MASSACHUSETTS INSTITUTE OF TECHNOLOGY
Cambridge, Massachusetts
Project MAC

Artificial Intelligence Project
Memo.85

Memorandum MAC-M-257
July 29, 1965

SYNTAX AND DISPLAY OF MATHEMATICAL EXPRESSIONS

by William A. Martin

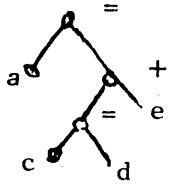
ABSTRACT

A LISP program converts a mathematical expression stored in list structure form, into a text-book style visual display. To do this, requires the selection and positioning of the individual symbols which make up the expression, using a combination of global and local information. This memo describes a table-driven picture-compiler which makes the necessary information available. Syntax rules have been written for a large class of mathematical expressions. These rules are simplified by the introduction of concepts concerning the relative position of symbols. In addition to the symbols and their coordinates the program sends a parsing of the symbols to the display. This program is a refinement of the system proposed by M.L. Minsky in Artificial Intelligence Memo.61, Mathscope; Part I.

I. Introduction

This memo describes a LISP program which computes the pictorial representation of a mathematical expression from a LISP S-expression representation of the expression. Since the language of mathematics contains a number of special cases, the computation will be introduced with a summary of the results produced by each part of the program and the general principles followed.

Because of the possible use of many transformations and identities, a mathematical expression can be represented in many equivalent forms. For instance a(b+c) could be represented as ab + ac. In certain contexts the picture of one of these forms may be easier to read than the picture of the others. For instance a - x might be preferred to a + (-1 · X). On the other hand, the use of several equivalent internal forms complicates the writing of routines needed to transform mathematical expressions. Furthermore, the most convenient internal form might be difficult to read. In the first pass of the picture compiler the internal form is transformed into an equivalent form which is easy to read. In addition, parentheses are added. The LISP expression resulting from this first pass has the form of a tree of subexpressions. Each node of the tree is a mathematical operation and the branches from that node are the arguments of that operation. At the ends of the branches are the individual variables and constants. The form of this internal tree is preserved throughout the remaining steps of the picture compilation. For example, the expression a = c · d + e has the tree:



Mathematical operations are represented in "pictures" by symbols placed in special arrangements. A classification of the representations will be given later. The situation is complicated by the fact that sometimes the same symbols and positional notations are used in different combinations to represent different mathematical operations. The second pass of the picture compiler rewrites the internal tree in terms of these common symbols and positions. The symbols and positions are expressed in terms of a set of special forms.

These forms are shown in Figure 1 through Figure 11. The compiler starts at the base of the internal tree and works out to the ends of the branches, rewriting the tree at each node. This rewriting introduces additional nodes into the internal tree, but these new nodes are marked so that the old structure is preserved. The display resulting from a node and its branches is call a picture part. Sometimes the form of a picture part depends on the dimensions of its arguments. In this case the second and third pass must be applied to the arguments before the second pass can be applied to this part.

In the third pass each picture part is inscribed in a rectangle. The compiler starts at the ends of the branches of the internal tree and works toward the base. The compiler first inscribes each individual constant and variable in a rectangle, defined by its width, height, and depth. A dimensioned rectangle is then chosen in turn for each larger picture part; i.e., operator with arguments. The dimensions of each rectangle depend on its own operator and also on the dimensions of those rectangles associated with the arguments of the picture part. In addition, we compute the relative positions of the arguments of each picture part with respect to the lower left hand corner of its circumscribed rectangle.

Once the entire expression has been dimensioned, it can be positioned on the oscilloscope face. The final pass of the compiler then sends to the display the name, size, and position of each symbol. These symbols are interspersed with non-displaying left and right pseudo-parentheses. These pseudo-parentheses group the picture symbols into a tree of sub-expressions identical with the internal tree structure obtained from pass one of the picture compiler. Thus, a light-pen reference to a picture symbol can be identified with the smallest subexpression in the picture tree which contains it, and with the corresponding LISP subexpression in the internal tree. This makes it possible easily to designate mathematically meaningful picture-parts as arguments for other operations.

I. Transformations to Facilitate Semantic Interpretation

A method is needed for referencing meaningful subparts of the LISP source expression by pointing with a light pen to symbols

in the displayed picture expression. Thus, before the picture is compiled, the source expression is transformed using mathematical relations into a form which makes it easy to establish a one-to-one mapping between subparts of the picture expression and mathematically meaningful subparts of the LISP source expression. Then, in addition to the picture symbols, the compiler sends the display a series of left and right pseudo-parentheses which parse the picture symbols into a tree identical with the tree formed by the transformed source expression. When the light pen is pointed at a picture symbol, the smallest picture subexpression which contains it is intensified and the corresponding LISP expression may then be referenced. The source expression is transformed in order to bring it into a better pictorial form, for instance, the order of the product is reversed so that $(\int_a^b cdx) \cdot d$ can be written $d \cdot \int_a^b cdx$.

Although the picture compiler is powerful enough to handle most all of the notations which arise in mathematics, compiler rules have been written for the mathematical expressions which arise in the solution of non-linear differential equiations. These expressions are represented by LISP S-expressions.* In brief, S-expressions are defined recursively as follows: Any number or string of alphanumeric characters is an S-expression. One or more S-expressions separated by spaces and surrounded by parentheses is an S-expression. Individual numbers and alphanumeric strings are called atoms. Individual variables and constants in the mathematical expressions are represented by atoms. The mathematical operators are represented by S-expressions consisting of the operator name, its arguments in some given order, and the atom NIL. NIL denotes the null S-expression; it is replaced by temporary results during the transformation of expressions and represents the property list of the mathematical operator. The individual mathematical operators currently in the program are:

(Note: most of the operators can take any number of arguments, in the obvious manner.)

1. (PLS A B C NIL) $\equiv$ A + B + C
2. (PRD A B C NIL) $\equiv$ A · B · C

* See LISP 1.5 Manual.

3. (FRT I J NIL) $\equiv \frac{I}{J}$

4. (PWR A B NIL) $\equiv A^B$

5. (DRV A B C D E NIL) $\equiv \frac{d^{B+D}}{dA^B dC^D} E$

6. (ITG D A B C NIL) $\equiv \int_A^B CdD$

7. (SUM A B C D NIL) $\equiv \sum_{A=B}^{C} D$

8. (EVL A B C D E NIL) $\equiv E \Big|_{A=B}^{C=D}$

9. (NAM A B C NIL) $\equiv C_{A,B}$

10. (F A B NIL) $\equiv F(A,B)$

11. (NAM A B (F C D NIL) NIL) $\equiv F_{A,B}(C,D)$

12. (FIL A NIL) $\equiv A!$

13. (ABS A NIL) $\equiv |A|$

Where A, B, C, D and E are arbitrary expressions, I and J are integers, and F is any atomic symbol not recognized as an operator. -X is represented by (PRD -1 X NIL) and $\frac{1}{X}$ is represented by (PWR X -1 NIL). If one of the factors of a sum orporduct is a number, it is often the left-most argument.

The first compiler pass performs the following transformations:

(PLS N A B NIL) $\longrightarrow$ (PLS A B N NIL)

(DRV S 1 (F S NIL)NIL) $\longrightarrow$ (DRV 1 1 (F S NIL)NIL)

-N $\rightarrow$ (NEG N NIL)

(PRD -1 A NIL) $\longrightarrow$ (NEG A NIL)

(PRD A (SUM A B C D NIL) E NIL) $\longrightarrow$ (PRD A E (SUM A B C D NIL)NIL)

(PRD A (PWR B -1 NIL)NIL) $\longrightarrow$ (DVD A B NIL)

(PRD A (PWR B (PRD _1 C NIL)NIL)NIL) $\longrightarrow$ (DVD A (PWR B C NIL)NIL)

(F A B NIL) $\longrightarrow$ (FUNCTION F A B NIL)

(PRD A (ITG A B C D NIL) E NIL) $\longrightarrow$ (PRD A E (ITG A B C D NIL)NIL)

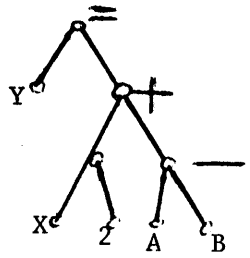
X $\rightarrow$ (ATOM X NIL)

(NAM X Y (F A B NIL)NIL) $\longrightarrow$ (FUNCTION (NAM X Y F NIL) A B NIL)

Where N is a number, X is a number or variable symbol, and S is a non-atomic subexpression

In addition to these transformations, levels of parentheses are inserted. For instance (PRD A (PLS B C NIL)NIL) $\longrightarrow$ (PRD A (PAREN (PLS B C NIL)NIL)NIL). No parentheses are used around the arguments of FUNCTION, PAREN, ABS, EVL or NAM. A size change always removes the need for parentheses. In addition, the integration and summation symbols stand in place of parentheses for operators on their left and a transcendental function is parenthesized if it is raised to a power. Otherwise, parentheses are placed around an argument of an operator if the main operator of the argument has lower precedence. Precedence is determined from the following list: PAREN, NAM, =, FUNCTION, FTL, PWR, FRT, PRD, =, DVD, ITG, DRV, NEG, =, PLS, =, SUM. Operators separated by = have equal precendence.

The expression resulting from these transformations is saved as the internal tree for the later analysis of light pen references. For example, the expression $Y = X^2 + \frac{A}{B}$ would be represented by the S-expression (EGN (ATOM Y NIL) (PLS(PWR (ATOM X NIL) (ATOM 2 NIL)NIL (DVD (ATOM A NIL) (ATOM B NIL)NIL)NIL)NIL). It has the tree structure.



II. Transformations to Facilitate Syntactic Construction

To a very large extent, mathematical notation becomes, in the mind of the individual, a model of the abstract concepts he is manipulating. For instance, consider the operation of canceling terms from both sides of an equation or from the numerator and denominator of an indicated division, or the operation of bringing a term to the other side of an equation, or the operation of matrix multiplication. Therefore, the notation must have properties which make it easy to visualize.

Lack of generality makes mathematical notation easier to read. Different mathematical operators should be represented by different sorts of notation and the mathematical operator with the highest precedence should be represented by the most compact form. For example, compare the equivalent Boolean expressions (((A U B) U C) U (D E)) and A + B + C + DE. Furthermore, when an operation is applied to complex arguments, it should be possible to visualize it in the same manner as when it is applied to simple ones. For instance, the introduction of parentheses or other such fences allows (A + B)C to be visualized in the same manner as DC, with the operation expressed as concatenation. Sometimes a change of size is used to aid in visualizing a complex argument as a single unit. As a final example, consider the choice of $\dot{f}$ to represent the drivative when it is to be thought of as an operation of high precedence, but the choice of $\frac{df}{dt}$ for the derivative when its properties as a ratio are to be used.

Although it is difficult to make strict classifications, it is useful to distinguish seven distinct pictorial forms used in mathematical notation. As stated above, the more compact forms are often used for operations of higher precedence with the exception that fences and size changes permit the use of complex arguments in the simpler forms; size changes allow complex forms to have high precedence.

The forms in rough order of decreasing precedence, with some examples of each are:

1. supersub $U_{1}{}'_{1}$: X^{2} : ${}^{2}T_{3}$

2. concatenation with variable size fence symbols and separation symbols in(x,y); $\left|X\right|$; $\binom{a}{b}$; (a)

3. concatenation 2X ; XYZ

4. binding symbol $\frac{a+b}{c}$: $\int_a^b xdx$; $\sum_{x-3}^{6} X$

5. hybrid $\frac{d^3f}{d^2xdy}$

6. concatenation with infix symbols x=y ; a + b + c ; A B ; A · B · C

7. title (E1) X

Since the same mechanisms, such as concatenation, are used in more than one of these seven forms, we will discuss the simpler ones first.

1. Concatenation

A rewrite rule may be associated with any operator. For example, if concatenation is used to represent multiplication then there might be a rule:

(PRD A B NIL) → (CONCAT A B NIL).

The computation is simplified if concatenation is considered to be a binary operator. It is therefore necessary to have recursive rules such as :

(PRD X_1 X_2 --- X_n NIL) → (CONCAT X_1 (CONCAT X_2 ---(CONCAT X_{n-1} X_nNIL) --- NIL)NIL).

This rule, however, makes a multiple level structure out of a single level structure and thus destroys the form of the internal tree which is needed to parse the symbols for the display. To remove this difficulty an indicator is put on the property list of the CONCAT form to show that it is not to be considered a node in the internal tree when the symbols are sent to the display by pass four. A pseudo-form, DELIMIT, is introduced to correspond to the old PRD operator. The shape of DELIMIT is just the shape of its argument. The rule above then becomes:

(PRD X_1 X_2 --- X_n NIL) → (DELIMIT(CONCAT X_1 (CONCAT X_{n-1} X_n (UNDELIMIT)) --- (UNDELIMIT)) (UNDELIMIT)) NIL).

2. Concatenation with variable size fence symbols and separation symbols.

In the example in(X,Y), the size of the comma does not depend on the size of the arguments X and Y so it can be handled by the form CONCAT. On the other hand, the size of the parentheses depends on the dimensions of the enclosed expression and so it is necessary to introduce a parenthesis form. The rule for functions is:

(FUNCTION X_1 X_2 --- X_n NIL) → (DELIMIT(CONCAT X_1 (PAREN(CONCAT X_2 (CONCAT(ATOM , (UNDELIMIT)) --- (CONCAT(ATOM , (UNDELIMIT) X_n (UNDELIMIT)) (UNDELIMIT)) (UNDELIMIT)) (UNDELIMIT)) (UNDELIMIT))NIL).

The size of the parentheses is chosen during the dimension pass. They are considered to be symbols introduced by the PAREN form, rather than arguments of the form. The dimension pass also tells the PAREN operator the depth of the parentheses contained in its argument. The PAREN operator then chooses parentheses, brackets, or braces accordingly. For example, the compiler would produce $\left\{a \cdot \left[c \cdot (b+d)\right]^{2}\right\}^{3}$. In the cases of transendental functions, the parentheses are ommitted if the argument is a single variable, a product of two variables, or a division.

3. Binding symbol

Often a size change is associated with a binding symbol. One might want to make the size change recursive; however, it takes only a few size changes to make the range of sizes too large. The current system uses characters which differ in size by a factor of two and in this case more than one size change is unsatisfactory. Fortunately, expressions occurring in practice would rarely require more than one size change if the recoursive rule were to be used. The second pass carries the size with it as it goes out the branches of the internal tree; the rewrite rule for the mathematical operator of any node of the tree may specify that certain branches from that node are to be rewritten using the smaller size. In order to present the rewrite rules, the smaller size has been indicated by a 1 on the rewritten expressions property list. In actual fact, the size is saved on a pushdown list where it can be retrieved by the

third pass. An example of a rule involving a size change is the rule for integrals where the expressions for the limits are re-written using the smaller size:

(ITG X1 (ATOM A NIL) (ATOM B NIL) X2 NIL) →
(ITG (CONCAT (ATOM D (UNDELIMIT)) X1 (UNDELIMIT))
(ATOM A (1)) (ATOM B (1)) X2 NIL).

4. Supersub

This notation is used for exponentiation and for subscripting. The placement of subscripts is a property of the particular variable or function name. The subscripts can be placed at any of the four corners, so that there are 15 possible spatial arrangements. To avoid using 15 distinct but similar operators, null arguments have been introduced. They are treated as picture parts having zero dimensions. When the rewriting pass discovers a NAM operator it picks up a subscript-placement-list associated with the variable or function name in question. This is a list of the form (X1 X2...Xn), where each Xi is one of the symbols NE, NW, SE, SW. The ith argument of NAM is then placed at corner Xi. Two or more arguments at the same corner are separated by commas. Placement of arguments must start at the NE corner and proceed counter-clockwise. When the subscript-placement-list is exhausted the remaining arguments are placed at the SE corner. For example if H has the subscript-placement-list (NW SW SW) then:

(NAM X1 X2 X3 X4 (ATOM H NIL) NIL → (SUPERSUB NIL X1
CONCAT X2 (CONCAT (ATOM , (UNDELIMIT)) X3 (UNDELIMIT))
(1 UNDELIMIT)) X4 (ATOM H NIL)NIL).

If a subscripted variable name is raised to a power the exponent takes the NE position if there are no NE subscripts. The rewrite rule uses SUPERSUB and the operator PWRUP which preserves the original tree by removing the exponent from the SUPERSUB level during the dimension pass and using it for its second argument.

For example:

(PWR (NAM X1 X2 NIL) X3 NIL → (PWRUP (SUPERSUB X3 NIL NIL
X1 X2 NIL) NIL NIL).

Note that this combination of levels must be handled carefully if one desires to use a scheme where duplicate subexpressions are rewritten and dimensioned only once.

5. Concatenation with infix symbols.

This form has two possible spatial representations. It it is too wide for one line, then some of the arguments can be placed on additional lines. In order to choose the correct form the arguments must be rewritten and dimensioned and the available line width must be known. The arguments and symbols are concatenated horizontally until a line is filled. Successive lines are then concatenated vertically.

In the case A + - B the connection symbol + can be omitted. Similiarly one might want to write A * B, but A(B+C), omitting the * when the parentheses remove the ambiguity. On the other hand, when the expression is continued on the next line one would still write:
A , but A
* (B+C) -B. This problem is resolved by associating with each connecting symbol lists of those operators which cause it to be surpressed. There is a list of these operators which surpress it from the left, and a list of those operators which surpress it from the right. In addition there is a list of operators which replace the connecting symbol when a new line is started. Before an argument is rewritten and dimensioned its main operator is checked against the above lists so that the connecting symbol may be omitted if necessary.

6. Title.

The title operator takes as arguments an expression and its name. The name is rewritten, dimensioned, and positioned on the left edge of the display; the expression must then fit into the remaining space.

7. Hybrid.

Some operators are simply combinations of the others.

In conclusion, rewriting is controlled by the available space, the type of operator, and the size and type of its arguments. The form is dependent on the dimensions only if line width is applied as a constraint.

III. Dimensions.

The dimension pass starts at the ends of the branches of the internal tree and works toward the base. At the ends of the branches are the individual variable names. These are either the names of special characters such as O , or a sequence of characters to be displayed by the character generator. The names of special characters are marked with a flag and have their dimensions associated with them. The width of sequences of generated characters must be computed using their size and the fact that in the SAL language used for display * and = are mapped into character generator case shifts and / is used as a quote character, and also the fact that some generated characters are non-spacing.

Five dimension functions must be written for each form. There are functions for the width, height, depth, symbols, and arguments. The compiler uses these functions to put the dimensions of each picture part on its property list. The three functions for the width, height, and depth have a list of the dimensioned arguments of the form as input. They use helping functions to retrieve the dimensions from the property lists of the arguments. The position of an argument or symbol is specified by the coordinates of the lower left hand corner of the rectangle containing it. The function for the arguments has as value an S-expression of the form ($\Delta X_1 \Delta Y_1$ ---- $\Delta X_n \Delta Y_n$) where ΔX_i, ΔY_i is the position of the i^{th} argument with respect to the lower left hand corner of the circumscribed rectangle. The function for the symbols has as output an S-expression of the form ($\Delta X_1 \Delta Y_1$ $NAME_1$ S_1 W_1 ---- $\Delta X_n \Delta Y_n$ $NAME_n$ S_n W_n) where the i^{th} symbol which is added to the display by this form has name $NAME_i$, size S_i, width W_i, and relative position $\Delta X_i \Delta Y_i$. The functions for the symbols and arguments have as input the width, height, and depth, as well as a list of the dimensioned arguments. The compution of the dimensions and relative positions by independent functions breaks down when symbols such as parentheses must be chosen, as it is not desirable to repeat this choice for each of the five functions. To solve this problem

a mechanism is set up by which the width function, which is executed first, can communicate its choice to the others.

Each of the forms currently in the system is pictured in the following figures. Dotted lines have been drawn to indicate how the arguments and center lines are placed. It might be well to note that the human eye is sensitive to even the smallest misalignments.

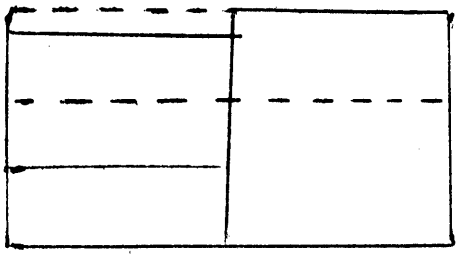


Figure 1: CONCAT

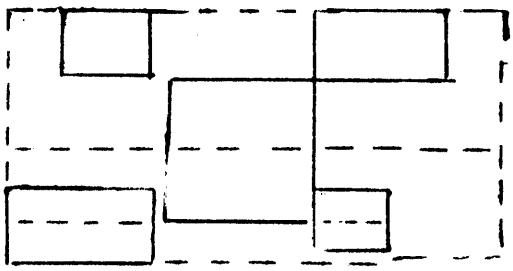


Figure 2: SUPSUB

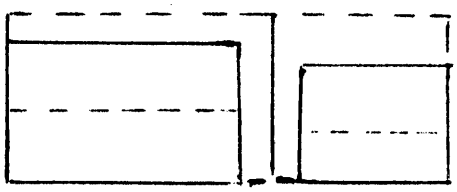


Figure 3: EVL

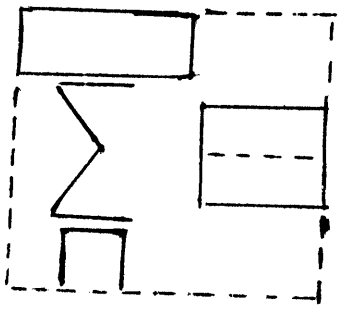


Figure 4: SUM

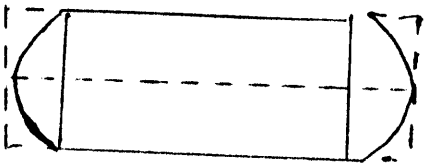


Figure 5: PAREN

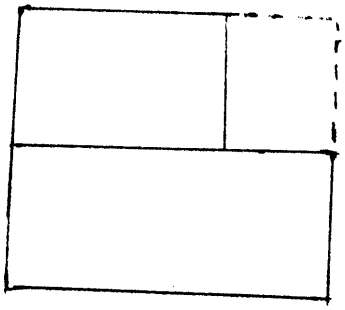


Figure 6: LVCONCAT

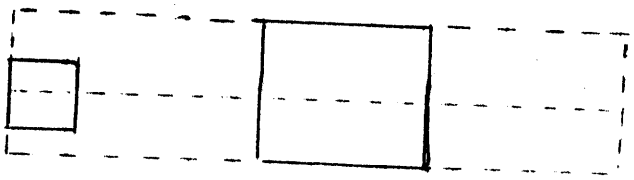


Figure 7: TITLE

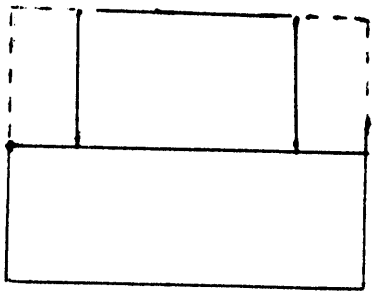


Figure 8: VCONCAT

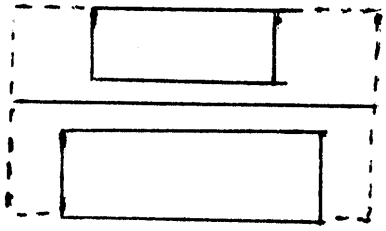


Figure 9: DVD

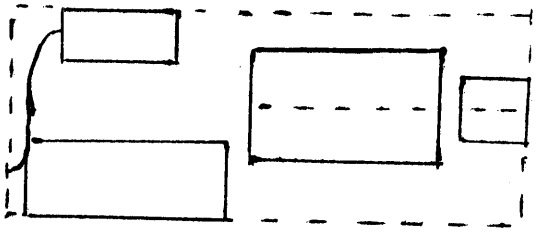


Figure 10: ITG

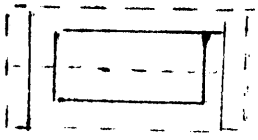


Figure 11: ABS

IV. Output to the Display

The LISP picture compiler program runs in Project MAC time sharing. After an expression has been dimensioned, the compiler outputs a description of the picture to be displayed over the dataphone to PDP-6 LISP. The picture is displayed on the PDP-6 scope using the SAL and MACROSAL languages. Each expression picture is set up as a single MACROSAL object with the picture tree marked by left and right pseudo-parentheses. The compiler starts at the base of the tree and goes out the branches, taking them in left to right order. The compiler sends each symbol to the display as it is encountered. If two successive symbols are not adjacent, then the compiler links them with a non-displaying relative vector. Whenever the compiler encounters a new node of the internal tree it follows the last symbol sent with a left pseudo-parenthesis. After the compiler has finished this node and all branches extending from it the corresponding right pseudo-parenthesis is sent. This information is also sent to the disk so that the picture can be reconstructed without being recompiled.

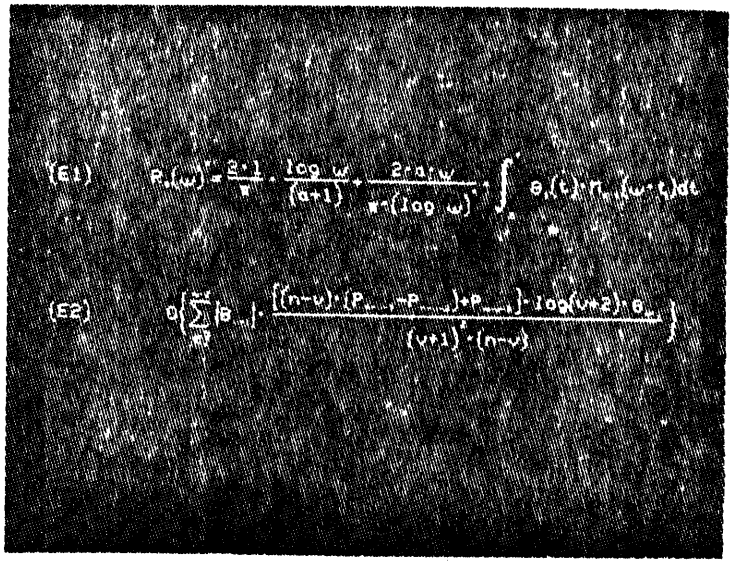
V. Example

The following figure is a photograph of two displayed expressions.
The source expression for E1 is:

```
(EQN(PWR(NAM O(R OMEGA NIL)NIL)*A NIL)
(PLS(PRD(PRD 2 *L(PWR PI -1 NIL)NIL)
(PRD(*LOG OMEGA NIL) (PWR(PLS *A 1 NIL)-1 NIL)NIL)NIL)
(PRD(PRD 2 *A OMEGA (PWR PI -1 NIL) (PWR(*LOG OMEGA NIL) (PRD -1 *A NIL)NIL)NIL)
(ITG *T O PI (PRD(NAM 1(THETA *T NIL)NIL)
(NAM(PLS *A -1 NIL) (M(PRD OMEGA *T NIL)NIL)NIL)NIL)NIL)NIL)NIL)NIL)
```

The source expression for E2 is:

```
(O(SUM *V(PRD *N(PWR 2 -1 NIL)NIL)
(PLS *N -2 NIL) (PRD(ABS(NAM(PLS *V 1 NIL)B NIL)NIL)
(PRD(PLS(PRD(PLS *N(PRD -1 *V NIL)NIL)
(PLS(NAM(PLS *N(PRD -1 *V NIL)-1 NIL) P NIL)
(PRD -1(NAM(PLS *N(PRD -1 *V NIL)-2 NIL)P NIL)NIL)NIL)
NIL)(NAM(PLS( *N(PRD -1 *V NIL)-2 NIL)P NIL)NIL)
(*LOG(PLS( *V 2 NIL)NIL)(NAM(PLS *V 1 NIL)
THETA NIL)(PWR(PLS *V 1 NIL)-2 NIL)
(PWR(PLS *N (PRD -1 *V NIL)NIL)-1 NIL)NIL)NIL)NIL)NIL)
```


(E1)
(E2)

VI. Conclusion

One measure of program complexity is the extent and predictability of the combinations of independent parts of the input data which must be made in order to determine the flow of program control. In this respect the display of mathematical expressions seems to lie midway in difficulty between the display of lines of english text and the display of arbitrary graphs. In the display of english text, the data is organized in a string, the only global property required is a character count. Hyphenation decisions are based on the word at the end of the line. On the other hand, the display of graphs seems to require the simultaneous positioning of several nodes, and might require a character algebra or an iterative approach. In the display of mathematical expressions, decisions can be made at each node of the expression tree, based on information collected from above and below. The central importance of this tree structure makes LISP a convenient language for the program.

Bibliography

The idea of rewriting a mathematical expression in terms of position operators and computing the dimensions recursively is contained in the memo by Minsky. He also suggests that the picture symbols should be parsed for Light pen reference. In his thesis Krakauer contributed several ideas about picture syntax.

References

M. L. Minsky, Mathscope; Part I. Artificial Intelligence Memo 61

L. J. Krakauer, Syntax and Display of Printed Format Mathematical Formulas, M.I.T., M. S. Thesis, 1964

The William Byrd Press, Mathematics in Type, The William Byrd Press, Richmond, Virginia, 1954

Mark B. Wells, MADCAP: A Scientific Compiler for a Displayed Formula Textbook Language: Communications of the ACM, Vol. 4, No. 1, January 1961

M. Klarer and J. May, An Experiment in a User Oriented Computer System, Communications of the ACM, Vol. 7, No. 5, May, 1964

William A. Martin, PDP-6 LISP Input-Output for the Display, Artificial Intelligence Memo 80

William A. Martin, PDP-6 LISP Input-Output for the Dataphone, Artificial Intelligence Memo 79