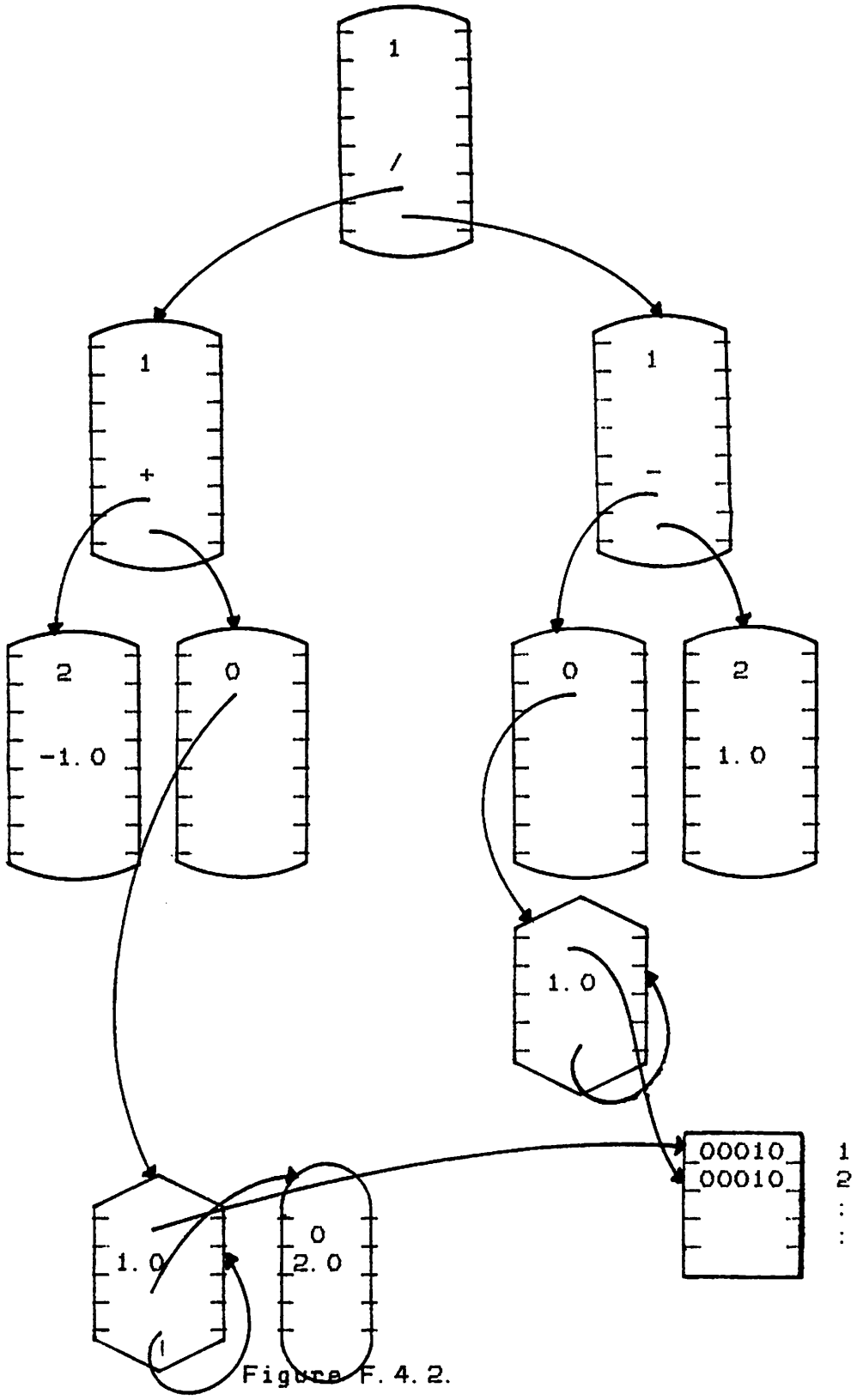
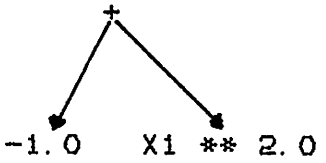


Figure F.4.2.

Data Structures After Exponentiation (X1 ** 2.0)

Figure F.5.

Addition for simplification of



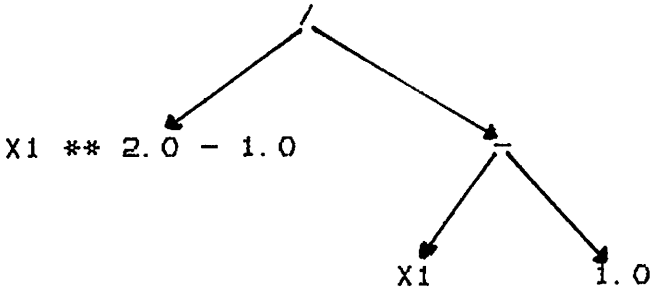


Figure F.5.1.

Binary Tree After Addition (-1.0 + X1 ** 2.0)

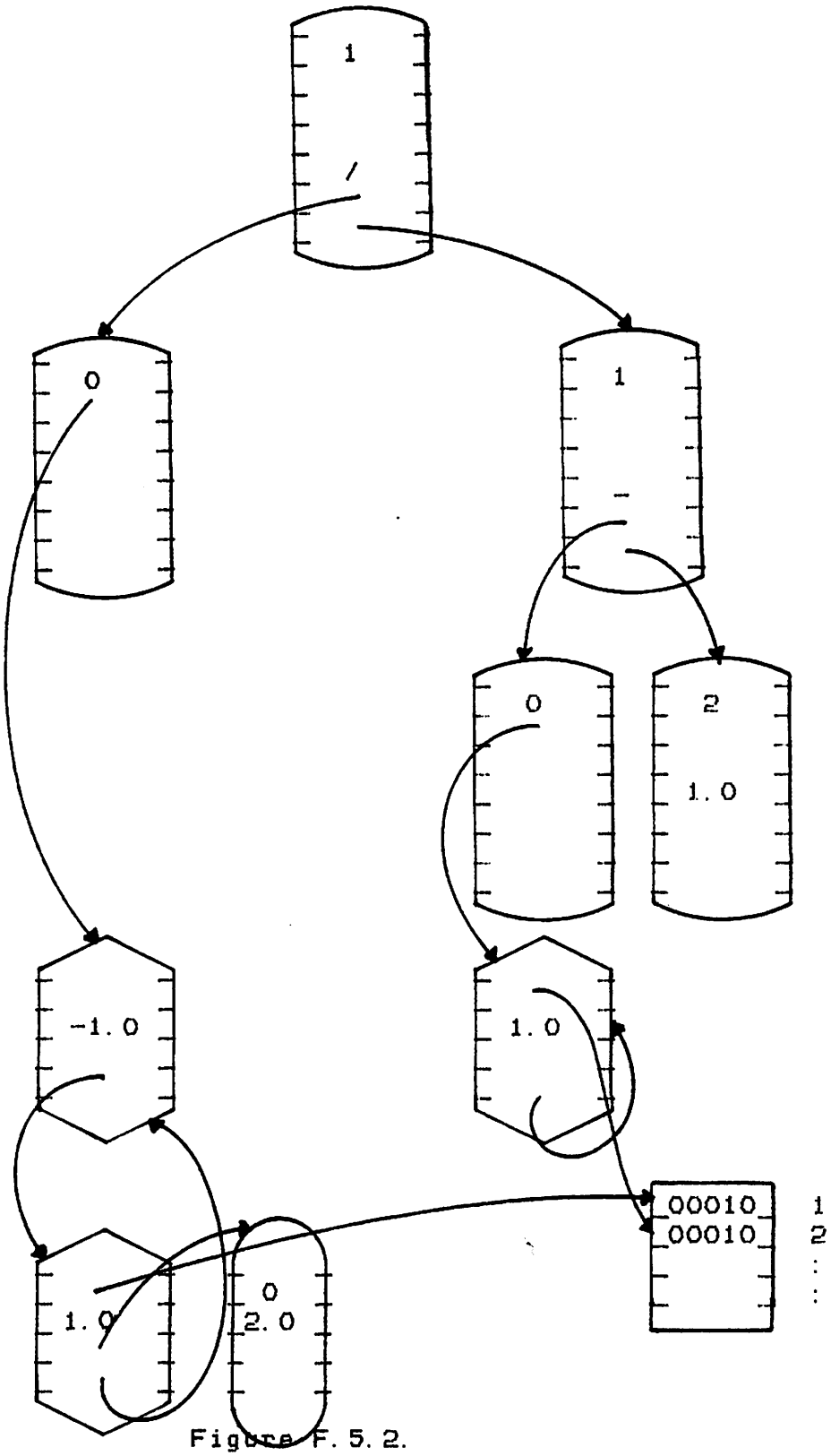
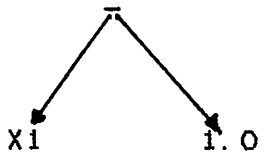


Figure F.5.2.

Data Structures After Addition (-1.0 + X1 ** 2.0)

Figure F.6.

Subtraction for simplification of



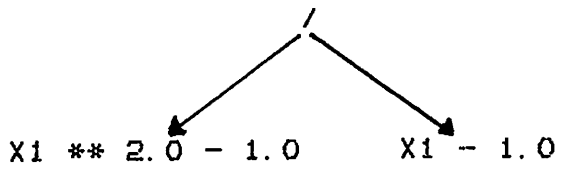


Figure F.6.1.

Binary Tree After Subtraction (X1 - 1.0)

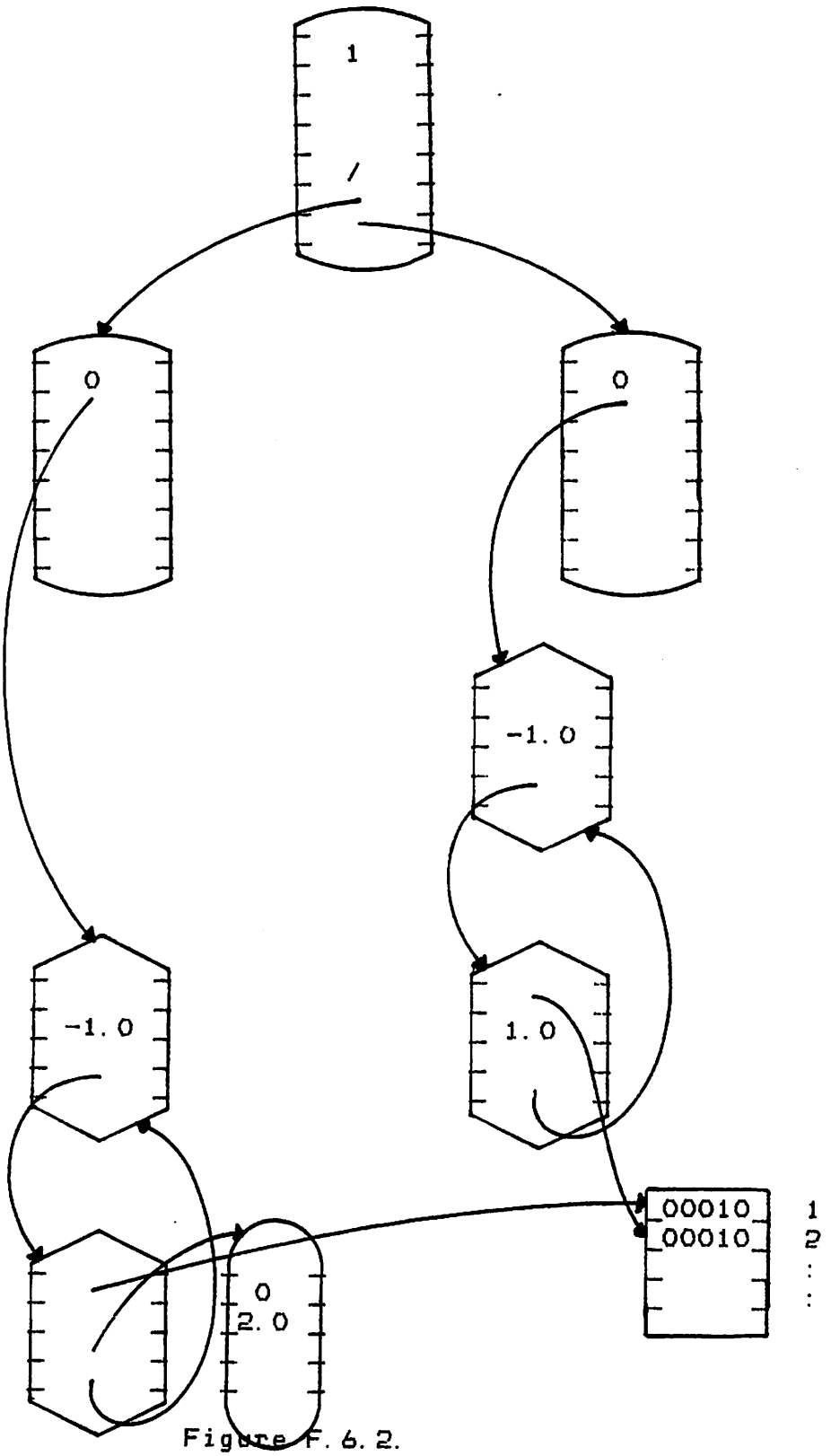
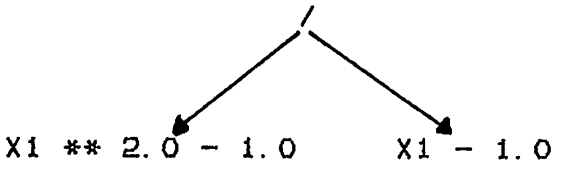


Figure F.6.2.

Data Structures After Subtraction (X1 - 1.0)

Figure F.7.

Division for simplification of



X1 + 1.0

Figure F.7.1.

Binary Tree After Division (X1 ** 2.0 - 1.0) / (X1 - 1.0)
Reduction Process Complete

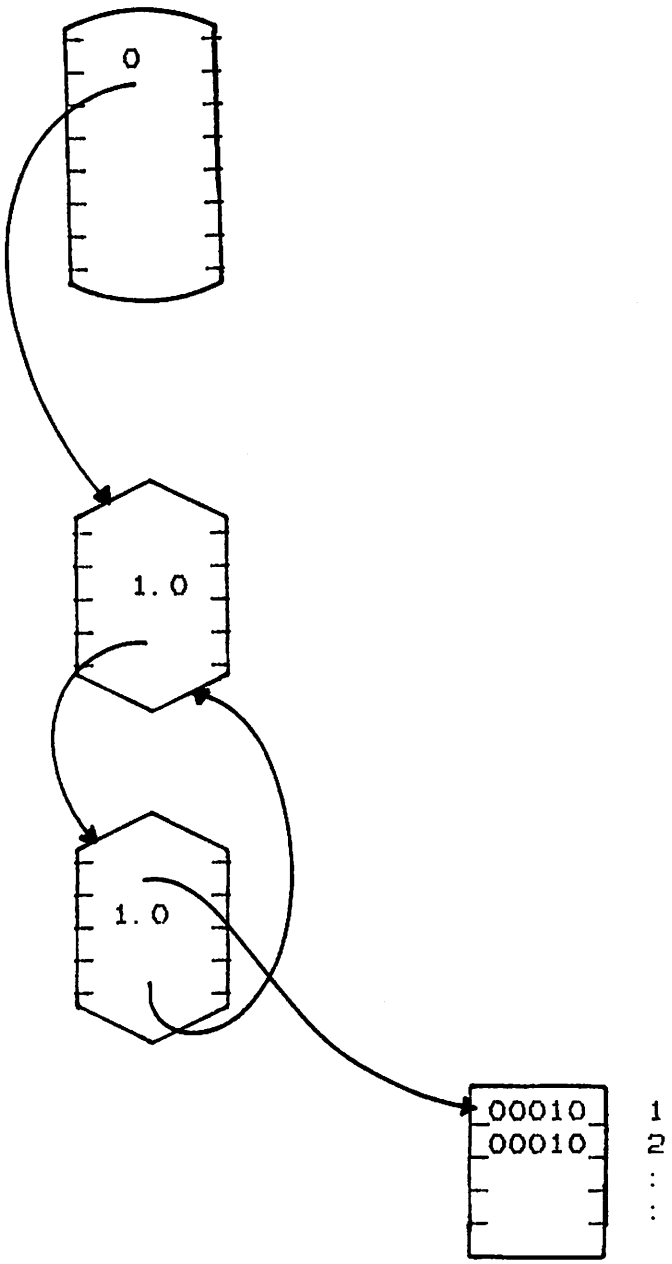


Figure F.7.2.

Data Structures After Division (X1 ** 2.0 - 1.0) / (X1 - 1.0)
Reduction Process Complete

REFERENCES

1. L. A. Clarke, "A System to Generate Test Data and Symbolically Execute Programs", IEEE Transactions on Software Engineering, September, 1976.

2. W. S. Brown, ALTRAN User's Manual, Bell Telephone Lab., v. 1, 1973.

3. L. J. Osterweil, L. A. Clarke, and D. W. Smith, "A FORTRAN System for Flexible Creation and Accessing of Data Bases", University of Colorado Report #CU-CS-052-74, August, 1974.

4. L. J. Osterweil, L. A. Clarke, and D. W. Smith, "A Data Base System Designed for Flexibility and Usability from FORTRAN", University of Colorado Report #CU-CS-072-75, July, 1975.

5. D. E. Knuth, Fundamental Algorithms, Addison Wesley, February, 1975.