Artificial Intelligence Project
Memo 61--

MASSACHUSETTS INSTITUTE OF TECHNOLOGY
Project MAC

A Proposal for a Mathematical Manipulation-Display System

by M. L. Minsky

Mathscope: A Compiler for Two-Dimensional Mathematical Picture-Syntax

Mathscope is a proposed program for displaying publication-quality mathematical expressions given symbolic (list-structure) representations of the expressions. The goal is to produce "portraits" of expressions that are sufficiently close to conventional typographic conventions that mathematicians will be able to work with them without much effort—so that they do not have to learn much in the way of a new language, no far as the representation of mathematical formulae is concerned. It remains to be seen whether this is a useful goal; it may turn out that for computer assistance with mathematics, we will want new representations. In any case, the system should be useful in several ways, e.g., for automatic typesetting of mathematical results reached by computer processing, and for better understanding of the syntax of this kind of picture language.

Work on this system will probably be done by several people, including some thesis work. This report sets down the current state of my thinking about it, to help to coordinate work on various parts of the system. It will be important to work the thing out more cleanly because the end result will be a rather large system that will be unmanageable unless neatly partitioned.
Input to the system will be LISP expressions made up of operators and variable symbols. The representation is to be consistent with those used by W. Martin in his mathematical transformation programs. The output is to be sets of pictures on a scope-keyboard light-pen console. A return path, using light-pen and typewriter, will control manipulation of the displayed pictures.

My current picture of the system has several parts.

OBJECTS

LISP-math expressions

expressions with explicit grouping structure

expression structure with dimension and relative coordinate structure

punctuated-string picture representation

pictures (expression-portraits)

PROGRAMS

heuristic picture-syntax compiler

coordinate-dimension compiler

list-structure - to - string translator

and

7090 - PDP-I transmission

PDP-I picture compiler and light-pen sub-expression detector

PDP-I mathematical text-editor and page coordinator

translator for back-to-LISP action requests and 7090 filing systems

PDP-I to 7090 action signals and references to files
In this system, the PDP-1 is to be used as a display terminal station. It remains to be decided to what extent the PDP-1 will be used to handle the page-editing and filing systems; if these systems are used primarily with LISP we will have to decide between LISP and lispzero, treating the 7090 as a device to perform implicit transformation encoding for various picture-strings. If, however, it is practical to let the user decide if good picture-maintenance from the 7090 becomes practical, it will be better to do everything within LISP. This decision will have to be made soon.

Sketch of the Part of the System

We imagine that the user is engaged in performing some sort of exploration. For example, he might be trying to find a solution to a differential equation. At the moment he has displayed on the screen one or two equations, and he has in his head the names of several other expressions or partial results already studied and filled away. He decides to perform some action, e.g., substituting for a displayed equation, solving it for some variable, expanding some subexpression in a expansion, or perhaps simply displaying something else. This action is decomposed into some combination of input-pen and keypad signals. These signals are encoded and transmitted to LISP, which computes or retrieves the desired new expressions and transmits them back to the display system. The latter then compiles and displays the desired new picture.

The basic ingredient of the system is the program-service that converts an internal mathematical expression into a conventional picture representation. This has interesting linguistic or physical implications, and is worth studying in its own right, since it is a key part of
something about mathematical expression and this area. To my knowledge, this kind of algebra-like expressions has never been studied, except by the group of K & K to which I belong.

\[ \text{(Input Internal Expressions)} \]

Consider first simple functions in expressions of the form

\[ a \cdot b \cdot c \cdot d \cdots (\text{etc.}) \]

This might originate as a result of some operation, or it may be... of the form

\[ (\text{PLUS A MULTIPLE OF B)} \] or \[ (\text{PLUS A MULTIPLE OF C)} \]

where a function of several arguments is represented by one variable, with the function used and followed by the arguments. In both cases, this structure and the picture is very in that simple to very complicated, and becomes even more complex when we go to more complicated operations and more complicated variables, e.g., with subscripts, superscripts, and so on. For example, that

1. Some parentheses are retained, others suppressed.

2. The multiplication operator does not always appear, etc.

3. All the operators here and up displayed as "infix" expressions, they are pictured as connecting physically the operands; while others, e.g., \[ (\text{ADD}, y) \] most appear in function form. In the case of \[ (\text{PLUS} x - 1) \], the result,

\[ x - y - z \]

yields the corresponding signs to do operations.

As we go further into the problem, we will encounter more serious problems.
with the various variables and parameters that are a part of the functional framework and the question becomes:

1. Can we formulate a reasonable approach to solving a large proportion of a certain population?

2. Can we make some progress in the process of handling both the non-ideal and the ideal cases, especially in special cases.

Preliminary Program Writing

The most natural thing to do to begin with was to write a program that generated a context-free grammar (I mean, a procedure that is defined by a set of production rules) and uses a QUOTIENTS technique. In making such a program, we would generate something like:

\[ \text{PROGRAM A (QUOTIENTS)} \]

where, now, everything in the picture is placed in a box. Then we would ask the question: how many parameters to be deployed are necessary? The question of new operators, how are they possible? How do we avoid the overloading and the effect of the multiplication in the different variables? Obviously, the answers will be similar. The problem with ordinary precedence conventions as described earlier, for example, this program might have the same order of operations as:

\[ \text{A B C D} \]

Bracketing: Structures and Procedures

At the next phase, we are now looking at the actual execution. The results of the basic processes are:

\[ \text{PRELIMINARY REPORT} \]
The dimensions of one such section have an interesting property. Consider a small sample of an object modified into a section that is an exploring or exploring on either of more expressive means. The picture is made to exhibit the best of all approaches, to expand a section by the additional property shown below. As will be noted later, a large section with the apparent changes of structure will impart insight into this property.

Each exploration, such as the rectangular sheet of each such section, is more for height, depth, and all three together.

\[ z \text{ and } f \text{ are proportional to } \] 

\[ y_g \text{ at the limit of } \] 

\[ y_g \text{ as the limit of } \]
All mathematical expressions are regarded as extending above and below a well-defined centerline. This is usually the level of a fraction-bar when this is the main connective.

\[ h_E \]  is the height of \( E \) above the centerline.
\[ d_E \]  is the depth of \( E \) below the centerline.
\[ v_E \]  is the width of \( E \).

Associated with each variable symbol \( E \) are its own \( h_E \), \( d_E \), and \( v_E \). Most operator symbols also have their own dimensions, but some (like the fraction-bar and parentheses) have dimension functions instead, as will be seen.

Consider a sum-expression

\[ e = (\text{PLUS} \; e \; f) \]

where \( e \) and \( f \) are subexpressions. The picture for this should have the form

\[ \begin{array}{c}
  e \\
  + \\
  f \\
\end{array} \]

wherein the centerlines of the subexpressions are lined-up with that of the "+" and otherwise the spacing is close. (It is understood that all expression rectangles are already set-up with appropriate clearances around the edges. It may be necessary to do this in some cases, especially in the final system.) The resulting picture for \( e \) will evidently satisfy the following equations:
Dimension Equations

\[ w_s = w_e + w_f \]
\[ h_s = \max(h_e, h_f, h_t) \]
\[ d_s = \max(d_e, d_f) \]

Coordinate Equations

\[ x_e = x_s \]
\[ y_e = y_s \]
\[ x_f = x_e + w_e \]
\[ y_f = y_e \]
\[ x_t = x_e + w_e + w_f \]

Observe that computation of the dimensions of \( d \) requires only the dimensions of the subexpressions, while the computation of the coordinates of the subexpressions of \( e \) require the coordinates of \( e \) and the dimensions of the subexpressions. This means that the dimensions must be computed first, from the ends of the tree, on a first pass. Only then can the coordinates be computed (absolutely or relatively) by backtracking through the tree on a second pass.

The dimension pass, thus, leads to an intermediate list structure which looks like this: \( s = (\text{PLUS} \ e \ f) \) becomes

\[ (((w_e + h_e d_e) (h_e h_a d_a) (w_f h_a d_a)) \ldots) \]

where \((I_e)\) and \((E_f)\) are the similar structures compiled for \( e \) and \( f \).

The new list structure has twice the depth of the old, with the interpolated levels carrying the dimensions of the subexpressions depending from them. Cumbersome, but anything more compact looks dangerous.

The dimension pass works from the ends of the structure:

\[
\text{dim}[s] = \text{prog}[[a;\text{dim}a;b];
\]

\[
a := \text{car}[s];
\]

\[
\text{dim} := \text{list}[a;w[a];h[a];d[a]];\]

\[
[\text{null}[a] \rightarrow \text{return}[\text{dim}a]];\]

\[
b := \text{cons}[\text{dim}a;\text{maplist}\text{cdm}[s];\ldots];\]

\[
\text{return}[\text{cons}[b;\text{dim}a;b]]]\]
where wfa[l], hfa[l], and d[x] are functions that get the dimensions of the symbol denoted by a;

wfa[l] is a predicate that tells whether a is an operator or just a variable; and

dimfa[a;b] is the function that does all the work—that is, it computed the dimension-triplet of the new expression, given an operator a and the structure obtained by light-ing the results of applying dimfa to each of the sub-expressions governed by the occurrence of the operator a.

At certain points of the process, because of special operators like those for subscripting, exponentiation, and range notations, a global scaling multiplier is applied to subexpressions. This, too, must be carried along in the first-dimension pass. When size is considered (for subscripts, exponents, etc.) the triple (w, h, d) will have to be made a 4-tuple (w, h, i, s), and g will multiply all dimensions of lower order. (It must be kept because it is needed in going back down in the coordinate pass.)

The function dimfa[a;b] embodies the dimension equations above, for each operator a. It can be quite complicated in the case of operators like PLUS and TIMES which have arbitrary numbers of arguments.

Question: should the new picture elements be introduced here or later in the coordinate-pass. One reason to do it in the dimension-pass is to keep the coordinate-pass simple enough to do in the 1090. This would have some value in handling presentation of subexpressions without bothering the 7090. In this case, the last program line has to be (grossly) modified, since the new elements have to be inserted in the strings (by introducing an additional CONCAT level). Other complications, like signs and integral limits, cannot be handled simply by CONCAT. For these, we
need to add 2-dimensional displacement information to the microcode structure in order that all the work of using digit be not repeated on the second pass.

The problem of operators like PLUS and TIMES is so complicated that it will probably be necessary to reduce them (in the pre-dimension pass) to binary operators. This will greatly simplify the coordinate pass, and make simpler the definition of the required functions for each operator. It does make for some difficulty in assignment of subexpressions to light-pen responses. However, this is an ambiguity already here, and it has to be resolved somehow. There seems to be a special problem concerning how the operator is to re-group sums at the console. Note that the program that reduces PLUS and TIMES to binary need not introduce any unnecessary PAREN display-elements.

**Expression-Structures with Coordinates**

The result of the second (coordinate) pass will replace the $(v,h,d)$ triples with $(x,y,\text{size})$ triples, with special information associated with peculiar operators. At this point one can compile special information, for example, for novel delimiters: one could easily ask to display one of the actual rectangles instead of simple parentheses; or one could specify overbars, horizontal braces, or even lines from one symbol to another.

Here is where our imaginations can be deployed in attempts to improve over what has been typographically practical. One can, for example, translate from a LISP conditional-range statement to the conventional mathematical conditional brace-notation. There are interesting research problems in formalizing the syntax and semantics of the "..." notation for e.g., formal power series. To what extent can one apply difference...
methods to differential problems by transforming the picture system; can one translate by this automatically from operator to Helmholtz notation? Is there any mathematical value in studying this system? It is unlikely that one will notice anything new in classical problems, but one might possibly discover interesting new formal computational methods.

Punctuated-String for Data-Transmission and Filing

At this point, the structure probably ought to be converted, using a modified copy-like function, into a linear list with left and right parenthesis elements that preserve the previous list-structure information. This master list is then dumped into some linear storage array and transmitted by channel to the PDP-1, if that is where the control programs are centered. A compiler in the PDP-1 can assemble from this symbol-coordinate structure-string a display program that can be called to the scope. The PDP-1 picture assembler includes light-pen traps for each picture-element; when a trap occurs, the PDP-1 should then compute the LISP-address of that subexpression which contains the sensed picture-element on its top level. (This information can be reconstructed by a scan through the linear coordinate string with parentheses, and referred back, in the 7090, to a stowed copy of the expression that generated it.)

This light-pen "responsibility" should help in achieving the "magic paper" effect. Touching a parenthesis will designate the full expression it delimits, and one can move that subexpression as a unit. Touching a function-name will seize it and its arguments. In the largest context, touching an equation-number will get that equation—if equation-numbers are treated as higher-than-top-level connectives, belonging to a text-manipulation meta-language.
Problems about Particular Operators

Division

In (MIV,e,f) one may use the symbol "f" if both g and h are single symbols. But the decision really depends on aesthetic aspects: the division bar "- - - -" eliminates parentheses of both g and h in many cases. Usually, this visual simplification will be worth the increase in vertical size. One might compute the perimeter of both versions and use this to make the decision. In any case, "---" is usually at or slightly above the centerline, the two subexpressions are centered on it, and its length is the maximum of w_e and w_f.

Parentheses

It is tempting to make the height of parentheses equal to h_e + d_e and centered. The width should not scale directly, but probably increases rather slowly (from a standard unit width) with height. It is easy to specify roofs or vincula, or boxes, etc., instead.

Subscripts

The entire subscripted expression a_f = sub(c,f) appears to be centered at

\[ y_f = y_e - \frac{d_e}{2} \]
\[ y_e = y_s \]
\[ x_f = x_e + w_e \]
\[ x_e = x_s \]

and the entire size of f is scaled down uniformly by a factor of about 2/3, hence

\[ w_s = w_e + \frac{2}{3} w_f \]
\[ c_s = c_e + \frac{2}{3} c_f \]
\[ h_s = \max(h_e, \frac{2}{3} h_f + d_e), \] but woe if the second argument...
Combined superscripts (or exponents) and subscripts probably have to be managed by a ternary operator \texttt{supsub}(e,g,3) as we have to show \( x_i^j \) instead of just \( x^j_i \). The same is necessary to handle indices, integrals, and other operators with upper and lower ranges. The ternary operators still seem consistent with the basic rectangular framework. There is always the danger of physical collisions of different subexpressions. It would be a great nuisance to have to check for these; fortunately, the syntax rules can almost always prevent this. The decision to use

\[
\int_v^u \frac{a + b}{c} \, dx \quad \text{or} \quad \int_v^u (x + y) \, dx
\]

can certainly be made easily by a patch in the dimension pass when the dimensions of the integrand become available. If the integrand is large enough, we replace the compact operator by the more vertical form. The same could be done in \texttt{supsub}; the collision of the superscript and subscript rectangles can be easily prevented (in most cases) by switching to a more vertical operator whenever necessary. These conventions can be made part of the \texttt{dim} function definitions, but probably should work by changing the operator name and then expecting its \texttt{dim} computation.

M. Minsky