Final Report: MAGIC PAPER - AN ON-LINE SYSTEM FOR THE MANIPULATION OF SYMBOLIC MATHEMATICS

by L. C. Clapp, D. E. Jordan, E. J. Wax, R. S. Wolf

COMPUTER RESEARCH CORPORATION
429 Watertown Street
Newton, Massachusetts 02158
(617) 969-7150

Report No. R 105-1

15 April 1966

Submitted to:

DYNAMICS PROCESSES BRANCH
DATA SCIENCES LABORATORY
AIR FORCE CAMBRIDGE RESEARCH LABORATORIES
HANSCOM FIELD
BEDFORD, MASSACHUSETTS

Final Report: MAGIC PAPER - AN ON-LINE SYSTEM FOR THE MANIPULATION OF SYMBOLIC MATHEMATICS

by L. C. Clapp, D. E. Jordan, E. J. Wax, R. S. Wolf

COMPUTER RESEARCH CORPORATION
429 Watertown Street
Newton, Massachusetts 02158
(617) 969-7150

Report No. R 105-1

15 April 1966

Submitted to:

DYNAMICS PROCESSES BRANCH
DATA SCIENCES LABORATORY
AIR FORCE CAMBRIDGE RESEARCH LABORATORIES
HANSCOM FIELD
BEDFORD, MASSACHUSETTS

TABLE OF CONTENTS

TABLE OF CONTENTS
(Cont'd.)

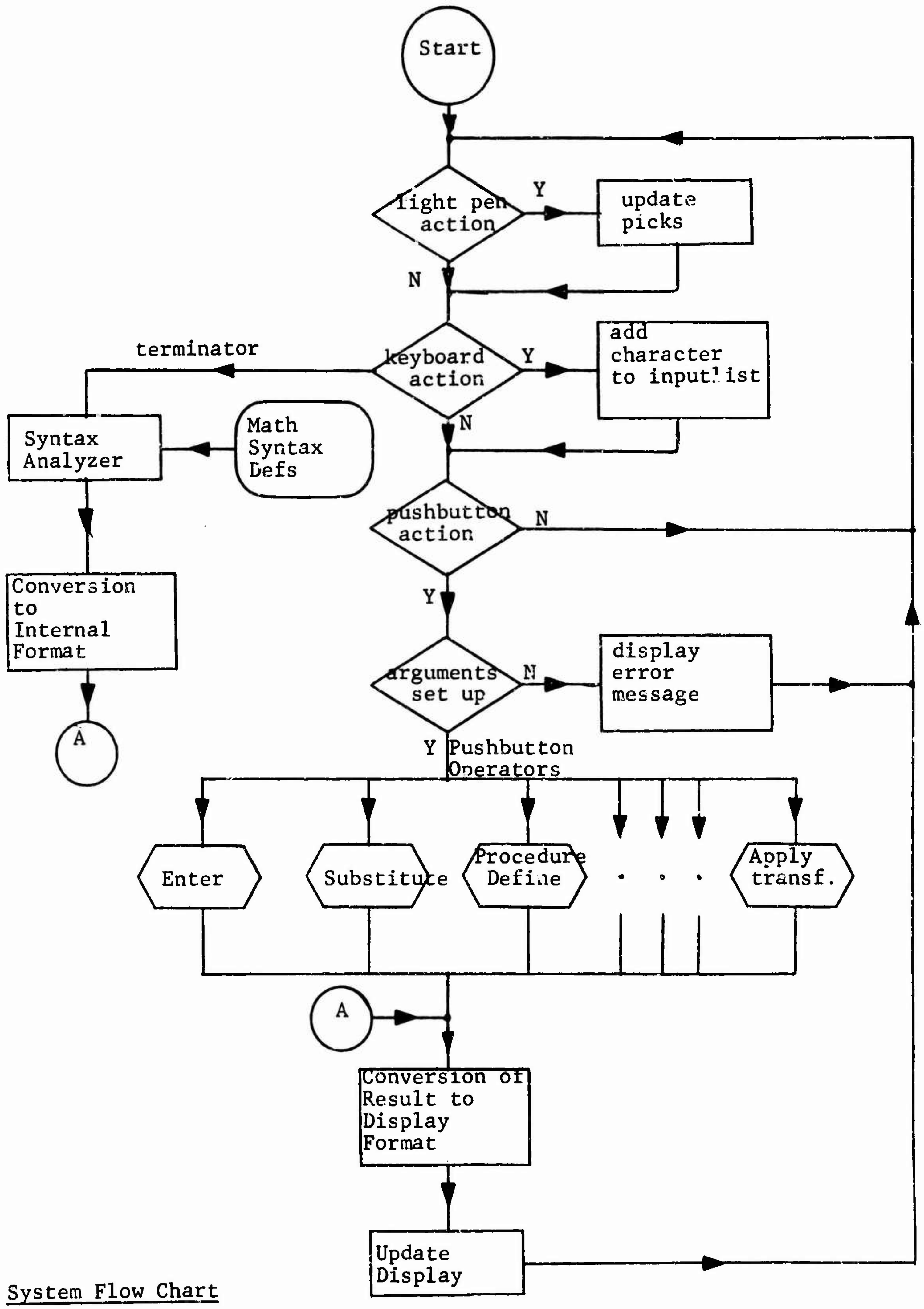
I. Introduction

This report describes the preliminary version of the MAGIC PAPER System which is being developed by Computer Research Corporation for the Dynamic Processes Branch of the Data Sciences Laboratory at the Air Force Cambridge Research Laboratories. Through a conversational interaction, the system aids the scientist, engineer or mathematician as he performs symbolic operations on linear algebraic equations. The user begins by entering his initial equations and conditions through a mathematical keyboard. As he types these equations, they are displayed on a flicker-free scope in standard mathematical notation. Using a push-button control panel and a light pen, he may select expressions and operations which are to be performed on them. If the operation is legal, the system generates a new equation which is then added to the scope display.

With the basic set of operations, the user may create new operators which can then be added to the system. He can also introduce special notational conventions. In other words, the user has considerable control which enables him to personalize the system to meet his own particular needs.

A flow chart indicating the major functional components and logic of the system is shown on the following page. The succeeding sections of this report describe the system in detail. Examples of the system's use and several significant features of the internal structure are described in the appendices, which also contain lists of the sub-routines used in the full system.

The MAGIC PAPER System is being implemented on the DX-1 processor at Hanscom Field and makes use of the Visual Information Processor (VIP) developed by Charles W. Adams Associates.



System Flow Chart

II. The Console

The system console consists of the following user devices:

1. A DEC type 30 cathode ray tube display and drum-driven display processor that provides a flicker-free display.

2. A DEC type 32 light pen.

3. A pushbutton panel.

4. A keyboard containing the Roman and Greek alphabets and a number of mathematical symbols.

5. Two foot switches.

A. User-Console Interaction

The focal point for controlling the system is the push-button panel. Each button on the panel corresponds to a system operator which is called whenever its button is depressed. These operators control the display format and manipulate expressions. The arguments required by the operators are either selected by the light pen or are entered through the keyboard. The expressions the user types on the keyboard and the results of his manipulations are displayed in conventional mathematical format on the scope. Thus a normal sequence of interaction would involve selecting a group of arguments with the light pen or entering them through the keyboard, invoking a push-button operator, and viewing the results on the scope.

B. Display Format

The equations entered and the results derived are organized into a system of "scrolls". The current scroll is usually displayed in an area occupying the lower 7/8 of the scope face. At the top is an area used for the display of the expression being entered. As each new result is developed or new equation entered it is placed at the end of the scroll. When the display area is filled with equations the scroll is automatically rotated to make room for new results. The user may also rotate the scroll using operators to be described later.

C. Use of the Light Pen

1. Selecting Arguments

The light pen is used to select displayed expressions to be used as arguments for pushbutton operators. Using techniques described in the next section, expressions on the scope may be brightened or underlined. We shall say that such an expression has been picked. After picking the desired expression, the user presses the Select Argument pushbutton to define that expression as an argument.

2. Picking Techniques

To pick a character the light pen is pointed at the character with the penswitch depressed. In response the picked character is brightened or underlined to show the user that the system recognizes his pick. At this point the user may release the penswitch to terminate the picking process, or shift the light pen to another character. If he merely shifts the pen the new character becomes the pick and the previous character is no longer picked. There are a number of ways to pick expressions larger than a single character. Each allows the user to pick only legal mathematical expressions. Consider the following example: given the equation

$$a(x + y)^2 + b = 0$$

suppose we wish to select the left hand side as an argument for an operator. We may use any of the following techniques.

1. Operator Pick - Picking the + preceding the b picks the whole left side of the above equation as shown in fig. 1a. In general, picking an operator causes that operator and all of its arguments to be picked.

2. Inclusive Pick - Picking the a and the b and depressing the Inclusive Pick button picks the left side of the equation (fig. 1b). In general the Inclusive Pick brightens the minimum legal expression containing both picked characters (e.g. the picks in fig. 1c yield the same expression).

3. Expanding Pick - Picking the x a second time picks the next higher expression, x + y. Each repick of x causes the next higher expression containing x to be picked (fig. 1d.).

   Note the difference between the actions of the operator pick and the expanding pick in an expression involving more than two arguments joined by an associative and commutative operator such as addition (fig. 2). The operator pick brightens only the arguments of the picked operator, while the expanding pick brightens the complete sum.

   In addition, an expanding or operator pick may be used to define the end points for an inclusive pick.

$$\underbrace{a(x+y)^2 \overset{\overset{\text{pick}}{\downarrow}}{+} b}_{\text{picked expression}} = 0$$

Fig. 1a. Example of Operator Pick

$$\underbrace{\overset{\overset{\text{pick 1}}{\downarrow}}{a}(x+y)^2 + \overset{\overset{\text{pick 2}}{\downarrow}}{b}}_{\text{picked expression}} = 0$$

Fig. 1b. Example of Inclusive Pick

$$\underbrace{a(\overset{\overset{\text{pick 1}}{\downarrow}}{x}+y)^2 + \overset{\overset{\text{pick 2}}{\downarrow}}{b}}_{\text{picked expression}} = 0$$

Fig. 1c. Another example of Inclusive Pick

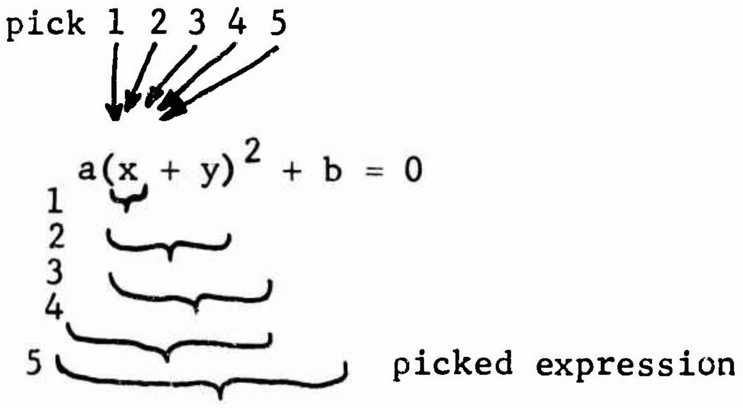


Fig. 1d. Example of Expanding Pick.

$$\underbrace{x \overset{\overset{\text{pick}}{\downarrow}}{+} y}_{\text{picked expression}} + z = 0$$

Fig. 2a.

$$\overset{\text{pick } 1\ \ 2}{\swarrow\swarrow}$$

$$\begin{array}{l} 1 \\ 2 \end{array}\ \underbrace{\underbrace{x}_{} + y + z}_{} = 0$$

picked expression

Fig. 2b.

$$\underbrace{a \overset{\text{pick 1}}{\overset{\downarrow}{+}} b} + c + d + \underbrace{e \overset{\text{pick 2}}{\overset{\downarrow}{+}} f} = 0$$

picked expression

Fig. 3.

The user may pick several, but not all, arguments of an associative and commutative operator by using the technique illustrated in figure 3. Once all the picks have been made and the desired expressions are brightened or underlined, the user presses the Select Argument button. Brightening and underlining are then terminated, and the user may select the next argument or invoke a pushbutton operator.

III. Elements of the System

A. Pushbutton Operators

Operators are preprogrammed routines assigned to pushbuttons and form the central language of the system. Operators perform the following types of tasks:

1. Scroll manipulation
2. Equation editing
3. Equation input
4. Application of mathematical operators to expressions, e.g., adding two equations, or raising an expression to a power.
5. Simplification
6. Application of transformations to expressions
7. Definition and execution of user procedures.

Each operator is initiated by selecting its arguments with the light pen or entering them through the keyboard and then depressing the appropriate pushbutton. For example, to enter an equation the user types the equation on the keyboard and then depresses Enter. If the user does not specify the arguments the system will tell him via the CRT what arguments are required for that operator.

The system is open-ended in that new operators may be programmed in DECAL-BBN making use of previously defined operators. A particularly valuable operator would be one that allowed the user to create new operators on-line in a higher level language than DECAL-BBN. A simple class of such operators, called user-defined procedures, can be created on-line and are described in Section III-C.

B. Transformations

In addition to the control operators described above, the system contains mathematical relationships in the form of identities or transformations. In effect, these transformations determine the "mathematics" known to the system. A transformation is simply a pair of expressions or equations, the second derived from the first by an unspecified sequence of manipulations. Consider the following example of a transformation:

$$(a + b)^2 \rightarrow a^2 + 2ab + b^2$$

Transformations are used with the "Transform Apply" operator whose arguments are a transformation and a selected expression. The operator "matches" the selected expression with the left hand part of the transformation to determine if they are of the same form and then determines the relations between the variables of the form (the left hand of the transformation) and the instance (the selected expression). The operator then replaces the variables in the right hand part of the transformation by the corresponding quantities in the instance to produce the result. Consider applying the above quadratic transformation to the instance $(x + 3)^2$. The operator matches the "x" with "a" and the "3" with "b" to produce the result $x^2 + (2)\ (a)\ (3) + 3^2$. Note that this result is not automatically simplified. In addition the transformation could have been applied to a more complex expression such as,

$$(xy + \frac{ab^2}{c})^2$$

The transformations in the system are organized into tables for easy reference by operators and users. These tables may be displayed instead of the current scroll by depressing the left foot switch. The user may modify the transformations tables or enter his own transformations into tables with appropriate operators.

C. User-Defined Procedures

An operator called Procedure Define allows the user to combine a series of operators into a single operator.

As an example, let us develop a procedure (SOLV2) to solve a pair of linear equations, A and B for the variables X and Y, assuming we have previously defined an operator SOLVE that solves a linear equation for a given variable. SOLV2 also uses the operators SUBST (for substitute), DISPLAY (to enter the expression into the current scroll), and ⇒ (assign result to following keyboard letter). Underlined words refer to single buttons on the pushbutton panel.

```
        DEFINE SOLV2 A B X Y

    A X SOLVE ⇒ S
    S B SUBST ⇒ T
    T Y SOLVE ⇒ R
      R DISPLAY
    R S SUBST ⇒ Q
      Q DISPLAY
        END
```

Solve equation A for X, assign result to S. Substitute for X in equation B, assign result to T. Solve T for Y, assign result to R. Display unsimplified solution for Y. Substitute for Y in equation giving X in terms of Y. Display solution for X.

IV. Input of Mathematical Expressions

A. Introduction

The user enters all mathematical expressions directly from the keyboard, using input notation closely resembling standard mathematical language. Except for a few types of notation and the conventions for typing two-dimensional information, all of which will be discussed in the next section, the user's intuitive notion of mathematical language will enable him to enter expressions correctly.

As the user types an expression, the characters are displayed in the upper portion of the scope. Exponents, subscripts, and superscripts are displayed normally; but no other formatting of the expression takes place. When the expression is completed, as signaled by the user typing a double carriage return, it is redisplayed in a completely formatted form. If the expression is ambiguous, the system, in most instances, notifies the user. However, by examining the redisplayed version the user can verify that the correct interpretation of the expression has been made. If he desires to change the expression he can either delete it and then type another, or he can modify it using the edit operator described in Section V.

The material in this chapter is divided into two sections. The first describes the notation built into the system and the rules for using this notation. The second describes the input and use of user-defined functions and operators, which once defined become part of the language. Extensive examples of input are shown in Table A.

B. Standard Notation

1. "Implied Multiplication"

In accordance with standard usage, a multiplication sign is understood between items which are strung together, e.g. 3X or XY. Numbers and expressions appearing in the middle of such strings must be parenthesized. Specifically, the first item may be a number, and every succeeding item must be a variable, a function, or a parenthesized expression. In addition, any of these items may be exponentiated (See Table A, Ex. 14, 20, 27).

However, a mutiplication sign may still be used in input. Note that the implied multiplication feature necessitates the use of single-letter variable names.

2. Numbers

Numbers may be of any type: positive or negative, integral, decimal (.1; 1.; 1.1; 1.10; etc.), fractional or mixed (2 1/2). Mixed numbers should have a single space between integer and fraction. The use of fractions often causes ambiguities, and the user should note that fractions are not treated like ordinary quotients, e.g., "1/2x" is taken to mean (1/2)x, but "1/ax" is taken to mean 1/(ax). For further examples, see Table A, ex. 1-4, 11, 17, 43-46.

3. "Line Changing" Notation

Input of two-dimensional expressions involving exponents, subscripts and superscripts requires special conventions because keyboard input is "linear". These operation are indicated by the $\uparrow$, "sub" and "super" keys, respectively. In this paper, we will denote the last two by $\vee$ and $\wedge$.

Variables and function names are the only items which may have "scripts" (superscripts or subscripts). A letter* (i.e., a variable) may have each type of script, and if it has both, <u>the superscript must come first</u>. Scripts may be multiple, containing several expressions with commas between them. Furthermore, scripts and exponents may themselves contain scripts and exponents, etc. (Note that a$\vee$ij? would not be interpreted as a double subscripted letter, but as a letter with the single subscript iXj. The double indexing notation can be indicated by a$\vee$i,j?). Each of the scripting operations is terminated by a question mark (?), which means "return to previous line" or equivalently, terminate last exponentiation ($\uparrow$), superscripting ($\wedge$), or subscripting ($\vee$) operation. The terminator may <u>not</u> be omitted, even in simple unambiguous cases like $x\uparrow 2?$, for x^2. Thus, $x^{(y_i^2)}$ may be input as $x \uparrow y \vee i \wedge 2???$. Another terminator symbol, the verticle bar (|), meaning "return to <u>main</u> line" may be used in place of one or more question marks. Thus a single | could have been typed instead of the three question marks in the above example. The user should note that his expression

* May be English or Greek, upper or lower case.

will be displayed in two-dimensional form and the line-changing characters will not be shown.

Subscript notation can be abbreviated in the following special case: if a letter is followed directly by an integral, decimal or mixed number, the number is interpreted as a subscript. But note that $x_{1/2}$ must be input as "x↓1/2?", since "x1/2" would be taken to mean $x_1/2$.

Examples of the use of line-changing notation may be found in Table A, Ex. 6-13, 15, 18, 23, 24.

4. Transcendental Functions:

The functions sin, cos, tan, csc, sec, cot, log (base 10) and ln (base e), are typed in exactly this form. Unlike normal functions they cannot be scripted, but they may be exponentiated. Thus, "sin↑2?x" and "(sin x)↑ 2?" are two ways of inputting $\sin^2 x$, or $(\sin x)^2$. Note that "sin↑ -1?x" does not mean arcsin x, and at present the inverse trigonometric functions are not in the system.

Spaces are not necessary between the function name and its argument and will be ignored. However, spaces may not appear in the middle of a function name. Writing "s i n x" is a simple way to input (s)(i)(n)(x).

In accordance with standard usage certain types of arguments for transcendental functions need not be in parentheses: for example sin 2x is sin(2x) and cos x sin y is cos (x) sin (y). The argument is determined by the following rules:

1. If the first item after the function name is parenthesized it is taken to be the entire argument.

2. If the first item after the function is not parenthesized, the argument is terminated by any infix operator except division (or implied multiplication) or by another transcendental function name. (See items 30-37 in table A for example.)

Examples of the above rules are shown in Table A, Ex. 24-42.

5. <u>Other Notation</u>

(a) +, -, X, / are used in the usual fashion. + and - have lower precedence than X and /, which in turn have lower precedence than implied multiplication and functional operators. However, the input processor is not strictly "precedence-driven"; there are too many special cases.

When several infix operators of the same precedence occur in sequence, grouping (i.e., algebraic interpretation) goes from left to right. So "a/b/cXd" is stored as ((a/b)/c) X d. (But note the special rules for numerical fractions). See Table A, items 27-29, 31-33, 37-42, 50, for examples.

(b) $\rightarrow$ is the symbol used to input transformations (see Section III-B) Technically, $\rightarrow$ is a binary infix operator whose operands are both expressions, or both equations. The pair of symbols $\leftrightarrow$ is used as a single operator to denote a transformation which may be performed in either direction (See Ex. 19 in Table A)

(c) Spaces are generally ignored by the input processor, but are needed to input mixed numbers.

(d) Expressions with unpaired parentheses are illegal. However, expressions of the form (...], {...), etc., will be accepted, and an appropriate comment made to the user.

C. User-Defined Notation

An important feature of the system is the ability to add mathematical functions and operators.* The method by which this is done on-line at the console is discussed in Section V-P and in Appendix B of this manual. The following is a discussion of the usage (i.e., input rules) of these "new" mathematical operators.

The input form of a new operator must be a single keyboard symbol. Thus, if the user wishes to add the determinant function to the system, he cannot input "Det" for this function (although he may have it displayed in this form). But any symbol which has no other special meaning may be used as a new operator. Function names need not be letters.

* Not to be confused with push-button operators.

Table B contains a list of the symbols which may not be used as new operators.

The symbol for the new operator must be placed into one or more (<u>any</u> combination) of the six lists in Table C. The first three lists provide for what might be called "operator" notation, and the last three are for "functional" notation. The examples in Table C should make these concepts clear, as they correspond quite closely to normal usage.

To clarify these ideas, let us consider a specific example: suppose the user wishes to make "f" a function of one variable, to be used in the standard notation " f (...)". To do this, he just adds the symbol "f" to the Unary Function List. If he decides that he would also like to be able to use "f" as a function of two or more variables, he can add the symbol to the Binary Function List and/or the Many-Place Function List. When using such functional notations, he must always parenthesize the argument(s), and must separate arguments with commas. Also, function symbols may have exponents, subscripts, and superscripts. (See Table A, numbers 13, 15) Another feature is that "f" may still be used as a variable, since it is a letter. Whenever it is written in an expression without an acceptable, parenthesized argument, the input-processor will assume it to be a simple variable (see Table A, nos 5 & 14; nos. 12, 16 and 17 also demonstrate functional notation).

Now suppose the user does not want to be forced to use parentheses. He can use notation like "f3x", just as he can write "sin 3x", etc., by making "f" a "Prefix Operator". (This type of notation is more commonly used with non-letter symbols, like $\sqrt{\ }$, etc.) He may still use "f" with a parenthisized argument, however use of operator notation is somewhat restricted. For example, "f(x,y)" is only legal if "f" is kept in the Binary Function List. A prefix operator may have an exponent, but no "scripts". An infix or suffix operator can have neither exponent nor scripts. Furthermore, no symbol in any of the operator lists may be used as a variable, even if it is a letter.

The feature of user-defined notation raises questions concerning the interpretation of ambiguous expressions. The input processor makes <u>most</u> of its interpretations on the basis of "precedence values". Where there is an ambiguity, the mathematical operator with the higher precedence is "performed" first. When the user adds a new function or operator, he assigns to it a precedence value. The input processor could use these values, however the current input processor assigns values automatically. The precedences are shown in Table D. The only other general rule is that in an expression like "a*b*c*d", where "*" is an infix operator, computation or grouping proceeds from left to right. See Table A, Nos. 32-51, for examples of interpretation of input. Parentheses may always be used to clarify the meaning of expressions.

V. Description of Pushbutton Operators and Their Use

The description of each operator includes the arguments required for its use and its results. As mentioned above all arguments are selected with the light pen or entered from the keyboard. The following commands are used for selecting equations or transformations on the scope from the keyboard:

.en - where n is an equation number
.tn - where n is the number of a transformation in a table.

Each such expression is terminated by a double carriage return (denoted here by //). All other information from the keyboard is assumed to be mathematical expressions.

A. Enter

Arguments: An equation or expression
Action: Enters its argument into the current scroll

B. Label Equation

Arguments: None
Action: Assigns the next label number to the current equation (beginning with 1)

C. Apply

Arguments: Operator, expression 1, ... , expression n
Action: The specified mathematical operator is applied to the expressions following and the result is entered into the scroll. The number of expressions following the operator must correspond to the normal number of arguments for that operator. Multiplication and

addition may take an indefinite number of arguments; subtraction, division, and exponentiation two arguments; most other operators take a single argument.

If an argument of an operator is an equation, the operator is applied to each side of the equation rather than the equation as a whole.

Examples of Use:
+ // a // b // c // <u>Apply</u>
will enter the result
a + b + c on the scroll
+ //.e1 //.e2 // <u>Apply</u>
will add equations 1 and 2 together and enter the sum on the scroll.
X //.e1 // 2 // <u>Apply</u>
will multiply both sides of equation 1 by 2 and enter the result.
Note that equations 1 and 2 in these examples could have been picked by the light pen in the appropriate order rather than selected by typing their numbers on the keyboard. The results are not simplified in any way.

D. <u>Apply Transformation</u>
Arguments: transformation, expression
Action: The expression is matched against the left part of the transformation to determine if the forms are similar and, if so, to determine the correspondences between the variables. A new expression is formed by substituting the corresponding value of each transformation variable into the right half of the transformation. This new expression is then entered

into the scroll. If no match is found the message "no match found" is displayed on the scope. If the transformation has been specified as going in either direction then the expression will be matched against both parts and the appropriate substitutions will be made.

E. Display Transformation Tables

By depressing the left foot switch and then releasing it, the display changes from normal scroll display to display of the transformation tables. This display is organized into pages which can be turned by picking the forward or backward arrows at the bottom of the display. Normal scroll display is continued by depressing the left foot switch again. Transformations may be picked from the display and used as arguments for other operations.

F. Restart

Restart returns the system to its state before the previous equation was formed, i. e. , it deletes the last result.

G. Cancel

Cancel deletes the last step the user made in setting up the arguments for an operation:

1. If the last step was to select an argument, that argument is deleted.
2. If the last step was an expanding pick, the level is contracted to the previous level.
3. If the last step was an inclusive pick, that pick is deleted.

H. Scroll Manipulation Operators

1. Rewind Scroll

   Returns the scroll to the starting equations.

2. Backup Scroll

   Shifts the display window back one equation.

3. Advance Scroll

   Shifts the scroll window forward one equation.

4. Unwind Scroll

   Places scroll window at current end of scroll.

5. Save Scroll

   Argument: (string for name)

   Action: Files the current scroll and assigns the given name to it.

6. Display Scroll

   Argument: (string for name)

   Action: Retrieves the named scroll and displays it in rewound position. Unless previous scroll was saved it is lost.

7. Start Scroll

   Action: Deletes current scroll (unless it is saved) and allows the user to start a new scroll.

I. Transformation Table Operators

1. Insert Information

   Argument: transformation

   Action: The selected transformations is inserted in the transformation table, and assigned the next higher number. A transformation may be formed by applying the transformation operator (↔) to two expressions or equations, or the transformation may be entered by the Enter operator. In the latter case the validity of the transformation is assumed.

2. Remove Transformation

   Argument: Transformation

   Action: The selected transformation is removed from its table and added to the end of the current scroll.

J. Delete

Argument: Expression which is itself a scroll entry.
Action: The selected expression is deleted from the current scroll.

K. Simplification Operators

1. Combine Terms
   Arguments: Expression, variable
   Action: Combines like powers of the specified variable within the selected expression. Elementary numerical simplification also takes place.
2. Combine Fractions
   Argument: An expression whose highest operator is addition or subtraction
   Action: The terms in the sum are combined into a single fraction with the least common denominator of all the fractions. If no member of the expression is a fraction or the highest operator is not subtraction or addition, no action is taken.
3. Simplify Products and Fractions
   Argument: An expression whose highest operator is multiplication or division.
   Action: The selected expression is transformed into a ratio of products, and is reduced to lowest terms.

L. Substitute

Arguments: An equation of the form expression a = expression b, an expression
Action: All occurrences of "a" in the selected expression are replaced by "b".

The equation "a=b" may be entered from the keyboard directly or may be picked from the scroll. The user may control the instances of "a" that are replaced, by picking a subexpression which contains only the desired instances.

M. Transpose

Arguments: An expression which is itself either one side of an equation or which is associated with all other expressions on one side on an equation by + or - operators.

Action: The expression is transposed to the other side of the equation and its sign is changed.

N. Procedure Define

Arguments: None

Action: The system records the definition of a direct procedure. The format of the definition is as follows:

```
Procedure Define        Pushbutton A B ... C
      { pushbutton operators and letters }
      { representing their operands      }
       END
```

Pushbutton is the operator button to which the direct procedure will be assigned. Since the unused pushbuttons are not labeled the user depresses the correct one to indicate which is desired. A, B, and C refer to letters on the keyboard which represent the arguments of the procedure. The following operators may be used in procedure definitions in addition to the normal operators:

1. A RESULT - indicates A is to be returned as the result of the procedure.
2. $\Rightarrow$ (store) - indicates that the result of the preceding operator is to be temporarily "assigned" to the letter following.

Literal mathematical expressions to be used as arguments of operators within a direct procedure definition should be enclosed in quotes (") to distinguish them from keys on the keyboard.

As an example let us define a procedure with converts an expression in cartesian co-ordinates (x and y) into the equivalent expression in polar co-ordinates, (r and θ)

Procedure Define	Pushbutton E
"x= r cos θ " E	Substitute $\Rightarrow$ A
"y= r sin θ " A	Substitute $\Rightarrow$ B
B DISPLAY	
END	

0. Edit

Argument: An entry in the scroll or the expression in the input area

Action: The argument expression is displayed in a linear format showing all characters including control characters, and a pointer is positioned before the first character in the expression. Using the commands described below the user may delete characters or insert new ones.

NCTE: The user must insure that any changes he makes through the edit operator are valid. All edited expressions are treated in exactly the same way as expressions entered from the keyboard, i.e., their validity is assumed.

Edit Commands:

1. The following commands are used to move the pointer:
   r - moves the pointer one character to the right.
   l - moves the pointer one character to the left.
2. Delete Characters
   d - deletes the character following the pointer.
3. Insert Characters
   i - the characters following the i are inserted before the pointer; the characters after the pointer are

shifted right. The insertion process is terminated after an upper case followed immediately by a lower case is typed.

4. End Editing

   z - The edit operator is terminated, the edited expression is re-analyzed by the input processor, and the new version replaces the old.

P. Add New Operator*

Argument: None

Action: This operator allows the user to add certain types of mathematical operators to the system. Because the information required by the system is quite detailed the system will ask the user questions requiring the user to choose between several alternatives, or to answer yes or no. The information requested is of the following types:

1. Input symbol
2. Input format (number of arguments,prefix, infix, etc.)
3. Mathematical properties (commutative, etc.)
4. Display format (which may be different from input format because of input limitations.)

* A more detailed description of this operator can be found in Appendix B.

Appendix A: Examples

This appendix presents two examples using the system to solve mathematical problems. In the examples we show the commands required to execute each step and the resulting equation, which is added to the scroll. We use the following notation to represent the user's commands:

// -	double carriage return (standard terminator)
"A" -	the expression in quotes has been picked with the light pen from the last equation and selected as an argument.
<u>B</u> -	B is one of the pushbutton operators described in section V.

For reference, the following transformations are used in the examples. They will be referenced by their numbers:

$$a(b + c) \rightarrow ab + ac \quad (1)$$
$$ab + ac \rightarrow a(b + c) \quad (2)$$

1. Solution of a linear equation in one variable

User's command		Resulting expression
$5z - 1/5 = 3(z + 13/15)$ //	<u>Enter</u>	$5z - \frac{1}{5} = 3(z + \frac{13}{15})$
.t1 // "$3(z + \frac{13}{15})$"	<u>Apply Transformation</u>	$5z - \frac{1}{5} = 3z + \frac{(3)(13)}{15}$
"$\frac{(3)(13)}{15}$"	<u>Simplify Product</u>	$5z - \frac{1}{5} = 3z + \frac{13}{5}$
"$3z$"	<u>Transpose</u>	$5z - 3z - \frac{1}{5} = \frac{13}{5}$
"$\frac{1}{5}$"	<u>Transpose</u>	$5z - 3z = \frac{13}{5} + \frac{1}{5}$

"$5z - 3z$"	<u>Simplify Sum</u>	$2z = \frac{13}{5} + \frac{1}{5}$
"$\frac{13}{5} + \frac{1}{5}$"	<u>Simplify Sum</u>	$2z = \frac{14}{5}$
$/\!/$"$2z = \frac{14}{5}$" $2/\!/$	<u>Apply</u> (ie. divide by 2)	$\frac{2z}{2} = \frac{14}{(2)(5)}$
"$\frac{2z}{2}$"	<u>Simplify Product</u>	$z = \frac{14}{(2)(5)}$
"$\frac{14}{(2)(5)}$"	<u>Simplify Product</u>	$z = \frac{7}{5}$

2. Derive the formula for the addition of velocities in special relativity from the equations of the Lorentz Transformation. In the frame of reference A let a body be moving in the Z direction with speed U. Let frame of reference A be moving in the Z direction with a speed v relative to reference frame A'. What then, according to the Lorentz transformation is the speed w of the body relative to A'?

User's Command	Resulting Expression	
$z' = (z - vt)/\sqrt{}(1 - v\uparrow 2?/c\uparrow 2?)/\!/$ <u>Enter</u> <u>Label</u>	$z' = \dfrac{z - vt}{\sqrt{1 - \frac{v^2}{c^2}}}$	1.
$t' = (t - vz)/c\uparrow 2?/\sqrt{}(1 - v\uparrow 2?/c\uparrow 2?)/\!/$ <u>Enter</u> <u>Label</u>	$t' = \dfrac{t - \frac{v^2}{c^2}}{\sqrt{1 - \frac{v^2}{c^2}}}$	2.
$z = ut$ $/\!/$ <u>Enter</u> <u>Label</u>	$z = ut$	3.

$\omega = z'/t'$ // Enter Label $\qquad \omega = \dfrac{z'}{t'}$ 4.

.e3//.e1// Substitute $\qquad z' = \dfrac{ut - vt}{\sqrt{1 - \frac{v^2}{c^2}}}$

.t2// "$ut - vt$" Apply Transformation Label $\qquad z' = \dfrac{t(u - v)}{\sqrt{1 - \frac{v^2}{c^2}}}$ 5.

.e3//.e2// Substitute $\qquad t' = \dfrac{t - \frac{vut}{c^2}}{\sqrt{1 - \frac{v^2}{c^2}}}$

.t2// "$t - \frac{vut}{c^2}$" Apply Transformation Label $\qquad t' = \dfrac{t\left(1 - \frac{vu}{c^2}\right)}{\sqrt{1 - \frac{v^2}{c^2}}}$ 6.

/ //.e5//.e6// Apply $\qquad \dfrac{z'}{t'} = \dfrac{\frac{t(u-v)}{\sqrt{1 - \frac{v^2}{c^2}}}}{\frac{t\left(1 - \frac{vu}{c^2}\right)}{\sqrt{1 - \frac{v^2}{c^2}}}}$

" $\dfrac{\frac{t(u-v)}{\sqrt{1 - \frac{v^2}{c^2}}}}{\frac{t\left(1 - \frac{vu}{c^2}\right)}{\sqrt{1 - \frac{v^2}{c^2}}}}$ " Simplify Product $\qquad \dfrac{z'}{t'} = \dfrac{u - v}{1 - \frac{vu}{c^2}}$

Label $\qquad \dfrac{z'}{t'} = \dfrac{u - v}{1 - \frac{vu}{c^2}}$ 7.

.e7//.e4// Substitute Label $\qquad \omega = \dfrac{u - v}{1 - \frac{vu}{c^2}}$ 8.

Appendix B: System Description for adding a new operator

New mathematical operators are added by the user through this systems operator. The user presses the push-button before specifying any information, and the following action then occurs:

1. The system asks the user to type the operator or function symbol. The user in return types a single keyboard symbol (he may first hit "upper case" or "Greek"). The user should check that the symbol is allowable (see Table B).
2. The system next asks the user to specify the input category or categories into which the operator belongs. The user's response is one or more of the list names in Table C, typed on separate lines. The user indicates that there are no more categories to come by typing two carriage returns in sequence.
3. The system must then specify the information for the Operator Table and the display program. The system first asks the user in which of two "modes" he wishes to supply the details, "explanatory" or "abbreviated". The user answers this question by typing "e" or "a".

In either mode, the system must obtain the information necessary for the operator's "code word". In explanatory mode, the system asks the following questions (the questions appear in sequence on the scope):

0) Is operator crazy?
1) Is operator prefix?
2) Is operator infix?
3) Is operator suffix?
4) Must argument area be parenthesized?
5) Must argument area not be parenthesized?
6) May operator have 5 or more arguments?
7) May operator have exactly 4 arguments?
8) May operator have exactly 3 arguments?
9) May operator have exactly 2 arguments?
10) May operator have exactly 1 argument?
11) Is operator a "string" operator?

12) Is operator associative?
13) Is operator commutative?
14) What is precedence value of operator (0 to 15)

The user answers all but the last question by typing "y" or "n".

Many of these questions seem to ask for the same information that was supplied in the input specifications but actually this is not so, since these questions refer to the output specifications, which may be entirely different from input.

The following is an explanation of these questions, by number:

0) Is operator crazy?

For display purposes, every operator is called either "crazy" or "normal". An operator is normal only if the following conditions are met: display for this operator involves only display of the input symbol, with no special control routines, in some combination (any 0,1,2 or 3) of the categories prefix, infix or suffix. Functional notation (i.e., parenthesized operand area and commas between operands, for a prefix or suffix operator) is <u>normal</u>. But any other complex display procedure makes an operator crazy.

1-3) Is operator prefix/infix/suffix?

An operator is prefix, infix or suffix if the display program must perform some action before, between, or after the arguments, respectively. In the case of a "normal" operator, this "action" is simply the display of the input symbol. In the case of a "crazy", "string" operator, the action is the display of a specific string of symbols (see question 11) An operator need not be in exactly one of these categories. Exponentiation, for example, is infix-suffix, since "action" (i.e., line-changing) must be done before and after the operand. See Example 3 for a useful prefix-suffix possibility. Implied multiplication, on the other hand, is in none of these categories.

The display program automatically places commas between operands of a prefix or suffix operator (unless it is also infix). Thus, to effect functional notation, an operator should be described as normal, prefix and "yespars". (see next question).

4) Must argument area be parenthesized?
Used mainly for functional operators. Note that "operand area" means neither the individual operands nor the entire expression (i.e., we want to see "f(x,y)", not "f(x),(y)" or "(fx,y)" (See example 1)

5) Must argument area not be parenthesized?
Some operators, especially prefix-suffix, like the standard notation for absolute value (example 3) should never have a parenthesized argument area. (We do not want "|(...)|").
Most non-functional operators would have both questions 4 and 5 answered No.
Questions 6-10 and 12-13 ask information which is used by algebraic programs as much as by the display program .

6-10) How many arguments may the function or operator have?
An operator may, of course, be in several of these categories. (For multiplication, for example, the answer would be Yes to all but the last question).

11) Is operator a string operator?
This question is asked only for crazy operators. A "string" operator is one for which the only "craziness" is that a string of one or more characters is displayed instead of the symbol used for input. Example 2 shows how the determinant function may be defined to effect the display of "Det (A)", even though input is of the form D(A). In Example 3, the absolute value function must be added as a string operator since a vertical bar may not be used for input. Here the "string" is the single symbol "|".

12) Is operator associative?
This question is not asked if the operator cannot have more than two arguments.

13) Is operator commutative?
This question is not asked if the operator is only unary.

14) What is precedence value of operator?
The user answers this question with a number from 0 to 15.

At present, these values are used by the output program to determine where parentheses are required. They will also be used by the input processor to interpret ambiguous constructions, especially those involving infix operators. Table D contains the precedence values of some of the permanent system operators.

The abbreviated mode allows the user to specify this operator information more quickly. In this mode, the system simply asks for the operator's output specifications. The user must know exactly how each of questions 0-13 would be answered. For each question which would be answered Yes, he types the appropriate word (see Table E). He types these words one to a line, in any order. For a question which would be answered No, he does nothing. After typing all the necessary words, he types the precedence value on a separate line, and then types a double carriage return to inform the system that all the information has been given.

If the operator is not crazy, the system now has all the input-output information it needs, and the operator is ready for use in the system. Otherwise, there is still more information to supply. If the operator is "string", the system simply asks for the string of characters to be displayed for the operator, which the user then types. It is suggested that in most cases the string should begin and end with a space, to aid visibility. The user terminates the character string by typing a double carriage return, and the operator is ready for use.

If the operator is crazy and not of the string variety, complete display specifications may be quite complicated. The system asks various questions, and the user describes the display format at the keyboard and/or by using the light pen and the scope.

Example 1:

The user wishes to use the symbol "f" in standard functional notation, as a function of two or more variables.

The following is a simplified version of the dialogue that might occur:

Information Required	User's Response
Input symbol:	f
Input list(s):	twofl
	mulfl
Mode:	a
Display Description:	prefix
	yespars
	multops
	10

Since the operator is normal, this is all the information that is asked. If the user had chosen explanatory mode, his answers to questions 0-10 and 12-14 would have been, respectively:

n,y,n,n,y,n,y,y,y,y,n,n,n,10

Example 2:

The user wishes to add the determinant function to the system. Display is to be of the form "Det A". This is a typical "string" operator. For input, let us say the user wishes to write "DA". The dialogue would then be:

Input symbol:	D
Input list(s)	popl
Mode:	a
Display Description:	crazy
	string
	prefix
	unary
	12
Display string:	Det

If the user had wanted to see this operator displayed with parenthesized arguments in all cases, he could have included "yespars" in the display description. Note that he does not have to make it "nopars" to see "Det A".
For input purposes, if the user had specified the "onefl" input list instead of "popl", he would have to input "D(A)"; "DA" would not be understood.

Example 3:
The absolute value function is added to the system. To be displayed in the usual way, it must be made a "crazy" operator; but it fits very conveniently into the "string" category as a "prefix - suffix" operator. Assuming the user wants to input "a(x)" for $|x|$, the conversation runs as follows:

Input symbol:	a
Input list(s):	onefl
Mode:	a
Display Description:	crazy
	string
	prefix
	suffix
	nopars
	unary
	14
Display String:	1

Appendix C: Proposed Extensions to the System

The system could be extended in several ways to:

1. Allow more manipulations to be performed automatically while keeping extensive control in the user's hands.
2. Allow richer forms of mathematics to be handled.
3. Allow numerical evaluation, computation and graphical display of results.

Particular extensions toward these goals might be:

1. Explicit declaration of various types of mathematical quantities - e.g., real variable, complex variables, vectors, matrices, etc. At present no declarations are necessary and the user must keep the nature of his variables in mind. Without declarations the system would be unable to distinguish transformations that applied only to vectors or matrices, for example.
2. Explicit function definitions
   In the present system a function may be defined (i.e., the rule for its evaluation specified) in two ways:
   1. as an equation entered in a scroll or
   2. as a transformation.

Since there may be a number of equations of the form "f(x) = ..." the present system provides no way for uniquely defining a function. In addition each of these ways of definition is restricted to those functions that have a closed form definition. However, many functions of interest cannot be defined in this way. The following methods of function definition are often useful:

a. closed form - a formula describing the value of the function in terms of dummy variables
   ex: $f(x,y) = x^2 + 3xy + 3y^2 + 4$

b. functional - the value of a functional operator applied to other functions
   ex: f' is the result of the application of the differentiation operator to f.
   h might be defined as $f \cdot g$ meaning
   $h(x) = f \cdot g(x) = f(x)\, g(x)$

c. numeric or tabular - a table giving the value of the function for a limited set of arguments, the remaining values computed by interpolation when needed. This is the representation used in the Culler-Fried System; it provides a very fast means for displaying graphs and curves, and allows powerful operations to be performed easily. For example differentiation is accomplished by taking the difference between adjacent pairs of values. The intersection of two curves may be found by applying the delta function to their difference.

d. operational - an algorithm (i.e. a computer program) that computes the value of the function for any argument in its domain.

An important use of such function definitions lies in numerical evaluation of complex expressions. Often a mathematician will introduce auxiliary functions during his derivations to keep his expressions simple. When he wishes to evaluate his final expression, explicit definition of his auxiliary functions allows the system to carry out the evaluations automatically.

Many functions may be defined by several of these types. In fact, the user should be able to easily change the definition to suit his needs. To investigate the mathematical properties of a function the user might prefer the closed form definition. To examine its shape or its relation to other functions, he might shift to the numerical definition. The operational definition might be appropriate for doing straight computation where speed and accuracy are important.

3. Application of logical constraints to expressions. Often an expression or equation is valid only for certain ranges of its variables. These constraints or side conditions often become very complex (or are forgotten). The conditions could be carried along during derivations and modified as new conditions are added. They may also be used to prevent fallacious reasoning (e.g. dividing by an expression which may be simplified to be zero, or evaluation a derivative at a point where it does not exist).
4. Extension of the procedure definition language to handle conditional and branching logic. In the present system the user may define a procedure as a sequence of existing operators. Currently complex operators involving loops and operations dependent on the ranges of variables must be programmed off-line in Decal-BBN. A more powerful procedure language would allow algorithmic operators such as polynomial factoring, symbolic differentiation, or the Euclidean algorithm to be developed on-line. Furthermore, these operators could be used by other operators.

5. Development of more powerful simplification operators. The notion of simplification is extremely ambiguous; it varies from problem to problem as well as from user to user. For this reason no single simplification operator would be satisfactory. The following scheme seems to offer both flexibility and power: The simplification operator would be written to work with transformations in several transformation tables. The operator would apply those transformations that were relevant and ignore the others. By changing the contents of the various tables the user could control the simplifications that would be made automatically. In addition he could add new tables as he developed results that would be useful in simplifications.

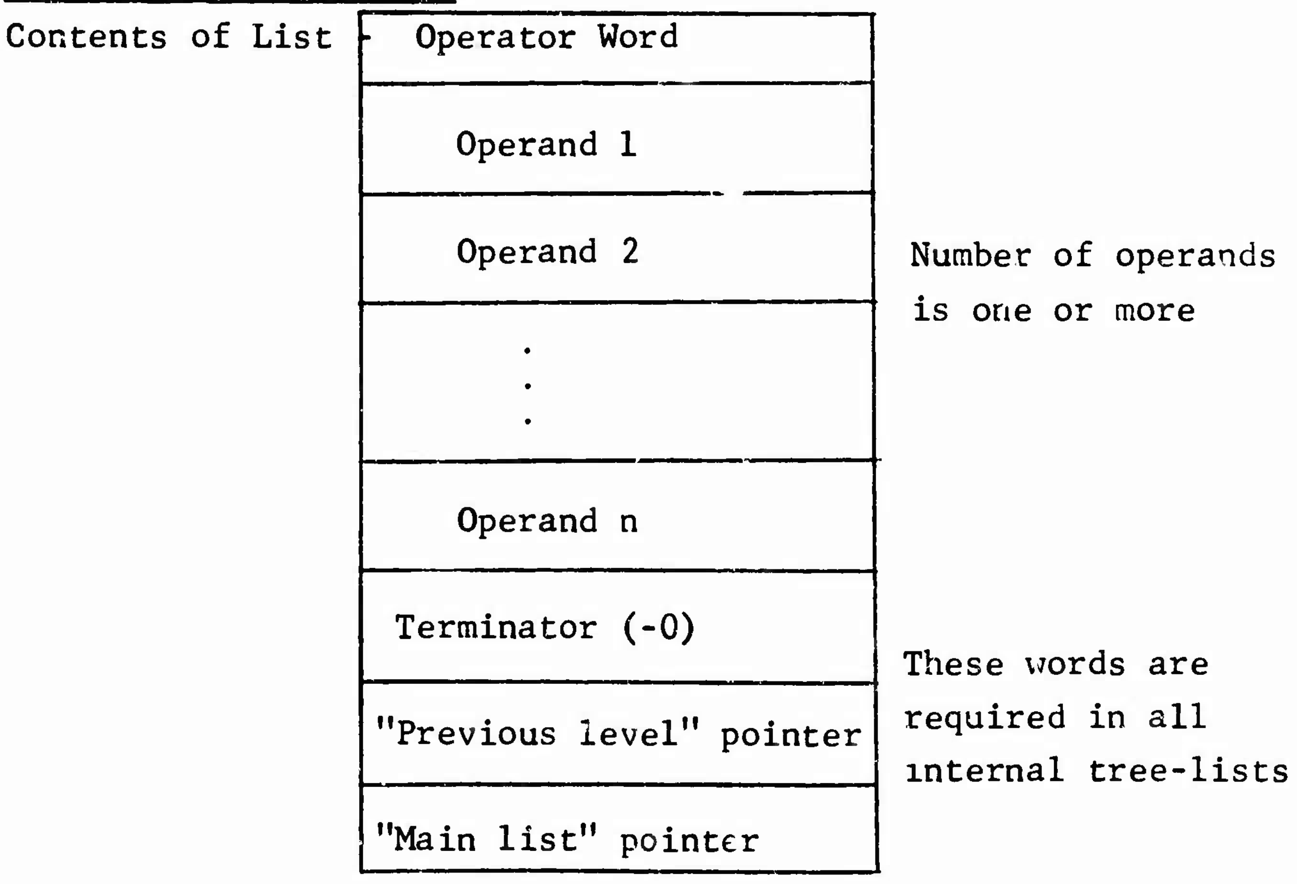
6. Use of external input. The user may define the values of his variables with data obtained from an analog to digital converter or another program. Thus the user may define one of his functions to be a section of the output of a speech sample. Similarly a least-squares technique could be used to fit a curve to incoming data so that it may be analyzed analytically.

Appendix D: Internal Format of Mathematical Expressions

I. Introduction

All mathematical input, including expressions, equations and transforms, is stored internally in tree structures. That is, information is stored in a list, in which one or more of the items may be a pointer to another list (or "branch"), etc. The first item in any branch specifies a mathematical operator, and the following item(s) specify the operand(s). If an operand is at all complex, it will be stored as a pointer to another list, which may consist of an operator followed by its operands, etc. So, the storage tree contains one list for every operator in the input.

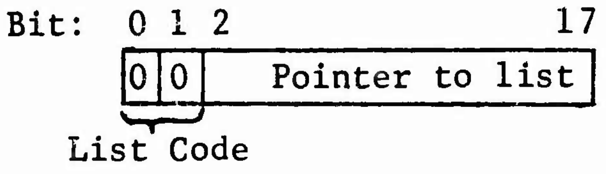
II. Branch Format in Trees:



-0 automatically terminates the operand area. The previous level pointer (which will be abbreviated "bptr", for "back-pointer") specifies the list in which this list is an operand. If this list is itself the main list of the tree (i.e., not an operand), its back-pointer is -0. The main list pointer (which will be abbreviated "mptr") specifies the top or main list of the tree. The mptr. of the main branch is not -0, even though the bptr. is.

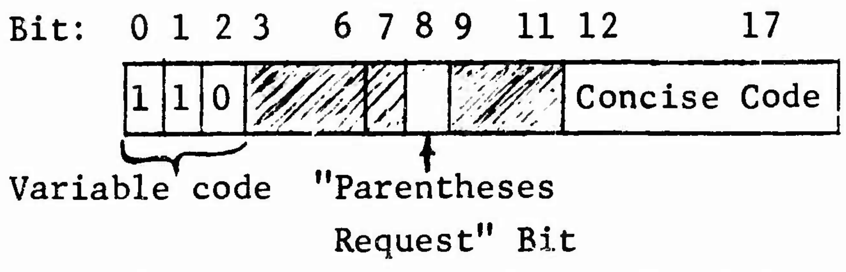
III. Format of Operand-Words:

Case A: Operand is a sub-branch:

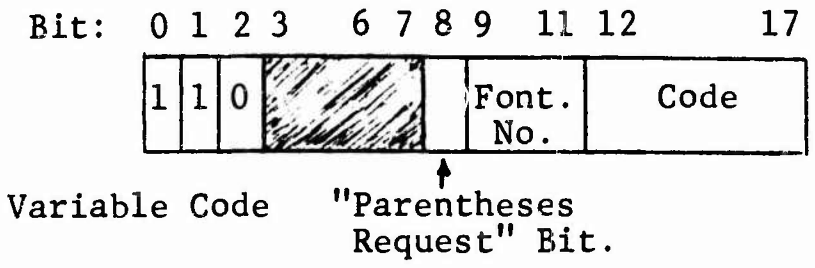


Case B: Operand is a variable:

i) Variable is lower-case English:



ii) Variable is a character from a Special Font:



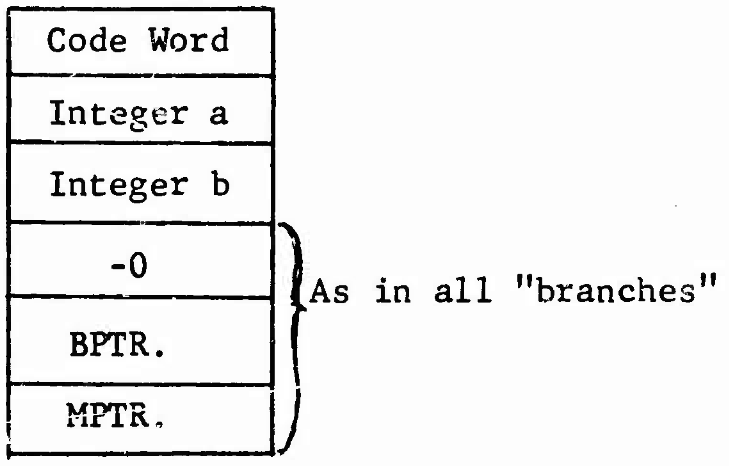
There are seven special fonts, some of which contain characters which may be used as variables. The English upper-case alphabet is in Font 1, and the 6-bit code for these letters is the concise code. Thus, the operand word for the variable "X" is 600127. Greek lower-case is found in Font 2.

Bit 8 (1000_8) is set when the user requests to have a variable displayed with parentheses. Bit 7 is used for similar information in operator words, and should not be set in variable words. The function par (word) may be used to mask out bits 7-8 in a word.

Case C: Operand is a number:

Bit:	0	1	2	
	1	1	1	Pointer to Number List

Every number in an input string generates a number list with the following format:



a and b are 18-bit positive integers whose meaning is specified by various bits in the code word:

1. SIGN: Bit 0 is on if the number is negative. All other information pertains only to the absolute value of the number.

2. Decimal, Fractional or Floating Point
   - i) Bit 14 (10_8) is set for decimals (the number is a.b.) (b is a binary decimal number)
   - ii) Bit 13 (20_8) is set for fractions (the number is a/b)
   - iii) Bits 13 and 14 (30_8) are set for a floating-point number, a and b are in the 28-bit floating-point format.

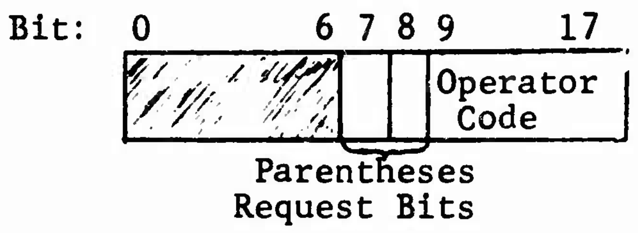
   At least one of these two bits must be on! Every number is originally stored in decimal or fractional form. Integers written without a decimal point are stored as fractions. Floating-point representation is set up and used exclusively by internal manipulative programs.

3. Useful Algebraic Information
   - i) Bit 12 specifies that the number is integer-valued. (in a decimal, b=0; in a fraction, b=1).
   - ii) Bit 11 specifies that the number is zero. (in a decimal, a=b=0; in a fraction, a=0).

4. Display Information
   - i) Bit 10 requests display as a mixed number (must be fractional).
   - ii) Bit 15 requests that the number be displayed with a fraction-line or a decimal point, even though it is an integer.
   - iii) Bit 16 requests a display of:
     "0.b", rather than simply ".b" or
     "a.0", rather than simply 'a.".
   - iv) Bit 17 causes zeros to be displayed at the end of the decimal part - they would normally be suppressed. (e.g., the user may wish to see "2.300000", instead of "2.3", in a list of 6-place decimals.)

5. Parentheses
Bit 8 requests that the number be displayed parenthesized, as with variables. Again, bit 7 should never be set.

IV. Format of Operator-Words



Bits 7 and 8 govern the display of the "sub-expression" consisting of the operator and its operands:

00 - no parentheses around expression, unless necessary for clarity of meaning.
01 - parentheses around expression, unconditionally
10 - brackets " " "
11 - braces " " "

The ig par(op) masks out bits 7-8 of a word

The operator code is determined as follows:

A. Operators inputted as single keyboard symbols are stored much like variables. Bits 12-17 contain the concise code, but 11 is set for upper case, and bit 10 for Greek letters and certain symbols.

At present, the following are "permanent" operator-symbols:

Symbol	Octal Code	Meaning
Sub	32	Subscript
Super	37	Superscript
$\uparrow$	111	Exponentiation
$\rightarrow$	120	"Is replaced by" (Symbol for a Transform)
+	154	
-	54	
X	173	

Note that, internally, "minus" exists only as a unary operator, while division does not exist at all.
["a-b" is stored as "a+(-b)"; "a/b" as "ax(b^{-1})"]

The user may decide to make almost any other keyboard symbol into an operator. To do this, he must place the symbol in one or more of the following lists:

Name	Meaning	Example
popl	Prefix-operator list	$\sqrt{x}$
iopl	Infix-operator list	x * y
sopl	Suffix-operator	x'
onefl	Unary Function list	f(x)
twofl	Binary Function list	β(x,y)
mulfl	Many-place Function list	Q(x,y,z,w)

The coding system for such operators is the same as that for variables and "permanent" operators.

B. "Special operators" include sine, cosine, etc. These are inputted verbatim but are stored as if they were single characters in the lower half of Font 4. They are also contained in the Prefix-Operator List. Specifically, the codes are:

Operator	Octal Code
Sin	401
Cos	402
Tan	403
csc	404
sec	405
cot	406
$\log_{10}$	407
ln	410

Note that the product of the variables s, i and n can be inserted into an input string in order by simply leaving a space in the middle of the word.

C. Certain operators are called "internal". That is, they are not written in input but are set up automatically in storage. These operators have codes in the upper half of Font 4. They are:

Symbol Representation	Octal Code	Name and Meaning
$\otimes$	441	"Implied Multiplication"
XX	442	"Doublescript" (superscript followed by subscript; has 3 operands)
,	443	"Comma"; denotes a superscript or subscript with several items separated by commas.
	444	"Singleton" This is used when input consists of a single variable or number. The variable or number is made the operand of a list whose operator, "singleton", is meaningless (like "+" followed by a single argument).
$\uparrow$ -1?	445	"Reciprocal"; an abbreviation for the exponent -1. THIS OPERATOR IS ONLY BEING USED TEMPORARILY
$\leftrightarrow$	446	"Double Transform"; used to indicate a transform which may be performed in either direction.

Appendix: E List of System Programs

Name	Purpose
Push, pull, tuck, tug pap	List insertion and removal routines
copypointers	To create an alias of a list
reset	Reset pointers to empty condition
typeinput	Inserts string of characters from typewriter into a list
get, giveup	Assign and maintain free list area
botcopy	Copies a list into the bottom of another
topcopy	Copies a list onto the top of another
copy	Copies a list into another
stuff	Inserts a word into a given numerical position in a list
yank	Reads the word in a given numerical position in a list
switch (n)	Skips the n following registors
tuckem	Tucks the following words into the list indicated

Name	Purpose
markbits	Marks a list item as follows: bits 0-1 00 01 11 10
fixbits	"does just that" (internal to isom) (undoes markbits)
typerr	Types the contents of a list called typlist (rightmost character). Then halts
Syntax analyzer (ssv)	Analyze input string according to syntax definition and produce a tree structive transformation
Transplant	Transform marklist output from Syntax analyzer into standard tree structure
Treechop	Transforms tree structure into character stream for display
mapcount	Finds place in tree corresponding to character number returned from VIP light pen identification
brtcntrl	Control brightening operations in response to light pen action
procdef	Interprets input characters as a direct procedure and adds it to the button table
transpose	Transposes a term

Name	Purpose
subadimal	Subtracts, add, divides, or multiplies an equation by an expression
eqdams	Divide, add, multiply or subtract two equations
combterm	Simplifies a summation by combining terms, canceling a-a, etc.
prodsimp	Simplifies products of terms, fractions, etc.
combfrac	Combine fractions into one fraction
transapply	Apply a transformation
subst	Substitute an expression into an equation
subst 1	Sub-routine of subst that takes its arguments in the form of an association list
intcntrl	Handles control of user interface, device, selects procedures to be called
eqinput	Handles input from soroban and displays typed characters for feedback

Name	Purpose
gotvip	goto VIP
plusimp 1	Removes nested and redundant plus operators from a tree
plusimp 2	Does numerical addition on top level of sum
mulsimp 1	Changes products of quotients into quotient of products
mulsimp 2	Concatenates nested and redundant multiply operations
mulsimp 3	Combine numerical factors
backptr	Finds backpointer of a list
eqname	Finds equation name of list
copytree	Makes a copy of a tree
match	Determines if two expressions are equivalent, (except for commut. and other operations)
tarzan	Given a branch in a tree, finds the corresponding branch in a copy of the tree
all insts	Finds all instances of a given expression in an equation

Name	Purpose
giveuptree	Returns a tree to free list area
signch	Negates an expression
pbackptr	Places a back pointer in a list
peqname	Places an equation name in a list
remove	Removes an operand from a list
replace	Replaces one operand with another
opdlist	Forms a list of operands, i.e. without operator and other pointers
categ	Determines whether an operand is a simple variable, a number, or another expression
copyopd	Generates a copy of an operand
copyev	Same as copyopd but inserts backpointer and eqname if operand is a list
topitem	<u>IG</u> find top item of a list
botitem	<u>IG</u> finds bottom item of a list
oppar	<u>IG</u> masks out operator - parenthesis bit
varpar	<u>IG</u> masks out variable parenthesis bit

Name	Purpose
conint (inlist)	Procedure Transforms list of digits into a single number
distab	Display a transformation table
editab	Edit a transformation table - i.e., add transf, delete transf., and place a transf. on scroll
formtrans	Form a transformation from two picked expressions
transbut	Assign a transformation to button table
procbut	Assign a procedure to button table
cancel	Undo last user step
restart	Return to status after last equation added to scroll
dispex	Call treechop, VIP, and add equation to scroll
backscroll	Rotate scroll one position backward
forscroll	Rotate scroll one position forward
rescroll	Rewind scroll
discroll	File current scroll and display new one

Name	Purpose
deleq	Delete equation from current scroll
enscratch	Transfer equation to scratch pad
label	Assign next eq. number
reset scroll	Return scroll to end where next equation to be added
call	Handles paging and sets up procedures for execution
control	Monitors the user's console and controls the system
opinfad	Subroutine of Treechop - contains the operator table. Given operator symbol, returns with format word from table

TABLE A

EXAMPLES OF THE INPUT LANGUAGE*

No.	Input Form	Interpretation
1.	0.000	0.000
2.	.1	$.1$
3.	1.	1
4.	2 7/8	$2\frac{7}{8}$
5.	α	α
6.	$x \wedge 2 ?$	$x^{(2)}$
7.	$x \vee 37 \uparrow y ? + \cos g \vee 2 ??$	$x_{(37^{y} + \cos g_{2})}$
8.	x 463 1/19	$x_{463\frac{1}{19}}$
9.	$\beta \wedge 6 ? \vee 2, 4, r + \ell\gamma ?$	$\beta^{(6)}_{(2,\, 4,\, r+\ell\gamma)}$
10.	$x 392 \uparrow e \uparrow i \theta ??$	$x_{392}^{e^{i\theta}}$

* In the examples, $\vee$ denotes a subscript; $\wedge$, a superscript; and $\uparrow$, exponentiation. Also, f and α denote user-defined functions; while * , and ' denote a prefix, an infix, and a suffix operator, respectively.

TABLE A Page 2

No.	Input Form	Interpretation
11.	(a + sin x)↑2 1/2 ?	$(a + \sin x)^{2\frac{1}{2}}$
12.	3.307↑−f(x) ?	$3.307^{-f(x)}$
13.	f↑x ? (a+b)	$f^{x}(a + b)$
14.	f↑2?x	$f^{2}{}_{\times}\, x$
15.	fV3+g?↑2?(2/3)	$f_{3+g}^{2}\left(\frac{2}{3}\right)$
16.	f(a + b + c, cos α)	$f(a + b + c, \cos\alpha)$
17.	f(1, 2, 3, 7 1/2)	$f(1, 2, 3, 7\frac{1}{2})$
18.	(a + b)↑2?↑y?	$\left[(a + b)^{2}\right]^{y}$
19.	(a + b)↑2?←→ a↑2? + 2ab + b↑2?	$(a+b)^{2} \longleftrightarrow a^{2} + 2ab + b^{2}$
20.	xyz	xyz
21.	x↑7?(a+3)↑6 ? f(l,m)	$x^{7}(a+3)^{6}f(l,m)$
22.	3.x3y33	$3\, x_{3}\, y_{33}$

TABLE A Page 3

No.	Input Form	Interpretation
23.	x2 1/4 y	$x_{2\frac{1}{4}}y$
24.	sin xy	sin (xy)
25.	sin ↑ 3 ? (x + 17y)	$\sin^3 (x+17y)$
26.	sin log 3xy	sin (log(3xy))
27.	xy sin 2.2 cos g	xy (sin 2.2)(cos g)
28.	−3.7 ↑ −1 ? xy3 log 4 1/2	$-3.7^{-1}\, xy_3 \log(4\frac{1}{2})$
29.	3 cos g * ab * (x+y) (here * is "user's infix operator")	(3 cos g) * (a b) * (x+y)
30.	a * b × cos y × 2.2x	(a * b) × (cos y) × (2.2 x
31.	a/b + cos y − z + a	$(\frac{a}{b})$ + cos y − z + a
32.	s i n x	(s)(i)(n)(x)
33.	sin ab	sin (a b)
34.	sin a ± b	(sin a) ± b

TABLE A Page 4

No.	Input Form	Interpretation
35.	sin a×b	$(\sin a) \times b$
36.	sin a/b	$\sin\left(\frac{a}{b}\right)$
37.	sin 3x↑2?/y sin π	$[\sin(3x^2/y)] \sin \pi$
38.	cos x sin y	$(\cos x)(\sin y)$
39.	cos x f(y)	$\cos[x f(y)]$
40.	log sin a * b	$(\log \sin a) * b$
41.	√ -x √ -y sin-x	$(\sqrt{-x})(\sqrt{-y})(\sin(-x))$
42.	√ -x f(y)	$\sqrt{-x f(y)}$
43.	y/2x	$\frac{y}{2x}$
44.	2 1/2x	$(2\frac{1}{2})x$
45.	2x/2/y	$\frac{\frac{2x}{2}}{y}$

TABLE A Page 5

No.	Input Form	Interpretation
46.	$1/2x$	$(\frac{1}{2})x$
47.	$a * b * c$	$(a * b) * c$
48.	$a/b + c$	$\frac{a}{b} + c$
49.	$a + b/c$	$a + \frac{b}{c}$
50.	$a * b - 1$	$(a * b) - 1$
51.	$\surd\ x'$	$\sqrt{(x')}$

TABLE B

Keyboard Symbols which may not be used as New Operator Symbols

+	(	$\uparrow$	.	"Upper Case"
-	)	"Super"	,	"Lower Case"
X	[	"Sub"	"Space"	"Greek Mode"
/	]	?	"Carriage Return"	"English Mode"
=	{	$\downarrow$	"Tab"	"Panic"
$\rightarrow$	}	$\leftarrow$	"Backspace"	0-9

TABLE C

Lists for New Operators and Functions

Name	Meaning	Example
Popl	Prefix-operator List	x
Iopl	Infix-operator List	x * y * z
Sopl*	Suffix-operator List	x'
Onefl	Unary Function List	f(x)
Twofl	Binary Function List	β(x,y)
Mulfl	Many-Place Function List	Q (x,y,z,w) 3 or more arguments

*The current version of the system does not accept suffix operators.

TABLE D - Precedence Values of Various Operators

Operator (s)	Precedence Value
Exponentiation	14
Superscript, Subscript, Implied Multiplication	13
Sin, cos, etc.	12
x, /	9
+, -	7
=	4

For user's functions, precedence value is irrelevant because the argument(s) is always in parentheses. For user's operators, the input processor ignores the user-defined value and assigns the following values automatically:

Prefix operators	12
Infix operators	10
Suffix operators	15

At the present time, the user-defined precedence values are relevant only during display operations for proper arrangement of the arguments in a mathematical expression.

TABLE E - Words for Describing Operators in Abbreviated Mode

Question No.	Descriptive Word
0	crazy
1	prefix
2	infix
3	suffix
4	yespars
5	nopars
6	unlimops
7	4 ops (<u>or</u> 4ops)
8	3 ops (<u>or</u> 3ops)
9	2 ops (<u>or</u> 2ops) <u>or</u> binary)
10	1 op (<u>or</u> 1op) <u>or</u> unary)
11	string
12	associative (<u>or</u> asstv)
13	commutative (<u>or</u> comtv)

Precedence is specified by a number.

The following words may be used as an abbreviation for several words:

multops: has combined meaning of words 6, 7, 8 and 9
arbops : has combined meaning of words 6, 7, 8, 9 and 10.

UNCLASSIFIED

Security Classification

DOCUMENT CONTROL DATA - R & D

(Security classification of title, body of abstract and indexing annotation must be entered when the overall report is classified)

1. ORIGINATING ACTIVITY *(Corporate author)* Computer Research Corporation 429 Watertown Street Newton, Massachusetts	2a. REPORT SECURITY CLASSIFICATION Unclassified 2b. GROUP

3. REPORT TITLE

Magic Paper - An On-Line System for the Manipulation of Symbolic Mathenatics

4. DESCRIPTIVE NOTES *(Type of report and inclusive dates)*
Final Report

5. AUTHOR(S) *(First name, middle initial, last name)*

Lewis C. Clapp, Dale E. Jordan, Ellen J. Wax, Robert S. Wolf

6. REPORT DATE 15 April 1966	7a. TOTAL NO. OF PAGES 65	7b. NO. OF REFS None
8a. CONTRACT OR GRANT NO. AF 19(628)-5098 b. PROJECT NO. J-105 c. d.	9a. ORIGINATOR'S REPORT NUMBER(S) Report No. R-105-1 9b. OTHER REPORT NO(S) *(Any other numbers that may be assigned this report)*	

10. DISTRIBUTION STATEMENT

Distribution of this document is unlimited

11. SUPPLEMENTARY NOTES	12. SPONSORING MILITARY ACTIVITY Dynamics Processes Branch Data Sciences Laboratory Air Force Cambridge Research Labs. Hanscom Field, Bedford, Mass.

13. ABSTRACT

This report describes the preliminary version of the MAGIC PAPER System which is being developed by Computer Research Corporation for the Dynamic Processes Branch of the Data Sciences Laboratory at the Air Force Cambridge Research Laboratories. Through a conversational interaction, the system aids the scientist, engineer or mathematician as he performs symbolic operations on linear algebraic equations. The user begins by entering his initial equations and conditions through a mathematical keyboard. As he types these equations, they are displayed on a flicker-free scope in standard mathematical notation. Using a push-button control panel and a light pen, he may select expressions and operations which are to be performed on them. If the operation is legal, the system generates a new equation which is then added to the scope display. With the basic set of operations, the user may create new operators which can then be added to the system. He can also introduce special notational conventions. The user has considerable control which enables him to personalize the system to meet his own particular needs.

DD FORM 1473