[*Mechanical Translation*, vol.5, no.1, July 1958; pp. 25-41]

A Programming Language for Mechanical Translation†

Victor H. Yngve, Massachusetts Institute of Technology, Cambridge, Massachusetts

A notational system for use in writing translation routines and related programs is described. The system is specially designed to be convenient for the linguist so that he can do his own programming. Programs in this notation can be converted into computer programs automatically by the computer. This article presents complete instructions for using the notation and includes some illustrative programs.

IT HAS BEEN SAID that the automatic digital computer can do anything with symbols that we can tell it in detail how to do. If we are interested in telling a digital computer to translate texts from one language into another language, we are faced with two tasks. We first have to find out in detail how to translate a text from one language to another. Then we have to "tell" the computer how to do it. This paper is concerned with the second task. We will present here a specially devised language in which the linguist can conveniently "tell" the computer to do things that he wants it to do.

The automatic digital computer has been designed to handle mathematical problems. It is able to carry out complicated routines in terms of a few different kinds of elementary operations such as adding two numbers, subtracting a number from another number, moving a number from one location to another, taking its next instruction from one of two places depending on whether a given number is negative or positive, and so on. In order to instruct the computer to carry out complicated routines, simple instructions for the elementary operations are combined into a program. The writing of a program to carry out even an apparently rather simple procedure can be an exacting task requiring a high degree of skill on the part of the programmer.

It has been the custom for the linguist who wanted to try out a certain approach to mechanical translation to ask an expert programmer to program his material rather than to learn the art of programming himself. Besides the usual inconveniences and difficulties attending the communication between experts in two separate fields, this practice has certain more basic difficulties: Neither the linguist nor the programmer has been able to be fully effective. The linguist has not become aware of the full power of the machine, and the programmer, not being a linguist, has not been able to use his special knowledge of the machine with full effectiveness on linguistic problems.

The solution offered here to these difficulties is an automatic programming system. The linguist writes the results of his research in a notation or language called COMIT, which has been specially devised to fill his needs. The programmer writes a conversion program or compiler capable of converting anything written in this notation into a program that can be run on the computer.* Thus the expense, time, and effort needed to separately program each linguistic approach is saved, and, even more important, the linguist is given direct access to the machine. He becomes more fully aware of its potentialities, and his research is greatly facilitated.

† This work was supported in part by the U. S. Army (Signal Corps), the U. S. Air Force (Office of Scientific Research, Air Research and Development Command), and the U.S.Navy (Office of Naval Research); and in part by the National Science Foundation.

* This is being done by the programming research staff of the M.I. T. Computation Center.

What COMIT Is

COMIT is an automatic programming system for an electronic digital computer that provides the linguist with a simple language in which he can express the results of his researches and in which he can direct the computer to analyze, synthesize, or translate sentences. It is capable of being programmed on any general purpose computer having enough storage and appropriate input and output equipment. The language has been devised to meet the needs of the linguist who wants to work in the fields of syntax and mechanical translation. Some of the linguistic devices and operations that COMIT has been designed to express are: immediate constituent structure, discontinuous constituents, coordination, subordination, transformations and rearrangements, change in the number of sentences or clauses in translation, agreement, government, selectional restrictions, recursive rules, etc.

A program written in COMIT consists of a number of rules written in a special notation. The computer executes these rules one at a time in a predetermined order. In seeking an appropriate notation in which to write the rules, we were guided by several considerations.

1. That the rules be convenient for the linguist - compact, easy to use, and easy to think in terms of.
2. That the rules be flexible and powerful — that they not only reflect the current linguistic views on what grammar rules are, but also that they be easily adaptable to other linguistic views,

A linguist can use the computer in the following simple way. He expresses the results of his linguistic research in COMIT. He transcribes his rules onto punched cards using a device with a typewriter keyboard. He supplies text or special instructions to the machine also on punched cards. He then gives these packs of cards to an operator and subsequently receives his results in the form of printed sheets from the machine.

The way that a COMIT program works in the computer is shown in figure 1. The rules making up the COMIT program can be thought of as stored in the computer at A. Material to be translated or otherwise operated on enters the computer under the control of the rules from the input B. It is operated on by the rules and translated in the workspace C. It then goes to the output E. The dispatcher D contains special information, stored there by the rules, which governs the flow of control or the order in which the rules of the program are carried out.

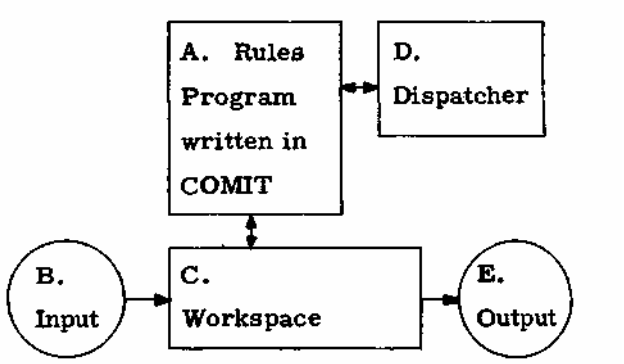


Fig. 1. How a COMIT program works in the computer.

The way in which COMIT rules are written, how they direct the computer to perform the desired operations, and how they are assembled into programs will now be described. The remainder of the paper is thus a complete manual of detailed instructions for using this special-purpose programming language.

COMIT Rules and Their Interpretation

A rule in COMIT has five sections, the name, the left half, the right half, the routing, and the go-to, each with its special functions. Figure 2 shows how a rule is divided into these

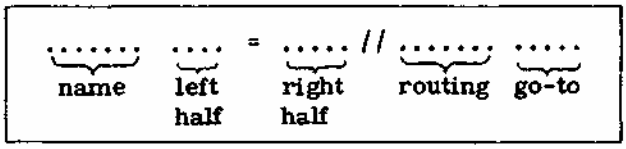


Fig. 2. The five sections of a rule in COMIT.

five sections. The name and left half are separated by a space, the left half and the right half are separated by an equal sign, the right half and the routing are separated by two fraction bars, and the routing and the go-to are separated by a space;

— flow of control —

We will discuss first the function of the name and the go-to, which have to do with the flow of control from one rule to another. A program written in COMIT always starts with the first rule in sequence. After a rule has been carried out, the computer obtains in the go-to the name of the next rule to be carried out. The name of each rule is to be found in the left-hand part of the name section of that rule. (The

right-hand part of the name section is reserved for the subrule name, to be discussed later.)

In addition there are three cases when control is automatically transferred to the next rule in sequence regardless of its name. One of these will be immediately clear; the other two will be clarified in the explanations of the left half and the routing. The three are: (1) an asterisk is written in the go-to, (2) the constituents written in the left half of the rule were not found in the workspace, (3) an *R in the routing finds no more material at the input. A rule to which control is always transferred automatically in this fashion so that a rule name is not needed, may have an asterisk in the name section in place of a rule name. When this automatic transfer of control takes place from the last rule in sequence so that there is no next rule, the COMIT program stops.

Figure 3 shows an example of how control proceeds from one rule to another under the direction of the rule name and the go-to sections. In this program, rule A would be the first one executed, then C, then the rule with an asterisk in the name section, then B, then C, then *, then back to B again, and so on round and round in what is known as a loop, until one of the conditions occurs in the rule marked asterisk that will automatically transfer control to the next rule D. After D has been executed, the program will stop.

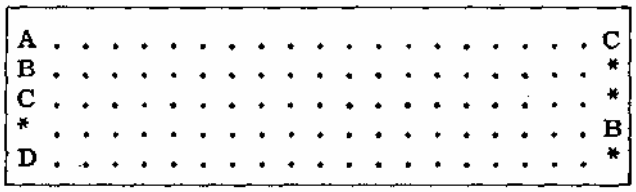


Fig. 3. A COMIT program to illustrate the flow of control under the direction of the rule name and the go-to sections of the rules.

As an aid to the memory, we will give a way in which each part of a rule in COMIT can be read in English. This will be done by providing English equivalents for all abbreviations used in COMIT, and by providing certain conventional wordings that will always be used between the various sections and between the various abbreviations. For the parts of the rule already discussed we need the following conventions: A rule is preceded by the word "in", rule names are preceded by the words "the rule", the go-to is preceded by the words "then go to", an * in the name section is read "this rule", an * in the go-to is read "the next rule, " and the rule is followed by a period to make a sentence. These conventions are enough to read the program in figure 3. These and the other conventions are conveniently tabulated in a later section. According to the conventions, the program in figure 3 should be read:

In/the rule A/... /then go to/the rule C/.
In/the rule B/... /then go to/the next rule/.
In/the rule C/... /then go to/the next rule/.
In/this rule /... /then go to/the rule B/.
In/the rule D/... /then go to/the next rule/.

The dispatcher also can influence the flow of control in the following way: A rule in COMIT may have several subrules. In figure 4, the rule B has four subrules. The rule name is

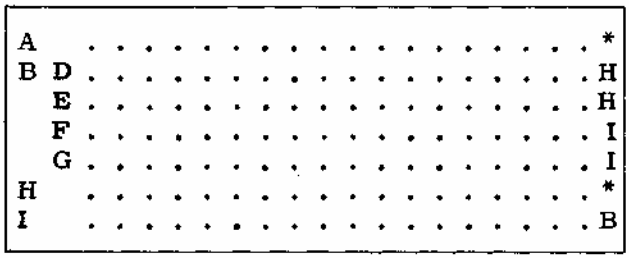


Fig. 4. A COMIT program to illustrate a rule with subrules. The rule B has four subrules.

in the left hand part of the name section of the first subrule. The name of each subrule is in the right hand part of the name section of that subrule. A rule that does not have several subrules may be thought of as a rule with just one subrule. A rule with only one subrule does not have a subrule name. When control is transferred to a rule with several subrules, the dispatcher is consulted for an indication of which subrule is to be carried out. For this purpose the dispatcher contains dispatcher entries. A dispatcher entry of the form B E would cause the computer to execute the subrule E in rule B each time it comes to that rule. If there is no entry in the dispatcher for this particular rule, or if there is an entry, but it contains more than one subrule name, the choice is made at random. In other words, if the dispatcher contains the entry B E G, the computer will choose at random between the two alternative subrules E and G. A dispatcher entry having a minus sign in front of its values (subrule names) has the same meaning as it would have if it had all its possible values except those following the minus sign. A dispatcher entry with a rule

name but no values has the same meaning as one with all possible values, that is, choose completely at random. The contents of the dispatcher are not altered by any of these processes. How the contents of the dispatcher may be altered will be discussed in the section on the routing.

The English reading of a rule with several subrules is the same as that for a rule with one subrule except that the words "consult the dispatcher and select" are read following the rule name. In figure 4, the rule B with four subrules is read:

In/the rule B/consult the dispatcher and select/
the subrule D/. . . /then go to/the rule H/.
the subrule E/... /then go to/the rule H/.
the subrule F/... /then go to/the rule I/,
the subrule G/... /then go to/the rule I/.

— workspace —

Having discussed the flow of control, we will turn to the workspace and describe how text to be translated or other material to be worked on is represented there. This will prepare us for a discussion of the remaining three parts of the rule whose function it is to operate on the material in the workspace.

Material is stored in the workspace as a series of constituents separated by plus signs. A constituent consists either of a symbol alone or a symbol and one or more subscripts. The symbol is written first. It may be the textual material itself, a word, phrase, or part of a word; or it may be any temporary word or abbreviation that the linguist finds convenient to use. Subscripts are of two kinds, logical subscripts and numerical subscripts. Logical subscripts are potential dispatcher entries and thus have the form of a rule name (subscript name) followed by one or more subrule names (values). Numerical subscripts are used for numbering and counting purposes. They consist of a period for the subscript name followed by an integer n in the range $0 \leq n < 2^{15}$. A constituent may have any number of logical subscripts, but only one numerical subscript.

An example of how linguistic material can be represented in the workspace is given in figure 5. This could be read in English as follows: "a constituent consisting of/the symbol IN/ with/the numerical subscript/1/ , followed by/ a constituent consisting of/the symbol DER/ with/the numerical subscript/2/ , followed by/ a constituent consisting of/the symbol ADJ/with/ the numerical subscript/3/ , and with/the subscript AFF/having/the value EN/ , followed by/a constituent consisting of/the symbol NOUN/ with/the numerical subscript/4/ , and with/the subscript GENDER/having/the value FEM/." The conventional wordings and the readings for the abbreviations used may be found tabulated near the end of this article.

```
. IN/ . 1 + DER/ . 2 + ADJ/ . 3, AFF EN
  + NOUN/ . 4, GENDER FEM
```

Fig. 5. Example of how linguistic material may be represented in the workspace.

- left half -

Having discussed the name and go-to sections and shown how material is represented in the workspace, we are now ready to discuss the remaining three sections of a rule. First we will take up the left half. A rule with several subrules may have no more than one left half. It is written in the first subrule. The function of the left half is to indicate to the computer which constituents in the workspace are to be operated on by the rest of the rule. The constituents in the workspace to be operated on are indicated by writing constituents in the left half that match them in certain definite respects.

A match condition between a constituent in the workspace and a constituent written in the left half will be recognized if the following conditions hold: (1) The symbols are identical. (2) If the constituent in the left half has any subscripts written on it, the constituent in the workspace must also have at least subscripts with the indicated subscript names — the order of writing the subscripts has no significance. (3) If the logical subscripts in the left half have any values indicated, the subscripts in the workspace must also have at least these values — again the order is unimportant. (4) If a numerical subscript is written in the left half, the numerical subscript in the workspace must have an identical numerical value, but if . G or . L is written in the left half before the value of a numerical subscript, a numerical subscript in the workspace will be matched if it has, respectively, a value greater than or less than the value written in the left half.

Dollar signs written in the left half have special meanings. $1 may be written in the left half to match any arbitrary symbol. If the $1 is followed by subscripts, they are matched in the normal fashion. A dollar sign followed by any number greater than 1 ($4) will match the

indicated number of constituents. It cannot have subscripts. A dollar sign without a number can be written as a constituent in the left half and can match any number of constituents in the workspace, including none. This is called an indefinite dollar sign, while those with numbers are called definite dollar signs.

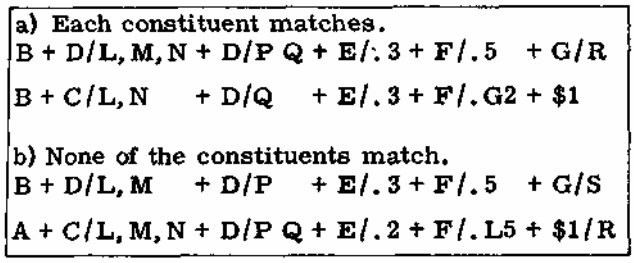


Fig. 6. Examples of match and no-match conditions. The top lines in a) and b) represent constituents in the workspace. The bottom lines represent constituents as written in the left half.

As an example of how constituents written in the left half can match constituents found in the workspace, figure 6 a shows several of the possibilities. Each constituent in the second line represents a constituent as it might be written in the left half. It matches the workspace constituent written directly above it in the first line. In figure 6 b, none of the constituents meet the match conditions.

The computer carries out a search for a match condition between each of the constituents written in the left half and corresponding constituents in the workspace in the following way: The first constituent on the left in the left half is compared in turn with each constituent in the workspace starting from the left until a match is found. The computer then attempts to match the next constituent in the left half with the next constituent in the workspace and so on until either all constituents written in the left half have been matched, or one constituent fails to match. In this case, the computer starts again with the first constituent in the left half and searches for another match in the workspace. Finally, either a match is found for all of the constituents and the computer goes on to execute the rest of the rule, or the computer cannot find the indicated structure in the workspace, in which case control is automatically transferred to the next rule. It can be seen that a structure will be found in the workspace only if it has matching constituents that are consecutive and in the same order as those written in the left half.

If an indefinite dollar sign is the first constituent in the left half, it will match all of the constituents in the workspace to the left of any constituent that is matched by the second constituent in the left half. If the indefinite dollar sign is the last constituent in the left half, it will match all of the constituents in the workspace to the right of any constituent that is matched by the next to the last constituent in the left half. If there are two or more indefinite dollar signs written in the same left half, they must be separated by constituents that are not dollar signs, or by $1 with subscripts, in order to prevent an ambiguity as to which constituents in the workspace are to be found by the several indefinite dollar signs.

If an indefinite dollar sign has constituents written on each side of it in the left half, the computer will first try to match all constituents to the left of the indefinite dollar sign. It does not have to search again for the constituents to the left of the dollar sign unless a number (as will be explained shortly) referring to a constituent to the left of the indefinite dollar sign is written to the right of the indefinite dollar sign. In this case, the computer will search for a new match for constituents to the left of the indefinite dollar sign if it fails to find a match with the constituents to the right of the indefinite dollar sign.

Constituents in the left half are conceived of as being numbered starting with one on the left. The leftmost constituent is called the number one constituent in the left half. When the constituents written in the left half have been successfully matched with constituents in the workspace, the constituents in the workspace that have been found are temporarily numbered by the computer in the same way as the constituents in the left half. The constituent in the workspace found by the number one constituent in the left half thus becomes the number one constituent in the workspace. The temporary numbering of constituents in the workspace remains until it is altered by the right half or until the rule has been completely executed. Its purpose is to allow expressions in the left half, right half and routing to refer to constituents in the workspace by their temporary number.

The various steps in a search are indicated in the example given in figure 7. The lower two lines give the constituents as they are written in the left half of a rule, and the way in

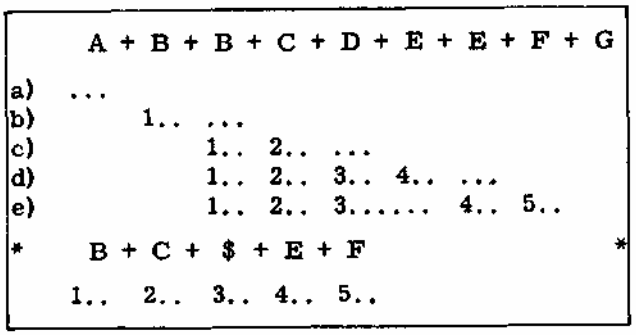


Fig. 7. Example of the search steps that the computer goes through in order to find in the workspace (top line) the structure written in the left half of the rule (next to bottom line).

which the computer numbers these constituents. The top line indicates the current contents of the workspace. Lines a) through e) represent the way in which the computer temporarily numbers the constituents in the workspace that have been successfully matched at each step of the search. The first step is indicated in line a): an attempted match between the number one constituent in the left half and the first constituent on the left in the workspace fails. In line b), the number one constituent matches the second constituent in the workspace, but an attempted match between the number two constituent in the left half and the third constituent in the workspace fails. In line c), the number one constituent in the left half matches the third constituent in the workspace, and the number two the fourth, but since the number three constituent is an indefinite dollar sign and can match any number of constituents including none, the next constituent, number four is matched with the fifth in the workspace. The match fails. Having already matched the constituents in the left half to the left of the indefinite dollar sign, the computer now tries to match the constituents to the right of the indefinite dollar sign. In line d), it finds a match of the number four constituent with the sixth, but the number five constituent in the left half fails to match the seventh constituent in the workspace. The computer then tries again with the number four constituent, and in e) finds a match between the number four and number five constituents in the left half and the seventh and eighth constituents in the workspace. Since all of the constituents in the left half have now been found in the workspace, the constituents in the workspace that have been found are left with the numbers as shown in line e). The third, fourth, fifth and sixth, seventh, and eighth constituents in the workspace become respectively the number one, two, three, four, and five constituents in the workspace. Note that two or more constituents in the workspace may be given one number if they are referred to by a dollar sign in the left half.

It is possible for the left half to be modified to some extent by what is found in the workspace . This can be done by writing a number as a constituent in the left half. The number then refers to the constituent already found in the workspace that has been given that number. The rest of the left half is then executed as if the constituent referred to in the workspace had been written originally in the left half in place of the number. A number written in the left half can only refer to a constituent in the workspace that has already been found by a constituent to the left of it in the left half. It can refer only to a single constituent, one matched by $1 for example. A number written in the left half cannot have subscripts written on it.

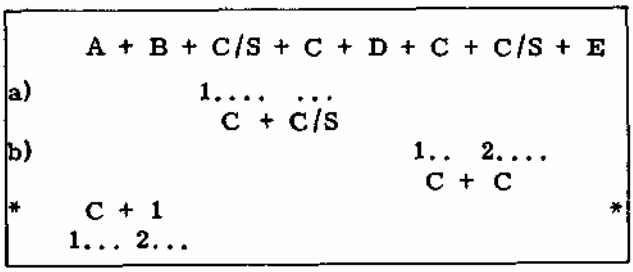


Fig. 8. Example of use of a number in the left half (bottom two lines). Attempted match indicated at a) fails, but the one at b) is successful. The contents of the workspace are represented on the top line.

Figure 8 gives an example of the use of a number in the left half. After two unsuccessful matches, the number one constituent in the left half finds the third constituent in the workspace. The number two constituent in the left half is then considered to be replaced by this constituent that has just been found (C/S). The match then fails because the fourth constituent in the workspace does not have at least the subscript S, required for a match condition. But when the number one constituent in the left half finally finds the sixth constituent in the workspace, the number two constituent in the left half is considered to be replaced by this constituent (C), and the next match is successful because this C will, according to the conditions for a match, find the C/S that is next in the workspace.

The English reading of the left half is the same as the reading of the material in the workspace except that it starts with ", search for a match in the workspace for", ends with ",and if not found, go to the next rule, but if found ", and includes conventional wordings for several abbreviations including the dollar signs and the numbers. For example, A/.G3 + $1 + $ + $2 + 2 in the left half would be read: ", search for a match in the workspace for /a constituent consisting of /the symbol A/with/the numerical subscript/greater than/3/, followed by/a constituent consisting of/any symbol/, followed by /a constituent consisting of/any number of constituents/, followed by/a constituent consisting of/two constituents/, followed by/a constituent consisting of/the number two constituent in the workspace /, and if not found, go to the next rule, but if found".

- right half -

The function of the right half is to indicate how the structures found in the workspace by the left half are to be altered. If there is no right half, the structures found in the workspace are left unaltered.

Rearrangement of the constituents found by the left half and temporarily numbered will take place when the appropriate numbers are written in the right half in the desired new order. If any of the numbers referring to constituents in the workspace are not written, these constituents will be deleted. The single digit zero as the only constituent in the right half will cause everything found by the left half to be deleted. The single digit zero is never entered in the workspace.

New constituents will be inserted in any desired place in the workspace when they are written complete with symbol and any desired subscripts and values in the desired place in the right half.

The computer will add or alter subscripts when they are written on a constituent or number in the right half. If this constituent already has a logical subscript with the same subscript name as the one that is being added, the two subscripts are combined in a special way called dispatcher logic. If there is no overlap in values, that is, if the two subscripts do not have any values in common, the old subscript is replaced by the new one. But if the two subscripts have any values in common, only the values that are common to the two will be retained. An example is shown in figure 9.

```
a)      A/NO SI,    CASE NOM GEN
        2.............................
b)      2/NO PL,    CASE -GEN DAT

c)      A/NO PL,    CASE NOM
        2.....................
```

Fig. 9. Example of the combining of subscripts by dispatcher logic. a) shows the number two constituent in the workspace, b) shows the entry in the right half, c) shows the resulting number two constituent in the workspace.

A logical subscript written in the right half with *C in place of its values complements the values of the subscript found in the workspace, that is, all the values that it has are replaced by just those values that it doesn't have. In other words, *C effectively adds a minus sign in front of the subscript values. In the case of numerical subscripts, the new value replaces, increases, or decreases the old depending on whether the value written in the right half follows the period immediately or with an intervening I or D. Since numbers are treated modulo 2^{15}, 1 added to 2^{15} - 1 will give 0, and 1 subtracted from 0 will give 2^{15}- 1. Subscripts will be deleted from a constituent when they are preceded by minus signs in the right half. A dollar sign preceded by a minus sign will cause all subscripts on that constituent to be deleted. Subscripts are added, altered, or deleted in the order from left to right in which they are written in the right half. The same subscript will be altered several times if several expressions involving it are written in the right half.

The computer will carry over subscripts from any single numbered constituent in the workspace to any other single numbered constituent indicated by the right half. For this purpose a subscript name in the right half is followed by an asterisk and a number indicating the number of the constituent from which the subscript is to be carried over. Carried over subscripts go onto the new constituent in the order from left to right in which they are written in the right half. Logical subscripts go onto the new constituent with dispatcher logic. Numerical subscripts carried over either replace, increase, or decrease the old value depending on whether . or .I. or .D. precedes the asterisk. A dollar sign preceding the asterisk will cause all the subscripts from the indicated constituent to be carried over.

After all of the operations indicated by the right half have been carried out on the constituents in the workspace, the numbered constituents remaining in the workspace and any new ones that have been added are given new temporary numbers by the computer in the order in which they are represented in the right half. These new temporary numbers will be of use when the routing is executed.

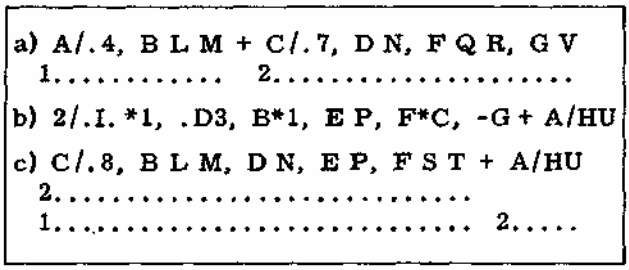


Fig. 10. An example of some right-half operations, a) the numbered constituents in the workspace initially, b) the right half, c) the numbered constituents in the workspace finally, and after renumbering.

An example of some of the operations indicated by a right half is given in figure 10. In this example, the number one constituent in the workspace is deleted. The number two constituent has its numerical subscript increased by the numerical subscript carried over from the number one constituent, and then decreased by 3 to give 8 (7 + 4 - 3 = 8). The B subscript is carried over from the number one constituent, the D subscript, not being mentioned, remains unaltered. The E subscript is added from the right half. The F subscript has its values complemented. (We assume that its possible values are Q, R, S, and T.) The G subscript is deleted. Finally, a new constituent is added to the workspace and the constituents in the workspace are renumbered.

The English reading of the right half involves only a few new wordings for abbreviations. These will be found in the section on English reading.

— routing —

The function of the routing section of the rule is to alter the contents of the dispatcher, control input and output functions, direct the computer to search a list, and add or remove plus signs in the workspace.

Dispatcher entries may be written in the routing section. When the routing part of the rule is executed by the computer, these entries are sent to the dispatcher where they combine with the entries there according to dispatcher logic. Logical subscripts on a constituent in the workspace may also be sent to the dispatcher as dispatcher entries. Conversely, dispatcher entries may be carried over as subscripts onto a constituent in the workspace. This latter, to return to the right half for a moment, is done by using the normal notation for carrying over subscripts but by using the letter D to refer to the dispatcher. 1 /CASE*D written in the right half would cause the CASE dispatcher entry to be carried over and added to the number one constituent in the workspace as a subscript. 2/$*D written in the right half would cause all of the dispatcher entries to be carried over as subscripts onto the number two constituent in the workspace. If the constituent in the workspace already has subscripts of the same kind, the dispatcher entries are combined with them according to dispatcher logic.

*D followed by a number in the routing section will cause all of the subscripts on the indicated numbered constituent in the workspace to be sent to the dispatcher as dispatcher entries where they combine with any entries already there according to dispatcher logic. When the computer executes a rule, subscripts designated in the routing section of the rule and dispatcher entries written directly in the routing section of the rule are sent to the dispatcher in the order in which they are written from left to right in the routing section. This is done after the left and the right halves are executed and before the go-to is executed. When subscripts are sent to the dispatcher from the workspace, they are not deleted from the workspace; when they are sent to the workspace from the dispatcher, they are not deleted from the the dispatcher.

COMIT has a special provision for rapid dictionary search. Dictionary entries may be written in a list which will be automatically alphabetized by the computer. This list may be entered from one or more rules called look-up rules. A look-up rule has two special features: *L in the routing section of a look-up rule, followed by one or more numbers referring to consecutively numbered constituents in the workspace, serves to indicate what structure in the workspace is to be looked up in a list. The name of a list, written in the go-to section of the look-up rule, serves to indicate what list the structure is to be looked up in. A list cannot be entered by an automatic transfer of control to the next rule.

When entering a list, the computer temporarily deletes all subscripts from the constituents in the workspace indicated by the *L, and all plus signs between the constituents, thus forming one long symbol. It is this long symbol that is looked up in the list.

The list itself has the following structure: The entries are separate rules. The first rule of a list has a hyphen followed by the name of the list in its name section. The rest of the list rules have nothing in their name sections. List rules have only one subrule each. The long symbol formed by a look-up rule is looked up in the left halves of the list rules. Each left half thus contains only one constituent with a symbol only and no subscripts. Each list rule may also have a right half, routing, and go-to. If the long symbol is found in the list, the corresponding right half is executed in normal fashion. If the number one is written in the right half of the list rule, the long symbol remains in the workspace. If the single number zero is written in the right half, the structure indicated by the look-up rule is deleted. If nothing is written in the right half of the list rule, the items temporarily deleted by the look-up rule are restored and the workspace remains unaltered. If the long symbol is not found in the list, the items temporarily deleted by the look-up rule are restored, leaving the workspace unaltered, and control is automatically transferred to the first rule after the list.

```
A          - + $ + -          //  *L2  B
*                                      C
-B           AND  = . . .              D
          BECAUSE = . . .              E
              BUT = . . .              F
                .  = . . .              .
                .  = . . .              .
                .  = . . .              .
*                                      G
```

Fig. 11. Example of a list rule with look-up rule and two rules to take care of failure to find the indicated structure.

An example of a list is given in figure 11. Rule A is the look-up rule. It serves to find any number of constituents between spaces in the workspace. (Spaces are indicated in the workspace by hyphens.) If the workspace does not have two spaces, the left half is not found and control is transferred to the next rule and then goes to C. If the indicated structure is found, the symbols of the constituents between the spaces are formed into one long symbol which is looked up in list B. If it is not found in the list, control goes to the rule after the list and then to G.

In addition to the look-up rule with its *L abbreviation, there are two other ways of altering the number of plus signs in the workspace.
*K followed by one or more numbers referring to consecutively numbered constituents in the workspace will cause the symbols of these constituents to be compressed into one long symbol, and any subscripts that they may have had will be lost.
*E followed by one or more numbers referring to consecutively numbered constituents in the workspace will cause the symbols of these constituents to be expanded by the addition of plus signs so that each character becomes a separate constituent. A list of characters is given in the center column of figure 12. Any subscripts that the original constituents may have had will be lost.

Only one of the abbreviations *L, *K, or *E may be used in any one rule, and when it is used, it must be last in the routing section to avoid confusion in the numbering of the constituents in the workspace.

The COMIT program communicates with the outside world through input and output functions under control of abbreviations in the routing section. Reading of input material and writing of output material can be done in any one of several channels and in any one of several formats as follows.

Channels. The particular computer that COMIT is being programmed for (IBM 704) has a number of magnetic tape units connected to it as well as a card reader and punch and a printer. Magnetic tapes may be prepared for the computer from information on punched cards, and material written on tape by the computer may later be read off on a printer or punched on cards. Each input or output abbreviation designates that reading or writing is to take place in channel A, B, C, or one of the others. Then, before the program is run on the computer, the operator connects the channels used by the programmer to various magnetic tape units, printers, etc. Any channel may be connected to any one of several input or output devices. This gives the maximum of flexibility of operation, and allows the output of one COMIT program to become the input of another no matter what channels are designated for input and output in the two programs.

The abbreviations *RW in the routing section followed by a channel designation will rewind the tape unit connected to that channel.

One channel, channel M, is reserved for monitoring purposes and cannot be rewound. It can only be written on. The COMIT programmer can write on this channel any information that may be of use to him later concerning the correct or incorrect operation of his program. Certain information is also written on this channel automatically if the machine discovers certain mistakes in the program during operation.

Material may be read or written in any one of several formats. Format S (specifiers) involves whole constituents, including symbols and subscripts. Format A is for text, and involves only symbols. Both format S and format A are designed for the particular characters available on the printers and card punches in current use. Other formats may be made available if and when other types of input or output equipment become available.

When material is punched on cards for reading into the computer in format S, it is punched in exactly the way that it is to appear in the workspace, including symbols, subscripts, and plus signs between constituents. Any number of characters up to a maximum of 72 may be punched on a card. When material extends over onto another card, the break between cards can be made at any point where a space is allowed, or anywhere in the middle of a symbol.

When the computer executes a rule with an abbreviation in the routing section that calls for reading in format S from a designated channel, the next constituent from the input is brought into the workspace where it replaces the designated numbered constituent. For example, *RSA2 would cause the computer to read in format S the next constituent from channel A and send it to the workspace where it will replace the number two constituent.

When the computer executes a rule with an abbreviation in the routing section that calls for writing in format S, the designated numbered constituents in the workspace are written in the designated channel. They are not deleted from the workspace by this process. For example, *WSM3 5 would cause the computer to write in format S in channel M the number three and the number five constituents from the workspace.

The computer will start a new line or card each time it executes an abbreviation calling for writing in format S. Each line requiring more than 59 characters will end after the next space, fraction bar, or comma, or before the next plus sign, or after 72 characters, whichever comes first. Lines are thus usually ended at a natural break.

Format A is for text, and involves only material written in the symbol sections of constituents . When material is transmitted between the workspace and the input or output channels under the direction of an abbreviation in the routing calling for format A, a special transliteration takes place. The purpose of this transliteration is to allow all of the characters available on the input and output devices to be used in the text. Since many of the available characters have special meanings in the rule — the plus sign separates constituents, the fraction bar separates symbol from subscripts, and so on — these must be represented in a different manner when they are written in the symbol part of a rule if ambiguities are to be eliminated. Accordingly, format A uses the transliteration scheme presented in figure 12.

character on card or tape for format A input	character in workspace, left half, or right half	character on tape or as printed after format A output
letter	letter	letter
.	.	.
,	,	,
space	-	space
number	*number	number
-	*-	-
+	*+	+
=	*=	=
/	*/	/
(	*(	(
)	*)	)
*	**	*
$	*$	$
end of line	*.	end of line
	*letter	letter
	*,	,

Fig. 12. Format A transliteration table. When the text characters of column one are read in by an *RA abbreviation, they appear in the workspace as in column two. When the characters of column two are written out by an *WA abbreviation, they appear in the output as in column three.

Note that the characters available for use in symbols consist of the letters, period, comma,

and hyphen, and an asterisk followed by any character but space.

The first column of figure 12 lists all of the characters available on the printer and card punch. The second column shows how these characters appear in the workspace after they have been brought in by an input operation calling for format A. Note that the letters, period and comma are brought in unchanged, the space becomes a hyphen in the workspace, and all other input characters are prefixed by an asterisk in the workspace. The end of line symbol *. is brought in after the last non-space character on the card.

The second column also lists all possible characters that can be written unambiguously in symbols in a rule. Some of the characters are single and some are double, consisting of an asterisk followed by another character. (An *E expand abbreviation written in the routing does not insert a plus sign between the asterisk and the other character of a double character.)

The third column of figure 12 shows how the characters of the second column will be printed after a write abbreviation calling for format A has been executed. The hyphen is written as a space, *. is interpreted as end of line, or carriage return, all other characters are unchanged except that the asterisk is removed from the double characters. Since the printer can print a maximum of 120 characters in a line, the computer will automatically end a line after 120 characters have been written if the *. abbreviation has not ended it sooner.

When the computer executes a rule with an abbreviation in the routing section that calls for reading in format A from a designated channel, the next character is brought in from the input, transliterated, and entered into the workspace in place of the designated constituent. For example, *RAB2 would cause the computer to read in format A the next character from channel B and send it to the workspace where it will replace the number two constituent.

When the computer executes a rule with an abbreviation in the routing section that calls for writing in format A, the symbols from the designated numbered constituents in the workspace are assembled into a long symbol, transliterated, and written in the designated channel. For example, *WAM1 2 4 would cause the computer to write in format A in channel M the symbols from the number one, two, and four constituents in the workspace. The workspace remains unchanged in this process.

The input and output abbreviations used in the routing section of a rule start with an asterisk followed by R or W for read or write, then there follows a letter designating format A or S, then a letter designating a channel, usually A, B, or C (or M in the case of a write abbreviation only) and finally one number in the case of a read abbreviation and one or more numbers in the case of a write abbreviation designating the numbered constituents in the workspace that are involved. Examples have been given in previous paragraphs.

Summary

This notational system is convenient and well adapted to a large class of problems including language translation and formal algebraic manipulation. The computer automatically converts programs in this notation into actual computer programs. Programs are written in the notation as a series of rules, each of which may have five parts, the name, the left half, the right half, the routing, and the go-to.

An arbitrary rule name may be written in the name section of each rule. In the go-to is written the name of the next rule to be executed.

The material to be operated on exists in the computer as a series of constituents in the workspace. The function of the left half is to indicate which constituents are to be operated on by the computer. This is done by writing in the left half only enough about the constituents or their context to uniquely identify them. In this way, the same rule can be made to apply in a variety of situations that are the same in certain respects. There is a convenient way of locating two or more constituents in the workspace that match each other in a certain way without having to know what the way is in which they match.

If the constituents indicated in the left half cannot be found in the workspace, control goes to the next rule instead of to the rule mentioned in the go-to. This is one type of program branch.

The function of the right half is to indicate what operations are to be performed on the constituents found by the left half. It is possible to add, delete, and rearrange constituents. It is also possible to add subscripts to any constituents, and to rearrange, delete, and calculate with them. There are two kinds of subscripts, numerical subscripts that can be used for counting and simple arithmetic operations, and logical subscripts that can conveniently be used for logical calculations. Both types of

subscripts may be used in the left half to help indicate the material to be operated on. They can thus enter into the condition for a program branch. Logical subscripts can in addition be sent to the dispatcher where, as dispatcher entries, they become effective in controlling n-way program branches. Each dispatcher entry controls which of several subrules is to be carried out in a given rule.

A third type of program branch is provided by the facility for looking up material from the workspace in a list expressed as a series of list rules. This facility can be used for dictionaries. The computer will automatically alphabetize all material in lists to facilitate the look-up operation.

The function of the routing section is to control input and output operations, to control flow of information to and from the dispatcher, to control list look-up operations, and to bring several constituents together into one constituent, or separate a constituent into several constituents, one for each character.

Input and output facilities provide the maximum of convenience for the user. In addition, the system has a number of checks built in that will help the programmer find any mistakes he may make in writing his program.

How to Read a Rule in COMIT

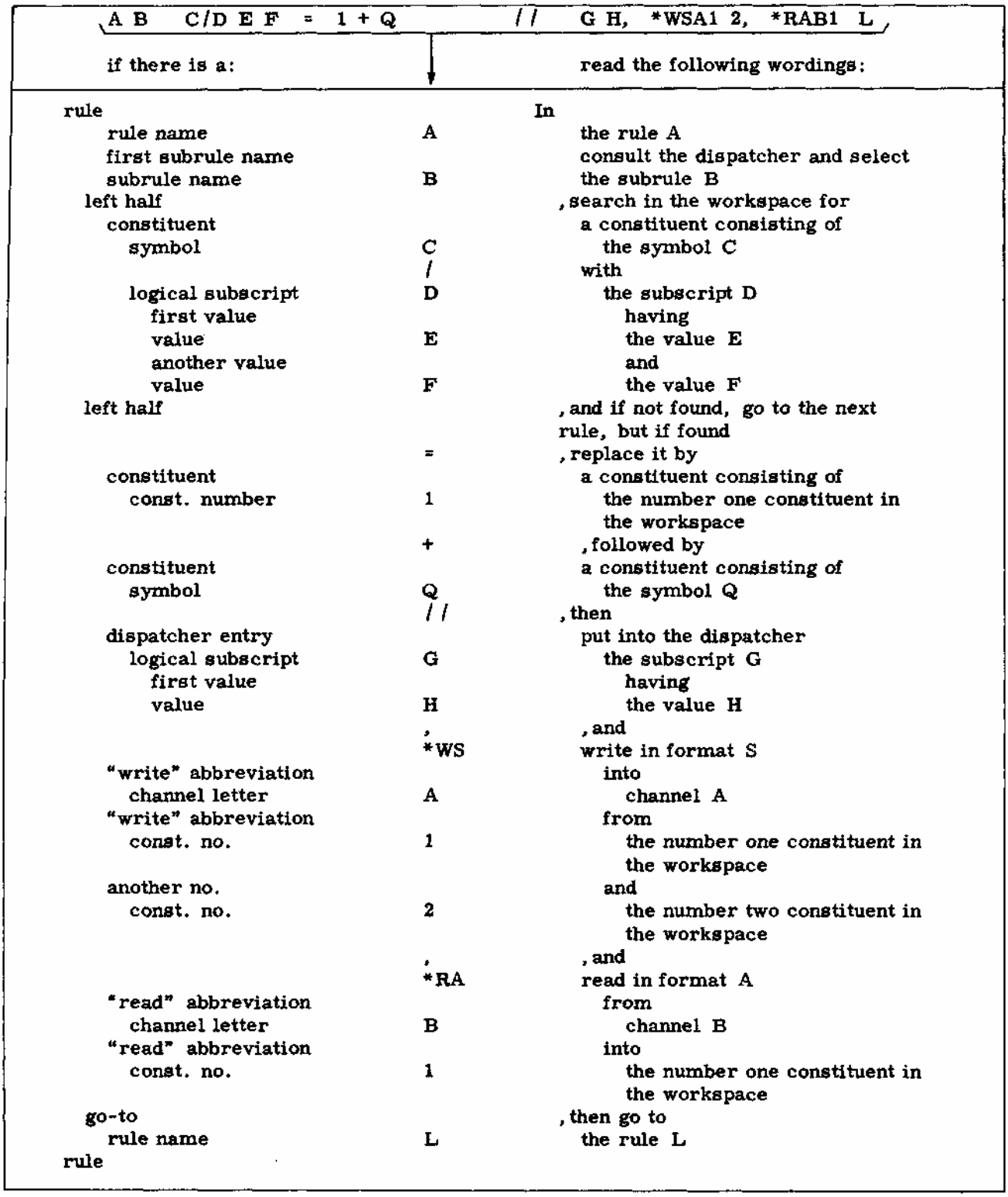
The purpose of this section is to present a summary of the various conventions used for reading a rule of COMIT in English. The readings are, of course, purely mnemonic, for they cannot describe completely what the computer does when it executes the rule.

The various abbreviations used in a rule are tabulated in figure 13. Some abbreviations have several different English readings depending on what part of the rule they are in. When this is the case, a note has been inserted in the table to give an indication of the contexts in which the abbreviation should be given the various readings.

In addition to the English readings associated with the abbreviations, there are conventional wordings that are not associated with any particular abbreviations, but instead with certain positions in the various sections and parts of the rule. In order to summarize these conventional wordings, figure 14 presents a sample rule and its complete reading. The wordings that are associated with the format are provided with an explanatory note giving the circumstances under which they are used.

=	, replace {it / them} by
+	, followed by
/	with
//	, then
-	the list (in column 1) deletion of (before a subscript) (when at the right end of a line, it is a hyphen, rule continues on next line) all values except (before a value)
,	, and (in the routing) , and with (in a constituent)
.	the numerical subscript
*	this rule (in column 1) carried over from (in subscript) the next rule (in go-to)
*C	with all values complemented
*D	put into the dispatcher the subscripts from
*E	expand
*K	compress
*L	look up in a list
*RA_	read in format A from channel _ into
*RS_	read in format S from channel _ into
*WA_	write in format A into channel _ from
*WS_	write in format S into channel _ from
*RW	rewind the tape unit of channel
$	any number of constituents all subscripts
$1	any symbol
$3	three constituents (etc.)
0	nothing
3	three (after., I, D, G, or L) the number three constituent in the workspace (etc.)
D	decreased by
G1	greater than
I	increased by
L	less than

Fig. 13. Abbreviations used in COMIT and their English readings.



A B C/D E F = 1 + Q	//	G H, *WSA1 2, *RAB1 L
if there is a:	↓	read the following wordings:
rule		In
rule name	A	the rule A
first subrule name		consult the dispatcher and select
subrule name	B	the subrule B
left half		,search in the workspace for
constituent		a constituent consisting of
symbol	C	the symbol C
	/	with
logical subscript	D	the subscript D
first value		having
value	E	the value E
another value		and
value	F	the value F
left half		,and if not found, go to the next rule, but if found
	=	,replace it by
constituent		a constituent consisting of
const. number	1	the number one constituent in the workspace
	+	,followed by
constituent		a constituent consisting of
symbol	Q	the symbol Q
	//	,then
dispatcher entry		put into the dispatcher
logical subscript	G	the subscript G
first value		having
value	H	the value H
	,	,and
	*WS	write in format S
"write" abbreviation		into
channel letter	A	channel A
"write" abbreviation		from
const. no.	1	the number one constituent in the workspace
another no.		and
const. no.	2	the number two constituent in the workspace
	,	,and
	*RA	read in format A
"read" abbreviation		from
channel letter	B	channel B
"read" abbreviation		into
const. no.	1	the number one constituent in the workspace
go-to		,then go to
rule name	L	the rule L
rule		

Fig. 14. Conventional wordings that are associated with the format of a rule. The left hand column names the various sections and parts of the sample rule with which the wordings of the last column are associated.

How to Write a Rule in COMIT

The purpose of this section is to present the conventions that must be adhered to when writing a COMIT rule.

General: The left hand 72 columns of the punched card are available for writing COMIT rules. The other 8 columns can be used for numbering the cards if so desired. If a rule requires more than 72 columns to write, a hyphen may be used at the end of one card and the rule continued on the next card in any column. To indicate a space between the hyphenated parts of the rule, leave a space before the hyphen.

Comments enclosed in parentheses are interpreted by the computer as spaces. No parentheses may be included within a comment. A comment continued onto the next card should be hyphenated.

Name section: The first subrule of a rule has a rule name starting in column one. A rule that is never referred to by name in a go-to or in the dispatcher may have an asterisk in column one instead of a name.

All subrules of a rule with more than one subrule have a subrule name. The subrule name is separated from the rule name by one or more spaces, otherwise it starts in any column after the first. A rule can have a maximum of 36 subrules. If there are several rules with the same rule name, they must have identical sets of subrule names.

The first rule of a list has a hyphen in column one followed by the list name. The rest of the rules in a list have nothing in the name section.

A name consists of 12 or fewer consecutive characters. The characters available are the letters of the alphabet, the numbers, and the period and hyphen in medial position, that is not at the beginning or end of the name.

Left half: The first subrule of a rule carries the left half if there is one. All list rules have a left half and only one subrule. The left half is separated from the name by one or more spaces, otherwise it starts in any column after the first.

When the left half could be confused with a subrule name, it should be followed by an equal

Example of subscript	Left half or right half	Binary representation of values or English reading of subscript
SUB	L R	000000...
SUB VAL	L R	100000...
SUB VAL VAL	L R	110000...
SUB -VAL VAL	L R	001111...
SUB -	L R	111111...
-SUB	R	deletion of/the subscript/SUB
SUB*4	R	the subscript/SUB/carried over from/the number four constituent in the workspace
$*4	R	all subscripts/carried over from/the number four constituent in the workspace
-$	R	deletion of/all subscripts
SUB*C	R	the subscript/SUB/with all values complemented
.3	L R	the numerical subscript/three
.G3	L	the numerical subscript/greater than/three
.L3	L	t. n. s./less than/three
.D3	R	t. n. s./decreased by/three
.I3	R	t. n. s./increased by/three
-.	R	deletion of/the numerical subscript
.*4	R	t. n. s./carried over from/...
.I.*4	R	t. n. s./increased by/t. n. s./carried over from/.
.D.*4	R	t. n. s./decreased by/t. n. s./carried over from/.

Fig. 15. A tabulation of all the types of subscripts allowed in the left and the right halves of rules.

sign to resolve the ambiguity. The possible ambiguity is between a left half consisting of a symbol with no subscripts in a rule with no subrule name or right half, and the subrule name of a first subrule with no left or right half.

The left half consists of one or more constituents separated by plus signs and optional spaces. A constituent may be a symbol or $1 with or without subscripts, or it may be a definite or indefinite dollar sign without subscripts, or it may be a number, without subscripts, referring to a numbered constituent already found in the workspace.

The left half of a list rule consists of a single constituent composed of a symbol only.

A symbol is any uninterrupted sequence of characters. A character in a symbol may be a letter; period, comma, or hyphen, or an asterisk followed by any character except space. These latter double characters are treated as single characters by the *E abbreviation. The characters have been summarized in figure 12.

If a constituent has subscripts, these follow the symbol and are separated from it by a fraction bar and optional spaces. Subscripts are separated from each other by commas and optional spaces.

A logical subscript has a subscript name written like a rule name. If it has values, these have the form of subrule names and are separated from it and from each other by one or more spaces. A logical subscript need not refer to a rule name, but if it does, its Values are restricted to the subrule names of that rule.

The types of logical and numerical subscript expressions available for use in the left half are tabulated in figure 15 and indicated by an L. The table also gives an indication of the meaning of the subscripts and how the logical subscript values are stored in the computer in terms of zeros and ones.

Right half: Any rule that has a left half may have right halves in its subrules. Each right half is marked by a preceding equal sign and optional spaces.

The right half consists of one or more constituents separated by plus signs and optional spaces. A constituent in the right half may be a symbol with or without subscripts, or it may be a number, with or without subscripts, referring to a numbered constituent in the workspace. The types of logical and numerical subscripts available for use in the right half are also listed in figure 15, and indicated by an R.

Routing section: The routing section, if written, is preceded by two fraction bars and optional spaces. In the routing section, dispatcher entries may be written in the same way that subscripts and values are written in the right half. In addition the input abbreviations *RAA, *RAB, etc., and *RSA, *RSB, etc. may be written followed by a number designating one numbered constituent in the workspace. The output abbreviations *WAA, *WAB, etc., and *WSA, *WSB, etc. may be followed by one or more numbers referring in any order to numbered constituents in the workspace. The *L, *K, and *E may be written followed by one or more numbers referring to consecutively numbered constituents in the workspace. The numbers are separated by one or more spaces. Separate entries in the routing section are separated by commas and one or more spaces. Only one *L, *K, or *E abbreviation may be written in any rule, and it must be the last thing written in the routing section.

Go-to: In the go-to is written either the name of the rule or list that is to be executed next, or an asterisk signifying that the next rule in sequence is to be executed next. The go-to is separated from the rest of the rule by one or more spaces.

The author wishes to express his appreciation to S. F. Best, F. C. Helwig, G. H. Matthews, A. Siegel, and M. R. Weinstein for their many helpful criticisms and suggestions.

Appendix

Some Sample Programs

We now present a few simple programs written in COMIT. These programs have been chosen for their illustrative and pedagogical value. In order to see how the computer carries out these programs, the reader may have to keep track of the contents of the workspace and dispatcher on a separate piece of paper while going through the programs.

The first seven examples show how some simple operations on text can be carried out. The first one will bring 25 characters of text into the workspace from the input. The remaining six will insert position markers in various places between the characters in the workspace or make various substitutions or order changes. The position markers must be chosen in such a way that they will not be confused with other constituents.

The ninth example is a simple word-for-word translation routine. The text is brought in a character at a time, and each character is looked up in a list to see if it is a letter or mark of punctuation. Each continuous string of letters between punctuation marks or spaces is looked up in the dictionary. The punctuation marks and spaces are carried over into the output text unchanged. Any word that is not found in the dictionary is printed in its original form and enclosed in parentheses. Alternative meanings are separated by fraction bars. An output line is printed as soon as a word is translated that makes the line exceed 55 characters in length. A slight additional complication would be needed to prevent a line from starting with

```
(1.    READ 25 CHARACTERS OF TEXT FROM CHANNEL B)
START          $ = 1 + Q/.0              BRING-IN
BRING-IN Q/.L25 = A + 1/.I1 //*RAB1      BRING-IN
*              Q = 0                              *

(2.    ADD A QD AFTER THE FIRST DOUBLE LETTER)
*         $1 + 1 = 1 + 2 + QD                     *

(3.    ADD A QF AFTER THE FIRST A IN THE WORKSPACE)
*              A = 1 + QF                         *

(4.    ADD A QL AFTER THE LAST A IN THE WORKSPACE)
*              $ = QL + 1                         *
A      QL + $ + A = 2 + 3 + 1                     A

(5.    REPLACE EVERY A IN THE WORKSPACE WITH A B-
       AND EVERY B WITH AN A)
1              A = QR                             1
2              B = A                              2
3             QR = B                              3

(6.    DELETE EVERY E BEFORE A LETTER MARKED WITH A-
       VOWEL SUBSCRIPT)
                (SUGGESTED BY D. DINNEEN)
DEL    E + $1/VOWEL = 2                       DEL

(7.    MOVE THE SECOND CONSTITUENT TO LAST PLACE-
       AND THE ORIGINALLY-LAST CONSTITUENT TO FIRST PLACE)
                (SUGGESTED BY S. ROGOVIN)
*                      $ = 1 + QM                 *
*      $1 + $1 + $ + $1 + QM = 4 + 1 + 3 + 2     *


(8.    PROGRAM TO BRING IN A NUMBER OF CONSTITUENTS AND PRINT OUT-
       ALL OF THE N FACTORIAL PERMUTATIONS OF THEM)
              (PROBLEM SUGGESTED BY B. ULVESTAD)
INPUT                              $ = Q + 1 + N      //*RSA1          INPUT
*                                  N = QL + QR/.1                        *
NUMBER Q + $ + $1 + QL + $ + QR + N = 1 + 2 + 4 + 3 + 5 + 7/.*6 + 6/.I1  NUMBER
*                Q + QL + $ + N + $ + QR = 3 + 2 + 6/.1 + 4 + 5          *
PRINT                          $1 + QL = 2 + 1          //*WAB2          PRINT
*                  QL + $ + $1 + QR + N = 2 + Q + 3 + 1 + 5 + 4          *
PERMUTE $1 + Q + $1 + $ + QL + $ + QR + N = 3 + 2 + 4 + 1 + 5 + 6 + 7 + 8/.D1 *
TEST              QL + $ + QR + $1/.G0 = 1 + 3 + 2 + 4                  NUMBER
MOVE               $1 + Q + $ + QR + $1 = 2 + 1 + 3 + 5 + 4             PERMUTE
```

a space or mark of punctuation, and to allow for the hyphenation of long words at the end of the line.

The eighth example illustrates another class of problems that COMIT is convenient for, that is, problems of an algebraic or manipulational nature.

Readers who would like to use the COMIT system should correspond with the author for further details.

```
(9.    WORD-FOR-WORD TRANSLATION ROUTINE)
PREPARE            $ = 1 + QR                          // WAS PUNC            READ
READ              QR = A + 1                           // *RAA1, *L1    WORD-BOUNDS
DONE                                                                    LAST-LINE
-WORD-BOUNDS        .                                                     IS-PUNC
                    ,                                                     IS-PUNC
                    -                                                     IS-PUNC
                   *. = 0                                                    READ
                   *=                                                    IS-PUNC
                   *-                                                    IS-PUNC
                   *+                                                    IS-PUNC
                   *(                                                    IS-PUNC
                   *)                                                    IS-PUNC
                   *$                                                    IS-PUNC
                   **                                                    IS-PUNC
                   */                                                    IS-PUNC
                   *0                                                    IS-PUNC
                   *1                                                    IS-PUNC
                   *2                                                    IS-PUNC
                    .                                                           .
                    .                                                           .
                    .                                                           .
                   *9                                                    IS-PUNC
OTHERS                                                                  IS-LETTER
IS-PUNC                                                                         *
WAS LETR                                                // WAS PUNC      LOOK-UP
    PUNC                                                                   READ
IS-LETTER                                                                       *
WAS LETR                                                                   READ
    PUNC     $1 + QR = QS + 1 + 2                       // WAS LETR        READ
LOOK-UP      QS + $ + $1 + QR                           // *L2        DICTIONARY
-DICTIONARY    ABER = BUT                                                       A
                UND = AND                                                       A
               MANN = MAN                                                       A
               FRAU = WOMAN*/WIFE*/MRS.                                         A
               KIND = CHILD                                                     A
             KINDER = CHILDREN                                                  A
                .       .                                                       .
                .       .                                                       .
                .       .                                                       .
NOT-FOUND    QS + $ + $1 + QR = *( + 2 + *) + 3 + 4                     LINE-PRINT
A            QS + $1          = 2                       // *E1         LINE-PRINT
LINE-PRINT      $56 + $ + QR                           // *WAB1 2        DELETE
TOO-SHORT                                                                  READ
DELETE                $ + QR = 2                                           READ
LAST-LINE             $ + QR = 1                        // *WAB1               *
```