Mathematical Symbol Processing

C. ABRAHAM[1]

University of Manitoba, Institute for Computer Studies
Winnipeg, Canada

T. PEARCEY[2]

CSIRO Division of Computing Research
Canberra, Australia

I. Introduction

Most of the early efforts to write computer programs which perform symbolic mathematical operations were directed toward polynomial manipulation including their differentiation and integration [1, 2]. In 1961, Bernick *et al.* [3] produced an interpretive routine to provide multiple capabilities for a general class of mathematical expression. More recent programs belonging to the same class are FORMAC [4] and Formula ALGOL, [5] but both suffer various kinds of restrictions.

This paper describes a program, Mathematical Language Processor (MLP). The scope of MLP is to give maximum flexibility to the user to perform any mathematical operation on a general expression. In some respects the program is similar to Formula ALGOL in the sense that the user can add his own subroutines in order to perform special operations. The program is written in FORTRAN language and directed towards FORTRAN calls, so that it may be adapted readily to any sufficiently large computer installation with FORTRAN facilities, either in a batch manner of usage, or via a visual display. In the latter case the user and system interact and the user can more readily control the course of any series of symbolic manipulations. MLP has

[1] Formerly of CSIRO Division of Computing Research, Canberra, Australia.
[2] Present affiliation is: Control Data, Canberra, Australia.

been written so far for a Control Data 3600 for which the data structure will be described.[3] Data formats can be adjusted to suit the particular installation.

II. Data Structure

The program MLP works with two kinds of symbols, external and internal.

1. An external symbol is 12 bits long. The least significant 8 bits represents one of 256 characters, the next 2 more significant bits indicate to which line, i.e., normal, superscript or subscript, the character belongs.

00 Normal line
01 Superscript line
10 Subscript line

The most significant 2 bits are not at present used. A 12 bit symbol is used because it can be represented easily in practice by a combination of pairs of characters from a 6 bit set.

2. An internal symbol is 24 bits long. The meaning of each bit will be explained later.

A. Expressions

The primary unit of information provided to MLP is a symbol string which may be an expression. An expression, which may be a mathematical equation, is preceded by a label of up to 2 symbols, which must be unique and enclosed in brackets. Variables are represented by single letters and multiplication is implied by concatenation except for numerical products where * is used.

$(A_1)\ Y = 3xz + 1.2$
$(A_2)\ 5T_n{}^m$
$(A_3)\ Z = 3.9 * 5.7y$

The lists and buffers needed by MLP are the following.

(1) EPL (Expression Label List) This list holds the expression labels as symbol pairs. If the user presents a label with only one symbol, blank is inserted in the front of the given symbol. The number of entries can be set either by the user or the installation. The number of entries is the same as for the lists, EPADDR and EPSNUM.

[3] Lately the data structure has been adapted and programs implemented for IBM 360 series.

(2) EPST (Expression Storage Buffer) This buffer holds the expressions in internal representation. The size depends on the user or the installation.

(3) EPADDR (Expression Address List) Each entry gives the relative starting address of the expression in the EPST buffer in the same order as the corresponding labels in EPL.

(4) EPSNUM (Expression Symbol Number List) Each entry holds the number of internal symbols belonging to the expression in the same order as in EPADDR.

When MLP is presented with an expression it checks the uniqueness of the label and then inserts it into the EPL and enters the expression onto the EPST buffer removing blanks and preceding each 12 bit symbol by 12 binary zeroes. The number of symbols belonging to the expression is entered into EPSNUM and the starting address of that expression relative to the EPST buffer is transferred to EPADDR.

B. Declarations

Secondary units of information are called "declarations." A declaration informs MLP of the meaning to be attached to a string of one or more consecutive symbols in expressions or previous declarations.

The following lists and buffers are used for the handling of declarations.

(1) DECBUFF (Declaration Buffer) This holds the strings of symbols representing the different declarations. The size of the buffer can be determined either by the user or the installation.

(2) DECLADD (Declaration Address List) Each entry holds the relative starting address of the declaration in DECBUFF.

The number of entries is determined by the user or the installation. All the following lists have the same size as DECLADD.

(3) DECLSNUM (Declaration Symbol Numbers List) Each entry holds the number of symbols belonging to the respective declarations.

(4) GROUPDES (Group Designator List) Each entry holds a designator which indicates that:

(a) either the declaration is an "operator" of single level or of mixed levels, i.e., a combination of symbols of various levels,

(b) the declaration is a "dependent" function of single or mixed levels, or

(c) the declaration is a definition of single or mixed levels.

(5) DEFSUM (Definition Symbol Number List) This list holds the number of symbols in the right part of each corresponding definition.

When MLP is presented with a declaration which involves more than one symbol, it checks to ensure that the declaration is not already in DECBUFF and if not, inserts it there, preceding the symbol by 12 binary zeroes, and plants the starting address of that declaration relative to DECBUFF onto the DECLADD list. The number of symbols belonging to the new declaration is entered onto the DECLSNUM list and the other necessary information into the GROUPDES and DEFSNUM lists.

III. Types of Declarations

There are four types of declarations.

A. Operator

This is a group of one or more symbols which correspond to a specific functional transformation of the succeeding subexpression, e.g., to COS can be attached the meaning of the trigonometrical functional transformation "cosine" via a table of definitions and properties of each operator or a definitional declaration. MLP inserts a unit into bit 22 of the corresponding internal symbol wherever the symbol is represented in an expression or a previous declaration pointed to from the specific expression. In the case of multisymbol operators MLP inserts the necessary information into DECBUFF and the associated lists as explained previously. It also replaces the respective consecutive string of symbols from the specific expressions or declaration by a pointer, which is an internal symbol, and points to an entry into the DECLADD. The format of a pointer is shown in Fig. 1a.

B. Independent Variable

A symbol can be declared as an independent variable. In such a case MLP inserts a unit into bit 21 of the respective symbol anywhere it appears in the relevant expressions and the declarations pointed to from that expression. The format of an internal symbol is shown in Fig. 1b.

C. Dependent Function

A string of one or more consecutive symbols can be declared as a dependent function, e.g., $f(x,y)$. In such a case MLP inserts the string of consecutive symbols into the DECBUFF. The string of symbols is replaced in the relevant expression or the declarations pointed to from that expression by a pointer as in Fig. 1c. If the string has only one symbol, DECBUFF and the other associated lists are not affected and MLP sets bit 20 to unity in that particular symbol anywhere it appears in the expression or in the declaration pointed to from that expression.

(a)

	1	1	0	0	0	L	0	0	P
Bit	23	22	21	20	19	18 17	16	15	14 - 0

(b)

	0	0	1	0	0	H	S
Bit	23	22	21	20 - 10	9	8	7 - 0

(c)

	1	0	0	1	0	L	0	0	P
Bit	23	22	21	20	19	18 17	16	15	14 - 0

FIG. 1. (a) A pointer for an operator. L = level indicator, P = address in DECLADD. (b) Independent variable code. H = level indicator, S = symbol. (c) Dependent function pointer. L = level indicator, P = address in DECLADD.

D. DEFINITION

To a string of one or more consecutive symbols can be associated a new string of symbols. This can be looked upon as a definition or a replacement. For instance assume that the equation $(\mathrm{AB})D = e^{ac}/c$ has to be provided to MLP, where $c = bT^{ij}_{nm}$ and AB is the expression label.

(1) For the equation $(\mathrm{AB})D = e^{ac}/c$, MLP would insert into EPST the string shown in Table I.

(2) For the definition $c = bT^{ij}_{nm}$ for equation AB, MLP would supply to DECBUFF the 24 bit strings of Table II.

TABLE I
EPST STRING FOR $D = e^{ac}/c$

Bit 9	8	7 - 0
0	0	D
0	0	=
0	0	e
0	1	a
0	1	c
0	0	/
0	0	c

TABLE II
DECBUFF STRING FOR
$C = bT^{ij}_{mn}$

Bit 9	8	7 - 0
0	0	c
0	0	=
0	0	b
0	0	T
1	0	n
1	0	m
0	1	i
0	1	j

The c's in the expression (AB) are replaced by a pointer P which points to the address in DECLADD which contains the address of the head of the definition, $c = bT^{ij}_{nm}$, in DECBUFF as in Fig. 2.

As a further example let the equation (A_1), $y = f(x)$ be transmitted to MLP where $f(x) = \sin x$, sin being the trigonometric function and x the independent variable. MLP inserts the equation into EPST and for the declaration (dependent function $f(x)$), it replaces the symbol $f(x)$ by an appropriate pointer to the corresponding entry in DECLADD.

Further, for the definition $f(x) = \sin x$ in equation (A_1), MLP replaces the $f(x)$ symbols in DECBUFF by a pointer in DECLADD corresponding to the appropriate definition in DECBUFF.

For the operator " sin " in equation (A_1), MLP will make a further change to DECBUFF, i.e., replace the string "sin" by a further pointer in DECLADD to its symbol string in DECBUFF.

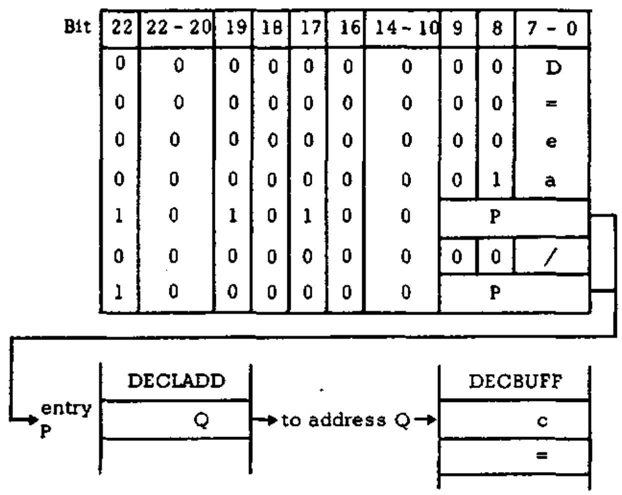

FIG. 2. Pointers in the expression $D = e^{ac}/c$.

In this way the representation of data corresponds to a combination of linear symbol strings which have a list type structure where necessary. Thus a symbol string may contain a pointer to a further string which itself contains another pointer. In particular, pointers in EPST to DECLADD may lead to entries in DECBUFF which further contain pointers to DECLADD and so on.

IV. Dynamic Lists and Buffers

MLP lists and buffers are dynamic and are currently treated as if they were stored in a linear string fashion. However, it is likely that MLP will be arranged to make the dynamic properties of these lists more readily manageable by link-listing blocks of symbols, the size of the block being about the size of an average expression. This would probably be a suitable compromise for obtaining reasonable speed with little additional demand for main storage areas.

V. The Language

Assuming that MLP is provided with an interface to general purpose displays, the language for its use with such a medium will consist of a combination of input and output, as well as transformation and definitional statements. In particular, in the transformation area there will be an expanding demand of operators such as simplify, factorize, substitute, reorder, etc., and, in the output area it must be possible to view any part of the file created by the user.

VI. Objectives

The aim of the first stage of the project is to provide subroutines for doing algebraic manipulation including simplification, factorization, and numerical evaluation of some classes of algebraic expressions.

The next stage is to implement into the system mathematical operations such as complex variable algebra, differentiation, integration, vector, matrix and tensor manipulation, and to produce solutions for systems of differential equations, by using suitable algorithms, or table look-up. An essential facility will be for the user to provide additional subroutines, thus making the system arbitrarily expandable.

REFERENCES

1. Kahrimanian, H. G., Analytical Differentiation by a Digital Computer, M.A. Thesis, Temple University, Philadelphia, Pennsylvania, 1953.
2. Williams, L. H., Algebra of Polynomials in Several Variables for a Digital Computer, *J. ACM* 9, 29 (1962).
3. Bernick, M. D., Callender, E. D., and Sonford, J. R., ALGY—An Algebraic Manipulation Program. *Proc. Western Joint Comput. Conf.* p. 389 (1961).
4. FORMAC (Operating and User's Preliminary Reference Manual), No. 7090 R2 IBM-0016, IBM Program Inform. Dept., Hawthorne, New York, 1965.
5. Perlis, A. J., Iturrioga, R., and Standish, T. A., A Preliminary Sketch of Formula ALGOL, p. 52. Carnegie Inst. Technol., Pittsburgh, Pennsylvania, 1965.