41P

Artificial Intelligence Projects--RLE and MIT Computation Center
Memo 51--

# METEOR: A LISP Interpreter for String Transformations

by
Daniel G. Bobrow

April 24, 1963

# METEOR: A LISP Interpreter for String Transformations

## 1. Introduction

Conditional expressions, composition and recursion are the basic operations used in LISP to define functions on list structures. Any computable function of arbitarily complex list structures may be described using these operations, but certain simple transformations of linear lists (strings) are awkward to define in this notation. Such transformations may be characterized (and caricaturized) by the following instructions for a transformation: "Take that substring there, and that other one starting with "Black", which has the substring mentioned third as the first; then inserts the second substring mentioned; omit the first and leave the unmentioned parts of the original string unchanged."

A notation for expressing such transformations is the basin for the COMIT programming language of Yngve.[1] There is a formal method for selecting substrings from a string and then indicating the structure of the transformed string. It is easy to write COMIT rules which perform string transformations such as rearrangement, deletion, insertion, and selection of elements from context. However, COMIT does not easily allow the general list processing that can be done in LISP.

A language in which both types of processing could be easily expressed would be desirable. As a compromise, to allow easy string manipulation within LISP, a LISP function, METEOR, has been written which will interpret COMIT-type rules, and perform the indicated string transformations. METEOR notation is similar to that used in COMIT, with some additional features, such as use of mnemonic names for substrings and shelves. This memo describes

---

1. V. Ynguve, "Introduction to COMIT Programming" and "COMIT Programming Reference Manual," MIT Press; June 1962.

FIGURE 1

```
 0.  METEOR((
 1.  (*    (ROSE) (FLOWER) * (SIMPLE REPLACEMENT))
 2.  (*     ((*P THE WORKSPACE IS))          *    (DEBUG PRINTOUT))
 3.  (*   (IS A ROSE)     (       * (DELETION))
 4.  (*    (A FLOWER IS)   (3 1 2)           *   (REARRANGEMENT))
 5.  (*   ((*P WS2))                *)
 6.  (*   ( FLOWER)   ( 1 OF RED)        *     (INSERTION) )
 7.  (*    (A FLOWER)   (THE 2)    * (REPLACEMENT IN CONTEXT))
 8.  (*   ((*P WS3))                *)
 9.  (*    (FLOWER)                              *   (NO OPERATION))
10.  (*      (RED)     (1 1)         * (DUPLICATION))
11.  (*   ((*P WS4))                *)
12.  (*    (OF ($.1))    (1)       *(SINGLE UNKNOWN CONSTITUENT))
13.  (*     (($.1))     (QUESTION1)                  *   (FIRST CONSTITUENT
14.  (*     ( (*P WS5))           *)
15.  (*   ( ($.2)    FLOWER ($.3))    (3 2 1) * (N CONSECUTIVE CONSTITUENTS))
16.  (*   ((*P WS6))                *)
17.  (*   (FLOWER $ ROSE)   (1 3)     *    (UNKNOWN NUM OF CONSTITUENTS))
18.  (*     ( (*P WS7))          *)
19.  (*     ($)    (START C A B D)   * (REPLACING ENTIRE WORKSPACE))
20.  (*   (START ($.1)   $ D)    (1 3 2 4)   *)
21.  (*   ((*P WS8))
22.  (*    ($)          END)
23.  )(A ROSE IS A ROSE IS A ROSE))
```

FIGURE 2

```
(THE WORKSPACE IS)
(A FLOWER IS A ROSE IS A ROSE)
(WS2)
(IS A FLOWER A ROSE)
(WS3)
(IS THE FLOWER OF RED A ROSE)
(WS4)
(IS THE FLOWER OF RED RED A ROSE)
(WS5)
(QUESTION IS THE FLOWER OF RED A ROSE)
(WS6)
(QUESTION OF RED A FLOWER IS THE ROSE)
(WS7)
(QUESTION OF RED A FLOWER ROSE)
(WS8)
(START A B C D)
```

in detail the types of program statements that can be interpreted by METEOR, and is hopefully independent of any knowledge of COMIT. Similarities to COMIT will be obvious to the knowledgeable reader, and occasional warnings about differences between METEOR and COMIT are inserted.

Section II of this memo is an introduction to METEOR by examples. Most of the features of the system are illustrated. The third section is a complete, exact specification of a METEOR program and its interpretation. Section IV is a collection of warnings for the unwary about the foibles of the system--a combination of the worst of LISP and COMIT--and paternal advice about how one can best use the system. Section V is a current listing of the program for METEOR, as of the date of this memo. All known bugs have been removed, and this program has successfully interpreted all examples given in the text. Reference to this listing and a LISP manual should resolve any unintentional ambiguities in Section III.

## II. Operations with Meteor Rules

In this section the structure and operation of some types of Meteor rules will be illustrated by simple examples. Figure 1 is a listing of a sample program as run in the LISP system under the Meteor Interpreter. The output from this program is shown in Figure 2. The entire program will be discussed in some detail. The LISP interpreter must be informed that it is to use the Meteor program. The card in line 0 performs this function. The two left parentheses open, respectively, the list of arguments for Meteor (it has two) and the list of rules (the first argument). Lines 1-22 are rules which will be applied successively to transform the list (called the workspace) given in line 23, i.e., "(A ROSE IS A ROSE IS A ROSE)".

---

2. LISP 1.5 Programmer's Manual, RLE Publications Office; 1963.

Replacement

Items in the workspace may be replaced. The rule in line 1 finds the first occurrence in the workspace of "ROSE" and replaces it by an occurrence of "FLOWER". The list "(ROSE)" is called the left half of this rule, and the list of one element "(FLOWER)" is the right half of this rule. The "left-half" selects elements from the workspace.

Printout of the Workspace

The contents of the workspace may be printed out with an identifying message by a rule such as that in line 2. The list "(*P message)" as the only element of the left half of a rule will cause a printout of the "message", first, and then the contents of the workspace. Line 2 prints out "(THE WORKSPACE IS)" and then the contents of workspace. This printout is shown in Figure 2. The workspace remains unchanged.

Deletion

Items may be deleted from the workspace. The rule in line 3 finds the first occurrence in the workspace of the three words "IS A ROSE". These are specified by the left half of this rule. The atom 0 (zero) as the right half of this rule specifies that these words should be deleted from the workspace, and nothing inserted in their place.

Rearrangement

Items in the workspace may be rearranged. The rule in line 4 finds the first occurrence of "A FLOWER IS" and reorders it as "IS A FLOWER". This is specified by the order of the elements in the right half, the list "(3 1 2)". In this right half, 3 refers to the element matched by the third element of the left half, 1 to the first, and 2 to the second. Deletion and rearrangement can be done simultaneously by not mentioning in the right half an item

matched in the left half, e.g., if the right half of this last rule were (3 2) the string "IS FLOWER" would be substituted for "A FLOWER IS" and this occurrence of "A" would no longer appear in the workspace.

The rule in line 5 causes a printout of the modified workspace. (See Figure 2).

Insertion

New material may be inserted into the workspace by a METEOR rule. The rule in line 6 finds the first occurrence of "FLOWER" in the workspace, and inserts, just after it, the elements "OF RED". The 1, of course, refers to the occurrence of "FLOWER" as the first (in this case, the only) left half constituent (pattern element). Insertions can be made before, after or between elements of the workspace identified (matched) by the left half of the rule.

Replacement in Context

Suppose we wish to replace the article "A" by "THE" when it appears immediately before the word "FLOWER". The left half "(A)" cannot be used to locate this occurrence of "A". This left half will locate the first occurrence of "A" in the workspace. However, the "A" found by ".(A flower)" is the appropriate one i.e., the one immediately preceding "FLOWER", and the stated transformation is performed by the rule in line 7. Line 8 prints out this transformed workspace.

If only a left half appears in a rule, as in line (9), an occurrence of "FLOWER" is found in the workspace, but no transformation is made, and the workspace remains unchanged.

Duplication

Items found by the left half may be duplicated in the right half by mentioning them more than once. For example, rule 10 inserts another copy of "RED" into the workspace immediately succeeding the first occurrence.

Unknown Constituents

Sometimes only the context of an item desired is known. To locate such an item we need a notation for an unknown. METEOR uses the symbol "($.1)" (that is, left paren, dollar, period, 1, right paren), which is similar to the COMIT notation. This symbol represents any single unknown constituent. In rule 12, the "($.1)" is used to find the item immediately after "OF" in the workspace. Since 2 (which would refer to this item found by the left half) does not appear in the right half, this item after "OF" (i.e., the element "RED") is deleted by this rule.

Left half searches are made from left to right in the workspace; therefore "($.1)" can be used to find the first constituent in the workspace. Rule 13 finds this first constituent and inserts "QUESTION" immediately before it. Line 14 prints out the contents of the workspace after this transformation.

The notation ($.n) is used to represent for several consecutive unknown constituents, where n may be any integer. Thus in rule 15, "($.2)" refers to the 2 constituents immediately before "FLOWER" and ($.3) to the 3 constituents just after. This rule rearranges these constituents and the modified workspace is printed out by 16.

For an unknown number of constituents the symbol "$" is used. In 17 the "$" will match all items of the workspace between "FLOWER" and "ROSE". The right half specifies that these items are to be deleted. The result is shown by the output produced by 18. Since "$" will match any number of constituents, including 0 (zero), it can be used alone in the left half to match the entire workspace, and for example, as in 19, used to replace the entire workspace by a new string.

Line 20 contains a slighty more complex rule which will move the element after "START" to the position before "D". The result is printed out by 21.

Line 22 is a standard rule to terminate a METEOR program. A

FIGURE 3

```
0.  METEOR((
1.  (CHANGE ( A ROSE) (THE FLOWER)     CHANGE (FLOW OF CONTROL))
2.  (RULE1    (FLOWER    )                 RULE3)
3.  (RULE2 * ((*P WSP))    END)
4.  (RULE3     (ROSE)                     CHANGE)
5.  (* * (ROSE)    (FLOWER)           RULE2)
6.  (* *   ( (*P WSEND))    END)
7.  )(A ROSE IS A ROSE IS A ROSE))
```

a. Program

```
(WSP)
(THE FLOWER IS THE FLOWER IS THE FLOWER)
```

b. Printout

FIGURE 4

```
0.  METEOR((
1.  (CHANGE    ($ ROSE)  (FLOWER)  (/ (*Q SHELF1  " PRETTY )) CHANGE)
2.  (*    ($)   ((*A SHELF1) 1)   (/(*D PNTRET RULE3))   *)
3.  (PRNTWS * ((*P THE WORKSPACE IS))    PNTRET)
4.  (RULE2   ($)                        END)
5.  (RULE3   (($.1) ($.1)) O (/(*S ODD 1) (*Q EVEN 2) (*D PNTRET RULE3))
6.        PRNTWS   (THIS IS A CONTINUATION OF THE PREVIOUS CARD))
7.  (*     ($)   ((*A ODD)   (*N EVEN))   (/ (*Q ODD (*N EVEN) ONLY)
8.         (*P ODD EVEN)   (*D PNTRET RULE2))     PRNTWS)
9.  )(A ROSE IS A ROSE IS A ROSE))
```

a. Program

```
(THE WORKSPACE IS)
(A PRETTY FLOWER IS A PRETTY FLOWER IS A PRETTY FLOWER)
(THE WORKSPACE IS)
(FLOWER IS A PRETTY FLOWER IS A PRETTY FLOWER)
(THE WORKSPACE IS)
(A PRETTY FLOWER IS A PRETTY FLOWER)
(THE WORKSPACE IS)
(FLOWER IS A PRETTY FLOWER)
(THE WORKSPACE IS)
(A PRETTY FLOWER)
(THE WORKSPACE IS)
(FLOWER)
(SHELF ODD CONTAINS (IS ONLY))
(SHELF EVEN CONTAINS (PRETTY IS PRETTY))
(THE WORKSPACE IS)
(A FLOWER A FLOWER A PRETTY)
```

b. Printout

METEOR program will come to a normal halt if it executes a rule for which there is a left half match ($ always gives such a match) and whose "go-to" is the atom "END". If no such rule ends the program an error occurs, METEOR complains, and prints out an error statement.

Comments Inside a METEOR Program

The list to the right of the second "*" in some rules are comments which are ignored by the interpreter. However, they take valuable space within computer storage (they are read in) and thus comments should be used sparingly if space is tight.

Flow of Control

In the program shown in Figure 1 the rules were executed sequentially, and no rule was executed more than once. However, repeated operations can be done by a rule. In order to apply the same rule to the workspace several times, we give this rule a name, which we write instead of the left-hand "*". The name of the first rule in the program in Figure 3 is CHANGE. This name is also inserted instead of the right hand "*". As long as the left half finds a match in the workspace, the transformation indicated by the right half will be made, and control goes to the rule named by the atom replacing the right hand "*", in this case CHANGE again. The workspace is searched from left to right each time the rule is entered. CHANGE finds the first occurrence of "A ROSE", changes it to "THE FLOWER" and repeats. When there are no more occurrences of "A ROSE" in the workspace, the left half "fails", no transformation is made and control goes to the next rule in sequence. RULE1 is the next rule. RULE1 tests to see if there is an occurrence of "FLOWER" in the workspace. If there were not, RULE2 would be interpreted next. However, in this case, since we have just inserted into the workspace several occurrences of "FLOWER", the left half succeeds and control goes to RULE3.

RULE3 tests for the occurrence of "ROSE" in the workspace,

and if there were such an occurrence control would go back to CHANCE. There are none in this case and control passes to the next sequential rule--in line 5 (an unnamed rule).

It is sometimes desirable to reverse the normal flow of control, and to go to the next rule when the left half finds a match and the transformation is made, or go to the rule specified in the "go-to" on failure of the left half. For example, this allows the program to exit from the middle of a multi-rule loop on failure of a condition. This type of flow is indicated in line 5 and line 6 by the "*" as the second element of the rule. Since line 5 is this reverse-flow type of rule, and since there are no occurrences of "ROSE" in the workspace, the left half fails and control goes to RULE2, which prints out the workspace.

To review, normal flow is to the specified rule on left half success, and to the next sequential rule on left half failure. If the second element of a rule (immediately following the name or first "*") is a "*", this flow of control is reversed. "*" in the go-to specifies the next sequential rule.

Temporary Storage (Shelves)

In the example in Figure 3, each time the rule CHANGE is entered, the entire workspace is searched. Successive searches can be made more efficient by removing material already searched, and placing it on a temporary storage area called a shelf. The program in Figure 4 stores material in one such area called "SHELF1".

Temporary storage on these shelves is controlled by instructions in the "routing" section of the rule; the routing instruction in line 1 of Figure 4 queues onto the right end of SHELF1 (in a first-in first-out list) the items associated with 1 by the left half (i.e., the ones matched by the $), and the word "PRETTY". This is repeated as many times as the left half match succeeds.

FIGURE 5

```
METEOR((
(*   (($.1))      $)
(BLAH ($)   END)
(RULE ($)   END)
)(RULE BLAH))
```

FIGURE 6

```
0.  METEOR((
1.  (DEAL  NUM ($.1)) (CARD ($.1))) ( (FN ADD1 NUM))(/ (*S *NUM
                                                        CARD))*)
2.  (* * ( ($.2))                    PRINT)
3.  (* ((NUM 5))   (1)               DEAL)
4.  (*    ($)                        DEAL)
5.  (PRINT   ($) (/ (*P /))   END)
6.  )(1 H1 H2 H3 H4 C1 C2 C3 D1 D2 D3 D4 S1 S2 S3 S4))
```

a. Program

```
((4 (S4 D4 C4 H4)) (3 (S3 D3 C3 H3))  (2 (S2 D2 C2 H2))
                                        (1 (S1 D1 C1 H1)))
```

b. Printout

When control is finally transferred to the rule in line 2 of Figure 4, All the material on the shelf is re-inserted into the workspace. It is called in by the list "(*A SHELF1)" in the right half of the rule. A "(*N SHELF1)" would retrieve the Next or first item only of this shelf.

Any type of item which can be inserted into the workspace by a right half rule can be put onto a shelf through a routing instruction. This is illustrated by the rule on lines 7 and 8.

Another type of routing instruction is illustrated in line 2. This "*D" routing instruction makes RULE3 the "go-to value" of PNTRET. Thus after transferring to PRNTWS, and interpreting this rule, the interpreter treats the go-to "PNTRET" as if it were "RULE3". This "go-to value" for PNTRET is reset again in line 8. Setting the "go-to value" of a return allows easy access and return from standard routines such as print routines or read routines. Figure 5 shows a method for communicating with recursive subroutines. The "$" in the go-to implies that the first element of the workspace (not its subscript as in COMIT) is the "go-to" for this rule. If a stack of returns is stored on a shelf (in a pushdown list) and inserted into the workspace, successive returns can be made from recursive subroutines. This of course leaves a residue (the return) in the workspace.

Figure 5 illustrates three features of METEOR. In the left half, names other than integers have been associated with the pattern elements. "NUM" is associated with the first "($.1)" and "CARD" with the second by making each left-half element a two element list with a name first and the pattern element second. If a match is found, the first element in the workspace will be associated with "NUM", and the second "CARD", not with 1 and 2, respectively, as previously. Thus in line 3, the "1" in the right half names itself, i.e., the integer 1.

In the right half, a LISP function "ADD1" (tagged with FN

FIGURE 7

Full Printout Given by LISP for a Sample METEOR Program

```
FUNCTION EVALQUOTE HAS BEEN ENTERED, ARGUMENTS..
METEOR
(((* DICT (BOY ((BOY / NOUN HE))) (GIRL ((GIRL ((GIRL / NOUN SHE))))
(LOOKUP ((WORD ($ . 1))) O (/ (*Q SENT (FN GETDCT WORD DICT))
     (*P SENT)) LOOKUP) (* ($) ((*A SENT)) END))
(THE BOY AND GIRL)

(SHELF SENT CONTAINS (THE))
(SHELF SENT CONTAINS (THE (BOY / NOUN HE)))
(SHELF SENT CONTAINS (THE (BOY / NOUN HE) AND))
(SHELF SENT CONTAINS (THE (BOY / NOUN HE) AND (GIRL / NOUN SHE)))

END OF EVALQUOTE, VALUE IS..
 THE (BOY / NOUN HE) AND (GIRL / NOUN SHE)))
```

FIGURE 8

```
 FUNCTION EVALQUOTE HAS BEEN ENTERED, ARGUMENTS..
METEOR
(((* ((BOY / NOUN SING)) ((*/ AND 1 (DOG / NOUN MALE))
(*/ OR 1 (BOY / SMALL MALE)) (*/ SUBST 1 (MAN / MALE)))
END)) (THE (BOY / NOUN SING SMALL) AT HOME))

END OF EVALQUOTE, VALUE IS..
(THE (BOY / NOUN) (BOY / NOUN SING SMALL MALE) (BOY / MALE) AT HOME)
```

FIGURE 9

```
METEOR
(((* (($.1) IS ($.2) $ THERE) ((*K 1 2 3 4)) END))
(WHO IT IS AT MY DOOR IS THERE NOW))
(WHO (IT IS AT MY DOOR IS) NOW)
```

FIGURE 10

```
METEOR
((( * (IS ($.1)) (1 (*E 2)) END))
(IS (ANYBODY AT HOME) NOW))

END OF EVALQUOTE, VALUE IS..
(IS ANYBODY AT HOME NOW)
```

for the METEOR interpreter) increments "NUM by 1.

In the routing instruction, the "*S" instruction stores "CARD" on a shelf. This shelf name is computed and the shelf name is specified indirectly. The "*" immediately after the *S indicates this indirect addressing. The "NUM" following the "*" specifies the name of the shelf; that is, the shelf name is the number associated with NUM by the left half match (i.e., before incrementing). The program in Figure 6 thus deals out "cards" in the workspace onto the four shelves 1, 2, 3 and 4 (not randomly though).

Dictionary Rule and Retrieval

Figure 7 illustrates a type of rule which stores definitions of words for very fast, hash-code retrieval. This type of rule is indicated by a second element which is atomic (not a list) and which is not "*". The remainder of the list is interpreted as a list of associated pairs, and retrieval done by the LISP function GETDCT obtains the second member, given the first. The retrieval method is faster than a binary search for lists of less than 16,000 atoms (more than will fit in the 7090), and needs no preliminary sorting. New definitions can be added at any time.

Figure 8 illustrates the use of subscripted (labeled) atoms and the three modes by which subscripts can be merged; i.e., intersection, union, substitution.

The program in Figure 9 shows how several elements of the workspace can be "compressed" (by the *K in the right half) so as to be treated as a single item, a list of these elements. The program in Figure 10 performs the inverse of this compression operation and expands (with the *E) a list which is a single item in the workspace, and brings the elements of this list to the "top level" of the workspace.

A list of characters can be compressed (with a *C operator on the right) to form a single atom whose print name is this string of characters. An atom can be expanded into a list of its characters

FIGURE 11

```
METEOR
(((*(($.1))   ( (*E =))      END))     (GARBAGE PILE))
(G A R B A G E PILE)

METEOR
((( * ( ($.1 ) ($.2))     ( (*C 1 2))    END)) (F O O ON ME))
(FOO ON ME)
```

FIGURE 12

```
(RULE1 * ( ($.1) $ ) (2 1)  (/ (*S SHOT 1)) NEXT  (COMMENT) )
```

| Rule name | Go-to reverse | Left-half Pattern | Right half | Routing Section | Go-to | Comment |
|---|---|---|---|---|---|---|

by a *E operation in the right half. These operations are illustrated in the two programs in Figure 11.

This has been a very brief survey, by example, of some of the types of operations that can be done within a METEOR program. The remainder of the memo gives exact specifications for the program. Not mentioned were input, and character string output operations.

## III. Specifications for a METEOR Program

METEOR is a LISP function of two arguments, RULES, a list of the transformation rules to be applied, and WORKSPACE, the list or string to which these transformations are to be applied. The flow of control from one rule to another will be described below. Figure 1 is an example of a METEOR program, and its use under the LISP system.

### A. A METEOR Rule

An individual METEOR rule is a list of not less than two nor more than seven elements. An example of a complete rule is found in Figure 12. The first element of the rule is called the "rule name", and it must be present. It must be a LISP atom, and is either the atom "*" or a unique atom (i.e., no other rule may have this name).

The second element of the list is optional. It too may be "*" or another LISP atom. If it is "*", normal flow of control is reversed. Normal flow of control in a METEOR program is as follows:

1. If a match is found between the pattern element of the rule and the workspace, control is passed to the rule specified by the atom in the "go-to" section of this rule.

2. Otherwise control passes to the next rule in the list RULES. If the second element of the rule is an atom other than "*", for example "DICT", the remainder of the rule is interpreted as a list of dictionary entries to be made. When the entries are made, con-

trol will automatically pass to the following rule. The list of dictionary entries is a list of pairs which is used as an argument for the LISP function DEFLIST. The first member of each pair must be an atom and the second a dictionary entry for this atom. The dictionary entry is stored permanently (for the entire LISP run) on the property list of its atom. The element which introduced the dictionary rule, in this case "DICT", is used as a flag to mark this entry on the property list of the atom. Thus several dictionary entries with different flags can be made for a single atom, and each may be retrieved later (by the function GETDCT, described below). Retrieval from the dictionary is very fast because LISP uses a hash-coded "bucket sort" to find the property lists of atoms.

The third element of the rule is a pattern statement which is used to select relevant items from the workspace. This third element must be present, and must be a list. If the workspace "matches" this pattern (how a match is achieved is described below), the rest of the rule is interpreted. If not, immediate transfer is made to another rule.

The fourth element of the rule is the atom 0 (zero) or a list, which describes the transformed workspace. This element is optional. If it is not present, the workspace remains unchanged.

The fifth element is an optional list which is identified by its initial element, the atom "/". This list is called the routing section of the rule, and it controls temporary storage of data and multiple branching of flow of control.

The sixth elements is the "go-to" section and specifies to which rule in the list RULES control will pass. It must be an atom which is the name of some rule in the program, one which has been given a "go-to value" in the routing section of some previously executed rule, or the atom "END". If this sixth element is not present, it is assumed to be the "*".

The seventh element is an optional list ignored by the inter-

preter. It may be present only if the sixth element is present. Since it is ignored it may be used to insert comments on the program.

B. The Pattern Section of a Meteor Rule (the "left half")

This section, the third element of the rule list, is a list of patterns which must be matched in the workspace. A match is achieved if each of the individual patterns matches some element or elements in the workspace. These matched elements must be in the same order as the patterns appearing in the list of patterns and form a single contiguous substring in the workspace. Search is done from left to right, and the first match obtained is used.

The LISP function which obtains the match is called COMITMATCH. It is a function of two arguments, RULE and WORKSPACE. RULE is the list of patterns to be matched in the list WORKSPACE. Each pattern is associated with a name and if a match is achieved, the value of COMITMATCH is a list of pairs containing the name and the substring of the workspace matched by the pattern corresponding to this name. If no match is acheived, the value of COMITMATCH is NIL.

B1. Direct Match

If an element of the pattern list is an atom, it will match the first identical atom in the workspace. It will also match an item in the workspace which is a list, but whose first element is this atom and whose second element is "/". This latter match is useful if one wishes to label atoms in the workspace by attaching "subscripts" to them. COMITMATCH will match subscripted and unsubscripted items. The usual form for such a labeled atom is a list of the form

(atom / subscript1, subscript2...subscriptk)

For example, the atom "BOY" as a pattern element will match the list "(BOY ? NOUN SINGULAR)" appearing in the workspace. However "(BOY NOUN SINGULAR)" as a pattern element will not match "BOY" in the workspace. (See section 1c.) This type of direct match can

be done for any list structure which can be considered a single element. The element to be matched must be "quoted", i.e., be the second element of a list whose first element is the atom "QUOTE". "(QUOTE (A B C))" will match, as a single element in the workspace, the list "(A B C)". If the workspace were (M N (A B C) P), a match would be found for this sublist.

B2. "Dollars" Match

The pattern word "($.1)" will match any single element of the workspace. In general, the form "($.n)", where n is any integer, will match n consecutive constituents of the workspace. The atom "$" alone will match an indefinite number of elements of the workspace, including zero. Thus if "$" is the only member of the pattern list, there will always be a match, even if the workspace is null (empty).

B3. Subscript Match

A pattern word of the form

(element1 / subscript1 subscript2...)

will match a constituent in the workspace which has the same first element, then "/" and then a list of subscripts which include those mentioned in the pattern word. Additional subscripts may be present in the workspace element. Order of the subscripts is unimportant.

The first element "element1" of this pattern word may be either an atom which must be matched exactly in the workspace item, or be "($.1)" which will match any single element. One can thus find a word which has specified labelling (subscripting) without knowing the word itself. For example, "(($.1) / NOUN)" will match any element labelled as a noun, such as "(PLATO ? MAN NOUN GREEK)", in the workspace.

B4. Names for Left Half Elements

Each individual pattern in the left half is associated with

a name. When a match is found for this individual pattern, the matched portion of the workspace is paired with the associated name. The names associated with individual patterns in the pattern list are usually successive integers, i.e., "1" with the first pattern, "2" with the second, etc. For example, in Fig. 12 the name associated with the "($.1)" is "1" and with BE is "2". Thus, if a match were found for this pattern, 1 would be associated with the element in the workspace immediately preceding "BE", and "2" would be associated with the occurrence of "BE" in the workspace.

To associated a name other than the integers with a pattern element, for example with "($.1)" in this rule, the pattern element itself may specify another name. For example, this element in the pattern is written "(FIRST ($.1))", associating the name "FIRST" with the element match by "($.1)". Associating names can be done similarly for any element in a left half pattern. The pattern element P is replaced by a two element list containing first, the mnemonic name, and second, this pattern element P. The element matched by P will be associated with the mnemonic name, if one is given, or with an integer giving the position of the pattern element in the left half (but not both).

B5. Matching with Left Half Names

Matching with the left half pattern is done from left to right in the workspace. To determine if an element appears twice in the workspace, a match can be found for a single element, and the name associated with this matched element can be used later in the left half to obtain a second match for this element. For example, if the left half pattern were "(($.1) BE, $, 1)", the "($.1)" would match the word preceding "BE" in the workspace, and the fourth element of this pattern would match a later occurrence of this same word. The first occurrence of the word would be associated (in this

Figure 13

Function Definition

```
DEFINE(((
      (CARMATCH (LAMBDA (WKSPACE LIST) (COND
            ((EQUAL (CAR LIST) (CAR WKSPACE))
            (COMS (CAR LIST) (CDR WKSPACE)))
      (T NIL))))
       ))
```

Rule

```
(*   ( (FIRST ($.2)) (SECOND (FN CARMATCH FIRST)))
          (SECOND FIRST)     *)
```

case) with the name "1", and the second occurrence with the name "4". Of the workspace were "(THIS COULD BE THE WORD COULD)" then the left half pattern above would match the workspace, and the list of associated pairs would be: (( 1 COULD) (2 BE) (3 THE WORD) (4 COULD)).

B6. LISP Functions for Matching

In addition to these standard patterns for matching, any LISP function can be used to determine a match. This may be a LISP function of any number of arguments, where the first argument is the workspace, and the rest are items found previously by the match. This function is used in the left half pattern in the following format:

(FN function name1, name2,..., namek)

"FN" is the signal used by the interpreter to mark this type of function match. For "function" one may insert the name of any function previously defined in LISP. Name1,..., namek are names associated with elements previously matched in this left half pattern. If there are k such names, the function must have k+1 arguments: the first argument is the remainder of the workspace (the part not yet used in the match), and the other arguments are elements of the workspace associated with name1,...,namek. The value of this function should be NIL if no match is found. It should be cons[m;w] if there is a match, where m is the portion of the workspace that is matched, (or some function of this matched string) and w is the remainder of the workspace past the matching elements. Figure 13 illustrates a use of such a function. CARMATCH will find a match if the first element of the workspace is the same as the first list element (CAR) of the list whose name is given as the second argument of CARMATCH. (The workspace is the implicit first argument of CARMATCH). If there is a match, CARMATCH returns with the matching element labelled with the subscript "SECOND".

B7. Printing the Workspace

If "((*P message))" is used on the left side, no match will be found, but the "message" will be printed out, followed by a print-out of the workspace. This is a useful function for debugging. The workspace remains unchanged.

B8. Matching Special Characters

Seven characters cannot be read by the LISP reader because they have syntactic meaning within LISP. They are "(",")", ",", "-", "+", "," and "(blank)". These may be read in, however, by the METEOR reader (see below). To write a rule which tries to match one of these characters in the workspace--or to insert such characters in the workspace--use the following lists respectively "(*LPAR)", "(*RPAR)", "(*COMMA)", "(*PLUS)", "(*PERIOD)", "(*DASH)", AND "(*BLANK).

The inner workings of METEOR are as follows for this case, for those who are interested. The second item in a list started with a "*" is evaluated by the LISP function EVAL. The workspace is then matched against the correct unspeakable item, or said item is inserted into the workspace--whichever is appropriate. The "*" in this case is acting as an "unquote" operator.

C. Right Hand Side (Transformed Workspace)

If a match is obtained by the left half pattern, the interpreter then uses the right half (the fourth element of the rule) to determine in what way the workspace is to be transformed. Only those elements of the workspace utilized in the left half match can be affected by the right half transformation. If the right half is the atom "0" (zero) then all those items which were matched will be deleted from the workspace.

If the right half is not the atom "0", then if it appears at all, it must be a list. If the right half is not present in a rule, the workspace remains unchanged.

The right half is a list of elements. Each element will be replaced by the item or items it names, and the resulting string put in the workspace in place of the substring matched by the left half. The only way to effect the contents of the workspace is through this list (the right half). This differs from COMIT, which allows additions and deletions from the workspace under control from the routing section. This is not allowed in a METEOR program. All additions from temporary storage "shelves" are done directly from instructions in this right half.

C1. Substitutions, Insertions and Rearrangement

The names which appear in the right hand side list can be of several types. The first is a name associated with a matched element. Its value is its paired matching substring of the workspace. The same name may appear any number of times on the right hand side. Thus, elements of the workspace may be duplicated. The names may appear in any order, and elements can be rearranged.

The second type of name is an atom or list which is not a name of this first type, and not a list beginning with one of the control characters "*K, *C, *E, */, *N, *A, or *W". Such an element is a name for itself, and will be inserted directly into the workspace; an atom will be inserted and a list will be concatenated in the workspace in the position in which it appears in the right half list. For example, see Figure 14 below. The element associated with the

(* (DID, ($.1) GO) (2,DOES,3) *) Rule

| | Workspace |
|---|---|
| Before | (DID HE GO HOME TODAY) |
| After | (HE DOES GO HOME TODAY) |

Figure 14

name "2" (i.e., "HE") becomes the first element, the atom DOES is then inserted, and then the element associated with "3" is placed

in the workspace. Since the name "1" is used on the left side but not on the right, the element paired with 1 is deleted from the workspace. Since "HOME TODAY" is not mentioned in the pattern, it is not affected and remains at the end of the workspace.

To reiterate, the names usually associated with matched left half patterns are the integers which specify the position of the pattern. If another name is associated a left half pattern, say with the third one, then this integer in the right half, in this case "3", is a name for itself, and is itself inserted into the right half. To insert "3" in the right half without renaming the corresponding left half pattern, one can quote it in the right half, using the element "(QUOTE 3)".

C2. Compression and Expansion

Sometimes it is desirable to compress several elements of the workspace into a single unit, or list. This is accomplished by using a right half element which is a list whose first element is "*K". The succeeding elements of this list are of any type which can be found in a right half rule including other lists starting with "*K". These items are all placed in the workspace on a single list. Unlike COMIT, no restriction is placed on the order of the names in the list. Figure 15 (below) gives an example of this type of rule.

```
(* (WILL ($.1) ($.1))   (2 (*K 1 3)) *)
```

Workspace

| | |
|---|---|
| Before | (WHERE WILL HE STAY TONIGHT) |
| After | (WHERE HE (WILL STAY) TONIGHT) |

Figure 15

An item on the right may be of the form:

"(*C name1 name2...namek)"

where name1,..., namek identify elements which are single characters. This entire item on the right side will be replaced by an atom whose

print name is the list of characters specified. An error will result if one of the elements specified is not a single alphanumeric character.

The inverse of this compression operation can be performed by a list of the form

"(*E namel)".

If namel specifies an atom, this "expand operator" will be replaced by a list of the letters in the print name of this atom. If namel specifies a list, the list will be concatenated into the workspace. Figure 16 gives an example of this latter operation.

```
(* *IS,($.1))  (1 (*E 2)) *)
                    Workspace
          Before      (WHAT IS (A METEOR PROGRAM))
          After       (WHAT IS A METEOR PROGRAM)
```

Figure 16

C3. Reading and Writing Operations

Strings of atoms may be written out by an element in the right half which is a list of the form:

(*W namel name2...namek).

The last part of this list is treated as if it were a right half rule. When it is evaluated, it is a list. The "*W" then causes the atoms in this list to be printed out without spaces. If these atoms are individual characters, then any sequence of characters can be printed. Remember that special characters such as "(" and ")" etc. must be referred to by "(*LPAR)" and "(*RPAR)" etc. To end the printline the last atom in the string must be "$EOR$". Any characters in the string past the "$EOR$" will not be printed out. The value of this operator (*W namel...namek) is NIL. Thus although it appears in the right half, it contributes nothing to the transformed workspace. For any element of the list to be printed which

is not atomic (i.e., a list) the string "***" will be printed.

Characters on cards may be read into the workspace (or onto shelves) by a list containing the single element *R; the list "(*R)" in a right half will be replaced by a list of the characters on the next card read. The card image will be read from tape if tape input is used. The list will end after the last non-blank character, and is a maximum of 72 columns. The first 72 columns are read from a card. If the card is blank this list will be empty.

For example, the rule:

(* ($) ((*R)) *)

will replace the contents of the workspace by a list of characters on the next card. If the card is blank, the workspace would then be the null. The rule

(* ($) ((*R)) *)

will concatenate this input list onto the right end of the workspace. As mentioned (*BLANK) will match blanks read in by this read operation, etc.

C4. Additions from Temporary Storage

Sometimes it is convenient to temporarily remove certain elements from the workspace. For example, while doing certain long searches, previously searched items may be placed on temporary storage areas called shelves (from the COMIT terminology). Material is placed on the shelves through instructions given in the routing section of a rule. Items can be returned from these shelves and deposited in the workspace by instructions in the right half rule.

A list in the right half consisting of the element "*A" followed by a shelf name will be replaced by the entire contents of the shelf, and the shelf will be emptied. For example, "(*A SH1)" in a right half rule will bring to the workspace the entire contents of shelf named "SH1".

A list of the form "(*N SH1)" will bring into the workspace the next (first) element of this shelf, i.e., the first element of the

list which this shelf contains. This first item is removed from the shelf by the operation. If the shelf is empty the "(*N SHL)" is ignored.

C5. Subscript Combination in the Right Half

An item of the workspace may have a subscript, or subscripts for labelling purposes. The format for such subscripted items is:

(atom / subscript1,...subscriptk).

An item may have its subscripts modified by operations in the right half. Modification is controlled by elements in the right half which are lists starting with the atom "*/". The second item of the modifier list specifies the method of combination for two sets of subscripts. The specifying item may be one of the atoms, "AND", "OR", or "SUBST". The third and fourth elements, $s_3$ and $s_4$, of the modifier list are the names of the items whose subscripts are to be merged. The third item may specify an atom in the workspace or a subscripted element, but the fourth must name an element which is a list whose second element is "/". An example of a modifier list which could appear in a right half is "(*/ OR BOY (MAN / MALE NOUN))". The resulting element in the workspace would be "(BOY / MALE NOUN)". If the second element in the modifier list is "AND", the subscripts of $s_3$ and $s_4$ will be merged by logical conjunction. If the intersection of the two subscript sets is empty, the resulting item would have no subscripts, and the item is made atomic, e.g., "BOY" instead of the list "(BOY /)".

If the second item is "OR", logical disjunction is used to combine the subscripts of $s_3$ and $s_4$. If the second item is a "SUBST" the subscripts of $s_4$ are substituted for those of $s_3$. Other methods of combination can easily be added to the program by modifying the LISP function SBMERGE. Figure 8 gives an example of the three types of subscript modification.

D1. The Routing Section of a Rule

The routing section is a list whose first element is the atom

"/". Each subsequent element is a list, called a routing instruction which begins with one of the atoms "*S", "*Q", "*X", "*P", or "*D". The next element after "*S", "*Q", or "*X" in a routing instruction names a shelf to be used. For example (*S SH1 NAME1) will store (*S) on the shelf SH1, the items which are named by NAME1. Except for the atom "*", shelf names may be any LISP atom including numbers or atoms already used for rule names. Shelves and their contents are stored as pairs of the form (name, contents) on a list associated with the free variable SHELF. Each time a shelving operation is done, a search is made through the list SHELF. Thus shelving operations will be done more rapidly if fewer shelf names are used.

"*S" will cause items to be stored on the front of the shelf named. The item last stored by a "*S" instruction will be the first obtained by a "*N" item in a right half instruction. A shelf built up by "*S" instructions is a push down list with the last item in first out.

A queue (a first-in first-out list) can be built up using the "*Q" routing instruction. "*Q" queues items onto the "back end" of the shelf named.

The name of the shelf to be used may be specified directly as previously indicated or may be obtained from the workspace. If the second element of the routing instruction is a "*", then the third item is the name of a workspace item (the name as found by the match routine). This workspace item is the name of the shelf to be used. This differs from the COMIT method for indirect addressing. COMIT uses a subscript of the workspace item to specify the name of a shelf indirectly. METEOR may be easily changed in this respect by changing the function names "INDIRECT".

The items to be placed on the shelf named are specified by the remainder of the routing instruction. An item to be shelved may be described in any way used to describe an item to be put into the workspace by a right half rule. In fact, the same LISP function (COMITRIN) is used both to collect on a list the items to be shelved,

and to arrange the transformed portion of the workspace. Copies of material in the workspace, new atoms, and material from other shelves may be placed directly onto a shelf in the same way they are placed into the workspace. This is again different from COMIT, where all items to be shelved must be moved through the workspace.

A routing instruction starting with the atom "*X" will exchange the contents of the shelf named with the contents of the workspace. Remember that names used in the routing instruction in this rule still refer to names associated with items found by the left half match with the original workspace.

A routing instruction starting with the atom "*P" will print out the message (SHELF s CONTAINS c) for each shelf "s" named in the list following the "*P". Only direct addressing of shelves is allowed. "c" is the contents of the shelf named as a list structure. This instruction is useful in debugging programs.

As an example, "(*P SH1 17 3.2 *Q)" in the routing instruction of a rule will print out the contents of the shelves named SH1, 17, 3.2, and *Q, in that order. (Note the variety of allowable names.) "*P /)" prints out the contents of all shelves.

D2. The Dispatcher Instruction in the Routing Section

The instruction "*D" in a routing instruction is the only one which actually affects the routing of control in the program. "*D" must be followed by two atoms. This pair of atoms is put as a pair on the list DISPCH. This list is searched each time a name is found in the "go-to" section of a rule, and the second member of the pair substituted for the first. If the first atom were, for example BRANCH, and the second were RULE1, then each time BRANCH occurs in the "go-to" section of a rule it is interpreted as a "go-to" to the rule named RULE1. Thus, "*D" provides a method of setting a switch on an n-way branch. It may be reset at anytime to any value. Any atom in the "go-to" section for which no value has been set by a "*D" routing instruction is assumed to be its own "go-to" value.

E. The Go-To Section of a Rule

The sixth (optional) element of a rule is an atom which tells which rule is to be used next. If it is the atom "*" (or is absent) control passes to the next rule in the list RULES. If it is an atom, which has not been given a "go-to" value by a "*D" instruction in some routing instruction, then this atom is the name of the next rule to be used. If it has been paired with another atom by a "*D" instruction, this paired atom is the name of the next rule interpreted. If this atom is "$", the first item in the workspace is used as the name of the next rule executed. This latter differs from COMIT. COMIT uses a subscript on the first element. Normal flow of control is to the rule named by the "go-to" of a rule, if a match is achieved by the left half. If no match is achieved, the next rule, in the list RULES, is used. However, if the second item of a rule is "*", this normal flow pattern is reversed. No match implies transfer of control to the rule specified in the "go-to", and a successful match transfers control to the succeeding rule. If there is a match the transformation given by the right half of this rule and its routing instructions are always interpreted before control is transferred.

F. Comment Field

The seventh (optional) section is a list which may contain any comments on this rule or the program, etc. It is ignored by the METEOR interpreter. These comments will use up space within the computer and therefore should be used sparingly.

IV. Warnings and Advice

METEOR is based on COMIT and written in LISP and has foibles for three. The LISP system is built on the parenthesis and is very stubborn about having the correct number of parentheses in the proper place. Thus for the want of one parenthesis or a pair, an entire METEOR program may fail. Remember the following:

1. The left half of a METEOR rule must be a list of patterns. If the only element of this left half is a list, for example ($.1) or (*P THE WORKSPACE IS), make sure these elements are enclosed in parentheses, i.e., the left halves are respectively "(($.1))" and "((*P THE WORKSPACE IS))".
2. A similar warning holds for the right half. If the only element in the right half is for example "(FN ADD1 NUM)", the right half is "((FN ADD1 NUM))" (note the number of parentheses). To perform a deletion, however, using "0" (zero) as the right half, this zero must not be enclosed in parentheses. This is the only exception to the rule that the right half is a list.
3. Check the number of parentheses at the end of the routing section. Remember this section is a list of lists.
4. Remember that "(*R)" not "*R" reads in a card, i.e., "(*R)" is a list of one element. Thus a right side containing just a "(*R)" must be "((*R))".
5. Comments may be used, but they take up space. If space is a problem, delete or shorten your comments.
6. A linear search through a list of shelves is made each time a shelf reference is made. The name of any shelf ever mentioned is put on this list. To speed searches, use fewer shelf names.
7. Since METEOR is a LISP function, it may be used recursively within itself. However, METEOR resets "SHELF" and "DISPCH" to NIL. To keep the same shelf contents and dispatcher settings, use, instead of "(METEOR RULES WORKSPACE)", "(METRIX RULES WORKSPACE SHELF DISPCH TRACK)". The listing will make clear the reason for this ad hoc statement.
8. If METEOR is used within a LISP program, and not executed as a pair for EVALQUOTE, remember to quote the list of rules, because LISP evaluates arguments inside functions (this is because of a LISP peculiarity).

9. Best use of METEOR can be obtained by using it with the LISP system and expressing each operation in the language best suited for it. Remember LISP functions can call METEOR and vice versa.
10. Placing items on a shelf does not automatically remove them from the workspace. To remove matched items from the workspace, there must be a right half.
11. The LISP read program does not distinguish between commas and blanks; therefore, a "," and a " " (blank) are interchangeable in a METEOR program.
12. Another foible of the LISP read program is that "/" can appear as a character in the middle of an atomic symbol. Therefore, when using "/" to separate an element from subscripts, the "/" must have a blank (or a comma) on each side of it.
13. COMIT and METEOR differ in the following ways not previously mentioned explicitly: COMIT subscripts have values. In METEOR an element can have subscripts, but values are not explicitly provided for.
    A "*N" instruction is ignored (brings in a null element) if the shelf is empty in METEOR. It does not cause a rule failure in METEOR as it does in COMIT.
    There is no random element available in any form in METEOR--as opposed to the random rule selection feature available in COMIT.
14. Recall that the atom "*P" is used in two different contexts. When used in the left half it is followed by a message to be printed to label the following

    printout workspace.

    In the routing instruction *P is followed by a shelf name or series of shelf names, and the contents of these shelves will be printed. Followed by a "/", *P will cause the list "SHELF" to be printed. This is a list of pairs, the

first of each pair being the shelf name, and the second the contents of the shelf.

15. METEOR is presently buried in the LISP parenthesis system. However, using METEOR as a bootstrap, a read program can be built which can read and convert from any fixed-format parenthesis-free (or more free) METEOR. Since they are kept symbolically within the LISP system METEOR programs can be generated or modified at run time by a METEOR program (or LISP function) and then executed.

16. The function which assembles the right half of a rule, i.e., "COMITRIN", effectively strips one pair of parentheses from all non-atomic symbols named in the right half. Thus, if "FOO" is not a name used in the left half, (and is therefore a name for itself), "COMITRIN" will treat both "FOO" and "(FOO)" in exactly the same way. If the workspace were "(ON YOU)", then both

```
"(* ($) (FOO 1) *)"
```

and

```
"(* ($) ((FOO) 1) *)"
```

would change the workspace to "(FOO ON YOU)", and both

```
"(* ($) (FOO FOO 1) *)"
```

and

```
"(* ($) ((FOO FOO) 1) *)"
```

change the workspace to

```
"(FOO FOO ON YOU)"
```

In setting up dictionary entries, care must be taken to provide the proper parenthesis level. For example, the pair "(TWICE (TWO TIMES))" as a dictionary entry, when inserted by "GETDCT" in "(ALMOST TWICE THE NUMBER)" will change it to "(ALMOST TWO TIMES THE NUMBER)", which is presumably what was wanted. Similarly, if "(BOY (BOY / MALE))" is entered into "(THE BOY WINS)"

the result is "(THE BOY / MALE WINS)" which however is not what was intended at all. Always put one more pair of parentheses than you wish to appear in the workspace around the second element of a dictionary pair.

17. The *E instruction expands an atom into a list of the characters in the print name of the atom. It concatenates into the workspace a list which is a single element of the workspace. However, note the following eccentricity; if the workspace were "(THE GOOD BOY)", the rule "(* (($.2),) (*E 1)) *)" would transform the workspace to "(THE BOY)". The *E operator, operating on a substring from the workspace (in this case the two elements matched by ($.2)) specifies the <u>first</u> element of this substring, and the others are deleted from the workspace.

18. Two auxilliary functions are provided with METEOR, TRACERULE and UNTRACERULE, to be used as debugging aids. A rule

    "(* ((FN TRACERULE)) *)"

    will cause no change in the workspace, but will cause a printout before each METEOR rule interpreted. The printout is:

    a. the workspace before interpretation of a rule
    b. the rule to be interpreted.

    A rule

    "(* ((FN UNTRACERULE)) *)"

    will turn off all tracing.

19. The auxilliary function TIME can be used only with MIT LISP 1.5. It causes a time printout. It is used in the right half of a rule, and its value is NIL. Thus to preserve the workspace and printout the time, use a rule of the form:

    "(* ($) (1 (FN TIME)) *)"

20. I would appreciate hearing about any bugs or eccentricities found in the METEOR program, and applications found for this language.

V. Listing as of April 24, 1963

```
        SETSET  M2447 716 BOBROW  METEOR COMPILATION

(LAMBDA (X) (COMPILE (DEFINE X)))    ((
(METEOR
    (LAMBDA (RULES WORKSPACE) (METRIX RULES WORKSPACE
NIL NIL NIL)))
(METRIX
      (LAMBDA (RULES WORKSPACE SHELF DISPCH TRACK) (PROG (PC GT A)
(SETQ RULES (MAPLIST
RULES (FUNCTION (LAMBDA (X) (PROG (A B)(SETQ B(CAR X)) (SETQ A
(LIST (CAR B)))  (SETQ B (CDR B))  (COND ((NOT (ATOM (CAR B)))
(GO NTATM)) ((NOT (EQ (CAR B) (QUOTE *))) (RETURN (CAR X))))
(SETQ A (ADDLAST A (CAR B)))  (SETQ B (CDR B))
NTATM
(RETURN (NCONC A (CONS (NAMER (CAR B)) (CDR B)))))))))
(SETQ PC RULES) START (COND ((NULL PC    ) (RETURN(LIST(QUOTE(NO END))
WORKSPACE RULES))))
(COND ( (NULL TRACK) (GO TRACK)))
(PRINT (QUOTE RULE))  (PRINT (CAR PC)) (PRINT (QUOTE WORKSPACE))
(PRINT WORKSPACE)    TRACK
(SETQ GT (DISPATCH (COMITRULE (CDAR PC))))
(COND((EQ GT   (QUOTE *))(GO NEXT))((EQ GT  (QUOTE  END))(RETURN
WORKSPACE)) ((EQUAL GT   (CAAR PC))(GO START)))
(SETQ A (TRANSFER GT RULES))    (COND
((EQ (CAR A) (QUOTE NONAME)) (RETURN (LIST A WORKSPACE
(LIST (QUOTE (FROM RULE)) (CAR PC)) (LIST (QUOTE (SHELF IS)) SHELF)))))
(SETQ PC A)
(GO START)   NEXT (SETQ PC (CDR PC))  (GO START))))
(TRANSFER
(LAMBDA (GT RL)  (PROG ()
START (COND
((NULL RL  )(RETURN (LIST (QUOTE NONAME) GT)))
   ((EQ GT (CAAR RL)) (RETURN RL)))   (SETQ RL (CDR RL)) (GO START))))
(DISPATCH
      (LAMBDA (GT)  (PROG (A)
(COND ((EQ GT (QUOTE *)) (RETURN GT)))
(SETQ A (GTPAIR GT DISPCH))
    (COND  ((NULL A)  (RETURN GT)))  (RETURN  (CAR A)))))
(GTPAIR
     (LAMBDA (NAME X) (PROG (A)
START   (COND ((NULL X) (RETURN NIL))
((EQUAL  (CAR X)  NAME)  (RETURN (CDR X))))
(SETQ X (CDDR X))  (GO START))))
(COMITRULE
          (LAMBDA (RULE      ) (PROG (LEFT A B C D E)
(SETQ A (CAR RULE))  (SETQ E STAR)
(COND ((NOT (ATOM A)) (GO START)
((EQ A (QUOTE *)) (GO STAR)))  (DEFLIST (CDR RULE) A)
(RETURN (QUOTE *))      STAR
(SETQ RULE (CDR RULE)) (SETQ E (FSTATM  RULE))
START (SETQ LEFT (COMITMATCH2(CAR RULE)   WORKSPACE))
(COND ((NULL LEFT) (RETURN E)))
```

```
LOOP (SETQ RULE (CDR RULE))     (SETQ A(CAR RULE))
(COND ((NULL RULE) (RETURN  (QUOTE *)))
((EG A (QUOTE $)) (SETQ A (CAR WORKSPACE)))
((EQUAL A O)  (GO ON))
((ATOM A) (GO SW)) ((EG (CAR A)  (QUOTE /))  (GO SV)) (T (GO ON)))
SW  (COND ((EG E (QUOTE *))  (RETURN A)))
(RETURN (QUOTE *))  ON
(SETQ WORKSPACE (COMITR LEFT A))    (GO LOOP)
SV (SHELVE LEFT A) (GO LOOP))))
(FSTATM
        (LAMBDA (RULE) (PROG (A)  START (SETQ A (CAR RULE))
(COND ((NULL RULE) (RETURN (QUOTE *)))
((EQUAL A O) (GO ON ))  ((ATOM A) (RETURN A)))  ON
(SETQ RULE (CDR RULE))  (GO START))))
(SHELVE
        (LAMBDA (PAIRS INST) (PROG (A B C D)     START
        (SETQ INST (CDR INST))    (COND (( NULL INST) (RETURN SHELF)))
(SETQ A (CAR INST))     (SETQ B (CAR A))  (SETQ C (CADR A))
SETQ D (CDDR A))    (COND
((EQ B (QUOTE *P))(GO PR))
((EQ B (QUOTE *D))    (RETURN (SETDIS C (CAR D))))
                          ((NOT (EQ C (QUOTE *))) (GO GETD)))
(SETQ C (INDIRECT (CAR D) PAIRS))    (SETQ D (CDR D))
GETD (SETQ D (COMITRIN PAIRS D))
(SETQ A (GTSHLF C))
(COND ((EQ B (QUOTE *S)) (GO ST1)) ((EQ B (QUOTE *Q))(GO QU1))
((EQ B (QUOTE *X)) (GO EX)))
(PRINT (LIST (QUOTE (SHELVING ERROR IN)) (CAR INST)))    (GO START)
PR (COND ((EQ C (QUOTE /)) (RETURN (PRINT SHELF))))     PRI
   (PRINT (LIST (QUOTE SHELF) C (QUOTE CONTAINS)  (CAR (GTSHLF C))))
(COND ((NULL D) (GO START))) (SETQ C (CAR D))
(SETQ D (CDR D))  (GO PR1)
EX   (SETQ B (CAR A)) (RPLACA A WORKSPACE)  (SETQ WORKSPACE B)(GO START)
QU1   (RPLACA A (NCONC (CAR A) D))    (GO START)
ST1  (RPLACA A (APPEND D (CAR A)))     (GO START))))
(SETDIS
     (LAMBDA (X Y) (PROG (A)
(SETQ A (GTPAIR X DISPCH))
(COND ((NULL A) (SETQ DISPCH (CONS  X (CONS Y DISPCH))))
(T (RPLACA A Y)))      (RETURN DISPCH))))
(GETDCT
       (LAMBDA (X Y)  (PROG (A)  (SETQ A (GET X Y))
(COND ((NULL A)  (RETURN X)))     (RETURN A))))
(INDIRECT
          (LAMBDA (X PAIRS) (GINAME X PAIRS)))
))
(LAMBDA (X) (COMPILE (DEFINE X)))   ((
(COMITR
        (LAMBDA (LEFT ORDER)(PROG (A B C)
(SETQ A (GINAME O LEFT))    (COND ((EQUAL A O)(SETQ A NIL))
((NULL A) (GO ON))  ((ATOM A) (SETQ A (LIST A))))
ON   (SETQ B (GTNAME (QUOTE WSEND) LEFT))
(COND((EQUAL ORDER O) (SETQ C NIL))
(T (SETQ C (COMITRIN LEFT ORDER)))) (RETURN (APPEND A (APPEND C B))))))
```

```
(COMITRIN
          (LAMBDA (LEFT ORDER) (PROG (A B)
START (COND ((NULL ORDER)(RETURN A)))
(SETQ B (GTNAME (CAR ORDER) LEFT)) (COND ((NULL B) (GO ON))
(( ATOM B) (SETQ B (LIST B))))      ON
(SETQ A (APPEND A B))
(SETQ ORDER (CDR ORDER)) (GO START) )))
(GTNAME
         (LAMBDA (NAME PRS) (PROG (A B C)(SETQ C (CAR NAME))
(COND ((ATOM NAME)(GO START))
      ((EQ C (QUOTE FN ))(RETURN (APPLY (CADR NAME)
(COMITRIN PRS (CDDR NAME))   NIL)))
((EQ C (QUOTE *K)) (RETURN (LIST (COMITRIN PRS (CDR NAME)))))
((EQ C (QUOTE *C)) (RETURN (COMPRESS (COMITRIN PRS (CDR NAME)))))
((EQ C (QUOTE *)) (RETURN (EVAL (CADR NAME) NIL)))
((EQ C (QUOTE *W))  (RETURN (WRITES (COMITRIN PRS (CDR NAME)))))
 ((EQ C (QUOTE QUOTE))  (RETURN (CDR NAME)))
((EQ C (QUOTE *E))(RETURN(EXPAND(GINAME (CADR NAME) PRS))))
((EQ C (QUOTE */))  (RETURN (LIST (SBMERGE (CDR NAME)))))
((EQ C (QUOTE *N))(RETURN (NEXT (CDR NAME))))
((EQ C (QUOTE *R))  (RETURN (MTREAD)))
((EQ C (QUOTE *A)) (RETURN (ALL (CDR NAME))))((EQ C (QUOTE QUOTE))
(RETURN (CADR NAME))))
START (COND ((NULL PRS) (RETURN NAME))) (SETQ A (CAR PRS))
(COND ((EQUAL NAME (CAR A))  (RETURN (CDR A))))
(SETQ PRS (CDR PRS))   (GO START))))
(EXPAND
        (LAMBDA (X) (COND
((ATOM X)  (MAPCON (GET (CDR X) (QUOTE PNAME))
(FUNCTION (LAMBDA (Y) (UNPACK (CAR Y))))))
(T   (CAR X)))))
(COMPRESS
       (LAMBDA (X)   (PROG ()
(CLEARBUFF)   (MAP X (FUNCTION (LAMBDA (X) (PACK (CAR X)))))
(RETURN (INTERN (MKNAM))))))
(MTREAD
   (LAMBDA () (PROG (A B C)        (SETQ A (STARTREAD))   (GO A)
START   (SETQ A (ADVANCE))    A  (COND ((EQ A  (QUOTE $EOF$))(RETURN A))
((EQ A  (QUOTE $EOR$))   (RETURN B))
((EQ A BLANK)   (SETQ C (NCONC C (LIST A))))
(T (GO B)))   (GO START) B(SETQ B (NONC B  (NONC C (LIST A))))
(SETQ C NIL)   (GO START))))
(ALL
    (LAMBDA (X)   (PROG (A B)
(COND ((EQ (CAR X) (QUOTE *))  (SETQ X (INDIRECT (CADR X) PRS)))
(T   (SETQ X (CAR X))))
(SETQ A (GTSHLF X))
(SETQ B (CAR A))  (RPLACA  A NIL)  (RETURN B))))
```

```
(NEXT
     (LAMBDA (X)    (PROG (A B C)
(COND ((EQ (CAR)     (QUOTE *))   (SETQ X (INDIRECT (CADR X) PRS)))
(T    (SETQ X (CAR X))))
(SETQ A (GTSHLF X))
(SETQ C (CAR A))
(COND ((NULL C) (RETURN NIL)))
(SETQ B (CAR C))   (RPLACA A (CDR C))    (RETURN (LIST B)))))
(GTSHLF
     (LAMBDA (X)   (PROG (A)
(SETQ A (GTPAIR X SHELF))   (COND ((NULL A) (GO A)))
(RETURN A)    A (SETQ A (CONS NIL SHELF))
(SETQ SHELF (CONS X A))   (RETURN A))))
(SBMERGE
   (LAMBDA (X) (PROG (A B C D)
(SETQ A (CAR X))   (SETQ B (GTNAME (CADR X) PRS))   (SETQ C
(CDDR  (GTNAME (CADDR X) PRS)))
(COND ((ATOM B) (SETQ B (LIST B    (QUOTE /)))))
(SETQ D (LIST (CAR B) (QUOTE /)))
(SETQ B (CDDR B))
(COND ((EQ A (QUOTE AND))   (GO AND))   ((EQ A (QUOTE OR)) (GO OR))
((EQ A (QUOTE SUBST)) (GO SUBST)))
(PRINT (LIST (QUOTE (SUBSCRIPT ERROR)) X))
(RETURN (NCONC D B))
AND    (SETQ A NIL)A (COND ((NULL B)    (RETURN (NCONC D A)))
((MEMBER (CAR B) C)      SETQ A (ADDLAST A (CAR B)))))
(SETQ B (CDR B)) (GO A)
OR  (SETQ A NIL)B (COND ((NULL B) (RETURN (NCONC D (APPEND A C))))
((NOT(MEMBER (CAR B) C)) (SETQ A (ADDLAST A (CAR B)))))
(SETQ B (CDR B)) (GO B)
SUBST  (COND ((NULL C) (RETURN (CAR D)))) (RETURN (NCONC D C)))))
))
(LAMBDA (X) (COMPILE (DEFINE X)))   ((
(COMITMATCH
           (LAMBDA(RULE WORKSPACE)(COMITMATCH2 (NAMER RULE)WORKSPACE)))
(COMITMATCH2
           (LAMBDA (RULE WORKSPACE)(PROG (A B)(SETQ A(CMATCH RULE
WORKSPACE NIL)) (COND ((NULL A) (RETURN NIL))
((EQ A (QUOTE $IMP)) (RETURN NIL))) (SETQ B (CONS
(QUOTE WSEND) (CDR A)))   (RETURN (ADDLAST (CAR A) B)))))
(CMATCH
       (LAMBDA (RULE WORKSPACE MPAIRS) (PROG
(RNAME A B C D E G H)     (SETQ RNAME (CAR RULE ))
(SETQ RULE (CDR RULE))    (SETQ B (CAR RULE))
(COND ((NULL RULE) (RETURN ( CONS MPAIRS WORKSPACE)))
((EQ B (QUOTE $)) (GO PDOLL))
((NULL WORKSPACE) (RETURN (QUOTE $IMP))))
```

```
(SETQ H (CAR B))          (COND
((EQ  H       (QUOTE $))(GO NDOLL))
((EQ H (QUOTE *)) (GO EVAL))
((EQ   H       (QUOTE *P))   (GO PRINT))
((EQ (CADR B) (QUOTE /))   (GO SUBMCH))
    ((ATOM B) (GO ATB))
((EQ H (QUOTE FN)) (SETQ B (CDR B)))
((EQ H (QUOTE QUOTE)) (GO ATB1)))
(SETQ E (CONS WORKSPACE (MAPLIST (CDR B)( FUNCTION (LAMBDA (X)(GTNAME
(CAR X) MPAIRS))))))    (SETQ B (APPLY (CAR B) E NIL))
WATB  (COND ((NULL B) (RETURN NIL))
((EQ B (QUOTE $IMP)) (RETURN B))
(T (RETURN (CMATCH(CONS(CDR RNAME) (CDR RULE))(CDR B)
(ADDLAST MPAIRS (CONS (CAR RNAME) (CAR B)))))))
PDOLL (SETQ D (CDR RNAME)) (SETQ RULE (CDR RULE))
(COND ((NULL RULE) (RETURN (LIST (ADDLAST MPAIRS
(CONS (CAR RNAME) WORKSPACE))))))
DLOOP (SETQ B (CMATCH (CONS D RULE) WORKSPACE MPAIRS))
(COND ((NULL WORKSPACE) (RETURN NIL))
((EQ B (QUOTE $IMP)) (RETURN B))
(B (RETURN (CONS (ADDLAST (CARB)(CONS  (CAR RNAME) C    ))(CDR B)))))
(SETQ C (ADDLAST C (CAR WORKSPACE)))
(SETQ WORKSPACE (CDR WORKSPACE ))(GO DLOOP)
SUBMCH   (SETQ B (SUBMCH B WORKSPACE))     (GO WATB)
PRINT  (PRINT (CDR B))  (PRINT WORKSPACE)   (RETURN (QUOTE $IMP))
EVAL   (SETQ B (EVAL (CADR B)  NIL))   (GO ATB2)
ATB1 (SETQ B (CADR B)) (GO ATB2)
ATB   (SETQ B  (GTNAME B MPAIRS))
ATB2 (COND ((EQUAL B (CAR WORKSPACE)) (SETQ B WORKSPACE))
((EQ B (CAAR WORKSPACE)) (SETQ B WORKSPACE))
(T   (SETQ B NIL)))     (GO WATB)
NDOLL (SETQ G (CDR B)) (SETQ B (DOLNM G WORKSPACE    )) (GO WATB ))))
(NAMER
       (LAMBDA (X) (PROG (A B C D E G) (SETQ D O)
(SETQ A (CAR X))
(COND ((EQ A (QUOTE $)) (GO START)) ((EQ (CADR A) (QUOTE $))(GO START)))
(SETQ A (QUOTE $)) (GO IN)
START (SETQ D (ADD1 D))  (COND ((NULL X) (RETURN (CONS E C))))
(SETQ A (CAR X))  (SETQ B (CDR A))  (SETQ X (CDR X))
(SETQ G (CAR A))
IN (COND ((ATOM A) (GO SNAME))
((OR (EQ G (QUOTE $))
(EQ G (QUOTE FN))
(EQ G STAR)
(EQ G (QUOTE *P))
(EQ G (QUOTE QUOTE))
(EQ (CAR B) (QUOTE /)))
(GO SNAME))
((NULL B) (GO SNMA)))
```

```
(SETQ E (ADDLAST E(CAR A)))(SETQ A (CAR B))  (GO OUT)
SNMA   (SETQ A (CAR A))  SNAME (SETQ E (ADDLAST E D))
OUT   (SETQ C (ADDLAST C A))  (GO START )  )))
(SUBMCH
(LAMBDA (X Y) (PROG (A B)
    (SETQ B (CAR Y))  (COND
((AND (NOT (EQUAL (CAR X) (QUOTE ($.1))))
(NOT (EQ (CAR X) (CAR B)))) (RETURN NIL)))
(COND
((EQ (CADR B) (QUOTE /)) (GO ON)) (T (RETURN NIL)))
ON (SETQ B ( CDDR B))   (SETQ A (CDDR X))
START (COND ((NULL A) (RETURN Y))((MEMBER (CAR A) B)
(SETQ A (CDR A)))  (T(RETURN NIL)))  (GO START))))
(DOLNM
       (LAMBDA (NUM WSPACE) (PROG (A) (COND
((NOT (EQUAL NUM1)) (GO START)))
(COND ((ATOM (CAR WSPACE)) (RETURN WSPACE)))
(RETURN (CONS (LIST (CAR WSPACE)) (CDR WSPACE)))
START (COND ((EQUAL NUM 0) (RETURN (CONS A WSPACE)))
((NULL WSPACE)(RETURN (QUOTE $IMP)))) (SETQ A (ADDLAST A (CAR WSPACE)))
(SETQ WSPACE (CDR WSPACE ))   (SETQ NUM (SUB1 NUM))(GO START))))
(ADDLAST
         (LAMBDA (X Y) (APPEND X (LIST Y))))
(WRITES
     (LAMBDA (X)   (PROG (A)   START  (SETQ A (CAR X))
(COND ((NULL X)  (RETURN NIL))   ((EQ A (QUOTE $EOR$)) (GO ON))
((ATOM A)  (PRIN1 A))  (T  (PRIN1  (QUOTE ***))))
(SETQ X (CDR X))   (GO START)
ON   (TERPRI1)   (RETURN NIL)  )))
(PRNTWS (LAMBDA (X Y) (PROG () (PRINT Y)(PRINT X)
(RETURN (QUOTE $IMP)))))
(TIME (LAMBDA () (PROG () (TEMPUS, FUGIT) (RETURN NIL))))
(TRACERULE (LAMBDA (X) (PROG ()
(SETQ TRACK *T*) (RETURN (QUOTE $IMP)))))
(UNTRACERULE  (LAMBDA (X) (PROG ()
(SETQ TRACK NIL)  (RETURN (QUOTE $IMP)))))
))
STOP)))))))
```

*This blank page was inserted to preserve pagination.*

**CS-TR Scanning Project**
**Document Control Form**

Date : 11/30/95

**Report #** AIM-51

Each of the following should be identified by a checkmark:
Originating Department:

☒ Artificial Intellegence Laboratory (AI)
☐ Laboratory for Computer Science (LCS)

Document Type:

☐ Technical Report (TR) ☒ Technical Memo (TM)
☐ Other:_____

## Document Information

Number of pages: 41 (45-IMAGES)
Not to include DOD forms, printer intstructions, etc... original pages only.

Originals are:
☒ Single-sided or
☐ Double-sided

Intended to be printed as :
☐ Single-sided or
☒ Double-sided

Print type:
☐ Typewriter ☐ Offset Press ☐ Laser Print
☐ InkJet Printer ☐ Unknown ☒ Other: COPY (OF TYPEWRITER)

Check each if included with document:

☐ DOD Form ☐ Funding Agent Form ☐ Cover Page
☐ Spine ☐ Printers Notes ☐ Photo negatives
☐ Other: _____

Page Data:

Blank Pages(by page number):_____

Photographs/Tonal Material (by page number):_____

Other (note description/page number):

| | Description : | Page Number: |
|---|---|---|
| (A) | IMAGE MAP: (1-41) UNNUMBERED TITLE PAGE, 1-40 (42-45) SCANCONTROL TRGTS (3) | |
| (B) | PAGES 25-33 HAVE TEARS IN LEFT MARGINS | |

Scanning Agent Signoff:
Date Received: 11/30/95 Date Scanned: 12/7/95 Date Returned: 12/14/95

Scanning Agent Signature: Michael W. Cook

# Scanning Agent Identification Target

Scanning of this document was supported in part by the **Corporation for National Research Initiatives,** using funds from the **Advanced Research Projects Agency** of the **United states Government** under Grant: **MDA972-92-J1029.**

The scanning agent for this project was the **Document Services** department of the **M.I.T Libraries.** Technical support for this project was also provided by the **M.I.T. Laboratory for Computer Sciences.**

**Scanned**

**Date:** 12/7/1995

**M.I.T. Libraries**
**Document Services**