MASSACHUSETTS INSTITUTE OF TECHNOLOGY

Project MAC

Artificial Intelligence Project
Memo 61--

Memorandum MAC-M-113
November 13, 1963

MATHSCOPE; PART I

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, so 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 consistant 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	PROGRAMS
LISP-math expressions	
	heuristic picture-syntax compiler
expressions with explicit grouping structure	
	coordinate-dimension compiler
expression structure with dimension and relative coordinate structure	
	list-structure - to - string translator and 7090 - PDP-1 transmission
punctuated-string picture representation	
	PDP-1 picture compiler and light-pen sub-expression detector
pictures (expression-portraits)	
	PDP-1 mathematical text-editor and page coordinator
user actions	
	translator for back-to-LISP action requests and 7090 filing systems
PDP-1 to 7090 action signals and references to files	

In this system, the PDP-1 is to be used as a display and [illegible] station. It remains to be decided to what extent the PDP-1 [illegible] handle the page-editing and filing systems; if the system is to be used primarily with CTSS we will lean toward more PDP-1 autonomy, [illegible] the 7090 as a device to perform indicated transformations [illegible] picture-strings. If, however, it is practical to get the [illegible] 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 Parts of the System

We imagine that the user is engaged in performing a [illegible] 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 filed away. He decides to perform some action, e.g., substituting in a displayed equation, solving it for some variable, expanding some subexpression in a certain way, or perhaps simply displaying something else. This action is requested by some combination of light-pen and keyboard signals. These [illegible] are encoded and transmitted to LISP, which computes or retrieves the required 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-sequence that converts an internal mathematical expression into a conventional [illegible] representation. This has interesting linguistic-or pictorial-language aspects, and is worth studying in its own right, since [illegible]

something about mathematical problem-solving [illegible] this area. To my knowledge, this kind of picture-language [illegible] studied, except by the group of R. A. [illegible]

Input (Internal) Expressions

Consider first simple functional compositions of [illegible] tors, like

$$a + \frac{b}{(c+d)(e+f)} .$$

This might originate (as a result of [illegible] operations), [illegible] of the form

(PLUS A (QUOTIENT B (TIMES (PLUS C [illegible]) (PLUS E F) [illegible]

where a function of several arguments is represented by [illegible] with the function name and followed by the arguments. [illegible] this structure and the picture is [illegible] in this simple [illegible] cated, and becomes more complex when we go to more graphic operators and more complicated variables, e.g., with subscripts, [illegible] for example, that

1. Some parentheses are retained, others suppressed

2. The multiplication operator does not appear [illegible] e.g., exponentiation, will appear only implicitly [illegible] notation

3. All the operators here end up displayed as "infix" [illegible] that is, they are pictured as connecting physically [illegible] while others, e.g., max(x,y) must appear in functional form. In the case of (PLUS X Y Z), the result,

   X + Y + Z

   yields two disconnected signs, as do parentheses.

As we go further into the problem, we will encounter more [illegible] problems.

e.g., the representation [illegible]

$$\sin^2 x$$

with the curious [illegible]

of the functional brackets [illegible]

1. Can we formulate a reasonably [illegible] a large proportion of a [illegible]

2. Can we make [illegible] handle this [illegible] special cases?

Picture Expressions with [illegible]

The most natural thing here seems to be [illegible] given a matherpr, generates a [illegible] procedure that is driven by a set of [illegible]

(PLUS A (QUOTIENT B [illegible]

we would generate something [illegible]

(PLUS A (QUOTIENT B [illegible]

where, now, everything in the picture [illegible] sion: the parentheses to be displayed [illegible] of new operators PARE [illegible] and the effect of the multiplication [illegible] tion. Obviously, the rules [illegible] ordinary precedence conventions, plus [illegible] example, this program might [illegible] kinds of brackets, and when to use $\frac{a}{b}$ [illegible]

Expression Structure with Dimensions

The next phase concerns skeleting [illegible]

---

* A preliminary version [illegible] plated by [illegible]

A.1. [illegible]

done in two steps:

(i) A dimension [illegible]

(ii) A coordinate [illegible]

The dimensions [illegible]

have an interesting [illegible]

consider a [illegible]

modified when we [illegible]

An expression is either [illegible]

or more expressions [illegible]

picture is made by [illegible]

of sub-expressions [illegible]

by the associated [illegible]

As will be noted later, [illegible]

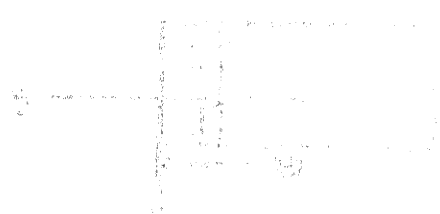
with the operation [illegible]

with respect to light [illegible]

Each expression [illegible]

each such rectangle [illegible]

height, depth and width of [illegible]



x and y are [illegible]

x_B is the [illegible]

y_B is the [illegible]

All mathematical expressions are regarded as extending above and below a well-defined centerline. This is normally 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.

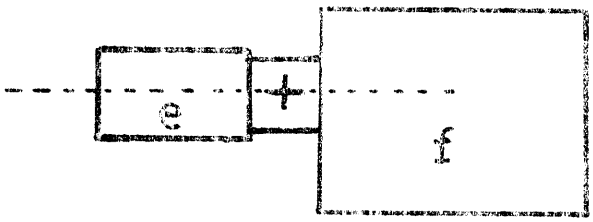
w_E is the width of E.

Associated with each variable symbol S are its own h_S, d_S, and w_S. 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

s = (PLUS e f)

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



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 more subtle way in the final system.) The resulting picture for s will evidently satisfy the following equations:

Dimension Equations

$$w_s = w_e + w_+ + w_f$$

$$h_s = \max(h_e, h_+, h_f)$$

$$d_s = \max(d_e, d_+, d_f)$$

Coordinate Equations

$$x_e = x_s$$

$$y_e = y_s$$

$$x_+ = x_s + w_e$$

$$y_+ = y_s$$

$$x_f = x_s + w_e + w_+$$

Observe that computation of the dimensions of s requires only the dimensions of the subexpressions, while the computation of the coordinates of the subexpressions of s require the coordinates of s 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 branching back through the tree on a second pass.

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

$$(((+\ w_+\ h_+\ d_+)\ ((E_e)\ w_e\ h_e\ d_e)\ ((E_f)\ w_f\ h_f\ d_f))\ x_s\ y_s)$$

where (E_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:

```
dim[s] = prog[[a;dima;b];

    a := car[s];
 dima := list[a;w[a];h[a];d[a]];
 [vbl[a] ⟶ return[dima]];
    b := cons[dima;maplist[cdr[s];λ[[x];dim[car[x]]]]];
 return[cons[b;dimf[a;b]]]]
```

where v[a], h[a], and d[a] are functions that get the dimensions of the symbol denoted by a;

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

dimf[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 list-ing the results of applying lin 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 local scaling multiplier is applied to subexpressions. This, too, has to 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,d,s), and s will multiply all dimensions of lower levels. (It must be kept because it is needed in going back down in the coordinate pass.)

The function dimf[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 PDP-1. 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 string (by introducing an additional CONCAT level!). Other complications, like signs and integral limits, can't be handled simply by CONCAT. For these, we

need to add 2-dimensional displacement information to the first-pass structure in order that all the work of using dimf 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,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 over-bars, 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 Leibnitz' 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 computation 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 stored 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 larger 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 (DIV,e,f) one may use the symbol "/" if both e and f are single symbols. But the decision really depends on esthetic aspects. The division bar "————" eliminates parentheses of both e and f 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 e_f = sub(e,f) appears to be centered at

$$y_f = y_e - d_e \qquad y_e = y_s$$

$$x_f = x_e + w_e \qquad x_e = x_s$$

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

$$w_s = w_e + 2/3 \cdot w_f$$

$$d_s = d_e + 2/3 \cdot d_f$$

$h_s = \max(h_e, 2/3 \cdot h_f + d_e)$, but woe if the second argument

dominates!

Similarly for superscripts.

Combined superscripts (or exponents) and subscripts probably have to be managed by a ternary operator supsub(x,s,g). If we are to allow x_i^j instead of just $x_i{}^j$. The same is necessary to handle sigmas, 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 becomes available. If the integrand is large enough, we replace the compact operator by the more vertical format. The same could be done in 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 connections can be made part of the dimf function definitions, but probably should work by changing the operator name and then repeating its dimf computation.

M. Minsky